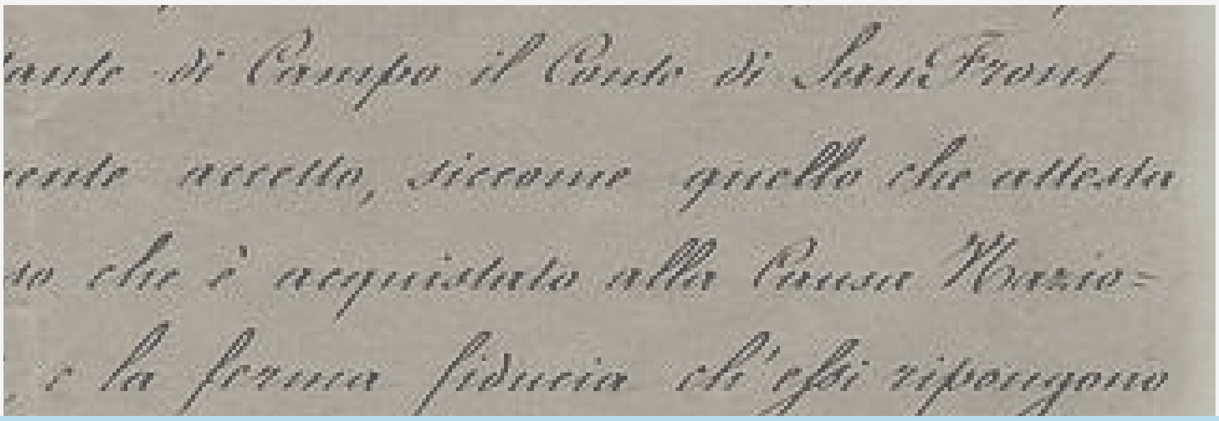
CCDS 322 - DSB

جامعة جدة
University of Jeddah

FINAL PROJECT REPORT

Applied machine learning

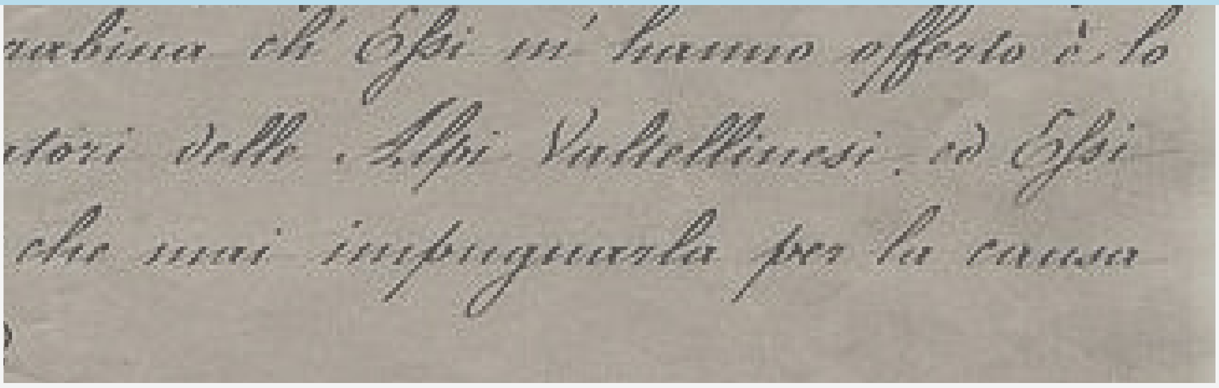


Handwritten Character Recognition

PREPARED BY

Marwh AL-hadi	2007881
Amira Saleh	2006782
Enas Khan	2007145
Zain Abdulwahab	1911102

APPROVED BY
Dr Ines Boufateh



2023

Table Of Content

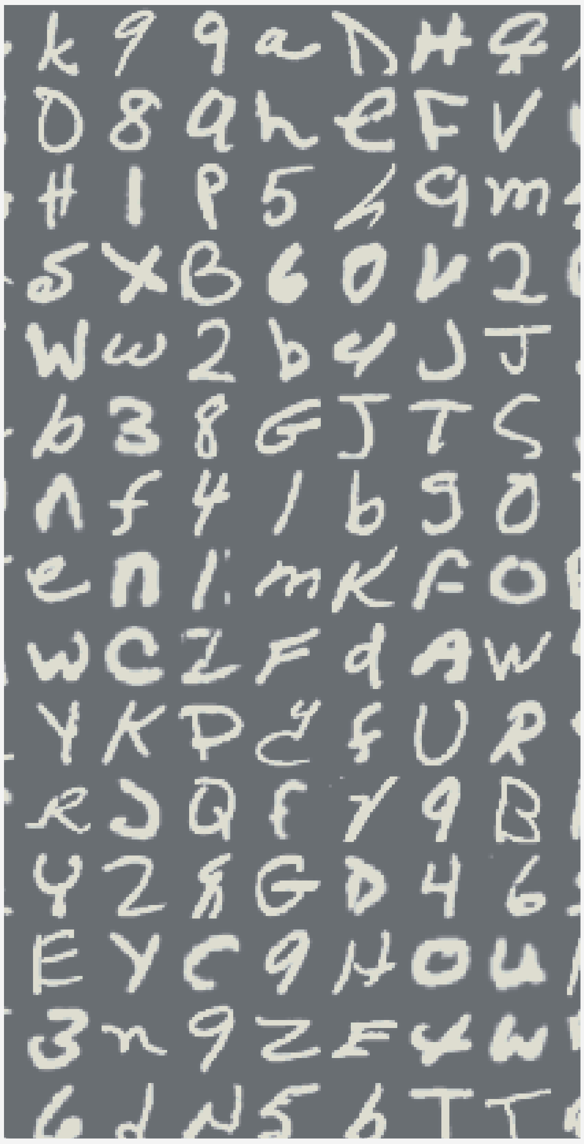
What is handwriting recognition?

Handwriting Recognition (HWR) is the capability of computers and mobile devices to receive and interpret handwritten inputs. The inputs might be offline(scanned from paper documents, images, etc.) or online (sensed from the movement of pens on a special digitizer, for example).
A handwriting recognition system also includes formatting, segmentation into individual characters, and training a language model that learns to frame meaningful words and sentences.
The most popular technique for handwriting recognition is Optical Character Recognition (OCR). It allows us to scan handwritten documents and then convert them into basic text through computer vision.

