Q2) Explore and implement a DNN architecture for the image/signal classification. You are free to explore on any open-source datasets related to signal/image classification

In this we have used Gender classfication image data set, it will show wether it is men or women

data link :https://www.kaggle.com/cashutosh/gender-classification-dataset (https://www.kaggle.com/cashutosh/gender-classification-dataset)

In [287]:

```
#import required libraries
import numpy as np
import pandas as pd
import os
import pathlib
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing import image
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from keras.layers import Conv2D,MaxPool2D,Dropout,Flatten,Dense
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import seaborn as sns
```

In [288]:

```
# Set Parameters
batch_size = 256
img_height = 256
img_width = 256
```

In [289]:

```
# loading data using keras
# training, validation

train_data = pathlib.Path("Gender_Classification/Training")
validation_data = pathlib.Path("Gender_Classification/Validation")
```

In [290]:

```
train_ds = tf.keras.utils.image_dataset_from_directory(
    train_data,
    image_size=(img_height, img_width),
    batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(
    validation_data,
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

```
Found 47009 files belonging to 2 classes.
Found 11649 files belonging to 2 classes.
```

In [293]:

```
print(train_ds.class_names)#labels of images from train
```

```
['female', 'male']
```

In [294]:

```
model = Sequential([

    layers.Flatten(input_shape = (img_height,img_width,3)), #flatten image input
    #hidden layer 1
    layers.Dense(64, activation='relu'),
    #hidden layer 2
    layers.Dense(80, activation='relu'),
    #hidden layer 3
    layers.Dense(30,activation='relu'),
    #hidden layer 4
    layers.Dense(10,activation='relu'),

    layers.Dense(2, activation = 'softmax') #output layer
])
```

In [295]:

```
# setting hyperparameters
model.compile(optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'])
```

In [296]:

```
#number of iterations
epoch = 10
history = model.fit(train_ds,epochs=epoch, validation_data=val_ds)
```

```
Epoch 1/10
184/184 [==============================] - 240s 1s/step - loss: 161.2794 - a
ccuracy: 0.6595 - val_loss: 9.3489 - val_accuracy: 0.7969
Epoch 2/10
184/184 [==============================] - 222s 1s/step - loss: 34.5108 - ac
curacy: 0.7167 - val_loss: 6.1595 - val_accuracy: 0.8305
Epoch 3/10
184/184 [==============================] - 228s 1s/step - loss: 9.3001 - acc
uracy: 0.7855 - val_loss: 8.1991 - val_accuracy: 0.7848
Epoch 4/10
184/184 [==============================] - 231s 1s/step - loss: 7.0084 - acc
uracy: 0.8150 - val_loss: 13.7933 - val_accuracy: 0.6521
Epoch 5/10
184/184 [==============================] - 241s 1s/step - loss: 7.7587 - acc
uracy: 0.8013 - val_loss: 4.5833 - val_accuracy: 0.8619
Epoch 6/10
184/184 [==============================] - 251s 1s/step - loss: 5.2469 - acc
uracy: 0.8302 - val_loss: 3.3783 - val_accuracy: 0.8476
Epoch 7/10
184/184 [==============================] - 266s 1s/step - loss: 2.8661 - acc
uracy: 0.8587 - val_loss: 3.0860 - val_accuracy: 0.8319
Epoch 8/10
184/184 [==============================] - 272s 1s/step - loss: 4.0076 - acc
uracy: 0.8278 - val_loss: 4.8536 - val_accuracy: 0.7690
Epoch 9/10
184/184 [==============================] - 277s 1s/step - loss: 3.1721 - acc
uracy: 0.8349 - val_loss: 6.9427 - val_accuracy: 0.6983
Epoch 10/10
184/184 [==============================] - 275s 1s/step - loss: 2.0283 - acc
uracy: 0.8610 - val_loss: 1.6017 - val_accuracy: 0.8747
```

In [316]:

```
pred = np.array([])
labels =  np.array([])
for x, y in val_ds:
    pred = np.concatenate([pred, np.argmax(model.predict(x), axis = -1)])
    labels = np.concatenate([labels, y.numpy()])
```

In [319]:

```
print('accuracy',accuracy_score(labels,pred))
print(model.evaluate(val_ds))
```

```
accuracy 0.8746673534208945
46/46 [==============================] - 57s 1s/step - loss: 1.6017 - accura
cy: 0.8747
[1.6017372608184814, 0.8746673464775085]
```

In [326]:

