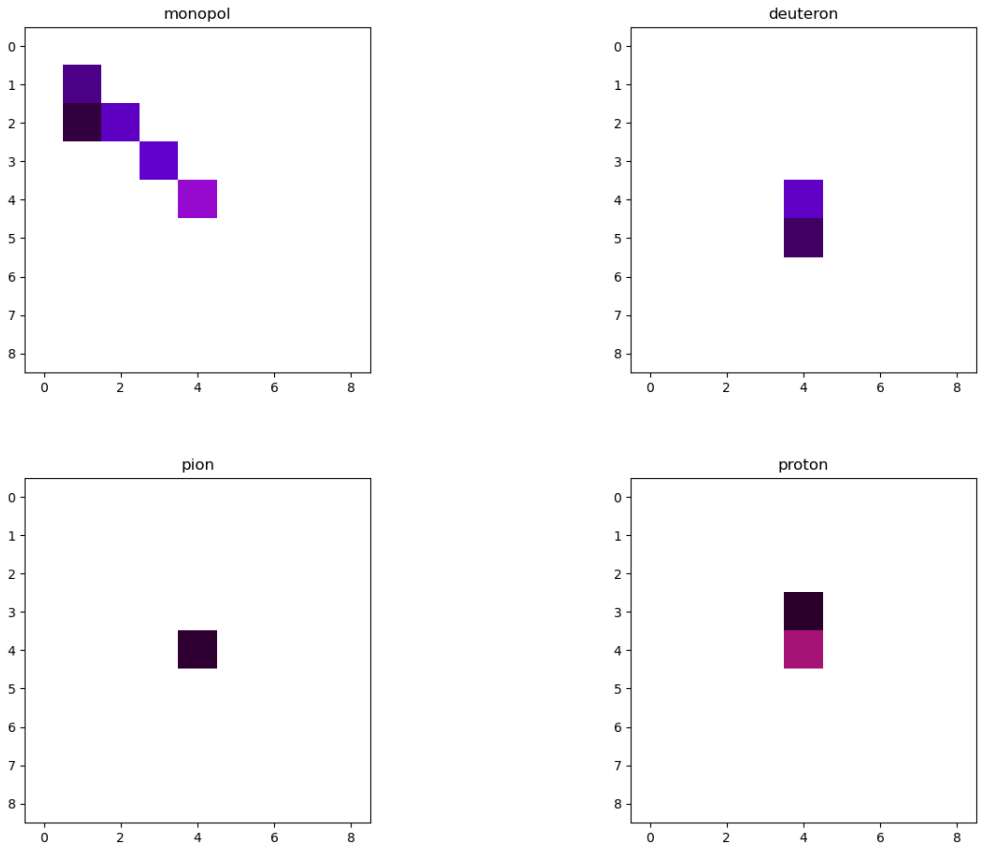
Multilayer perceptron for particle identification on simulated data from the pixel detector of Belle 2 using Tensorflow.Keras on GPU and CPU

Marvin Peter

October 9, 2020



Contents

1. Introduction

This elaboration deals with the topic of using “Multilayer Perceptrons” for particle identification as part of the analysis of simulated Belle 2 pixel detector data.

2. Multilayer Perceptron

A “Multilayer Perceptron” or short MLP is a special form of neural network that utilizes supervised learning for training. The base concept can be explained as following:
The MLP consists of the general structure (nodes), ordered in an input layer, an output layer and multiple hidden layers in the middle. Layers are connected between each other like shown in figure 1.

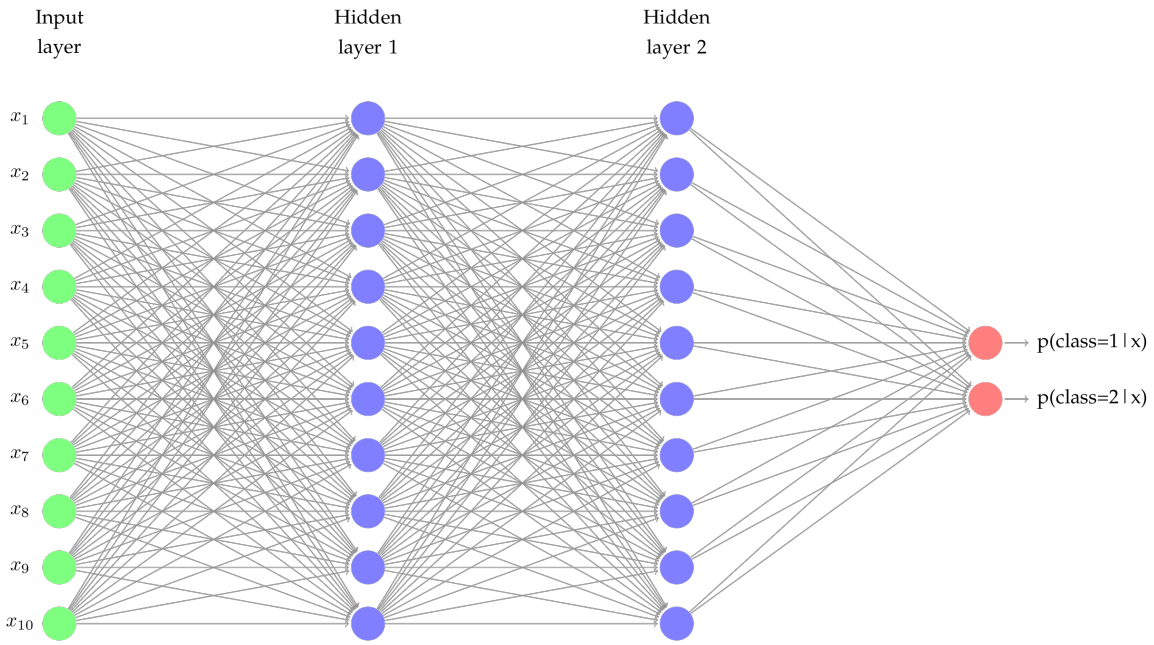


Figure 1: example MLP structure

Each connection line is a weight function between two nodes. To train the network training data is needed (supervised learning). After each run the error is calculated using a special loss function and the result is propagated back through the network, changing the weights between nodes accordingly. The training function always tries to minimize the loss which results in the increase of categorizing accuracy. If everything works fine this process will converge like shown in figure In the training process the training data is fed into the neural network in batches while the batch size can affect the performance of the training process (depending of CPU or GPU computation etc.). One epoch is completed after all data was used for training the network once. Usually a network is trained for a lot of epochs. Often a neural network consists not only of dense (fully connected) layers but also dropout layers that randomly deactivate nodes in single epochs so that the other nodes will be trained as well. This reduces overfitting and reduce generalization error.

2.1. Convolutional Neural Network (CNN)

A convolutional neural network (CNN) is a special case of a MLP that uses convolution layers at the front of the neural network to extract features from the image that can then be used by "normal" dense layers for categorization. When training a CNN it can learn any type of traditional image filter and more.

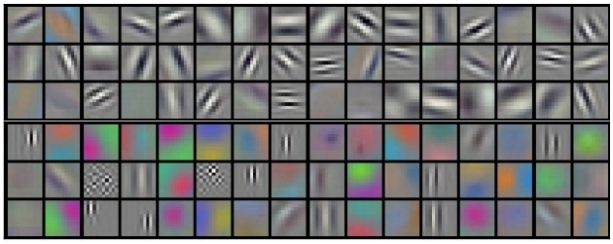


Figure 2: example filters learned by AlexNet. (source: Stanford University)

Usually a CNN consists of convolution layers, dropout layers and pooling layers at the front and fully connected (dense) layers at the back as shown in figure 3.

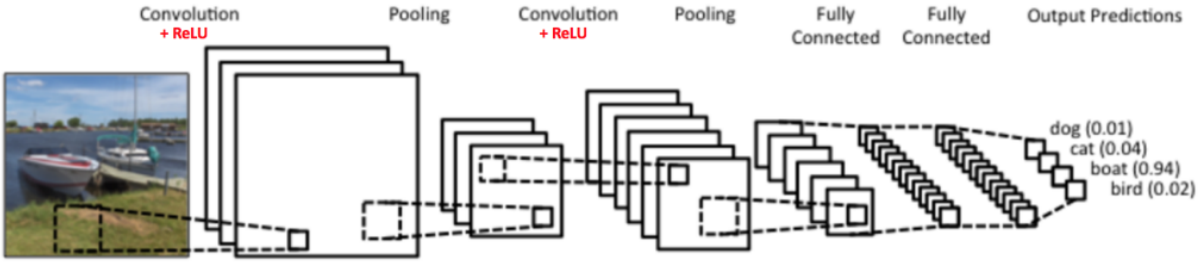


Figure 3: simple convolutional neural network that can categorize between dogs, cats, boats and birds. (source: ujjwalkarn)

Convolutional layers are similar to traditional image filters as they also have kernels (masks) that run over the image and there is a convolution between the image and the kernel.
In traditional image processing this is usually a matrix and the entries of this matrix determine the function of the kernel. A CNN can learn new filters: a feature extraction filter is also nothing else than a convolutional filter specially trained to (for example) extract a boat out of an image (very oversimplified).