Benefits of handwriting recognition:
1- Better data storage
2- Faster information retrieval
3- Improved accessibility
4- Better customer service

THE GOAL

We have chosen to work on a handwriting recognition model (HWR). So we selected a dataset that contains images of a handwritten letters and numbers. even though the english alphabet consists of 26 letters from A to Z and 10 numbers are included also in the dataset from 0 to 9, the datasets includes all the possible cases for writing each character. In our opinion, the current chosen dataset would help with build an accurate useful a HWR model. The dataset was found on Kaggle. And since we need the model to be able to analyze visual input, we will be building our model with the Convolutional Neural Networks (CNN).



About the data

This dataset contains 3,410 images of handwritten characters in English. This is a classification dataset that can be used for Computer Vision tasks. It contains 62 classes with 55 images of each class. The 62 classes are 0-9, A-Z and a-z.

Author = *de Campos, T.~E. and Babu, B.~R. and Varma, M.*
Title= *Character recognition in natural images*
Booktitle = *Proceedings of the International Conference on Computer*
Vision Theory and Applications, Lisbon, Portugal
Month= *February*
Year= *2009*
Source *= Kaggle website*

What is the hypothesis for this investigation?

using handwritten character recognition helps to convert the writings in the papers into a readable electronic format. Handwriting recognition applications are booming, widely used.

Therefore, using our data set, we will create a model by using the neural network, assuming that it will analyze the images of the handwriting , recognize it, compare it with the label, and identify if it is numbers or charecrer ,to ensure accuracy and then write it

in this project, the question is, will our model be able to accurately read the handwritten English alphabet and numbers?

GOOGLE COLAB

- Collab is a product from Google Research.
- requires no setup to use.
- Colab allows anybody to write and execute arbitrary python code through the browser.
- providing access free of charge to computing resources including GPUs.
- well suited to machine learning, data analysis, and education.

LIBRARIES

01 **NUMPY**

stand for numerical python to make the numerical calculation easier instead of coding it from scratch by its functions and high-performance multidimensional array objects. Otherwise facilitates math operations on arrays and their vectorization.

02 **PANDAS**

stand for Python Data Analysis, it can be considered as the excel of python. In excel, which we used to store and manipulate data, is an open-source Python package that provides high-performance, easy-to-use data structures and data analysis tools for the labeled data in the Python programming language

03 **SCIKIT LEARN**

provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python, focusing on modeling data; not manipulating data. We have NumPy and Pandas for summarizing and manipulation.

LIBRARIES

04 MATPLOTLIB

Matplotlib is the plotting library that provides an object-oriented API for embedding plots into applications. this library helps easily visualize the huge amount of data and have a rudimentary interpretation of your data

05 TORCH OR PYTORCH

PyTorch is a machine learning framework. It is open source and is based on the popular Torch library. PyTorch is designed to provide good flexibility and high speeds for deep neural network implementation

06 OS LIBRARY

The OS module in Python provides functions for creating and removing a directory (folder), fetching its contents, changing and identifying the current directory, etc.

07 TENSORFLOW

TensorFlow is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow

08 KERAS LIBRARY

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

09 OPEN-CV

OpenCV-Python is a library of Python bindings designed to solve computer vision problems.

the suitable algorithm

Now we will detail, describe and analyze the algorithms/model we used and why we adopted it and considered it the most appropriate

data cleaning

The data used is clean, so we did not need to clean it

Import libraries

Import the libraries (mentioned in the previous slide) because they are essential in structuring the code effectively. and allow reusing code in them

```
import pandas as pd
import matplotlib.pyplot as plt
import cv2 as cv
import random

from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout, Activation
from keras import optimizers
from keras_preprocessing.image import ImageDataGenerator
from keras.models import load_model

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout
```

Reading The CSV Dataset File

takes a path to a CSV file and reads the data into a Pandas DataFrame object.

head() : displays the first five rows of the data frame only to check that read it right

```
df = pd.read_csv('/content/project/english.csv')
df.head()
```

The output

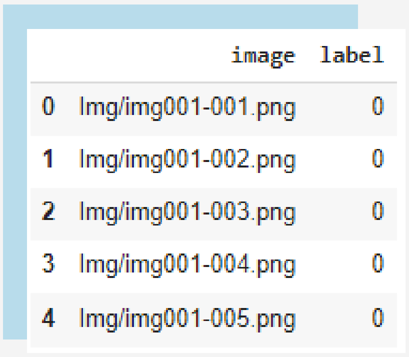


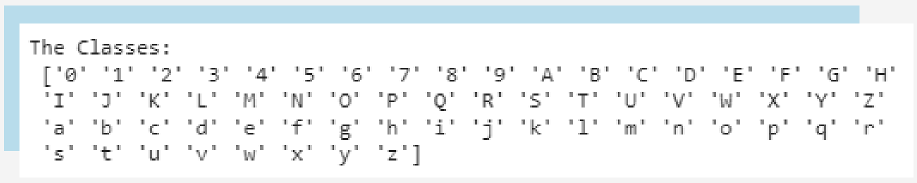
	image	label
0	Img/img001-001.png	0
1	Img/img001-002.png	0
2	Img/img001-003.png	0
3	Img/img001-004.png	0
4	Img/img001-005.png	0

Define Labels

Take unique labels only to determine classes in the data

```
classes = df['label'].unique()
print(f'\nThe Classes:\n {classes} ')
```

The output



```
The Classes:
 ['0' '1' '2' '3' '4' '5' '6' '7' '8' '9' 'A' 'B' 'C' 'D' 'E' 'F' 'G' 'H'
 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P' 'Q' 'R' 'S' 'T' 'U' 'V' 'W' 'X' 'Y' 'Z'
 'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm' 'n' 'o' 'p' 'q' 'r'
 's' 't' 'u' 'v' 'w' 'x' 'y' 'z']
```

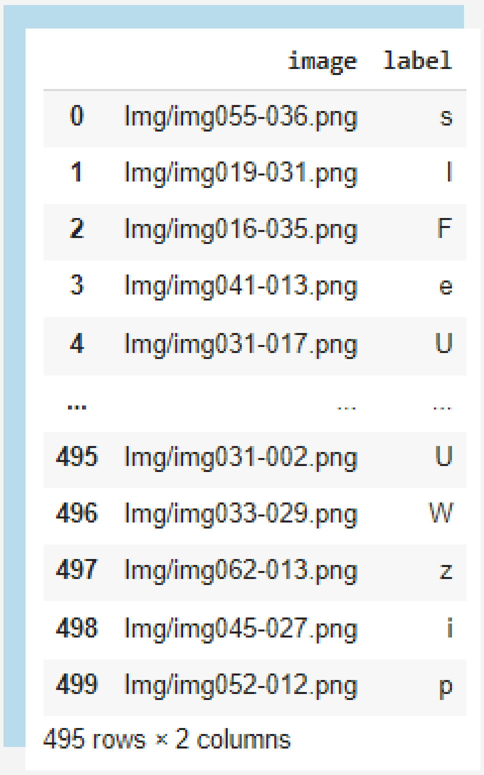
All letters of the English alphabet and numbers 0 - 9

Processing The Data

Mapping Labels with Images and Splitting The Dataset

```
# Define the data path
DATADIR = "/content/project"
# Read the csv file
dataset = pd.read_csv(DATADIR + '/english.csv')
# Get a 500 random values/rows
rand = random.sample(range(len(dataset)), 500)
# Make the random 500 as a validation data
validation_set = pd.DataFrame(dataset.iloc[rand, :].values, columns=['image', 'label'])
# Drop the 500 from the orignal data set
dataset.drop(rand, inplace=True)
# Get a 5 random rows/values from the validation set
rand = random.sample(range(len(validation_set)), 5)
# from the 5 random Create the test set
test_set = pd.DataFrame(validation_set.iloc[rand, :].values, columns=['image', 'label'])
# Drop the 5 from the validation set
validation_set.drop(rand, inplace=True)
# Show the validation set as a example
validation_set
```

The output



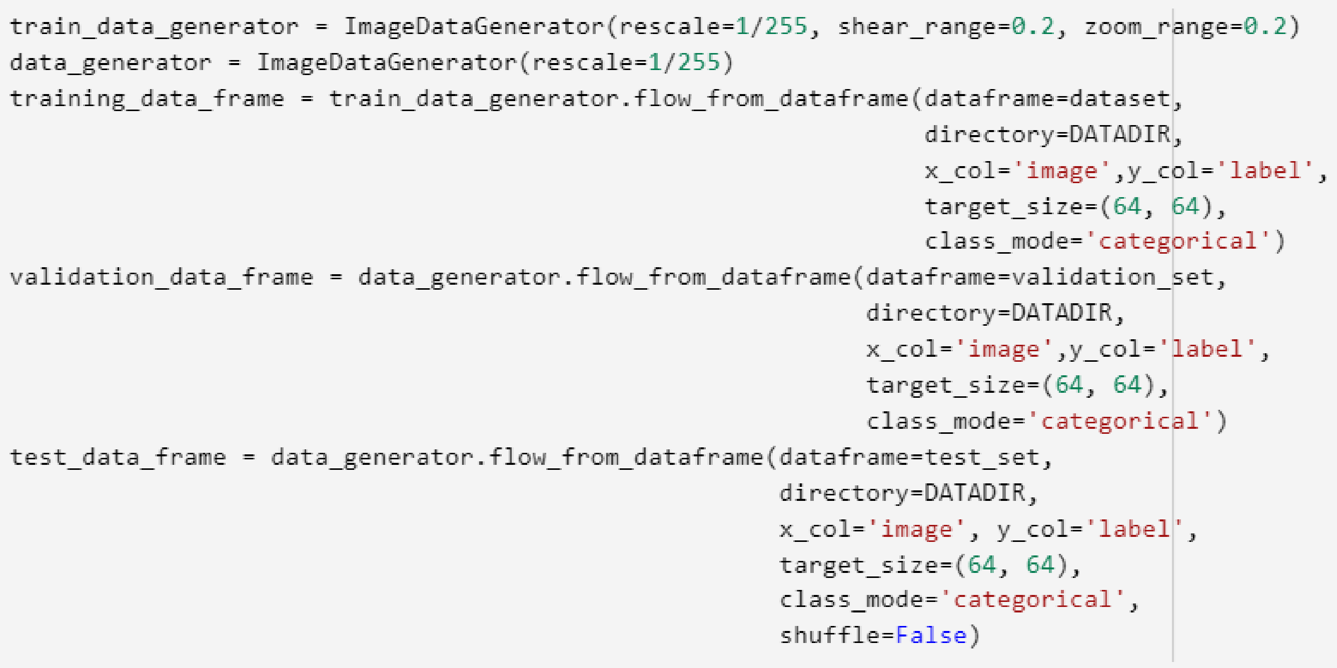
```
                 image  label
0    Img/img055-036.png     s
1    Img/img019-031.png     I
2    Img/img016-035.png     F
3    Img/img041-013.png     e
4    Img/img031-017.png     U
...                 ...   ...
495  Img/img031-002.png     U
496  Img/img033-029.png     W
497  Img/img062-013.png     z
498  Img/img045-027.png     i
499  Img/img052-012.png     p
495 rows × 2 columns
```

After we divided the data, we displayed validation data

Processing The Data (Cont.)

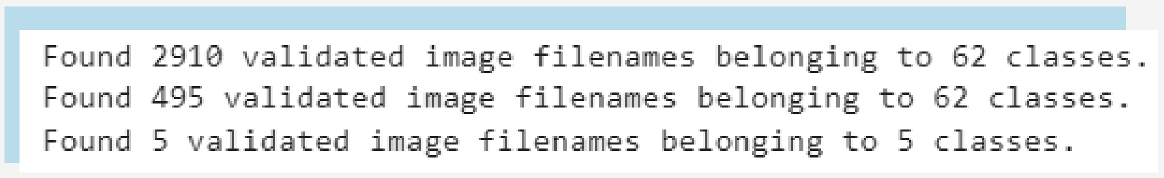
Keras ImageDataGenerator class allows the users to perform image augmentation while training the model.

The flow_from_dataframe() method takes the Pandas DataFrame and the path to a directory and generates batches of augmented/normalized data..



```
train_data_generator = ImageDataGenerator(rescale=1/255, shear_range=0.2, zoom_range=0.2)
data_generator = ImageDataGenerator(rescale=1/255)
training_data_frame = train_data_generator.flow_from_dataframe(dataframe=dataset,
                                                               directory=DATADIR,
                                                               x_col='image',y_col='label',
                                                               target_size=(64, 64),
                                                               class_mode='categorical')
validation_data_frame = data_generator.flow_from_dataframe(dataframe=validation_set,
                                                           directory=DATADIR,
                                                           x_col='image',y_col='label',
                                                           target_size=(64, 64),
                                                           class_mode='categorical')
test_data_frame = data_generator.flow_from_dataframe(dataframe=test_set,
                                                     directory=DATADIR,
                                                     x_col='image', y_col='label',
                                                     target_size=(64, 64),
                                                     class_mode='categorical',
                                                     shuffle=False)
```

The output



```
Found 2910 validated image filenames belonging to 62 classes.
Found 495 validated image filenames belonging to 62 classes.
Found 5 validated image filenames belonging to 5 classes.
```

Building The Model

Why did we choose to work on a neural network specifically (CNN), and why did we consider it the most appropriate algorithm?

