2023

DS 7333 | Quantifying the World

CASE STUDY 6 | PARTICLE DETECTION

JOEY HERNANDEZ | DANNY CHANG

Introduction

Background

The exploration of particle physics has propelled our understanding of the fundamental components of the universe, with the detection and identification of subatomic particles being crucial to this endeavor. As particle accelerators achieve higher energies, generating vast amounts of data, traditional analytical methods are increasingly challenged by the scale and complexity involved. The discovery of new particles, such as those that may lie beyond the current scope of the Standard Model, requires innovative approaches that can sift through the noise and discern the signals of novel occurrences.

In response to this challenge, neural networks have emerged as a powerful tool in particle detection. These computational models, inspired by the functioning of the human brain, excel in recognizing complex patterns and making predictions from large datasets. Their application to particle physics is predicated on their ability to learn from data and improve over time, thus enhancing the search for new particles with unprecedented efficiency. By training on known particle interactions, neural networks can subsequently identify anomalies indicative of new particles, representing a fusion of machine learning and fundamental physics that promises to expand the horizons of both fields.

Objective

The principal objective of this study is to design and implement a sophisticated neural network-based system capable of detecting and predicting the existence of new particles in experimental data. This system aims to transcend the limitations of current manual analysis techniques, which are labor-intensive and susceptible to error, and provide a scalable, automated alternative that enhances accuracy and reliability in particle identification. By harnessing the predictive power of neural networks, the project endeavors to facilitate the discovery of particles that may not align with the predictions of the Standard Model, thereby potentially opening new avenues in the field of physics.

Data Inspection

Before delving into the modeling phase, a thorough inspection of the dataset was conducted to ensure its integrity and appropriateness for training a neural network. The dataset presented is pre-processed and consists of instances classified into two categories, signifying the Non-detection (0) or detection (1) of the new particle. An integral part of the data inspection is to verify that the dataset is free from common issues that could potentially skew the training process or bias the model's predictions.

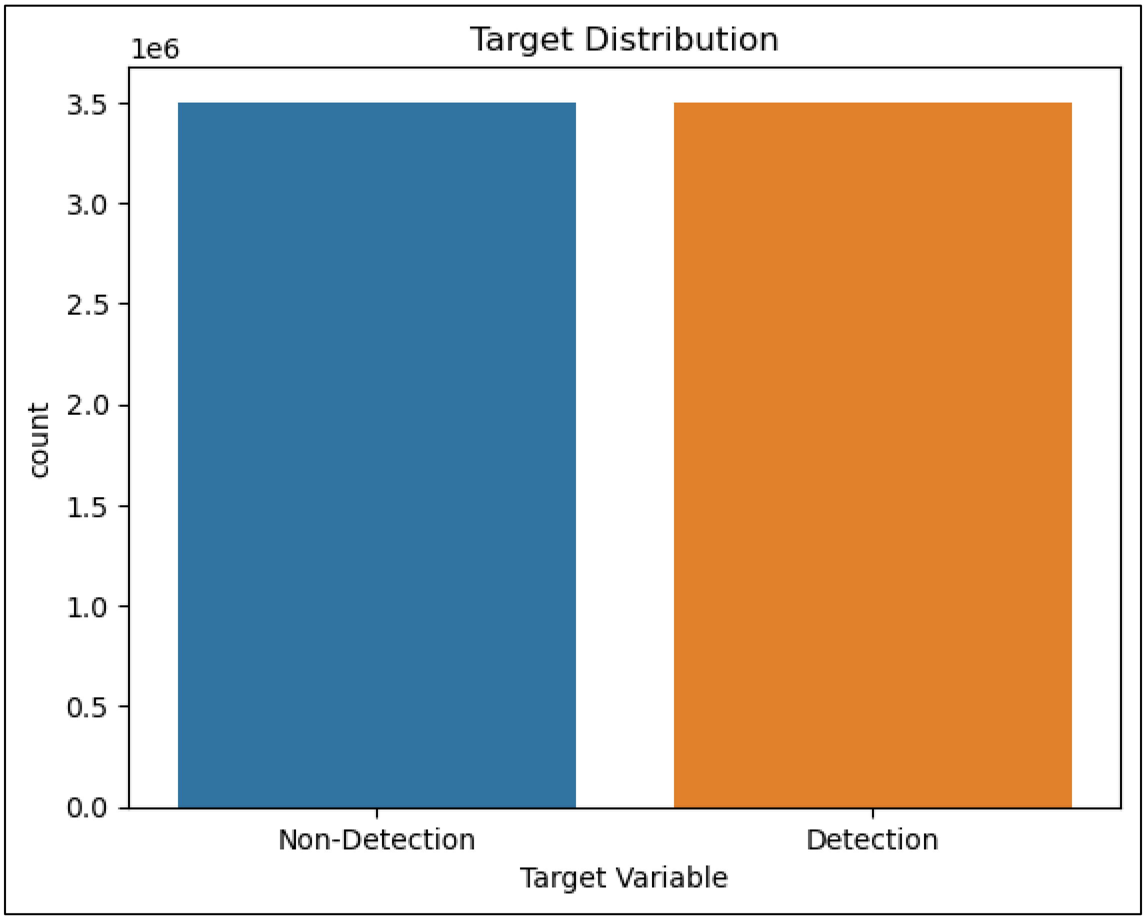
The dataset is confirmed to have no missing values, which implies that there is a complete record for every feature across all instances. This completeness is essential for the training of the neural network, as missing values can introduce ambiguity and affect the model’s ability to learn effectively. Furthermore, the readiness of the data indicates that prior steps such as data cleaning and preprocessing have been meticulously performed, facilitating a smoother transition into the model training phase.

An examination of the target variable distribution reveals a relatively balanced classification problem, with 3,500,879 instances labeled as '1' and 3,499,121 instances labeled as '0'. This balance is critical in machine learning, particularly in classification tasks, because it prevents the model from developing a bias toward the more frequent class. In the context of particle detection, an imbalanced dataset could lead the neural network to favor false negatives or false positives, which would be detrimental to the identification of new particles.

The balanced nature of the classes in this dataset is favorable for the following reasons:

1. It allows the neural network to learn from a dataset that has a symmetrical representation of both outcomes, ensuring that the model's predictions are not skewed by an overrepresentation of one class over the other.
2. It simplifies the process of model evaluation since metrics such as accuracy can be used more effectively when the class distribution is even.
3. It reduces the need for additional data processing techniques such as oversampling or under sampling, which might be necessary if a significant class imbalance were present.

Figure 1: *Count plot illustrating the distribution of the classification target.*



Description*: The count plot displays the class distribution for our target variable, highlighting the evenness between the two instances. This visualization aids in identifying class balance present in the dataset.*

Table 1: *Table detailing the distribution of target class.*

Distribution of Target Variable		
Class	**Class Counts**	**Class Percentages**
Non-Detection	3499121	49.99%
Detection	3500879	50.01%

Description*:* The table presents the distribution of occurrences of the data *target* variable. It succinctly highlights the number of instances for each class and the corresponding percentages.

Modeling

Dense Neural Network (DNN)

Utilizing the TensorFlow framework with its Keras API, we design a Dense Neural Network (DNN) model to engage with the task of predicting the existence of new particles from collision data. The DNN is constructed to include an input layer, which will receive input data corresponding to the shape determined by the feature set from the dataset—specifically, an array length of 28. Following the input, the network architecture unfolds into two hidden layers, each featuring 32 neurons and employing the ReLU activation function. This function is chosen for its non-linear properties and its renowned efficiency in deep learning contexts.

The culmination of the network is marked by an output layer consisting of a single neuron. Here, the sigmoid activation function is engaged, producing a binary output that signifies the probability of the detection or non-detection of a new particle, in line with the binary nature of our target variable.

Our compilation of the DNN leverages the binary cross-entropy loss function, a standard for binary classification problems. This is complemented by the Adam optimizer, a method favored for its adaptability and ability to accelerate the convergence process during training.

Integral to our approach is the prevention of overfitting. While not explicitly included in the initial model construction, we remain cognizant of the need for techniques like dropout and regularization in future iterations should the model excessively adapt to the training data at the cost of its predictive power on new data.

The training process is conducted on a scaled dataset, segmented into training and testing sets, with 80% of the data allocated for training. The model iterates over the data in mini batches, refining its weights incrementally with each epoch. We ensure rigorous evaluation by employing a test set to validate the model's generalization capabilities. The iterative training process continues until the validation performance metrics align with our objective standards. Following satisfactory validation, the model's ultimate performance and predictive efficacy are gauged using the reserved test set. This step is critical as it provides an unbiased assessment of the model's real-world applicability.

Model Design and Training Specifications:

Our deep neural network (DNN) model was meticulously architected to perform binary classification tasks, aimed at discerning two distinct classes in the dataset. The architecture unfolds as follows:

1. The input layer accepts data of shape (28,), which corresponds to the number of features in our dataset.

2. Sequentially, two hidden layers, each with 32 neurons, utilize the ReLU (Rectified Linear Unit) activation function, a choice driven by ReLU's ability to mitigate the vanishing gradient problem and accelerate the training process.
3. The output layer comprises a single neuron with a sigmoid activation function, which is particularly suitable for binary classification, as it maps the output between 0 and 1, indicating the probability of the input being in one of the two classes.

The compilation of the model is crucial, dictating the learning process. We employed the Adam optimizer, celebrated for its adaptive learning rate capability, which inherently adjusts as the training progresses, enhancing the convergence to the minimum loss. The loss function specified was Binary Crossentropy, a natural fit for binary classification problems, as it quantifies the distance between the predicted probabilities and the actual binary outcomes. Tracking the 'accuracy' metric offers a direct measure of the model's performance on the training data.

A pivotal element in our training approach was the integration of early stopping, a form of regularization used to prevent overfitting. Overfitting occurs when a model learns patterns specific to the training data, compromising its ability to generalize. Our early stopping callback monitored the validation loss ('val_loss'), ceasing training if five consecutive epochs failed to present any improvement. This mechanism is a testament to the model's capacity to generalize, indicating that the learning reached its peak when the monitored metric ceased to improve.

By configuring the model to potentially iterate through 1000 epochs with a batch size of 200, we allow it ample opportunity to learn from the data. However, the safety net of early stopping ensures that we halt the training at the optimal moment, striking a balance between underfitting and overfitting, as evidenced by the monitored validation loss stabilization. Thus, the model's training was completed not by exhausting the epochs, but by responding to the early stopping criteria, a clear indication of the model's readiness for evaluation and deployment.

Results:

Our Dense Neural Network (DNN), constructed and trained to classify the existence of new particles, has demonstrated promising performance in its classification tasks. The evaluation metrics, presented in the classification report, highlight an accuracy of 0.88 across the test dataset of 1,400,000 samples. This indicates that the DNN correctly identified the presence or absence of new particles with high reliability.

The precision for class 0 (non-detection) stands at 0.90, suggesting that the model has a high likelihood of correctly identifying true negatives, whereas for class 1 (detection), the precision is 0.86, indicating a slightly lower, yet still robust, likelihood of correctly identifying true positives. The recall scores exhibit a complementary distribution, with class 0 having a recall of 0.86 and class 1 scoring higher at 0.90. These recall measures show that the model is slightly more sensitive to detecting actual instances of new particles than correctly asserting their absence.

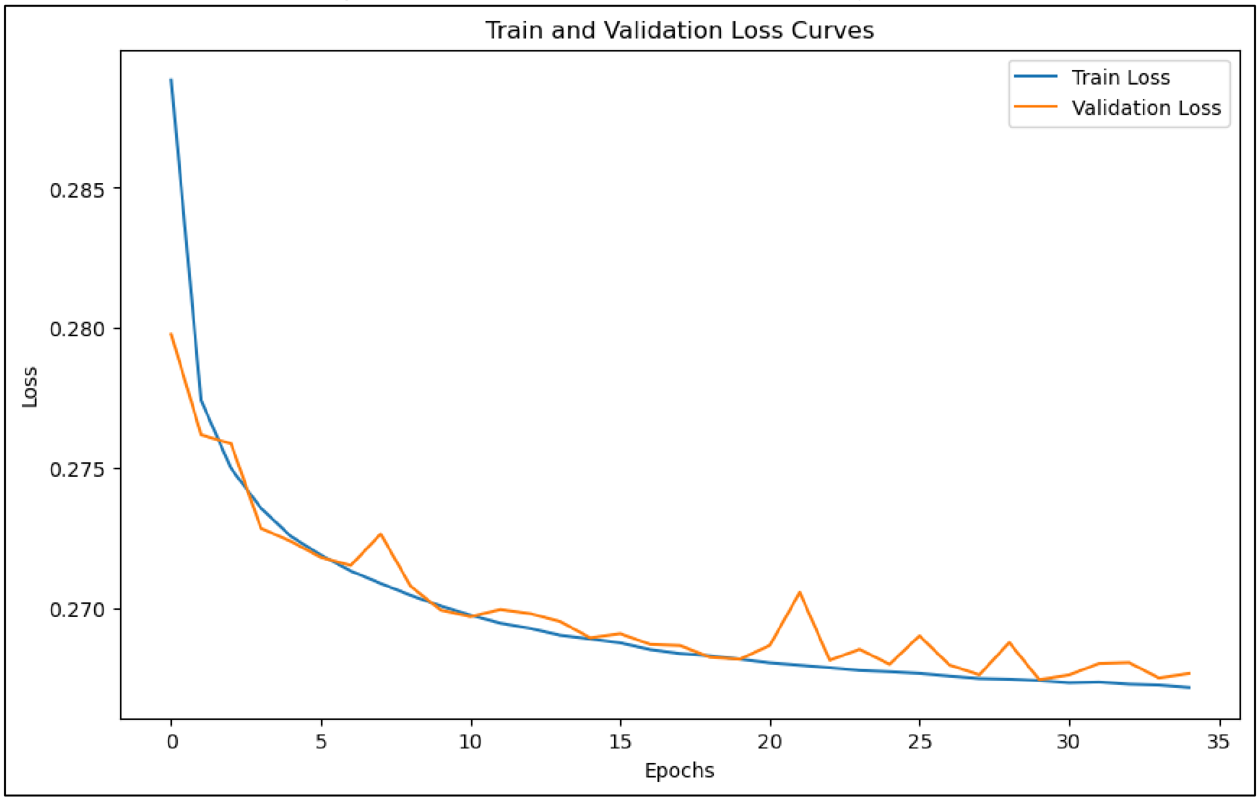
Both classes have an f1-score of 0.88, reflecting a balanced harmony between precision and recall for the model.

The confusion matrix reinforces these findings, displaying a substantial number of true positives (630,774) and true negatives (600,722), while false positives and false negatives are comparatively lower (69,249 and 99,255 respectively). This matrix visually confirms the model's strength in classifying both classes with a high degree of accuracy.

Further validation of the model's efficacy is observed in the Receiver Operating Characteristic (ROC) curve. The curve showcases an area under the curve (AUC) of 0.96, which is a strong indicator of the model's excellent discriminative ability. The ROC curve illustrates the trade-off between the true positive rate and false positive rate at various threshold settings, and an AUC close to 1.0 signifies that the model can distinguish between the classes with high confidence.

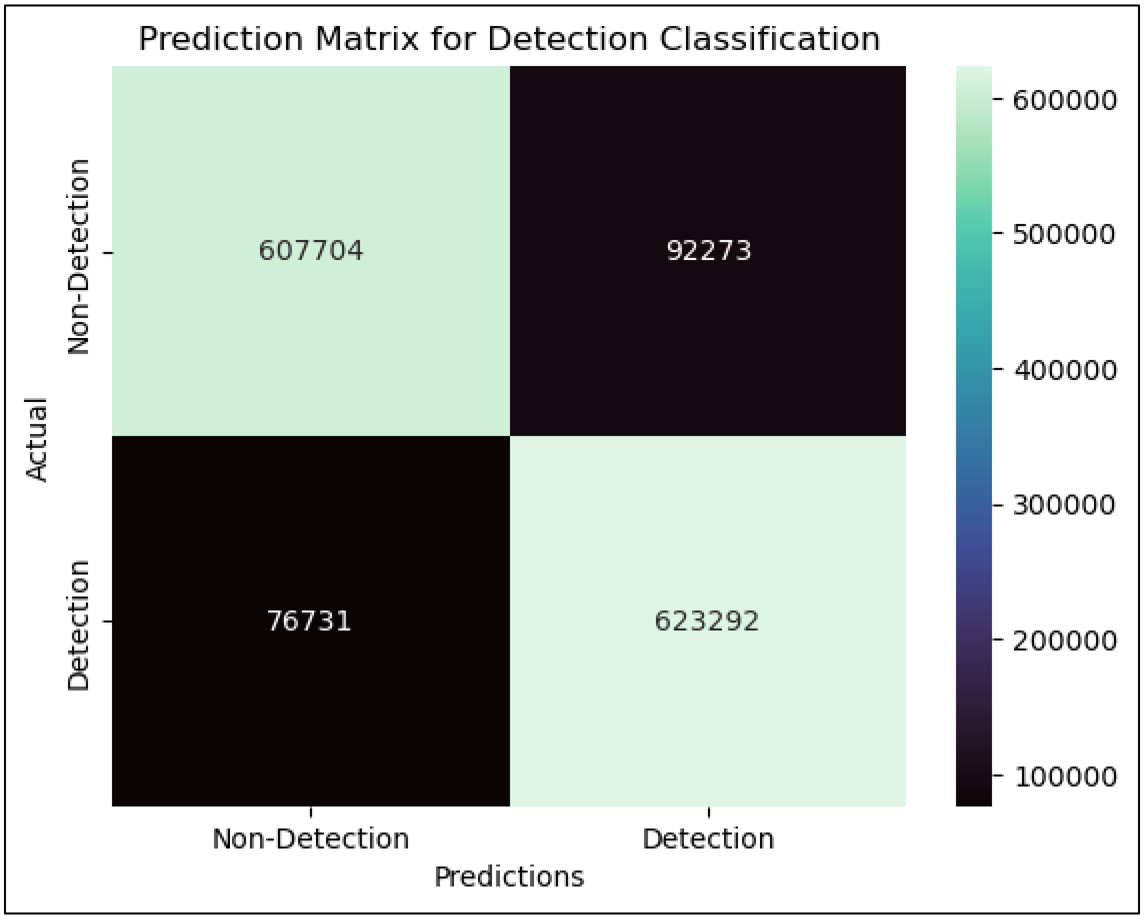
Collectively, these results illustrate the robust performance of the DNN in detecting the presence of new particles within the given dataset. The high values of precision, recall, and AUC point towards a reliable model that could potentially aid in further particle detection research and experimentation.