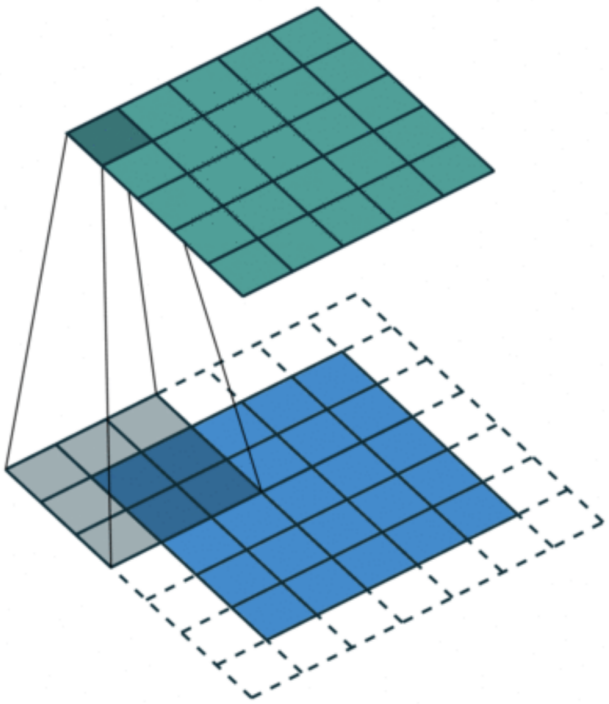


Figure 4: convolution kernel,
more info: Wikipedia: Kernel (image processing)

3. Installation

For this project a few tools were used, most importantly TensorFlow 2.0 with built in Keras. NVIDIA CUDA is optional and only used for utilizing the NVIDIA graphics card.

3.1. CUDA

To install CUDA follow the instructions in the official nvidia cuda installation guide. `https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html`

3.2. cudnn

The libcudnn package is needed for cuda accelerated TensorFlow and needs to be downloaded from nvidia:
`https://developer.nvidia.com/cudnn`
To download it a free nvidia developer program membership is needed.

3.3. TensorFlow

Make sure pip is installed and if not get it with 'sudo apt install python3-pip' Do 'pip3 install tensorflow' and 'pip3 install tensorflow-gpu' (maybe want 'pip3 install tensorboard').

3.4. Visual Studio (Code)

Visual Studio Code is a very easy to install code editor that has extensions for most programming languages. It is open source and publicly available for unix. If using a windows machine, visual studio community is a good option and cuda is automatically added to visual studio during installation. The Python extension for visual studio community comes with a python package installer that provides even the newest TensorFlow version 2.0.

3.5. Older hardware

If the graphics card has a computing capability lower than 3.5 it is not supported by TensorFlow. However, it is possible to get it working by compiling TensorFlow with a slightly different configuration. Instructions on how to do this can be found here: `https://medium.com/@mccann.matt/compiling-tensorflow-with-cuda-3-0-support-42d8fe0bf3b5`

3.6. Eclipse

If you prefer eclipse: download eclipse (preferrably a version that already has packages for C/C++) and do the following steps:

- for PyDev follow this guide: `https://www.pydev.org/manual_101_install.html`
- for Nsight: in eclipse got to 'help'→'Install New Software' then click 'Add'. Now type in the name of the extension 'NsightEE', click on 'Archive' and select the file at '/usr/local/cuda-10.1/nsightee_plugins/com.nvidia.cuda.repo-1.0.0-SNAPSHOT.zip'
  (not needed at all)

4. Keras

The open source Keras library can be best described as an relatively easy to learn high level machine learning interface that sits on top of frameworks like TensorFlow, R, Theano or others. Since version 2.0 of TensorFlow, Keras is included by default. To get started there is an official quickstart tutorial that shows the base functionality in an example on MNIST data:
`https://www.tensorflow.org/tutorials/quickstart/beginner`

5. Building a MLP

For the creation of a custom MLP the structure has to be chosen wisely. Main points of question are the number of layers and number of nodes per layer but there are a lot more components one can tweak. Here is an (incomplete) list:

- number and size of dense layers (dense layer means fully connected layer)
- dropout layers (described on page)
- activation function (per layer)
- loss function
- optimizer function

This is what building a MLP in Tensorflow.Keras looks like inside the code:

```
# design model
inputs = keras.Input(shape=(6,))
x = keras.layers.Dense(6,activation='sigmoid',input_dim=(6,))(inputs)
x = keras.layers.Dropout(0.25)(x)
x = keras.layers.Dense(6,activation='sigmoid',input_dim=(6,))(x)
x = keras.layers.Dropout(0.25)(x)
x = keras.layers.Dense(6,activation='sigmoid',input_dim=(6,))(x)
x = keras.layers.Dropout(0.25)(x)
x = keras.layers.Dense(6,activation='sigmoid',input_dim=(6,))(x)
x = keras.layers.Dropout(0.25)(x)
x = keras.layers.Dense(6,activation='sigmoid',input_dim=(6,))(x)
x = keras.layers.Dropout(0.25)(x)
outputs = keras.layers.Dense(num_classes,activation='sigmoid',input_dim=(6,))(x)
model = keras.Model(inputs=inputs, outputs=outputs)

model.summary() # prints summary of the model to console

# compile the model
model.compile(loss='binary_crossentropy',
optimizer='adam',
metrics=['accuracy'])

```

The code snippet shows building and compiling of a MLP with input layer, output layer, 4 hidden dense layers and 5 dropout layers which will be discussed later.

The simplest way of training the model is to feed it the whole dataset of training data in the form of (n,6) and (n,1) for x and y respectively.

```
# train the model
model.fit(x_train,
y_train,
batch_size=batches_size,
epochs=epoch_num,
validation_data=(x_valid,y_valid),
callbacks=[tensorboard],
verbose=1)

```

6. Analysis of 6-Dimensional Input Data (after PCA)

The data used in the first part of the project is simulated data from the pixel detector of the BELLE 2 experiment. A principal component analysis (PCA) was performed by Stephanie Käs and the resulting parameters of the data are:

charge	min charge	seed	size	size in u	size in v

The first approach was to build a MLP for binary classification and see if it can find the differences between two different particle types / background. The sets of simulated data that were analysed here are:

background	tetra-quarks	anti-deuterons	pions

The model is trained by using equal numbers of events from two datasets and the MLP should decide if it is one or the other type.

6.1. Simulated anti-deuterons against background

The ROC-curve of anti-deuteron vs. background(figure 5).

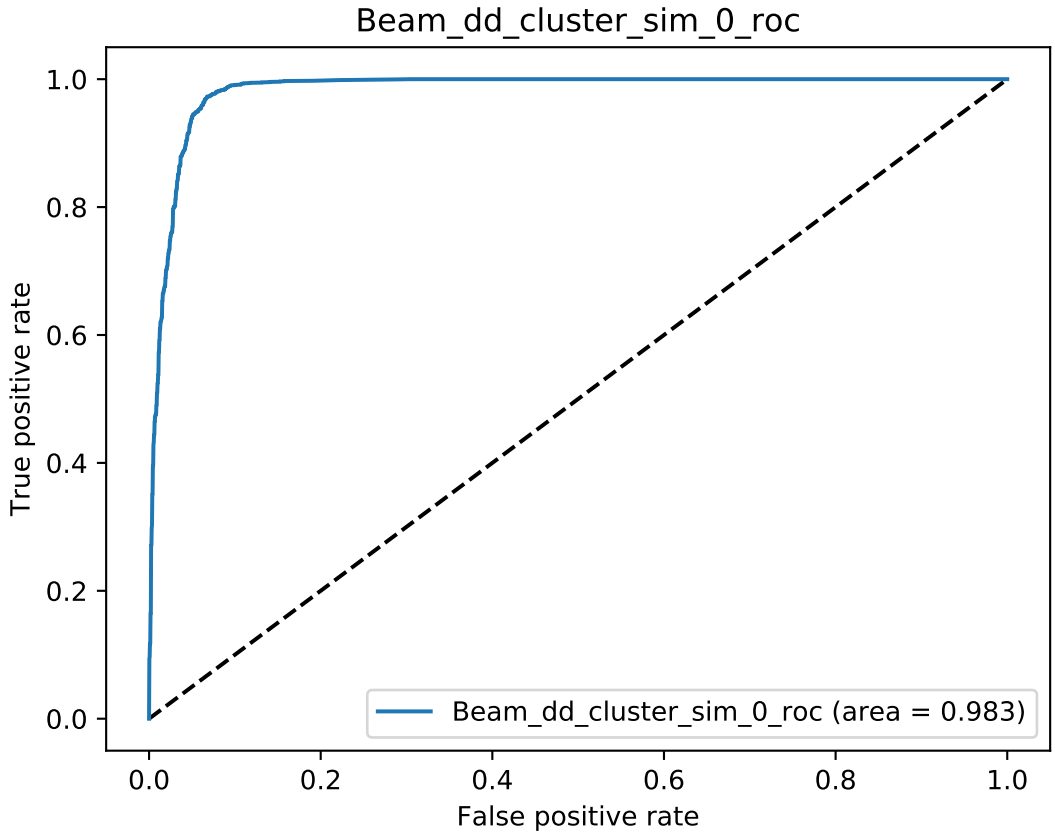


Figure 5: anti-deuteron ROC-curve

6.2. Simulated tetra-quarks against background

The area under the ROC-curve was at a maximum of 0.992 (figure 6) and could not be increased more.

