Case Study 6: Particle Detection

Matt Farrow

July 18, 2022

1 Introduction

The goal of this study is to develop a dense neural network that maximizes accuracy when detecting the particle.

2 Methods

2.1 Data Examination

An initial examination of the data revealed 7,000,000 observations and 29 features, including `label`, the response variable, `f0` – `f26`, and `mass`. There is no missing data, and the response variable is almost exactly weighted between the two classes (Table 1).

Class	Count
1.0	3,500,879
0.0	3,499,121

Table 1: Response Variable Count

The only work that was done prior to building the model was to convert the response variable, `label`, to an integer.

2.2 Model Preparation & Execution

The data was split into test and training data sets using a 90/10 split with a stratified shuffle and a `MinMaxScaler` was applied. To set a baseline for the neural network's performance, I first ran a model using logistic regression to determine how it performed on the data. That model produced an accuracy score of 83.7%.

Two neural networks were built for this project in order to explore how different parameters affected the final accuracy performance. The first network was comprised of an input layer, two dense layers with rectified linear unit (relu) activation functions, and a final dense layer with a 'sigmoid' activation (Figure 1).

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               2900

 dense_1 (Dense)             (None, 100)               10100

 dense_2 (Dense)             (None, 1)                 101

=================================================================
Total params: 13,101
Trainable params: 13,101
Non-trainable params: 0
_________________________________________________________________
```

Figure 1: Model 1 Summary

Finally, the model was compiled using a binary cross-entropy loss function with an 'adam' optimizer. This network was fit using batch sizes of 10,000 and 1,000 epochs.

The second neural network alternated dense and dropout layers in an effort to prevent overfitting and ran with batch sizes of 1,000 and 100 epochs.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 28)                0

 dense_3 (Dense)             (None, 128)               3712

 dropout (Dropout)           (None, 128)               0

 dense_4 (Dense)             (None, 128)               16512

 dropout_1 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 1)                 129

=================================================================
Total params: 20,353
Trainable params: 20,353
Non-trainable params: 0
_________________________________________________________________
```

3 Results

3.1 Model 1

In plotting the loss of the training and testing data, I noticed a relatively smooth decrease in the training loss (Figure 2). The testing loss follows a similar slope, but there was much more variation among the epochs.

Figure 2: Loss Over Epochs

The classification report for the first neural network shows the model performed with an accuracy of 88%. This result was similar to other variations of batch size and epochs (Figure 3).

	precision	recall	f1-score	support
0	0.89	0.87	0.88	349912
1	0.87	0.89	0.88	350088
accuracy			0.88	700000
macro avg	0.88	0.88	0.88	700000
weighted avg	0.88	0.88	0.88	700000

Figure 3: Model 1 Classification Report

The confusion matrix shows where misclassifications occurred. There were almost 37,000 false positives and 46,000 false negatives (Figure 4).

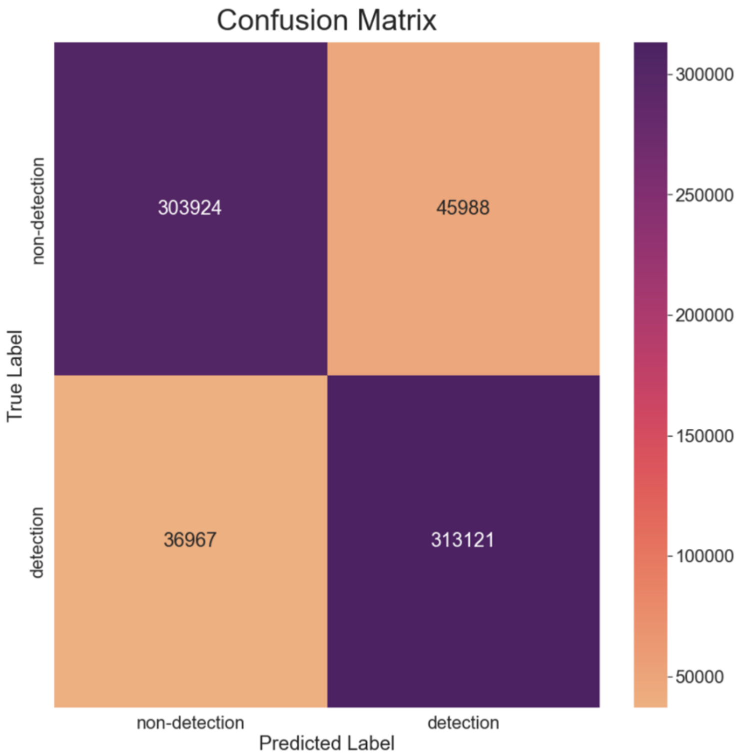


Figure 4: Model 1 Confusion Matrix

3.2 Model 2

The classification report for the second neural network shows the model performed with an accuracy of 83% (Figure 5). In this case the use of the dropout layers decreased the model's performance.

	precision	recall	f1-score	support
0	0.80	0.87	0.83	349912
1	0.85	0.78	0.82	350088
accuracy			0.82	700000
macro avg	0.83	0.82	0.82	700000
weighted avg	0.83	0.82	0.82	700000

Figure 5: Model 2 Classification Report

The confusion matrix shows where misclassifications occurred. The greatest change from the first model is the significant increase in false positives–almost 77,000–an increase of almost 40,000 (Figure 6).

