FAKE INSTAGRAM ACCOUNT DETECTION USING NEURAL NETWORKS AND MACHINE LEARNING

PSY 5018H

BY SHUBHAVI ARYA

aryax014@umn.edu

LITERATURE REVIEW

This section discusses the recent work that has been done on the issue of detecting fake accounts on various social media networks. The purpose of this paper is to identify distinct features specific to fake user accounts in contrast with accounts of real users on various social media networks. Social media is used by billions of people these days and has become a part of daily life.

Alece Frunze and Aleksey Frolov (2021) discuss the identification of fake users on the social media network VK (2). This study used several machine learning methods to classify fake accounts including gaussian naive bayes, bernoulli naive bayes, support vector machine, decision tree method, random forest method, MLP and sequential neural network. Random forest algorithm provided the best outcome for detecting fake account with an accuracy of 97%. However, gaussian naive bayes was better at detecting fake accounts like ARU but had a high margin of error. This study suggested that such research can be particularly helpful for researchers to filter fake accounts for various kinds of social media research, for advertisers to be able to verify genuine communities to target as well as for general users to be cautious and avoid harm from fraud.

Buket Erşahin, Özlem Aktaş, Deniz Kılınç and Ceyhun Akyol (2017) conducted a study to detect fake accounts on Twitter (3). They used a classification method from machine learning techniques to detect fake accounts on Twitter. This study used Entropy Minimization Discretization (EMD), a supervised discretization technique, on numerical features and analyzed the results of the Naive Bayes algorithm. With EMD, they used Minimal Description Length (MDL) as the stopping criteria. They were able to get an accuracy of 90.41% with Naive Bayes and preprocessing the date with the discretization technique had a significant effect on improving the accuracy. By using discretization, this study was able to solve the issue of normality assumption of continuous data of social media datasets used by Naive Bayes (3). The study suggested that feature selection and analyzing the contents of posts on Twitter can improve the accuracy for fake user detection. The study also stated that fake accounts often have repeated content and repeated links for advertising and other purposes and using similarity algorithms can be particularly beneficial for improving the accuracy of fake account detection. The study also suggested that feature selection can also improve the results.

Cao Xiao, David Freeman and Theodore Hwa (2015) conducted a study to detect clusters of fake accounts in online social media networks, particularly LinkedIn (4). The study described a scalable approach to identify groups of fake accounts created by the same individual. The study used a supervised machine learning procedure for calssifying clusters of accounts as fake or real. The main features used in the classification process were user name, email address, work place or university, frequencies of patterns within the clusters such as if all the emails included a common letter/digit pattern as well as comparison of text frequencies for the previous features across the entire user data. The study was able to achieve a AUC of 0.98 on held-out test data and a AUC of 0.95 on out-of-sample testing data.

Estee Van Der Walt and Jan Elof (2017) conducted a study using machine learning to detect fake identities on social media and compared the detection performance by a bot from a human (5). The study found that even though they were dealing with human user accounts, the profiles had characteristics specific to bots such as a URL in the profile description. Outliers on the distribution of length of user names of accounts indicated potential fake accounts. The study tried to use techniques that have been used to detect fake accounts generated by bots to detect fake accounts generated by humans but was not similarly successful in this approach.

Stefano Cresci, Roberto Di Pietro,Marinella Petrocchi, Angelo Spognardi and Maurizio Tesconi (2015) created a method for efficient detection of fake Twitter followers (6). Fake follower accounts on Twitter increase the number of followers for users and create artificial and fake popularity on Twitter for the target accounts which has a direct impact on politics, economy and the society at large. The authors of this study trained some machine-learning classifiers according to a set of rules and features and were able to get a 95% accuracy. The authors also performed an information fusion-based sensitivity analysis to assess the global sensitivity of each of the features employed by the classifier (6).

Social media helps people to communicate, share ideas, learn and meet new individuals. Social media is also increasingly used for business and advertising. Fake user accounts are a serious concern and can adversely affect people's trust and spreading of fake information as well as lead to terrorist recruitement and suicide. It is important to therefore continue to work on and develop more better and efficient algorithms to stop the creation of fake accounts and minimize their effects.

PROBLEM STATEMENT AND BUSINESS CASE

Fake accounts are a major problem on social media. Many social media influencers use fake instagram accounts to create an illusion to have an increased number of followers (1). Fake accounts could be used to impersonate other people and also sell fake products and services (1).

In this project, I will be using python programming and machine learning to create an instagram fake profile detector. I will be building and training a deep neural network model to detect fake instagram accounts.

The criteria we will be using to consider whether an instagram account is fake or not will depend on the following factors:

- **Profile Picture: If the account has a random profile picture or does not have a profile picture, it is likely that the account is a fake account.**
- **Full Name: If the account doesn't have a full name, it is likely that the account is a fake account**
- **Description Length: If the account's bio or description is highly fabricated or randomly generated, then it is likely that the account is a fake account**
- **Private: If an account seems shady and is also private, it is likely that the account is a fake account.**
- **Posts: If any accounts that already seems shady also doesn't have any pictures or posts, it is likely that the account is a fake account**
- **Followers: Instagram is generally used to share posts with friends and family and then to also extend to other followers in the instagram community. If there is an account with no followers or random followers with no consistent people commenting or engaging with the posts on the account, it is likely that the account is a fake account.**

We are going to mark each of the above criteria satisfied by the account as 1 and if it does not meet the criteria above, we will mark that criteria as 0.