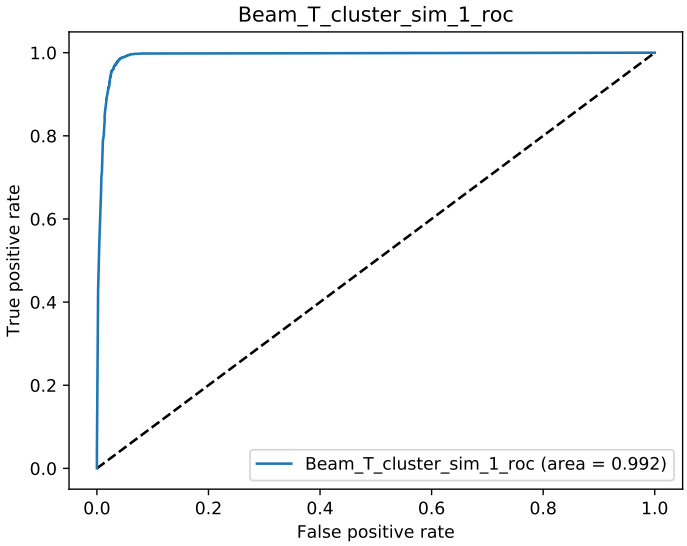


Figure 6: tetra-quark ROC-curve

6.3. Simulated pions against background

It was not possible (yet) to distinguish between event and background for pions probably because of the background made of a significant percentage of pion events. If this is the case the ROC-curve looks somewhat like in figure 7

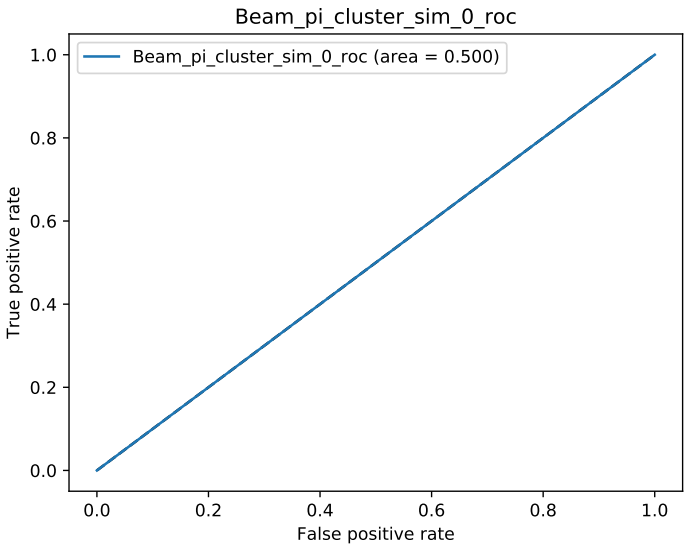


Figure 7: pion ROC-curve first model

There is still hope as one can see in figure 8.

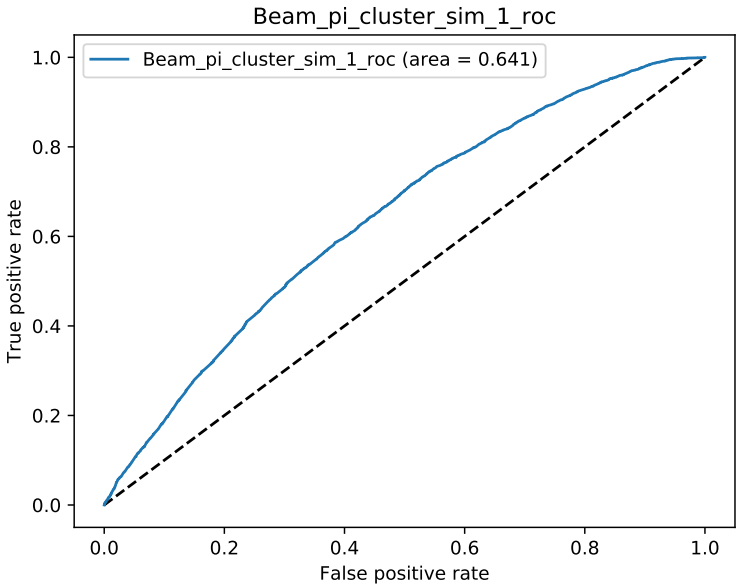


Figure 8: pion ROC-curve second model

6.4. Simulated anti-deuterons against simulated tetra-quarks

The MLP does not seem to be able to differentiate between an anti-deuteron or a tetra-quark (figure 9).

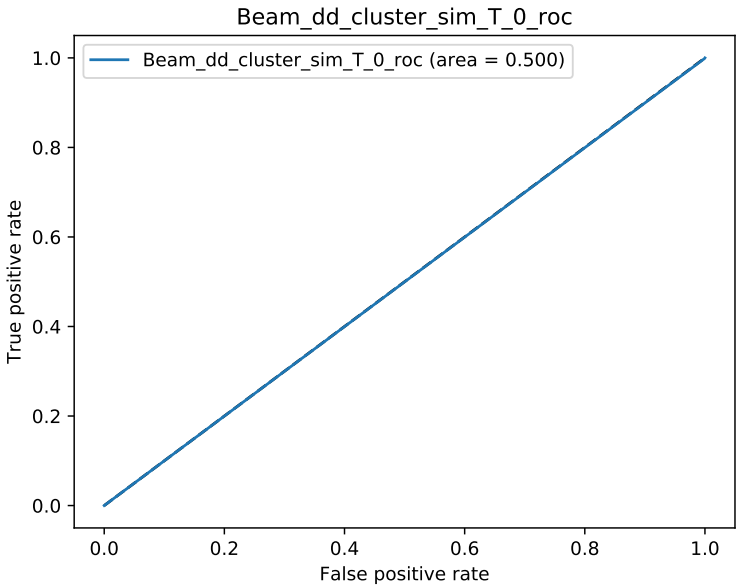


Figure 9: antideuteron / tetra-quark ROC-curve

6.5. Simulated tetra-quarks against simulated pions

It is interesting to see that distinguishing tetra-quarks from pions is even easier than from background. This is congruent with the assumption that background is to a big percentage composed of pions.

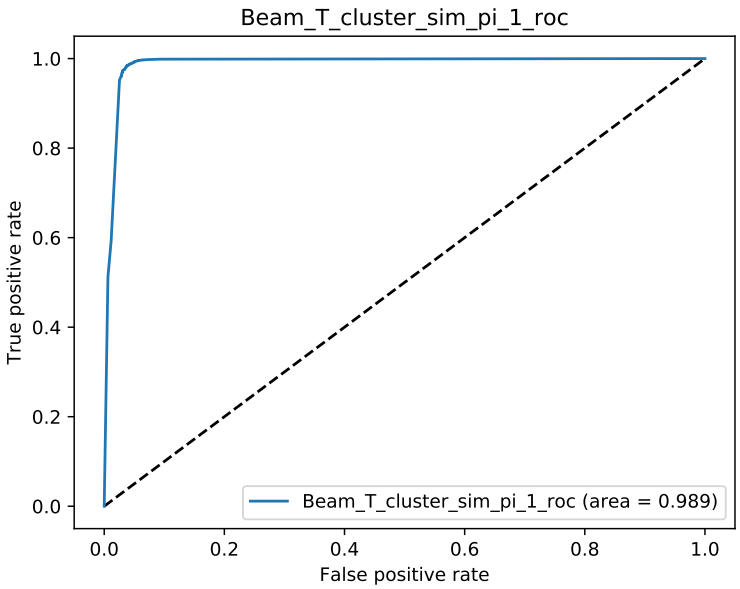


Figure 10: tetra-quark / pion ROC-curve

6.6. Conclusion 6-dim input data

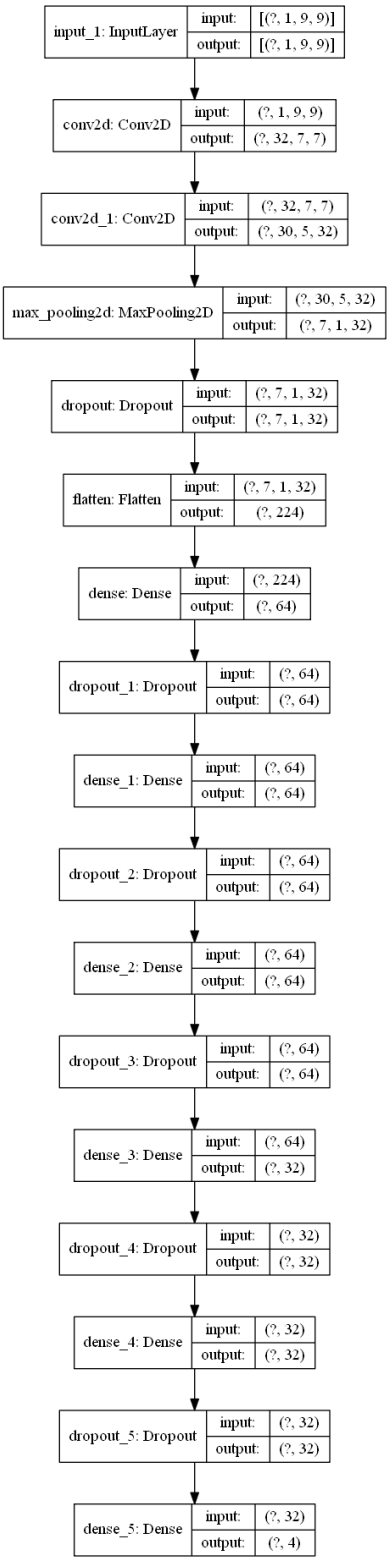
It is easy for the MLP to find differences between heavy particles like anti-deuterons or tetra-quarks and background (e.g. pions). This analysis does not show if the neural network can help with identifying exotic states when there are other (more abundant) heavy particles around like protons, neutrons etc. The neural network has not been able to differentiate between simulated tetra-quark and anti-deuteron data as seen in figure 10. Maybe if using the raw pixel data as input for training, the neural network could learn to categorize even very similar looking particles.