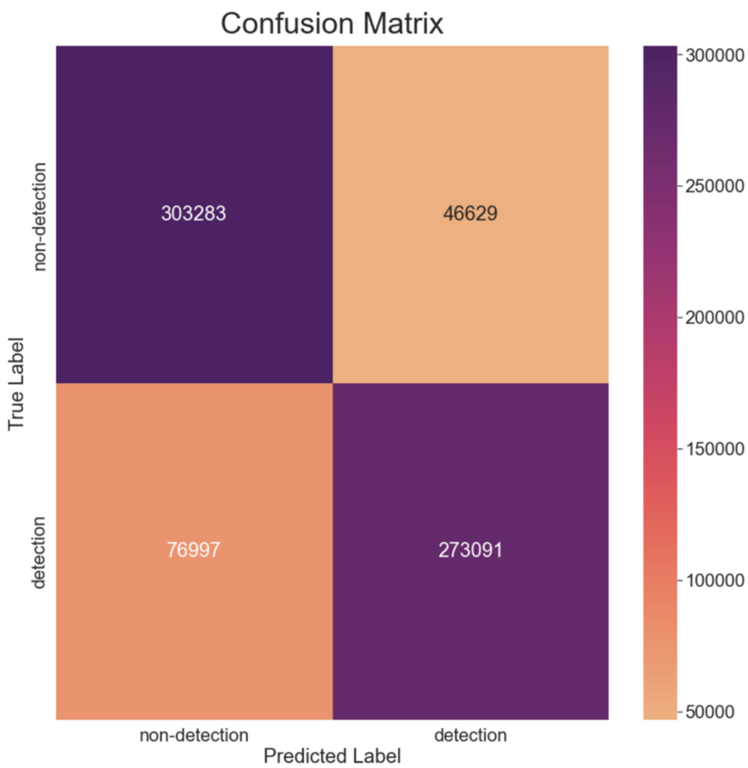


Figure 6: Model 2 Confusion Matrix

4 Conclusion

In conclusion, the first neural network, with its' relatively simple structure performed the best in terms of accuracy. That said, the second neural network ran much more quickly, albeit with a notable drop in accuracy. If I were to redo the second neural network in the future, I'd explore the parameters of the dropout layers to see if an increase in accuracy could be achieved while keeping the model's speed performance.

Appendix

Code

Code begins on the following page.

Case Study 6

Description

Build a dense neural network to accurately detect the particle. The goal is to maximize your accuracy. Include a discussion of how you know your model has finished training as well as what design decisions you made while building the network.

Submit your assignment to the Assignments section of the online campus. For more information regarding case studies, see the syllabus.

In [1]:
```
# Import general libraries
import graphviz
import joblib
import numpy as np
import pandas as pd
import pickle

# Import plotting libraries
import matplotlib as mpl
import matplotlib.pyplot as plt
import pydot
import seaborn as sns

# Import sklearn libraries
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler

# Set up OS-level processes
import os
cwd = os.getcwd()
d = os.path.dirname(cwd)
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'


# Import tensorflow libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import InputLayer, Dense, Activation, Dropout
from tensorflow.keras.callbacks import EarlyStopping

# Set pandas display options
pd.options.display.max_rows = 99999
pd.options.display.max_columns = 99999
```

In [2]:
```
# %%time

# # Read in data and save it as a pickle file
# data = pd.read_csv('all_train.csv')
# data.to_pickle("data.pkl")
```

In [3]:
```
# Load pickle file
df = pd.read_pickle("data.pkl")
```

Exploratory Data Analysis

In [4]:
```
df.shape
```

Out[4]: (7000000, 29)

In [5]:
```
df.head()
```

Out[5]:

	# label	f0	f1	f2	f3	f4	f5	f6	
0	1.0	-0.346368	0.416306	0.999236	0.475342	0.427493	-0.005984	1.989833	0.3
1	1.0	1.708236	-0.319394	-1.241873	-0.887231	-0.871906	-0.005984	-0.001047	-1.0
2	0.0	-0.360693	1.794174	0.264738	-0.472273	-0.292344	-1.054221	-1.150495	1.4
3	1.0	-0.377914	-0.103932	-0.649434	-2.125015	-1.643797	-0.005984	1.011112	-1.0
4	0.0	-0.067436	-0.636762	-0.620166	-0.062551	1.588715	-0.005984	-0.595304	-1.2

In [6]:
```
df.tail()
```

Out[6]:

	# label	f0	f1	f2	f3	f4	f5	f
6999995	0.0	1.617264	-0.537084	-1.275867	0.650799	-1.511621	0.850488	0.59639
6999996	0.0	-0.511357	0.270927	0.085989	-0.243802	-1.035668	-0.005984	-0.12721
6999997	1.0	0.062408	-0.987203	0.570667	1.517195	0.639548	-1.054221	1.11523
6999998	1.0	1.659131	1.096223	0.562821	1.627193	0.767236	-1.054221	1.07999
6999999	1.0	0.002034	0.744152	-0.908839	-0.770454	1.008405	-1.054221	-0.37015

