Distinguishing protons and electrons from detector signature using densely connected neural network

Master's Thesis
University of Turku
Department of Physics
and Astronomy
Physics
2020
Antti Sorsa
Referees:
Prof. Rami Vainio
MSc. Philipp Oleynik

The originality of this thesis has been checked in accordance with the University of Turku quality assurance system using the Turnitin OriginalityCheck service.

UNIVERSITY OF TURKU
Department of Physics and Astronomy

SORSA, ANTTI: Distinguishing protons and electrons from detector signature using densely connected neural network
Master's Thesis, 63 p.
Physics
May 2020

The FORESAIL group is a collaboration of four Finnish scientific institutions, and one of the research subjects is the detection of radiation using small satellites. One of the challenging tasks is the measurement of relativistic particles, and one way to implement this is to use a series of silicon pixel detector plates and aluminium and tantalum absorbers. Resources aboard a small satellite are limited, as well as the link speed to Earth. Therefore an efficient solution for automatic particle categorization onboard would be beneficial.

In this work, the radiation environment and particle dynamics in Earth's magnetosphere are shortly studied along with some commonly used particle detection techniques, such as the ΔE-E method. For the particle categorization solution several neural network development tools and network types were compared. The proposed neural network was implemented with Google TensorFlow and Python and the parameters of the network were calibrated using automatic loops. The main objective was to explore whether a small-scale neural network would be sufficient in categorizing electrons and protons along with their energy and incident angle.

The detector was modelled using Geant4, and also the test material was generated with it using Monte Carlo simulations. The Geant4 output was parsed in Python into the TensorFlow program, where it was used in training the neural network. Then the detection accuracy was tested using simulated and also modelled realistic proton and electron spectra.

The network was able to distinguish any two input sets quite well, not showing much degradation in detection accuracy when changing energies and other properties of the particles. However, multiple categorization ability of the designed network didn't prove very good, and most probably would require more complex neural network architecture or different type of design.

Keywords: Geant4, Particle detection, Simulation

Table of Contents

1 Introduction

Development of small satellites has advanced considerably in the 2010s, and also the minuscule nanosatellites have been used successfully in various scientific missions. A nanosatellite typically weighs only a few kilograms, and therefore it is relatively less cumbersome to launch into orbit compared to traditional satellites. Hence the costs and the project planning time can be cut down, and new technologies can be realized relatively quick.

The FORESAIL project is a collaboration between four Finnish space research groups, and it has the goal to start the usage of nanosatellites in scientific space missions. The missions consist of measurements of Earth's magnetosphere and its behaviour, e.g. in relation to solar wind. Also understanding the dangers of space as an operational environment is an important objective considering future space missions [1]. Earth's magnetosphere consists of several different areas each having different characteristics regarding particle radiation and other related properties. Therefore in the central part of many FORESAIL missions is the ability to measure the particle radiation. The main objectives of FORESAIL-1 are the measurements of energy and flux of energetic particle loss to the atmosphere and energetic neutral atoms of solar origin [2].

In the field of machine learning, neural networks have become ubiquitous during the last decade, and have applications ranging from face recognition to sound generation. Also the development of neural network tools has enabled many straightforward ways to implement neural networks for many applications. Typically neural networks require a set of training material in order to perform in their desired task, and their maturity is defined by the error rate when processing the training material. [3]

The purpose of this thesis is to study the possibilities of using machine learning and neural networks to distinguish energetic protons and electrons with partially overlapping energy spectra using measurement data from a multi-cell silicon radiation detector. In addition to energy and the type of the particle, pitch angle should also be distinguished. The detector model used in this work consists of several detector plates, and each of them creates a signal relative to the energy deposited by an incident particle. The detector plates will be placed in a 3D configuration, through

which it is possible in theory to deduce the direction, energy and the spread of a resulting particle shower.

Initially, radiation environment of Earth's magnetosphere and sources of radiation are introduced. For the implementation part, several options and machine learning in general in addition to neural networks were studied in order to find the most suitable solution for the task, keeping in mind the resource limitations as the solution is aimed to be implemented on a nanosatellite. Neural networks require training material, which was created using the industry standard Geant4 simulation framework, into which a model for the detector was constructed by modifying an example calorimeter model. The selected neural network solution was trained using the simulated data, and the parameters were tweaked to improve the detection efficiency. However, the implementation of the selected solution for a physical satellite module is beyond the scope of this work. Implementation possibilities for the solution are however shortly discussed.

2 Radiation and detectors

2.1 Units for radiation

Energy of radiation is conveniently measured in electron volts (eV), which in terms of SI unit Joule (J) is defined as

$$1\text{eV} = 1.602 \cdot 10^{-19}\text{J}$$

In case of electromagnetic radiation, the energy of a photon can be acquired from the relation

$$E = h\nu$$

where h is the Planck constant and ν the frequency of the photon. A photon with a wavelength $\lambda = c/\nu$ has an energy

$$E = \frac{1.240 \cdot 10^{-6}}{\lambda}$$

where λ is in meters and E in eV. [4]

The total energy E of a particle with mass m_0 can be calculated by

$$E = \sqrt{(pc)^2 + (m_0c^2)^2}$$

where p is the particle's momentum. Fuerthermore, particle *flux* is defined as the number of particles travelling through a unit area per unit time, and its unit is m^{-2} s^{-1}. [4]

2.2 Dynamics of particles

Movement of a charged particle can be described using the Lorentz force equation

$$\frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} = q(\mathbf{E} + \mathbf{v} \times \mathbf{B}) + \mathbf{F}_{non-EM} \tag{1}$$

where $\mathbf{F}_{non-EM}$ in most cases consists only on gravitational force $\mathbf{F} = m\mathbf{g}$, which many times is negligible compared to electromagnetic forces. Solving equation (1) is not straightforward in most cases, but it is possible to get useful results using

several approximations. [5]

2.2.1 Guiding center approximation

First simplification in non-relativistic case ($\mathbf{p} = \mathrm{m}\mathbf{v}$) to equation (1) is to model a situation in which the electric field $\mathbf{E}$ is very small and the magnetic field $\mathbf{B}$ doesn't change, ie. $\mathbf{E} = 0$ and $\mathbf{B} =$ constant. Equation (1) then takes the form

$$m\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = q(\mathbf{v} \times \mathbf{B})$$

When $\mathbf{B}$ is chosen to be along the z-axis $\mathbf{B} = B\mathbf{e}_z$, the component-wise equations become

$$\begin{aligned} m\dot{v}_x &= \quad qBv_y \\ m\dot{v}_y &= -qBv_x \\ m\dot{v}_z &= \quad 0 \end{aligned}$$

which show that velocity along the magnetic field line is a constant. By taking another time derivative the equations take the form

$$\begin{aligned} \ddot{v_x} &= -\omega_c^2 v_x \\ \ddot{v_y} &= -\omega_c^2 v_y \end{aligned}$$

which describe a harmonic oscillator with an angular frequency (i.e. the cyclotron frequency) $\omega_c = qB/m$. By solving the equations with respect to coordinates, it can be seen that the equations depict a circular movement in an xy-plane with the circle radius (i.e. the Larmor radius) of

$$r_L = \frac{mv_\perp}{|q|B} \tag{2}$$

where $v_\perp = \sqrt{v_x^2 + v_y^2}$. The central point of the circular movement is called the guiding center, and the time elapsed for one rotation around it

$$\tau_L = \frac{2\pi}{|\omega_c|} \tag{3}$$

where τ_L is the rotation period (i.e. the Larmor period). [5]

2.2.2 Drifting motion

Approximating equation (1) can also be done by taking into account the electric field by defining $\mathbf{E}$ as a constant. Then the equation of parallel to $\mathbf{B}$ is

$$m\dot{v}_{\parallel} = qE_{\parallel}$$

Marking the perpendicular electric field along the x axis, the components of the equation are

$$\begin{aligned} \dot{v}_x &= \omega_c v_y + \frac{q}{m} E_x \\ \dot{v}_y &= -\omega_c v_x \end{aligned}$$

which become

$$\begin{aligned} \ddot{v_x} &= -\omega_c^2 v_x \\ \ddot{v_y} &= -\omega_c^2 (v_y + \frac{E_x}{B}) \end{aligned}$$

into which one can mark $v_y' = v_y + E_x/B$, and yield the same equations as in guiding center approximation, the difference being that now the guiding center is moving along the y axis with a velocity E_x/B. The *(electrical) drift velocity* is therefore

$$\mathbf{v}_E = \frac{\mathbf{E} \times \mathbf{B}}{B^2} \tag{4}$$

which doesn't depend on the charge or the mass of the particle. Drifting due to an electric field is presented in figure 1. [5]

Equation (4) can be expressed in more general form to describe drift movement as

$$\mathbf{v}_D = \frac{\mathbf{F}_{\perp} \times \mathbf{B}}{qB^2} \tag{5}$$

where $\mathbf{F}$ is the force affecting on the particles (note that $\mathbf{F} = q\mathbf{E}$ gives the electric

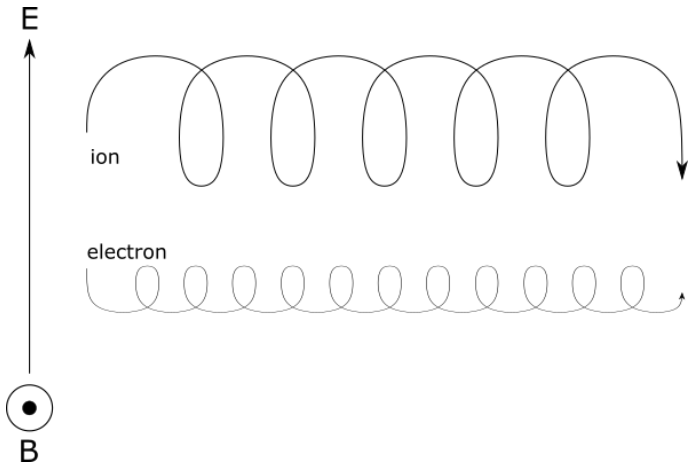

Fig. 1: Drifting motion of a positive and a negative particle when both magnetic and electric fields are constant. Adapted from [5]

.

force, sonsistent with Eq. (4).) Equation for *gravitational drift* is therefore

$$\mathbf{v}_g = \frac{m\mathbf{g} \times \mathbf{B}}{qB^2}$$

which describes the separation of particles based on their m/q perpendicular to gravitation and the magnetic field. [5]

When considering a magnetic field that only changes a little during one rotation of a particle around the guiding center, the gyro-averaged force caused by the magnetic field on a particle is given as

$$\mathbf{F} = -\mu \nabla B$$

where $\mu = W_\perp / B$ is the magnetic moment. When inserted into equation (5) yields the equation for *gradient drift*

$$\mathbf{v}_G = \frac{\mu}{qB^2}\mathbf{B} \times (\nabla B) = \frac{W_\perp}{qB^3}\mathbf{B} \times (\nabla B) \tag{6}$$

As can be seen from equation (6), the gradient drift depends on the energy and

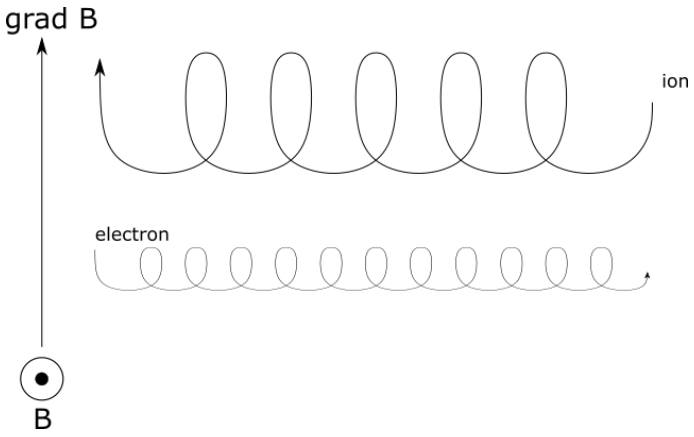

Fig. 2: Gradient drifting of ions and electrons. Adapted from [5]

charge of the particle, and the particles with opposite charges flow into opposite directions creating current. Gradient drifting for ions and electrons is visualized in figure 2. [5]

Changing magnetic field commonly also has curvature, and therefore the guiding center experiences the centrifugal force

$$\mathbf{F}_C = -mv_{\parallel}^2 \frac{\mathbf{R}_C}{R_C^2}$$

where R_C is the radius of curvature for the magnetic field and $w_{\parallel}$ is the velocity of the guiding center along B. Inserting this into 5, the formula for *curvature drift* can be expressed as [5]

$$\mathbf{v}_C = -\frac{mv_{\parallel}^2}{q} \frac{\mathbf{R}_C \times \mathbf{B}}{R_C^2 B^2}$$

2.2.3 Magnetic mirror and magnetic bottle

In the guiding center approximation the total energy W and the magnetic moment $\mu = W_{\perp}/B$ of the particle are conserved. If the particle moves towards increasing B, the $W_{\perp}$ increases and $W_{\parallel}$ tends to zero. The pitch angle of the particle in a magnetic field is α and $v_{\perp} = v \sin \alpha$, the magnetic moment is

$$\mu = \frac{mv^2 \sin^2 \alpha}{2B}$$

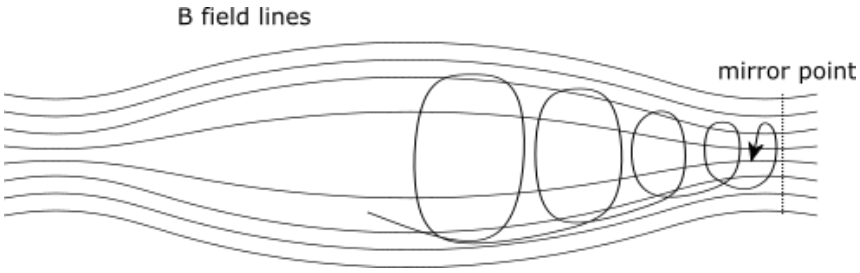

Fig. 3: Particle switching direction from a mirror point in a magnetic bottle. Adapted from [6].

where the only changing variables are α and B and they have the relation

$$\frac{\sin^2 \alpha_1}{\sin^2 \alpha_2} = \frac{B_1}{B_2}$$

As the guiding center of the particle moves towards an increasing B, α tends to 90° and $W_\parallel = 0$. The guiding center is still affected by the *mirror force* $\mathbf{F} = -\mu \nabla_\parallel B$, which turns the motion of the particle around.

When the magnetic field lines are shaped in a way that there are two magnetic mirrors in their ends, a *magnetic bottle* is formed. A particle is trapped into a magnetic bottle when the pitch angle is between

$$\arcsin \sqrt{\frac{B_0}{B_m}} \leq \alpha_0 \leq 180 - \arcsin \sqrt{\frac{B_0}{B_m}}$$

where B_m is the mirror magnetic field magnitude and B_0 the reference field magnitude. This is depicted in figure 3. If the particle in a magnetic bottle has α_0 outside the given range, it is said to reside in the *loss cone*, and it will escape the magnetic bottle.

Motion of the particle as it bounces in the magnetic bottle is close to periodic. The bounce period is defined as

$$\tau_b = 2 \int_{s_m}^{s'_m} \frac{\mathrm{d}s}{v_\parallel(s)} = \frac{2}{v} \int_{s_m}^{s'_m} \frac{\mathrm{d}s}{(1 - B(s)/B_m)^{1/2}} \tag{7}$$

where s is the arc length along the guiding center's track and s_m and s'_m are the

coordinates of mirror points. In order for the approximation to be valid, the bounce period must be much longer than the Larmor period, i.e. $\tau_b \gg \tau_L$, and also the magnetic field must be uniform enough, i.e.

$$\tau_b \frac{\mathrm{d}B/\mathrm{d}t}{B} \ll 1$$

in every point of the trajectory of the particle. [6]

2.2.4 Acceleration of particles

As charged particles encounter magnetic fields and field gradients, they gain energy by *Fermi acceleration* as they bounce between inhomogeneities carried by plasma flows. Fermi acceleration requires the particles to exceed thermal energies and to be in a collisionless environment, as collisions thermalize the energies. [7]

Shock front formed during a solar flare or a coronal mass ejection can accelerate particles by several different processes. In shock-drift acceleration the particles gain energy via grad-B drift along the shock front, and it is most effective when the shock propagates nearly perpendicular to the magnetic field and the electric induction field is the largest. Diffusive shock acceleration bounces particles back and forth as the plasma evolves around the shock front, and it dominates at nearly parallel shocks. Stochastic acceleration is at work behind the shock front in a turbulent flow with a lot of magnetic field and velocity fluctuations and can be considered a Fermi process. [8][9]

2.2.5 Radiation interactions with matter

Charged particles interact mainly with other particles electromagnetically through Coulomb force. Depending on the energy of the incident particle, the target particle usually goes through excitation, in which the target particle's electrons are transferred to a higher energy orbital. The excitation has to be released in form of a photon with energy equivalent to the orbital energy difference. If the incident particle gives a larger amount of energy to the electron, it might get removed completely from the atom, causing *ionization*. In case of e.g. an incident alpha particle, the total energy of it is much greater than it can lose to a single electron, and therefore

it will interact with many electrons and lose speed in the process. This leaves an amount of ionization or excitation that depends on the material. [10][11]

The *stopping power* or *specific energy loss* can be defined as the amount of energy lost per unit distance:

$$S = -\frac{\mathrm{d}E}{\mathrm{d}x}$$

As the incident particle moves faster, it spends less time in the vicinity of the electrons, thus losing less energy to them. Therefore it can travel a longer way into the material. This can be modelled using the *Bethe formula*:

$$-\frac{\mathrm{d}E}{\mathrm{d}x} = \frac{4\pi}{m_e c^2} \cdot \frac{nz^2}{\beta^2} \cdot \left(\frac{e^2}{4\pi\epsilon_0}\right)^2 \cdot \left[\ln\left(\frac{2m_e c^2 \beta^2}{I(1-\beta^2)}\right) - \beta^2\right] \tag{8}$$

where v and z are the velocity and charge of the primary particle, $\beta = v/c$ and m_e and e are electron rest mass and charge. Also the charge z is in elementary charges. I is the average excitation and ionization potential of the material, and it usually is experimentally determined for different materials. For non-relativistic particles the formula can be approximated in a considerably simpler form as the β terms are small. Different charged particles have different energy losses, and furthermore the target material greatly affects the stopping power; the greatest stopping power is achieved with a high atomic number high density material. This is illustrated for some particles travelling through air in figure 5. [10]

The penetration depth of the incident particle can be shown using the Bragg curve. From it can be seen the effect of the incident particle slowing down and yielding more and more energy into the surrounding material until stopping. For example, the energy loss of an alpha particle penetrating material is shown in figure 4. In addition to a single alpha particle, also the behaviour of a beam of alpha particles with same initial velocity is shown. The difference in penetration depth is due to statistical variations of the energy loss. [10]

In the case of heavier incident particles, such as fission fragments, the specific energy loss decreases as the particle moves into the absorber material. This is due to the heavy particle's tendency to grab multiple electrons starting right after intruding into the absorber material and thus decreasing the effective charge of the particle. Particles with lower initial effective charge such as the alpha particle, electron pickup

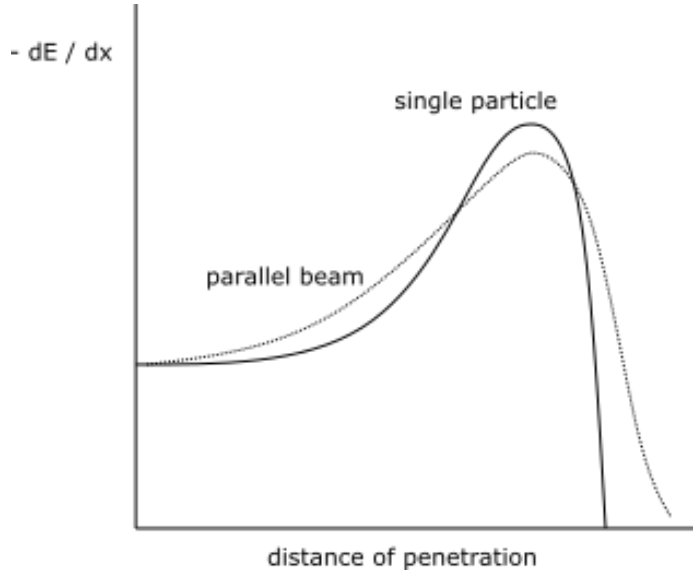

Fig. 4: Energy loss of an alpha particle and a beam of alpha particles, showing how the specific energy loss increases as the particle loses energy in the material before stopping. Adapted from [10].

only becomes significant at the end of the absorption event. The effect of charge pickup strength in relation to ion energy is illustrated in figure 6. [12]

Many times the incident particle possesses sufficient energy to cause ionization in the target material in the form of *secondary electrons*, which can also be called *delta rays*. Delta rays can also cause further ionization in the absorber. An incident alpha particle typically causes at most 10 secondary particles, but a heavier fission product fragment may cause hundreds of secondary particles. Most of the secondary particles are launched near the surface of the absorber, and the amount is quite closely proportional to the energy carried by the incident particle. Often the specific energy loss $-\mathrm{d}E/\mathrm{d}x$ for a given particle and material is a good approximation also when estimating the amount of secondary particles. Whereas the incident particles consisting of atomic nuclei travel through the absorbing material in somewhat direct path, the fast incident electrons take a lot more curved path through the absorbing medium. Electrons also lose their energy at a lower rate, but due to their mass equality with the orbital electrons of the absorbing medium the electron may also lose a major part of its energy in a single encounter. [12]

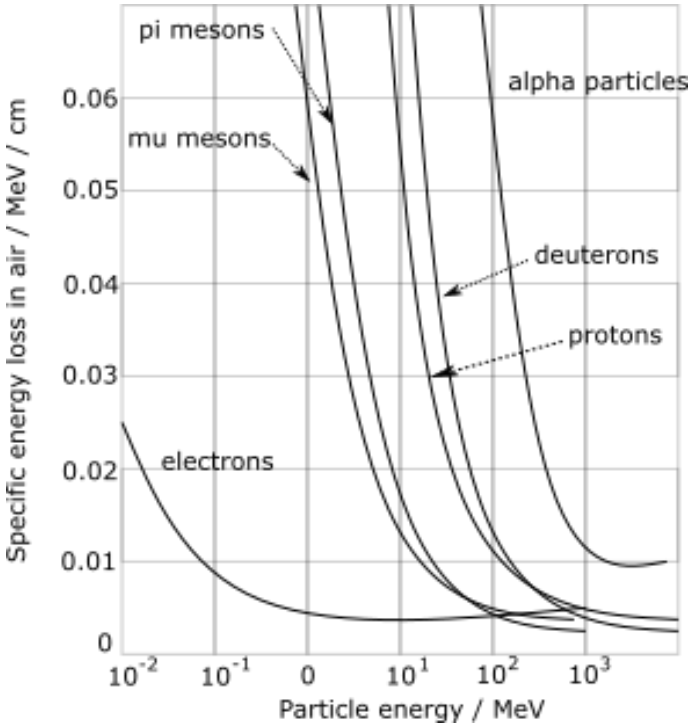

Fig. 5: Specific energy loss in air for particles with different energies. Adapted from [10].

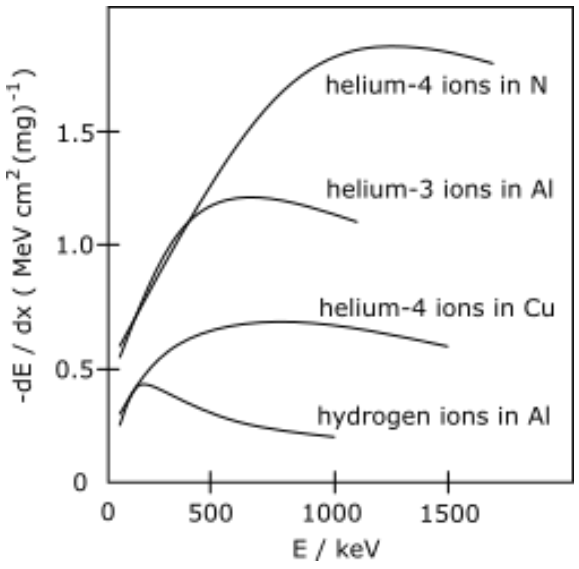

Fig. 6: Effect of ion initial energy to charge pickup when penetrating material. Adapted from [10].

2.3 Radiation processes

2.3.1 Electron processes

One common source for free electrons is a radioisotope that decays by beta-minus emission. The decaying nuclide can give the beta particle a considerable energy, ranging commonly from 0.1 to 1 MeV [4]. As the nuclide decays, its nucleus may be left in an excited state, which can relax by the process of *internal conversion* and yielding the conversion energy to one of its orbital electrons. The free electrons formed due to internal conversion usually have monoenergetic spectra ranging from keV to MeV range, and can usually be observed in the beta emission spectrum as spikes [13]. *Auger electrons* are similar to internal conversion electrons, with the difference being that the Auger electrons get their energy from the excited state of the atom itself, e.g. due to electron capture happened in the lower electron orbital and a higher orbital electron is fetched down to fill the gap. In the process an X-ray photon is emitted, and it may knock off one of the outer electrons from the atom giving it the leftover energy. As the energy received in the Auger process is much less than in internal conversion, the Auger electrons typically have energy of only a few keV, and are quickly absorbed or stopped by the surrounding material [13]. Most high energy electron radiation in space originates from plasma acceleration processes.