***Figure 2**: Train and Validation Loss Over Epochs*



***Description**: This plot illustrates the model's training and validation loss per epoch, demonstrating how the losses decrease and converge over time, indicating the model's learning and stabilization, with minimal overfitting as evidenced by the parallel trajectories of both curves.*

Figure 3: *A visual of the Neural Network prediction matrix.*



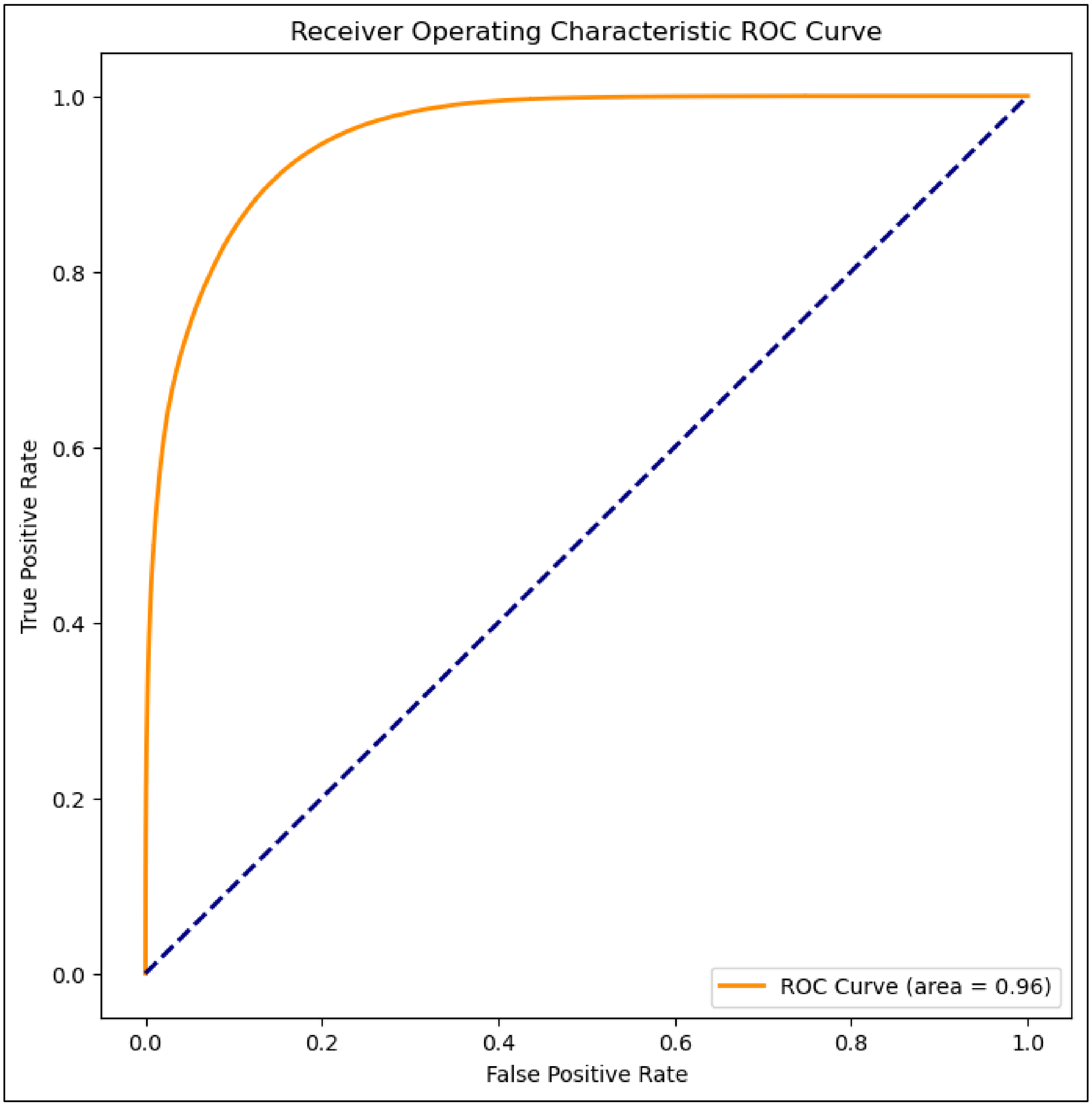
Description*: The plot above illustrates the confusion matrix for the predictions made by the Dense Neural Network model. On the x-axis, we have the predicted labels, and on the y-axis, we have the true labels.*

Table 2 : *Classification Report for Neural Network Classification Model*

Classification Report				
	Precision	**Recall**	**F1- Score**	**Support**
Non-Detection	0.89	0.87	0.88	699977
Detection	0.87	0.89	0.88	700023
Accuracy			0.88	1400000
Macro Avg	0.88	0.88	0.88	1400000
Weighted Avg	0.88	0.88	0.88	1400000

Description*: The classification report indicates that the model achieved a precision of 0.89 for class 0 and 0.87 for class 1, a recall of 0.87 for class 0 and 0.89 for class 1, and an overall accuracy of 88%*

***Figure 4**: ROC Curve Analysis Demonstrating Model's Discriminative Performance*



***Description**: The plot above illustrates the Receiver Operating Characteristic (ROC) curve, highlighting the model's excellent capability in distinguishing between the classes with a high area under the curve (AUC) of 0.96.*

Conclusion

This research represents a significant stride in the field of particle physics, endeavoring to automate and refine the detection of new particles in experimental data through a sophisticated neural network-based approach. The design of the deep neural network, tailored to tackle a nearly balanced classification problem, reflects an architectural optimization for identifying patterns that could suggest the presence of novel particles beyond the Standard Model's scope.

The network's configuration—beginning with an input layer designed to process 28 distinct features, followed by two hidden layers each featuring 32 neurons activated by the ReLU function—was chosen to address and circumvent the challenges of gradient-based optimization. This design choice promotes an effective flow of gradients during training, which is crucial for the learning process of deep neural networks. The concluding layer, equipped with a sigmoid activation function, serves as a decisive indicator, producing a probabilistic output that classifies experimental observations into two categories: 'detection' and 'non-detection'.

Evaluating the model's performance, the confusion matrix and classification report showcase promising results, with an accuracy of 0.88, along with precision, recall, and F1-scores that are consistent across both classes. Such metrics not only suggest a high level of reliability in the model's predictions but also imply a substantial enhancement over traditional manual analysis methods, which are often prone to human error and are not scalable for large data volumes.

The success of the neural network in this study illuminates the path forward for automated systems in particle detection, hinting at the immense potential of machine learning applications in advancing our understanding of the universe. By automating the detection process, researchers can sift through extensive datasets more efficiently and with greater precision, paving the way for the possible discovery of particles that challenge current theoretical frameworks in physics.

Appendix

In [ ]:
```
import pandas as pd
```

In [ ]:
```
df = pd.read_csv('all_train.csv')
df.head()
```

Out[ ]:

	# label	f0	f1	f2	f3	f4	f5	f6	f7	f8	...	f18	f19	f20
0	1.0	-0.346368	0.416306	0.999236	0.475342	0.427493	-0.005984	1.989833	0.344530	1.566297	...	4.105282	0.267826	0.378718
1	1.0	1.708236	-0.319394	-1.241873	-0.887231	-0.871906	-0.005984	-0.001047	-1.038225	0.655748	...	-1.178141	-0.877361	-1.483769
2	0.0	-0.360693	1.794174	0.264738	-0.472273	-0.292344	-1.054221	-1.150495	1.423404	1.270098	...	-1.199511	0.539020	-1.590629
3	1.0	-0.377914	-0.103932	-0.649434	-2.125015	-1.643797	-0.005984	1.011112	-1.040340	-0.541991	...	0.463763	-0.006583	1.089122
4	0.0	-0.067436	-0.636762	-0.620166	-0.062551	1.588715	-0.005984	-0.595304	-1.238987	0.336844	...	-0.552837	-1.418494	-0.562982

5 rows × 29 columns

note that the label is 1 / 0 ... this will impact our loss function

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
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
 5   f4       float64
 6   f5       float64
 7   f6       float64
 8   f7       float64
 9   f8       float64
 10  f9       float64
 11  f10      float64
 12  f11      float64
 13  f12      float64
 14  f13      float64
 15  f14      float64
 16  f15      float64
 17  f16      float64
 18  f17      float64
 19  f18      float64
 20  f19      float64
 21  f20      float64
 22  f21      float64
 23  f22      float64
 24  f23      float64
 25  f24      float64
 26  f25      float64
 27  f26      float64
 28  mass     float64
dtypes: float64(29)
memory usage: 1.5 GB
```

no missing values

In [ ]:
```
df.isna().sum()
```

Out[ ]:

```
# label    0
f0         0
f1         0
f2         0
f3         0
f4         0
f5         0
f6         0
f7         0
f8         0
f9         0
f10        0
f11        0
f12        0
f13        0
f14        0
f15        0
f16        0
f17        0
f18        0
f19        0
f20        0
f21        0
f22        0
f23        0
f24        0
f25        0
f26        0
mass       0
dtype: int64
```

In [ ]:

```
df.describe().transpose()
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
# label	7000000.0	0.500126	0.500000	0.000000	0.000000	1.000000	1.000000	1.000000
f0	7000000.0	0.016125	1.004417	-1.960549	-0.728821	-0.039303	0.690080	4.378282
f1	7000000.0	0.000477	0.997486	-2.365355	-0.733255	0.000852	0.734783	2.365287
f2	7000000.0	0.000027	1.000080	-1.732165	-0.865670	0.000320	0.865946	1.732370
f3	7000000.0	0.010561	0.995600	-9.980274	-0.609229	0.019633	0.679882	4.148023
f4	7000000.0	-0.000105	0.999867	-1.732137	-0.865802	-0.000507	0.865765	1.731978
f5	7000000.0	0.002766	1.000957	-1.054221	-1.054221	-0.005984	0.850488	4.482618
f6	7000000.0	0.018160	0.986775	-3.034787	-0.756609	-0.149953	0.768669	3.720345
f7	7000000.0	0.000025	0.996587	-2.757853	-0.701415	-0.000107	0.701319	2.758590
f8	7000000.0	0.000435	1.000007	-1.732359	-0.865654	0.001385	0.866598	1.731450
f9	7000000.0	-0.006870	1.001938	-1.325801	-1.325801	0.754261	0.754261	0.754261
f10	7000000.0	0.017543	0.994151	-2.835563	-0.723727	-0.128573	0.647864	4.639335
f11	7000000.0	-0.000161	0.998450	-2.602091	-0.703293	-0.000576	0.704100	2.602294
f12	7000000.0	-0.000329	1.000078	-1.732216	-0.866599	-0.001282	0.865832	1.732007
f13	7000000.0	0.001739	0.999737	-1.161915	-1.161915	0.860649	0.860649	0.860649
f14	7000000.0	0.017246	0.999465	-2.454879	-0.699618	-0.097493	0.634705	5.535799
f15	7000000.0	0.000483	0.998429	-2.437812	-0.707026	0.000298	0.708371	2.438369
f16	7000000.0	-0.000554	0.999861	-1.732145	-0.866247	-0.001377	0.864942	1.732738
f17	7000000.0	0.004960	1.001006	-0.815440	-0.815440	-0.815440	1.226331	1.226331
f18	7000000.0	0.011648	1.002725	-1.728284	-0.742363	-0.089925	0.642319	5.866367
f19	7000000.0	-0.000113	1.000038	-2.281867	-0.720685	-0.000067	0.720492	2.282217
f20	7000000.0	0.000077	1.000033	-1.731758	-0.865685	-0.000442	0.865957	1.732740
f21	7000000.0	0.000291	1.000170	-0.573682	-0.573682	-0.573682	-0.573682	1.743123
f22	7000000.0	0.012288	1.010477	-3.631608	-0.541794	-0.160276	0.481219	7.293420
f23	7000000.0	0.009778	1.005418	-4.729473	-0.511552	-0.314403	0.163489	9.333287
f24	7000000.0	0.005270	1.009990	-20.622229	-0.354387	-0.326523	-0.233767	14.990636
f25	7000000.0	-0.001761	0.984451	-3.452634	-0.692510	-0.357030	0.475313	5.277313
f26	7000000.0	0.015331	0.982280	-2.632761	-0.794380	-0.088286	0.761085	4.444690
mass	7000000.0	1000.107387	353.425487	499.999969	750.000000	1000.000000	1250.000000	1500.000000

In [ ]:

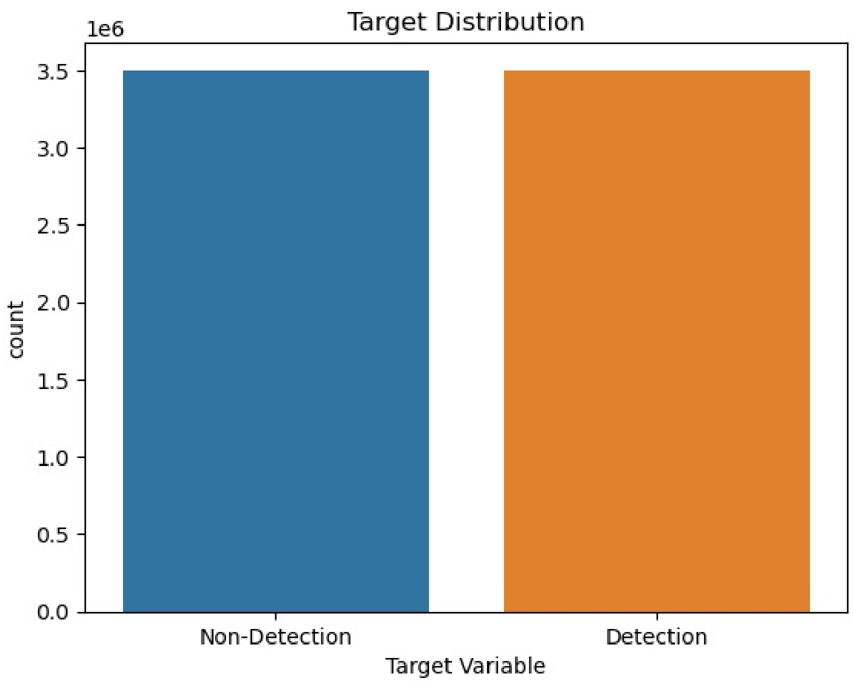
```
df.rename(columns={'# label': 'target'}, inplace=True)
df['target'] = df['target'].astype(int)
```

In [ ]:

```
df['target'].value_counts()
```

```
Out[ ]: 1    3500879
        0    3499121
        Name: target, dtype: int64
```

```
In [ ]: import matplotlib.pyplot as plt
        import seaborn as sns
        sns.countplot(data = df, x = 'target')
        plt.title('Target Distribution')
        plt.xlabel('Target Variable')
        plt.xticks(ticks=[0,1],labels= ['Non-Detection','Detection'])
        plt.show()
```



```
In [ ]: from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler

        X = df.drop('target', axis=1)
        y = df['target']

        scaler = StandardScaler()

        X_scaled = scaler.fit_transform(X)

        X_train, X_test, y_train, y_test = train_test_split(X_scaled,y, train_size=.8, random_state=12)
```

```
In [ ]: X_train.shape

Out[ ]: (5600000, 28)
```

```
In [ ]: import tensorflow as tf
        import numpy as np

        dnn = tf.keras.Sequential()

        dnn.add(tf.keras.Input(shape = (28,)))
        dnn.add(tf.keras.layers.Dense(32, activation = 'relu'))
        dnn.add(tf.keras.layers.Dense(32, activation='relu'))
        dnn.add(tf.keras.layers.Dense(1, activation='sigmoid'))

        dnn.compile(optimizer='adam', loss = 'BinaryCrossentropy', metrics=['accuracy']) # string loss gives us default
```

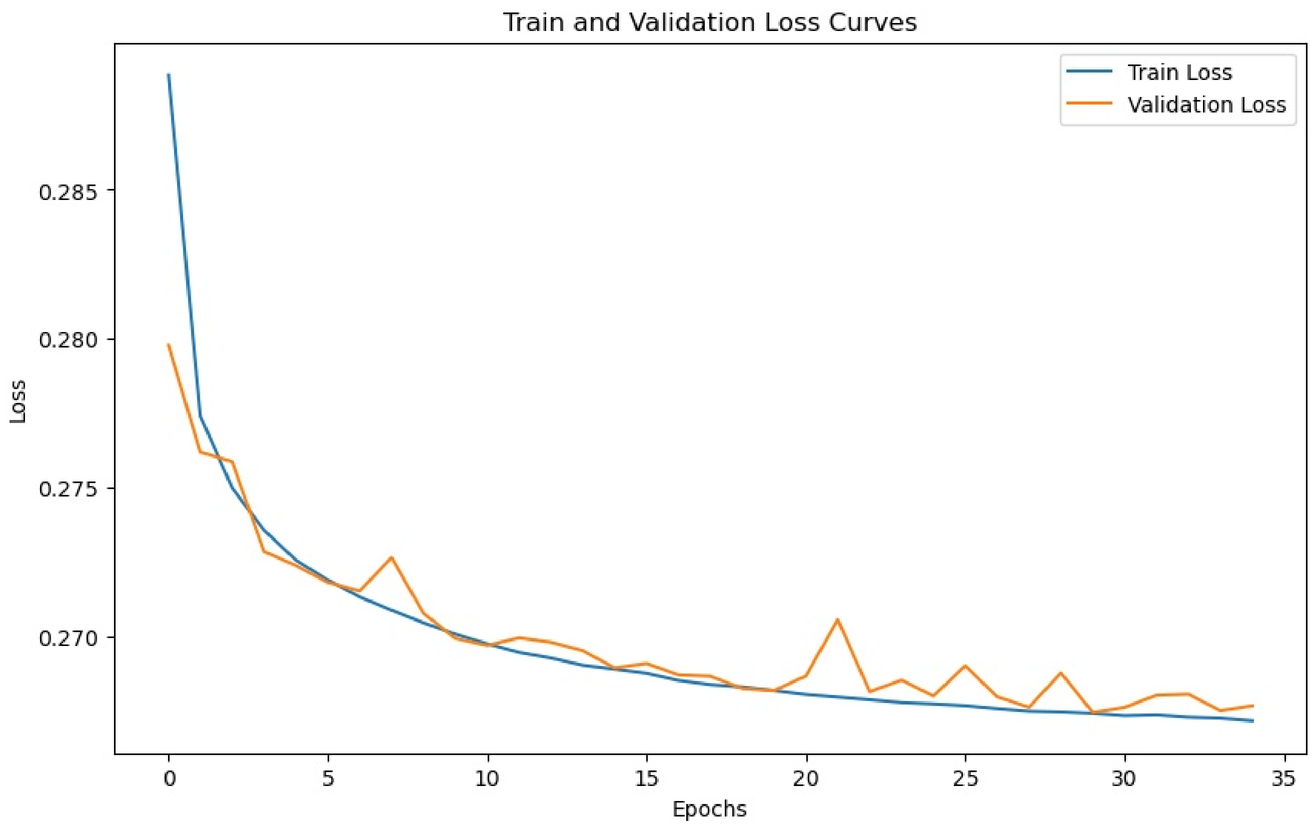
```
In [ ]: from tensorflow.keras.callbacks import EarlyStopping

        saftey = EarlyStopping(monitor = 'val_loss', patience = 5)
        history = dnn.fit(X_train, y_train, epochs=1000, batch_size=200, callbacks=[saftey],validation_split=.2)
```

```
Epoch 1/1000
22400/22400 [==============================] - 28s 1ms/step - loss: 0.2889 - accuracy: 0.8663 - val_loss: 0.279
8 - val_accuracy: 0.8722
Epoch 2/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2774 - accuracy: 0.8734 - val_loss: 0.276
2 - val_accuracy: 0.8749
Epoch 3/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2750 - accuracy: 0.8749 - val_loss: 0.275
9 - val_accuracy: 0.8752
Epoch 4/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2736 - accuracy: 0.8758 - val_loss: 0.272
8 - val_accuracy: 0.8767
Epoch 5/1000
```

```
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2725 - accuracy: 0.8764 - val_loss: 0.272
4 - val_accuracy: 0.8770
Epoch 6/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2719 - accuracy: 0.8768 - val_loss: 0.271
8 - val_accuracy: 0.8769
Epoch 7/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2713 - accuracy: 0.8770 - val_loss: 0.271
5 - val_accuracy: 0.8772
Epoch 8/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2709 - accuracy: 0.8772 - val_loss: 0.272
6 - val_accuracy: 0.8767
Epoch 9/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2704 - accuracy: 0.8775 - val_loss: 0.270
8 - val_accuracy: 0.8777
Epoch 10/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2701 - accuracy: 0.8776 - val_loss: 0.269
9 - val_accuracy: 0.8783
Epoch 11/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2697 - accuracy: 0.8778 - val_loss: 0.269
7 - val_accuracy: 0.8781
Epoch 12/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2694 - accuracy: 0.8781 - val_loss: 0.269
9 - val_accuracy: 0.8783
Epoch 13/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2693 - accuracy: 0.8781 - val_loss: 0.269
8 - val_accuracy: 0.8780
Epoch 14/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2690 - accuracy: 0.8783 - val_loss: 0.269
5 - val_accuracy: 0.8783
Epoch 15/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2689 - accuracy: 0.8784 - val_loss: 0.268
9 - val_accuracy: 0.8789
Epoch 16/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2687 - accuracy: 0.8784 - val_loss: 0.269
1 - val_accuracy: 0.8786
Epoch 17/1000
22400/22400 [==============================] - 28s 1ms/step - loss: 0.2685 - accuracy: 0.8785 - val_loss: 0.268
7 - val_accuracy: 0.8791
Epoch 18/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2684 - accuracy: 0.8786 - val_loss: 0.268
6 - val_accuracy: 0.8790
Epoch 19/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2683 - accuracy: 0.8788 - val_loss: 0.268
2 - val_accuracy: 0.8795
Epoch 20/1000
22400/22400 [==============================] - 28s 1ms/step - loss: 0.2682 - accuracy: 0.8788 - val_loss: 0.268
2 - val_accuracy: 0.8790
Epoch 21/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2680 - accuracy: 0.8789 - val_loss: 0.268
7 - val_accuracy: 0.8789
Epoch 22/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2679 - accuracy: 0.8789 - val_loss: 0.270
5 - val_accuracy: 0.8782
Epoch 23/1000
22400/22400 [==============================] - 28s 1ms/step - loss: 0.2679 - accuracy: 0.8789 - val_loss: 0.268
1 - val_accuracy: 0.8794
Epoch 24/1000
22400/22400 [==============================] - 32s 1ms/step - loss: 0.2678 - accuracy: 0.8790 - val_loss: 0.268
5 - val_accuracy: 0.8790
Epoch 25/1000
22400/22400 [==============================] - 28s 1ms/step - loss: 0.2677 - accuracy: 0.8790 - val_loss: 0.268
0 - val_accuracy: 0.8795
Epoch 26/1000
22400/22400 [==============================] - 32s 1ms/step - loss: 0.2677 - accuracy: 0.8791 - val_loss: 0.269
0 - val_accuracy: 0.8789
Epoch 27/1000
22400/22400 [==============================] - 30s 1ms/step - loss: 0.2676 - accuracy: 0.8791 - val_loss: 0.268
0 - val_accuracy: 0.8792
Epoch 28/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2675 - accuracy: 0.8792 - val_loss: 0.267
6 - val_accuracy: 0.8797
Epoch 29/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2674 - accuracy: 0.8792 - val_loss: 0.268
8 - val_accuracy: 0.8790
Epoch 30/1000
22400/22400 [==============================] - 28s 1ms/step - loss: 0.2674 - accuracy: 0.8792 - val_loss: 0.267
4 - val_accuracy: 0.8797
Epoch 31/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2673 - accuracy: 0.8792 - val_loss: 0.267
6 - val_accuracy: 0.8797
Epoch 32/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2673 - accuracy: 0.8792 - val_loss: 0.268
0 - val_accuracy: 0.8795
Epoch 33/1000
22400/22400 [==============================] - 28s 1ms/step - loss: 0.2673 - accuracy: 0.8792 - val_loss: 0.268
0 - val_accuracy: 0.8790
Epoch 34/1000
22400/22400 [==============================] - 26s 1ms/step - loss: 0.2672 - accuracy: 0.8792 - val_loss: 0.267
5 - val_accuracy: 0.8797
```

```
Epoch 35/1000
22400/22400 [==============================] - 27s 1ms/step - loss: 0.2671 - accuracy: 0.8793 - val_loss: 0.267
6 - val_accuracy: 0.8795
```

In [ ]:
```
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Train and Validation Loss Curves')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



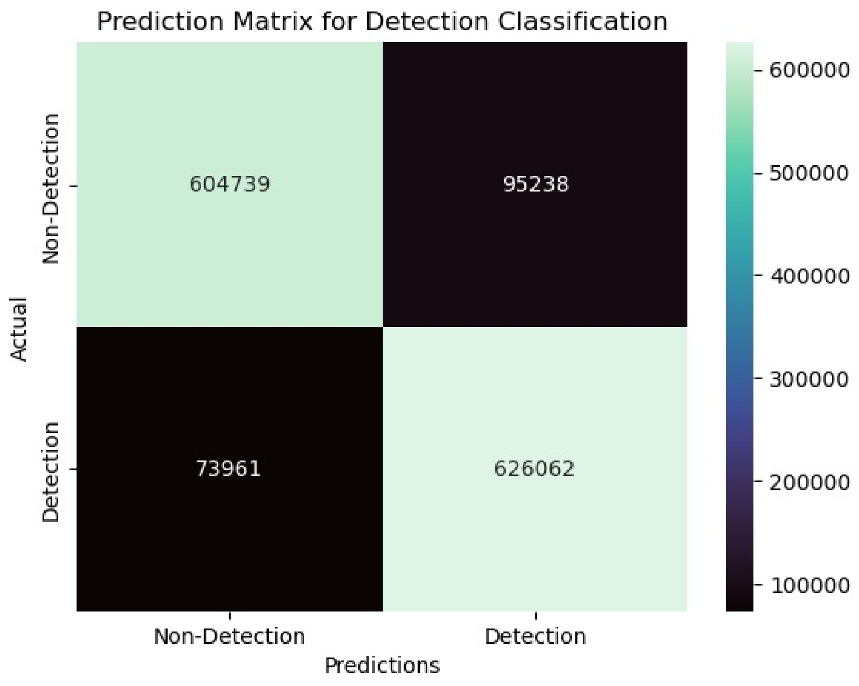
In [ ]:
```
from sklearn.metrics import accuracy_score

y_pred_prob = dnn.predict(X_test).flatten()

acc = []

threshold = np.linspace(0,1,100)

for t in threshold:
    y_pred = (y_pred_prob > t).astype(int)
    accuracies = accuracy_score(y_test,y_pred)
    acc.append(accuracies)

max_accuracy_thres = threshold[np.argmax(acc)]
print(f'Threshold that maximizes accuracy: {max_accuracy_thres}')
```

```
43750/43750 [==============================] - 27s 615us/step
Threshold that maximizes accuracy: 0.5050505050505051
```

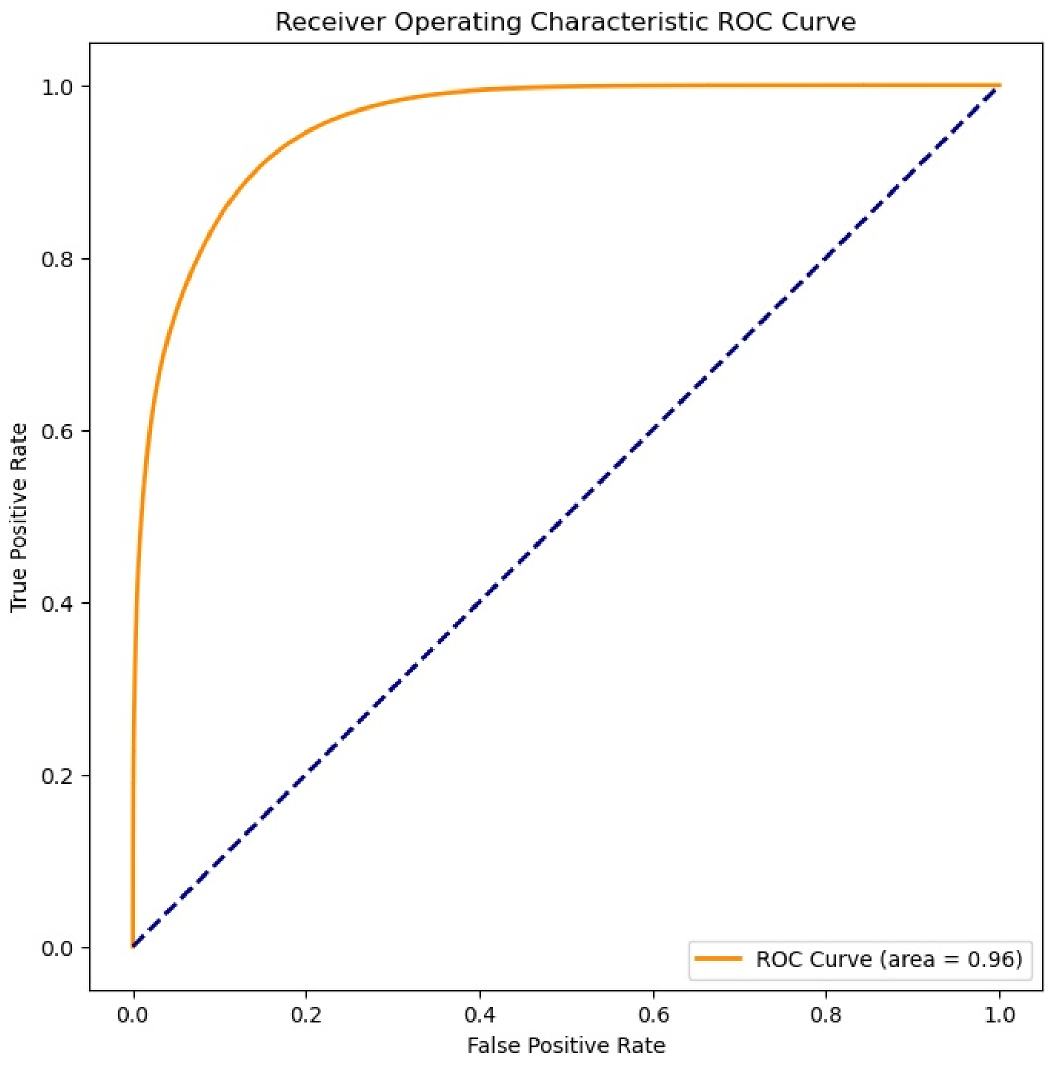
In [ ]:
```
y_pred = (y_pred_prob > 0.5353535353535354).astype(int)
```

In [ ]:
```
from sklearn.metrics import classification_report, confusion_matrix

pred_matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(pred_matrix, annot=True, cmap = 'mako', fmt='.0f')
plt.title('Prediction Matrix for Detection Classification')
plt.xlabel('Predictions')
plt.xticks(ticks=[.5,1.5], labels=['Non-Detection','Detection'])
plt.ylabel('Actual')
plt.yticks(ticks=[0.5,1.5],labels=['Non-Detection','Detection'])

print('Classification Report:\n', classification_report(y_test,y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.86      0.88    699977
           1       0.87      0.89      0.88    700023

    accuracy                           0.88   1400000
   macro avg       0.88      0.88      0.88   1400000
weighted avg       0.88      0.88      0.88   1400000
```



In [ ]:
```python
from sklearn.metrics import roc_curve, auc

fpr, tpr,thresholds = roc_curve(y_test,y_pred_prob)
roc_auc = auc(fpr,tpr)

plt.figure(figsize=(8,8))
lw = 2
plt.plot(fpr,tpr,color = 'darkorange', lw=lw, label = 'ROC Curve (area = %0.2f)' % roc_auc)
plt.plot([0,1],[0,1], color = 'navy',lw=lw, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic ROC Curve')
plt.legend(loc='lower right')
plt.show()
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js