In [7]:
```
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
```

In [8]:

```
df.describe()
```

Out[8]:

	# label	f0	f1	f2	f3	
count	7.000000e+06	7.000000e+06	7.000000e+06	7.000000e+06	7.000000e+06	7.00000
mean	5.001256e-01	1.612528e-02	4.770022e-04	2.686578e-05	1.056081e-02	-1.05
std	5.000000e-01	1.004417e+00	9.974864e-01	1.000080e+00	9.956003e-01	9.9986
min	0.000000e+00	-1.960549e+00	-2.365355e+00	-1.732165e+00	-9.980274e+00	-1.7321
25%	0.000000e+00	-7.288206e-01	-7.332548e-01	-8.656704e-01	-6.092291e-01	-8.65
50%	1.000000e+00	-3.930319e-02	8.523957e-04	3.199154e-04	1.963316e-02	-5.0701
75%	1.000000e+00	6.900799e-01	7.347832e-01	8.659464e-01	6.798818e-01	8.6576
max	1.000000e+00	4.378282e+00	2.365287e+00	1.732370e+00	4.148023e+00	1.73197

In [9]:

```python
# Missing values?
df.isnull().sum()
```

Out[9]:

```
# 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 [10]:
```
# Number of unique values per feature
df.nunique()
```

Out[10]:
```
# label          2
f0         1142885
f1         2311224
f2         1849439
f3         1359060
f4         1849442
f5              10
f6         1129730
f7         2383592
f8         1850813
f9               2
f10        1063674
f11        2358787
f12        1850048
f13              2
f14         979658
f15        2320116
f16        1851285
f17              2
f18         882518
f19        2263637
f20        1850152
f21              2
f22        1142839
f23         568614
f24         393987
f25         862612
f26         931270
mass             5
dtype: int64
```

In [11]:
```
df['# label'].value_counts()
```

Out[11]:
```
1.0    3500879
0.0    3499121
Name: # label, dtype: int64
```

In [12]:
```
# Convert label to an integer
df['# label'] = df['# label'].astype(int)

# Verify conversion
df['# label'].value_counts()
```

Out[12]:
```
1    3500879
0    3499121
Name: # label, dtype: int64
```

Helper Functions

In [13]:
```python
def get_confusion_matrix(y, yhat, mat_title = "Confusion Matrix"):
    plt.figure(figsize = (15,15))
    sns.set(font_scale = 2)
    x_axis_labels = ['non-detection','detection']
    y_axis_labels = ['non-detection','detection']
    cm_n = confusion_matrix(y, yhat)
    ax = sns.heatmap(cm_n,
                     cmap = 'flare',
                     annot = True,
                     fmt = '2d',
                     xticklabels = x_axis_labels,
                     yticklabels = y_axis_labels)
    ax.set(xlabel = 'Predicted Label',
           ylabel = 'True Label')
    ax.set_title(mat_title,
                 fontdict = {'fontsize':36},
                 pad = 15)

def get_classification_report(x_train, y_train, x_test, y_test, pred, model)
    print("Classification Report:")
    print(classification_report(y_test,pred))
    get_confusion_matrix(y_test,pred)
```

Set up Train/Test Data

In [14]:
```python
# Define X & y
X, y = df.drop('# label', axis = 1), df['# label']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = 0.10,
                                                    shuffle = True,
                                                    random_state = 123,
                                                    stratify = y)

# Define and scale the testing and training data
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Baseline Model with Logistic Regression

In [15]:
```python
lr = LogisticRegression(solver = 'lbfgs', max_iter = 1000)
lr.fit(X_train, y_train)
print(lr.score(X_test, y_test))
```

```
0.8369685714285714
```

Build Initial Neural Network

In [16]:

```
# Build a sequential NN model using Tensorflow and Keras
model = Sequential()
model.add(InputLayer(input_shape = (28,)))
model.add(Dense(units = 100, activation = 'relu'))
model.add(Dense(units = 100, activation = 'relu'))
model.add(Dense(units = 1,   activation = 'sigmoid'))
model.compile(loss = 'binary_crossentropy', optimizer = 'adam')
```

In [26]:

```
%%time

model.fit(x = X_train,
          y = y_train,
          batch_size = 10000,
          epochs = 100,
          validation_data = (X_test, y_test),
          verbose = 1,
          callbacks = [EarlyStopping(monitor = 'val_loss',
                                     mode = 'min',
                                     verbose = 1,
                                     patience = 15)]
          )
```

```
Epoch 1/100
630/630 [==============================] - 8s 12ms/step - loss: 0.2644 - val
_loss: 0.2646
Epoch 2/100
630/630 [==============================] - 8s 13ms/step - loss: 0.2639 - val
_loss: 0.2651
Epoch 3/100
630/630 [==============================] - 9s 14ms/step - loss: 0.2637 - val
_loss: 0.2674
Epoch 4/100
630/630 [==============================] - 9s 15ms/step - loss: 0.2635 - val
_loss: 0.2652
Epoch 5/100
630/630 [==============================] - 9s 15ms/step - loss: 0.2636 - val
_loss: 0.2652
Epoch 6/100
630/630 [==============================] - 9s 15ms/step - loss: 0.2635 - val
_loss: 0.2646
Epoch 7/100
630/630 [==============================] - 8s 13ms/step - loss: 0.2635 - val
_loss: 0.2648
Epoch 8/100
630/630 [==============================] - 8s 13ms/step - loss: 0.2634 - val
_loss: 0.2645
Epoch 9/100
630/630 [==============================] - 8s 13ms/step - loss: 0.2634 - val
_loss: 0.2642
Epoch 10/100
630/630 [==============================] - 9s 14ms/step - loss: 0.2637 - val
_loss: 0.2643
Epoch 11/100
630/630 [==============================] - 9s 14ms/step - loss: 0.2634 - val
_loss: 0.2641
Epoch 12/100
```

```
630/630 [==============================] - 9s 14ms/step - loss: 0.2633 - val
_loss: 0.2642
Epoch 13/100
630/630 [==============================] - 9s 15ms/step - loss: 0.2633 - val
_loss: 0.2651
Epoch 14/100
630/630 [==============================] - 9s 15ms/step - loss: 0.2634 - val
_loss: 0.2644
Epoch 15/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2633 - va
l_loss: 0.2644
Epoch 16/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2632 - va
l_loss: 0.2637
Epoch 17/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2632 - va
l_loss: 0.2641
Epoch 18/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2632 - va
l_loss: 0.2671
Epoch 19/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2632 - va
l_loss: 0.2644
Epoch 20/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2633 - va
l_loss: 0.2653
Epoch 21/100
630/630 [==============================] - 9s 15ms/step - loss: 0.2631 - val
_loss: 0.2640
Epoch 22/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2632 - va
l_loss: 0.2639
Epoch 23/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2630 - va
l_loss: 0.2654
Epoch 24/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2630 - va
l_loss: 0.2644
Epoch 25/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2631 - va
l_loss: 0.2638
Epoch 26/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2631 - va
l_loss: 0.2638
Epoch 27/100
630/630 [==============================] - 9s 14ms/step - loss: 0.2630 - val
_loss: 0.2637
Epoch 28/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2628 - va
l_loss: 0.2636
Epoch 29/100
630/630 [==============================] - 9s 15ms/step - loss: 0.2629 - val
_loss: 0.2637
Epoch 30/100
630/630 [==============================] - 9s 15ms/step - loss: 0.2630 - val
_loss: 0.2634
Epoch 31/100
```

```
630/630 [==============================] - 10s 15ms/step - loss: 0.2627 - va
l_loss: 0.2637
Epoch 32/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2628 - va
l_loss: 0.2643
Epoch 33/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2628 - va
l_loss: 0.2641
Epoch 34/100
630/630 [==============================] - 9s 14ms/step - loss: 0.2629 - val
_loss: 0.2637
Epoch 35/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2628 - va
l_loss: 0.2637
Epoch 36/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2626 - va
l_loss: 0.2636
Epoch 37/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2628 - va
l_loss: 0.2634
Epoch 38/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2627 - va
l_loss: 0.2638
Epoch 39/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2626 - va
l_loss: 0.2631
Epoch 40/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2626 - va
l_loss: 0.2636
Epoch 41/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2627 - va
l_loss: 0.2637
Epoch 42/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2626 - va
l_loss: 0.2636
Epoch 43/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2624 - va
l_loss: 0.2639
Epoch 44/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2626 - va
l_loss: 0.2644
Epoch 45/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2625 - va
l_loss: 0.2646
Epoch 46/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2624 - va
l_loss: 0.2637
Epoch 47/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2625 - va
l_loss: 0.2634
Epoch 48/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2624 - va
l_loss: 0.2638
Epoch 49/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2626 - va
l_loss: 0.2630
Epoch 50/100
```

```
630/630 [==============================] - 10s 16ms/step - loss: 0.2624 - va
l_loss: 0.2632
Epoch 51/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2624 - va
l_loss: 0.2637
Epoch 52/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2624 - va
l_loss: 0.2634
Epoch 53/100
630/630 [==============================] - 11s 17ms/step - loss: 0.2622 - va
l_loss: 0.2632
Epoch 54/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2622 - va
l_loss: 0.2639
Epoch 55/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2624 - va
l_loss: 0.2629
Epoch 56/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2622 - va
l_loss: 0.2630
Epoch 57/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2622 - va
l_loss: 0.2633
Epoch 58/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2622 - va
l_loss: 0.2634
Epoch 59/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2620 - va
l_loss: 0.2636
Epoch 60/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2620 - va
l_loss: 0.2630
Epoch 61/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2621 - va
l_loss: 0.2637
Epoch 62/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2620 - va
l_loss: 0.2630
Epoch 63/100
630/630 [==============================] - 9s 15ms/step - loss: 0.2619 - val
_loss: 0.2629
Epoch 64/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2621 - va
l_loss: 0.2644
Epoch 65/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2619 - va
l_loss: 0.2630
Epoch 66/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2621 - va
l_loss: 0.2627
Epoch 67/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2618 - va
l_loss: 0.2629
Epoch 68/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2621 - va
l_loss: 0.2634
Epoch 69/100
```

```
630/630 [==============================] - 10s 15ms/step - loss: 0.2618 - va
l_loss: 0.2628
Epoch 70/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2619 - va
l_loss: 0.2633
Epoch 71/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2618 - va
l_loss: 0.2629
Epoch 72/100
630/630 [==============================] - 9s 15ms/step - loss: 0.2617 - val
_loss: 0.2625
Epoch 73/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2618 - va
l_loss: 0.2630
Epoch 74/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2618 - va
l_loss: 0.2635
Epoch 75/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2617 - va
l_loss: 0.2634
Epoch 76/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2616 - va
l_loss: 0.2631
Epoch 77/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2617 - va
l_loss: 0.2628
Epoch 78/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2617 - va
l_loss: 0.2624
Epoch 79/100
630/630 [==============================] - 11s 17ms/step - loss: 0.2616 - va
l_loss: 0.2624
Epoch 80/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2616 - va
l_loss: 0.2624
Epoch 81/100
630/630 [==============================] - 9s 14ms/step - loss: 0.2616 - val
_loss: 0.2625
Epoch 82/100
630/630 [==============================] - 9s 15ms/step - loss: 0.2616 - val
_loss: 0.2640
Epoch 83/100
630/630 [==============================] - 9s 15ms/step - loss: 0.2617 - val
_loss: 0.2628
Epoch 84/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2615 - va
l_loss: 0.2629
Epoch 85/100
630/630 [==============================] - 9s 15ms/step - loss: 0.2615 - val
_loss: 0.2635
Epoch 86/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2615 - va
l_loss: 0.2630
Epoch 87/100
630/630 [==============================] - 10s 16ms/step - loss: 0.2615 - va
l_loss: 0.2628
Epoch 88/100
```

```
630/630 [==============================] - 9s 15ms/step - loss: 0.2615 - val
_loss: 0.2635
Epoch 89/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2614 - va
l_loss: 0.2625
Epoch 90/100
630/630 [==============================] - 10s 15ms/step - loss: 0.2613 - va
l_loss: 0.2626
Epoch 91/100
630/630 [==============================] - 9s 15ms/step - loss: 0.2613 - val
_loss: 0.2651
Epoch 92/100
630/630 [==============================] - 9s 15ms/step - loss: 0.2613 - val
_loss: 0.2628
Epoch 93/100
630/630 [==============================] - 9s 15ms/step - loss: 0.2613 - val
_loss: 0.2629
Epoch 93: early stopping
CPU times: user 42min 15s, sys: 15min 4s, total: 57min 19s
Wall time: 14min 52s
```