2.3.2 Electromagnetic processes

Braking radiation or *bremsstrahlung* is type of electromagnetic radiation that occurs when a charged particle is caused to decelerate, e.g. due to an electric or a magnetic field. It produces a continuous EM spectrum, and includes free-free emission caused by colliding electrons in a hot plasma, *cyclotron radiation* caused by a charged particle decelerated by a magnetic field, and *synchrotron radiation*, which is produced when a relativistic particle interacts with a magnetic field. The effects of special relativity must be taken into account in order to predict the wavelength of the resulting radiation. [14]

Compton scattering occurs when a high-energy photon collides with an electron, giving it a portion of its energy. The inverse Compton scattering happens when an electron gives its energy to a photon, making it a very high energy one. [15]

2.3.3 Heavy particle processes

Most of the heavy elements found in the universe are a product of nuclear fusion in stars, and the elements heavier than iron are forged in supernova explosions. The superheavy elements commonly go through spontaneous fission, in which two lighter elements are produced, and they can continue their fission until a stable element is reached. The final fission products are typically categorized into two medium heavy element groups, i.e the "light" and "heavy" groups centering around mass numbers 108 and 143 respectively. The main process producing helium atoms is the alpha decay, which happens when an unstable heavy element loses an alpha particle through barrier penetration mechanism. Elements with shorter half-lives generally yield more energetic particles. [16]

2.4 Radiation from the Sun

Sun fuses its hydrogen into helium with the rate of mass loss about $5 \cdot 10^9$ kg/s, of which 10^9 kg/s is exerted as solar wind and the rest as radiation [17]. The irradiance at Earth's radius is called the solar constant, which is approximately 1367 ± 3 W/m^2. Every proton-proton fusion in the core of the Sun creates a deuterium core along with an electron neutrino, which easily escapes the Sun through other all the other layers. The deuterium then fuses with a proton forming a helium-3 nucleus. This process also releases a gamma-ray photon. Furthermore, two helium-3 nuclei created this way can fuse together into a helium-4 nuclei yielding two free protons available for further reactions. [18]

Majority of the energy released is in the form of high energy photons. However, unlike neutrinos, they are not able to travel into space due to interactions with the Sun's material; photons constantly absorb and emit inside the radiation zone of 0.75 R_{Sun}. Considering a travel time of a single photon, it can be thought to take over 170 000 years for a photon to reach the outer layer of the Sun and be released into space. During this absorption-emission process the wavelength of the photons is transferred towards the lower energy spectrum, i.e. higher wavelength. [18]

The outer layer of the Sun is called the convection layer, and it consists of plasma flowing away from the Sun and back again after it has cooled down. Convection layer reaches into the thin visible surface of the Sun, the photosphere. The next layer of

the Sun's atmosphere is the chromosphere, in which the temperature rises, first slowly, but then very rapidly before reaching the corona of the Sun. Temperature in the corona rises to few million Kelvin in a distance of few thousand kilometres. [18]

Corona is the source of solar wind. The high temperature causes the H and He atoms to lose all their electrons, and many heavier atoms like N^{6+}, O^{6+}, O^{7+} and Fe^{13+} to be highly ionized as well. However, the solar wind consists mostly of protons, alpha particles and electrons. The temperature of the corona causes the solar wind to escape the Sun as supersonic flow of plasma, and the solar wind can be considered as an extension of the Sun's corona reaching all the way into the heliopause. [18]

Solar wind can mainly be considered to be of fast or slow kind, the fast wind reaching speeds of 800 km/s whereas the slow wind less than half of that. Fast solar wind originates from the coronal holes, and the slow wind from other areas of the corona. The particle content of the fast and slow solar wind differ a bit, as well as their density. At 1 AU, fast solar wind density is approximately about 3 cm^{-3}, which is about three times less compared to slow solar wind. Portion of alpha particles is about 2 percent in the slow solar wind and about 3.6 percent in the fast wind. The rest of the both kinds of wind is comprised of protons and electrons. The main energy scale of the solar wind particles lies between 0.5 keV to 10 keV. [18]

As the solar wind plasma is an excellent conductor, it causes the magnetic field to be frozen into it. Hence the magnetic field follows the solar wind all the way to the edge of the heliosphere. The magnetic field of the Sun is called the *interplanetary magnetic field (IMF)*. As the Sun rotates, the magnetic field close to it rotates along with it and can be considered radial. But further away as the magnetic field is carried with the expanding solar wind, the magnetic field gets wound around the Sun. This is known as the *Parker spiral* (depicted in figure 7), and at Earth's distance the angle of the spiral is 25-50°, depending on the speed of the solar wind. At the outskirts of the solar system the angle increases, and at the radius of Neptune it is close to 90°. Also, due to random fluctuations in the solar wind, the direction of the IMF may change significantly at Earth's radius. Further away, the rotation of the field gets more dominant. [18]

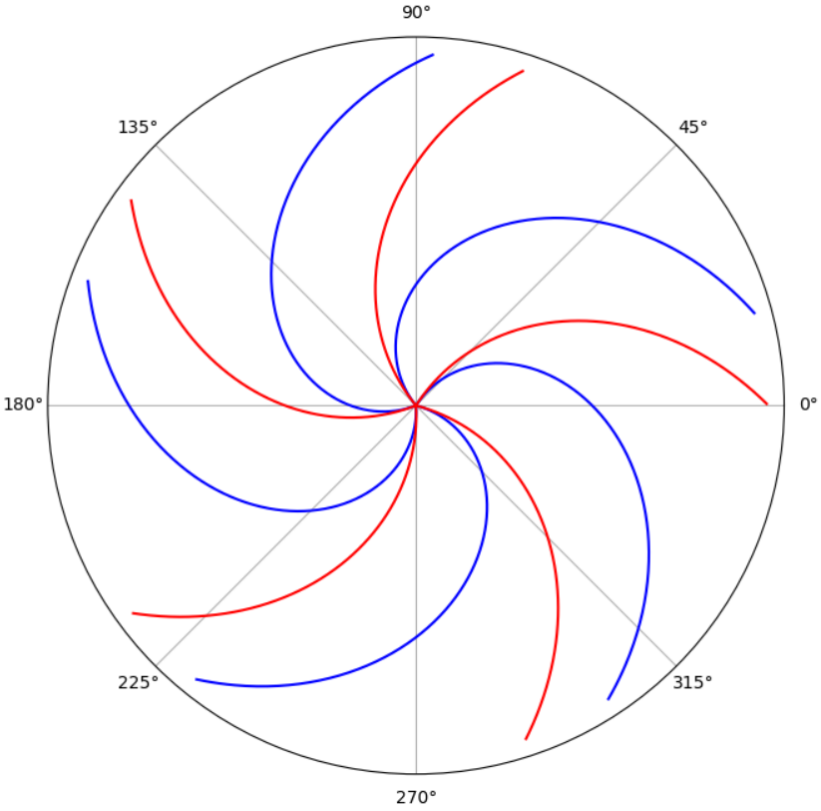

Fig. 7: The Parker spiral, showing the rotation of the magnetic field lines carried by the solar wind. As the velocity of solar wind is doubled, the field lines become more direct (red lines). Modelled using $\phi = \phi_0 - (\Omega/V) \cdot r$, where V is the velocity of the solar wind, Ω the Sun's rotational velocity and ϕ_0 the source longitude on the Sun.

2.4.1 Solar flares and coronal mass ejections

Solar flares and coronal mass ejections (CMEs) both are massive plasma phenomena of the Sun, and are caused by disturbances in Sun's inner structure. The disturbances affect through emerging magnetic flux in the photosphere and cause realignment of the fields in the corona. The field lines can suddenly reconnect and release enormous amount of material to space, consisting of ionized particles and electromagnetic radiation from visible light to x-rays. Solar flares usually cause a peak in electromagnetic radiation, whereas CMEs propel massive amounts of charged particles as well. Usually solar flares and CMEs occur at the same time, and solar flares usually are a precursor to a CME [19]. However, both can happen on their own, depending on the causing event on the Sun, and there is some controversy whether CME's and flares should be treated as the same or separate phenomenon [20].

2.4.2 Solar energetic particles

Solar energetic particles (SEPs) are high energy particles propelled from the Sun by a solar flare or a coronal mass ejection. They have energies ranging from 10 keV to relativistic GeV range [8], accelerated by a flare or a CME.

They are generally divided into impulsive or gradual SEP events, where impulsive events are powered by flares and gradual events by CMEs [8]. The particle content and the spread differs with the gradual and impulsive events, as does the frequency of the events [9]. Table 1 shows the typical properties of different solar events. Figure 8 shows a comparison of formation of gradual and impulsive events.

As the CME develops, three different phases can be seen: (i) initiation before the flare, (ii) an impulsive acceleration where the flare rises, and (iii) propagation phase, where the CME speed doesn't accelerate anymore. The CME speed stays approximately constant during the flare, but when there is no flare activity happening during the CME, the acceleration is slower (i.e. gradual acceleration phase). [8]

Acceleration of particles in the CME event occurs at the shock front of the CME, provided that the CME front is fast enough. The typical speed of a CME is more than 1500 km/s, but the slowest CMEs that are associated with SEP events may

Table 1: Characteristics of typical solar events. Adapted from [9].

Property	Impulsive	Gradual
Electron/proton	$\sim 10^2$ - 10^4	50-100
^{3}He/^{4}He	~ 1	$\sim 4 \cdot 10^4$
Fe / O	~ 1	~ 0.1
H / He	~ 10	~ 100
Q_{Fe}	~ 20	~ 14
SEP Duration	<1-20	<1-3 days
Longitude cone	<30°	<100°- 200°
Seed particles	Heated corona	Ambient corona or solar wind
Radio type	III	II
X-ray duration	$\sim$ 10 min - 1 h	$> \sim$ 1 h
Coronagraph	N/A	CME
Solar eind	N/A	IP Shock
Events/year	$\sim$ 1000	$\sim$ 10

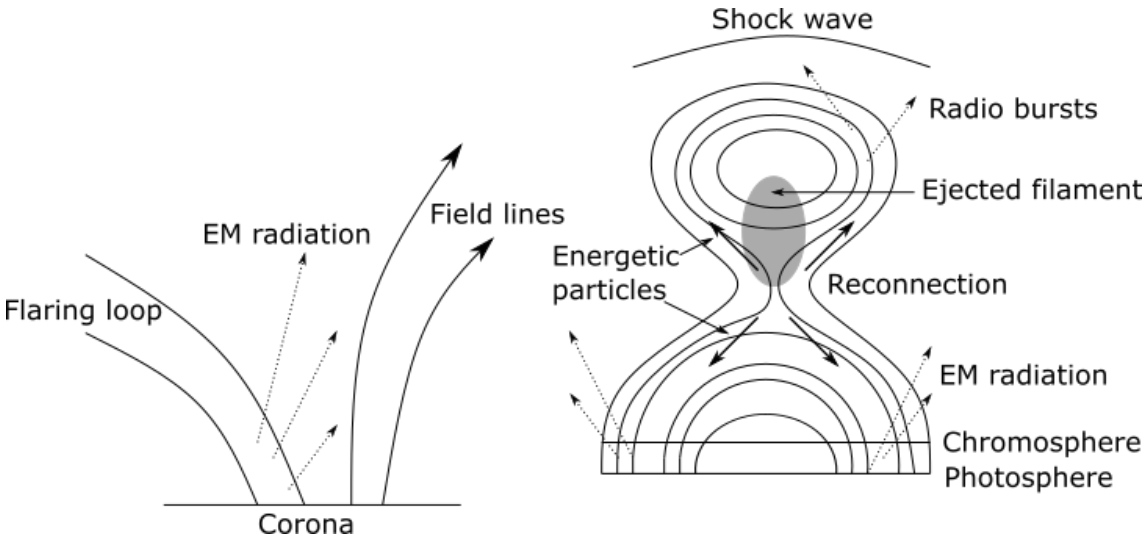

Fig. 8: Comparsion of particle acceleration in impulsive (left) and gradual (right) solar flares. Adapted from [8].

have speed of approximately 800 km/s [9], so they can hardly outrun the fast solar wind. The reconnection region also accelerates particles that then produce EM radiation when interacting with solar atmosphere, but most likely they are unable to escape over the CME shock front. SEPs are typically produced by impulsively accelerated CMEs, which are accompanied by coronal shocks and flares and move by constant speed. Prominence-related CMEs accelerate initially a longer time, and can produce a shock later compared to impulsively accelerated CMEs. Most of the higher energy particles seem to be accelerated closer to the Sun, suggesting that the dynamics near the Sun contribute a lot to the acceleration process [8][9]. Gradual SEP events are generally more interesting, as they propel high energy particles, including >10 MeV protons, which can pose a threat to astronauts and satellites above the low Earth orbit. [21]

2.5 Cosmic radiation

Cosmic radiation consists of particle or EM radiation originating from outside our solar system. It can come from inside the Milky Way or anywhere from the universe provided that the line-of-sight is clear. However, cosmic particles can travel back and forth bouncing from galactic magnetic fields. Solar activity affects the efficiency with which cosmic particles can reach us, e.g. in the form of solar cycles and sunspot activity; the lower energy end is subject to variations due to solar activity, and for the higher end spectrum not enough measurement data exists due to very low flux [22]. An approximate spectra of cosmic rays is presented in figure 9.

Cosmic particles are mostly high energy protons, but surprisingly large amount of light elements, including lithium, beryllium and boron, are found [24]. This is due to the interactions of cosmic rays while they travel through space to reach us; especially at energies of GeV range, heavy atoms from supernovas can collide with stationary atoms in galactic dust clouds and break into lighter elements [25]. Protons make up almost 90 percent of cosmic high energy radiation, whereas alpha particles have around 10 percent ratio and the remaining 1 percent consists of higher atomic number nuclei [26][27].

In recent decades many high energy EM sources have been found in other active galaxies, and especially intense X-ray radiation is a clear sign of other heavy radi-

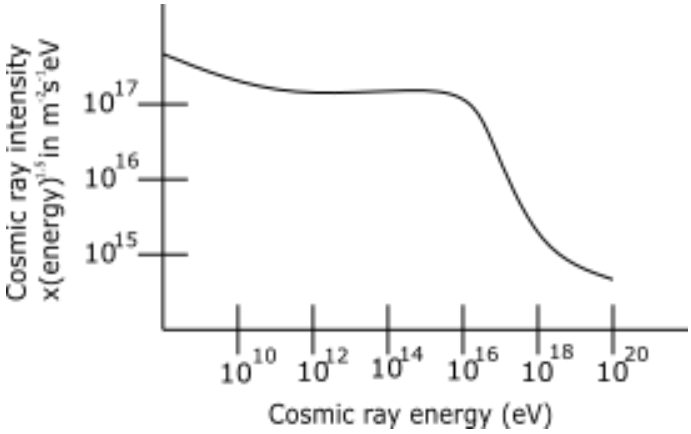

Fig. 9: Approximation of cosmic ray spectra around the Earth. Adapted from [23].

ation activity as well [28]. Our own Milky Way galaxy contains several sources of high energy X-ray and γ-ray radiation. Large portion of the high-energy radiation originates from the central regions of the galaxy, mainly from the accretion disks of massive black holes; as the material falls into the black hole, the speed of it increases constantly and it also interacts with other energetic material falling in, producing a lot of radiation [29].

Another great source of energetic cosmic rays are supernovae [25]. During the collapse protons of the matter convert to neutrons, releasing a burst of neutrinos in the process. Increase in neutrino flux is the first sign of a supernova, as was observed during the supernova of year 1987. Afterwards when the outer layers of the star have bounced outwards from the dense neutron core and from the radiation pressure of neutrinos, the immense temperature causes the supernova to shine brightly, releasing energetic cosmic rays in the process [30].

In addition to X-rays and charged particles, also neutrons and high energy gamma rays are observed from time to time. A neutron will decay by itself in less then 15 minutes, but a relativistic neutron can live longer through time dilatation and arrive from further, even anywhere inside our own galaxy. Gamma rays originating from outside our solar system are observed from time to time, like e.g. from Cygnus X-3 [31]. Also gamma ray bursts are seen sporadically, both longer and shorter in duration. Many theories of origins of the bursts have been formulated [32], e.g. shorter bursts have been suggested to be a result from two neutron stars

merging.

2.6 Radiation in Earth's vicinity

2.6.1 Earth's dipole field

Most magnetic fields can be approximated with a dipole field when observed from a distance. The dipole field is also the simplest approximation for a planetary magnetic fields, and it also forms a magnetic bottle, in which particles can be trapped. Near the ground the approximation doesn't hold so well, and also the axis of the dipole usually differs from the rotational axis. With Earth's magnetic field, this angle is approximately 11°. [33]

In the dipole field approximation the source of the magnetic field is considered a point, the field is defined so that it can be constructed from a scalar potential (i.e. $\nabla \times \mathbf{B} = 0$)

$$\Psi = -\mathbf{k}_0 \cdot \nabla \frac{1}{r} = -k_0 \frac{\sin\lambda}{r^2} \tag{9}$$

where $k_0 = \mu_0 M_E / 4\pi \approx 8 \cdot 10^{15}$ Wb m , $M_E \approx 8 \cdot 10^{22}$ Am^2 is the magnetic moment of the Earth and λ the latitude. This yields for magnetic field

$$\mathbf{B} = \frac{1}{r^3}[3(\mathbf{k}_0 \cdot \mathbf{e}_r)\mathbf{e}_r - \mathbf{k}_0]$$

which in componentwise representation reads

$$\begin{aligned} B_r &= -\frac{2k_0}{r^3}\sin\lambda \\ B_\lambda &= \frac{k_0}{r^3}\cos\lambda \\ B_\phi &= 0 \end{aligned}$$

where ϕ is the longitude. The dipole field with the coordinates is shown in figure 10. [33]

Guiding center approximation can be used with the dipole field provided that the particle's Larmor radius is much smaller than the field's radius of curvature

$$R_C(\lambda) = \frac{r_0}{3}\cos\lambda \frac{(1+3\sin^2\lambda)^{3/2}}{2-\cos^2\lambda}$$

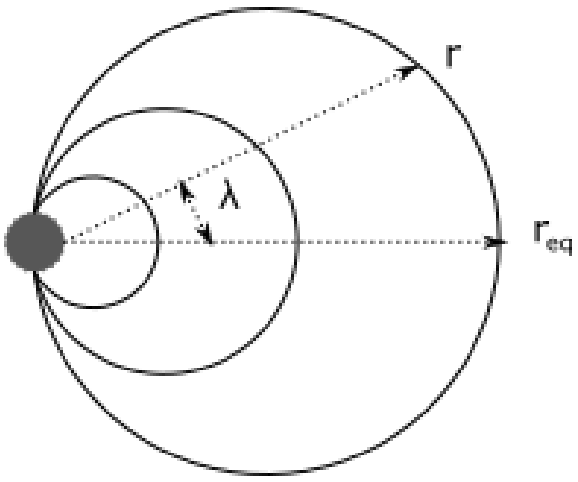

Fig. 10: The dipole field of the Earth. Adapted from [33].

Guiding centers of particles move across the dipole magnetic field of the Earth. The gradient and curvature of the field point inwards and the direction of the field is from north to south pole, and hence positive particles move to the west and negative particles to the east. [33]

2.6.2 Radiation areas of Earth's magnetosphere

The dipole field of the Earth blocks effectively the solar wind from entering directly to Earth, thus creating a cavity in the solar wind. With other planets with no prominent magnetic field, the solar wind is stopped by the *ionosphere*, which is induced by the incoming radiation. However, this induced magnetosphere is much weaker than an actual magnetosphere. For example Venus, Mars and comets near the Sun have induced magnetospheres. [34]

Earth's dipole field also blocks higher energy particles from entering the magnetosphere. The *geomagnetic cuttoff* prohibits majority of particles below 10 MeV from entering the inner magnetosphere, except on the magnetic poles where the cutoff goes to zero. At the equator level, protons from 10 to 500 MeV usually stop at the distance of 4 Earth radii, whereas cosmic ray protons with GeV energies can reach closer and over 15 GeV protons can reach Earth's upper atmosphere [35].

The magnetic field of the Earth collides with the magnetic field of the solar wind, creating a *shock front* at approximately 13 Earth's radii away towards the Sun. The

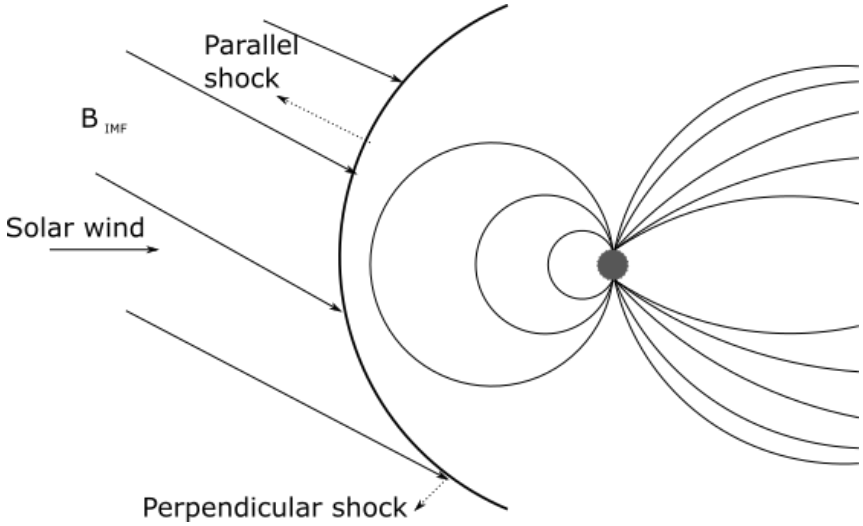

Fig. 11: The shock front of the Earth. Adapted from [36]

solar wind compresses Earth's magnetic field on the dayside, and creates a tail of it on the nightside. The distance where Earth's magnetic field ends is called the *magnetopause*. As the magnetic pressures $B^2/2\mu_0$ and the dynamic pressures $\rho_m V^2$ of the solar wind and atmosphere are equal at different distances, it creates a region of random plasma activity between the magnetopause and the shock called the *magnetosheath*. The magnetosheath is approximately 3 R_E thick directly towards the Sun and gets thicker on the sides. Earth's shock front is depicted in figure 11. [34]

At the point where magnetic field of the solar wind and the Earth's dipole field are perpendicular, the particles of the solar wind have a window to enter into the polar cusp. Other possibilities for the plasma of the solar wind to penetrate into Earth's magnetosphere are through diffusion or magnetic reconnection. [34]

Geostationary orbit (GEO) resides in the distance of 36000 km (6.6 R_E) and is an important location for several commercial satellites. The radiation content of the GEO may alter significantly during a coronal mass ejection. Inside the GEO the dipole field is quite a good approximation, and the radiation belts mainly reside within it. [34]

Van Allen belts

The Van Allen belts were discovered in the 1950's by the spacecraft Explorer I. The lower Van Allen belt resides at height of 5000 km from Earth's surface, whereas the higher one reaches height of 20000 km. The inner Van Allen belt consists mainly of protons and the outer one mainly of electrons [37].

Cosmic radiation particles are not so prevalent in the belts, but a large portion of solar wind particles are trapped into the magnetic bottle and have a very hard time trying to escape it [33]. The high-energy protons in the inner belt are believed to originate from the decay of neutrons produced in the Earth's atmosphere by cosmic rays [38], whereas the $<$50 MeV protons are solar energetic protons trapped in Earth's magnetic field [39]. Energies of the inner belt protons are typically in the range of 0.1 MeV - 40 MeV, whereas the electrons of the outer belt have energies from keV to MeV range [40]. Therefore most of the protons are non-relativistic, but a considerable portion of the electrons of the outer belt are in relativistic range. During magnetic storms, the electron amount of the outer belt can grow considerably. As about 10 percent of solar wind consists of helium ions, only the outer belt includes any considerable helium ion content [38].

Ring current

As the positive ions in the Earth's magnetosphere move to west and the negative electrons to the east, an electric currect goes around the Earth, towards west. The main carriers of the current are protons with an energy range of 20-200 keV, which is considerably lower energy compared to the particles residing in the radiation belts. However, the particle density in the ring current region is much higher. Ring current also reflects heavily on the events on the magnetosphere, and it can be used as an indicator for a geomagnetic storm. [40] Compared to the Van Allen belts, particles of the ring current are less effectively trapped, and hence the ring current dissipates over a calmer season of solar activity due to interactions with other regions' particles and through diffusion, causing proton precipitation to the atmosphere [41].

Low Earth orbit

Low Earth orbit (LEO) is the orbit in which the majority of man-made space objects reside, and therefore knowing the radiation conditions there is crucial to many space missions. LEO is up to 2000 km above the ground, and therefore it can co-exists with the inner Van Allen radiation belt at times depending on the space weather and magnetosphere conditions. The inner Van Allen belt may reach as low as few hundred kilometres above the Earth, especially on the south Atlantic anomaly, which is a cavity in Earth's magnetic field caused by uneven distribution of conductive material in Earth's crust. [42]

2.7 Radiation detectors

Radiation measurement in space adds some challenges to conventional radiation measurement environments. Comparing to ground-based measurement, space is much harsher due to harmful additional radiation. Therefore, the used instruments must be protected from other type of radiation than that what is measured to prevent the damage to the system. Also, the resources available are limited, and most detectors residing in space get their operational power from solar cells.

Silicon detectors are quite commonly used due to their low noise and high counting rate [43], therefore making them ideal also for space applications. Also, the manufacturing of silicon based detectors is relatively cheap and easy due to the amount of modern electronics facilities specialized in manufacturing of silicon chips.

2.7.1 Measurement techniques

Some commonly used measurement techniques include pulse shape analysis (PSA) and ΔE-E measurement. In PSA, the particle type is detected using the signal it causes in detector electronics. In ΔE-E measurement the energy of the incident particle is measured in two detectors [44]. This way the particle energy loss can be estimated using the properties of the detectors, such as materials and their thickness. Some materials for slowing down the incident particles can also be used, e.g. when detecting very high energy radiation. Also in some cases the particles need to be completely stopped, and then e.g. a scinttillator crystal (like cesium iodide) element

can be used [45]. For example, in FAZIA detectors [45] 1-2 silicon layers of thickness 300 to 500 μm are used in front of 4 cm CsI(Tl) layer for the final readout, and by using multiple detector configurations, also angle measurement can be achieved.

When using two detectors with ΔE-E measurement configuration, an informative visualization of the hits can be drawn. Considering the Bethe formula (equation 8) for non-relativistic particles ($1 \gg \beta^2 = (v/c)^2 = 2E/mc^2$), the final two terms can be approximated as

$$\ln\left(\frac{2m_e c^2\beta^2}{I(1-\beta^2)}\right) - \beta^2 \approx \ln\left(\frac{2m_e v^2}{I}\right)$$

and replacing $\beta^2 = 2E/mc^2$ and $v^2 = 2E/m$, the whole equation becomes

$$\begin{aligned} -\frac{\mathrm{d}E}{\mathrm{d}x} &= \frac{4\pi}{m_e c^2} \cdot \frac{nz^2}{2E/mc^2} \cdot \left(\frac{e^2}{4\pi\epsilon_0}\right)^2 \cdot \ln\left(\frac{4m_e E}{Im}\right) \\ &= \frac{mz^2}{E} \cdot \frac{4\pi n}{2m_e} \cdot \left(\frac{e^2}{4\pi\epsilon_0}\right)^2 \cdot \ln\left(\frac{4m_e E}{Im}\right) \end{aligned}$$

As the ln term changes relatively slowly as a function of E, it can be approximated as a constant. Grouping it with the other constants as $C = 4\pi n e^2/2m_e \cdot (e^2/4\pi\epsilon_0)^2 \cdot \ln(4m_e E/Im)$ the approximation for ΔE as a function of E can be obtained:

$$\Delta E = \frac{Cmz^2\Delta x}{E}$$

It can be seen that the heavier the particle, the greater the energy loss is when energy is the same. Particles with different mass and nuclear charge create a different hyperbola when visualized. An example of this can be seen in figure 12. For a real-world example, ΔE-E measurement is implemented in the compact Radiation Monitor (RADMON) of Aalto-1 CubeSat, as it uses a silicon detector and a CsI(Tl) scintillation detector in series to measure low-MeV electrons and >10 MeV protons [46][47].

When using a silicon strip detector in multilayer 3D configuration, then in addition to incident particle energy, they can provide more accurate time-of-flight (TOF) and angle information, as well as more accurate and broader energy range information [48]. For example, the AMS-02 telescope aboard the International Space Station

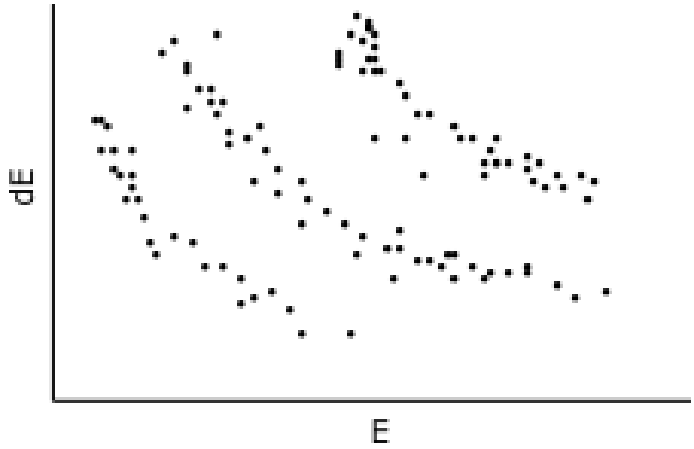

Fig. 12: Example of an output of a ΔE-E measurement, where each hyperbola formed by the measurement dots corresponds to a different particle type.

consists of 9 silicon tracker layers along with several other tracker types such as time-of-flight measurement unit and a magnet for measuring the Larmor radius and thus the rigidity of the incident particle. [49]. The telescope is capable of measuring leptons and hadrons separately, with energies from MeV to TeV range, along with angle information. For identification of leptons, a boosted decision tree classifier is used to distinguish the 3D particle shower shapes.

When measuring large amounts of particles, the recovery time of the detector must be fast enough. Also, when considering a low resource environment with a limited link speed where to send the measurement data to, automation of the categorization and analysis of the detector signal will become beneficial as it most probably is not possible to send all the raw measurement data home. However, it might become challenging to implement a sufficient automation solution considering e.g. the available power.

3 Machine learning

Machine learning (ML) commonly denotes a class of computational methods, in which a mathematical model based on a set of training data is constructed. However, the term may also be used when describing any statistical method performed on a dataset in order to extract some relevant information on, e.g. using a covariance matrix to check dataset values for correlation while adjusting e.g. the threshold of every parameter based on their importance. [50] Multiple ML methods can also be used together to provide more comprehensive evaluation of the dataset, and this is called *ensemble learning*, of which examples are regression trees and random forests [51].

In its most basic form, ML can denote any type of regression or classification task, e.g. binary classification of dataset values into two different possible outcome groups using a simple linear classifier function or more complex polynomial non-linear classifier [52]. The classification threshold can be modified by adjusting the parameters of the system depending on the dataset. Multiple functions are needed when categories are added.

Other type of functions may also be used when categorizing the data. One commonly used function type is the radial basis function (RBF), in which the value of the function depends only on the distance of the origin. This enables classification in scenarios where there are multiple localized target groups, and thus classification is achieved using lesser amount of functions. [53] The benefit of using an RBF function over a linear function collection can be seen in figures 13 and 14, where the classification of the same dataset is achieved using only one RBF function for which 4 linear functions are needed.

Sometimes it may be difficult to distinguish the values of the dataset using simple classification methods. One commonly used helping method is the *kernel trick*, in which a dimension is added into the dataset, with which it is easy to perform the classification. The requirement of this is that the extra dimension can be calculated easily using original variables. [54]

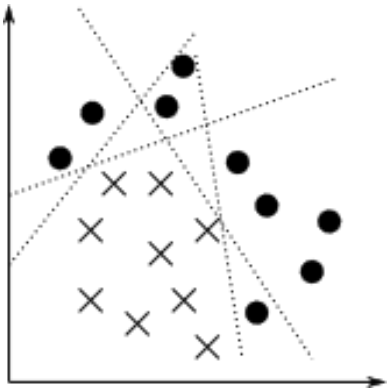

Fig. 13: Value set separated by four linear functions.

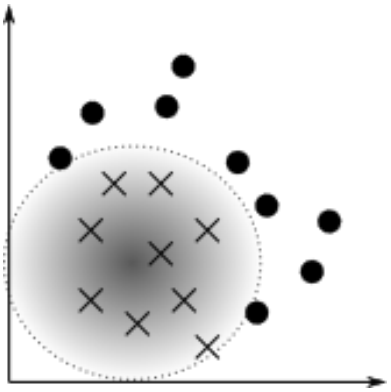

Fig. 14: Value set separated by a single radial basis function.

3.1 Preprosessing of data

Processing data using machine learning methods is very data-heavy. In many applications it is mandatory to reduce the amount of data to be processed to be as compact as possible without losing the most crucial bits of information. A common method to achieve compactness is dimensional reduction of the input data. In context of machine learning, the term *feature learning* [55] is often used when analysing the most important properties of the dataset in question.

3.2 Feature selection

When performing real world machine learning tasks, it is usually most beneficial to receive some information on the features of the dataset right away, without using any heavy computation as principal component analysis. Different *feature selection* methods are meant for simplifying the data before it is fed to a machine learning solution [56]. One of the most straightforward ways of performing feature selection is to use the correlation matrix, as in first stages of principal component analysis [57]. Ruling out features based on a threshold value of their variance is an easy task, after which the dataset can be processed with the remaining features.

Chi-squared test (χ^2 test) is a common statistical technique that calculates the amount of features present in the positive and negative values of the dataset. When the measurable variables are counts, the reduced χ^2 test can be used, and it is

defined as

$$\chi^2 = \sum_{k=1}^{n} \frac{(O_k - E_k)^2}{E_k}$$

where O_k and E_k are respectively the observed and expected counts of the feature k in the dataset of n cases. The expected count E_k can be calculated e.g. by using another set of values from another measurement of the same type.

3.3 Neural networks

Neural networks are an attempt to replicate the behaviour of biological brain and implement the way a brain processes data using an artificial system. Basically neural networks are a collection of functions working together on an input value set. The functions' inputs and outputs may be connected to each other arbitrarily, but usually the functions are in the form of layers and the data moves into one direction.

3.3.1 Basic concepts

The basic element of a neural network is the *neuron*, which models a real-life brain neuron. It can have several input and output connections from and to several other neurons. The inputs of the neuron are combined using a certain rule, and the output is calculated based on the desired function of the neuron. Most elementary type of a neuron is the perceptron, which takes a single or multiple inputs and calculates a binary output value based on them, e.g. by using a certain threshold value. [58]

Commonly neurons have an assignable weight value for every input, and it is used to emphasize or diminish the value from a certain input. Thus it's possible to customize the neuron for a certain type of task by selecting the input values that are meaningful considering its output. The output "intensity" of the neuron can be controlled by setting a bias value, which commonly is simply added (or subtracted) to the calculated output. For a simple perceptron neuron, the output o can be calculated using the definition

$$o = \begin{cases} 1 & \text{if } \sum_{j=1}^{n}(w_j i_j) + b > 0.5, \\ 0 & \text{otherwise.} \end{cases}$$

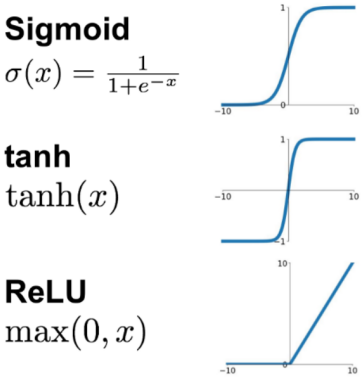

Fig. 15: Common activation functions. Figure from [60]

where i_j is the jth input, w_j is the weight for the corresponding input and b is the bias. [59]

Real biological neurons don't produce binary outputs, but have some kind of smooth rule on which the output is calculated. A straightforward way to accomplish this is to run the neuron's output function through an *activation function*, which ultimately defines the amplitude of the output [60]. Common activation functions are e.g. tanh and the Sigmoid function, which is defined as

$$S = \left\{1 + \exp\left[\sum_{j=1}^{n} (w_j i_j) - b\right]\right\}^{-1} \tag{10}$$

It can be seen that the Sigmoid function yields a result between 0 and 1, and is thus beneficial for simulating the neuron's output. When considering classification, the activation function of the neuron defines which kind of classification is performed, e.g. linear classification is achieved using a linear activation function. An important detail is the limit of the functions; whereas the Sigmoid and tanh functions have a limit, the ReLU function does not, and this has to be taken into consideration to prevent exploding gradients [61]. Three commonly used activation functions are compared in figure 15.

Neural networks commonly have multiple *layers*. The layers consist of neurons. The layer that receives data from outside the neural network is called the input layer,

whereas the layer that sends data out from the network is the output layer. The layers in between are called hidden layers, and they don't have a direct connection to outside world, and in simple networks are only connected to the previous and consecutive layer. [58]

3.3.2 Types of neural networks

Neural networks exist in several different types, usually depending on the application. In addition to different neuron types or properties, e.g. activation function and use of bias, neural networks can have very different structures. The connections between neurons are usually modelled as synchronous, but in pursuit of real life neurons they also may be asynchronous, taking into account the different distance between neurons. Most commonly neural networks differ in their connectivities between neurons and layers.

Linear classifier is one of the simplest machine learning constructs, and it is used to classify data based on a classifier function f, while y being the result, $\vec{w}$ the weight vector and $\vec{x}$ the input vector:

$$y = f(\vec{w} \cdot \vec{x}) = f\Big(\sum_j w_j x_j\Big)$$

Most commonly linear classifier uses some threshold value for the classifier function to separate different input cases. It is also possible to use multiple linear classifiers to accomplish more delicate detection of input cases, e.g. when positive cases lie only between small area. This was visualized in figure 13. Simple linear classification is achieved using a single neuron, in which the classifier function (i.e. the activation function) is some linear function. [62]

Feedforward neural network (FNN) is the most common type of neural network. In it the input layer is connected to consecutive layers of processing neurons and finally to the output layer. In FNN the data flows forward layer by layer, and the output values of neurons can be calculated easily as vector or matrix operations. By selecting a first layer neuron count to be lower than the number of inputs, the FNN will effectively perform a feature selection in the style of principal component analysis as it reduces the number of dimensions in the input space and only the

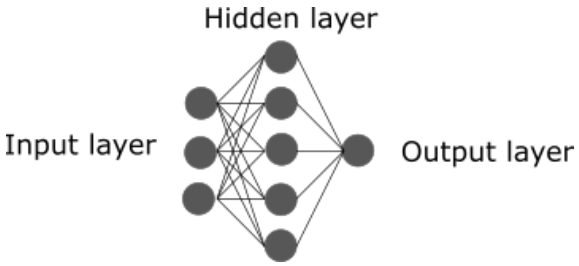

Fig. 16: Simple feedforward neural network with a single hidden layer.

most important features will have an effect on the output. For example *autoencoder* works this way. [58] An FNN with one hidden layer is depicted in figure 16.

Convolutional neural network (CNN) is more sophisticated version of a feedforward neural network. It is especially tailored for image and other spacial information handling tasks, where the consecutive values of the input data are related to each other in some way, e.g. as pixels in a picture are. CNNs achieve dimensional reduction and also scale well to bigger data sets, which makes them a prominent choice in many real-world applications. The way CNNs work is that the first layer takes patch samples of the data, and the patches include partially the same values. In a case of 2D image, this could be e.g. 3x3 patch of pixels that moves through the whole image row by row. Different type of filters can be applied to the patch, e.g. maximum or mean value, depending on the desired usage. After filtering and pooling the dataset has shrunk to much smaller size. Then if necessary, i.e. if the dataset contains too many dimensions, flattening is performed making the multidimensional data a 1D vector, which then can be fed into a standard FNN layer, i.e. fully-connected layer. CNN can also be implemented in 3 dimensions. A 2D CNN is depicted in figure 17. [63]

Any neural network that has more than one hidden layer can be called a *deep neural network* (DNN) [64]. However, many times deep neural network denotes some more complex construct than just linear feedworward neural network with steady layers of neurons. Deep neural networks usually have different types of neurons mixed together, they may have connections backwards as well and can even be a combination of several different standard neural network types. Prime modern example of a DNN is the *generative adversarial network* (GAN) that uses two neural

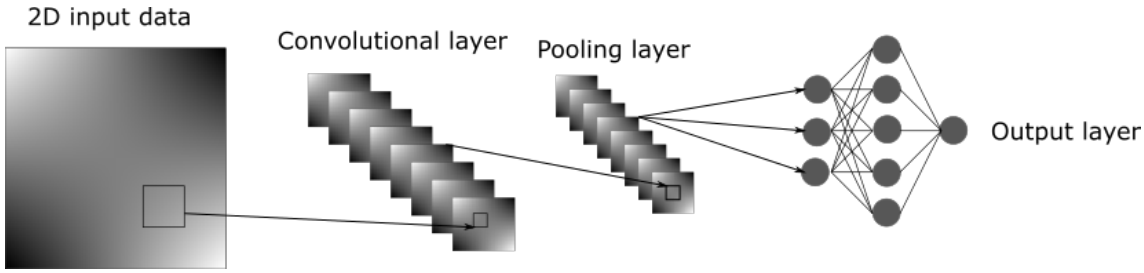

Fig. 17: Typical convolutional neural network with the initial convolutional layers and the feedforward layers before the output.

networks working against each other [65]. The other network is the trainee and the other is the trainer, which has a set of parameters towards which the trained network is schooled. Training of DNNs is usually very computationally heavy, but the resulting neural nets can have very impressive capabilities ranging from realistic image creation to self-operating vehicles.

3.3.3 Training neural networks

Training of a neural network means adjusting the values of the network's parameters in order to provide better results in respect to the training values. Two main categories of training methods are supervised and unsupervised training. The difference between the two methods is that in supervised training the network is provided with a goal output related to the given input, whereas in unsupervised training there's no target output but the network will be left to decide the target configuration with more vague boundary parameters. This often requires the network to be designed using more intelligent neuron structure, i.e. that the neurons are "intelligent" enough to so that they can guide the network into the right direction. [66][67]

Most common method of supervised training is called propagation training. In it the trained neural network is fed with a set of input values with a desired output. The used training algorithm goes through a set of iterations with the data until the error rate produced by the neural network is within the desired limit. [66][67]

During each training iteration the network loops through the data by doing a forward pass and a backward pass. The forward pass means just inputting the network data and measuring the output from the output layer. Some network designs

may store the outputs of each individual layer. The backward pass calculates the error rate, which is the difference between all actual and desired output values. Then it goes through all the layers of the network in reversed order correcting the values of neuron parameters. [66][67]

3.3.4 Basic training parameters

Training sample means one single value of training data, e.g. address details of a person or any set of multidimensional data.

Batch size is the amount of training samples that are propagated through the neural network. Usually this is a portion of the dataset, as it may not be cost effective to use all the values of the dataset, but e.g. one tenth of it. This requires less memory and speeds up the training of the network. [68][66]

Training epoch defines an operation in which the whole training set, i.e. every batch, is fed through the network once. Iterations is the number of batches needed to complete one epoch. [69]

Learning rate of the neural network defines how large adjustments the training algorithm makes during every training run. Learning rate is one of the most crucial parameters of the network, as improper values might lead e.g. to an exploding or vanishing gradient problem, in which the weight and bias values of the neurons grow or shrink uncontrollably rendering the neurons useless for the learning task. Many times it is difficult to determine the correct learning rate for the learning task, and therefore the learning process might need to be restarted several times. [70]

During training, the neural network keeps a statistic of the progress using loss and accuracy metrics. Accuracy is a percent, but loss is not; depending on the chosen neural network parameters, loss is a cumulative error rate occurred. The *training accuracy* and *training loss* are the accuracy and loss during the running of the epoch. After an epoch the performance is tested using *validation accuracy* and *validation loss*, with which the goodness of the neural network usually is defined; when an adequate validation loss is achieved (or when it doesn't change anymore), the training is stopped to prevent overfitting. [66]

3.3.5 Cost function

Important concern when implementing a neural network is to assess how cumbersome it is to get it to perform ideally. This can be summarized by a cost function. Cost function involves summing all the differences between output and input as a function of all weights and biases of the neural network, and in a simple feedforward neural network when using a quadratic error the cost function has a form

$$C(x, w, b) = \frac{1}{2n} \sum_{x} ||y(x) - a(x, w, b)||^2 \tag{11}$$

where w and b are a collection of weights and biases of the neural network neurons, x the input vector, y the desired output vector, n is the total number of training inputs and a the actual output vector, i.e. the activation of a neuron. Also, the cost of a single training example is [67]

$$C = \frac{1}{2}||y - a^L||^2 = \frac{1}{2} \sum_{j} \left(y_j - a_j^L\right)^2$$

As the neural network performs a backward pass and modifies the values of neurons, what it actually needs to calculate is how to minimize the cost function (11) when weight (or bias) values of the neurons change. During backpropagation, every neuron has to be taken into account, and that results in a very computationally heavy operation for any larger neural network. Reason of slow initial development of the neural networks was exactly the computational cost of performing the backpropagation. The gradient descent algorithm was the first usable way to train the neural network somewhat effectively. [67][66]

3.3.6 Design considerations and optimisation

Number of layers needed for the neural network to perform a task varies a lot, and also there is no single consensus which is the correct amount. However, many simple tasks are usually well possible with only one hidden layer. If the dataset values are linearly classifiable, there is no need for hidden layers at all, but everything can be done using only one layer of neurons that acts as both as an input and an output layer. [71][72]

Good practice in most cases is to have the first layer neuron count equal to the dimensions of the dataset. This ensures that no variable is overlooked in the initial training phase. No definite rules exist, but what has proven to work well in most cases is to select the number of hidden layer neurons between the input and the output layer. [73]

One practical formula for most supervised cases of hidden layer neuron count is [74]

$$N_h = \frac{N_s}{\alpha \cdot (N_i + N_o)}$$

where N_h is the number of hidden layer neurons, N_s the number of variables in the dataset, α a scaling value and N_i and N_o the number of inputs and outputs. The value for α usually varies between 2 and 10, depending on the neural network.

Neuron count of the output layer is directly proportional to the task at hand. E.g. if the neural network's task is to perform four-class classification, two output neurons should be enough to provide four different output values. However, for better certainty usually four output neurons are used. In many real-world scenarios the best way to find a working neuron configuration is to run experiments. This is especially true if the configuration of the dataset is clandestine, e.g. there are many variables and the relation between them is not clear right away. Different kinds of test loops that train and measure the neural network with different parameters can be ran, e.g. *genetic algorithms* select a few best parameter candidates and improve upon them until there is only one clear winner configuration, and maybe try to improve that little further. [74]

Most commonly every neural network has a group of neurons that at the end of the training prove to have little to no effect on the end result. Usually it is therefore best to eliminate these neurons completely save computing resources. This can be computationally hard to find out when a larger neural network is in concern, but most likely pruning will be worthwhile. [71]

In the training phase of neural network when measuring the expected output value of a neuron in relation to actual output value, the neural network makes adjustments to the weight and bias values based on the difference. Depending on the used algorithm, it's possible that some cells get adjusted constantly too much or too little, and their values become very large or very close to zero. This renders the

neuron either to output zero or the maximum value independent of the input, thus making the neuron useless. The neuron's activation function is the most important factor affecting this. E.g. using the RELU function some neurons might end up exploding. Using a function that has an upper bound, e.g. the Sigmoid or tanh function, one can prevent exploding gradients. The functions were visualized in figure 15. [75]

3.3.7 Overfitting and underfitting

Overfitting and underfitting are very common issues associated with neural networks. They occur when the employed neural network model is not optimal to process the dataset that is being fed to it. This can also mean that the used dataset is not optimal, e.g. it contains a lot of irrelevant cases which falsify the learning of the model. Clear signs of overfitting and underfitting while training the neural network are the inconsistencies when comparing validation and training loss; e.g. when the neural network overfits, it is learning the training set really well (training loss decreases), but it fails to capture the most essential parts of the dataset and thus fails to improve the performance with actual test cases. [76]

Quite often the first reason for overfitting and underfitting is the complexity of the model; if the used dataset implies only linear relationship with the data to be categorized, a multilayer neural network with non-linear perceptron neurons will surely overfit it. However, if the used dataset is very complex, it will require an equivalently complex neural network to be able to handle it. Depending on the training algorithm, overfitting and underfitting may occur naturally, and in this case one can add *dropout layers* between the standard layers. The dropout layers will randomly cut off connections between neurons to prevent their overusage. [76]

In some cases, in e.g. when the same neural network model is used to learn multiple different datasets with variable complexity, a complex neural network may be required but not at all times. To prevent overfitting or underfitting, one can then constantly monitor the training process of the model and stop the training when the accuracy is no longer changing in a desired rate. When considering the dataset, it is also possible to perform feature selection before entering the dataset into the neural network. This will help the neural network to target the variables that are

the most meaningful, and prevent overfitting and underfitting that way. [76]

4 Implementation

4.1 Choosing the machine learning environment

Several ready made solutions were investigated for this work as well as an custom implementation with .NET Framework from scratch. However, it quickly became clear that implementing neural networks require countless hours of optimization and proofing in order to work in any more complex than the simplest of scenarios. Therefore a ready made solution has to be used.

4.1.1 Considered solutions

Neurosolutions is a Windows-based neural network software, which offers a wide variety of different neural network types and data computation possibilities [77]. However, apparently the software has become very commercial these days opposed to what it was years before, and only short trial runs of the program were available. This fact added to the apparent licensing costs and further usability considerations with small-scale neural networks didn't make Neurosolutions the most beneficial looking solution.

Encog is a Java and .NET based neural network framework, which also offers a wide range of different machine learning constructs [78]. It was one of the first non-commercial neural network solutions available, and though it gained some popularity in the past, it seems majority of its users have moved on to another solutions leaving Encog seem a bit outdated.

Deeplearning4j [79] is another free neural network framework, but as the time of writing this, it was primarily only available for Java, which made it a bit more cumbersome to work with under the used tools. Also, as it didn't seem to be so popular compared to leading solutions, it wouldn't have a strong ecosystem of examples and support.

Neural networks are also available through using Matlab and its *Deep Learning Toolbox*, or formerly *Neural Network Toolbox* [80]. Although the performance of Matlab is well recognized in scientific computing, the solution most likely is too heavy considering the simple task at hand. Also as with Neurosolutions, the commercial version comes with a licensing cost.

4.1.2 Google TensorFlow

Google TensorFlow is a neural network framework that has gained a lot of popularity in the last years. It is free, and it has a lot of examples and tutorials available, and also it supports various programming languages and environments. [81]

KERAS API is a high-level library that provides straightforward access to TensorFlow libraries [82]. It can be used for faster prototyping, testing and it's ideal in simpler neural network models and when looking for an optimal neural network solution. It runs on top of Python programming language, and in addition to CPU processing, it supports GPU processing.

4.2 Simulation framework Geant4

CERN provides a free net-based simulation tool Geant4, which offers a massive amount of sophisticated simulations for particle collisions [83]. The simulation results can be used to train the neural network. Geant4 software can also be downloaded and used on a desktop computer directly. However, any realistic real-world simulation requires processing power beyond desktop PCs, and they are performed on a cluster supercomputer.

Geant4 provides simulations for several different physics processes , i.e. they each describe how particles interact with a material. Seven major categories of processes are provided by Geant4: electromagnetic, hadronic, decay, photolepton-hadron, optical, parametrization, and transportation. Also a great set of different materials is available through standard Geant4 libraries, and it's also possible to create custom materials by defining the properties e.g. conductivity. [84]

Commonly the Geant4 software is meant to be used in Linux operating systems. However, as the program core doesn't rely on any external operating system dependent libraries, it can be used in any environment, provided that the compiler supports the C++ version Geant4 is created with. For this work, the program was compiled under Windows using Microsoft Visual Studio 2015 and its VC++ compiler. Refer to Appendix A.1 for usage under Windows.

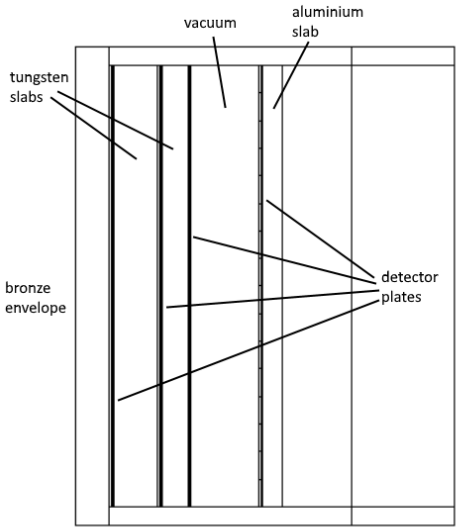

Fig. 18: Geant4 detector model with different parts shown.

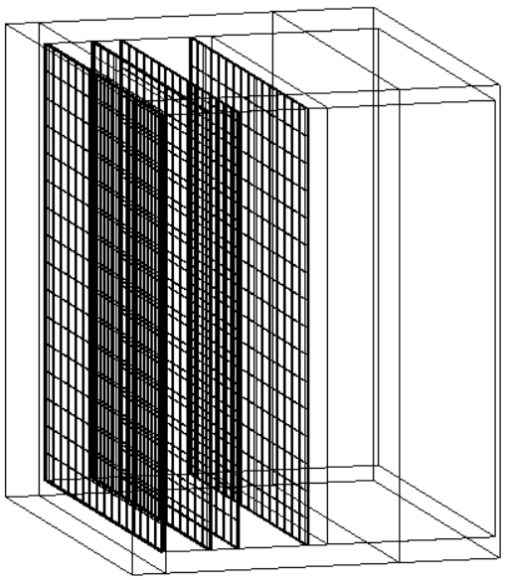

Fig. 19: Geant4 detector from an angle visualizing the 4 detector plates consisting 16 × 16 cells each.

4.3 The detector model

Geant4 provides a very comprehensive set of example programs for various particle interaction situations. For this work, a model for a 4 tier calorimeter with each tier consisting of 16 × 16 detector cells was selected and modified. Each cell is 6 × 6 mm of area and 300 micrometres thick. The housing of the detector is 4 mm thick bronze, and it prevents most of the low energy radiation from entering from the sides. There's an aluminium slab 4.3 mm thick in front of the detector entrance. Between the first and the second silicon detector cell grids there is a vacuum. Before the third and the fourth cell grid there are two tungsten slabs, which slow down, scatter and create showers of the incident particles. The first tungsten slab 5.5 mm and the bottom tungsten slab 9.5 mm thick. Around the silicon detector cells there are 0.5 mm of clearance. The detector layout is visualized in figures 18 and 19.

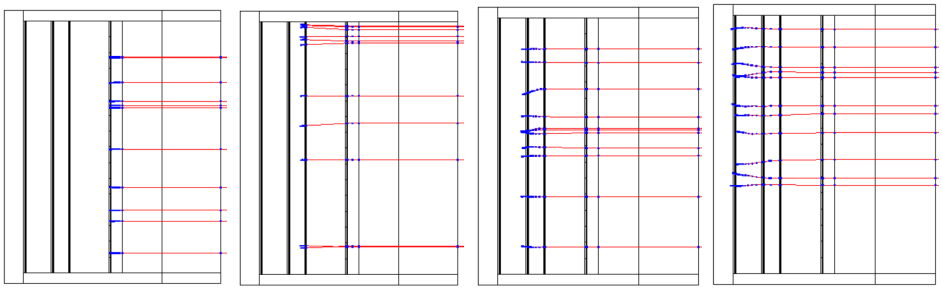

Fig. 20: Detector penetration depth with protons. From left to right: 30 MeV, 50 MeV, 100 MeV and 150 MeV.

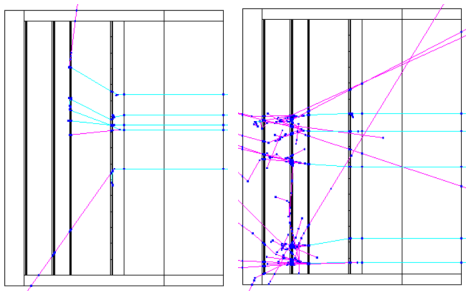

Fig. 21: Detector penetration depth with electrons, left with 5 MeV and right with 50 MeV electrons.

4.4 Simulated data

Simulation included a set of Monte Carlo valued set of certain particle type from arbitrary directions, and the visualization tool shows the secondary particles also their further interactions with any other detector plates. These collisions can be recorded into a text file as a table from which the neural network program can read the events for processing. Also, Geant4 supports a visualization of the events. For the constructed model, the penetration depths of various particle energies is visualized in figure 20 - the first detector plate is reached with the proton energy of 30 MeV, the second plate with 50 MeV, third at 100 MeV and the final plate at 150 MeV. Electrons behave differently causing loads of secondary particles, which can reach the detector plates even though the main particle has combined previously. This is visualized in figure 21 with 5 and 50 MeV electrons.

As the goal is to work with an electron spectrum and compare it to galactic

proton spectrum, the Geant4 program was modified to return a random energy value for each incident particle in a given interval. For a more realistic spectrum, the program was also extended to give energy values based on a real measured spectrum. Refer to Appendix A.4.1 for the code used.

4.5 Neural network implementation

As the purpose of the program is to be able to distinguish between different particles and their energies, the task at hand is a classification problem. The multilayer detector configuration can be considered a geometric or image recognizing issue.

The most natural choice for a neural network architecture would be one which natively takes into account the 3 dimensions of the input data. One of these architectures is the convolutional neural network (CNN). CNN should be ideal also in a sense that it can take into account the neighbouring values of a single value in the dataset, so it should provide the most natural way of detecting particle collisions and the secondary products hitting and scattering from the detector.

However, convolutional neural networks can be computationally very heavy, and as the goal in this work is to find a suitable solution for usage in space, with most likely limited resources, CNN implementation may be out of question; training and finding optimal parameters with a desktop PC takes a lot more time and the completed solution most likely would need heaps of more working memory for temporary data storage. Furthermore, it is generally easier to handle single-dimension data in a software, all the cells in the different layers of the detector can be thought as a one big vector consisting all the detector plates, while each detector cell being a single variable in the dataset. The problem therefore can be considered a sequence classification issue. However, as the positional data would be useful to take into account, rows of hits of each detector cell were combined one after another, i.e. row 1 of detector plate 1 is followed by row 1 of detector plate 2 and so on. However, this naturally induces some skewing of the dependencies between detector cells. However, as the consecutive plate information for a single row per plate is reserved, most likely the neural network can figure out some dependencies. The forming of a single vector from detector plate rows is illustrated in figure 22.

Most likely the dataset will contain a large number of futile variables. Therefore it

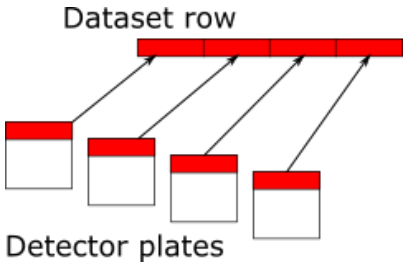

Fig. 22: Process of restructuring the input data from detector cells into a single vector.

would be wise to perform some kind of dimensionality reduction on the data. When considering the what a neural network does to the input data when the number of neurons is reduced on a hidden layer, dimensionality reduction will be achieved. However, it also might be useful to perform some reductions of variables even before feeding the dataset into the neural network. As we are not interested in particles arriving sideways into the detector, events which hit only one layer can be forgotten. Also events starting from 3rd layer up can be neglected, as there is not sufficient amount of cells to provide a clear detection on a physical basis then.

Neural networks are many times used in finding hidden features of the dataset which are not apparent or not even clearly definable. Also they perform dimensional reduction when the number of neurons on a consecutive layer is reduced below the input neuron count (i.e. the length of the input vector), so it was decided to attempt to use the whole dataset without filtering it first and feed it to the neural network.

The optimal design parameters were found using a nested for loop, in which primary parameters are varied one by one and a test run is executed. The parameters were recorded and the best values were presented at the end of the loop. The parameters included the amount of hidden layers, the size of the layer, number of training epochs and training batch size. For the CNN, the convolutional parts till the first densely connected layer were left alone, as looping through them would have been very time consuming. Instead, a common basic structure for 3D CNN was selected, and the densely connected part of it was optimized. The chosen designs for the CNN and the FNN are presented in figures 23 and 24, in which the input vector is on top, and flows down through different layers. The types of the layers are on the left of each layer, and on the right side the input and output data formats.

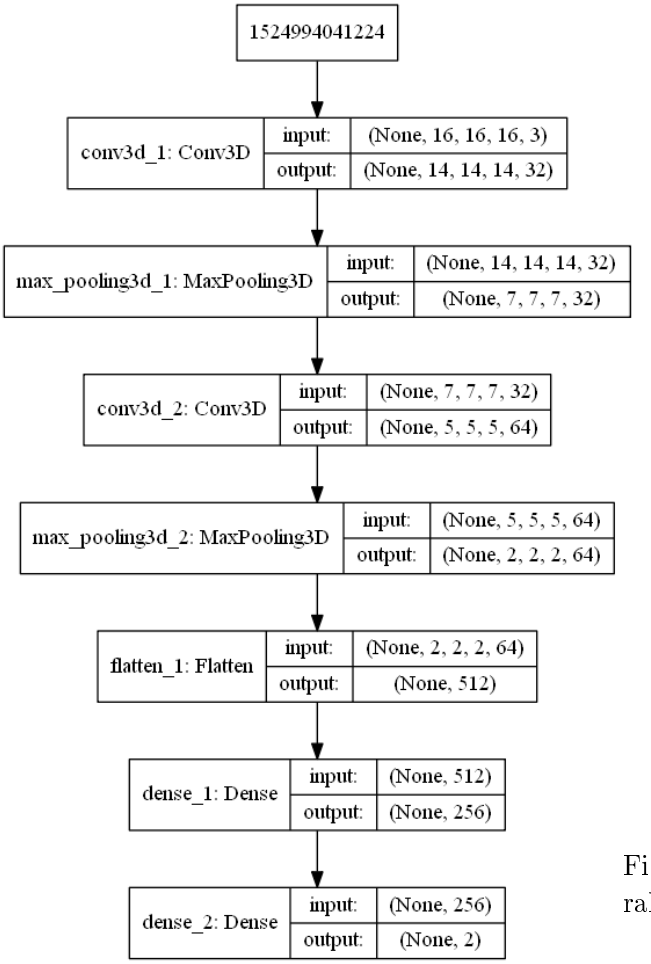

Fig. 23: Design for the 3D convolutional neural network.

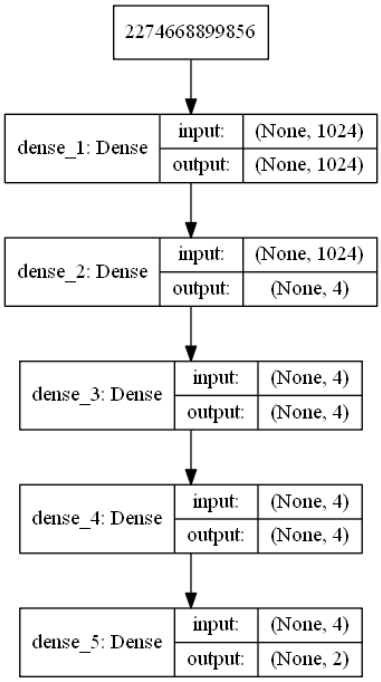

Fig. 24: Design for the feedforward neural network.

Number of indices tells the number of dimensions, and the length of each dimension is specified. *None* means the length of the input/output dimension is arbitrary. For example, it can be seen that 3D CNN operation requires 5 dimensional vectors for data handling, and the standard FNN parts operate in 2 dimensions.

5 Testing the solution

The selected neural network designs were tested with dataset from the Geant4 program with 6000 values for training, and another 6000 split between validation and the actual testing. The main test dataset was created with a flat probability of particle energies from the specified range. Finally also the realistic spectrum was tested. As convolutional neural networks consume a lot of resources, practical testing of much bigger datasets is not feasible on a desktop PCs. Therefore same amount of values was mainly used with FNN for a fair comparison. Also, with FNN the effect on increasing the amount of training value was studied, as well as using binary energy values for the test data.

It is noteworthy to mention that as the initialization of neural networks uses random values for weights and biases of individual neurons, the test accuracy is a little different in every run. For the testing, the best outcome was recorded, as it is possible to save the weights for the neural network for the actual solution and thus get the exact same configuration when the best configuration is found.

Detection accuracy in the tests denotes the neural net's ability to distinguish between the two used datasets; the accuracy tells the amount percentage of correct predictions when testing the trained neural network using the portion of the dataset reserved for testing. 50% of the test set values is from the first dataset and the other 50% from the other used dataset, and the whole test set is tested with each run.

5.1 Convolutional neural network

Computationally 3D CNN can be very intensive, and it also proved to take a lot of time to train the neural network in order to get an acceptable detection accuracy. Furthermore, running a single training session on a CNN takes over an hour, whereas with an FNN around 30 seconds. With 6000 values of flat $<$50 MeV electrons vs flat $<$2 GeV protons, the CNN configuration reached 80% detection accuracy. Refer to Appendix A.2.1 for test run.

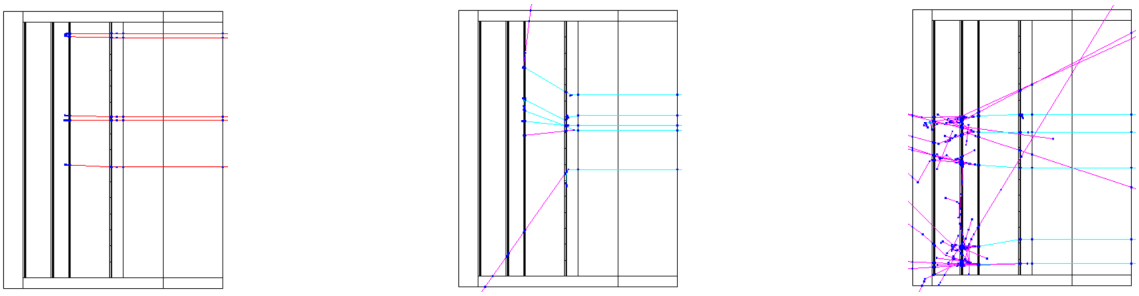

Fig. 25: Comparison of proton and electron penetration depths and energy signatures. 50 MeV protons on the left, 5 MeV electrons in the middle and 50 MeV electrons on the right.

5.2 Feedforward network

Compared to CNN, the feedforward network was considerably less heavy to train. Also, it proved to give convincing initial accuracy, without needing to tweak the parameters too much. Despite the fairly simple configuration on the network, with 6000 training values of flat <50 MeV electrons vs flat <2 GeV protons an accuracy of 95% was reached (refer to Appendix A.3.3). Also a test run with 140000 values was conducted, and it didn't prove to improve the detection accuracy. A test run is presented in Appendix A.3.4.

5.3 Distinguishing protons from electrons

Main task of the design was to be able to separate electrons from protons along with their properties including energy and angle. Of most interest is the case when the energy signatures in the detector overlap as much as possible.

With similar energies, electrons produce much more secondary particles, whereas protons tend to travel through the detector and combine with the detector material without further interactions. This is illustrated in figure 25 with 5 MeV electrons and 50 MeV protons and electrons. Because of the energy signature, similar energy monoenergetic particles can be separated easily by the neural network, reaching 99% detection efficiency (refer to Appendix A.3.1). Figures 26 and 27 illustrate the whole test sets of 50 MeV protons and electrons, showing the activations produced by them and their secondary particles.

When the proton energy is 5 MeV and the electron energy 50 MeV (both monoenergetic), the energy signatures look pretty similar (refer to figures 28 and 25),

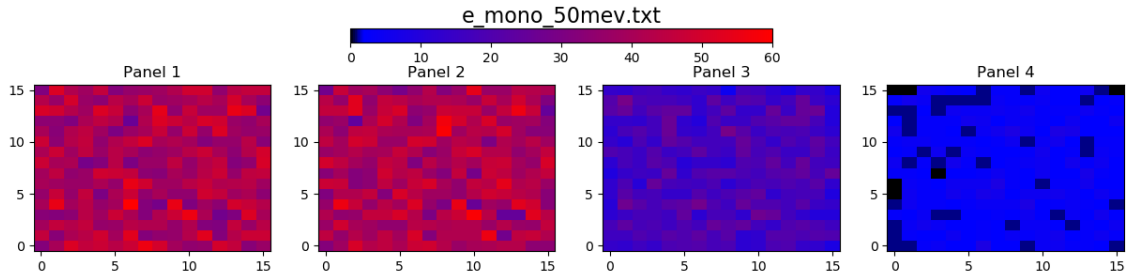

Fig. 26: Hits of 6000 monoenergetic 50 MeV electrons to the detector cells with all the activation counts of the detector cells from primary and secondary particles. It can be seen that after the first two plates the the penetration is considerably hindered.

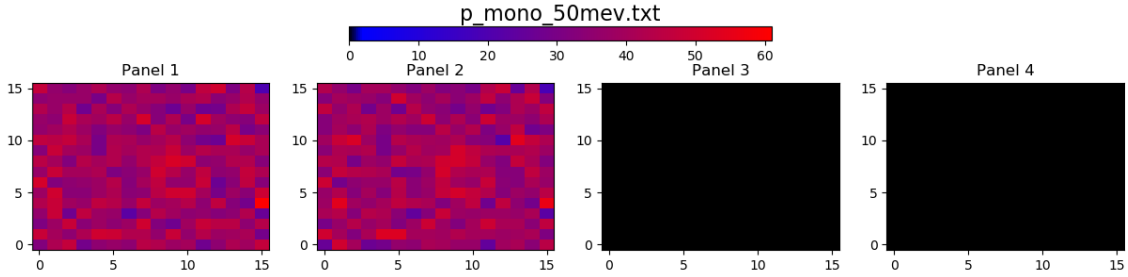

Fig. 27: Hits of 6000 monoenergetic 50 MeV protons, showing the complete stopping after two detector plates.

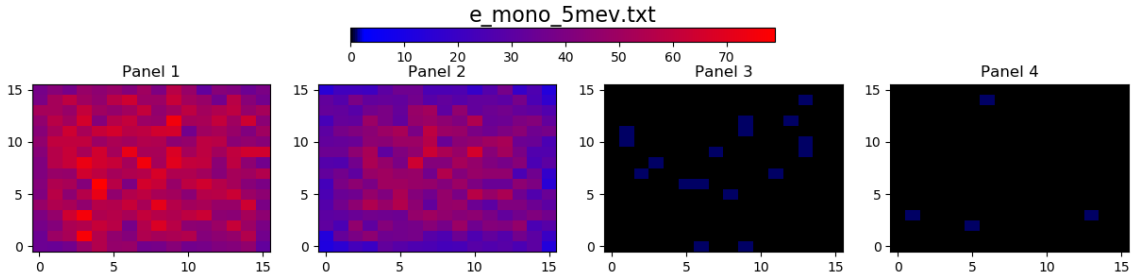

Fig. 28: Hits of 6000 monoenergetic 5 MeV electrons. Most of the hits come from the first two detector plates, showing a similar trend compared to 50 MeV protons.

and this leads the neural network to struggle a bit when distinguishing the particles from each other; an accuracy of 96% can be achieved. Refer to Appendix A.3.2 for the test run.

In addition to flat spectra, another spectra were tested. As galactic protons are never very low energy ones, also a proton spectrum with starting energies from 30 MeV, reaching up to 2 GeV, was used. Against an electron spectrum of <50 MeV the detection accuracy drops a bit, and reads 94% (Appendix A.3.5).

The effect of zenith angle was also tested with direct 50 MeV protons vs angled 5 MeV electrons (both monoenergetic). In Geant4 simulation the angle has to be defined with the ratio of axis, e.g. 2x, 2y reprecent an angle of $\arctan 2/2 = 45°$. With electron angles of $\arctan 1/5 = 11.35°$ and $\arctan 1/2 = 26.57°$ the neural net seemed to still be able to separate the particles pretty easily with detection rates of about 98 % (refer to Appendix A.3.8 and A.3.9), which is a bit better than with 0° zenith angle.

5.4 Binary energy values

The neural network program was constructed in a way that using a switch it is possible to only take a hit yes/no value for every cell which is hit. This was done as it is not certain whether the physical detector solution is going to have energy measurement available in the detection cells, but only a pulse signaling a hit. As logically expected, using flat <50 MeV electrons vs flat 30-2000 MeV protons, the detection efficiency drops a bit due to decrease of information, reaching 93% accuracy. Refer to Appendix A.3.6 for test run.

5.5 Realistic spectrum

Testing with spectra with an equal probability of high and low energy particles is a little unrealistic. In reality, e.g. many-GeV particles are much less common. However, when training the neural network, it is beneficial to make sure that the network can equally well detect particles at any range. For more realistic spectra testing, the Geant4 program was modified to give particle energies based approximately on actual measured Jovian electron spectrum [85] and galactic proton spectrum [86].

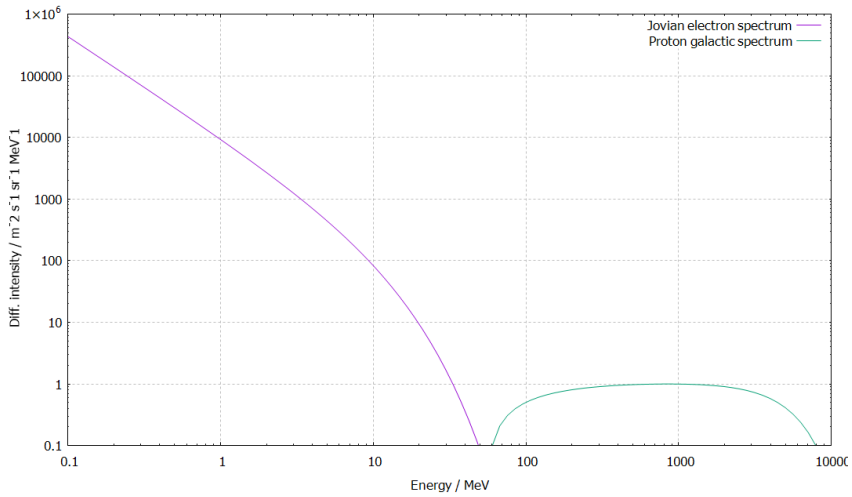

Fig. 29: Differential intensities of electrons and protons of the simulated realistic spectrum.

The equation for electron spectrum is

$$N_{e-}(E) = I_D \cdot (E/E_0)^{\gamma} e^{-E/E_{break}}$$

where $I_D = 10285.7\ \mathrm{m}^{-2}\ \mathrm{s}^{-1}\ \mathrm{sr}^{-1}\ \mathrm{MeV}^{-1}$, $\gamma = 1.63$ and $E_{break} = 9.4$ MeV. For the proton spectrum, the measured spectrum was fitted with a gaussian function using the least squares method with the equation

$$N_p(E) = a \cdot e^{\frac{(E-b)^{-2}}{2 \cdot c^2}} + \frac{d}{E}$$

where $e = 2.71828$ and the fit parameters $a = 1.11864$, $b = -271.451$, $c = 3776.26$ and $d = -60.9773$. The generating code used can be seen in Appendix A.4.1. The spectra are shown in figure 29. Using this spectrum also for training, the neural network achieves 93% accuracy. The test run is shown in Appendix A.3.7.

5.6 Using different values for testing and training

Finally, the neural network solution was tested by using a different training value set and a testing set. This enables to investigate whether the network is able to e.g. distinguish discreet particle energies when only trained with a spectrum. The

network was trained with flat 30-2000 MeV protons and flat <50 MeV electron spectrum, and it was verified how well it can distinguish between monoenergetic electrons and the proton spectrum. Some drops in the detection efficiency can be seen as a function of electron energy: 5 MeV (94%), 10 MeV (95%), 25 MeV (95%) and 50 MeV (84%). However, it can be said that the network still can detect the particle type pretty well. Comparing to values when specific electron energies are trained against the proton spectrum show some differences: 5 MeV (97%), 10 MeV (97%), 25 MeV (96%) and 50 MeV (92%). The testing runs are listed in Appendix A.3.10 and A.3.11.

In the same manner, also the variation of zenith angle was tested with the training of <50 MeV flat electron spectrum vs flat 30-2000 MeV proton spectrum, and the configuration was tested with monoenergetic 5 MeV electrons in 11-26° angle. This way the network reached 80% (11°) and 91% (26°) accuracy. The testing is shown in Appendix A.3.12 - little bit strangely the network worked better with a greater zenith angle.

6 Discussion

Based on the Geant4 model, the electrons and protons have very different energy signature when hitting the detector. Electrons tend the create a lot of secondary particles, whereas protons interact mildly when hitting the detector. Most natural way of distinguishing the particles would be to compare the signatures. However, when the proton energy is approximately ten times that of the electron energy, the signatures seem alike, especially with electron energies from 3 to 5 MeV and proton energies from 30 to 50 MeV. The main difference then seems to be that protons usually cause the consecutive reactions themselves, whereas electrons cause secondary reactions, which don't necessarily reside on the same trail as the primary particle.

Unlike e.g. the machine learning solution used aboard the AMS-02 spectrometer (a boosted decision tree) [49], a simpler solution suitable for nanosatellite realization was searched, but it sets certain limitations. It would be useful to have more processing power onboard for more complex categorization, as well as more downlink bandwidth (9 Mbits on the AMS-02 compared to few kbits on the FORESAIL satellites).

TensorFlow provides an extensive amount of parameters for the neural networks. These include different loss functions, optimization methods, various different activation functions for different layers and many other tweaks which can be beneficial for a certain type of application. Especially the parameters and the design of the CNN could be further investigated, as most likely it would reveal many ways to improve efficiency and performance of the network, even though the CNN seems to be quite a complex solution even in smaller applications; the variable count needed for the CNN is great.

Pruning of neurons might be useful, especially considering the solution to be implemented, as in space environment saving energy and materials is always a bonus.

When considering the implementation of the network on a physical solution, *NVIDIA TensorRT* could be suitable solution. It is an inference library that optimizes the designed and trained neural network by compiling it into machine code while performing all modern compiler optimizations on the source code, and thus making the network faster and more compact, especially when parallel processing

can be utilized in the form of (NVIDIA) GPUs. TensorRT includes direct support for TensorFlow, and can therefore be used to create a delivery-ready neural network for various environments. [87] However, if the chip onboard a satellite is as complex as a GPU, the housing must be sufficient for proper radiation protection.