7. Analysis of 9 x 9 pixel data

In this dataset there are “images” of events on the pixel-detector with size of 9 x 9 pixels. The dataset has data for following simulated particle types:

background	deuterons	monopoles	pions	protons

Since the data is in the form of 2-dimensional arrays (2D-image) a convolutional neural network is probably the best way to go. On the right is the chosen CNN that utilizes two convolution layers and a pooling layer together with five hidden dense layers. When taking a look into the dataset it looks like all particles are looking alike.



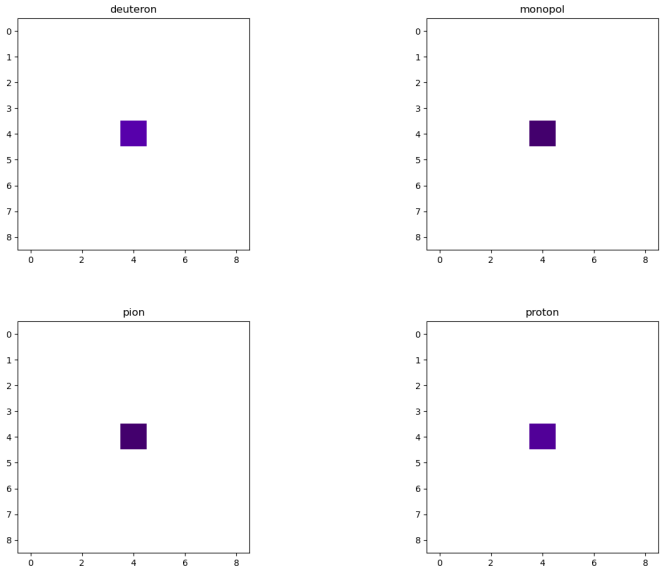


Figure 11: different particles, very similar looking

After looking for a while it looks like background is always a very strong signal and very easy to differentiate from the others by just looking at it. It is also noticeable that monopoles tend to create larger, very elliptical traces more frequently. See 13 for a colour-map.

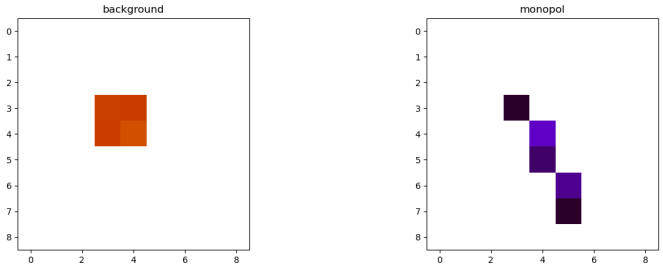


Figure 12: typical background (not explainable) and typical monopole

Figure 13: colour-map

7.1. Simulated monopoles against other particles and background

In this analysis the approach was to let the neural network categorise all particle types first.

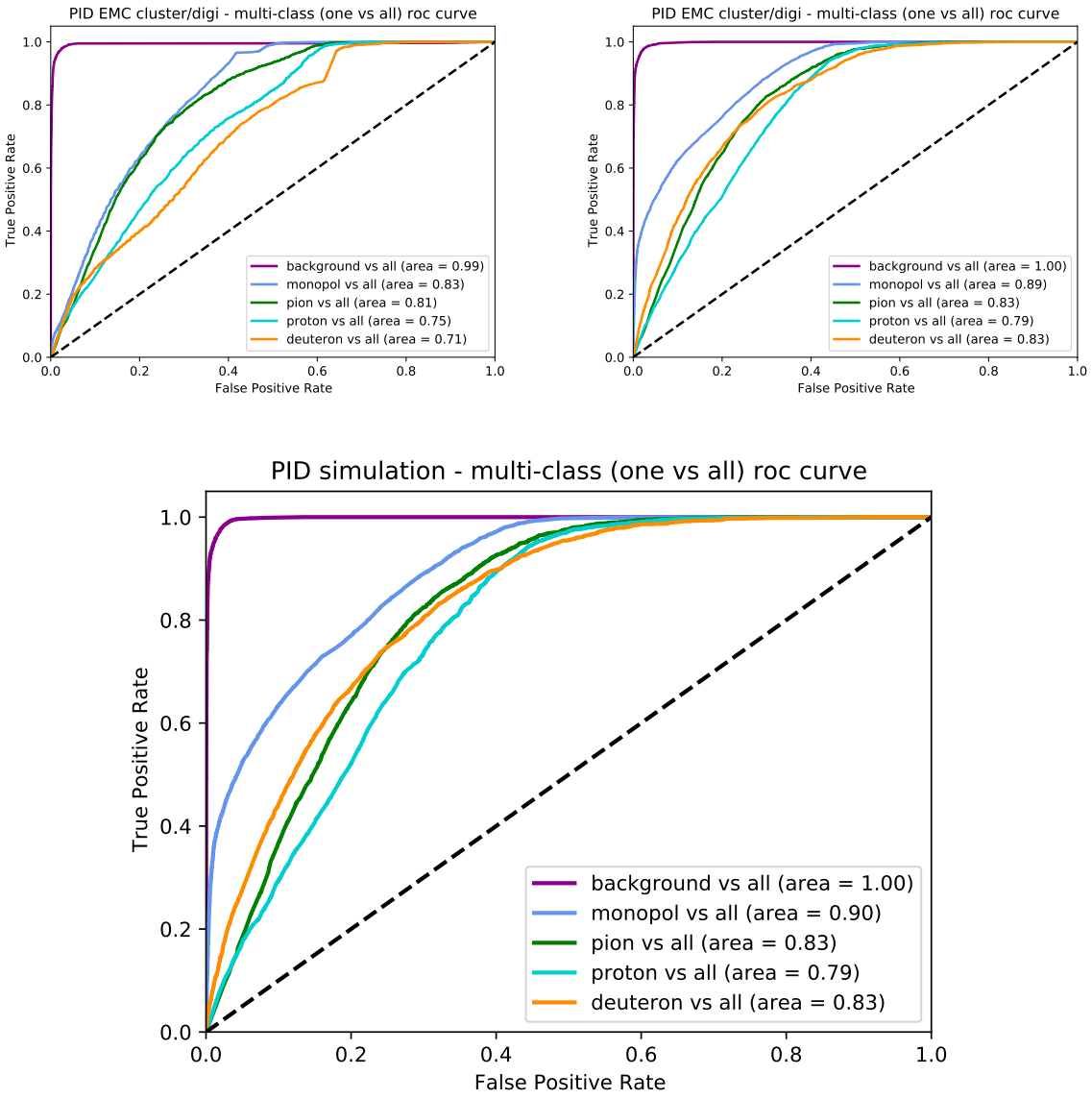


Figure 14: ROC-curve of CNN (all categories) after - 10 epochs (top-left), 100 epochs (top-right), 2000 epochs (bottom)

This shows exactly what was already seen in the images in 12: background seems to be very different to the other categories and can be identified easily by the CNN. Next step the CNN should categorise between all but the background category.

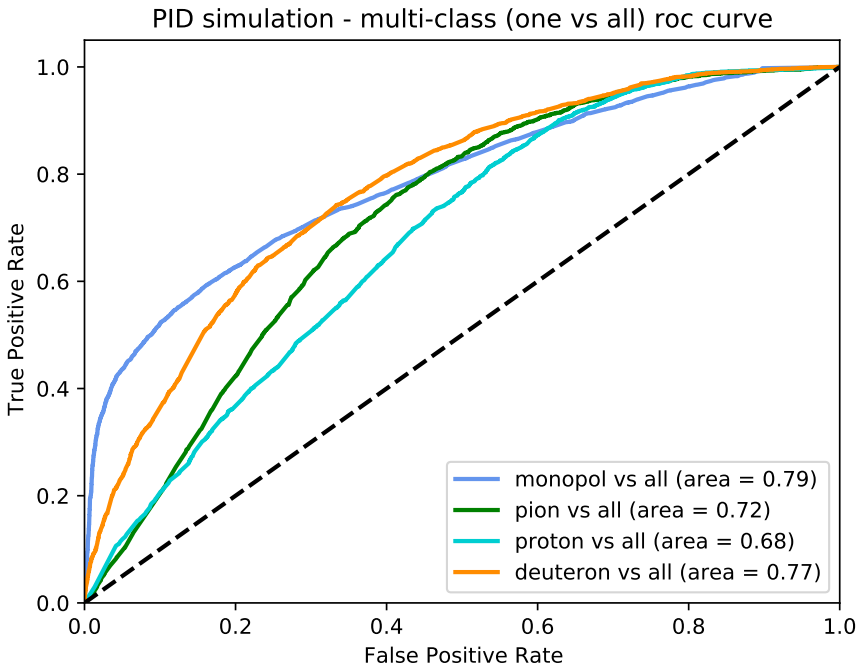


Figure 15: ROC-curve of (all but background category) CNN after 200 epochs

As shown in 15 from the simulated data used in the dataset this neural network could be used to accurately (with 99 % certainty) identify every third monopole. The question here is if the data in this dataset portrays the real world well (background seems very unrealistic). Since there are only four different particle types in this dataset it seems very unlikely that this CNN could find monopoles in real world data right now.