Out[26]:
```
<keras.callbacks.History at 0x1423068f0>
```

In [27]:
```
# Visualize model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               2900

 dense_1 (Dense)             (None, 100)               10100

 dense_2 (Dense)             (None, 1)                 101

=================================================================
Total params: 13,101
Trainable params: 13,101
Non-trainable params: 0
_________________________________________________________________
```

In [28]:
```
model_loss = pd.DataFrame(model.history.history)
```

In [29]:
```
def plot_early_stop_rounds(model_loss):
    plt.subplots(figsize = (10, 6))
    plt.plot(model_loss['loss'],lw = 2.5, label = 'train')
    plt.plot(model_loss['val_loss'],lw = 2.5, label = 'test')
    plt.title('Train/Test Loss vs. Epochs',fontdict = {'fontsize':20}, pad =
    plt.ylabel('Loss')
    plt.xlabel('Epochs')
    plt.legend()
    plt.show()
```

In [30]:
```
plot_early_stop_rounds(model_loss)
```

In [31]:
```
# Save the train/test loss history
file_name_model_history = "train_test_history.sav"
joblib.dump(model_loss, file_name_model_history)
```

Out[31]:
```
['train_test_history.sav']
```

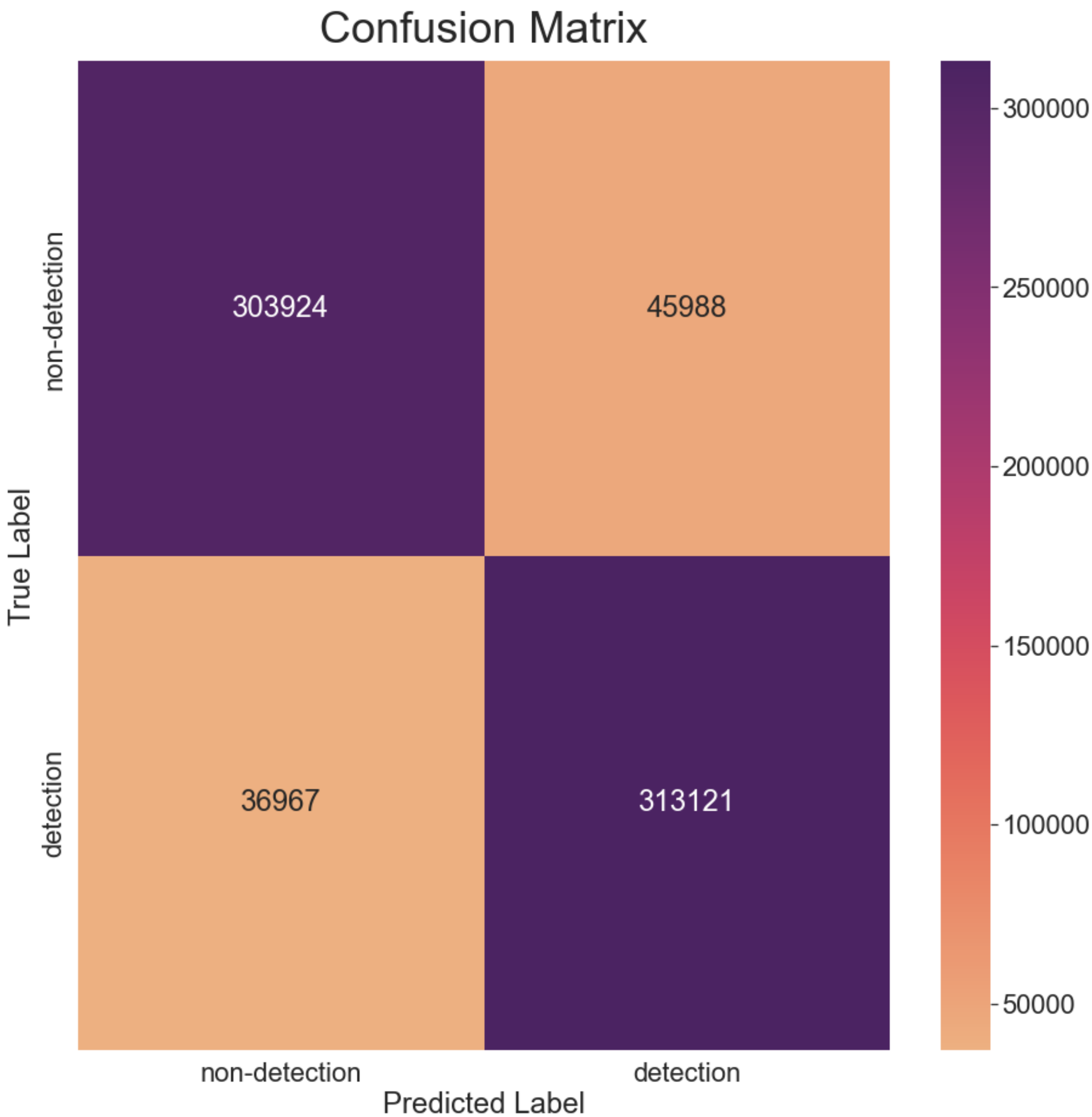
In [32]:
```
# Load the loss history
model_loss = joblib.load(file_name_model_history)
```

In [33]:
```
predictions = (model.predict(X_test) > 0.5).astype("int32")
get_classification_report(X_train,
                          y_train,
                          X_test,
                          y_test,
                          predictions,
                          model)
```

```
21875/21875 [==============================] - 12s 526us/step
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.87      0.88    349912
           1       0.87      0.89      0.88    350088

    accuracy                           0.88    700000
   macro avg       0.88      0.88      0.88    700000
weighted avg       0.88      0.88      0.88    700000
```



