Name: Pratiksha Rale

Seat No: B224011

Project Name: Fake Signature Detection using Machine learning & Deep Learning

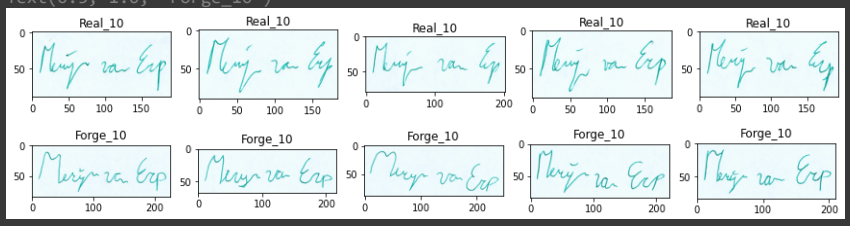
Importing the necessary libraries

```
import pandas as pd
import numpy as np
import skimage.io as sk
from skimage import img_as_ubyte
from skimage.io import imread
from scipy import spatial
from tensorflow.keras.layers import Dense, Flatten, Input, Lambda, MaxPooling2D, Conv2D, Dropout, BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
from keras.models import Sequential
from glob import glob
from PIL import Image
import cv2
import matplotlib.pyplot as plt
```

Displaying the of Image of Forge and Real Signature

```
image1 = sk.imread("C:/Users/Admin/Real-Forge-Signature-Detection-main/Train/real/001001_000.png")
image2 = sk.imread("C:/Users/Admin/Real-Forge-Signature-Detection-main/Train/real/001001_001.png")
image3 = sk.imread("C:/Users/Admin/Real-Forge-Signature-Detection-main/Train/real/001001_002.png")
image4 = sk.imread("C:/Users/Admin/Real-Forge-Signature-Detection-main/Train/real/001001_003.png")
image5 = sk.imread("C:/Users/Admin/Real-Forge-Signature-Detection-main/Train/real/001001_004.png")

fig, ax = plt.subplots(1,5, figsize = (15,10))

ax[0].imshow(image1)
ax[0].set_title("Real_10")
ax[1].imshow(image2)
ax[1].set_title("Real_10")
ax[2].imshow(image3)
ax[2].set_title("Real_10")
ax[3].imshow(image4)
ax[3].set_title("Real_10")
ax[4].imshow(image5)
ax[4].set_title("Real_10")

image6 = sk.imread("C:/Users/Admin/Real-Forge-Signature-Detection-main/Train/forged/021001_000.png")
image7 = sk.imread("C:/Users/Admin/Real-Forge-Signature-Detection-main/Train/forged/021001_001.png")
image8 = sk.imread("C:/Users/Admin/Real-Forge-Signature-Detection-main/Train/forged/021001_002.png")
image9 = sk.imread("C:/Users/Admin/Real-Forge-Signature-Detection-main/Train/forged/021001_003.png")
image10 = sk.imread("C:/Users/Admin/Real-Forge-Signature-Detection-main/Train/forged/021001_004.png")

fig, ax1 = plt.subplots(1,5, figsize = (15,10))

ax1[0].imshow(image6)
ax1[0].set_title("Forge_10")
ax1[1].imshow(image7)
ax1[1].set_title("Forge_10")
ax1[2].imshow(image8)
ax1[2].set_title("Forge_10")
ax1[3].imshow(image9)
ax1[3].set_title("Forge_10")
ax1[4].imshow(image10)
ax1[4].set_title("Forge_10")
```

```
Text(0.5, 1.0, 'Forge_10')
```



Data Acquisition

Handwritten signatures are **collected** and some **unique features** are **extracted** to **create knowledge base for each and every individual**. A **standard database** of **signatures** for every **individual** is **needed** for **evaluating performance** of the **signature verification system** and **also for comparing the result obtained using other techniques' on the same database**.

Data Pre-Processing & Model Building

```
train_path = 'C:/Users/Admin/Real-Forge-Signature-Detection-main/Train'
test_path = 'C:/Users/Admin/Real-Forge-Signature-Detection-main/Test'
```

```
Image_Width = 512
Image_Height = 512
Image_Size = (Image_Width, Image_Height)
Image_Channel = 3
batch_size=15
```

```
model = Sequential()

## Conv layer 1
model.add(Conv2D(32, (3,3), activation='relu', input_shape=(Image_Width,Image_Height, Image_Channel)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

## Conv layer 2
model.add(Conv2D(64, (3,3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

## Conv layer 3
model.add(Conv2D(128, (3,3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

## Conv layer 4
model.add(Conv2D(256, (3,3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

## Conv layer 5
model.add(Conv2D(256, (3,3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

## Conv layer 6
model.add(Conv2D(512, (3,3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256,activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(2, activation='softmax'))
```