8. Performance improvements on GPU

On small neural networks computing on the GPU does not yield much performance improvements due to limited options for parallelisation (if there are a lot of nodes it can be parallelized much better). But also the overhead introduced by copying the data to and from the GPU is the main limiting factor for training on the GPU. To reduce this overhead a big batch size can be used but there is no fix value for this because it heavily depends on the number of input nodes and the size of the neural network. In general a batch size as big as possible without exceeding memory limits should be used. For the example MLP and dataset used here it made a big difference in computation speed and GPU utilization if the batch size was 8192 instead of 32. Following a few performance tests: Test network as shown in figure 16 was used for this test because the CNN from section 7 created an error on the CPU (probably because of some differences in libraries used).

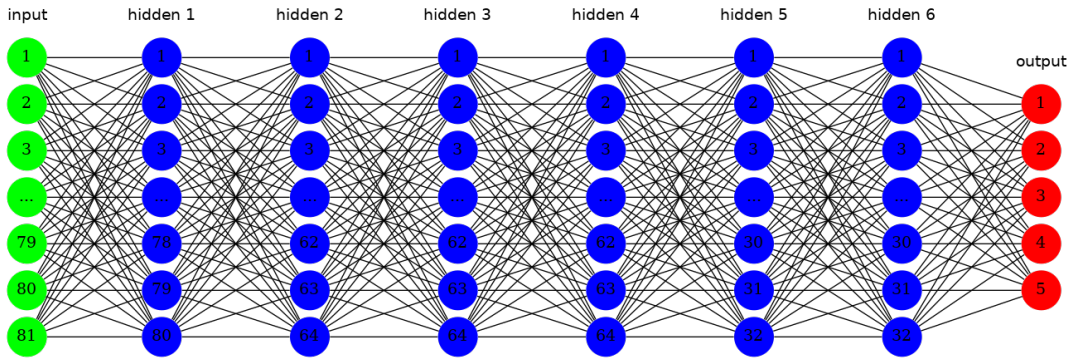


Figure 16: simple MLP with fully connected layers

The dataset from section 7 was used to train the MLP. The time it took to train a certain number of epochs with different batch sizes was measured on CPU and GPU and used as benchmark. The machine is a windows 10 on an Intel i7 6800K quad-core CPU running 4.4GHz with a NVIDIA GTX1080 GPU.

device	epochs	batch-size	duration [s]	time / epoch [s]	utilization [%]
CPU	200	40960	138.30	0.69	96
CPU	200	12288	159.45	0.80	91
CPU	200	8192	166.02	0.83	90
CPU	200	4096	185.76	0.93	85
CPU	50	2048	49.65	0.99	74
CPU	50	1024	49.49	0.99	66
CPU	50	512	66.44	1.33	56
CPU	50	256	95.91	1.92	45
CPU	50	128	168.91	3.38	38
CPU	50	32	451.51	9.03	40
GPU	200	40960	25.72	0.13	41
GPU	200	20480	28.44	0.14	31
GPU	200	12288	32.00	0.16	21
GPU	200	8192	39.80	0.20	20
GPU	200	4096	52.48	0.26	12
GPU	200	2048	81.02	0.41	8
GPU	50	1024	39.95	0.80	7
GPU	50	512	71.51	1.43	6
GPU	50	256	135.06	2.70	5
GPU	50	128	276.46	5.53	6
GPU	50	32	1211.57	24.23	5

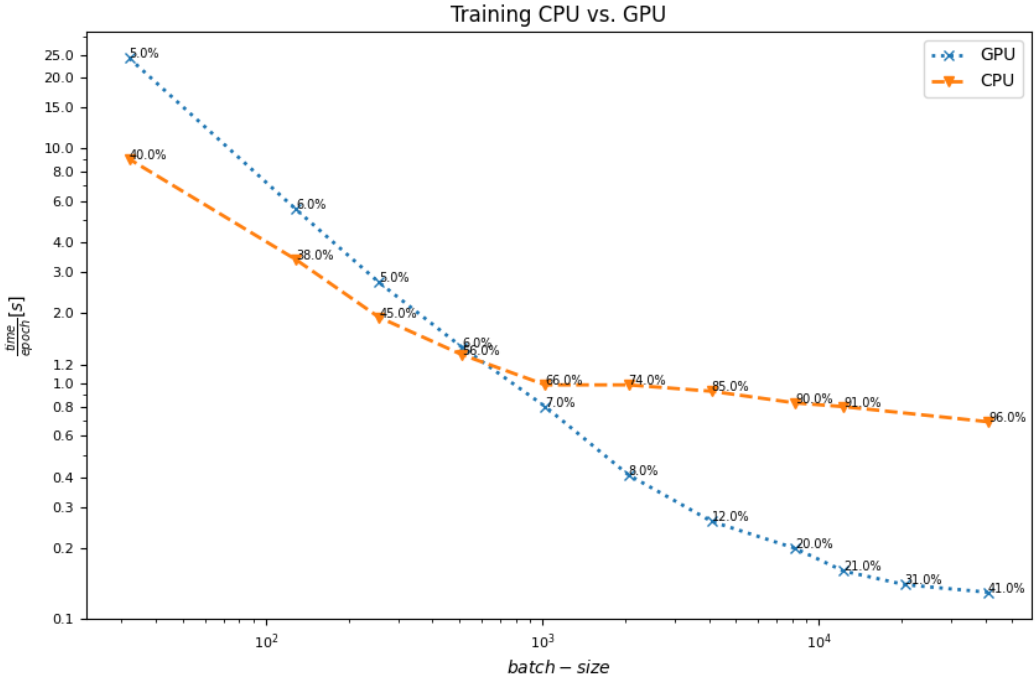


Figure 17: Training speed CPU vs. GPU (with utilization percentage)

What is interesting: the GPU utilization only jumped up when the batch-size increased. For the convolutional neural network from the Analysis of 9 x 9 pixel data it even hit 70 % load. This means it is a lot more efficient to train with a high batch-size and training on the GPU can yield performance increase of more than a factor of 4. Though it has to be thought about the fact that a bigger batch size also leads to a much slower convergence of the training. This can be seen here:

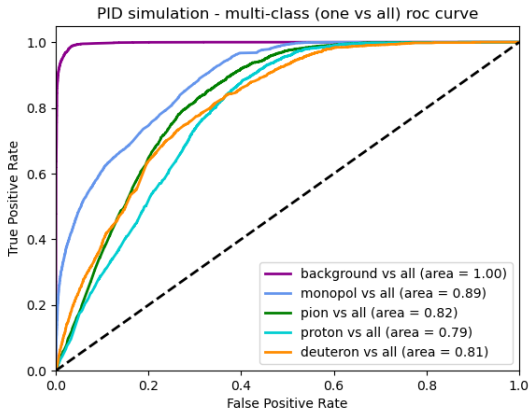


Figure 18: GPU batch-size=8192

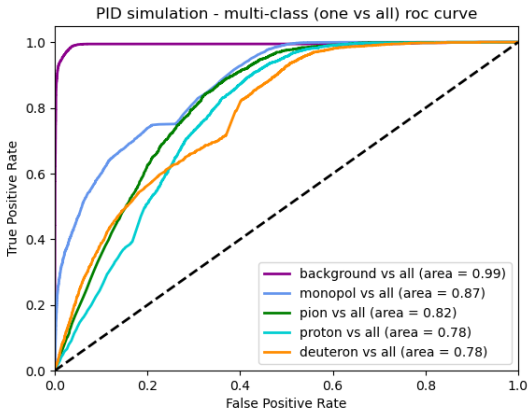


Figure 19: GPU batch-size=40960

Even though both runs were done for 200 epochs the run with the bigger batch-size looks much less complete than the other one. There is always a trade-off between batch-size and number of epochs needed. This results in a sweet spot in batch-size that is unique to the neural network, dataset and the hardware used which will give the best performance. This is one example for the importance of getting a good feeling for the parameters one can tweak when training neural networks.

9. Compare CNN with conventional MLP

When comparing figures 20 and 21 it is noticeable that the ROC-curves look very similar. This might indicate that the type of neural network used is not really important when used on this dataset. If trained long enough both neural networks were able to generate more or less the same results and the conclusion is that there are some dataset specific features that define a maximum differentiation of categories. This would then mean a larger and more diverse dataset is needed for training to improve neural network performance or maybe there is no chance for much improvement. There could also be a better designed neural network that could give improvements even though a variety of different small neural networks were already tested. For example a ResNet could be trained for this task but the current understanding is that this particular problem is not so big (9 x 9 pixel is very small and there are only 5 output categories) as that a ResNet would make much sense.

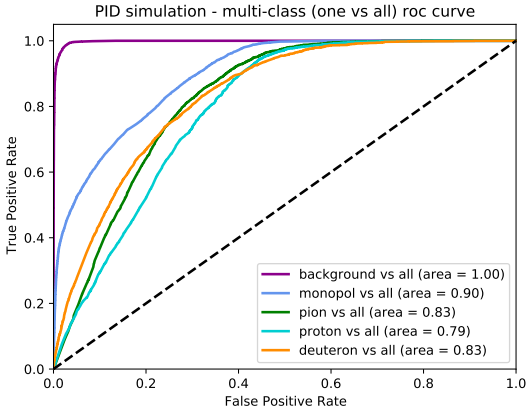


Figure 20: CNN from section 7 after 2000 epochs

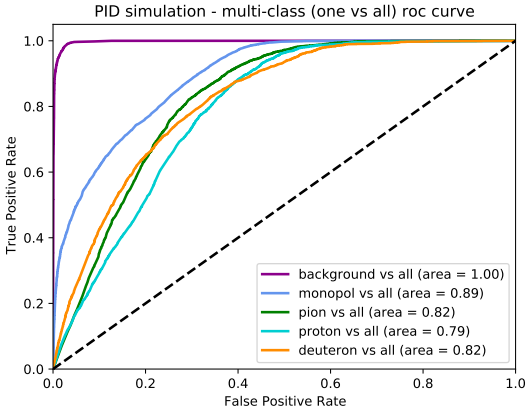


Figure 21: MLP from figure 16 after 2000 epochs

10. Conclusion

It was possible to train different types of neural networks (simple MLP and CNN) on simulated pixel data from the Belle 2 pixel-detector. In each part of the analysis it was very easy for the neural networks to differentiate between an exotic particle and background but it is not sure how good this would work in a real world scenario since there is just not enough variety in the dataset to cover all possible particles. For example if a tetra-quark looks very similar to some other heavy particle it might not be possible for the neural network to differentiate between them. There might be a big potential for neural networks to act as filters before the real analysis though.

Furthermore it is really weird that the background in the dataset in section 7 looks so different to all the other categories. Usually one would expect a mix between a lot of different particles in the background but it looks like all background events are larger than any pion, proton or deuteron events. It should be checked if the simulated data makes sense in this particular case.

It could not really been proved that raw pixel-data as input works better than the 6-dimensional input from the principal component analysis but in theory it should be a bit better.

The largest information gain from this analysis was the training speed performance graph in figure 17 which shows exactly when and how the GPU can be used to accelerate training of neural networks. For simple MLPs the performance on GPU was about a factor of 4 greater than on CPU, only utilizing about 41 % of the GPU. For convolutional neural networks like in section 7 the GPU does give an even greater performance improvement, utilizing about 70 % of the GPU. It is very important though to set the batch-size to a large enough but not too large value which depends heavily on the shape of input-data and overall structure of the neural network.

In the future it would be very nice to use a larger and much more diverse dataset to train the CNN even better.

A. Appendix

The whole code used.

A.1. Code for 6 dimensional input

Data preprocessing: Mix data from files and in 1:1 data to background ratio.

```
from __future__ import print_function
import time
import sys, getopt
import numpy as np
import pandas as pd

def mem_usage(pandas_obj):
    if isinstance(pandas_obj,pd.DataFrame):
        usage_b = pandas_obj.memory_usage(deep=True).sum()
    else: # we assume if not a df it's a series
        usage_b = pandas_obj.memory_usage(deep=True)
    usage_mb = usage_b / 1024 ** 2 # convert bytes to megabytes
    return "{:03.2f} MB".format(usage_mb)

def write_to_hdf5(path,pandas_obj,batch_size=32):
    store = pd.HDFStore(path,mode='w')
    number_batches = len(pandas_obj)/batch_size
    print('len/batch_size: ', number_batches)
    number_batches = int(number_batches)
    print('len/batch_size: ', number_batches)
    if isinstance(pandas_obj,pd.DataFrame):
        for i in range(int(number_batches)-1):
            store['id_{0}'.format(i)] = pd.DataFrame(pandas_obj).loc[i*32:(i+1)*32-1]
        print(len(pandas_obj)-32*number_batches)
        #print(pandas_obj[32*number_batches:len(pandas_obj)])
    store.close()

def write_to_directory(path,pandas_obj,batch_size=128):
    number_batches = len(pandas_obj)/batch_size
    print('len/batch_size: ', number_batches)
    number_batches = int(number_batches)
    print('len/batch_size: ', number_batches)
    if isinstance(pandas_obj,pd.DataFrame):
    for i in range(int(number_batches)-1):
        store['id_{0}'.format(i)] = pd.DataFrame(pandas_obj).loc[i*32:(i+1)*32-1]
    print(len(pandas_obj)-32*number_batches)

seed = 42
directory = 'C:\\Users\\Marvin\\Documents\\Github\\simulated_data\\' # windows
out_directory = directory+'preprocessed_data\\'
#print('inputfile=',in_filename, 'outputfile=', out_filename)
#bg_file = 'Beam_BG_cluster_sim_103429.txt'
bg_file = 'Beam_BG_cluster_sim.txt'
sig_files = ['Beam_dd_cluster_sim.txt','Beam_pi_cluster_sim.txt','Beam_T_cluster_sim.
    txt']
columnnames = ['total_cluster_charge','cluster_seed_charge','cluster_minimum_charge','
    total_cluster_size','cluster_size_u','cluster_size_v']

start_time = time.time()
for num in range(len(sig_files)):
    df = pd.read_csv(directory+bg_file, sep=' ',
                                names=columnnames, header=None,
                                usecols=[0,1,2,3,4,5], dtype='uint16')
```

```
    sig = np.uint16(0)
    df.insert(loc=6,column='sig',value=sig)

    data = pd.read_csv(directory+sig_files[num], sep=' ',
                                names=columnnames, header=None,
                                usecols=[0,1,2,3,4,5], dtype='uint16')
    sig = np.uint16(1)
    data.insert(loc=6,column='sig',value=sig)
    df = pd.concat([df,data],ignore_index=True,copy=False)
    df = df.sample(frac=1,random_state=seed).reset_index(drop=True)
    #df = df.reindex(np.random.RandomState(seed=seed).permutation(df.index)).
    reset_index(drop=True)
    write_to_directory(directoy+sig_files[num]+'_bg',df)
    #df.to_parquet(directoy+'dataframe_'+sig_files[num]+'_bg.parquet')
    #write_to_hdf5(out_directory+'data_'+sig_files[num]+'_bg.hdf5',df)

print("--- %s seconds ---" % (time.time()-start_time))
#dataframe.to_parquet(directory+'dataframe_t_bg.parquet')
#dataframe.to_sql(name='ALL', con=conn, if_exists='replace', index=False)

#for num in range(len(sig_files)):
#    for chunk in pd.read_csv(directory+sig_files[num], sep=' ',
#                             names=columnnames, header=None,
#                             usecols=[0,1,2,3,4,5],
#                             chunksize=2000000, dtype='uint16'):
#        max = chunk.max(0).to_numpy()
#        min = chunk.min(0).to_numpy()
#        print(max)
#        print(min)
#        for i in range(len(max_values)):
#            if max_values[i] < max[i]:
#                max_values[i] = max[i]
#                min_values[i] = min[i]
#print(max_values)
#print(min_values)
#exit(0)
```

Split the data into training, validation and evaluation sets for each particle-background combination.

```
from __future__ import print_function
import sys, getopt
import numpy as np
import pandas as pd

def mem_usage(pandas_obj):
    if isinstance(pandas_obj,pd.DataFrame):
        usage_b = pandas_obj.memory_usage(deep=True).sum()
    else: # we assume if not a df it's a series
        usage_b = pandas_obj.memory_usage(deep=True)
    usage_mb = usage_b / 1024 ** 2 # convert bytes to megabytes
    return "{:03.2f} MB".format(usage_mb)
seed = 42
directory = 'C:\\Users\\Marvin\\Documents\\Github\\simulated_data\\' # windows
dataframe_name = 'dataframe_t_bg_light.parquet'
dataframe = pd.read_parquet(directory+dataframe_name)
print(dataframe)
print('mem_usage: ',mem_usage(dataframe))
dataframe = dataframe.reindex(np.random.RandomState(seed=seed).permutation(dataframe.
    index))
print(dataframe)
print('mem_usage: ',mem_usage(dataframe))
```

```
dataframe.reset_index(drop=True, inplace=True)
print(dataframe)
print('mem_usage: ',mem_usage(dataframe))
dataframe.loc[0:np.floor(0.7*dataframe.shape[0])].to_parquet(directory+'
    training_data_t_bg_light.parquet')
print('training saved')
validation = dataframe.loc[np.floor(0.7*dataframe.shape[0])+1:np.floor(0.9*dataframe.
    shape[0])]
validation.reset_index(drop=True, inplace=True)
validation.to_parquet(directory+'validation_data_t_bg_light.parquet')
print('validation saved')
evaluation = dataframe.loc[np.floor(0.9*dataframe.shape[0])+1:dataframe.shape[0]]
evaluation.reset_index(drop=True, inplace=True)
evaluation.to_parquet(directory+'evaluation_data_t_bg_light.parquet')
print('evaluation saved')
```