CNN | *Convolutional neural network*

Convolutional neural network (CNN) is a type of deep learning algorithm used to apply image recognition and classification tasks. This algorithm can provide us with highly accurate results for large volumes of data without the need to extract features.
As shown in the Diagram below, The network consists of multiple layers (convolutional, pooling, and fully connected layer).
The model with the CNN algorithm is trained by using a dataset of labeled images to recognize the patterns and features. Then the trained model can be used for new input of images.
With the benefits of CNN, there are two challenges, small datasets, and overfitting.
The CNN is a method that is a perfect way for medical radiology imaging, detecting patterns, classifying large amounts of visualized datasets

A CNN typically has three layers:

- convolutional layer
- pooling layer
- fully connected layer

Building The Model (Cont.)

Methodology. A convolutional neural network (CNN) was trained to classify handwritten digits in the EMNIST dataset (A). Dropout was applied to 50% of nodes in the fully connected layer (B). Transfer learning was performed to retrain the network to recognize uppercase letters (C).

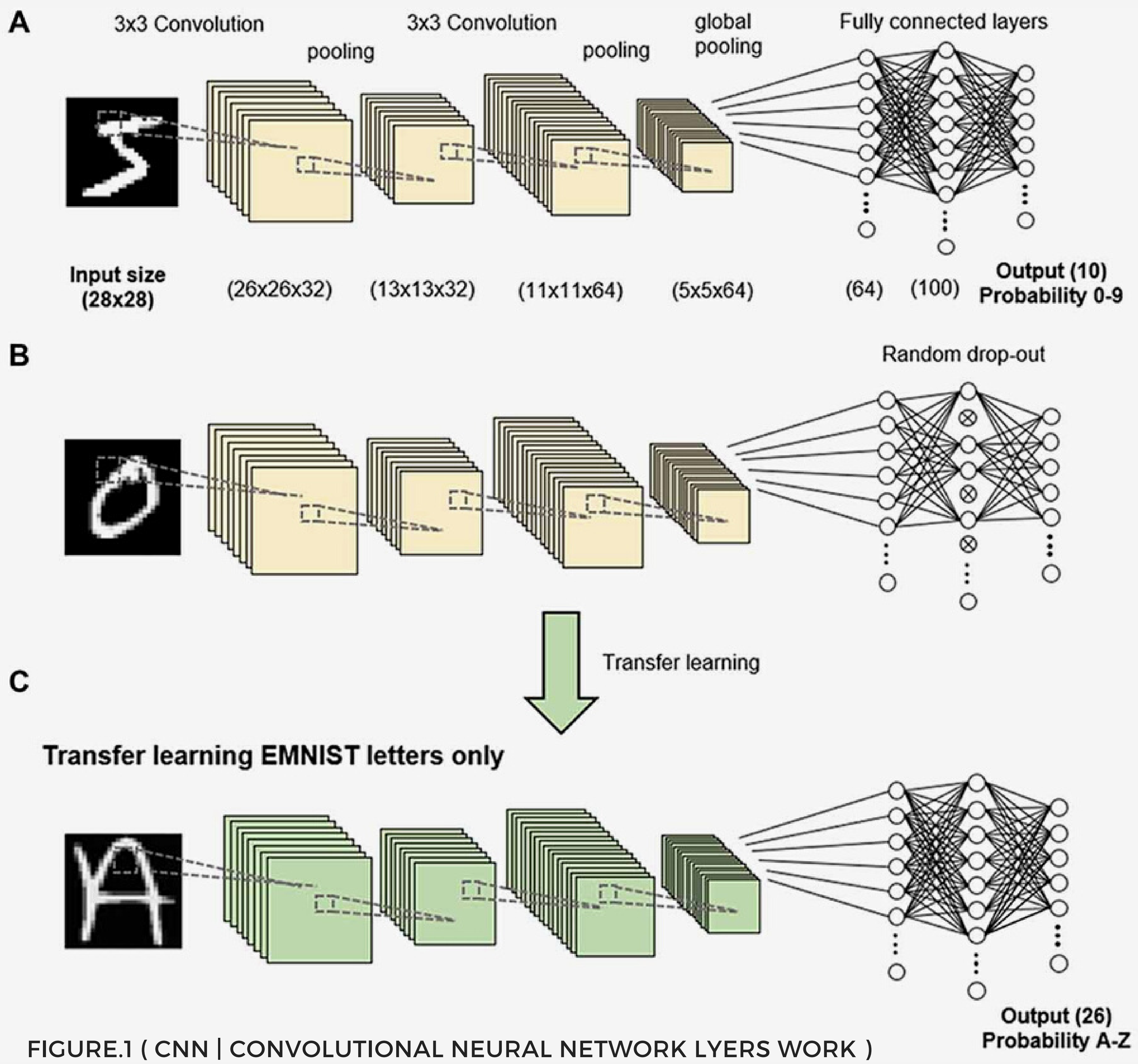
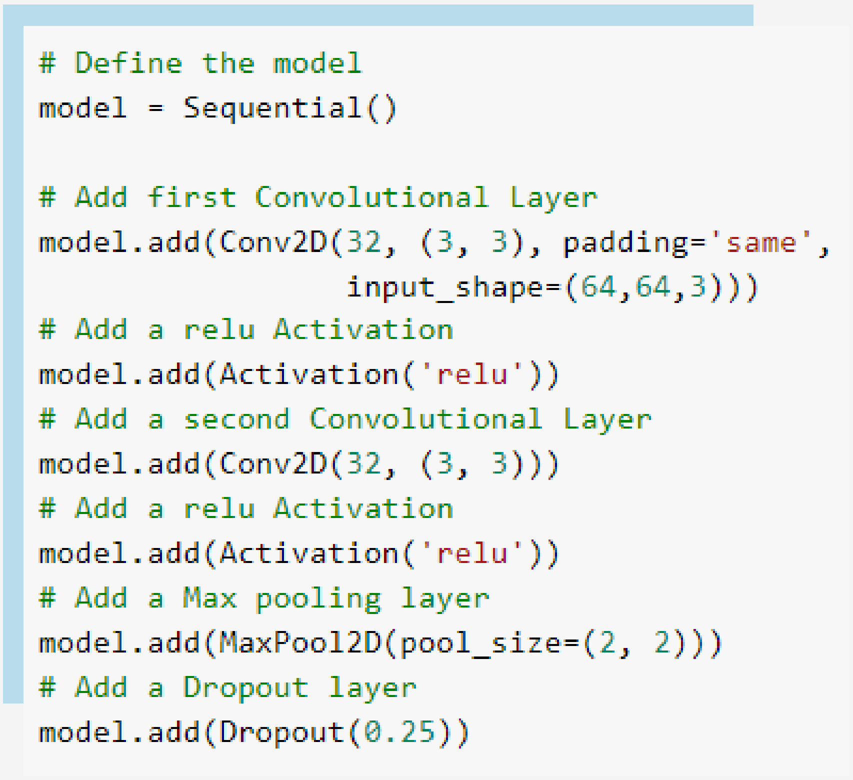


FIGURE.1 (CNN | CONVOLUTIONAL NEURAL NETWORK LYERS WORK)

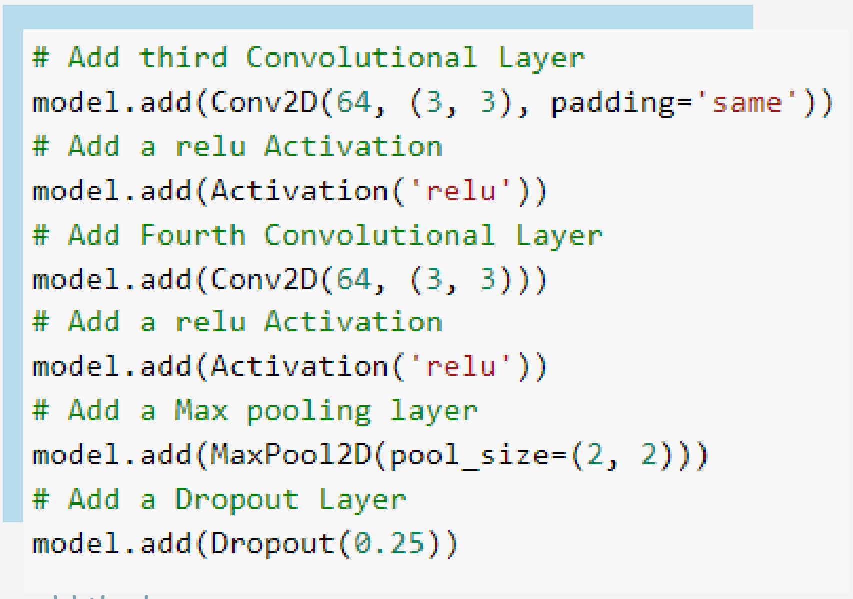
Building The Model (Cont.)

A CNN can be instantiated as a Sequential model because each layer has exactly one input and output and is stacked together to form the entire network.



```
# Define the model
model = Sequential()

# Add first Convolutional Layer
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=(64,64,3)))
# Add a relu Activation
model.add(Activation('relu'))
# Add a second Convolutional Layer
model.add(Conv2D(32, (3, 3)))
# Add a relu Activation
model.add(Activation('relu'))
# Add a Max pooling layer
model.add(MaxPool2D(pool_size=(2, 2)))
# Add a Dropout layer
model.add(Dropout(0.25))
```

add the layers

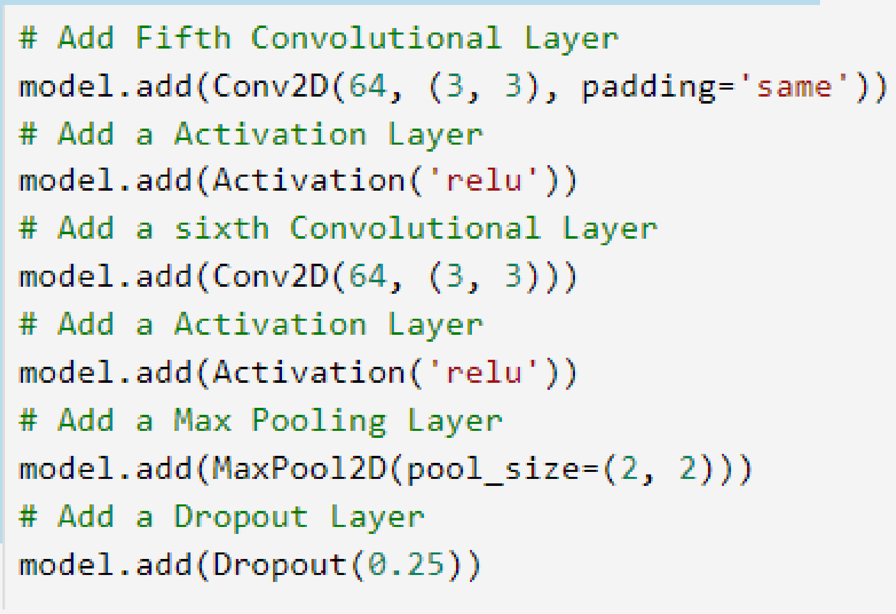


```
# Add third Convolutional Layer
model.add(Conv2D(64, (3, 3), padding='same'))
# Add a relu Activation
model.add(Activation('relu'))
# Add Fourth Convolutional Layer
model.add(Conv2D(64, (3, 3)))
# Add a relu Activation
model.add(Activation('relu'))
# Add a Max pooling layer
model.add(MaxPool2D(pool_size=(2, 2)))
# Add a Dropout Layer
model.add(Dropout(0.25))
```

add the layers

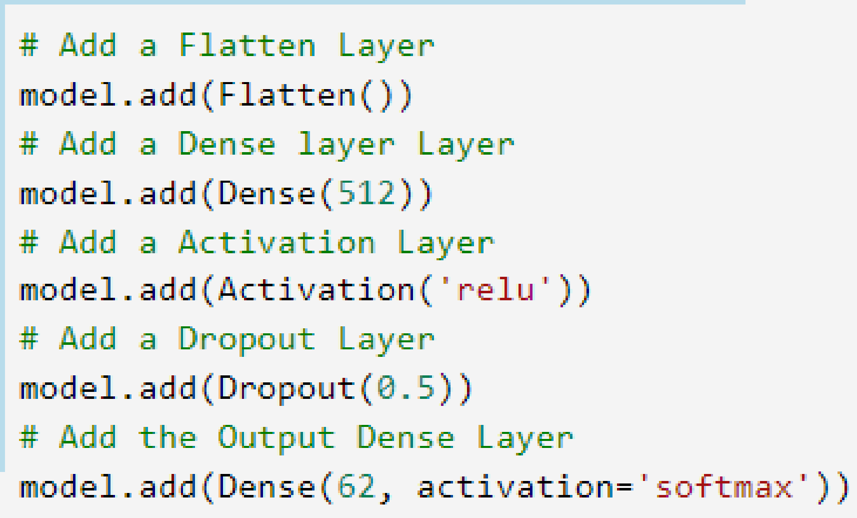
Building The Model (Cont.)

A CNN can be instantiated as a Sequential model because each layer has exactly one input and output and is stacked together to form the entire network.



```
# Add Fifth Convolutional Layer
model.add(Conv2D(64, (3, 3), padding='same'))
# Add a Activation Layer
model.add(Activation('relu'))
# Add a sixth Convolutional Layer
model.add(Conv2D(64, (3, 3)))
# Add a Activation Layer
model.add(Activation('relu'))
# Add a Max Pooling Layer
model.add(MaxPool2D(pool_size=(2, 2)))
# Add a Dropout Layer
model.add(Dropout(0.25))
```

add the layers

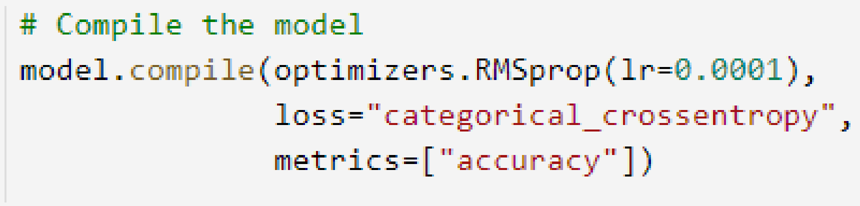


```
# Add a Flatten Layer
model.add(Flatten())
# Add a Dense layer Layer
model.add(Dense(512))
# Add a Activation Layer
model.add(Activation('relu'))
# Add a Dropout Layer
model.add(Dropout(0.5))
# Add the Output Dense Layer
model.add(Dense(62, activation='softmax'))
```

add the layers

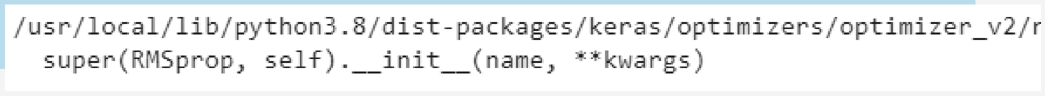
Building The Model (Cont.)

is required to finalise the model and make it completely ready to use. For compilation, we need to specify an optimizer and a loss function. We can compile a model by using compile attribute



```
# Compile the model
model.compile(optimizers.RMSprop(lr=0.0001),
              loss="categorical_crossentropy",
              metrics=["accuracy"])
```

The output



```
/usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/r
  super(RMSprop, self).__init__(name, **kwargs)
```

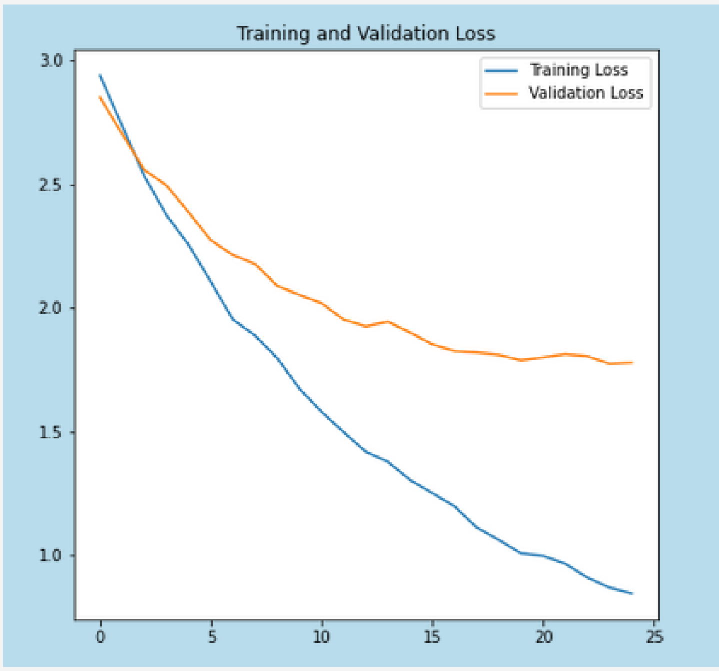
Train the model

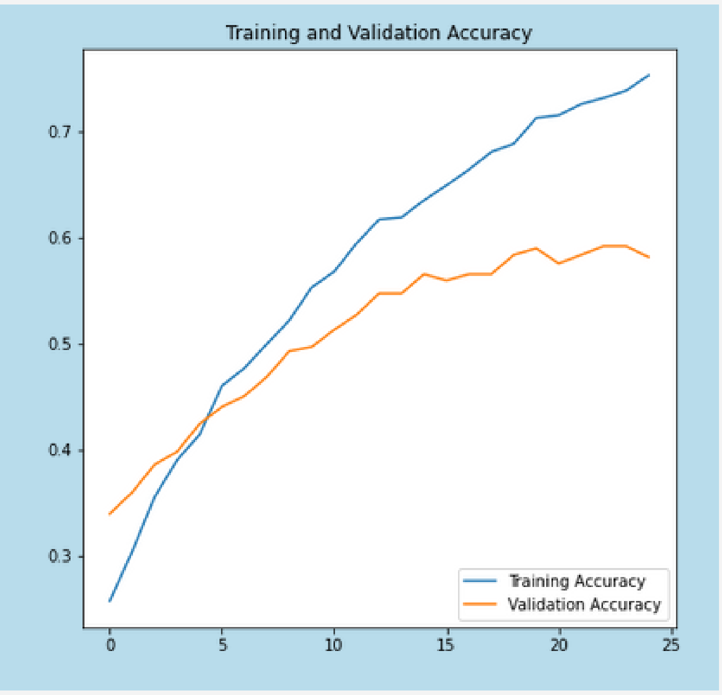
To avoid the problem of Overfitting while training the model with the available samples. with a large number of epochs, the model might occur to be overfitting, so we must choose an appropriate number of epochs for our data according to the number of batches we got on hand.

how to Choose the optimal number of epochs to train

In the beginning: we trained the data on 25 epochs to note the accuracy and loss percentage, but when represented it looked like this:

We noticed that the loss rate is very high and the accuracy is very low, and we know that we can get better than this





The reason for this is because we have a lot of data and the periods are few, so we decided that it should be increased

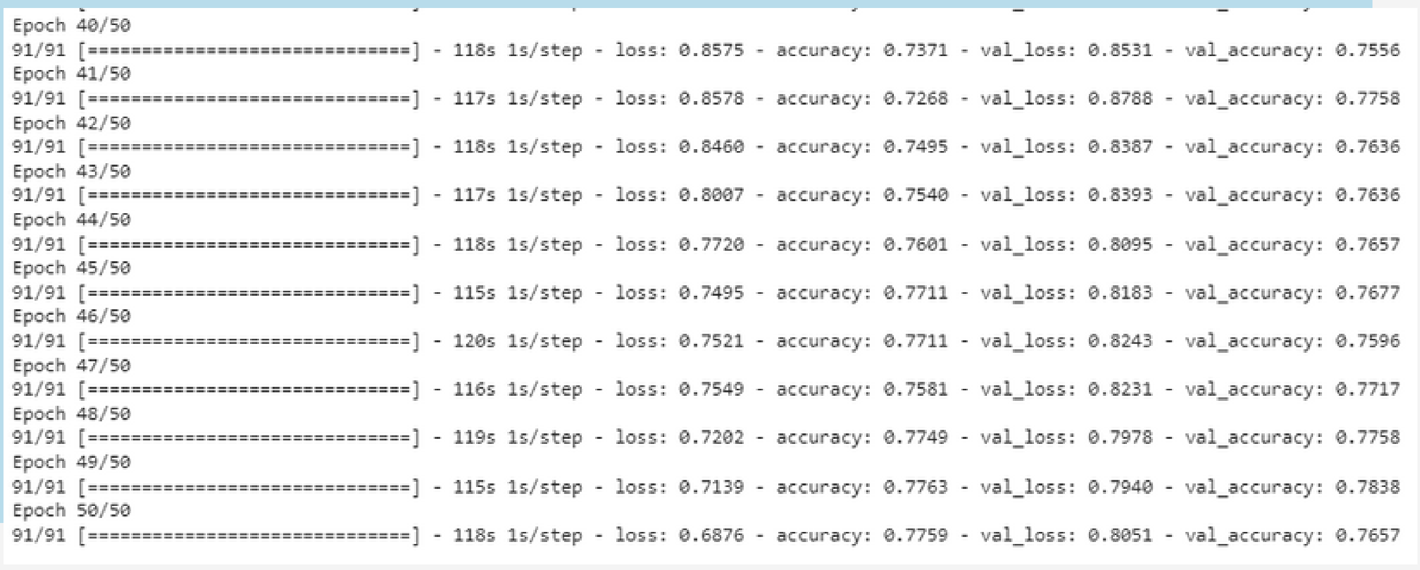
Train the model (Cont.)

We doubled the number of pods to improve performance

history: is an object returned after training a model in Keras which contains information about the training process

```
# Train the model for 50 epochs
history = model.fit(training_data_frame, validation_data=validation_data_frame, epochs=50)
```

The output



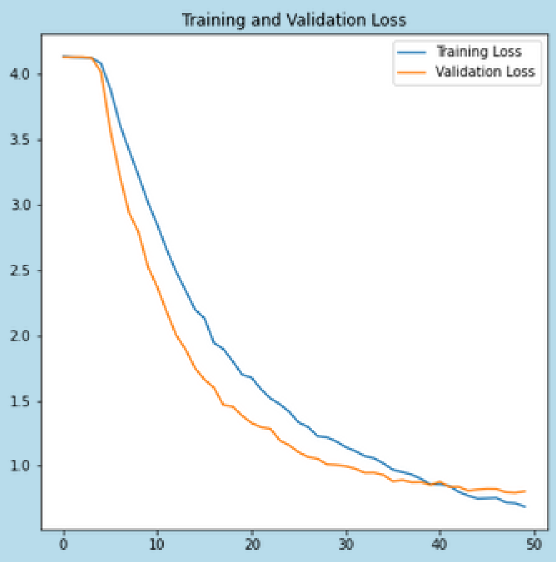
```
Epoch 40/50
91/91 [==============================] - 118s 1s/step - loss: 0.8575 - accuracy: 0.7371 - val_loss: 0.8531 - val_accuracy: 0.7556
Epoch 41/50
91/91 [==============================] - 117s 1s/step - loss: 0.8578 - accuracy: 0.7268 - val_loss: 0.8788 - val_accuracy: 0.7758
Epoch 42/50
91/91 [==============================] - 118s 1s/step - loss: 0.8460 - accuracy: 0.7495 - val_loss: 0.8387 - val_accuracy: 0.7636
Epoch 43/50
91/91 [==============================] - 117s 1s/step - loss: 0.8007 - accuracy: 0.7540 - val_loss: 0.8393 - val_accuracy: 0.7636
Epoch 44/50
91/91 [==============================] - 118s 1s/step - loss: 0.7720 - accuracy: 0.7601 - val_loss: 0.8095 - val_accuracy: 0.7657
Epoch 45/50
91/91 [==============================] - 115s 1s/step - loss: 0.7495 - accuracy: 0.7711 - val_loss: 0.8183 - val_accuracy: 0.7677
Epoch 46/50
91/91 [==============================] - 120s 1s/step - loss: 0.7521 - accuracy: 0.7711 - val_loss: 0.8243 - val_accuracy: 0.7596
Epoch 47/50
91/91 [==============================] - 116s 1s/step - loss: 0.7549 - accuracy: 0.7581 - val_loss: 0.8231 - val_accuracy: 0.7717
Epoch 48/50
91/91 [==============================] - 119s 1s/step - loss: 0.7202 - accuracy: 0.7749 - val_loss: 0.7978 - val_accuracy: 0.7758
Epoch 49/50
91/91 [==============================] - 115s 1s/step - loss: 0.7139 - accuracy: 0.7763 - val_loss: 0.7940 - val_accuracy: 0.7838
Epoch 50/50
91/91 [==============================] - 118s 1s/step - loss: 0.6876 - accuracy: 0.7759 - val_loss: 0.8051 - val_accuracy: 0.7657
```

Run epochs 50 times and calculate the accuracy , the loss

Visualizing The Accuracy loss of The Training and Validation of The Model

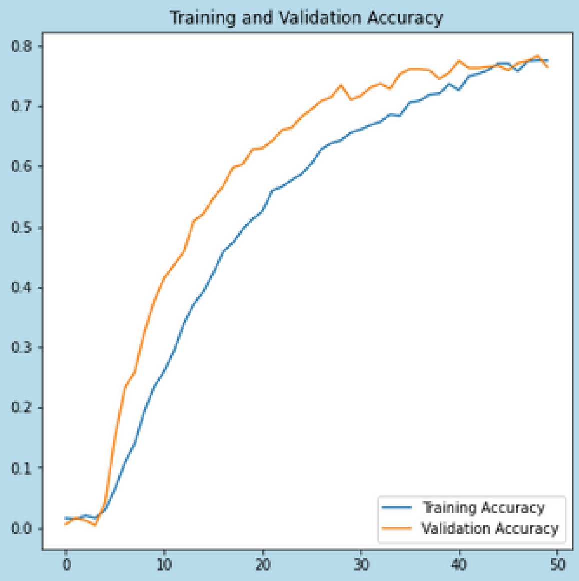
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(50)

plt.figure(figsize=(15, 15))
plt.subplot(2, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

The output



Shows training and validation loss as a function of the number of epochs. These graphs are useful for evaluating model performance during training and detecting potential problems such as overfitting or underfitting

Visualizing The Accuracy loss of The Training and Validation of The Model (Cont.)



