real-forge-signature-detection

April 15, 2024

1 Import Libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import os
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, MaxPooling2D, MaxPool2D, Dense,␣
 ↪Flatten, Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import visualkeras
```

2 Loading the Data and Normalizing it

[2]:
```
# load the dataset
real_path = 'signature_dataset/original dataset'
forge_path = 'signature_dataset/fraud dataset'

# set the image size to 128x128
img_size = (128, 128)

real_images = []
for img_name in os.listdir(real_path):
    img = cv2.imread(os.path.join(real_path, img_name), cv2.COLOR_RGB2GRAY)
    img = cv2.resize(img, img_size)
    real_images.append(img)
real_images = np.array(real_images)

forge_images = []
for img_name in os.listdir(forge_path):
    img = cv2.imread(os.path.join(forge_path, img_name), cv2.COLOR_RGB2GRAY)
    img = cv2.resize(img, img_size)
```

```
    forge_images.append(img)
forge_images = np.array(forge_images)

# normalize the data
real_images = real_images.astype('float32') / 255.0
forge_images = forge_images.astype('float32') / 255.0
```

3 Create labels for the real and forged signatures

[3]:
```
num_forge_images = len(forge_images)
num_real_images = len(real_images)

# Create labels for the real and forged signatures
forge_labels = np.zeros(num_forge_images, dtype=int)
real_labels = np.ones(num_real_images, dtype=int)

# Concatenate the real and forged images and labels
X = np.concatenate((real_images, forge_images), axis=0)
y = np.concatenate((real_labels, forge_labels), axis=0)
```

[4]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, shuffle=True)
```

[5]:
```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(4891, 128, 128, 3) (1223, 128, 128, 3) (4891,) (1223,)
```

4 Making the CNN model

[6]:
```
# Create a Sequential model
model = Sequential()


# Add a convolutional layer
model.add(Conv2D(filters=16, kernel_size=(3,3), activation='relu',␣
 ↪input_shape=(128, 128, 3)))
# Add a convolutional layer
model.add(Conv2D(filters=16, kernel_size=(3,3), activation='relu'))
# Add a max pooling layer
model.add(MaxPooling2D(pool_size=(2,2)))
# Add a dropout layer to reduce overfitting
model.add(Dropout(rate=0.2))


# Add a convolutional layer
model.add(Conv2D(filters=32, kernel_size=(3,3), activation='relu'))
```