```
print(classification_report(labels,pred))
```

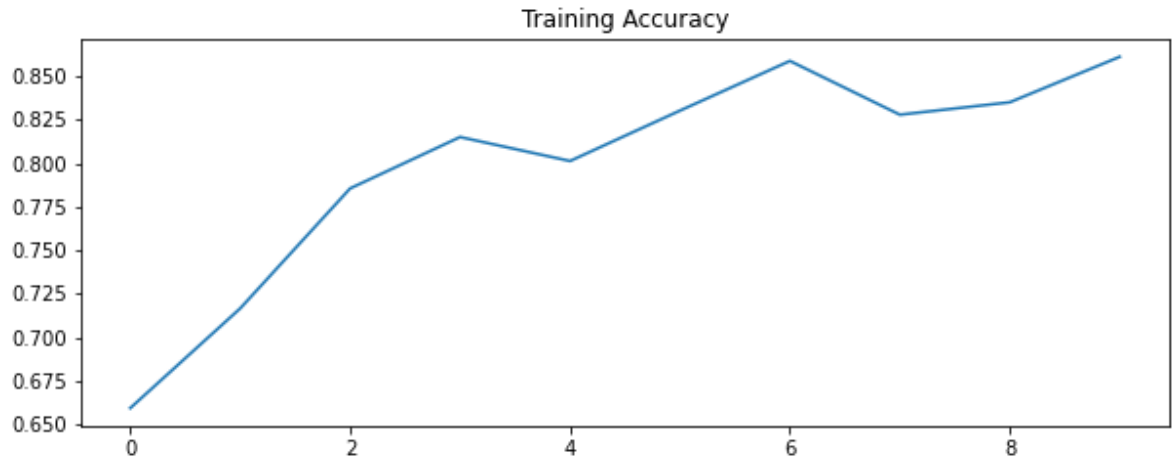
```
              precision    recall  f1-score   support

         0.0       0.82      0.97      0.89      5841
         1.0       0.96      0.78      0.86      5808

    accuracy                           0.87     11649
   macro avg       0.89      0.87      0.87     11649
weighted avg       0.89      0.87      0.87     11649
```

In [331]:

```
acc = history.history['accuracy']
plt.figure(figsize=(10, 8))
plt.subplot(2, 1, 1)
plt.title('Training Accuracy')
plt.plot(acc)
```

Out[331]:

```
[<matplotlib.lines.Line2D at 0x1671db3aca0>]
```

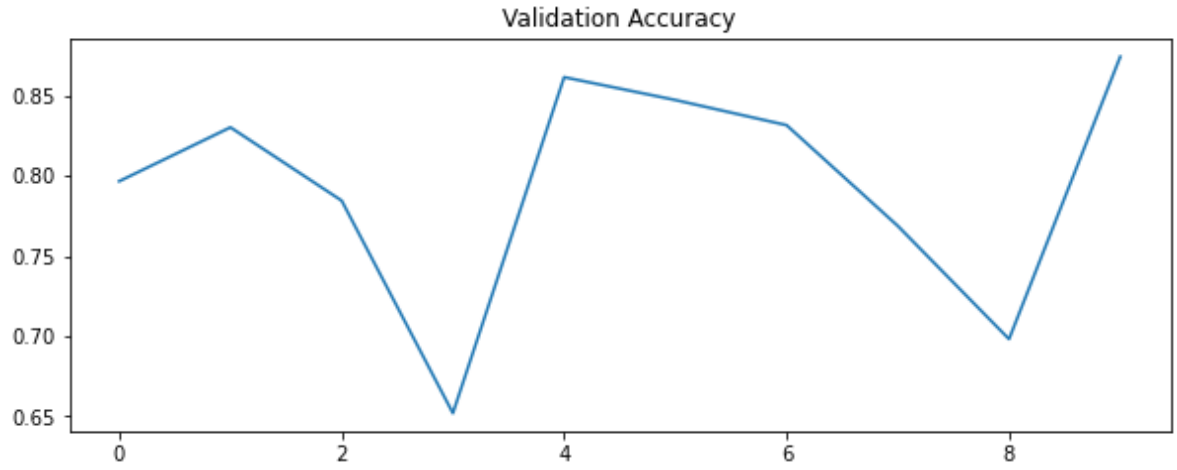


In [328]:

```
val_acc = history.history['val_accuracy']
plt.figure(figsize=(10, 8))
plt.subplot(2, 1, 1)
plt.title('Validation Accuracy')
plt.plot(val_acc)
```

Out[328]:

```
[<matplotlib.lines.Line2D at 0x1671d9ea340>]
```

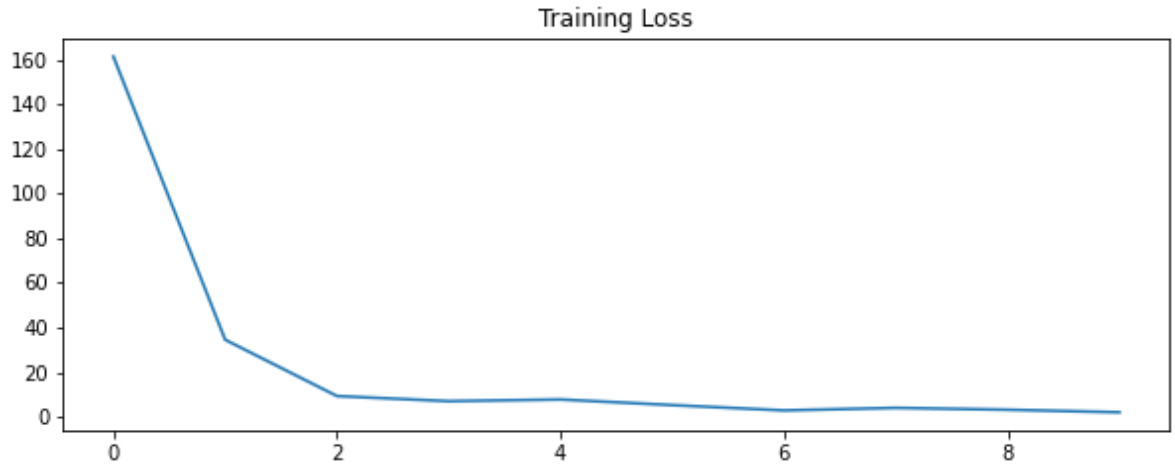


In [329]:

```
loss = history.history['loss']
plt.figure(figsize=(10, 8))
plt.subplot(2, 1, 1)
plt.title('Training Loss')
plt.plot(loss)
```

Out[329]:

```
[<matplotlib.lines.Line2D at 0x1671d982670>]
```

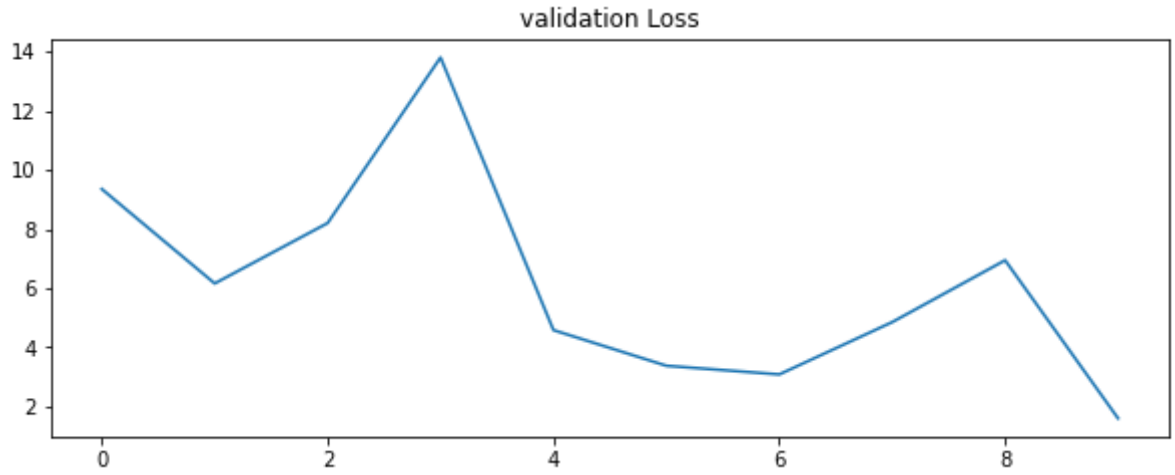


In [333]:

```
loss = history.history['loss']
plt.figure(figsize=(10, 8))
plt.subplot(2, 1, 1)
plt.title('validation Loss')
plt.plot(val_loss)
```

Out[333]:

```
[<matplotlib.lines.Line2D at 0x1671d6dd8e0>]
```



In [ ]: