New Particle Detection using a Dense Neural Network

1. Introduction

We've been provided with a large data set to develop a model that can assist in predicting the existence of a new particle. The size of the data and the need for a high level of accuracy suggests a neural network with a classifier would be the best approach.

2. Data

The task at hand given to the team is to predict the existence of a new particle. A dense neural network and classification are recommended for this problem, and the goal is to maximize the prediction accuracy. A general layout of the data can be seen in Table 1.

Table 1. The raw format of the dataset before preprocessing.

```
RangeIndex: 7000000 entries, 0 to 6999999
    Data columns (total 29 columns):
           #   Column   Dtype
          ---  ------   -----
          0   # label  float64
          1   f0       float64
          2   f1       float64
          3   f2       float64
          4   f3       float64
                   ...
          24  f23      float64
          25  f24      float64
          26  f25      float64
          27  f26      float64
          28  mass     float64
          dtypes: float64(29)
          memory usage: 1.5 GB
```

The given data contains 7 million entries, and the features that describe them include 28 attributes and 1 response variable named "# label". The distribution of the response variable is observed and plotted (Fig. 1).

To aid the effort of training an efficient neural network, all 29 features including the response variable are scaled from 0 to 1 using `MinMaxScaler(feature_range=(0, 1))`.

Then, the data is split into two parts, where 80% will be used as the training set and 20% will be used as the test set.

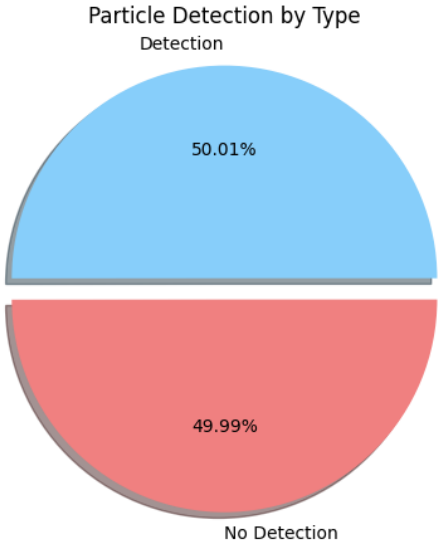

Fig. 1. The distribution of "Detection" and "No Detection" are equally similar.

3. Dense Neural Network

Other than the response variable, the other 28 features are assumed to be sequences that can be used in the `Sequential()` model Tensorflow's keras in Python.

3.1. Design

The first neural network is built to conduct a preliminary assessment. An input layer with a shape of 28 is used as the first layer, followed by a dense layer using the rectified linear unit (ReLU) activation function with 128 nodes; this is a highly preferred tool in deep learning because "deep convolutional neural networks with ReLUs train several times faster than their equivalents with tanh units" (Krizhevsky et. al., 2017). Additionally, dropout with a fraction of 0.2 is used after the dense layer to reduce overfitting. For the output layer, since the classification task at hand only deals with 2 classes, either a dense layer using the sigmoid activation function with 1 node or the softmax activation function with 2 nodes can be used. Simply put, the prior is chosen because it has a faster update time. Then to compile, the following are used: the Adam optimizer with a 1e-3 learning rate and the rest of the arguments as defaults, the binary cross-entropy loss due to this project's task of binary classification, and the accuracy metric since the goal of this project is to maximize it.

3.2. Hypertuning

The built-in early stopping function is used to monitor the validation loss with patience set to 3 epochs; training will be stopped when there are no improvements to the validation loss after 3 epochs. To fit this model, both the train and the test set are fitted using 10 epochs and a batch size of 100. The classification report accuracy was substantial, resulting in 0.87.

The number of epochs is then increased to 100 to determine the most optimal number of epochs before early stoppings stop the training. The neural network turns out to stop training at epoch 17, however the accuracy did not improve. Likewise, adding another dense layer did not significantly improve the accuracy.

The team thought this was a good stopping point. At epoch 17, the loss comes out to be 0.2849, and the validation loss comes out to be 0.8696. The classification report and confusion matrix of this project can be seen in Table 2 and Fig. 2 respectively.

5. Results

Table 2. The classification report for the dense neural network.

	precision	recall	f1-score	support
0.0	0.88	0.85	0.87	699151
1.0	0.86	0.89	0.87	700849
accuracy			0.87	1400000
macro avg	0.87	0.87	0.87	1400000
weighted avg	0.87	0.87	0.87	1400000

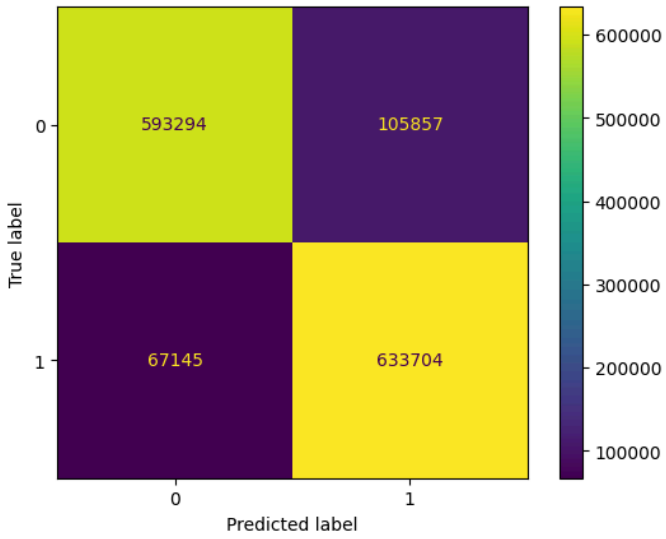

Fig. 2. The confusion matrix for the dense neural network.

5. Conclusion

The team developed a deep neural network model with a classifier to identify the existence of new particles. After scaling the data, splitting the data into an 80/20 training/test split, the team utilized a rectified linear unit (ReLU) activation function and a dense layer using the sigmoid

activation function to produce a binary output. The model developed produced an accuracy of 0.87.

6. References

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional Neural Networks. *Communications of the ACM*, *60*(6), 84–90. https://doi.org/10.1145/3065386

7. Appendix

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import MinMaxScaler

# import tensorflow as tf

from tensorflow.python.keras import Sequential

from tensorflow.python.keras.layers import InputLayer, Dense, Dropout

from tensorflow.python.keras.callbacks import EarlyStopping

from sklearn.metrics import classification_report




def do_my_study():



    df = pd.read_csv('all_train.csv')



    df = preproc(df)



    X_train, X_test, y_train, y_test = model_prep(df)
```

```
    model = do_DenseNeuralNetwork(X_train, y_train, X_test, y_test)
    display_class_report(model, X_test, y_test)


def preproc(df):


    # Change target label
    df.rename(columns={'# label': 'detection'}, inplace=True)


    return df


def model_prep(df):
    X = df.loc[:, df.columns != 'detection']
    y = df['detection']


    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_train = scaler.fit_transform(X)


    scaled_train_df = pd.DataFrame(scaled_train, columns=X.columns.values)
```

```
    X_train, X_test, y_train, y_test = train_test_split(
        scaled_train_df, y, test_size=0.2, random_state=42)

    return X_train, X_test, y_train, y_test


def do_DenseNeuralNetwork(X_train, y_train, X_test, y_test):

    model = Sequential()
    model.add(InputLayer(input_shape=(28,)))
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(.2))
    # model.add(Dense(64, activation='relu'))
    # model.add(Dropout(.2, input_shape=(2,)))
    # model.add(Dense(32, activation='relu'))
    # model.add(Dropout(.2))
    # model.add(Dense(16, activation='relu'))
    # model.add(Dropout(.2))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer='Adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

```
    callback = EarlyStopping(monitor='val_loss', patience=3)


    model.fit(
        X_train, y_train,
        epochs=100,
        validation_data=(X_test, y_test),
        callbacks=[callback],
        batch_size=100)


    return model


def display_class_report(model, X_test, y_test):
    y_pred = model.predict(X_test)
    y_pred = np.round(y_pred)


    print(classification_report(y_test, y_pred))


if __name__ == "__main__":


    do_my_study()
```