The actual MLP program

```
from __future__ import print_function
import sys, getopt
import os
from time import time
import tensorflow as tf
import numpy as np
from tensorflow import keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras import layers
from tensorflow.keras import utils
from tensorflow.keras.callbacks import TensorBoard
from datagenerator import DataGenerator
import pandas as pd

def main(argv):
    global in_filename,out_filename
    out_filename='C:\\Users\\Marvin\\Documents\\Github\\simulated_data\\model.h5'
    in_filename='C:\\Users\\Marvin\\Documents\\Github\\simulated_data\\out'
    try:
        opts, args = getopt.getopt(argv,"hi:o:",["in=","out="])
    except getopt.GetoptError:
        print('in=<filename> or -i <filename>  : declare input file')
        print('out=<filename> or -o <filename>  : declare output file')
        sys.exit(2)
    for opt, arg in opts:
        if opt == '-h':
            print('in=<filename> or -i <filename>  : declare input file')
            print('out=<filename> or -o <filename>  : declare output file')
            sys.exit()
        elif opt in ("-i", "--in"):
            in_filenames = opt
            #in_filename = arg
        elif opt in ("-o", "--out"):
            out_filename = arg

# start of program
if __name__ == "__main__":
   main(sys.argv[1:])


dir_path = os.path.dirname(os.path.realpath(__file__))
print(dir_path)

batches_size = 32
```

```
num_classes = 1
epoch_num = 100
activ_func1 = 'sigmoid'
activ_func2 = 'sigmoid'

# load data
#x_train = np.load(in_filename+"_x_train.npy")
#y_train = np.load(in_filename+"_y_train_2.npy")
#num_classes = y_train.shape[1]
number_sig = 0
number_bg = 0
directory = 'C:\\Users\\Marvin\\Documents\\Github\\simulated_data\\' # windows
maxima = np.full((7,),0.)
#training = pd.read_parquet(directory+'training_data_t_bg.parquet')
training = pd.read_parquet(directory+'dataframe.parquet')
print(training.shape)
exit(0)
number_sig += len(training['sig'].eq(1))
number_bg += len(training['sig'].eq(0))
print('signals: ',number_sig,' background: ',number_bg,' signal/bg: ',number_sig/
    number_bg)
i = 0
for index in training.columns:
    max = training[index].max()
    print('validation ,',index,' max: ',max)
    if max>maxima[i]:
        maxima[i] = max
    i = i+1
y_train = training['sig'].to_numpy(dtype=float ,copy=True)
print('copy y_train done...')
print(y_train.shape)
x_train = training.drop(columns='sig').to_numpy(dtype=float ,copy=True)
print('copy x_train done...')
print(x_train.shape)
del training
validation = pd.read_parquet(directory+'validation_data_t_bg_light.parquet')
i = 0
for index in validation.columns:
    max = validation[index].max()
    print('validation ,',index,' max: ',max)
    if max>maxima[i]:
        maxima[i] = max
    i = i+1

y_valid = validation['sig'].to_numpy(dtype=float ,copy=True)
print('copy y_valid done...')
print(y_valid.shape)
x_valid = validation.drop(columns='sig').to_numpy(dtype=float ,copy=True)
print('copy x_valid done...')
print(x_valid.shape)
del validation

evaluation = pd.read_parquet(directory+'evaluation_data_t_bg_light.parquet')
i = 0
for index in evaluation.columns:
    max = evaluation[index].max()
    print('validation ,',index,' max: ',max)
    if max>maxima[i]:
        maxima[i] = max
    i = i+1
y_eval = evaluation['sig'].to_numpy(dtype=float ,copy=True)
```

```
105 print('copy y_eval done...')
106 print(y_eval.shape)
107 x_eval = evaluation.drop(columns='sig').to_numpy(dtype=float,copy=True)
108 print('copy x_eval done...')
109 print(x_eval.shape)
110 del evaluation
111
112 print('maxima: ',maxima)
113 print(x_eval)
114 for i in range(len(maxima)-1):
115     x_train[:,i] = x_train[:,i]/maxima[i]
116     x_valid[:,i] = x_valid[:,i]/maxima[i]
117     x_eval[:,i] = x_eval[:,i]/maxima[i]
118
119 print(x_eval)
120 # setup datagenerator
121 #+training_generator = DataGenerator(directory=directory, dataframe_name='
        training_data_light.parquet')
122 #validation_generator = DataGenerator(directory=directory, dataframe_name='
        validation_data_light.parquet')
123 #evaluation_generator = DataGenerator(directory=directory, dataframe_name='
        evaluation_data_light.parquet')
124
125 # design model
126 inputs = keras.Input(shape=(6,))
127 x = keras.layers.Dense(6,activation=activ_func1,input_dim=(6,))(inputs)
128 x = keras.layers.Dropout(0.25)(x)
129 x = keras.layers.Dense(6,activation=activ_func1,input_dim=(6,))(x)
130 x = keras.layers.Dropout(0.25)(x)
131 x = keras.layers.Dense(6,activation=activ_func1,input_dim=(6,))(x)
132 x = keras.layers.Dropout(0.25)(x)
133 x = keras.layers.Dense(6,activation=activ_func1,input_dim=(6,))(x)
134 x = keras.layers.Dropout(0.25)(x)
135 x = keras.layers.Dense(4,activation=activ_func1,input_dim=(6,))(x)
136 x = keras.layers.Dropout(0.25)(x)
137 x = keras.layers.Dense(4,activation=activ_func1,input_dim=(4,))(x)
138 x = keras.layers.Dropout(0.25)(x)
139 outputs = keras.layers.Dense(num_classes,activation=activ_func2,input_dim=(4,))(x)
140 model = keras.Model(inputs=inputs, outputs=outputs)
141 model.summary()
142 keras.utils.plot_model(model, directory+'my_model.png', show_shapes=True)
143
144 # compile the model
145 if num_classes>1:
146     model.compile(loss='categorical_crossentropy',
147                   optimizer='RMSprop',
148                   metrics=['accuracy'])
149 else:
150     model.compile(loss='binary_crossentropy',
151                   optimizer='RMSprop',
152                   metrics=['accuracy'])
153
154 #tensorboard = TensorBoard(log_dir=dir_path+"/logs/{}".format(time()))#linux
155 tensorboard = TensorBoard(log_dir=directory+"\\logs\\{}".format(time()))
156
157 # train the model
158 #model.fit_generator(generator=training_generator,
159 #                    validation_data=validation_generator)#,
160 #                    #use_multiprocessing=True,
161 #                    #workers=3)
162
```

```
# train the model
model.fit(x_train,
          y_train,
          batch_size=batches_size,
          epochs=epoch_num,
          validation_data=(x_valid,y_valid),
          #validation_split=0.1,
          callbacks=[tensorboard],
          verbose=1)

# evaluate
score = model.evaluate(x_eval, y_eval, verbose=0)
#score = model.evaluation_generator(generator=evaluation_generator)#,
#                                   #use_multiprocessing=True,
#                                   #workers=3)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
#model.save(dir_path+'/model.h5')#linux
model.save(directory+"model.h5")
```