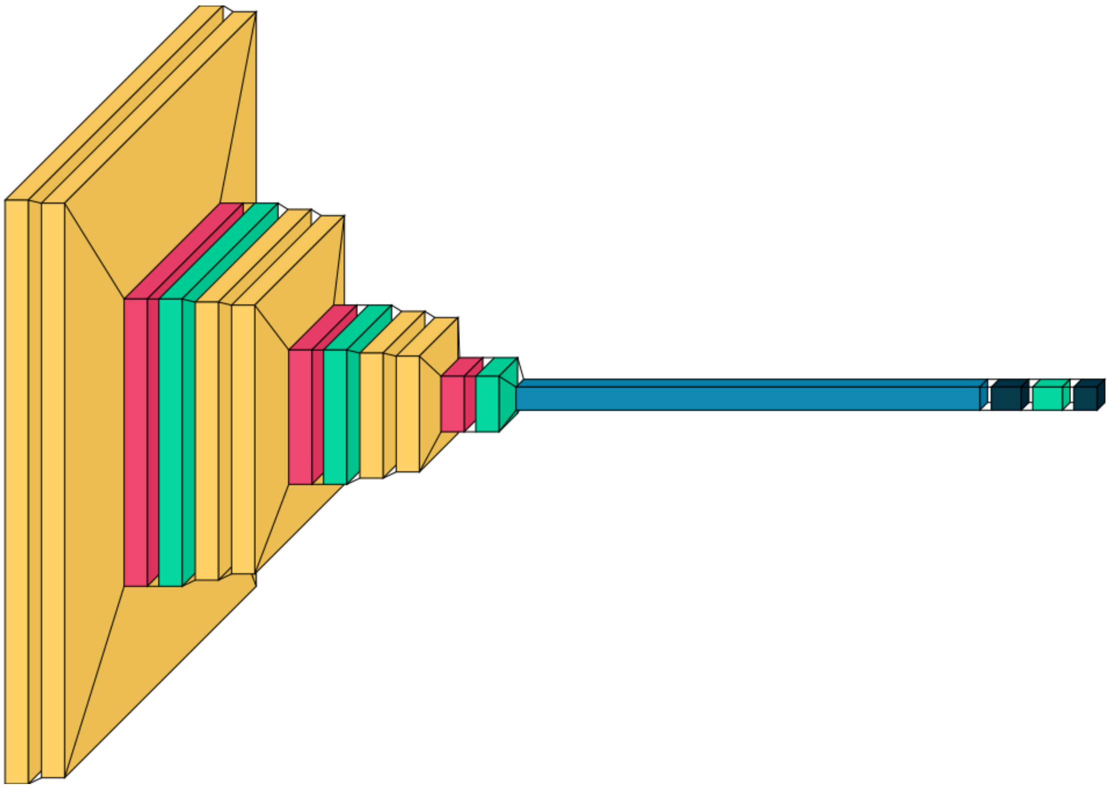
```
# Add a convolutional layer
model.add(Conv2D(filters=32, kernel_size=(3,3), activation='relu'))
# Add a max pooling layer
model.add(MaxPooling2D(pool_size=(2,2)))
# Add a dropout layer to reduce overfitting
model.add(Dropout(rate=0.2))


# Add a convolutional layer
model.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu'))
# Add a convolutional layer
model.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu'))
# Add a max pooling layer
model.add(MaxPooling2D(pool_size=(2,2)))
# Add a dropout layer to reduce overfitting
model.add(Dropout(rate=0.2))


# Flatten the output from the convolutional layers
model.add(Flatten())
# Add a fully connected layer with 128 neurons and a relu activation function
model.add(Dense(units=256, activation='relu'))
# Add a dropout layer to reduce overfitting
model.add(Dropout(rate=0.5))
# Add the output layer with a sigmoid activation function for binary␣
↪classification
model.add(Dense(units=1, activation='sigmoid'))


# Print a summary of the model architecture
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 16)      448

 conv2d_1 (Conv2D)           (None, 124, 124, 16)      2320

 max_pooling2d (MaxPooling2  (None, 62, 62, 16)        0
 D)

 dropout (Dropout)           (None, 62, 62, 16)        0

 conv2d_2 (Conv2D)           (None, 60, 60, 32)        4640

 conv2d_3 (Conv2D)           (None, 58, 58, 32)        9248
```

```
 max_pooling2d_1 (MaxPoolin  (None, 29, 29, 32)        0
 g2D)

 dropout_1 (Dropout)         (None, 29, 29, 32)        0

 conv2d_4 (Conv2D)           (None, 27, 27, 64)        18496

 conv2d_5 (Conv2D)           (None, 25, 25, 64)        36928

 max_pooling2d_2 (MaxPoolin  (None, 12, 12, 64)        0
 g2D)

 dropout_2 (Dropout)         (None, 12, 12, 64)        0

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 256)               2359552

 dropout_3 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 1)                 257

=================================================================
Total params: 2431889 (9.28 MB)
Trainable params: 2431889 (9.28 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[7]:
```
visualkeras.layered_view(model)
```

[7]:



[8]:
```
# EarlyStopping callback
early_stopping = EarlyStopping(monitor='val_loss',         # Monitor validation␣
 ↪loss
                               min_delta=0.001,            # Minimum change in␣
 ↪the monitored quantity to qualify as an improvement
                               patience=4,                 # Number of epochs␣
 ↪with no improvement after which training will be stopped
                               restore_best_weights=True,  # Restore model␣
 ↪weights to the best iteration
                               verbose=0)                  # Verbosity mode (0:␣
 ↪silent, 1: update messages)

# ReduceLROnPlateau callback
reduce_learning_rate = ReduceLROnPlateau(monitor='val_accuracy',  # Monitor␣
 ↪validation accuracy
                                         patience=2,              # Number of␣
 ↪epochs with no improvement after which learning rate will be reduced
                                         factor=0.5,              # Factor by␣
 ↪which the learning rate will be reduced (new_lr = lr * factor)
                                         verbose=1)               # Verbosity␣
 ↪mode (0: silent, 1: update messages)
```

5 Evaluating the Model

[9]:
```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