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

```
model.summary()
```

```
conv2d_1 (Conv2D)           (None, 253, 253, 64)      18496

batch_normalization_1 (Batc  (None, 253, 253, 64)     256
hNormalization)

max_pooling2d_1 (MaxPooling  (None, 126, 126, 64)     0
2D)

dropout_1 (Dropout)         (None, 126, 126, 64)      0

conv2d_2 (Conv2D)           (None, 124, 124, 128)     73856

batch_normalization_2 (Batc  (None, 124, 124, 128)    512
hNormalization)

max_pooling2d_2 (MaxPooling  (None, 62, 62, 128)      0
2D)

dropout_2 (Dropout)         (None, 62, 62, 128)       0

conv2d_3 (Conv2D)           (None, 60, 60, 256)       295168

batch_normalization_3 (Batc  (None, 60, 60, 256)      1024
hNormalization)

max_pooling2d_3 (MaxPooling  (None, 30, 30, 256)      0
2D)

dropout_3 (Dropout)         (None, 30, 30, 256)       0

conv2d_4 (Conv2D)           (None, 28, 28, 256)       590080

batch_normalization_4 (Batc  (None, 28, 28, 256)      1024
hNormalization)

max_pooling2d_4 (MaxPooling  (None, 14, 14, 256)      0
2D)

dropout_4 (Dropout)         (None, 14, 14, 256)       0

conv2d_5 (Conv2D)           (None, 12, 12, 512)       1180160

batch_normalization_5 (Batc  (None, 12, 12, 512)      2048
hNormalization)

max_pooling2d_5 (MaxPooling  (None, 6, 6, 512)        0
2D)

dropout_5 (Dropout)         (None, 6, 6, 512)         0

flatten (Flatten)           (None, 18432)             0

dense (Dense)               (None, 256)               4718848

batch_normalization_6 (Batc  (None, 256)              1024
hNormalization)

dropout_6 (Dropout)         (None, 256)               0
```

What are callbacks?

- Callbacks is a powerfull tool which is used to customize the behaviour of keras or tensorflow model during training, evaluaton or inference

```
from keras.callbacks import EarlyStopping, ReduceLROnPlateau

early_stop = EarlyStopping(patience=10)
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc', patience=2, verbose=1, factor=0.5, min_lr=0.00001)
callbacks = [early_stop, learning_rate_reduction]
```

Data Pre-processing

```
# Scaling all the images between 0 to 1 and applying Data Augmentation

train_datagen = ImageDataGenerator(rotation_range=15,
                                   rescale=1./255,
                                   shear_range=0.1,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1,)
```

```
train_generator = train_datagen.flow_from_directory('C:/Users/Admin/Real-Forge-Signature-Detection-main/Train',
                                                    target_size=Image_Size,
                                                    batch_size=32,
                                                    class_mode = 'categorical')
```

```
Found 120 images belonging to 2 classes.
```

```
# Performing only scaling on the test dataset

test_datagen = ImageDataGenerator(rescale=1./255)
```

```
test_generator = test_datagen.flow_from_directory('C:/Users/Admin/Real-Forge-Signature-Detection-main/Test',
                                                  target_size=Image_Size,
                                                  batch_size = 32,
                                                  class_mode='categorical')
```

```
Found 120 images belonging to 2 classes.
```