I will be using 2 datasets that I have found on the internet - one for training the neural model and other for testing our neural model.

THE PROJECT

IMPORT DATASETS AND LIBRARIES

```
In [ ]:
```

```
# !pip install tensorflow==2.0
```

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import Accuracy

from sklearn import metrics
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report,accuracy_score,roc_curve,confusion_matrix
```

In [ ]:

```
from jupyterthemes import jtplot
jtplot.style(theme='monokai', context='notebook', ticks=True, grid=False)
# setting the style of the notebook to be monokai theme
```

In [ ]:

```
# Load the training dataset
instagram_df_train = pd.read_csv('insta_train.csv')
instagram_df_train
```

Out[ ]:

	profile pic	nums/length username	fullname words	nums/length fullname	name==username	description length	external URL	private	#posts	#followers	#follows	fake
0	1	0.27	0	0.00	0	53	0	0	32	1000	955	0
1	1	0.00	2	0.00	0	44	0	0	286	2740	533	0
2	1	0.10	2	0.00	0	0	0	1	13	159	98	0
3	1	0.00	1	0.00	0	82	0	0	679	414	651	0
4	1	0.00	2	0.00	0	0	0	1	6	151	126	0
...	...	...	...	...	...	...	...	...	...	...	...	...
571	1	0.55	1	0.44	0	0	0	0	33	166	596	1
572	1	0.38	1	0.33	0	21	0	0	44	66	75	1
573	1	0.57	2	0.00	0	0	0	0	4	96	339	1
574	1	0.57	1	0.00	0	11	0	0	0	57	73	1
575	1	0.27	1	0.00	0	0	0	0	2	150	487	1

576 rows × 12 columns

In [ ]:

```
# Load the testing data
instagram_df_test = pd.read_csv('insta_test.csv')
instagram_df_test
```

Out[ ]:

	profile pic	nums/length username	fullname words	nums/length fullname	name==username	description length	external URL	private	#posts	#followers	#follows	fake
0	1	0.33	1	0.33	1	30	0	1	35	488	604	0
1	1	0.00	5	0.00	0	64	0	1	3	35	6	0
2	1	0.00	2	0.00	0	82	0	1	319	328	668	0
3	1	0.00	1	0.00	0	143	0	1	273	14890	7369	0
4	1	0.50	1	0.00	0	76	0	1	6	225	356	0
...	...	...	...	...	...	...	...	...	...	...	...	...
115	1	0.29	1	0.00	0	0	0	0	13	114	811	1
116	1	0.40	1	0.00	0	0	0	0	4	150	164	1
117	1	0.00	2	0.00	0	0	0	0	3	833	3572	1
118	0	0.17	1	0.00	0	0	0	0	1	219	1695	1
119	1	0.44	1	0.00	0	0	0	0	3	39	68	1

120 rows × 12 columns

EXPLORATORY DATA ANALYSIS

In [ ]:

```
# Getting dataframe info
instagram_df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 576 entries, 0 to 575
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   profile pic           576 non-null    int64
 1   nums/length username  576 non-null    float64
```

```
 2   fullname words         576 non-null    int64
 3   nums/length fullname   576 non-null    float64
 4   name==username         576 non-null    int64
 5   description length     576 non-null    int64
 6   external URL           576 non-null    int64
 7   private                576 non-null    int64
 8   #posts                 576 non-null    int64
 9   #followers             576 non-null    int64
 10  #follows               576 non-null    int64
 11  fake                   576 non-null    int64
dtypes: float64(2), int64(10)
memory usage: 54.1 KB
```

In [ ]:

```
# Get the statistical summary of the dataframe
instagram_df_train.describe()
```

Out[ ]:

	profile pic	nums/length username	fullname words	nums/length fullname	name==username	description length	external URL	private	#posts	#followers	#follows	fake
count	576.000000	576.000000	576.000000	576.000000	576.000000	576.000000	576.000000	576.000000	576.000000	5.760000e+02	576.000000	576.000000
mean	0.701389	0.163837	1.460069	0.036094	0.034722	22.623264	0.116319	0.381944	107.489583	8.530724e+04	508.381944	0.500000
std	0.458047	0.214096	1.052601	0.125121	0.183234	37.702987	0.320886	0.486285	402.034431	9.101485e+05	917.981239	0.500435
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000e+00	0.000000	0.000000
25%	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	3.900000e+01	57.500000	0.000000
50%	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	9.000000	1.505000e+02	229.500000	0.500000
75%	1.000000	0.310000	2.000000	0.000000	0.000000	34.000000	0.000000	1.000000	81.500000	7.160000e+02	589.500000	1.000000
max	1.000000	0.920000	12.000000	1.000000	1.000000	150.000000	1.000000	1.000000	7389.000000	1.533854e+07	7500.000000	1.000000

In [ ]:

```
# Checking if null values exist
instagram_df_train.isnull().sum()
```

Out[ ]:

```
profile pic             0
nums/length username    0
fullname words          0
nums/length fullname    0
name==username          0
description length      0
external URL            0
private                 0
#posts                  0
#followers              0
#follows                0
fake                    0
dtype: int64
```

In [ ]:

```
# Get the number of unique values in the "profile pic" feature
instagram_df_train['profile pic'].value_counts()
```

Out[ ]:

```
1    404
0    172
Name: profile pic, dtype: int64
```

In [ ]:

```
# Get the number of unique values in "fake" (Target column)
instagram_df_train['fake'].value_counts()
```

Out[ ]:

```
1    288
0    288
Name: fake, dtype: int64
```

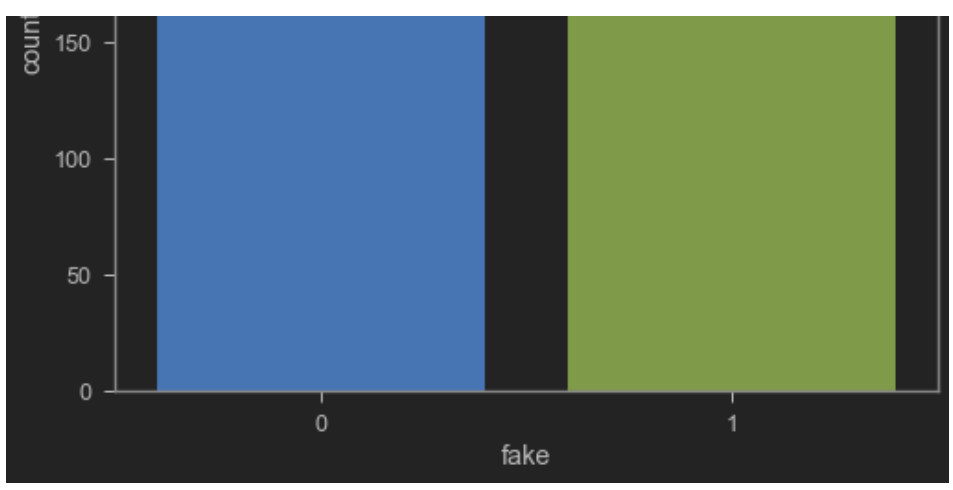
DATA VISUALIZATION

In [ ]:

```
# Visualize the data
sns.countplot(instagram_df_train['fake'])
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x14bcddd3148>
```

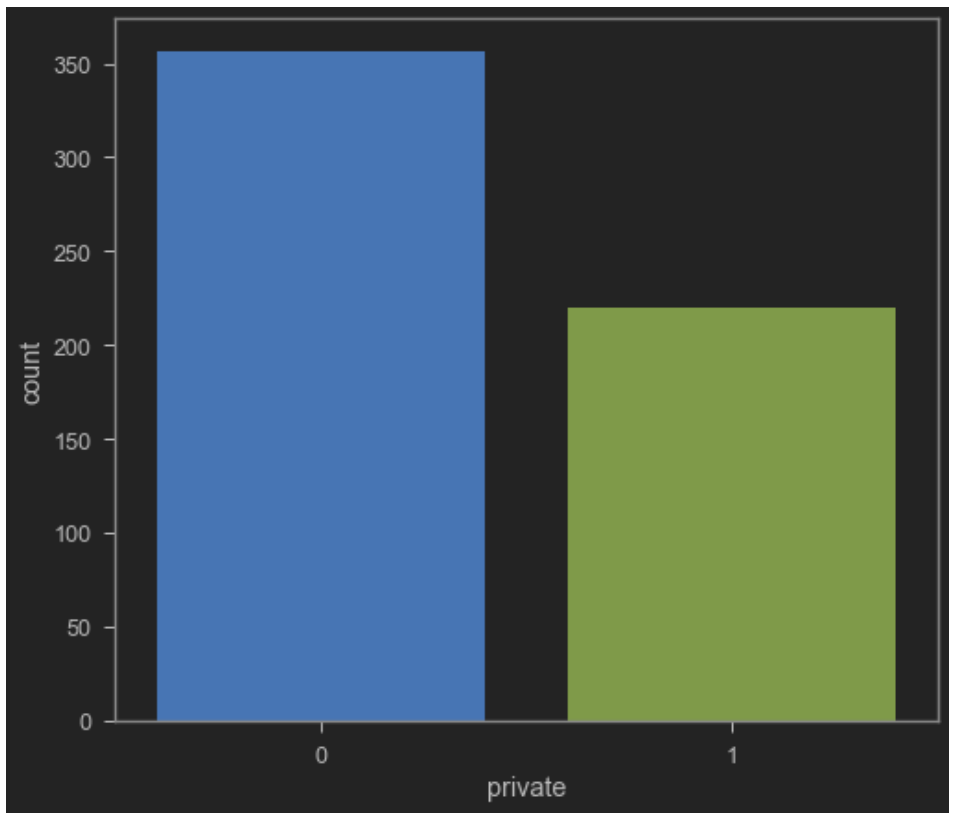


In [ ]:

```
# Visualize the private column data
sns.countplot(instagram_df_train['private'])
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x14bce6701c8>
```

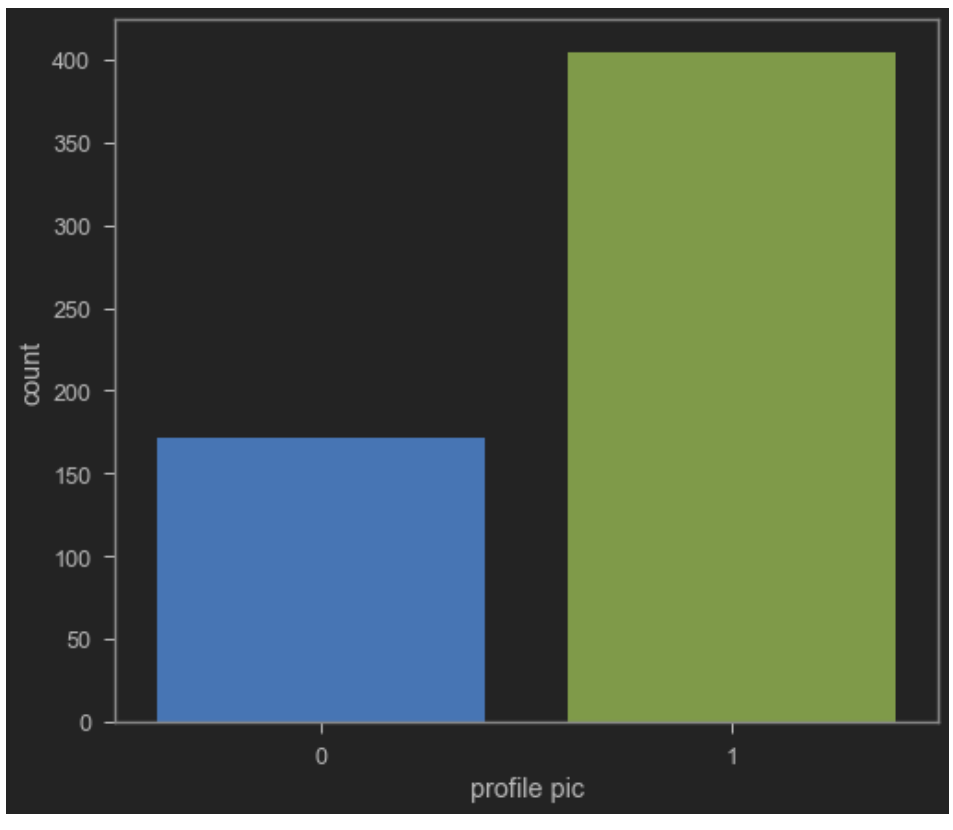


In [ ]:

```
# Visualize the "profile pic" column data
sns.countplot(instagram_df_train['profile pic'])
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x14bce579208>
```

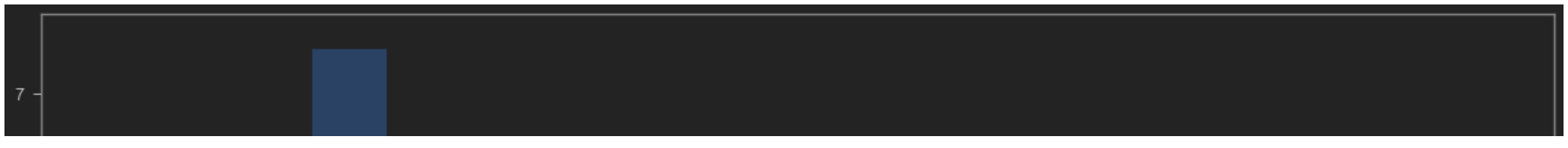


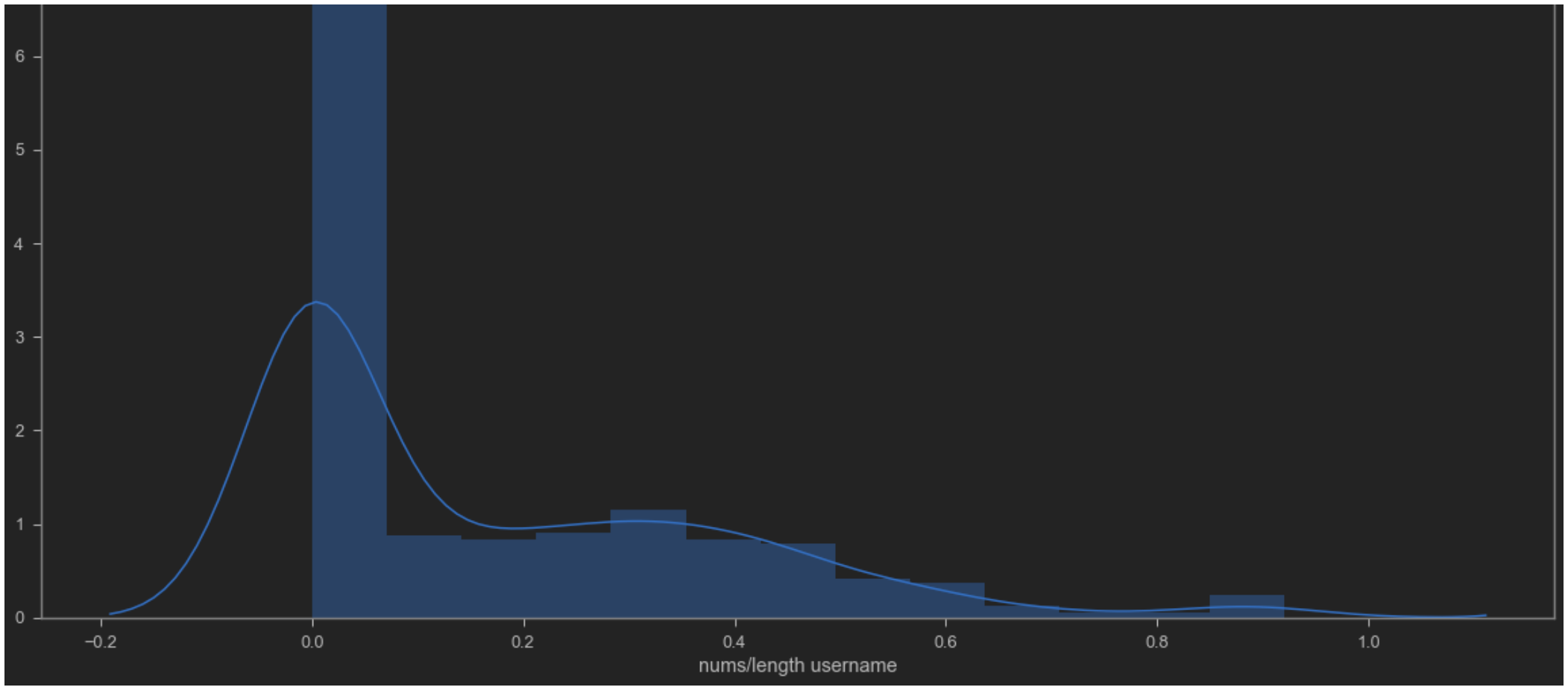
In [ ]:

```
# Visualize the data
plt.figure(figsize = (20, 10))
sns.distplot(instagram_df_train['nums/length username'])
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x14bce5b3fc8>
```





In [ ]:

```
# Correlation plot
plt.figure(figsize=(20, 20))
cm = instagram_df_train.corr()
ax = plt.subplot()
sns.heatmap(cm, annot = True, ax = ax)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x14bceadbf48>
```

PREPARING THE DATA TO FEED THE MODEL

In [ ]:

```
# Training and testing dataset (inputs)
X_train = instagram_df_train.drop(columns = ['fake'])
X_test = instagram_df_test.drop(columns = ['fake'])
X_train
```

Out[ ]:

	profile pic	nums/length username	fullname words	nums/length fullname	name==username	description length	external URL	private	#posts	#followers	#follows
0	1	0.27	0	0.00	0	53	0	0	32	1000	955
1	1	0.00	2	0.00	0	44	0	0	286	2740	533
2	1	0.10	2	0.00	0	0	0	1	13	159	98
3	1	0.00	1	0.00	0	82	0	0	679	414	651
4	1	0.00	2	0.00	0	0	0	1	6	151	126
...	...	...	...	...	...	...	...	...	...	...	...
571	1	0.55	1	0.44	0	0	0	0	33	166	596
572	1	0.38	1	0.33	0	21	0	0	44	66	75
573	1	0.57	2	0.00	0	0	0	0	4	96	339
574	1	0.57	1	0.00	0	11	0	0	0	57	73
575	1	0.27	1	0.00	0	0	0	0	2	150	487

576 rows × 11 columns

In [ ]:

```
X_test
```

Out[ ]:

	profile pic	nums/length username	fullname words	nums/length fullname	name==username	description length	external URL	private	#posts	#followers	#follows
0	1	0.33	1	0.33	1	30	0	1	35	488	604
1	1	0.00	5	0.00	0	64	0	1	3	35	6
2	1	0.00	2	0.00	0	82	0	1	319	328	668
3	1	0.00	1	0.00	0	143	0	1	273	14890	7369
4	1	0.50	1	0.00	0	76	0	1	6	225	356
...	...	...	...	...	...	...	...	...	...	...	...
115	1	0.29	1	0.00	0	0	0	0	13	114	811
116	1	0.40	1	0.00	0	0	0	0	4	150	164
117	1	0.00	2	0.00	0	0	0	0	3	833	3572
118	0	0.17	1	0.00	0	0	0	0	1	219	1695
119	1	0.44	1	0.00	0	0	0	0	3	39	68

120 rows × 11 columns

In [ ]:

```
# Training and testing dataset (Outputs)
y_train = instagram_df_train['fake']
y_test = instagram_df_test['fake']
```

In [ ]:

```
y_train
```

Out[ ]:

```
0      0
1      0
2      0
3      0
4      0
      ..
571    1
572    1
573    1
574    1
575    1
Name: fake, Length: 576, dtype: int64
```

In [ ]:

```
y_test
```

Out[ ]:

```
0      0
1      0
2      0
3      0
4      0
      ..
115    1
116    1
117    1
118    1
119    1
Name: fake, Length: 120, dtype: int64
```

In [ ]:

```python
# Scale the data before training the model
from sklearn.preprocessing import StandardScaler, MinMaxScaler

scaler_x = StandardScaler()
X_train = scaler_x.fit_transform(X_train)
X_test = scaler_x.transform(X_test)
```

In [ ]:

```python
y_train = tf.keras.utils.to_categorical(y_train, num_classes = 2)
y_test = tf.keras.utils.to_categorical(y_test, num_classes = 2)
```

In [ ]:

```
y_train
```

Out[ ]:

```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       ...,
       [0., 1.],
       [0., 1.],
       [0., 1.]], dtype=float32)
```

In [ ]:

```
y_test
```

Out[ ]:

```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
```

```
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.]], dtype=float32)
```

In [ ]:

```
Training_data = len(X_train)/( len(X_test) + len(X_train) ) * 100
Training_data
```

Out[ ]:

```
82.75862068965517
```

82.8% is training data.

In [ ]:

```
Testing_data = len(X_test)/( len(X_test) + len(X_train) ) * 100
Testing_data
```

Out[ ]:

```
17.24137931034483
```

17.2% is testing data.

BUILDING AND TRAINING A SIMPLE DEEP LEARNING MODEL

In [ ]:

```
import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
```

```
model = Sequential()
model.add(Dense(50, input_dim=11, activation='relu'))
model.add(Dense(150, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(25, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(25, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(2, activation='softmax'))
model.summary()

# Trainable params: 12,727
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 50)                600
_________________________________________________________________
dense_1 (Dense)              (None, 150)               7650
_________________________________________________________________
dropout (Dropout)            (None, 150)               0
_________________________________________________________________
dense_2 (Dense)              (None, 25)                3775
_________________________________________________________________
dropout_1 (Dropout)          (None, 25)                0
_________________________________________________________________
dense_3 (Dense)              (None, 25)                650
_________________________________________________________________
dropout_2 (Dropout)          (None, 25)                0
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 52
=================================================================
Total params: 12,727
Trainable params: 12,727
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

In [ ]:

```
epochs_hist = model.fit(X_train, y_train, epochs = 20,  verbose = 1, validation_split = 0.1)
```

```
Train on 518 samples, validate on 58 samples
Epoch 1/20
518/518 [==============================] - 1s 2ms/sample - loss: 0.6262 - accuracy: 0.7008 - val_loss: 0.6031 - val_accuracy: 0.72
41
Epoch 2/20
518/518 [==============================] - 0s 187us/sample - loss: 0.4862 - accuracy: 0.8263 - val_loss: 0.4284 - val_accuracy: 0.
7931
Epoch 3/20
518/518 [==============================] - 0s 191us/sample - loss: 0.3756 - accuracy: 0.8842 - val_loss: 0.2876 - val_accuracy: 0.
8276
Epoch 4/20
518/518 [==============================] - 0s 195us/sample - loss: 0.3148 - accuracy: 0.9054 - val_loss: 0.2265 - val_accuracy: 0.
8793
Epoch 5/20
518/518 [==============================] - 0s 203us/sample - loss: 0.2853 - accuracy: 0.9131 - val_loss: 0.1760 - val_accuracy: 0.
9138
Epoch 6/20
518/518 [==============================] - 0s 189us/sample - loss: 0.2945 - accuracy: 0.8938 - val_loss: 0.2075 - val_accuracy: 0.
9138
Epoch 7/20
518/518 [==============================] - 0s 176us/sample - loss: 0.2664 - accuracy: 0.9112 - val_loss: 0.1944 - val_accuracy: 0.
9138
Epoch 8/20
518/518 [==============================] - 0s 178us/sample - loss: 0.2452 - accuracy: 0.9093 - val_loss: 0.1378 - val_accuracy: 0.
9310
Epoch 9/20
518/518 [==============================] - 0s 205us/sample - loss: 0.2658 - accuracy: 0.9054 - val_loss: 0.1881 - val_accuracy: 0.
9138
Epoch 10/20
518/518 [==============================] - 0s 199us/sample - loss: 0.2474 - accuracy: 0.9015 - val_loss: 0.1615 - val_accuracy: 0.
9138
Epoch 11/20
518/518 [==============================] - 0s 189us/sample - loss: 0.2513 - accuracy: 0.9247 - val_loss: 0.1522 - val_accuracy: 0.
9310
Epoch 12/20
518/518 [==============================] - 0s 199us/sample - loss: 0.2454 - accuracy: 0.9228 - val_loss: 0.1676 - val_accuracy: 0.
9138
Epoch 13/20
518/518 [==============================] - 0s 189us/sample - loss: 0.2108 - accuracy: 0.9228 - val_loss: 0.1728 - val_accuracy: 0.
8966
Epoch 14/20
518/518 [==============================] - 0s 207us/sample - loss: 0.2065 - accuracy: 0.9189 - val_loss: 0.1938 - val_accuracy: 0.
8966
Epoch 15/20
518/518 [==============================] - 0s 216us/sample - loss: 0.2098 - accuracy: 0.9266 - val_loss: 0.1850 - val_accuracy: 0.
8966
Epoch 16/20
518/518 [==============================] - 0s 230us/sample - loss: 0.2100 - accuracy: 0.9170 - val_loss: 0.1740 - val_accuracy: 0.
9138
Epoch 17/20
518/518 [==============================] - 0s 199us/sample - loss: 0.1971 - accuracy: 0.9266 - val_loss: 0.1809 - val_accuracy: 0.
9138
Epoch 18/20
518/518 [==============================] - 0s 189us/sample - loss: 0.2106 - accuracy: 0.9208 - val_loss: 0.1806 - val_accuracy: 0.
9138
Epoch 19/20
```

```
518/518 [==============================] - 0s 203us/sample - loss: 0.1891 - accuracy: 0.9247 - val_loss: 0.1634 - val_accuracy: 0.
9138
Epoch 20/20
518/518 [==============================] - 0s 208us/sample - loss: 0.1884 - accuracy: 0.9266 - val_loss: 0.2048 - val_accuracy: 0.
9138
```

We have reached 93% accuracy with this above model in the few minutes we have trained it.

ASSESSING THE PERORMANCE OF TRAINED MODEL

In [ ]:

```
print(epochs_hist.history.keys())
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

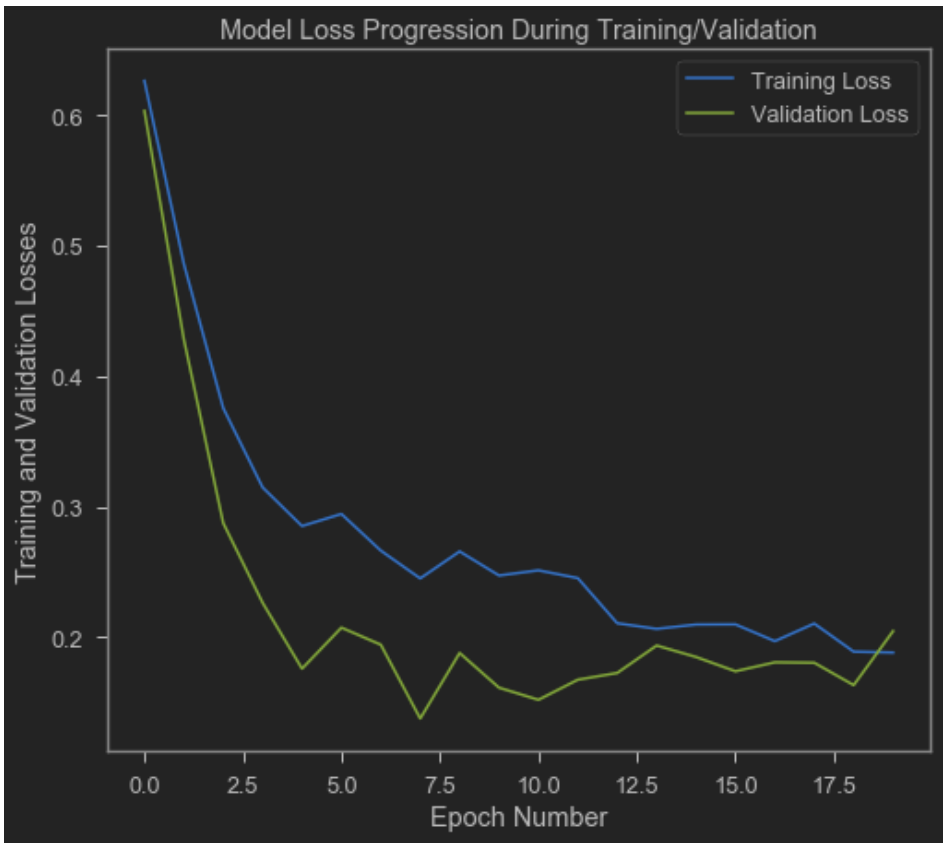
In [ ]:

```
plt.plot(epochs_hist.history['loss'])
plt.plot(epochs_hist.history['val_loss'])

plt.title('Model Loss Progression During Training/Validation')
plt.ylabel('Training and Validation Losses')
plt.xlabel('Epoch Number')
plt.legend(['Training Loss', 'Validation Loss'])
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x14bd0ff6c48>
```



In [ ]:

```
predicted = model.predict(X_test)
```

In [ ]:

```
predicted_value = []
test = []
for i in predicted:
    predicted_value.append(np.argmax(i))

for i in y_test:
    test.append(np.argmax(i))
```

In [ ]:

```
print(classification_report(test, predicted_value))
```

```
              precision    recall  f1-score   support

           0       0.86      0.93      0.90        60
           1       0.93      0.85      0.89        60