[10]:
```
history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    batch_size=32,
    epochs=30,
    callbacks=[early_stopping, reduce_learning_rate],
    verbose=1
)
```

```
Epoch 1/30
98/98 [==============================] - 68s 681ms/step - loss: 0.6913 -
accuracy: 0.5557 - val_loss: 0.6342 - val_accuracy: 0.6329 - lr: 0.0010
Epoch 2/30
98/98 [==============================] - 63s 641ms/step - loss: 0.6262 -
accuracy: 0.6581 - val_loss: 0.5847 - val_accuracy: 0.6885 - lr: 0.0010
Epoch 3/30
98/98 [==============================] - 75s 766ms/step - loss: 0.5930 -
accuracy: 0.6866 - val_loss: 0.5541 - val_accuracy: 0.7244 - lr: 0.0010
Epoch 4/30
98/98 [==============================] - 61s 621ms/step - loss: 0.5069 -
accuracy: 0.7526 - val_loss: 0.4419 - val_accuracy: 0.7956 - lr: 0.0010
Epoch 5/30
98/98 [==============================] - 64s 654ms/step - loss: 0.4137 -
accuracy: 0.8141 - val_loss: 0.4053 - val_accuracy: 0.8144 - lr: 0.0010
Epoch 6/30
98/98 [==============================] - 67s 681ms/step - loss: 0.3575 -
accuracy: 0.8434 - val_loss: 0.3190 - val_accuracy: 0.8692 - lr: 0.0010
Epoch 7/30
98/98 [==============================] - 59s 604ms/step - loss: 0.3076 -
accuracy: 0.8669 - val_loss: 0.3093 - val_accuracy: 0.8823 - lr: 0.0010
Epoch 8/30
98/98 [==============================] - 60s 615ms/step - loss: 0.2682 -
accuracy: 0.8900 - val_loss: 0.2697 - val_accuracy: 0.8970 - lr: 0.0010
Epoch 9/30
98/98 [==============================] - 64s 658ms/step - loss: 0.2316 -
accuracy: 0.9053 - val_loss: 0.2627 - val_accuracy: 0.8962 - lr: 0.0010
Epoch 10/30
98/98 [==============================] - 65s 661ms/step - loss: 0.1807 -
accuracy: 0.9246 - val_loss: 0.2897 - val_accuracy: 0.8994 - lr: 0.0010
Epoch 11/30
98/98 [==============================] - 63s 647ms/step - loss: 0.1490 -
accuracy: 0.9442 - val_loss: 0.3017 - val_accuracy: 0.8782 - lr: 0.0010
```

```
Epoch 12/30
98/98 [==============================] - 62s 630ms/step - loss: 0.1330 -
accuracy: 0.9448 - val_loss: 0.2507 - val_accuracy: 0.9092 - lr: 0.0010
Epoch 13/30
98/98 [==============================] - 62s 636ms/step - loss: 0.1011 -
accuracy: 0.9605 - val_loss: 0.2496 - val_accuracy: 0.9158 - lr: 0.0010
Epoch 14/30
98/98 [==============================] - 62s 633ms/step - loss: 0.1046 -
accuracy: 0.9644 - val_loss: 0.2157 - val_accuracy: 0.9231 - lr: 0.0010
Epoch 15/30
98/98 [==============================] - 63s 643ms/step - loss: 0.0673 -
accuracy: 0.9753 - val_loss: 0.2993 - val_accuracy: 0.9150 - lr: 0.0010
Epoch 16/30
98/98 [==============================] - ETA: 0s - loss: 0.0777 - accuracy:
0.9744
Epoch 16: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
98/98 [==============================] - 63s 641ms/step - loss: 0.0777 -
accuracy: 0.9744 - val_loss: 0.2354 - val_accuracy: 0.9231 - lr: 0.0010
Epoch 17/30
98/98 [==============================] - 62s 634ms/step - loss: 0.0405 -
accuracy: 0.9865 - val_loss: 0.2601 - val_accuracy: 0.9305 - lr: 5.0000e-04
Epoch 18/30
98/98 [==============================] - 66s 679ms/step - loss: 0.0324 -
accuracy: 0.9892 - val_loss: 0.2764 - val_accuracy: 0.9305 - lr: 5.0000e-04
Epoch 19/30
98/98 [==============================] - ETA: 0s - loss: 0.0375 - accuracy:
0.9873
Epoch 19: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
98/98 [==============================] - 65s 660ms/step - loss: 0.0375 -
accuracy: 0.9873 - val_loss: 0.2734 - val_accuracy: 0.9289 - lr: 5.0000e-04
```

6 Testing Loss and Accuracy

[11]:
```python
train_loss, train_acc = model.evaluate(X_train, y_train, verbose=1)

print("The accuracy of the model for training data is:", train_acc * 100)
print("The Loss of the model for training data is:", train_loss)
```

```
153/153 [==============================] - 9s 56ms/step - loss: 0.0308 -
accuracy: 0.9973
The accuracy of the model for training data is: 99.7342050075531
The Loss of the model for training data is: 0.03080846555531025
```

[12]:
```python
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=1)

print("The accuracy of the model for testing data is:", test_acc * 100)
print("The Loss of the model for testing data is:", test_loss)
```

```
39/39 [==============================] - 2s 54ms/step - loss: 0.2157 - accuracy:
0.9231
The accuracy of the model for testing data is: 92.31398105621338
The Loss of the model for testing data is: 0.21567381918430328
```

[13]:
```python
model.save("model/signature_detection.keras")
```

[16]:
```python
model.save("model/signature_detection.h5")
```

```
C:\Users\ASUS\AppData\Local\Programs\Python\Python310\lib\site-
packages\keras\src\engine\training.py:3079: UserWarning: You are saving your
model as an HDF5 file via `model.save()`. This file format is considered legacy.
We recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(
```

[17]:
```python
prediction = model.predict(X_test)
```

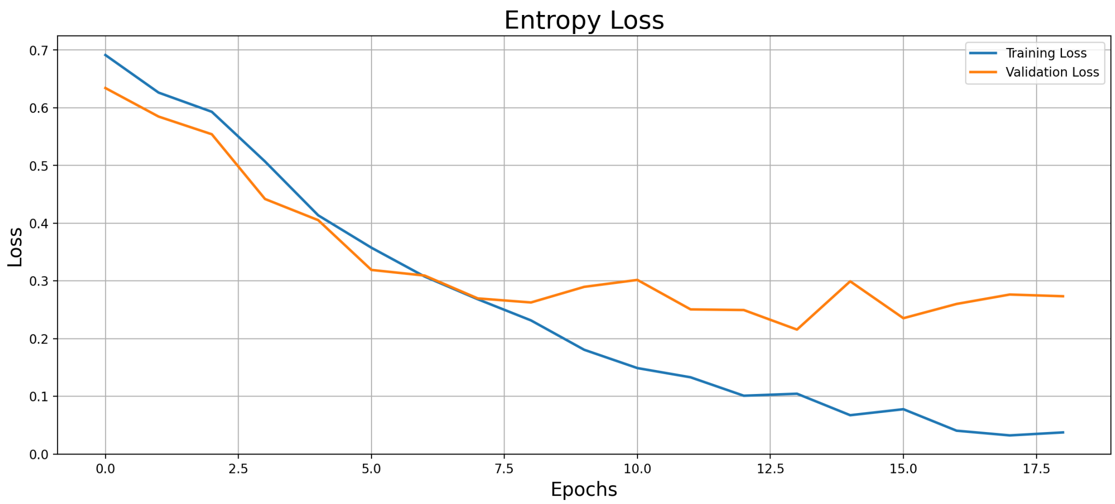
```
39/39 [==============================] - 2s 52ms/step
```