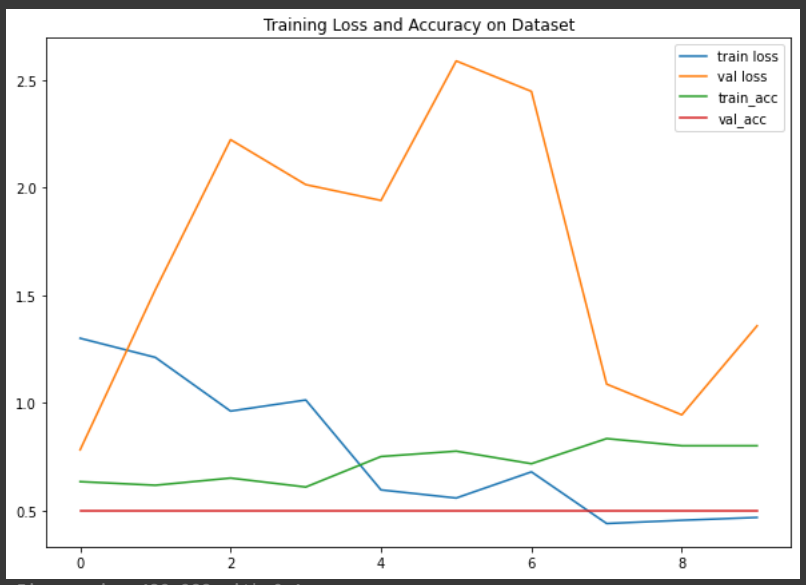
```
epochs = 10

history = model.fit_generator(train_generator,
                              epochs=epochs,
                              validation_data=test_generator,
                              validation_steps=len(test_generator),
                              steps_per_epoch=len(train_generator),
                              callbacks=callbacks)
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_8900\3027511203.py:3: UserWarning: `Model.fit_generator` is deprecated and will be removed i
  history = model.fit_generator(train_generator,
Epoch 1/10
4/4 [==============================] - ETA: 0s - loss: 1.2993 - accuracy: 0.6333  WARNING:tensorflow:Learning rate reduction is conditic
4/4 [==============================] - 777s 218s/step - loss: 1.2993 - accuracy: 0.6333 - val_loss: 0.7811 - val_accuracy: 0.5000 - lr:
Epoch 2/10
4/4 [==============================] - ETA: 0s - loss: 1.2104 - accuracy: 0.6167  WARNING:tensorflow:Learning rate reduction is conditic
4/4 [==============================] - 548s 128s/step - loss: 1.2104 - accuracy: 0.6167 - val_loss: 1.5247 - val_accuracy: 0.5000 - lr:
Epoch 3/10
4/4 [==============================] - ETA: 0s - loss: 0.9609 - accuracy: 0.6500  WARNING:tensorflow:Learning rate reduction is conditic
4/4 [==============================] - 873s 219s/step - loss: 0.9609 - accuracy: 0.6500 - val_loss: 2.2217 - val_accuracy: 0.5000 - lr:
Epoch 4/10
4/4 [==============================] - ETA: 0s - loss: 1.0126 - accuracy: 0.6083  WARNING:tensorflow:Learning rate reduction is conditic
4/4 [==============================] - 809s 192s/step - loss: 1.0126 - accuracy: 0.6083 - val_loss: 2.0128 - val_accuracy: 0.5000 - lr:
Epoch 5/10
4/4 [==============================] - ETA: 0s - loss: 0.5950 - accuracy: 0.7500  WARNING:tensorflow:Learning rate reduction is conditic
4/4 [==============================] - 604s 143s/step - loss: 0.5950 - accuracy: 0.7500 - val_loss: 1.9393 - val_accuracy: 0.5000 - lr:
Epoch 6/10
4/4 [==============================] - ETA: 0s - loss: 0.5572 - accuracy: 0.7750   WARNING:tensorflow:Learning rate reduction is conditi
4/4 [==============================] - 2568s 797s/step - loss: 0.5572 - accuracy: 0.7750 - val_loss: 2.5873 - val_accuracy: 0.5000 - lr:
Epoch 7/10
4/4 [==============================] - ETA: 0s - loss: 0.6788 - accuracy: 0.7167  WARNING:tensorflow:Learning rate reduction is conditic
4/4 [==============================] - 742s 175s/step - loss: 0.6788 - accuracy: 0.7167 - val_loss: 2.4457 - val_accuracy: 0.5000 - lr:
Epoch 8/10
4/4 [==============================] - ETA: 0s - loss: 0.4389 - accuracy: 0.8333  WARNING:tensorflow:Learning rate reduction is conditic
4/4 [==============================] - 726s 170s/step - loss: 0.4389 - accuracy: 0.8333 - val_loss: 1.0864 - val_accuracy: 0.5000 - lr:
Epoch 9/10
4/4 [==============================] - ETA: 0s - loss: 0.4540 - accuracy: 0.8000  WARNING:tensorflow:Learning rate reduction is conditic
4/4 [==============================] - 868s 204s/step - loss: 0.4540 - accuracy: 0.8000 - val_loss: 0.9436 - val_accuracy: 0.5000 - lr:
Epoch 10/10
4/4 [==============================] - ETA: 0s - loss: 0.4671 - accuracy: 0.8000  WARNING:tensorflow:Learning rate reduction is conditic
4/4 [==============================] - 1509s 430s/step - loss: 0.4671 - accuracy: 0.8000 - val_loss: 1.3570 - val_accuracy: 0.5000 - lr:
```

Plotting the Accuracy and Losses

```
plt.figure(figsize=(10,7))
```

```
plt.plot(history.history['loss'], label='train loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.plot(history.history['accuracy'], label='train_acc')
plt.plot(history.history['val_accuracy'], label='val_acc')
plt.title("Training Loss and Accuracy on Dataset")
plt.legend()
plt.show()
plt.savefig('lossval_loss')
```




```
<Figure size 432x288 with 0 Axes>
```

Saving our model

```
from tensorflow.keras.models import load_model

model.save('forge_real_signature_model.h5')
```