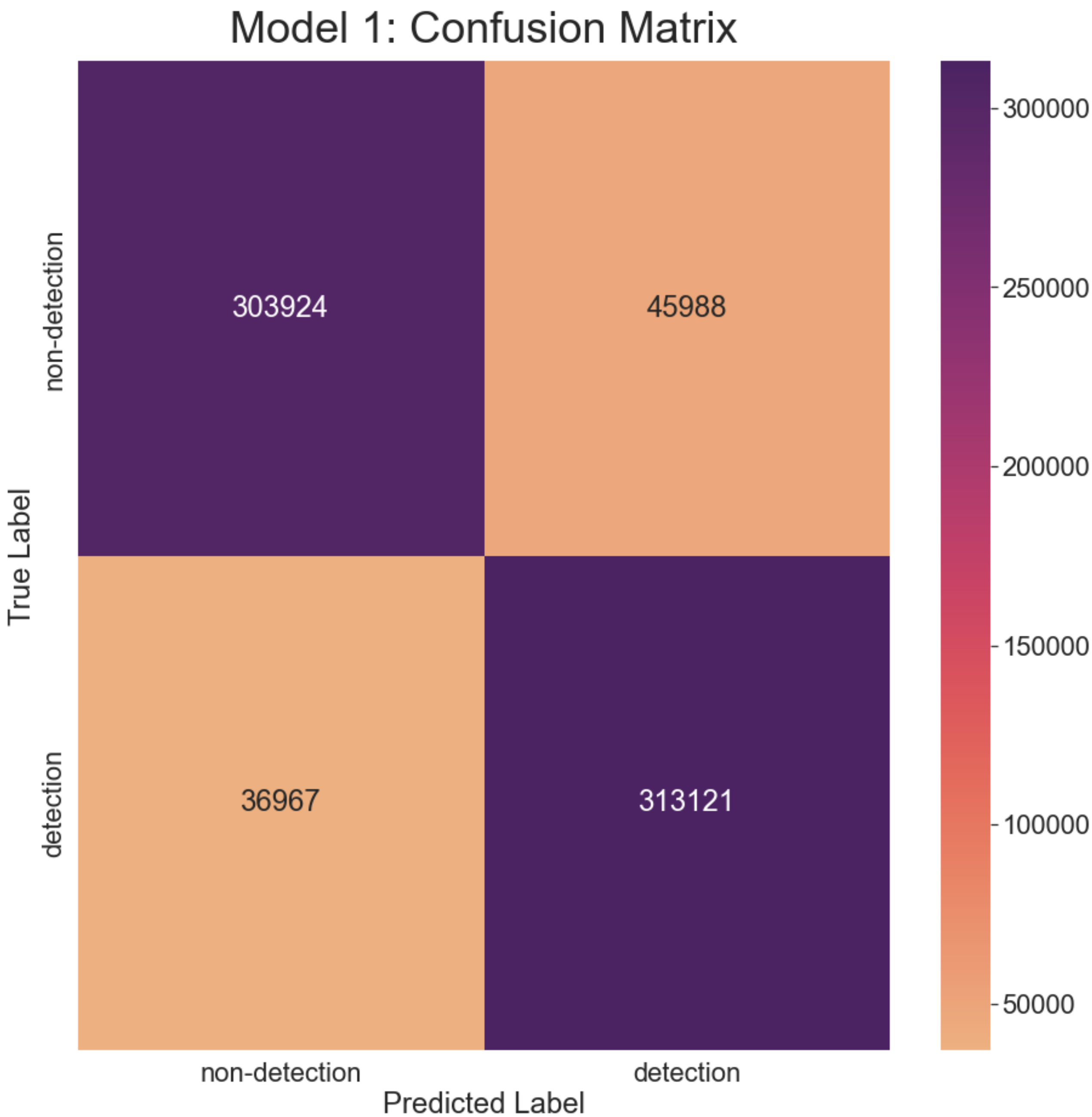
In [34]:
```
# Define path to keras model
path = cwd + "/keras_model"

# Save model
model.save(path)
```

```
INFO:tensorflow:Assets written to: /Users/matt/Documents/GitHub/7333-qtw/Cas
e Study 6/keras_model/assets
```

In [35]:
```
# Load model
model = keras.models.load_model(path)
```

In [36]:
```
get_confusion_matrix(y_test, predictions, mat_title = 'Model 1: Confusion Ma
```



Define 2nd Model

In [37]:
```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape = (28,)),
    tf.keras.layers.Dense(128, activation = 'relu'),
    tf.keras.layers.Dropout(.2, input_shape = (2,)),
    tf.keras.layers.Dense(128, activation = 'relu'),
    tf.keras.layers.Dropout(.2, input_shape = (2,)),
    tf.keras.layers.Dense(1, activation = 'linear')
])
```

In [38]:
```
# Compile model and define metrics
model.compile(optimizer = 'adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics = ['accuracy'])
```

In [39]:
```
# Define early stopping
early_stop = EarlyStopping(monitor = 'val_loss', patience = 3)
```

In [41]:
```
# Fit the model
history = model.fit(X_train,
                    y_train,
                    epochs = 10,
                    validation_data = (X_test, y_test),
                    callbacks = [early_stop],
                    batch_size = 1000)
```

```
Epoch 1/10
6300/6300 [==============================] - 24s 4ms/step - loss: 0.4627 - a
ccuracy: 0.7887 - val_loss: 0.3736 - val_accuracy: 0.8305
Epoch 2/10
6300/6300 [==============================] - 24s 4ms/step - loss: 0.4289 - a
ccuracy: 0.8081 - val_loss: 0.3622 - val_accuracy: 0.8330
Epoch 3/10
6300/6300 [==============================] - 27s 4ms/step - loss: 0.4301 - a
ccuracy: 0.8049 - val_loss: 0.3860 - val_accuracy: 0.8245
Epoch 4/10
6300/6300 [==============================] - 26s 4ms/step - loss: 0.4244 - a
ccuracy: 0.8110 - val_loss: 0.3689 - val_accuracy: 0.8370
Epoch 5/10
6300/6300 [==============================] - 25s 4ms/step - loss: 0.4208 - a
ccuracy: 0.8119 - val_loss: 0.4331 - val_accuracy: 0.8234
```

In [42]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 28)                0

 dense_3 (Dense)             (None, 128)               3712

 dropout (Dropout)           (None, 128)               0

 dense_4 (Dense)             (None, 128)               16512

 dropout_1 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 1)                 129

=================================================================
Total params: 20,353
Trainable params: 20,353
Non-trainable params: 0
_________________________________________________________________
```

In [43]:
```
model.evaluate(X_test, y_test, batch_size = 1000)
```

```
700/700 [==============================] - 1s 2ms/step - loss: 0.4331 - accu
racy: 0.8234
```

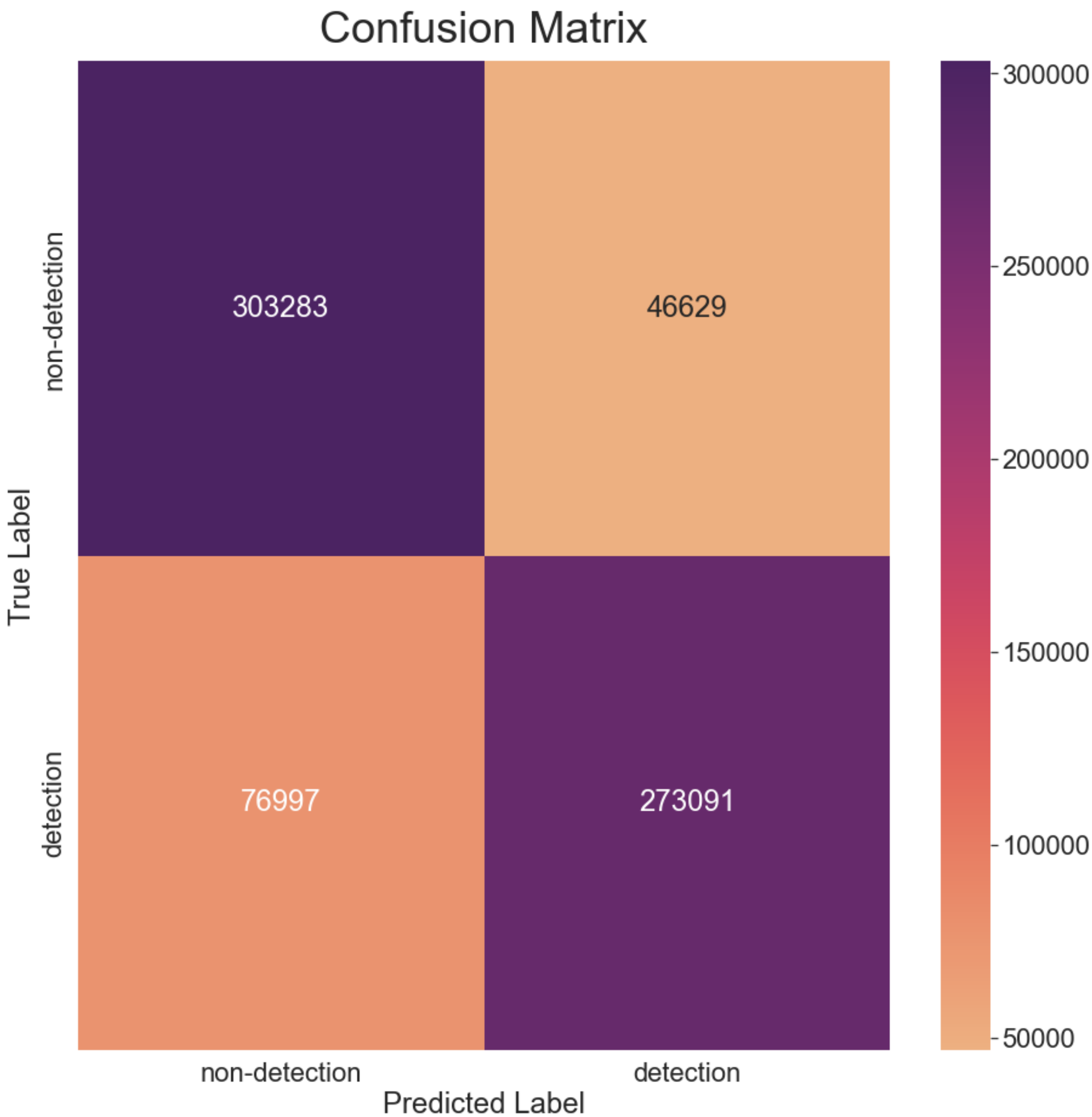
Out[43]:
```
[0.4331013262271881, 0.8233914375305176]
```

In [44]:
```
predictions = (model.predict(X_test) > 0.5).astype("int32")
get_classification_report(X_train,
                          y_train,
                          X_test,
                          y_test,
                          predictions,
                          model)
```

```
21875/21875 [==============================] - 11s 498us/step
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.87      0.83    349912
           1       0.85      0.78      0.82    350088

    accuracy                           0.82    700000
   macro avg       0.83      0.82      0.82    700000
weighted avg       0.83      0.82      0.82    700000
```


Confusion Matrix
True Label
non-detection
detection
303283
46629
76997
273091
Predicted Label
300000
250000
200000
150000
100000
50000