[18]:
```python
prediction
```

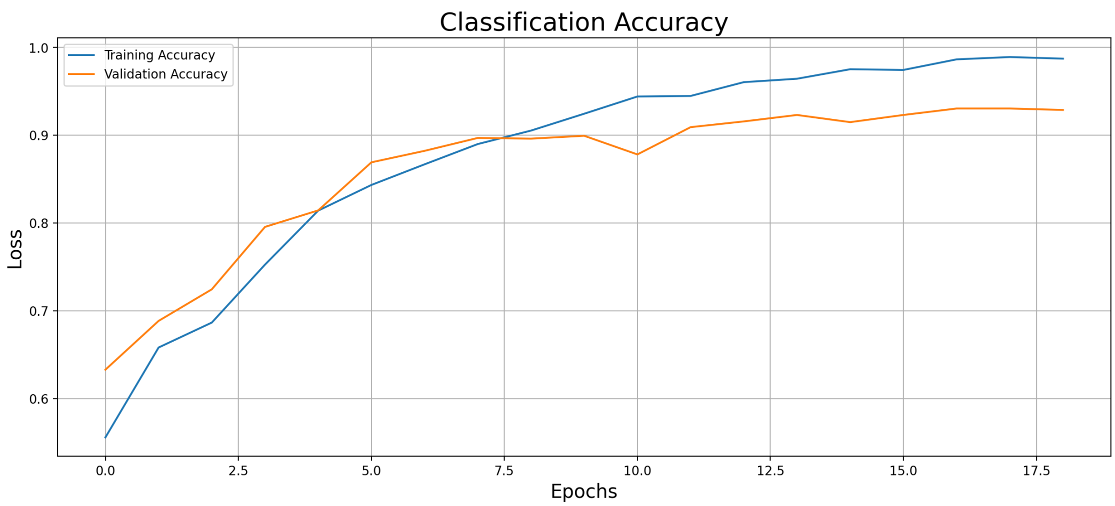
[18]:
```
array([[0.00214481],
       [0.07519121],
       [0.02211507],
       …,
       [0.02339804],
       [0.00028699],
       [0.00366422]], dtype=float32)
```

7 Ploting

[19]:
```python
plt.figure(figsize=(15, 6), dpi=200)

plt.title('Entropy Loss', fontsize=20)
plt.xlabel('Epochs', fontsize=15)
plt.ylabel('Loss', fontsize=15)
plt.plot(history.history['loss'], label='Training Loss', linewidth=2)
plt.plot(history.history['val_loss'], label='Validation Loss', linewidth=2)
plt.legend()
plt.grid(True)

plt.show()
```



[20]:
```
plt.figure(figsize=(15, 6), dpi=200)

plt.title('Classification Accuracy', fontsize=20)
plt.xlabel('Epochs', fontsize=15)
plt.ylabel('Loss', fontsize=15)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label = 'Validation Accuracy')
plt.legend()
plt.grid(True)

plt.show()
```



8 Detection of Real and Forged Signature

[27]:
```python
# Load a signature image
# You can change the image path and check if it is forged or real
# img = cv2.imread('dataset/train/original dataset/original_01_01.jpg', cv2.
↪IMREAD_GRAYSCALE)
img = cv2.imread('signature_dataset/fraud dataset/forgeries_12_23.png', cv2.
↪COLOR_RGB2GRAY)
img = cv2.resize(img, (128, 128))
img = np.array(img).reshape(1, 128, 128, 3) / 255.0

# Predict the class of the signature image
prediction = model.predict(img)

if prediction > 0.7:
    print("The signature is real.")
else:
    print("The signature is forged.")
```

```
1/1 [==============================] - 0s 42ms/step
The signature is forged.
```

[28]:
```python
# Load a signature image
# You can change the image path and check if it is forged or real
img = cv2.imread('signature_dataset/original dataset/original_13_20.png', cv2.
↪COLOR_RGB2GRAY)
# img = cv2.imread('dataset/train/fraud dataset/forgeries_12_23.png', cv2.
↪IMREAD_GRAYSCALE)
img = cv2.resize(img, (128, 128))
img = np.array(img).reshape(1, 128, 128, 3) / 255.0

# Predict the class of the signature image
prediction = model.predict(img)

if prediction > 0.7:
    print("The signature is real.")
else:
    print("The signature is forged.")
```

```
1/1 [==============================] - 0s 43ms/step
The signature is real.
```

[ ]:
```python

```