Making our prediction with our model

```
pred = model.predict(test_generator)
pred
```

```
           [0.92747235, 0.07252764],
           [0.9298644 , 0.07013559],
           [0.92566067, 0.0743394 ],
           [0.9223653 , 0.0776347 ],
           [0.92722374, 0.07277625],
           [0.9279615 , 0.07203847],
           [0.93262815, 0.0673718 ],
           [0.9297335 , 0.07026652],
           [0.92806417, 0.07193583],
           [0.9266913 , 0.07330873],
           [0.932597  , 0.06740302],
           [0.9274633 , 0.07253671],
           [0.9338528 , 0.06614722],
           [0.92427456, 0.07572537],
           [0.9356428 , 0.06435726],
           [0.9288637 , 0.0711363 ],
           [0.92429096, 0.07570906],
           [0.92471707, 0.07528289],
           [0.92489094, 0.07510906],
           [0.9285509 , 0.07144909],
           [0.9257659 , 0.07423417],
           [0.9265266 , 0.07347334],
           [0.92734754, 0.07265247],
           [0.9239349 , 0.07606518],
           [0.92736375, 0.07263624],
           [0.9294843 , 0.07051568],
           [0.9309777 , 0.06902236],
           [0.92795056, 0.07204948]], dtype=float32)
```

```
import numpy as np

pred = np.argmax(pred, axis=1)
pred
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

Loading our model

```
model = load_model('forge_real_signature_model.h5')
```

```
from tensorflow.keras.preprocessing import image

img = image.load_img('C:/Users/Admin/Real-Forge-Signature-Detection-main/Test/forged/021001_000.png', target_size=(512,512))
```

```
x = image.img_to_array(img)
x
```

```
array([[[242., 252., 253.],
        [242., 252., 253.],
        [241., 252., 253.],
        ...,
        [241., 251., 253.],
        [240., 252., 253.],
        [240., 252., 253.]],

       [[242., 252., 253.],
        [242., 252., 253.],
        [241., 252., 253.],
        ...,
        [241., 251., 253.],
        [240., 252., 253.],
        [240., 252., 253.]],

       [[242., 252., 253.],
        [242., 252., 253.],
        [241., 252., 253.],
        ...,
        [241., 251., 253.],
        [240., 252., 253.],
        [240., 252., 253.]],

       ...,
```

```
             [[243., 252., 253.],
              [243., 252., 253.],
              [244., 252., 252.],
              ...,
              [238., 250., 254.],
              [239., 251., 253.],
              [239., 251., 253.]],

             [[243., 252., 253.],
              [243., 252., 253.],
              [244., 252., 252.],
              ...,
              [238., 250., 254.],
              [239., 251., 253.],
              [239., 251., 253.]],

             [[243., 252., 253.],
              [243., 252., 253.],
              [244., 252., 252.],
              ...,
              [238., 250., 254.],
              [239., 251., 253.],
              [239., 251., 253.]]], dtype=float32)
```

```
x.shape
```

```
(512, 512, 3)
```

```
x = x/255

from tensorflow.keras.applications.resnet50 import preprocess_input

x=np.expand_dims(x,axis=0)
img_data=preprocess_input(x)
img_data.shape
```

```
(1, 512, 512, 3)
```

```
model.predict(img_data)
```

```
1/1 [==============================] - 9s 9s/step
array([[1., 0.]], dtype=float32)
```

```
a=np.argmax(model.predict(img_data), axis=1)
```

```
1/1 [==============================] - 3s 3s/step
```

```
if(a==1):
    print("The signature is not fraud")
else:
    print("The signature is fraud")
```

```
The signature is fraud
```

- Our model has successfully predicted that it is a Forge signature.

```
import pickle
with open('model.pkl', 'wb') as file:
      pickle.dump(model, file)
```

```
.........0
.........1
.........10
.........11
.........12
.........13
.........14
.........15
.........16
.........17
.........18
.........19
.........2
.........20
.........21
.........22
.........23
.........24
.........25
.........26
.........27
.........28
.........29
.........3
.........30
.........31
.........32
.........33
.........34
.........35
.........36
.........37
.........38
.........39
.........4
.........40
.........41
.........42
.........43
.........44
.........45
.........46
```

Conclusion

A model that can learn from signatures and make predictions as to whether the signature in question is a forgery or not, has been successfully implemented. This model can be deployed at various government offices where handwritten signatures are used as a means of approval or authentication. While this method uses CNNs to learn the signatures, the structure of our fully connected layer is not optimal. This implementation may be considered extreme. In the model created in this work, two classes are created for each user (Real and forgery). We have 30 users and thus we have a model with 60 classes to predict. The best accuracy we got was 95.5%.