At the moment the data from the Geant4 client has to be fetched separately and then manually fed into the neural network program. For easier usage and faster experimentation, a single client program which controls both the Geant4 simulation and the neural network could be fairly straightforward to implement. The Geant4 C++ client could be used directly from another .NET program, and the Python TensorFlow solution could be ported fairly easily into it also. For this a .NET version of TensorFlow would naturally create more cohesion to the software without needing to use the Python and its libraries.

The next step developing the solution would be to test and investigate the possibilities of multi-category detection, as the detector would need to be able to distinguish particles along with their zenith angles and energies. The architecture of the designed TensorFlow program enables the loading of arbitrary number of datasets for the neural network to separate, but the categorizing and target value mapping currently is rather crude and would need to be verified to work properly with multiple categories. However, as the two-category tests with different zenith angles, energies and also ranges of energies seemed to provide mostly great results, using this kind of solution for multi-category scenario might be possible. Especially if e.g. another neural network could be implemented alongside for a specific property, e.g. one neural network for only detecting energy, other for zenith angle and a simple master network section for deciding the final category of the particle.

7 Conclusions

This thesis work was about researching for a feasible machine learning solution for particle separation in low resource environment, such as a small satellite. For this, the radiation environment of Earth's magnetosphere and particle dynamics was first investigated, and after that some modern machine learning techniques were presented. Also some possibilities of physically implementing the solution were discussed.

Keeping in mind the simplicity requirement of the upcoming solution, the light-weight and well-supported TensorFlow library was chosen for the task. Several neural network types were studied, and the most obvious choice for the neural network type for 3D image detection was the convolutional neural network. However, it quickly became clear that the resource consumption both in the training and in the future implementation phase would be massive. Therefore, a simpler feedforward neural network solution was studied, and its parameters, such as number of layers and neurons per layer, were found using automatic test loops.

The final solution done in this work consisted of a network with 3 hidden layers of 4 neurons each shown in figure 24, as it seemed to still provide promising results without being too complicated.

Only two categories of particle types were tested at the same time. The final solution would require to be able to distinguish between particle energies, zenith angles and particle type, but this solution at the moment only can separate two categories of particle types with different energies and zenith angles. However, it seems to do it very well, and e.g. changing the zenith angle and using overlapping and different energy scales doesn't affect the detection so much. The detection rates are mostly over 95%, which can be considered very good. Complete list of test results is found in Appendix A.5.

References

[1] R. Vainio. *Pienet satelliitit ratkomassa avaruusfysiikan suuria kysymyksiä.* [referenced 15.12.2019]. URL: https://www.helsinki.fi/en/news/science-news/pienet-satelliitit-ratkomassa-avaruusfysiikan-suuria-kysymyksia.

[2] M. Palmroth et al. "FORESAIL-1 CubeSat Mission to Measure Radiation Belt Losses and Demonstrate Deorbiting". In: *JGR Space Physics* 124.7 (2019), pp. 5783–5799. DOI: https://doi.org/10.1029/2018JA026354.

[3] T. Yiu. *Understanding neural networks.* [referenced 18.12.2019]. URL: https://towardsdatascience.com/understanding-neural-networks-19020b758230.

[4] G. Knoll. *Radiation Detection and Measurement, 3rd edition.* John Wiley & Sons, 2000, pp. 3–6.

[5] H. Koskinen. *Johdatus plasmafysiikkaan ja sen avaruussovellutuksiin, 2nd edition.* Helsingin yliopisto, 2011, pp. 31–38.

[6] H. Koskinen. *Johdatus plasmafysiikkaan ja sen avaruussovellutuksiin, 2nd edition.* Helsingin Yliopisto, 2011, pp. 41–43.

[7] *Fermi acceleration.* [referenced 5.1.2020]. URL: https://en.wikipedia.org/wiki/Fermi_acceleration.

[8] M Kallenrode. "Current views on impulsive and gradual solar energetic particle events". In: *Journal of Physics G: Nuclear and Particle Physics* 29.5 (2003). DOI: https://doi.org/10.1088/0954-3899/29/5/316.

[9] M. Desai and J. Giacalone. "Large gradual solar energetic particle events". In: *Living Reviews in Solar Physics* 13.3 (2016). DOI: https://doi.org/10.1007/s41116-016-0002-5.

[10] G. Knoll. *Radiation Detection and Measurement, 3rd edition.* John Wiley & Sons, 2000, pp. 30–33.

[11] K. Kleinknecht. *Detectors for particle radiation, 2nd edition.* Cambridge University Press, 1998, p. 9.

[12] G. Knoll. *Radiation Detection and Measurement, 3rd edition.* John Wiley & Sons, 2000, pp. 42–43.

[13] K. Kleinknecht. *Detectors for particle radiation, 2nd edition.* Cambridge University Press, 1998, pp. 1–2.

[14] I. Robson. *Active Galactic Nuclei.* Wiley, 1996, p. 123.

[15] I. Robson. *Active Galactic Nuclei.* Wiley, 1996, p. 131.

[16] G. Knoll. *Radiation Detection and Measurement, 3rd edition.* John Wiley & Sons, 2000, pp. 9–10.

[17] M. Zuber and D. Smith. "Measuring solar massloss and internal structure from monitoring the orbits of the planets". In: *Lunar and Planetary Science* XLVII (2017). DOI: https://www.hou.usra.edu/meetings/lpsc2017/pdf/2281.pdf.

[18] H. Koskinen. *Johdatus plasmafysiikkaan ja sen avaruussovellutuksiin, 2nd edition.* Helsingin Yliopisto, 2011, pp. 129–135.

[19] NASA. *CME Week: The Difference Between Flares and CMEs.* [referenced 9.1.2020]. URL: https://www.nasa.gov/content/goddard/the-difference-between-flares-and-cmes.

[20] M. Youssef. "On the relation between the CMEs and the solar flares". In: *NRIAG Journal of Astronomy and Geophysics* 1.2 (2012), pp. 172–178. DOI: https://doi.org/10.1016/j.nrjag.2012.12.014.

[21] D. Reames. "The two sources of solar energetic particles". In: *Space Science Reviews* 175.53 (2013). DOI: https://doi.org/10.1007/s11214-013-9958-9.

[22] R. Clay and B. Dawson. *Cosmic Bullets.* Allen & Unwin pty lt, 1997.

[23] R. Clay and B. Dawson. *Cosmic Bullets.* Allen & Unwin pty lt, 1997, p. 13.

[24] R. Clay and B. Dawson. *Cosmic Bullets.* Allen & Unwin pty lt, 1997, p. 92.

[25] L. Drury. "Origin of Cosmic Rays". In: *Astroparticle Physics* 39-40 (2012), pp. 52–60. DOI: https://doi.org/10.1016/j.astropartphys.2012.02.006.

[26] CERN. *Cosmic rays: particles from outer space.* [referenced 12.1.2020]. URL: https://home.cern/science/physics/cosmic-rays-particles-outer-space.

[27] W. Suparta1 and S. Zulkeple. "Spatial Analysis of Galactic Cosmic Ray Particles in Low Earth Orbit/Near Equator Orbit Using SPENVIS". In: *Journal of Physics Conference Series* 495.1 (2014). DOI: https://doi.org/10.1088/1742-6596/495/1/012040.

[28] I. Robson. *Active Galactic Nuclei*. Wiley, 1996, p. 163.

[29] I. Robson. *Active Galactic Nuclei*. Wiley, 1996, p. 228.

[30] R. Clay and B. Dawson. *Cosmic Bullets*. Allen & Unwin pty lt, 1997, pp. 54–68.

[31] R. Clay and B. Dawson. *Cosmic Bullets*. Allen & Unwin pty lt, 1997, p. 106.

[32] B. Abbott et al. "Search for Gravitational Waves Associated with 39 Gamma-Ray Bursts Using Data from the Second, Third, and Fourth LIGO Runs". In: *Physical Review D* 77 (2008). DOI: 10.1103/PhysRevD.77.062004.

[33] H. Koskinen. *Johdatus plasmafysiikkaan ja sen avaruussovellutuksiin, 2nd edition*. Helsingin Yliopisto, 2011, pp. 45–51.

[34] H. Koskinen. *Johdatus plasmafysiikkaan ja sen avaruussovellutuksiin, 2nd edition*. Helsingin Yliopisto, 2011, pp. 163–183.

[35] B. Kress et al. "Modeling geomagnetic cutoffs for space weather applications". In: *JGR Space Physics* 120.7 (2015), pp. 5694–5702. DOI: https://doi.org/10.1002/2014JA020899.

[36] H. Koskinen. *Johdatus plasmafysiikkaan ja sen avaruussovellutuksiin, 2nd edition*. Helsingin Yliopisto, 2011, p. 159.

[37] K. Kleinknecht. *Detectors for particle radiation, 2nd edition*. Cambridge University Press, 1998, p. 202.

[38] Encyclopedia Britannica. *Causes of auroral displays*. [referenced 15.1.2020]. URL: https://www.britannica.com/science/ionosphere-and-magnetosphere/Causes-of-auroral-displays.

[39] W. Li and M. Hudson. "Earth's Van Allen Radiation Belts: From Discovery to the Van Allen Probes Era". In: *JGR Space Physics* 124.11 (2019), pp. 8319–8351. DOI: https://doi.org/10.1029/2018JA025940.

[40] H. Koskinen. *Johdatus plasmafysiikkaan ja sen avaruussovellutuksiin, 2nd edition*. Helsingin Yliopisto, 2011, pp. 177–181.

[41] D. Summers et al. "Energetic Proton Spectra Measured by the Van Allen Probes". In: *JGR Space Physics* 122.10 (2017), pp. 10, 129–10, 144. DOI: https://doi.org/10.1002/2017JA024484.

[42] S. Martines. "Analysis of LEO Radiation Environment and its Effects on Spacecraft's Critical Electronic Devices". MA thesis. EMBRY-RIDDLE AERONAUTICAL UNIVERSITY, 2011.

[43] Z. Fei et al. "Readout electronics of silicon detectors used in space cosmic ray charges measurement". In: (2013). DOI: arXiv:1307.3041v1.

[44] N. Le Neindre et al. "Comparison of charged particle identification using pulse shape discrimination and ΔE-E methods between front and rearside injection in silicon detectors". In: *NUCL INSTRUM METH A* 701 (2013), pp. 145–152. DOI: https://doi.org/10.1016/j.nima.2012.11.005.

[45] S. Carboni et al. "Particle identification using the ΔE-E technique and pulse shape discrimination with the silicon detectors of the FAZIA project". In: *NUCL INSTRUM METH A* 664.1 (2012), pp. 251–263. DOI: https://doi.org/10.1016/j.nima.2011.10.061.

[46] P. Oleynik et al. "Calibration of RADMON radiation monitor onboard Aalto-1 CubeSat". In: *Advances in Space Research* (2019). DOI: https://doi.org/10.1016/j.asr.2019.11.020.

[47] J. Gieseler et al. "Radiation monitor RADMON aboard Aalto-1 CubeSat: First results". In: *Advances in Space Research* (2019). DOI: https://doi.org/10.1016/j.asr.2019.11.023.

[48] T. Pázmándi et al. "Space dosimetry with the application of a 3D silicon detector telescope: response function and inverse algorithm". In: *Radiation Protection Dosimetry* 120.1-4 (2006), pp. 401–4. DOI: https://doi.org/10.1093/rpd/nci539.

[49] N. Tomassetti. "AMS-02 in space: physics results, overview, and challenges". In: *Nucl. Part. Phys.* 265-266 (2015), pp. 245–247. DOI: 10.1016/j.nuclphysbps.2015.06.063.

[50] K. Hao. *What is machine learning?* [referenced 17.2.2019]. URL: https://www.technologyreview.com/2018/11/17/103781/what-is-machine-learning-we-drew-you-another-flowchart/.

[51] T. Mitchell. *Machine Learning.* lecture notes [referenced 23.2.2019]. URL: http://www.cs.cmu.edu/%5C%7Etom/mlbook-chapter-slides.html.

[52] S. Date. *The Binomial Regression Model: Everything You Need to Know.* [referenced 3.3.2019]. URL: https://towardsdatascience.com/the-binomial-regression-model-everything-you-need-to-know-5216f1a483d3.

[53] T. Ahadli. *Most Effective Way To Implement Radial Basis Function Neural Network for Classification Problem.* [referenced 24.2.2019]. URL: https://towardsdatascience.com/most-effective-way-to-implement-radial-basis-function-neural-network-for-classification-problem-33c467803319.

[54] H. Kandan. *Understanding the kernel trick.* [referenced 25.2.2019]. URL: https://towardsdatascience.com/understanding-the-kernel-trick-e0bc6112ef78.

[55] C. Shorten. *Unsupervised Feature Learning.* [referenced 25.2.2019]. URL: https://towardsdatascience.com/unsupervised-feature-learning-46a2fe399929.

[56] *What Are Feature Selection Techniques In Machine Learning?* [referenced 5.3.2019]. URL: https://www.analyticsindiamag.com/what-are-feature-selection-techniques-in-machine-learning/.

[57] R. Shaikh. *Feature Selection Techniques in Machine Learning with Python.* [referenced 5.3.2019]. URL: https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e.

[58] J. Heaton. *Programming neural networks with ENCOG 2 in Java.* Heaton Research inc, 2010, pp. 27–46.

[59] A. Chandra. *Perceptron Learning Algorithm: A Graphical Explanation Of Why It Works.* [referenced 15.2.2019]. URL: https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975.

[60] P. Jain. *Complete Guide of Activation Functions.* [referenced 3.3.2019]. URL: https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044.

[61] J. Heaton. *Programming neural networks with ENCOG 2 in Java.* Heaton Research inc, 2010, pp. 85–100.

[62] S. Sharma. *What the Hell is Perceptron?* [referenced 5.3.2019]. URL: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53.

[63] S. Saha. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way.* [referenced 14.3.2019]. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[64] A. Karpathy et al. *Generative models.* [referenced 23.3.2019]. URL: https://blog.openai.com/generative-models/.

[65] C. Nicholson. *A Beginner's Guide to Generative Adversarial Networks (GANs).* [referenced 23.3.2019]. URL: https://skymind.ai/wiki/generative-adversarial-network-gan.

[66] J. Heaton. *Programming neural networks with ENCOG 2 in Java.* Heaton Research inc, 2010, pp. 119–146.

[67] M. Nielsen. *How the backpropagation algorithm works.* [referenced 21.4.2019]. URL: http://neuralnetworksanddeeplearning.com/chap2.html.

[68] *What is batch size in neural network?* [referenced 5.4.2019]. URL: https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network.

[69] S. Sharma. *Epoch vs Batch Size vs Iterations.* [referenced 5.4.2019]. URL: https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9.

[70] H. Zulkifli. *Understanding Learning Rates and How It Improves Performance in Deep Learning.* [referenced 5.4.2019]. URL: https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10.

[71] *How to choose the number of hidden layers and nodes in a feedforward neural network?* [referenced 15.4.2019]. URL: https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw.

[72] A. Gad. *Beginners Ask "How Many Hidden Layers/Neurons to Use in Artificial Neural Networks?"*. [referenced 15.4.2019]. URL: https://towardsdatascience.com/beginners-ask-how-many-hidden-layers-neurons-to-use-in-artificial-neural-networks-51466afa0d3e.

[73] J. Heaton. *Introduction to Neural Networks in Java*. Heaton Research inc, 2005.

[74] J. Brownlee. *How to Configure the Number of Layers and Nodes in a Neural Network*. [referenced 15.4.2019]. URL: https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/.

[75] R. Singh. *Pruning Deep Neural Networks*. [referenced 5.4.2019]. URL: https://towardsdatascience.com/pruning-deep-neural-network-56cae1ec5505.

[76] N. Schlüter. *Don't Overfit! — How to prevent Overfitting in your Deep Learning Models*. [referenced 12.4.2019]. URL: https://towardsdatascience.com/dont-overfit-how-to-prevent-overfitting-in-your-deep-learning-models-63274e552323.

[77] Neurosolutions. *Neurosolutions*. [referenced 22.4.2019]. URL: http://www.neurosolutions.com/neurosolutions/.

[78] *Encog Machine Learning Framework*. [referenced 22.4.2019]. URL: https://www.heatonresearch.com/encog/.

[79] *Deep Learning for Java*. [referenced 22.4.2019]. URL: https://deeplearning4j.org/.

[80] *Deep Learning Toolbox*. [referenced 22.4.2019]. URL: https://se.mathworks.com/products/deep-learning.html.

[81] Karlijn Willems. *TensorFlow Tutorial For Beginners*. [referenced 22.4.2019]. URL: https://www.datacamp.com/community/tutorials/tensorflow-tutorial.

[82] *Keras*. [referenced 22.4.2019]. URL: https://keras.io/.

[83] CERN. *Geant4: A simulation toolkit*. [referenced 22.4.2019]. URL: https://geant4.web.cern.ch/.

[84] CERN. *Physics processes*. [referenced 22.4.2019]. URL: http://geant4-userdoc.web.cern.ch/geant4-%20userdoc/UsersGuides/ForApplicationDeveloper/html/TrackingAndPhysics/physicsProcess.html.

[85] A. Vogt et al. "Jovian electrons in the inner heliosphere". In: *Astronomy and Astrophysics* 613.A28 (2018). DOI: https://doi.org/10.1051/0004-6361/201731736.

[86] R. Strauss et al. "On the propagation times and energy losses of cosmic rays in the heliosphere". In: *Journal of Geophysical Research Atmospheres* 116.A12 (2011), pp. 12105–. DOI: https://doi.org/10.1029/2011JA016831.

[87] NVIDIA. *TensorRT*. [referenced 18.4.2020]. URL: https://developer.nvidia.com/tensorrt.

[88] S. Aggarwal. *3D-MNIST Image Classification*. [referenced 5.5.2020]. URL: https://medium.com/shashwats-blog/3d-mnist-b922a3d07334.

[89] *A simple Conv3D example with Keras*. [referenced 5.5.2020]. URL: https://www.machinecurve.com/index.php/2019/10/18/a-simple-conv3d-example-with-keras/.

[90] *Save and load models*. [referenced 5.5.2020]. URL: https://www.tensorflow.org/tutorials/keras/save_and_load.

[91] *Load a pandas.DataFrame*. [referenced 5.5.2020]. URL: https://www.tensorflow.org/tutorials/load_data/pandas_dataframe.

[92] *Basic classification: Classify images of clothing*. [referenced 5.5.2020]. URL: https://www.tensorflow.org/tutorials/keras/classification.

[93] *TensorFlow Examples*. [referenced 5.5.2020]. URL: https://github.com/aymericdamien/TensorFlow-Examples.

A Appendices

A.1 Using Geant4 command line interface

Graphical user interface of Geant4 is implemented using Linux library, and it cannot therefore fully be utilized under Windows. Mainly, the mouse user interface doesn't work, but one has to use keyboard commands into the Geant4 console to perform e.g. scaling and rotating operations along with simulation-starting commands. When the program starts, the console window is opened along with another window showing the visualization of the detector.

With the console, the view can be zoomed and rotated appropriately by e.g. the following commands:

```
/vis/viewer/zoom
/vis/drawView 66 22
```

Next the particle event setup is configured. The particle type and energy of each particle can be configured with the commands

```
/gun/particle e-
/gun/energy %d MeV
```

And finally the simulation can be started by specifying the amount of events for the beamOn command:

```
/run/beamOn 100
```

When producing a bigger number of events directly to be read in the machine learning solution, the command line interface is more suitable. The particle type however has to be configured into the program source code and compiled first. After that the simulation can be ran with the following command, the first parameter being the number of events, the second one the energy of a single event in MeV and finally the name of the output file:

```
calorimeter.exe 1111 111 > out.txt
```

A.2 Test outputs for CNN

A.2.1 <50 MeV electrons vs <2 GeV protons 6000 values

```
Using TensorFlow backend.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_max_50mev.txt with 6000 events.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_max_2gev.txt with 6000 events.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv3d_1 (Conv3D)            (None, 14, 14, 14, 32)    2624
_________________________________________________________________
max_pooling3d_1 (MaxPooling3 (None, 7, 7, 7, 32)       0
_________________________________________________________________
conv3d_2 (Conv3D)            (None, 5, 5, 5, 64)       55360
_________________________________________________________________
max_pooling3d_2 (MaxPooling3 (None, 2, 2, 2, 64)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 514
=================================================================
Total params: 189,826
Trainable params: 189,826
Non-trainable params: 0
_________________________________________________________________
None
Train on 6000 samples, validate on 3000 samples
Epoch 1/33
```

```
2020-01-30 21:33:17.324107: I T:\src\github\tensorflow\tensorflow\core\
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
    this TensorFlow binary was not compiled to use: AVX2
2020-01-30 21:33:31.884370: W T:\src\github\tensorflow\tensorflow\core\
    framework\allocator.cc:108] Allocation of 279936000 exceeds 10% of
    system memory.
 375/6000 [>.............................] - ETA: 7:33 - loss: 0.6920 - acc
     : 0.55472020-01-30 21:34:01.663423: W T:\src\github\tensorflow\
     tensorflow\core\framework\allocator.cc:108] Allocation of 279936000
     exceeds 10% of system memory.
 750/6000 [==>...........................] - ETA: 6:59 - loss: 0.7053 - acc
     : 0.52672020-01-30 21:34:31.362571: W T:\src\github\tensorflow\
     tensorflow\core\framework\allocator.cc:108] Allocation of 279936000
     exceeds 10% of system memory.
1125/6000 [====>.........................] - ETA: 6:29 - loss: 0.6872 - acc
    : 0.59472020-01-30 21:35:01.178804: W T:\src\github\tensorflow\
    tensorflow\core\framework\allocator.cc:108] Allocation of 279936000
    exceeds 10% of system memory.
1500/6000 [======>.......................] - ETA: 5:58 - loss: 0.6789 - acc
    : 0.61272020-01-30 21:35:30.879845: W T:\src\github\tensorflow\
    tensorflow\core\framework\allocator.cc:108] Allocation of 279936000
    exceeds 10% of system memory.
6000/6000 [==============================] - 508s 85ms/step - loss: 0.5946
    - acc: 0.7395 - val_loss: 0.7352 - val_acc: 0.5003
Epoch 2/33
6000/6000 [==============================] - 565s 94ms/step - loss: 0.6282
    - acc: 0.7538 - val_loss: 0.7371 - val_acc: 0.5003
Epoch 3/33
6000/6000 [==============================] - 550s 92ms/step - loss: 0.5257
    - acc: 0.7802 - val_loss: 0.3737 - val_acc: 0.8560
Epoch 4/33
6000/6000 [==============================] - 549s 91ms/step - loss: 0.7160
    - acc: 0.7297 - val_loss: 0.5308 - val_acc: 0.5013
Epoch 5/33
6000/6000 [==============================] - 535s 89ms/step - loss: 0.6073
```

```
    - acc: 0.7028 - val_loss: 0.5287 - val_acc: 0.8143
Epoch 00005: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png

# Evaluate on test data
3000/3000 [==============================] - 38s 13ms/step
test loss, test acc: [0.5551064610481262, 0.8046666383743286]
```

A.3 Test outputs for FNN

A.3.1 Monoenergetic 50 MeV protons vs monoenergetic 50 MeV electrons

```
Using TensorFlow backend.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_mono_50mev.txt with 6000 events.
Saved e_mono_50mev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_mono_50mev.txt with 6000 events.
Saved p_mono_50mev.txt.png
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 8200
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 72
```

```
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 18
=================================================================
Total params: 1,057,890
Trainable params: 1,057,890
Non-trainable params: 0
_________________________________________________________________
None
Train on 6000 samples, validate on 3000 samples
Epoch 1/33
2020-01-30 21:14:16.500527: I T:\src\github\tensorflow\tensorflow\core\
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
    this TensorFlow binary was not compiled to use: AVX2
6000/6000 [==============================] - 1s 224us/step - loss: 0.6228 -
     acc: 0.6361 - val_loss: 0.5522 - val_acc: 0.7197
Epoch 2/33
6000/6000 [==============================] - 1s 146us/step - loss: 0.4947 -
     acc: 0.7299 - val_loss: 0.4457 - val_acc: 0.7288
Epoch 3/33
6000/6000 [==============================] - 1s 146us/step - loss: 0.4033 -
     acc: 0.7385 - val_loss: 0.3771 - val_acc: 0.7387
Epoch 4/33
6000/6000 [==============================] - 1s 146us/step - loss: 0.3468 -
     acc: 0.7435 - val_loss: 0.3335 - val_acc: 0.7437
Epoch 5/33
6000/6000 [==============================] - 1s 146us/step - loss: 0.3075 -
     acc: 0.7463 - val_loss: 0.3011 - val_acc: 0.7457
Epoch 6/33
6000/6000 [==============================] - 1s 146us/step - loss: 0.2752 -
     acc: 0.7480 - val_loss: 0.2718 - val_acc: 0.7465
Epoch 7/33
6000/6000 [==============================] - 1s 147us/step - loss: 0.2433 -
     acc: 0.7493 - val_loss: 0.2408 - val_acc: 0.7467
Epoch 8/33
6000/6000 [==============================] - 1s 146us/step - loss: 0.2111 -
```

```
    acc: 0.7867 - val_loss: 0.2138 - val_acc: 0.9748
Epoch 9/33
6000/6000 [==============================] - 1s 146us/step - loss: 0.1800 -
    acc: 0.9943 - val_loss: 0.1828 - val_acc: 0.9887
Epoch 10/33
6000/6000 [==============================] - 1s 146us/step - loss: 0.1450 -
    acc: 0.9985 - val_loss: 0.1486 - val_acc: 0.9910
Epoch 11/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.0983 -
    acc: 0.9992 - val_loss: 0.0988 - val_acc: 0.9930
Epoch 12/33
6000/6000 [==============================] - 1s 146us/step - loss: 0.0568 -
    acc: 0.9995 - val_loss: 0.0688 - val_acc: 0.9928
Epoch 13/33
6000/6000 [==============================] - 1s 146us/step - loss: 0.0298 -
    acc: 0.9997 - val_loss: 0.0474 - val_acc: 0.9940
Epoch 14/33
6000/6000 [==============================] - 1s 146us/step - loss: 0.0149 -
    acc: 0.9997 - val_loss: 0.0429 - val_acc: 0.9930
Epoch 15/33
6000/6000 [==============================] - 1s 146us/step - loss: 0.0077 -
    acc: 0.9998 - val_loss: 0.0340 - val_acc: 0.9942
Epoch 16/33
6000/6000 [==============================] - 1s 146us/step - loss: 0.0042 -
    acc: 0.9998 - val_loss: 0.0317 - val_acc: 0.9948
Epoch 17/33
6000/6000 [==============================] - 1s 147us/step - loss: 0.0027 -
    acc: 0.9997 - val_loss: 0.0309 - val_acc: 0.9947
Epoch 00017: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png

# Evaluate on test data
3000/3000 [==============================] - 0s 39us/step
test loss, test acc: [0.03344448283314705, 0.9913333058357239]
```

A.3.2 Monoenergetic 50 MeV protons vs monoenergetic 5 MeV electrons

```
Using TensorFlow backend.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_mono_5mev.txt with 6000 events.
Saved e_mono_5mev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_mono_50mev.txt with 6000 events.
Saved p_mono_50mev.txt.png
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 8200
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 18
=================================================================
Total params: 1,057,890
Trainable params: 1,057,890
Non-trainable params: 0
_________________________________________________________________
None
Train on 6000 samples, validate on 3000 samples
Epoch 1/33
2020-01-30 21:16:01.346428: I T:\src\github\tensorflow\tensorflow\core\
```

```
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
    this TensorFlow binary was not compiled to use: AVX2
6000/6000 [==============================] - 1s 230us/step - loss: 0.6255 -
     acc: 0.6133 - val_loss: 0.5657 - val_acc: 0.7007
Epoch 2/33
6000/6000 [==============================] - 1s 177us/step - loss: 0.5202 -
     acc: 0.7927 - val_loss: 0.4901 - val_acc: 0.8417
Epoch 3/33
6000/6000 [==============================] - 1s 189us/step - loss: 0.4507 -
     acc: 0.8683 - val_loss: 0.4428 - val_acc: 0.8570
Epoch 4/33
6000/6000 [==============================] - 1s 197us/step - loss: 0.4038 -
     acc: 0.9089 - val_loss: 0.4110 - val_acc: 0.9033
Epoch 5/33
6000/6000 [==============================] - 1s 187us/step - loss: 0.3708 -
     acc: 0.9378 - val_loss: 0.3900 - val_acc: 0.9278
Epoch 6/33
6000/6000 [==============================] - 1s 208us/step - loss: 0.3459 -
     acc: 0.9536 - val_loss: 0.3771 - val_acc: 0.9368
Epoch 7/33
6000/6000 [==============================] - 1s 195us/step - loss: 0.3264 -
     acc: 0.9677 - val_loss: 0.3665 - val_acc: 0.9492
Epoch 8/33
6000/6000 [==============================] - 1s 190us/step - loss: 0.3104 -
     acc: 0.9809 - val_loss: 0.3631 - val_acc: 0.9553
Epoch 9/33
6000/6000 [==============================] - 1s 172us/step - loss: 0.2966 -
     acc: 0.9898 - val_loss: 0.3588 - val_acc: 0.9605
Epoch 00009: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png

# Evaluate on test data
3000/3000 [==============================] - 0s 62us/step
test loss, test acc: [0.3550744950771332, 0.9601666927337646]
```

A.3.3 Flat <50 MeV electrons vs flat <2 GeV protons 6000 values

```
Using TensorFlow backend.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_max_50mev.txt with 6000 events.
Saved e_max_50mev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_max_2gev.txt with 6000 events.
Saved p_max_2gev.txt.png
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 8200
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 18
=================================================================
Total params: 1,057,890
Trainable params: 1,057,890
Non-trainable params: 0
_________________________________________________________________
None
Train on 6000 samples, validate on 3000 samples
Epoch 1/33
2020-01-30 21:08:11.430954: I T:\src\github\tensorflow\tensorflow\core\
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
```

```
   this TensorFlow binary was not compiled to use: AVX2
6000/6000 [==============================] - 1s 226us/step - loss: 0.6776 -
    acc: 0.5910 - val_loss: 0.6583 - val_acc: 0.6533
Epoch 2/33
6000/6000 [==============================] - 1s 147us/step - loss: 0.6279 -
    acc: 0.7589 - val_loss: 0.6152 - val_acc: 0.7030
Epoch 3/33
6000/6000 [==============================] - 1s 153us/step - loss: 0.5579 -
    acc: 0.8327 - val_loss: 0.5577 - val_acc: 0.7975
Epoch 4/33
6000/6000 [==============================] - 1s 187us/step - loss: 0.4819 -
    acc: 0.8868 - val_loss: 0.5025 - val_acc: 0.8615
Epoch 5/33
6000/6000 [==============================] - 1s 164us/step - loss: 0.4057 -
    acc: 0.9227 - val_loss: 0.4472 - val_acc: 0.9005
Epoch 6/33
6000/6000 [==============================] - 1s 184us/step - loss: 0.3308 -
    acc: 0.9456 - val_loss: 0.3977 - val_acc: 0.9083
Epoch 7/33
6000/6000 [==============================] - 1s 164us/step - loss: 0.2621 -
    acc: 0.9565 - val_loss: 0.3558 - val_acc: 0.9243
Epoch 8/33
6000/6000 [==============================] - 1s 161us/step - loss: 0.2039 -
    acc: 0.9657 - val_loss: 0.3269 - val_acc: 0.9298
Epoch 9/33
6000/6000 [==============================] - 1s 200us/step - loss: 0.1551 -
    acc: 0.9735 - val_loss: 0.3061 - val_acc: 0.9403
Epoch 10/33
6000/6000 [==============================] - 1s 216us/step - loss: 0.1154 -
    acc: 0.9788 - val_loss: 0.3038 - val_acc: 0.9423
Epoch 11/33
6000/6000 [==============================] - 1s 163us/step - loss: 0.0872 -
    acc: 0.9839 - val_loss: 0.2952 - val_acc: 0.9430
Epoch 12/33
6000/6000 [==============================] - 1s 158us/step - loss: 0.0653 -
```

```
    acc: 0.9880 - val_loss: 0.3016 - val_acc: 0.9432
Epoch 13/33
6000/6000 [==============================] - 1s 156us/step - loss: 0.0505 -
    acc: 0.9898 - val_loss: 0.3100 - val_acc: 0.9473
Epoch 00013: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png

# Evaluate on test data
3000/3000 [==============================] - 0s 40us/step
test loss, test acc: [0.3053668737411499, 0.9468333125114441]
```

A.3.4 Flat <50 MeV electrons vs flat <2 GeV protons 140000 values

```
Using TensorFlow backend.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
   fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
    25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
    3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_max_50mev_150000.txt with 136490 events.
Saved e_max_50mev_150000.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
   fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
    25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
    3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_max_2gev_150000.txt with 140000 events.
Saved p_max_2gev_150000.txt.png
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 8200
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 72
```

```
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 18
=================================================================
Total params: 1,057,890
Trainable params: 1,057,890
Non-trainable params: 0
_________________________________________________________________
None
Train on 138245 samples, validate on 69122 samples
Epoch 1/33
2020-01-30 21:23:30.716285: I T:\src\github\tensorflow\tensorflow\core\
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
    this TensorFlow binary was not compiled to use: AVX2
138245/138245 [==============================] - 15s 108us/step - loss:
    0.6825 - acc: 0.6700 - val_loss: 0.6711 - val_acc: 0.4873
Epoch 2/33
138245/138245 [==============================] - 14s 103us/step - loss:
    0.6493 - acc: 0.7270 - val_loss: 0.6397 - val_acc: 0.4873
Epoch 3/33
138245/138245 [==============================] - 14s 103us/step - loss:
    0.5990 - acc: 0.8597 - val_loss: 0.5812 - val_acc: 0.8390
Epoch 4/33
138245/138245 [==============================] - 14s 104us/step - loss:
    0.5417 - acc: 0.9138 - val_loss: 0.5344 - val_acc: 0.9219
Epoch 5/33
138245/138245 [==============================] - 14s 103us/step - loss:
    0.4900 - acc: 0.9232 - val_loss: 0.4802 - val_acc: 0.9164
Epoch 6/33
138245/138245 [==============================] - 14s 104us/step - loss:
    0.4404 - acc: 0.9277 - val_loss: 0.4713 - val_acc: 0.8944
Epoch 7/33
138245/138245 [==============================] - 15s 105us/step - loss:
    0.3981 - acc: 0.9300 - val_loss: 0.4022 - val_acc: 0.9205
Epoch 8/33
138245/138245 [==============================] - 15s 107us/step - loss:
```

```
    0.3562 - acc: 0.9361 - val_loss: 0.3679 - val_acc: 0.9282
Epoch 9/33
138245/138245 [==============================] - 15s 107us/step - loss:
    0.3127 - acc: 0.9417 - val_loss: 0.3785 - val_acc: 0.9109
Epoch 10/33
138245/138245 [==============================] - 15s 107us/step - loss:
    0.2769 - acc: 0.9442 - val_loss: 0.2991 - val_acc: 0.9340
Epoch 11/33
138245/138245 [==============================] - 15s 106us/step - loss:
    0.2400 - acc: 0.9514 - val_loss: 0.2989 - val_acc: 0.9265
Epoch 12/33
138245/138245 [==============================] - 15s 105us/step - loss:
    0.2125 - acc: 0.9555 - val_loss: 0.2913 - val_acc: 0.9274
Epoch 00012: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png

# Evaluate on test data
2020-01-30 21:26:25.765787: W T:\src\github\tensorflow\tensorflow\core\
    framework\allocator.cc:108] Allocation of 283127808 exceeds 10% of
    system memory.
69123/69123 [==============================] - 3s 39us/step
test loss, test acc: [0.2091025710105896, 0.9525628089904785]
```

A.3.5 Flat <50 MeV electrons vs flat 30 - 2000 MeV protons

```
Using TensorFlow backend.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_max_50mev.txt with 6000 events.
Saved e_max_50mev.txt.png
Saved e_max_50mev.txt var.png
Saved e_max_50mev.txt cov sums.png
```

```
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_30_to_2gev.txt with 6000 events.
Saved p_30_to_2gev.txt.png
Saved p_30_to_2gev.txt var.png
Saved p_30_to_2gev.txt cov sums.png
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 8200
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 18
=================================================================
Total params: 1,057,890
Trainable params: 1,057,890
Non-trainable params: 0
_________________________________________________________________
None
Train on 6000 samples, validate on 3000 samples
Epoch 1/33
2020-02-01 16:50:00.337517: I T:\src\github\tensorflow\tensorflow\core\
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
    this TensorFlow binary was not compiled to use: AVX2
6000/6000 [==============================] - 2s 286us/step - loss: 0.6797 -
     acc: 0.6568 - val_loss: 0.6598 - val_acc: 0.7828
Epoch 2/33
6000/6000 [==============================] - 1s 185us/step - loss: 0.6320 -
     acc: 0.8578 - val_loss: 0.6091 - val_acc: 0.8758
Epoch 3/33
```

```
6000/6000 [==============================] - 1s 179us/step - loss: 0.5644 -
    acc: 0.9032 - val_loss: 0.5469 - val_acc: 0.8945
Epoch 4/33
6000/6000 [==============================] - 1s 191us/step - loss: 0.4845 -
    acc: 0.9289 - val_loss: 0.4876 - val_acc: 0.9047
Epoch 5/33
6000/6000 [==============================] - 1s 193us/step - loss: 0.4042 -
    acc: 0.9397 - val_loss: 0.4194 - val_acc: 0.9175
Epoch 6/33
6000/6000 [==============================] - 1s 197us/step - loss: 0.3271 -
    acc: 0.9522 - val_loss: 0.3667 - val_acc: 0.9228
Epoch 7/33
6000/6000 [==============================] - 1s 190us/step - loss: 0.2608 -
    acc: 0.9578 - val_loss: 0.3271 - val_acc: 0.9287
Epoch 8/33
6000/6000 [==============================] - 1s 192us/step - loss: 0.2041 -
    acc: 0.9661 - val_loss: 0.3039 - val_acc: 0.9333
Epoch 9/33
6000/6000 [==============================] - 1s 212us/step - loss: 0.1656 -
    acc: 0.9685 - val_loss: 0.2872 - val_acc: 0.9365
Epoch 10/33
6000/6000 [==============================] - 1s 207us/step - loss: 0.1296 -
    acc: 0.9772 - val_loss: 0.2881 - val_acc: 0.9365
Epoch 11/33
6000/6000 [==============================] - 1s 217us/step - loss: 0.1068 -
    acc: 0.9782 - val_loss: 0.2847 - val_acc: 0.9400
Epoch 00011: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png

# Evaluate on test data
3000/3000 [==============================] - 0s 58us/step
test loss, test acc: [0.2698492705821991, 0.9401666522026062]
```

A.3.6 Flat <50 MeV electrons vs flat 30 - 2000 MeV protons binary values

```
C:\Users\User\Desktop\gradu\tensorflow>py 20.py
Using TensorFlow backend.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_max_50mev.txt with 6000 events.
Saved e_max_50mev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_30_to_2gev.txt with 6000 events.
Saved p_30_to_2gev.txt.png
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 8200
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 18
=================================================================
Total params: 1,057,890
Trainable params: 1,057,890
Non-trainable params: 0
_________________________________________________________________
None
Train on 6000 samples, validate on 3000 samples
Epoch 1/33
```

```
2020-02-02 12:21:08.699899: I T:\src\github\tensorflow\tensorflow\core\
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
    this TensorFlow binary was not compiled to use: AVX2
6000/6000 [==============================] - 1s 213us/step - loss: 0.5992 -
     acc: 0.6448 - val_loss: 0.5054 - val_acc: 0.7690
Epoch 2/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.4280 -
     acc: 0.8918 - val_loss: 0.3966 - val_acc: 0.9098
Epoch 3/33
6000/6000 [==============================] - 1s 146us/step - loss: 0.3310 -
     acc: 0.9412 - val_loss: 0.3278 - val_acc: 0.9288
Epoch 4/33
6000/6000 [==============================] - 1s 149us/step - loss: 0.2546 -
     acc: 0.9619 - val_loss: 0.2823 - val_acc: 0.9278
Epoch 5/33
6000/6000 [==============================] - 1s 146us/step - loss: 0.1988 -
     acc: 0.9692 - val_loss: 0.2677 - val_acc: 0.9275
Epoch 6/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.1637 -
     acc: 0.9726 - val_loss: 0.2393 - val_acc: 0.9388
Epoch 7/33
6000/6000 [==============================] - 1s 146us/step - loss: 0.1388 -
     acc: 0.9755 - val_loss: 0.2484 - val_acc: 0.9357
Epoch 8/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.1197 -
     acc: 0.9769 - val_loss: 0.2529 - val_acc: 0.9358
Epoch 00008: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png

# Evaluate on test data
3000/3000 [==============================] - 0s 38us/step
test loss, test acc: [0.25404590368270874, 0.9350000023841858]
```

A.3.7 <50 MeV electrons vs 30 - 2000 MeV protons, both realistic values

```
C:\Users\User\Desktop\gradu\tensorflow>py 20.py
Using TensorFlow backend.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_max_50mev_realistic2.txt with 6000 events.
Saved e_max_50mev_realistic2.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_30_to_2000mev_realistic.txt with 6000 events.
Saved p_30_to_2000mev_realistic.txt.png
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 8200
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 18
=================================================================
Total params: 1,057,890
Trainable params: 1,057,890
Non-trainable params: 0
_________________________________________________________________
None
Train on 6000 samples, validate on 3000 samples
Epoch 1/33
2020-05-11 17:20:51.565500: I T:\src\github\tensorflow\tensorflow\core\
```

```
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
    this TensorFlow binary was not compiled to use: AVX2
6000/6000 [==============================] - 1s 238us/step - loss: 0.6871 -
     acc: 0.6326 - val_loss: 0.6779 - val_acc: 0.7853
Epoch 2/33
6000/6000 [==============================] - 1s 147us/step - loss: 0.6596 -
     acc: 0.8176 - val_loss: 0.6460 - val_acc: 0.8315
Epoch 3/33
6000/6000 [==============================] - 1s 148us/step - loss: 0.6082 -
     acc: 0.8789 - val_loss: 0.5974 - val_acc: 0.8430
Epoch 4/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.5374 -
     acc: 0.9038 - val_loss: 0.5430 - val_acc: 0.8423
Epoch 5/33
6000/6000 [==============================] - 1s 150us/step - loss: 0.4586 -
     acc: 0.9244 - val_loss: 0.4882 - val_acc: 0.8637
Epoch 6/33
6000/6000 [==============================] - 1s 150us/step - loss: 0.3828 -
     acc: 0.9369 - val_loss: 0.4380 - val_acc: 0.8985
Epoch 7/33
6000/6000 [==============================] - 1s 148us/step - loss: 0.3144 -
     acc: 0.9448 - val_loss: 0.3998 - val_acc: 0.9060
Epoch 8/33
6000/6000 [==============================] - 1s 148us/step - loss: 0.2589 -
     acc: 0.9519 - val_loss: 0.3806 - val_acc: 0.9108
Epoch 9/33
6000/6000 [==============================] - 1s 150us/step - loss: 0.2116 -
     acc: 0.9593 - val_loss: 0.3600 - val_acc: 0.9143
Epoch 10/33
6000/6000 [==============================] - 1s 149us/step - loss: 0.1741 -
     acc: 0.9642 - val_loss: 0.3628 - val_acc: 0.9182
Epoch 11/33
6000/6000 [==============================] - 1s 148us/step - loss: 0.1457 -
     acc: 0.9675 - val_loss: 0.3464 - val_acc: 0.9222
Epoch 12/33
```

```
6000/6000 [==============================] - 1s 145us/step - loss: 0.1218 -
    acc: 0.9727 - val_loss: 0.3537 - val_acc: 0.9240
Epoch 13/33
6000/6000 [==============================] - 1s 148us/step - loss: 0.0992 -
    acc: 0.9776 - val_loss: 0.3490 - val_acc: 0.9242
Epoch 00013: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png

# Evaluate on test data
3000/3000 [==============================] - 0s 39us/step
test loss, test acc: [0.2815898060798645, 0.937833309173584]
```

A.3.8 Monoenergetic 50 MeV protons vs monoenergetic 5 MeV electrons 11° zenith angle

```
C:\Users\User\Desktop\gradu\tensorflow>py 20.py
Using TensorFlow backend.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
   fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
    25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
    3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_50_11deg.txt with 6000 events.
Saved p_50_11deg.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
   fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
    25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
    3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_5_11deg.txt with 6000 events.
Saved e_5_11deg.txt.png
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
```

```
dense_2 (Dense)           (None, 8)              8200
_________________________________________________________________
dense_3 (Dense)           (None, 8)              72
_________________________________________________________________
dense_4 (Dense)           (None, 2)              18
=================================================================
Total params: 1,057,890
Trainable params: 1,057,890
Non-trainable params: 0
_________________________________________________________________
None
Train on 6000 samples, validate on 3000 samples
Epoch 1/33
2020-04-25 13:07:53.016112: I T:\src\github\tensorflow\tensorflow\core\
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
    this TensorFlow binary was not compiled to use: AVX2
6000/6000 [==============================] - 1s 220us/step - loss: 0.6448 -
     acc: 0.6131 - val_loss: 0.5880 - val_acc: 0.6263
Epoch 2/33
6000/6000 [==============================] - 1s 147us/step - loss: 0.5406 -
     acc: 0.6982 - val_loss: 0.4951 - val_acc: 0.7158
Epoch 3/33
6000/6000 [==============================] - 1s 149us/step - loss: 0.4602 -
     acc: 0.7203 - val_loss: 0.4318 - val_acc: 0.7250
Epoch 4/33
6000/6000 [==============================] - 1s 149us/step - loss: 0.4076 -
     acc: 0.7305 - val_loss: 0.3937 - val_acc: 0.7350
Epoch 5/33
6000/6000 [==============================] - 1s 150us/step - loss: 0.3734 -
     acc: 0.7355 - val_loss: 0.3688 - val_acc: 0.7362
Epoch 6/33
6000/6000 [==============================] - 1s 150us/step - loss: 0.3507 -
     acc: 0.7389 - val_loss: 0.3537 - val_acc: 0.7365
Epoch 7/33
6000/6000 [==============================] - 1s 150us/step - loss: 0.3328 -
```

```
    acc: 0.7420 - val_loss: 0.3429 - val_acc: 0.7373
Epoch 8/33
6000/6000 [==============================] - 1s 150us/step - loss: 0.3178 -
    acc: 0.7444 - val_loss: 0.3360 - val_acc: 0.7373
Epoch 9/33
6000/6000 [==============================] - 1s 149us/step - loss: 0.3030 -
    acc: 0.7454 - val_loss: 0.3144 - val_acc: 0.7377
Epoch 10/33
6000/6000 [==============================] - 1s 148us/step - loss: 0.2653 -
    acc: 0.8020 - val_loss: 0.2704 - val_acc: 0.9735
Epoch 11/33
6000/6000 [==============================] - 1s 150us/step - loss: 0.2139 -
    acc: 0.9893 - val_loss: 0.2244 - val_acc: 0.9728
Epoch 12/33
6000/6000 [==============================] - 1s 152us/step - loss: 0.1603 -
    acc: 0.9911 - val_loss: 0.1706 - val_acc: 0.9745
Epoch 13/33
6000/6000 [==============================] - 1s 156us/step - loss: 0.1103 -
    acc: 0.9927 - val_loss: 0.1219 - val_acc: 0.9812
Epoch 14/33
6000/6000 [==============================] - 1s 155us/step - loss: 0.0752 -
    acc: 0.9933 - val_loss: 0.1003 - val_acc: 0.9807
Epoch 15/33
6000/6000 [==============================] - 1s 156us/step - loss: 0.0520 -
    acc: 0.9942 - val_loss: 0.0798 - val_acc: 0.9833
Epoch 16/33
6000/6000 [==============================] - 1s 156us/step - loss: 0.0399 -
    acc: 0.9943 - val_loss: 0.0769 - val_acc: 0.9828
Epoch 17/33
6000/6000 [==============================] - 1s 153us/step - loss: 0.0310 -
    acc: 0.9960 - val_loss: 0.0703 - val_acc: 0.9830
Epoch 00017: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png
```

```
# Evaluate on test data
3000/3000 [==============================] - 0s 40us/step
test loss, test acc: [0.10350437462329865, 0.981333315372467]
```

A.3.9 Monoenergetic 50 MeV protons vs monoenergetic 5 MeV electrons 26° zenith angle

```
C:\Users\User\Desktop\gradu\tensorflow>py 20.py
Using TensorFlow backend.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_50_26deg.txt with 6000 events.
Saved p_50_26deg.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_5_26deg.txt with 6000 events.
Saved e_5_26deg.txt.png
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 8200
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 18
=================================================================
Total params: 1,057,890
Trainable params: 1,057,890
Non-trainable params: 0
```

```
-----------------------------------------------------------------
None
Train on 6000 samples, validate on 3000 samples
Epoch 1/33
2020-04-25 13:06:20.381769: I T:\src\github\tensorflow\tensorflow\core\
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
    this TensorFlow binary was not compiled to use: AVX2
6000/6000 [==============================] - 1s 226us/step - loss: 0.6846 -
     acc: 0.5476 - val_loss: 0.6678 - val_acc: 0.4998
Epoch 2/33
6000/6000 [==============================] - 1s 148us/step - loss: 0.6380 -
     acc: 0.5141 - val_loss: 0.6032 - val_acc: 0.5388
Epoch 3/33
6000/6000 [==============================] - 1s 153us/step - loss: 0.5688 -
     acc: 0.7410 - val_loss: 0.5307 - val_acc: 0.8720
Epoch 4/33
6000/6000 [==============================] - 1s 151us/step - loss: 0.4944 -
     acc: 0.9050 - val_loss: 0.4548 - val_acc: 0.8980
Epoch 5/33
6000/6000 [==============================] - 1s 150us/step - loss: 0.4189 -
     acc: 0.9324 - val_loss: 0.3814 - val_acc: 0.9490
Epoch 6/33
6000/6000 [==============================] - 1s 153us/step - loss: 0.3491 -
     acc: 0.9567 - val_loss: 0.3161 - val_acc: 0.9708
Epoch 7/33
6000/6000 [==============================] - 1s 150us/step - loss: 0.2886 -
     acc: 0.9733 - val_loss: 0.2599 - val_acc: 0.9753
Epoch 8/33
6000/6000 [==============================] - 1s 151us/step - loss: 0.2377 -
     acc: 0.9773 - val_loss: 0.2158 - val_acc: 0.9863
Epoch 9/33
6000/6000 [==============================] - 1s 152us/step - loss: 0.1967 -
     acc: 0.9825 - val_loss: 0.1780 - val_acc: 0.9833
Epoch 10/33
6000/6000 [==============================] - 1s 152us/step - loss: 0.1625 -
```

```
    acc: 0.9847 - val_loss: 0.1494 - val_acc: 0.9810
Epoch 11/33
6000/6000 [==============================] - 1s 150us/step - loss: 0.1352 -
    acc: 0.9858 - val_loss: 0.1225 - val_acc: 0.9877
Epoch 12/33
6000/6000 [==============================] - 1s 151us/step - loss: 0.1117 -
    acc: 0.9875 - val_loss: 0.1030 - val_acc: 0.9888
Epoch 13/33
6000/6000 [==============================] - 1s 150us/step - loss: 0.0920 -
    acc: 0.9880 - val_loss: 0.0862 - val_acc: 0.9878
Epoch 14/33
6000/6000 [==============================] - 1s 151us/step - loss: 0.0780 -
    acc: 0.9880 - val_loss: 0.0745 - val_acc: 0.9883
Epoch 15/33
6000/6000 [==============================] - 1s 153us/step - loss: 0.0663 -
    acc: 0.9889 - val_loss: 0.0674 - val_acc: 0.9895
Epoch 16/33
6000/6000 [==============================] - 1s 151us/step - loss: 0.0582 -
    acc: 0.9902 - val_loss: 0.0603 - val_acc: 0.9883
Epoch 17/33
6000/6000 [==============================] - 1s 150us/step - loss: 0.0535 -
    acc: 0.9894 - val_loss: 0.0570 - val_acc: 0.9888
Epoch 18/33
6000/6000 [==============================] - 1s 150us/step - loss: 0.0488 -
    acc: 0.9905 - val_loss: 0.0543 - val_acc: 0.9897
Epoch 00018: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png

# Evaluate on test data
3000/3000 [==============================] - 0s 39us/step
test loss, test acc: [0.06543352454900742, 0.9868333339691162]
```

A.3.10 Monoenergetic 5, 10, 25, 50 MeV electrons vs flat 30-2000 MeV protons

```
C:\Users\User\Desktop\gradu\tensorflow>py 20.py
Using TensorFlow backend.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_mono_5mev.txt with 6000 events.
Saved e_mono_5mev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_30_to_2gev.txt with 6000 events.
Saved p_30_to_2gev.txt.png
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 8200
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 18
=================================================================
Total params: 1,057,890
Trainable params: 1,057,890
Non-trainable params: 0
_________________________________________________________________
None
Train on 6000 samples, validate on 3000 samples
```

```
Epoch 1/33
2020-05-11 20:05:08.553448: I T:\src\github\tensorflow\tensorflow\core\
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
    this TensorFlow binary was not compiled to use: AVX2
6000/6000 [==============================] - 1s 212us/step - loss: 0.6891 -
     acc: 0.5655 - val_loss: 0.6799 - val_acc: 0.5552
Epoch 2/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.6596 -
     acc: 0.6508 - val_loss: 0.6407 - val_acc: 0.8428
Epoch 3/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.5983 -
     acc: 0.8326 - val_loss: 0.5740 - val_acc: 0.9107
Epoch 4/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.5096 -
     acc: 0.9340 - val_loss: 0.4886 - val_acc: 0.9307
Epoch 5/33
6000/6000 [==============================] - 1s 147us/step - loss: 0.4051 -
     acc: 0.9708 - val_loss: 0.3969 - val_acc: 0.9405
Epoch 6/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.3045 -
     acc: 0.9802 - val_loss: 0.3174 - val_acc: 0.9587
Epoch 7/33
6000/6000 [==============================] - 1s 150us/step - loss: 0.2192 -
     acc: 0.9860 - val_loss: 0.2545 - val_acc: 0.9643
Epoch 8/33
6000/6000 [==============================] - 1s 149us/step - loss: 0.1544 -
     acc: 0.9894 - val_loss: 0.2067 - val_acc: 0.9673
Epoch 9/33
6000/6000 [==============================] - 1s 148us/step - loss: 0.1032 -
     acc: 0.9912 - val_loss: 0.1801 - val_acc: 0.9685
Epoch 10/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.0693 -
     acc: 0.9937 - val_loss: 0.1619 - val_acc: 0.9728
Epoch 11/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.0460 -
```

```
     acc: 0.9948 - val_loss: 0.1529 - val_acc: 0.9753
Epoch 12/33
6000/6000 [==============================] - 1s 144us/step - loss: 0.0297 -
     acc: 0.9962 - val_loss: 0.1503 - val_acc: 0.9772
Epoch 13/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.0206 -
     acc: 0.9972 - val_loss: 0.1546 - val_acc: 0.9777
Epoch 14/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.0132 -
     acc: 0.9980 - val_loss: 0.1580 - val_acc: 0.9773
Epoch 00014: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png

# Evaluate on test data
3000/3000 [==============================] - 0s 39us/step
test loss, test acc: [0.15977025032043457, 0.9776666760444641]



C:\Users\User\Desktop\gradu\tensorflow>py 20.py
Using TensorFlow backend.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_mono_10mev.txt with 6000 events.
Saved e_mono_10mev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_30_to_2gev.txt with 6000 events.
Saved p_30_to_2gev.txt.png
-----------------------------------------------------------------
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 8200
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 18
=================================================================
Total params: 1,057,890
Trainable params: 1,057,890
Non-trainable params: 0
_________________________________________________________________
None
Train on 6000 samples, validate on 3000 samples
Epoch 1/33
2020-05-11 20:07:05.285993: I T:\src\github\tensorflow\tensorflow\core\
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
    this TensorFlow binary was not compiled to use: AVX2
6000/6000 [==============================] - 1s 209us/step - loss: 0.6774 -
     acc: 0.5522 - val_loss: 0.6540 - val_acc: 0.6023
Epoch 2/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.6263 -
     acc: 0.6562 - val_loss: 0.5988 - val_acc: 0.6420
Epoch 3/33
6000/6000 [==============================] - 1s 147us/step - loss: 0.5581 -
     acc: 0.7152 - val_loss: 0.5356 - val_acc: 0.6945
Epoch 4/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.4888 -
     acc: 0.7321 - val_loss: 0.4791 - val_acc: 0.7373
Epoch 5/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.4263 -
     acc: 0.7433 - val_loss: 0.4157 - val_acc: 0.7382
Epoch 6/33
```

```
6000/6000 [==============================] - 1s 144us/step - loss: 0.3753 -
    acc: 0.7462 - val_loss: 0.3749 - val_acc: 0.7372
Epoch 7/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.3329 -
    acc: 0.7478 - val_loss: 0.3390 - val_acc: 0.7408
Epoch 8/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.2998 -
    acc: 0.7478 - val_loss: 0.3160 - val_acc: 0.7415
Epoch 9/33
6000/6000 [==============================] - 1s 144us/step - loss: 0.2734 -
    acc: 0.7481 - val_loss: 0.2928 - val_acc: 0.7415
Epoch 10/33
6000/6000 [==============================] - 1s 148us/step - loss: 0.2507 -
    acc: 0.7484 - val_loss: 0.2777 - val_acc: 0.7418
Epoch 11/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.2328 -
    acc: 0.7489 - val_loss: 0.2661 - val_acc: 0.7422
Epoch 12/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.2172 -
    acc: 0.7491 - val_loss: 0.2609 - val_acc: 0.7412
Epoch 13/33
6000/6000 [==============================] - 1s 144us/step - loss: 0.2039 -
    acc: 0.7508 - val_loss: 0.2487 - val_acc: 0.7783
Epoch 14/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.1914 -
    acc: 0.9027 - val_loss: 0.2427 - val_acc: 0.9705
Epoch 15/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.1796 -
    acc: 0.9957 - val_loss: 0.2325 - val_acc: 0.9783
Epoch 16/33
6000/6000 [==============================] - 1s 147us/step - loss: 0.1679 -
    acc: 0.9983 - val_loss: 0.2345 - val_acc: 0.9737
Epoch 17/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.1559 -
    acc: 0.9986 - val_loss: 0.2207 - val_acc: 0.9798
```

```
Epoch 18/33
6000/6000 [==============================] - 1s 148us/step - loss: 0.1445 -
    acc: 0.9987 - val_loss: 0.2189 - val_acc: 0.9775
Epoch 19/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.1298 -
    acc: 0.9987 - val_loss: 0.2111 - val_acc: 0.9788
Epoch 00019: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png

# Evaluate on test data
3000/3000 [==============================] - 0s 39us/step
test loss, test acc: [0.24657656252384186, 0.9758333563804626]



C:\Users\User\Desktop\gradu\tensorflow>py 20.py
Using TensorFlow backend.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
   fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
    25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
    3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_mono_25mev.txt with 6000 events.
Saved e_mono_25mev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
   fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
    25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
    3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_30_to_2gev.txt with 6000 events.
Saved p_30_to_2gev.txt.png
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
```

```
dense_2 (Dense)           (None, 8)              8200
_________________________________________________________________
dense_3 (Dense)           (None, 8)              72
_________________________________________________________________
dense_4 (Dense)           (None, 2)              18
=================================================================
Total params: 1,057,890
Trainable params: 1,057,890
Non-trainable params: 0
_________________________________________________________________
None
Train on 6000 samples, validate on 3000 samples
Epoch 1/33
2020-05-11 20:08:52.294753: I T:\src\github\tensorflow\tensorflow\core\
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
    this TensorFlow binary was not compiled to use: AVX2
6000/6000 [==============================] - 1s 213us/step - loss: 0.6817 -
     acc: 0.6857 - val_loss: 0.6629 - val_acc: 0.8523
Epoch 2/33
6000/6000 [==============================] - 1s 144us/step - loss: 0.6360 -
     acc: 0.8898 - val_loss: 0.6082 - val_acc: 0.9045
Epoch 3/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.5633 -
     acc: 0.9281 - val_loss: 0.5393 - val_acc: 0.9090
Epoch 4/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.4783 -
     acc: 0.9522 - val_loss: 0.4645 - val_acc: 0.9275
Epoch 5/33
6000/6000 [==============================] - 1s 147us/step - loss: 0.3921 -
     acc: 0.9643 - val_loss: 0.3983 - val_acc: 0.9337
Epoch 6/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.3152 -
     acc: 0.9695 - val_loss: 0.3414 - val_acc: 0.9422
Epoch 7/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.2485 -
```

```
    acc: 0.9740 - val_loss: 0.3024 - val_acc: 0.9450
Epoch 8/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.1961 -
    acc: 0.9752 - val_loss: 0.2739 - val_acc: 0.9487
Epoch 9/33
6000/6000 [==============================] - 1s 147us/step - loss: 0.1551 -
    acc: 0.9804 - val_loss: 0.2580 - val_acc: 0.9510
Epoch 10/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.1231 -
    acc: 0.9829 - val_loss: 0.2494 - val_acc: 0.9527
Epoch 11/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.0973 -
    acc: 0.9863 - val_loss: 0.2457 - val_acc: 0.9537
Epoch 12/33
6000/6000 [==============================] - 1s 144us/step - loss: 0.0769 -
    acc: 0.9901 - val_loss: 0.2473 - val_acc: 0.9535
Epoch 13/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.0610 -
    acc: 0.9915 - val_loss: 0.2474 - val_acc: 0.9545
Epoch 00013: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png

# Evaluate on test data
3000/3000 [==============================] - 0s 39us/step
test loss, test acc: [0.2119911164045334, 0.9601666927337646]



C:\Users\User\Desktop\gradu\tensorflow>py 20.py
Using TensorFlow backend.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
   fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
    25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
    3\nSkipping line 1091: expected 1 fields, saw 4\n'
```

```
Loaded e_mono_50mev.txt with 6000 events.
Saved e_mono_50mev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_30_to_2gev.txt with 6000 events.
Saved p_30_to_2gev.txt.png
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 8200
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 18
=================================================================
Total params: 1,057,890
Trainable params: 1,057,890
Non-trainable params: 0
_________________________________________________________________
None
Train on 6000 samples, validate on 3000 samples
Epoch 1/33
2020-05-11 20:09:27.887018: I T:\src\github\tensorflow\tensorflow\core\
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
    this TensorFlow binary was not compiled to use: AVX2
6000/6000 [==============================] - 1s 210us/step - loss: 0.6836 -
     acc: 0.6488 - val_loss: 0.6724 - val_acc: 0.7420
Epoch 2/33
6000/6000 [==============================] - 1s 144us/step - loss: 0.6504 -
     acc: 0.8097 - val_loss: 0.6438 - val_acc: 0.7653
Epoch 3/33
```

```
6000/6000 [==============================] - 1s 145us/step - loss: 0.5933 -
    acc: 0.8458 - val_loss: 0.5841 - val_acc: 0.8075
Epoch 4/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.4993 -
    acc: 0.8718 - val_loss: 0.5228 - val_acc: 0.7827
Epoch 5/33
6000/6000 [==============================] - 1s 147us/step - loss: 0.4085 -
    acc: 0.8982 - val_loss: 0.4694 - val_acc: 0.8143
Epoch 6/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.3267 -
    acc: 0.9242 - val_loss: 0.4291 - val_acc: 0.8358
Epoch 7/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.2649 -
    acc: 0.9395 - val_loss: 0.3973 - val_acc: 0.8770
Epoch 8/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.2151 -
    acc: 0.9513 - val_loss: 0.3727 - val_acc: 0.8832
Epoch 9/33
6000/6000 [==============================] - 1s 144us/step - loss: 0.1734 -
    acc: 0.9592 - val_loss: 0.3586 - val_acc: 0.8902
Epoch 10/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.1421 -
    acc: 0.9688 - val_loss: 0.3539 - val_acc: 0.8958
Epoch 11/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.1179 -
    acc: 0.9732 - val_loss: 0.3692 - val_acc: 0.9052
Epoch 00011: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png

# Evaluate on test data
3000/3000 [==============================] - 0s 39us/step
test loss, test acc: [0.3255460262298584, 0.9210000038146973]
```

A.3.11 Monoenergetic 5, 10, 25, 50 MeV electrons vs flat 30-2000 MeV protons - trained with <50 MeV e

```
C:\Users\User\Desktop\gradu\tensorflow>py 20_custom_validation.py
Using TensorFlow backend.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_max_50mev.txt with 6000 events.
Saved e_max_50mev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_30_to_2gev.txt with 6000 events.
Saved p_30_to_2gev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_mono_5mev.txt with 6000 events.
Saved e_mono_5mev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_30_to_2gev.txt with 6000 events.
Saved p_30_to_2gev.txt.png
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
```

```
dense_2 (Dense)           (None, 8)              8200
_________________________________________________________________
dense_3 (Dense)           (None, 8)              72
_________________________________________________________________
dense_4 (Dense)           (None, 2)              18
=================================================================
Total params: 1,057,890
Trainable params: 1,057,890
Non-trainable params: 0
_________________________________________________________________
None
Train on 6000 samples, validate on 3000 samples
Epoch 1/33
2020-05-11 20:11:54.586975: I T:\src\github\tensorflow\tensorflow\core\
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
    this TensorFlow binary was not compiled to use: AVX2
6000/6000 [==============================] - 1s 244us/step - loss: 0.6767 -
     acc: 0.6497 - val_loss: 0.6521 - val_acc: 0.7582
Epoch 2/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.6156 -
     acc: 0.7965 - val_loss: 0.5879 - val_acc: 0.7792
Epoch 3/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.5288 -
     acc: 0.8624 - val_loss: 0.5104 - val_acc: 0.8560
Epoch 4/33
6000/6000 [==============================] - 1s 144us/step - loss: 0.4364 -
     acc: 0.8983 - val_loss: 0.4386 - val_acc: 0.8835
Epoch 5/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.3512 -
     acc: 0.9255 - val_loss: 0.3884 - val_acc: 0.9093
Epoch 6/33
6000/6000 [==============================] - 1s 165us/step - loss: 0.2821 -
     acc: 0.9440 - val_loss: 0.3424 - val_acc: 0.9007
Epoch 7/33
6000/6000 [==============================] - 1s 162us/step - loss: 0.2255 -
```

```
    acc: 0.9568 - val_loss: 0.3069 - val_acc: 0.9245
Epoch 8/33
6000/6000 [==============================] - 1s 178us/step - loss: 0.1809 -
    acc: 0.9649 - val_loss: 0.2953 - val_acc: 0.9333
Epoch 9/33
6000/6000 [==============================] - 1s 187us/step - loss: 0.1465 -
    acc: 0.9718 - val_loss: 0.2799 - val_acc: 0.9338
Epoch 10/33
6000/6000 [==============================] - 1s 191us/step - loss: 0.1221 -
    acc: 0.9747 - val_loss: 0.2782 - val_acc: 0.9353
Epoch 11/33
6000/6000 [==============================] - 1s 167us/step - loss: 0.1017 -
    acc: 0.9795 - val_loss: 0.2818 - val_acc: 0.9368
Epoch 00011: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png

# Evaluate on test data
3000/3000 [==============================] - 0s 46us/step
test loss, test acc: [0.19277648627758026, 0.9478333592414856]



C:\Users\User\Desktop\gradu\tensorflow>py 20_custom_validation.py
Using TensorFlow backend.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
   fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
    25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
    3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_max_50mev.txt with 6000 events.
Saved e_max_50mev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
   fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
    25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
    3\nSkipping line 1091: expected 1 fields, saw 4\n'
```

```
Loaded p_30_to_2gev.txt with 6000 events.
Saved p_30_to_2gev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_mono_10mev.txt with 6000 events.
Saved e_mono_10mev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_30_to_2gev.txt with 6000 events.
Saved p_30_to_2gev.txt.png
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 8200
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 18
=================================================================
Total params: 1,057,890
Trainable params: 1,057,890
Non-trainable params: 0
_________________________________________________________________
None
Train on 6000 samples, validate on 3000 samples
Epoch 1/33
2020-05-11 20:14:32.118353: I T:\src\github\tensorflow\tensorflow\core\
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
    this TensorFlow binary was not compiled to use: AVX2
```

```
6000/6000 [==============================] - 1s 222us/step - loss: 0.6761 -
    acc: 0.5573 - val_loss: 0.6549 - val_acc: 0.7480
Epoch 2/33
6000/6000 [==============================] - 1s 150us/step - loss: 0.6261 -
    acc: 0.7103 - val_loss: 0.6092 - val_acc: 0.8367
Epoch 3/33
6000/6000 [==============================] - 1s 148us/step - loss: 0.5675 -
    acc: 0.7975 - val_loss: 0.5654 - val_acc: 0.8437
Epoch 4/33
6000/6000 [==============================] - 1s 150us/step - loss: 0.5084 -
    acc: 0.8658 - val_loss: 0.5328 - val_acc: 0.8690
Epoch 5/33
6000/6000 [==============================] - 1s 150us/step - loss: 0.4590 -
    acc: 0.9009 - val_loss: 0.5118 - val_acc: 0.8952
Epoch 6/33
6000/6000 [==============================] - 1s 181us/step - loss: 0.4165 -
    acc: 0.9309 - val_loss: 0.5009 - val_acc: 0.9110
Epoch 7/33
6000/6000 [==============================] - 1s 179us/step - loss: 0.3846 -
    acc: 0.9477 - val_loss: 0.4974 - val_acc: 0.9188
Epoch 8/33
6000/6000 [==============================] - 1s 168us/step - loss: 0.3563 -
    acc: 0.9593 - val_loss: 0.4959 - val_acc: 0.9293
Epoch 00008: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png

# Evaluate on test data
3000/3000 [==============================] - 0s 44us/step
test loss, test acc: [0.401607871055603, 0.9526666402816772]



C:\Users\User\Desktop\gradu\tensorflow>py 20_custom_validation.py
Using TensorFlow backend.
```

```
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_max_50mev.txt with 6000 events.
Saved e_max_50mev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_30_to_2gev.txt with 6000 events.
Saved p_30_to_2gev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_mono_25mev.txt with 6000 events.
Saved e_mono_25mev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_30_to_2gev.txt with 6000 events.
Saved p_30_to_2gev.txt.png
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 8200
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 18
=================================================================
```

```
Total params: 1,057,890
Trainable params: 1,057,890
Non-trainable params: 0
_________________________________________________________________
None
Train on 6000 samples, validate on 3000 samples
Epoch 1/33
2020-05-11 20:15:48.841803: I T:\src\github\tensorflow\tensorflow\core\
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
    this TensorFlow binary was not compiled to use: AVX2
6000/6000 [==============================] - 1s 211us/step - loss: 0.6874 -
     acc: 0.5732 - val_loss: 0.6782 - val_acc: 0.7417
Epoch 2/33
6000/6000 [==============================] - 1s 144us/step - loss: 0.6619 -
     acc: 0.7438 - val_loss: 0.6486 - val_acc: 0.7997
Epoch 3/33
6000/6000 [==============================] - 1s 148us/step - loss: 0.6186 -
     acc: 0.8309 - val_loss: 0.6041 - val_acc: 0.8378
Epoch 4/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.5599 -
     acc: 0.8760 - val_loss: 0.5482 - val_acc: 0.8588
Epoch 5/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.4907 -
     acc: 0.9044 - val_loss: 0.4906 - val_acc: 0.8867
Epoch 6/33
6000/6000 [==============================] - 1s 144us/step - loss: 0.4180 -
     acc: 0.9270 - val_loss: 0.4360 - val_acc: 0.9008
Epoch 7/33
6000/6000 [==============================] - 1s 148us/step - loss: 0.3513 -
     acc: 0.9400 - val_loss: 0.3905 - val_acc: 0.9123
Epoch 8/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.2938 -
     acc: 0.9493 - val_loss: 0.3553 - val_acc: 0.9208
Epoch 9/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.2430 -
```

```
    acc: 0.9561 - val_loss: 0.3252 - val_acc: 0.9265
Epoch 10/33
6000/6000 [==============================] - 1s 149us/step - loss: 0.2011 -
    acc: 0.9623 - val_loss: 0.3094 - val_acc: 0.9320
Epoch 11/33
6000/6000 [==============================] - 1s 144us/step - loss: 0.1675 -
    acc: 0.9676 - val_loss: 0.2958 - val_acc: 0.9302
Epoch 12/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.1379 -
    acc: 0.9732 - val_loss: 0.2920 - val_acc: 0.9358
Epoch 13/33
6000/6000 [==============================] - 1s 148us/step - loss: 0.1159 -
    acc: 0.9765 - val_loss: 0.2890 - val_acc: 0.9358
Epoch 00013: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png

# Evaluate on test data
3000/3000 [==============================] - 0s 44us/step
test loss, test acc: [0.2077401876449585, 0.9558333158493042]



C:\Users\User\Desktop\gradu\tensorflow>py 20_custom_validation.py
Using TensorFlow backend.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
   fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
    25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
    3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_max_50mev.txt with 6000 events.
Saved e_max_50mev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
   fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
    25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
    3\nSkipping line 1091: expected 1 fields, saw 4\n'
```

```
Loaded p_30_to_2gev.txt with 6000 events.
Saved p_30_to_2gev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_mono_50mev.txt with 6000 events.
Saved e_mono_50mev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_30_to_2gev.txt with 6000 events.
Saved p_30_to_2gev.txt.png
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 8200
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 18
=================================================================
Total params: 1,057,890
Trainable params: 1,057,890
Non-trainable params: 0
_________________________________________________________________
None
Train on 6000 samples, validate on 3000 samples
Epoch 1/33
2020-05-11 20:16:34.469448: I T:\src\github\tensorflow\tensorflow\core\
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
    this TensorFlow binary was not compiled to use: AVX2
```

```
6000/6000 [==============================] - 1s 212us/step - loss: 0.6867 -
    acc: 0.5538 - val_loss: 0.6764 - val_acc: 0.6682
Epoch 2/33
6000/6000 [==============================] - 1s 144us/step - loss: 0.6596 -
    acc: 0.6624 - val_loss: 0.6449 - val_acc: 0.6930
Epoch 3/33
6000/6000 [==============================] - 1s 148us/step - loss: 0.6165 -
    acc: 0.7412 - val_loss: 0.6028 - val_acc: 0.7933
Epoch 4/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.5622 -
    acc: 0.8223 - val_loss: 0.5585 - val_acc: 0.7857
Epoch 5/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.5033 -
    acc: 0.8765 - val_loss: 0.5146 - val_acc: 0.8523
Epoch 6/33
6000/6000 [==============================] - 1s 144us/step - loss: 0.4468 -
    acc: 0.9103 - val_loss: 0.4768 - val_acc: 0.8918
Epoch 7/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.3927 -
    acc: 0.9359 - val_loss: 0.4481 - val_acc: 0.9028
Epoch 8/33
6000/6000 [==============================] - 1s 150us/step - loss: 0.3452 -
    acc: 0.9488 - val_loss: 0.4254 - val_acc: 0.9123
Epoch 9/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.3027 -
    acc: 0.9585 - val_loss: 0.4082 - val_acc: 0.9215
Epoch 10/33
6000/6000 [==============================] - 1s 144us/step - loss: 0.2671 -
    acc: 0.9642 - val_loss: 0.3964 - val_acc: 0.9262
Epoch 11/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.2347 -
    acc: 0.9720 - val_loss: 0.3993 - val_acc: 0.9253
Epoch 12/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.2059 -
    acc: 0.9766 - val_loss: 0.3991 - val_acc: 0.9280
```

```
Epoch 00012: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png

# Evaluate on test data
3000/3000 [==============================] - 0s 39us/step
test loss, test acc: [0.7468618154525757, 0.8486666679382324]
```

A.3.12 Monoenergetic 5 MeV electrons 11-26° zenith angle vs flat 30-2000 MeV protons - trained with <50 MeV e

```
C:\Users\User\Desktop\gradu\tensorflow>py 20_custom_validation.py
Using TensorFlow backend.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_max_50mev.txt with 6000 events.
Saved e_max_50mev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_30_to_2gev.txt with 6000 events.
Saved p_30_to_2gev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_5_11deg.txt with 6000 events.
Saved e_5_11deg.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
    fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
     25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
     3\nSkipping line 1091: expected 1 fields, saw 4\n'
```

```
Loaded p_30_to_2gev.txt with 6000 events.
Saved p_30_to_2gev.txt.png
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 8200
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 18
=================================================================
Total params: 1,057,890
Trainable params: 1,057,890
Non-trainable params: 0
_________________________________________________________________
None
Train on 6000 samples, validate on 3000 samples
Epoch 1/33
2020-05-11 21:40:50.881055: I T:\src\github\tensorflow\tensorflow\core\
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
    this TensorFlow binary was not compiled to use: AVX2
6000/6000 [==============================] - 1s 217us/step - loss: 0.6796 -
     acc: 0.6334 - val_loss: 0.6595 - val_acc: 0.6915
Epoch 2/33
6000/6000 [==============================] - 1s 148us/step - loss: 0.6320 -
     acc: 0.7689 - val_loss: 0.6081 - val_acc: 0.8317
Epoch 3/33
6000/6000 [==============================] - 1s 144us/step - loss: 0.5596 -
     acc: 0.8429 - val_loss: 0.5352 - val_acc: 0.8605
Epoch 4/33
6000/6000 [==============================] - 1s 145us/step - loss: 0.4736 -
     acc: 0.8941 - val_loss: 0.4566 - val_acc: 0.8755
Epoch 5/33
```

```
6000/6000 [==============================] - 1s 145us/step - loss: 0.3820 -
    acc: 0.9260 - val_loss: 0.3806 - val_acc: 0.9037
Epoch 6/33
6000/6000 [==============================] - 1s 148us/step - loss: 0.2972 -
    acc: 0.9460 - val_loss: 0.3197 - val_acc: 0.9180
Epoch 7/33
6000/6000 [==============================] - 1s 148us/step - loss: 0.2317 -
    acc: 0.9574 - val_loss: 0.2872 - val_acc: 0.9157
Epoch 8/33
6000/6000 [==============================] - 1s 148us/step - loss: 0.1822 -
    acc: 0.9660 - val_loss: 0.2569 - val_acc: 0.9330
Epoch 9/33
6000/6000 [==============================] - 1s 147us/step - loss: 0.1476 -
    acc: 0.9712 - val_loss: 0.2452 - val_acc: 0.9343
Epoch 10/33
6000/6000 [==============================] - 1s 149us/step - loss: 0.1217 -
    acc: 0.9746 - val_loss: 0.2412 - val_acc: 0.9342
Epoch 11/33
6000/6000 [==============================] - 1s 157us/step - loss: 0.1023 -
    acc: 0.9769 - val_loss: 0.2432 - val_acc: 0.9363
Epoch 00011: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png

# Evaluate on test data
3000/3000 [==============================] - 0s 39us/step
test loss, test acc: [0.7371160387992859, 0.8033333420753479]



C:\Users\User\Desktop\gradu\tensorflow>py 20_custom_validation.py
Using TensorFlow backend.
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
   fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
    25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
```

```
    3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_max_50mev.txt with 6000 events.
Saved e_max_50mev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
   fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
    25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
    3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_30_to_2gev.txt with 6000 events.
Saved p_30_to_2gev.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
   fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
    25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
    3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded e_5_26deg.txt with 6000 events.
Saved e_5_26deg.txt.png
b'Skipping line 5: expected 1 fields, saw 2\nSkipping line 6: expected 1
   fields, saw 2\nSkipping line 7: expected 1 fields, saw 2\nSkipping line
    25: expected 1 fields, saw 2\nSkipping line 26: expected 1 fields, saw
    3\nSkipping line 1091: expected 1 fields, saw 4\n'
Loaded p_30_to_2gev.txt with 6000 events.
Saved p_30_to_2gev.txt.png
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 8200
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 18
=================================================================
Total params: 1,057,890
Trainable params: 1,057,890
Non-trainable params: 0
```

```
_________________________________________________________________
None
Train on 6000 samples, validate on 3000 samples
Epoch 1/33
2020-05-11 21:42:40.492394: I T:\src\github\tensorflow\tensorflow\core\
    platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
    this TensorFlow binary was not compiled to use: AVX2
6000/6000 [==============================] - 1s 240us/step - loss: 0.6845 -
     acc: 0.5621 - val_loss: 0.6732 - val_acc: 0.5593
Epoch 2/33
6000/6000 [==============================] - 1s 161us/step - loss: 0.6564 -
     acc: 0.7047 - val_loss: 0.6443 - val_acc: 0.5903
Epoch 3/33
6000/6000 [==============================] - 1s 164us/step - loss: 0.6105 -
     acc: 0.8135 - val_loss: 0.5963 - val_acc: 0.8722
Epoch 4/33
6000/6000 [==============================] - 1s 164us/step - loss: 0.5504 -
     acc: 0.8853 - val_loss: 0.5468 - val_acc: 0.8678
Epoch 5/33
6000/6000 [==============================] - 1s 153us/step - loss: 0.4841 -
     acc: 0.9229 - val_loss: 0.4945 - val_acc: 0.9102
Epoch 6/33
6000/6000 [==============================] - 1s 158us/step - loss: 0.4140 -
     acc: 0.9433 - val_loss: 0.4380 - val_acc: 0.9262
Epoch 7/33
6000/6000 [==============================] - 1s 157us/step - loss: 0.3325 -
     acc: 0.9546 - val_loss: 0.3850 - val_acc: 0.9298
Epoch 8/33
6000/6000 [==============================] - 1s 161us/step - loss: 0.2612 -
     acc: 0.9630 - val_loss: 0.3545 - val_acc: 0.9327
Epoch 9/33
6000/6000 [==============================] - 1s 157us/step - loss: 0.2033 -
     acc: 0.9681 - val_loss: 0.3319 - val_acc: 0.9352
Epoch 10/33
6000/6000 [==============================] - 1s 156us/step - loss: 0.1569 -
```

```
    acc: 0.9741 - val_loss: 0.3203 - val_acc: 0.9368
Epoch 11/33
6000/6000 [==============================] - 1s 159us/step - loss: 0.1242 -
    acc: 0.9789 - val_loss: 0.3149 - val_acc: 0.9372
Epoch 12/33
6000/6000 [==============================] - 1s 161us/step - loss: 0.0978 -
    acc: 0.9832 - val_loss: 0.3216 - val_acc: 0.9402
Epoch 00012: early stopping
Saved Training and validation loss.png
Saved Training and validation accuracy.png

# Evaluate on test data
3000/3000 [==============================] - 0s 39us/step
test loss, test acc: [0.4889197051525116, 0.9108333587646484]
```

A.4 Source codes

A.4.1 Geant4 energy distribution

```
int GetEnergyFromRange(int min, int max)
{
       double energy = rand() % (max - min) + min;
       return GetEnergyFromSpectra(energy);
}

double GetEnergyFromSpectra(double E)
{
       double r = ((double)rand() / (RAND_MAX));

       //electrons:
       double maxIntensity = 100000.0;
       double Id = 10285.7;
       double gamma = -1.63;
       double Ebreak = -9.4;
       double E0 = 1;
```

```
        double e = 2.71828;

        //double Ee = Id * pow((E / E0), gamma) / 10 / maxIntensity;
        double Ee = Id * pow((E / E0), gamma) * pow(e, (-E / Ebreak));
        return Ee > r ? E : -1.0;

        // protons:
        int a = 1;
        int    b = -271.451;
        int    c = 3776.26;
        int    d = -60.9773;

        double Ep = a*pow(e, -(((E - b)*(E - b)) / (2 * (c*c)))) + d / E;
        return Ep > r ? E : -1.0;
}
```

A.4.2 Tensorflow Python program

```
import numpy as np
import pandas as pd
import csv
import math
import matplotlib as mpl
from matplotlib import pyplot as plt
from matplotlib import colors
import matplotlib.gridspec as gridspec

import keras
from keras.callbacks import EarlyStopping
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM, ELU, Flatten, Embedding,
    SpatialDropout1D, Conv3D, MaxPooling3D
from keras.utils import plot_model, to_categorical
from keras import initializers
```

```
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from keras import backend as K
from sklearn import preprocessing
import tensorflow as tf

# np.set_printoptions(threshold=10000)
np.set_printoptions(threshold=np.nan)
np.set_printoptions(precision=3)

#-----------------------
# parameters
#-----------------------

DATASETS = []
DATASETS.append("e_5_11deg.txt")
DATASETS.append("p_mono_50mev.txt")

VALIDATIONSETS = []
VALIDATIONSETS.append("e_mono_50mev.txt")
VALIDATIONSETS.append("p_max_2gev.txt")

TRAININGEPOCHS = 33
BATCHSIZEDIVIDER = 16
HIDDENLAYERS = 2

PANELS = 4
CELLSPERSIDE = 16
PANELCELLS = CELLSPERSIDE*CELLSPERSIDE
TOTALCELLS = PANELS*PANELCELLS

USEBINARYVALUES = 0   # binarize input or not
MAXSETVALUES = 6000
VERIFICATIONS = MAXSETVALUES

#-----------------------
```

```
# functions
#-----------------------

# p = plate, c = cell, e = energy (MeV)
def parseEvent(pc):
	p = int(pc[pc.index("P")+1])
	c = int(pc[pc.index("C")+1:pc.index(":")])
	e = float(pc[pc.index(":")+1:])
	return p, c, e

def getEventHits(event):
	hits = []
	for i in range(1, len(event)): # start at 1 because 0 is "HITS"
		hits.append(parseEvent(event[i]))

	return hits

def mapHitsToInputsCNN(hits):
	hits3d = np.zeros((CELLSPERSIDE, CELLSPERSIDE, CELLSPERSIDE, 3))
	x = 0
	y = 0

	# map input cells into a single vector - hits[i][0] = panel number,
	   hits[i][1] = cell number, hits[i][2] = energy:
	for i in range(len(hits)):
		x = hits[i][1] % CELLSPERSIDE
		y = int(hits[i][1] / CELLSPERSIDE)
		hits3d[x, y, hits[i][0]] = hits[i][2]

	return hits3d

def readEventsCNN(dataSet, binarize = USEBINARYVALUES):
	df = pd.read_csv(dataSet, error_bad_lines=False)
	allEvents = []
	a = 0
```

```
        for i in range(len(df)):
                event = df.iloc[i][0].rsplit()
                if event[0] == 'HITS:':
                        hits = getEventHits(event)
                        hits3d = mapHitsToInputsCNN(hits)
                        allEvents.append(hits3d)
                        a += 1

                if a == MAXSETVALUES:
                        break

        allEventsNA = np.ndarray(np.shape(allEvents))
        for i in range(len(allEventsNA)):
                allEventsNA[i] = allEvents[i]

        print("Loaded " +str(dataSet) + " with " + str(len(allEventsNA)) + "
             events.")
        # plotDetector(hitCounts, dataSet)

        return allEventsNA

def mapHitsToInputs(hits, hitCounts, binarize=0):
        row = [0]*TOTALCELLS
        index = 0

        # map input cells into a single vector - hits[i][0] = panel number,
           hits[i][1] = cell number, hits[i][2] = energy:
        for i in range(len(hits)):

                # read rows plate by plate:
                index = PANELCELLS*hits[i][0] + hits[i][1]

                # read rows through the detector:
                # index = (hits[i][1] % CELLSPERSIDE) + CELLSPERSIDE*hits[i
                    ][0] + (int(hits[i][1] / CELLSPERSIDE))*PANELS*
```

```
                    CELLSPERSIDE

                hitCounts[index] += 1

                if binarize == 0:
                        row[index] = hits[i][2]
                else:
                        if hits[i][2] > 0:
                                row[index] = 1
                        else:
                                row[index] = 0

        return row

def readEvents(dataSet, binarize = USEBINARYVALUES):
        df = pd.read_csv(dataSet, error_bad_lines=False)
        allEvents = []
        hitCounts = [0]*TOTALCELLS
        a = 0
        for i in range(len(df)):
                event = df.iloc[i][0].rsplit()
                if event[0] == 'HITS:':
                        hits = getEventHits(event)
                        row = mapHitsToInputs(hits, hitCounts, binarize)
                        allEvents.append(row)
                        a += 1

                if a == MAXSETVALUES:
                        break

        allEventsNA = np.ndarray(np.shape(allEvents))

        for i in range(len(allEventsNA)):
                allEventsNA[i] = allEvents[i]
```

```
        #normalize and reshape:
        # allEventsNA = allEventsNA / allEventsNA.max()
        allEventsNA = allEventsNA[:,:].reshape(allEventsNA.shape[0], len(
            allEventsNA[0])).astype( 'float32' )

        print("Loaded " +str(dataSet) + " with " + str(len(allEventsNA)) + "
             events.")
        plotDetector(hitCounts, dataSet)

        return allEventsNA

def evaluateModel():
        failCount = 0

        for i in range(len(x_test)):
                p = model.predict(np.expand_dims(x_test[i],0))[0][0]
                result = p if p==1 else (p * len(datasetsRead)) // 1
                target = y_test[i] * (len(datasetsRead) - 1)

                if result != target:
                        failCount += 1

                if i % 1000 == 0:
                        print(str(i) + ' values tested')

        print('Testing finished with ' + str(len(x_test)) + ' values')
        failPercentage = round(100*(failCount/(len(x_test))), 4)

        f = open('test.txt','a')

        #model.summary(print_fn=lambda x: f.write(x + '\n'))
        print('Total incorrect predictions: ', failCount, '/', len(x_test),
            ',', failPercentage, '%')
        print(str(nrLayers) +';' + str(nrEpochs) + ';' + str(
            batchSizeDivider) + ';' + str(failPercentage), file=f)
```

```
        f.close()

        return failPercentage

def constructModel(hiddenLayerCount):
        model = Sequential()
        model.add(Dense(int(x_train.shape[1]), input_dim=x_train.shape[1],
            activation='relu'))
        # model.add(Dense(512, activation='relu', kernel_initializer='
            glorot_uniform', bias_initializer='zeros'))
        for i in range(hiddenLayerCount):
                # model.add(Dense(int(x_train.shape[1]/(2**(i+1))),
                    activation='relu'))  # consecutive layer is always 1/2 of
                     the previous one
                # model.add(Dense(int(x_train.shape[1]), activation='relu'))
                # model.add(Dropout(0.3))
                model.add(Dense(8, activation='relu', kernel_initializer='
                    glorot_uniform', bias_initializer='zeros'))


        model.add(Dense(len(DATASETS), activation='sigmoid'))
        model.compile(loss='binary_crossentropy',
                              optimizer='rmsprop',
                              metrics=['accuracy'])

        return model

def constructCNN(hiddenLayerCount, lr_f):

        no_classes = len(DATASETS)
        learning_rate = 0.001 * lr_f*lr_f
        sample_shape = (CELLSPERSIDE, CELLSPERSIDE, CELLSPERSIDE, 3)

        model = Sequential()
        model.add(Conv3D(32, kernel_size=(3, 3, 3), activation='relu',
```

```
            kernel_initializer='he_uniform', input_shape=sample_shape))
        model.add(MaxPooling3D(pool_size=(2, 2, 2)))
        model.add(Conv3D(64, kernel_size=(3, 3, 3), activation='relu',
            kernel_initializer='he_uniform'))
        model.add(MaxPooling3D(pool_size=(2, 2, 2)))
        model.add(Flatten())
        model.add(Dense(256, activation='relu', kernel_initializer='
            he_uniform'))
        model.add(Dense(no_classes, activation='softmax'))
        model.compile(loss=keras.losses.categorical_crossentropy,
                                    optimizer=keras.optimizers.Adam(lr=
                                        learning_rate),
                                    metrics=['accuracy'])

        return model

def plotDetector(detectorVector, title):
        hits = unwindCellVector(detectorVector)
        maxHits = np.max(hits)

        cmap2 = mpl.colors.LinearSegmentedColormap.from_list('my_colormap',
            [(0,'black'),(0.03, 'blue'),(1, 'red')], 256)

        fig, ax = plt.subplots(1, PANELS, figsize=(15, 3)) #,
            constrained_layout=True
        for i in range(PANELS):
                im = ax[i].imshow(hits[i], interpolation='none', aspect='auto
                    ', cmap = cmap2, vmin=0, vmax=maxHits, origin='lower')
                # print(hits[i])
                ax[i].set_title('Panel ' + str(i+1))
                # ax[i].set_xticks(np.arange(0, CELLSPERSIDE, 4))
                # ax[i].grid(True)

        fig.subplots_adjust(top=0.7)
```

```
        cbar_ax = fig.add_axes([0.35, 0.85, 0.3, 0.05 ])
        fig.colorbar(im, cax=cbar_ax, orientation="horizontal")
        fig.suptitle(title, fontsize=16)
        # plt.tight_layout()
        # plt.show()
        plt.savefig(title + '.png')
        print('Saved ' + title + '.png')

def plotHistory(history):
        plt.clf()
        loss = history.history['loss']
        val_loss = history.history['val_loss']
        epochs = range(1, len(loss) + 1)
        plt.plot(epochs, loss, 'g', label='Training loss')
        plt.plot(epochs, val_loss, 'y', label='Validation loss')
        plt.title('Training and validation loss')
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.legend()
        # plt.show()

        plt.savefig('Training and validation loss' + '.png')
        print('Saved ' + 'Training and validation loss' + '.png')

        plt.clf()
        acc = history.history['acc']
        val_acc = history.history['val_acc']
        epochs = range(1, len(acc) + 1)
        plt.plot(epochs, acc, 'g', label='Accuracy')
        plt.plot(epochs, val_acc, 'y', label='Validation accuracy')
        plt.title('Training and validation accuracy')
        plt.xlabel('Epochs')
        plt.ylabel('acc')
        plt.legend()
        # plt.show()
```

```
        plt.savefig('Training and validation accuracy' + '.png')
        print('Saved ' + 'Training and validation accuracy' + '.png')

def unwindCellVector(vector):
        detector = np.ndarray(shape=(CELLSPERSIDE, CELLSPERSIDE))
        detectorFinal = []
        row = 0

        detector[0][0] = vector[0]
        for i in range(1, len(vector)):
                if i % CELLSPERSIDE == 0:
                        row += 1

                if row > CELLSPERSIDE -1:
                        row = 0
                        detectorFinal.append(detector.copy())

                detector[row][i%CELLSPERSIDE] = vector[i]

        detectorFinal.append(detector.copy())

        return detectorFinal

def calculateCovariance(trainingSet, title):
        covarianceMatrix = []
        covarianceMatrix = np.cov(trainingSet.T)
        covarianceMatrix = covarianceMatrix / covarianceMatrix.max() # make
            it a correlation matrix

        variances = [0]*TOTALCELLS
        variances = np.diag(covarianceMatrix)
        plotDetector(variances, title + ' var')

        np.fill_diagonal(covarianceMatrix, 0) # reset diagonal for easier
```

```
            covariance summation

        covarianceSums = [0]*TOTALCELLS
        covarianceSums = covarianceMatrix.sum(axis=1)
        plotDetector(covarianceSums, title + ' cov sums')

        np.savetxt(title + "_cov.txt", covarianceMatrix , delimiter=", ",
            fmt='%1.3f')

def saveCurrentWeightsToText():
        weightsAfter = model.get_weights()
        w = open('model_weights.txt','w')
        print(weightsAfter, file=w)

        # for i in range(3, len(weightsAfter)):
                # print(weightsAfter[i])
                # print(weightsAfter[i].shape)

        # get difference of weights:
        # weightsDelta = weightsAfter.copy()
        # for i in range(len(weightsBefore)):
                # weightsDelta[i] = weightsBefore[i] - weightsAfter[i]
        # print(weightsDelta)
        print("Saved weights.")


#-----------------------
# the program
#-----------------------

# read the data (read_csv seems to omit first row):
datasetsRead = []
for i in range(len(DATASETS)):
        datasetsRead.append(readEvents(DATASETS[i]))
        # datasetsRead.append(readEventsCNN(DATASETS[i]))
```

```
        # calculateCovariance(datasetsRead[i], str(DATASETS[i]))

# generate training matrix:
first = 1
for i in datasetsRead:
        if first == 1:
                x_train = i[:int(len(i)/2)]
                x_test = i[int(len(i)/2):]
                first = 0
        else:
                x_train = np.append(x_train, i[:int(len(i)/2)], axis=0)
                x_test = np.append(x_test, i[int(len(i)/2):], axis=0)

# assign goal output value for each dataset:
y_train = np.zeros(len(x_train))
y_test = np.zeros(len(x_test))
a = 0
for i in range(1, len(x_train)):
        if i%(int(MAXSETVALUES/2))== 0:
                a += 1 / (len(datasetsRead)-1)
        y_train[i] = a
        y_test[i] = a

y_train = to_categorical(y_train).astype(np.integer)
y_test = to_categorical(y_test).astype(np.integer)


# Reserve half of the samples for validation
x_val = x_test[len(x_test)//4:3*len(x_test)//4]
y_val = y_test[len(y_test)//4:3*len(x_test)//4]
# temp = x_test[:len(x_test)//4]
# temp = np.append(temp, x_test[3*len(x_test)//4:], axis=0)
# x_test = temp
# temp = y_test[:len(y_test)//4]
# temp = np.append(temp, y_test[3*len(y_test)//4:], axis=0)
```

```
# y_test = temp




# read the data (read_csv seems to omit first row):
valsetsRead = []
for i in range(len(VALIDATIONSETS)):
        valsetsRead.append(readEvents(VALIDATIONSETS[i]))

# generate training matrix:
first = 1
for i in valsetsRead:
        if first == 1:
                #x_train = i[:int(len(i)/2)]
                x_test = i[int(len(i)/2):]
                first = 0
        else:
                #x_train = np.append(x_train, i[:int(len(i)/2)], axis=0)
                x_test = np.append(x_test, i[int(len(i)/2):], axis=0)

# assign goal output value for each dataset:
# y_train = np.zeros(len(x_train))
y_test = np.zeros(len(x_test))
a = 0
for i in range(1, len(x_train)):
        if i%(int(MAXSETVALUES/2))== 0:
                a += 1 / (len(valsetsRead)-1)
        # y_train[i] = a
        y_test[i] = a

# y_train = to_categorical(y_train).astype(np.integer)
y_test = to_categorical(y_test).astype(np.integer)
```

```
# Reserve half of the samples for validation
# x_val = x_test[len(x_test)//4:3*len(x_test)//4]
# y_val = y_test[len(y_test)//4:3*len(x_test)//4]
temp = x_test[:len(x_test)//4]
temp = np.append(temp, x_test[3*len(x_test)//4:], axis=0)
x_test = temp
temp = y_test[:len(y_test)//4]
temp = np.append(temp, y_test[3*len(y_test)//4:], axis=0)
y_test = temp




# reset log file:
f = open('test.txt','w')
f.close()

bestRate = 100
bestNrLayers = 0
bestnrEpochs = 0
bestbatchSizeDivider = 0

bestAcc = 0

batchSizeDivider = BATCHSIZEDIVIDER
nrEpochs = TRAININGEPOCHS
nrLayers = HIDDENLAYERS
learningRateX = 3

# for nrLayers in range(2, 10):
# for nrEpochs in range(3, 33, 5):
```

```
# for batchSizeDivider in range(8, 64):

for i in range(1):

        # model = constructCNN(nrLayers, learningRateX)
        model = constructModel(nrLayers)
        print(model.summary())

        callbacks = [
    EarlyStopping(
        # Stop training when ‘val_loss‘ is no longer improving
        monitor='val_loss',
        # "no longer improving" being defined as "no better than 1e-2 less"
        min_delta=1e-2,
        # "no longer improving" being further defined as "for at least 2
            epochs"
        patience=2,
        verbose=1)
        ]


        plot_model(model, to_file='model' + str(nrLayers) + '.png',
            show_shapes=True)

        history = model.fit(x_train, y_train,
                      epochs=nrEpochs,
                      batch_size=int(len(x_train)/batchSizeDivider),
                      verbose=1,
                      shuffle=True,
                      callbacks=callbacks,
                      validation_data=(x_val, y_val))

        plotHistory(history)
        # failPercentage = evaluateModel()
```

```
# The returned "history" object holds a record
# of the loss values and metric values during training
# print('\nhistory dict:', history.history)

# Evaluate the model on the test data using `evaluate`
print('\n# Evaluate on test data')
results = model.evaluate(x_test, y_test, batch_size=len(x_test))
print('test loss, test acc:', results)

acc = results[1]

if acc > bestAcc:
       bestAcc = acc

# print("Best acc so far: " + str(bestAcc) + " with nrLayers: " +
    str(nrLayers))
# if failPercentage < bestRate:
       # bestRate = failPercentage
       # bestNrLayers = nrLayers
       # bestnrEpochs = nrEpochs
       # bestbatchSizeDivider = batchSizeDivider
       # model.save_weights("model_weights_" + str(bestRate) + "_" +
            str(bestNrLayers) + "_" + str(bestnrEpochs) + "_" + str(
           bestbatchSizeDivider) + ".h5")
       # saveCurrentWeightsToText()

# print('nrLayers: ' + str(nrLayers) +
       # ', nrEpochs:' + str(nrEpochs) +
       # ', batchSizeDivider:' + str(batchSizeDivider) +
       # ', failPercentage: ' + str(failPercentage))

# print('Best configuration so far: ' + 'nrLayers: ' + str(
    bestNrLayers) +
# ', nrEpochs:' + str(bestnrEpochs) +
# ', batchSizeDivider:' + str(bestbatchSizeDivider) +
```

```
        # ', failPercentage: ' + str(bestRate))

        print('\n')

        # print('\n# Generate predictions for 11 samples')
        # predictions = model.predict(x_test[:22])
        # print(predictions)




# print('Run finished. Best configuration: ' + 'nrLayers: ' + str(
    bestNrLayers) +
                            # ', nrEpochs:' + str(bestnrEpochs) +
                            # ', batchSizeDivider:' + str(
                                bestbatchSizeDivider) +
                            # ', failPercentage: ' + str(bestRate))
```

[88] [89] [90] [91] [92] [93]

A.5 Test results overview

Scenario	Det. acc.
Monoen. 50 MeV protons vs monoen. 50 MeV electrons	99%
Monoen. 50 MeV protons vs monoen. 5 MeV electrons	96%
Flat <50 MeV electrons vs flat <2 GeV protons 6000 values	94%
Flat <50 MeV electrons vs flat <2 GeV protons 140000 values	95%
Flat <50 MeV electrons vs flat 30 - 2000 MeV protons	94%
Flat <50 MeV electrons vs flat 30 - 2000 MeV protons binary values	93%
<50 MeV electrons vs 30 - 2000 MeV protons, both realistic values	93%
Monoen. 50 MeV protons vs monoen. 5 MeV electrons 11°	98%
Monoen. 50 MeV protons vs monoen. 5 MeV electrons 26°	98%
Monoen. 5 MeV e- vs <2000 MeV p flat, trained flat <50 MeV e-	94%
Monoen. 10 MeV e- vs 30-2000 MeV p flat, trained flat <50 MeV e-	95%
Monoen. 25 MeV e- vs 30-2000 MeV p flat, trained flat <50 MeV e-	95%
Monoen. 50 MeV e- vs 30-2000 MeV p flat, trained flat <50 MeV e-	84%
Monoen. 5 MeV e- 11° vs 30-2000 MeV p flat, trained flat <50 MeV e-	80%
Monoen. 5 MeV e- 26° vs 30-2000 MeV p flat, trained flat <50 MeV e-	91%