A.2. Code for 9x9 pixel data input

```
from __future__ import print_function
from time import time
import subprocess
import sys, getopt
from io import StringIO
import os
import numpy as np
import random
from pathlib import Path
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras import backend
from tensorflow.keras import layers
from tensorflow.keras import utils
from tensorflow.keras.callbacks import TensorBoard
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from matplotlib import pyplot as plt
from itertools import cycle
from tqdm import tqdm

output_categories = ['background','monopol','pion','proton', 'deuteron']

filenames = ["monopol_bg_9x9.txt","pions.txt","protons.txt","deuterons.txt"]

gpu_activated = True
if not gpu_activated:
    os.environ["CUDA_VISIBLE_DEVICES"] = "-1" # used to deactivate the gpu

directory = 'C:\\Users\\Marvin\\Documents\\Github\\simulated_data\\pixeldata\\' #
    windows
outputdirectory = directory+'processed_pixel_data' +'\\'
#directory = '/home/user/vertiefungsmodul/simulated_data/'
#filenames = ['Beam_T_cluster_sim.txt','Beam_dd_cluster_sim.txt','Beam_pi_cluster_sim.
    txt','Beam_BG_cluster_sim.txt']
#bgfilename = [s for s in filenames if "BG" in s][0] # get first element of list that
    contains only filenames with 'BG' in it
```

```
batches_size = 256
epoch_num = 50
load_data = 'data'
load_model = ''
evaluate_only = False
save_data = ''
filename = 'pid'
output_caption = 'PID simulation'
session_id = 0 # does automatically change if files still exist with this id

if load_data != '':
    load_data = outputdirectory + load_data + '.npy'
if save_data != '':
    save_data = outputdirectory + save_data + '.npy'
if (load_model != ''):
    load_model = outputdirectory + 'model_' + load_model + '.h5'
seed = 42

#outputdirectory = directory+'preprocessed_data/'+filename+'/'
Path(outputdirectory).mkdir(parents=True, exist_ok=True)

start_time = time()
data = []
if load_data == '':
    data = np.loadtxt(directory + filenames[0],delimiter=' ')
    data = np.delete(data,[np.array(np.where([y[0]==0. for y in data])[0])],axis=0) #
    delete all rows where first value in row is 0
    data[:,0] = np.full(len(data),0)
    for i in range(1,len(filenames)):
        more_data = np.loadtxt(directory + filenames[i],delimiter=' ')
        more_data = more_data[:,1:-3]
        more_data[:,0] = np.full(len(more_data),i)
        print(more_data.shape)
        data = np.append(data,more_data,axis=0)
    np.random.shuffle(data)
    if save_data != '':
        np.save(save_data,data,allow_pickle=True)
else:
    data = np.load(load_data,allow_pickle=True)
max_energy = data[:,1:].max()
min_energy = data[:,1:].min()
print("min max")
print(max_energy,min_energy)

x_data = data[:,1:]
x_data = (x_data-min_energy)/(max_energy-min_energy)
#x_data = x_data.reshape((-1,9,9))
#x_data = x_data.reshape((-1,1,9,9))
x_data = keras.backend.constant(x_data)

y_data = data[:,0]
y_data = keras.utils.to_categorical(y_data,num_classes=len(output_categories))

# determine the number of entries per particle type
entries_by_category = []
for i in range(len(output_categories)):
    n = 0
    if len(output_categories) > 1:
        n = len([elem for elem in y_data if elem[i]==1.])
        print(output_categories[i], ": ", n)
        entries_by_category.append(n)
```

```
96      else:
97          n = len([elem for elem in y_data if elem==1.])
98          print(output_categories[i], ": ",n)
99          print("background: ",len(y_data)-n)
100         entries_by_category.append(n)
101
102 thres0 = int(np.floor(0.7*len(x_data)))
103 thres1 = int(np.floor(0.9*len(x_data)))
104 thres2 = len(x_data)
105 x_train = x_data[0:thres0]
106 y_train = y_data[0:thres0]
107 x_val = x_data[thres0:thres1]
108 y_val = y_data[thres0:thres1]
109 x_eval = x_data[thres1:thres2]
110 y_eval = y_data[thres1:thres2]
111
112 #np.savetxt(outputdirectory+'x_train',x_train,delimiter=' ')
113 #np.savetxt(outputdirectory+'y_train',y_train,delimiter=' ')
114 #np.savetxt(outputdirectory+'x_val',x_val,delimiter=' ')
115 #np.savetxt(outputdirectory+'y_val',y_val,delimiter=' ')
116
117 print('x_train: ',x_train.shape)
118 print('y_train: ',y_train.shape)
119 print('x_val: ',x_val.shape)
120 print('y_val: ',y_val.shape)
121 print('x_eval: ',x_eval.shape)
122 print('y_eval: ',y_eval.shape)
123 # design model
124 if (len(output_categories)>1):
125     activ_method_0 = tf.keras.activations.swish #tf.keras.layers.LeakyReLU()
126     activ_method_1 = 'softmax'
127 else:
128     activ_method_0 = tf.keras.activations.swish # 'relu' #tf.keras.layers.LeakyReLU()
129     activ_method_1 = 'sigmoid'
130
131 model = []
132 if load_model == '':
133     inputs = keras.layers.Input(shape=x_train[0].shape)
134     #x = keras.layers.Conv2D(filters=32,kernel_size=(3,3),activation=activ_method_0,
        input_shape=x_train[0].shape,data_format = 'channels_first')(inputs)
135     #x = keras.layers.Conv2D(filters=32,kernel_size=(3,3),activation=activ_method_0)(x)
136     #x = keras.layers.MaxPooling2D(pool_size=(4,4))(x)
137     #x = keras.layers.Dropout(0.25)(x)
138     #x = keras.layers.Flatten()(x)
139     x = keras.layers.Dense(80, activation=activ_method_0)(inputs)
140     x = keras.layers.Dropout(0.5)(x)
141     x = keras.layers.Dense(64, activation=activ_method_0)(x)
142     x = keras.layers.Dropout(0.5)(x)
143     x = keras.layers.Dense(64, activation=activ_method_0)(x)
144     x = keras.layers.Dropout(0.5)(x)
145     x = keras.layers.Dense(64, activation=activ_method_0)(x)
146     x = keras.layers.Dropout(0.5)(x)
147     x = keras.layers.Dense(32, activation=activ_method_0)(x)
148     x = keras.layers.Dropout(0.25)(x)
149     x = keras.layers.Dense(32, activation=activ_method_0)(x)
150     x = keras.layers.Dropout(0.25)(x)
151     outputs = keras.layers.Dense(len(output_categories), activation=activ_method_1)(x)
152     model = keras.Model(inputs=inputs, outputs=outputs)
153
154     model.summary()
155
```

```
    # compile the model:
    model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
else:
    model = keras.models.load_model(load_model)
model_file = outputdirectory+'model_structure_{}.png'.format(session_id)
while (Path(model_file).is_file()):
    session_id = session_id +1
    model_file = outputdirectory+'model_structure_{}.png'.format(session_id)
keras.utils.plot_model(model, model_file, show_shapes=True)

# train the model
start_time = time()
if not evaluate_only:
    model.fit(x_train,
              y_train,
              batch_size=batches_size,
              epochs=epoch_num,
              #validation_split=0.1,
              validation_data = (x_val, y_val),
              verbose=1)


# evaluate
score = model.evaluate(x_eval, y_eval, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
model.save(outputdirectory+'model_{}.h5'.format(session_id))
logfile = outputdirectory+filename+'_logfile_{}.txt'.format(session_id)
if (not Path(logfile).is_file()):
    f = open(logfile, "a")
    f.write("log_time_now;epochs;batches_size;seconds;gpu_activated;evaluation_loss;
    evaluation_accuracy;area_roc\n")
else:
    f = open(logfile, "a")

n_classes = len(output_categories)
plt.cla()
if (len(output_categories)>1):
##### create plot of one vs all roc curves
    # Plot linewidth.
    lw = 2

    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    y_score = model.predict(x_eval)
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_eval[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Compute micro-average ROC curve and ROC area
    fpr["micro"], tpr["micro"], _ = roc_curve(y_eval.ravel(), y_score.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

    # Compute macro-average ROC curve and ROC area

    # First aggregate all false positive rates
    all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
```

```
216
217     # Then interpolate all ROC curves at this points
218     mean_tpr = np.zeros_like(all_fpr)
219     for i in range(n_classes):
220         mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
221
222     # Finally average it and compute AUC
223     mean_tpr /= n_classes
224
225     fpr["macro"] = all_fpr
226     tpr["macro"] = mean_tpr
227     roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])
228
229     # Plot all ROC curves
230     plt.figure(1)
231     #plt.plot(fpr["micro"], tpr["micro"],
232     #         label='micro-average ROC curve (area = {0:0.2f})'
233     #               ''.format(roc_auc["micro"]),
234     #         color='peru', linestyle=':', linewidth=4)
235
236     #plt.plot(fpr["macro"], tpr["macro"],
237     #         label='macro-average ROC curve (area = {0:0.2f})'
238     #               ''.format(roc_auc["macro"]),
239     #         color='crimson', linestyle=':', linewidth=4)
240     color_array = ['darkmagenta','cornflowerblue', 'green', 'darkturquoise', '
    darkorange', 'darkslategrey','midnightblue', 'gold', 'lime']
241     colors = cycle(color_array)
242     for i, color in zip(range(n_classes), colors):
243         plt.plot(fpr[i], tpr[i], color=color, lw=lw,
244                  label='{0} vs all (area = {1:0.2f})'
245                  ''.format(output_categories[i], roc_auc[i]))
246
247     plt.plot([0, 1], [0, 1], 'k--', lw=lw)
248     plt.xlim([0.0, 1.0])
249     plt.ylim([0.0, 1.05])
250     plt.xlabel('False Positive Rate')
251     plt.ylabel('True Positive Rate')
252     if output_caption!="":
253         plt.title(output_caption+' - multi-class (one vs all) roc curve')
254     else:
255         plt.title('multi-class (one vs all) roc curve')
256     plt.legend(loc="lower right")
257     plt.savefig(outputdirectory+filename+'_{}_roc.png'.format(session_id))
258     plt.savefig(outputdirectory+filename+'_{}_roc.pdf'.format(session_id))
259
260     # Zoom in view of the upper left corner.
261     plt.figure(2)
262     plt.xlim(0, 0.2)
263     plt.ylim(0.8, 1)
264     plt.plot(fpr["micro"], tpr["micro"],
265              label='micro-average ROC curve (area = {0:0.2f})'
266                    ''.format(roc_auc["micro"]),
267              color='deeppink', linestyle=':', linewidth=4)
268
269     plt.plot(fpr["macro"], tpr["macro"],
270              label='macro-average ROC curve (area = {0:0.2f})'
271                    ''.format(roc_auc["macro"]),
272              color='navy', linestyle=':', linewidth=4)
273
274     colors = cycle(color_array)
275     for i, color in zip(range(n_classes), colors):
```

```
276         plt.plot(fpr[i], tpr[i], color=color, lw=lw,
277                  label='{0} vs all (area = {1:0.2f})'
278                  ''.format(output_categories[i], roc_auc[i]))
279 
280     plt.plot([0, 1], [0, 1], 'k--', lw=lw)
281     plt.xlabel('False Positive Rate')
282     plt.ylabel('True Positive Rate')
283     if output_caption!="":
284         plt.title(output_caption+' - multi-class (one vs all) roc curve')
285     else:
286         plt.title('multi-class (one vs all) roc curve')
287     plt.legend(loc="lower right")
288     plt.savefig(outputdirectory+filename+'_{}_roc_zoom.png'.format(session_id))
289     plt.savefig(outputdirectory+filename+'_{}_roc_zoom.pdf'.format(session_id))
290 
291 if (len(output_categories)==1):
292     # code to create roc curve for single particle vs background
293     y_pred_keras = model.predict(x_eval).ravel()
294     fpr_keras, tpr_keras, thresholds_keras = roc_curve(y_eval, y_pred_keras) # fpr=
    false positive, tpr=true positive
295     auc_keras = auc(fpr_keras, tpr_keras)
296     plt.figure(1)
297     plt.plot([0, 1], [0, 1], 'k--')
298     if output_caption!="":
299         plt.plot(fpr_keras, tpr_keras, label=output_caption+' ROC curve (area = {:.3f})
    '.format(auc_keras))
300     else:
301         plt.plot(fpr_keras, tpr_keras, label=filename+'_roc (area = {:.3f})'.format(
    auc_keras))
302     plt.xlabel('False positive rate')
303     plt.ylabel('True positive rate')
304     if output_caption!="":
305         plt.title(output_caption+' binary classification ROC curve')
306     else:
307         plt.title(filename+'_roc')
308     plt.legend(loc='best')
309     plt.savefig(outputdirectory+filename+'_{}_single_category_roc.png'.format(
    session_id))
310     plt.savefig(outputdirectory+filename+'_{}_single_category_roc.pdf'.format(
    session_id))
311 
312 f.write("{};{};{};{};{};{};{}\n".format(time(),epoch_num,batches_size,time()-start_time
    ,gpu_activated,score[0],score[1]))
313 for i in range(len(output_categories)):
314     f.write(output_categories[i]+': {}\n'.format(entries_by_category[i]))
315 f.write('Cluster img size: {}\n'.format(x_train[0].shape))
316 f.close()
```