Convolutional Neural Network

Importing the libraries

```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
tf.__version__
```

```
     '2.5.0'
```

Tensorflow is API that does image detection and stuff, ImageDataGenerator in Keras API prepares the image to be fed in algorithm

Part 1 - Data Preprocessing

Preprocessing the Training set

```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
```

Image Data generator converts image in usable data. It rescales the value of each pixel from [0,255] to [0,1]. Shear distorts the image a bit & we have dezoomed the image

```
training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')
```

```
     Found 12 images belonging to 3 classes.
```

Takes the dataframe and the path to a directory + generates batches.The generated batches contain augmented/normalized data. Image given is 64 in height 64 in width. batch of 32 is converted to binary data

Preprocessing the Test set

```
test_datagen = ImageDataGenerator(rescale = 1./255)
test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'binary')
```

```
Found 6 images belonging to 3 classes.
```

We do the same with test data. This is the test dataset we can keep we don't sheer or zoom them.

Part 2 - Building the CNN

Initialising the CNN

```
cnn = tf.keras.models.Sequential()
```

Cnn object created to explain this imagine a machine learning bot...cnn is basically that bot

Step 1 - Convolution

```
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu', input_shape=[64,
```

The image that we descaled is now converted into multidiminsional array

Step 2 - Pooling

```
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

Pooling the features of the convoluted array

Adding a second convolutional layer

```
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

Same for another layer, to make it more efficient the more you add the more slightly efficient it becomes, you can add 100 layers, but it offers diminishing returns, so 2 is enough

Step 3 - Flattening

```
cnn.add(tf.keras.layers.Flatten())
```

Flatten 2D array to 1D array

Step 4 - Full Connection

```
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))
```

The dense function is a bit complex, I will send a PDF to explain. But this is where ML mathematics happens.

Step 5 - Output Layer

```
cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Part 3 - Training the CNN

Compiling the CNN

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

Imagine the c++ compiler, this step prepare the cnn robot ready to learn whatever you want

Training the CNN on the Training set and evaluating it on the Test set

```
cnn.fit(x = training_set, validation_data = test_set, epochs = 25)
```

```
     Epoch 1/25
```

```
1/1 [==============================] - 1s 1s/step - loss: 0.8968 - accuracy: 0.0000e+00
Epoch 2/25
1/1 [==============================] - 1s 579ms/step - loss: -0.6981 - accuracy: 0.1667
Epoch 3/25
1/1 [==============================] - 1s 578ms/step - loss: -2.3733 - accuracy: 0.1667
Epoch 4/25
1/1 [==============================] - 1s 556ms/step - loss: -4.6449 - accuracy: 0.1667
Epoch 5/25
1/1 [==============================] - 1s 570ms/step - loss: -7.6930 - accuracy: 0.1667
Epoch 6/25
1/1 [==============================] - 1s 558ms/step - loss: -12.0760 - accuracy: 0.166
Epoch 7/25
1/1 [==============================] - 1s 583ms/step - loss: -18.1425 - accuracy: 0.166
Epoch 8/25
1/1 [==============================] - 1s 570ms/step - loss: -25.5418 - accuracy: 0.166
Epoch 9/25
1/1 [==============================] - 1s 580ms/step - loss: -34.7227 - accuracy: 0.166
Epoch 10/25
1/1 [==============================] - 1s 556ms/step - loss: -46.5854 - accuracy: 0.166
Epoch 11/25
1/1 [==============================] - 1s 562ms/step - loss: -61.8439 - accuracy: 0.166
Epoch 12/25
1/1 [==============================] - 1s 569ms/step - loss: -78.5343 - accuracy: 0.166
Epoch 13/25
1/1 [==============================] - 1s 543ms/step - loss: -100.8282 - accuracy: 0.16
Epoch 14/25
1/1 [==============================] - 1s 554ms/step - loss: -126.2228 - accuracy: 0.16
Epoch 15/25
1/1 [==============================] - 1s 579ms/step - loss: -157.9597 - accuracy: 0.16
Epoch 16/25
1/1 [==============================] - 1s 571ms/step - loss: -195.1814 - accuracy: 0.16
Epoch 17/25
1/1 [==============================] - 1s 572ms/step - loss: -239.6632 - accuracy: 0.16
Epoch 18/25
1/1 [==============================] - 1s 551ms/step - loss: -288.1270 - accuracy: 0.16
Epoch 19/25
1/1 [==============================] - 1s 575ms/step - loss: -341.8114 - accuracy: 0.16
Epoch 20/25
1/1 [==============================] - 1s 581ms/step - loss: -417.5803 - accuracy: 0.16
Epoch 21/25
1/1 [==============================] - 1s 562ms/step - loss: -500.2334 - accuracy: 0.16
Epoch 22/25
1/1 [==============================] - 1s 563ms/step - loss: -593.1028 - accuracy: 0.16
Epoch 23/25
1/1 [==============================] - 1s 556ms/step - loss: -696.2051 - accuracy: 0.16
Epoch 24/25
1/1 [==============================] - 1s 558ms/step - loss: -808.5278 - accuracy: 0.16
Epoch 25/25
1/1 [==============================] - 1s 563ms/step - loss: -933.8250 - accuracy: 0.16
<tensorflow.python.keras.callbacks.History at 0x7fd0bc8be650>
```

Train the data on training set and compare it with the test set as validation_data

Part 4 - Making a single prediction

```
import numpy as np
from keras.preprocessing import image
test_image = image.load_img('dataset/single_prediction/test_image.jpeg', target_size = (64, 6
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = cnn.predict(test_image)
training_set.class_indices
if result[0][0] == 1:
    prediction = 'blank'
else:
    prediction = 'written'
print(prediction)
```

```
blank
```

✓ 0s completed at 2:48 PM ● ×