    accuracy                           0.89       120
   macro avg       0.89      0.89      0.89       120
weighted avg       0.89      0.89      0.89       120
```

We are able to achieve around 89% accuracy which is pretty good. Other performance indicators are also quite good. We have 86% precision on the 0 class and 93% precision on the 1 class. We have 93% recall for the 0 class and 85% recall for the 1 class. We also have the f1 score which is the harmonic mean between the precision and the recall as 98% for the 0 class and 89% for the 1 class. So, the overall performance is pretty good.

Now, let us print the confusion matrix.

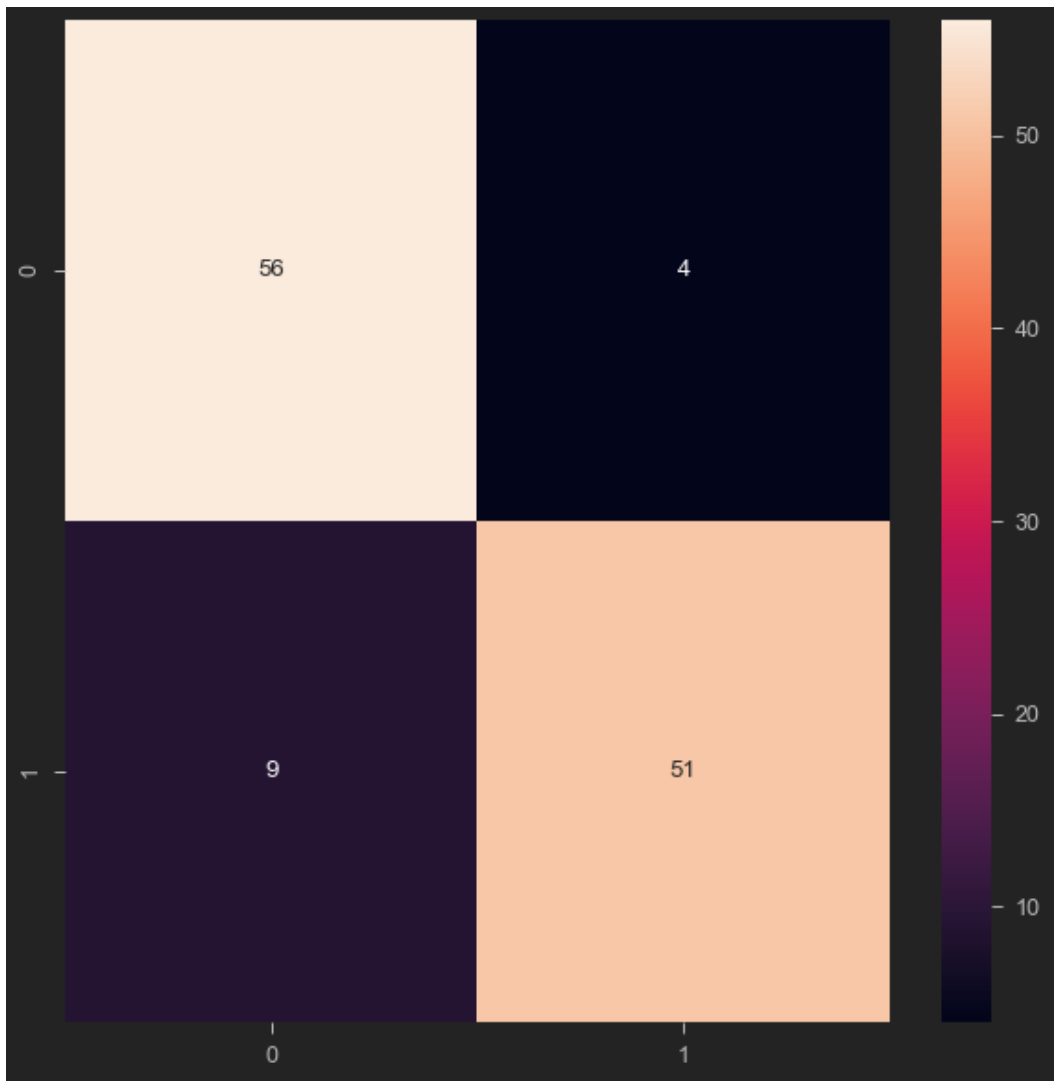
In [ ]:

```
plt.figure(figsize=(10, 10))
```

```
cm = confusion_matrix(test, predicted_value)
sns.heatmap(cm, annot = True)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x14bd0fbe488>
```



The confusion matrix shows that we were able to correctly classify 56 and 51 samples. However, we misclassified 9 and 4 i.e 13 samples. This is quite good performance of our neural model.

REFERENCES

1. https://www.bustle.com/p/how-to-tell-if-instagram-account-is-fake-based-on-these-6-glaring-red-flags-18778777
2. A. D. Frunze and A. A. Frolov, "Methods for Detecting Fake Accounts on the Social Network VK," 2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus), 2021, pp. 342-346, doi: 10.1109/ElConRus51938.2021.9396670.
3. B. Erşahin, Ö. Aktaş, D. Kılınç and C. Akyol, "Twitter fake account detection," 2017 International Conference on Computer Science and Engineering (UBMK), 2017, pp. 388-392, doi: 10.1109/UBMK.2017.8093420.
4. Cao Xiao, David Mandell Freeman, and Theodore Hwa. 2015. Detecting Clusters of Fake Accounts in Online Social Networks. In Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security (AISec '15). Association for Computing Machinery, New York, NY, USA, 91–101. DOI:https://doi-org.ezp3.lib.umn.edu/10.1145/2808769.2808779
5. Van Der Walt, Estee, & Eloff, Jan. (2018). Using Machine Learning to Detect Fake Identities: Bots vs Humans. IEEE Access, 6, 6540-6549.
6. Stefano Cresci, Roberto Di Pietro, Marinella Petrocchi, Angelo Spognardi, Maurizio Tesconi, Fame for sale: Efficient detection of fake Twitter followers, Decision Support Systems, Volume 80, 2015, Pages 56-71, ISSN 0167-9236, https://doi.org/10.1016/j.dss.2015.09.003.