Shows accuracy for training and verification based on the number of epochs

As it is clear in the previous two representations, this is the appropriate number because the percentage of loss in verification is less and the accuracy is higher, and it is also clear that it will increase after 50

save and loud the modle

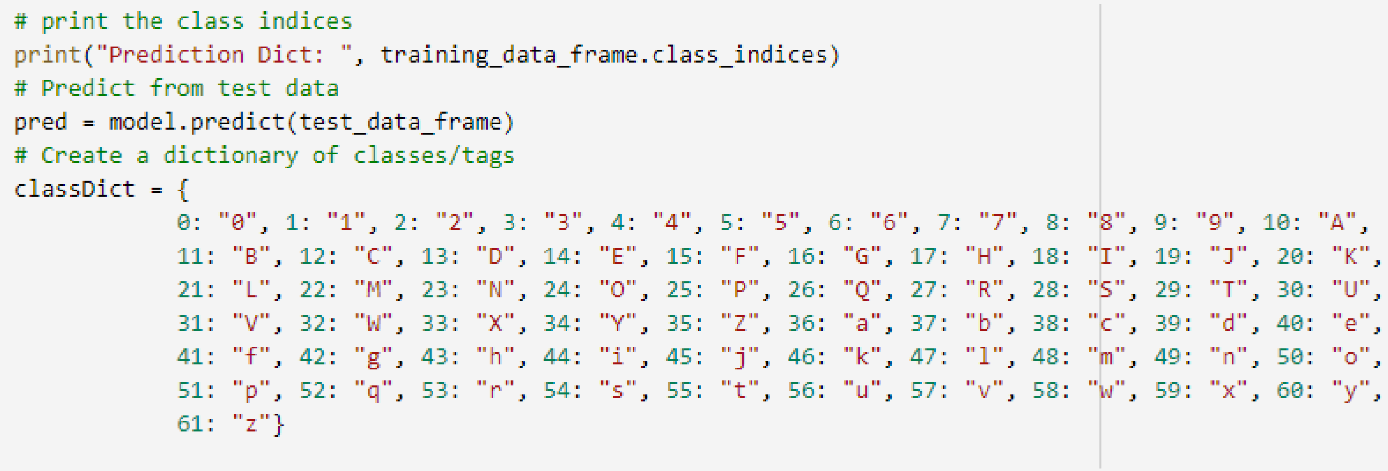
This means a model can resume where it left off and avoid long training times. Saving also means you can share your model and others can recreate your work.

```
# Save the model as model.h5
model.save('model.h5')
# Load the model
model = load_model('model.h5')
```

Predict from test data

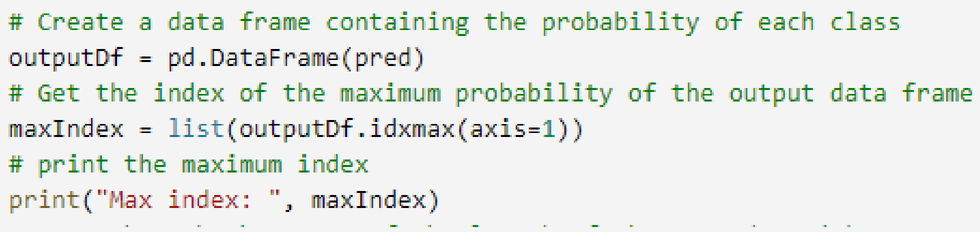
Finally, we can evaluate the model by applying the model to the new data (test data)

Create a dictionary of classes/tags to calculate containing the probability of each class



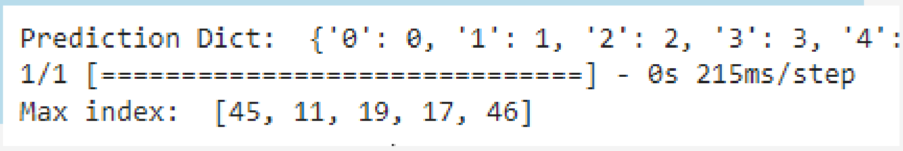
```
# print the class indices
print("Prediction Dict: ", training_data_frame.class_indices)
# Predict from test data
pred = model.predict(test_data_frame)
# Create a dictionary of classes/tags
classDict = {
            0: "0", 1: "1", 2: "2", 3: "3", 4: "4", 5: "5", 6: "6", 7: "7", 8: "8", 9: "9", 10: "A",
            11: "B", 12: "C", 13: "D", 14: "E", 15: "F", 16: "G", 17: "H", 18: "I", 19: "J", 20: "K",
            21: "L", 22: "M", 23: "N", 24: "O", 25: "P", 26: "Q", 27: "R", 28: "S", 29: "T", 30: "U",
            31: "V", 32: "W", 33: "X", 34: "Y", 35: "Z", 36: "a", 37: "b", 38: "c", 39: "d", 40: "e",
            41: "f", 42: "g", 43: "h", 44: "i", 45: "j", 46: "k", 47: "l", 48: "m", 49: "n", 50: "o",
            51: "p", 52: "q", 53: "r", 54: "s", 55: "t", 56: "u", 57: "v", 58: "w", 59: "x", 60: "y",
            61: "z"}
```

Taking the highest probability based on the image (letter or symbol) and printing the max index depends on the model predict



```
# Create a data frame containing the probability of each class
outputDf = pd.DataFrame(pred)
# Get the index of the maximum probability of the output data frame
maxIndex = list(outputDf.idxmax(axis=1))
# print the maximum index
print("Max index: ", maxIndex)
```

The output



```
Prediction Dict:  {'0': 0, '1': 1, '2': 2, '3': 3, '4':
1/1 [==============================] - 0s 215ms/step
Max index:  [45, 11, 19, 17, 46]
```

indexes have the highest probability

get best model index

Find the best accuracy and loss values for validation and training

```
minimum_val_loss = np.min(history.history['val_loss'])
best_model_index = np.where(history.history['val_loss'] == minimum_val_loss)[0][0]

best_loss = str(history.history['loss'][best_model_index])
best_acc = str(history.history['accuracy'][best_model_index])
best_val_loss = str(history.history['val_loss'][best_model_index])
best_val_acc = str(history.history['val_accuracy'][best_model_index])
```

```
print(best_loss)
print(best_acc)
print(best_val_acc)
```

The output

```
0.7138698697090149
0.7762886881828308
0.7838383913040161
```

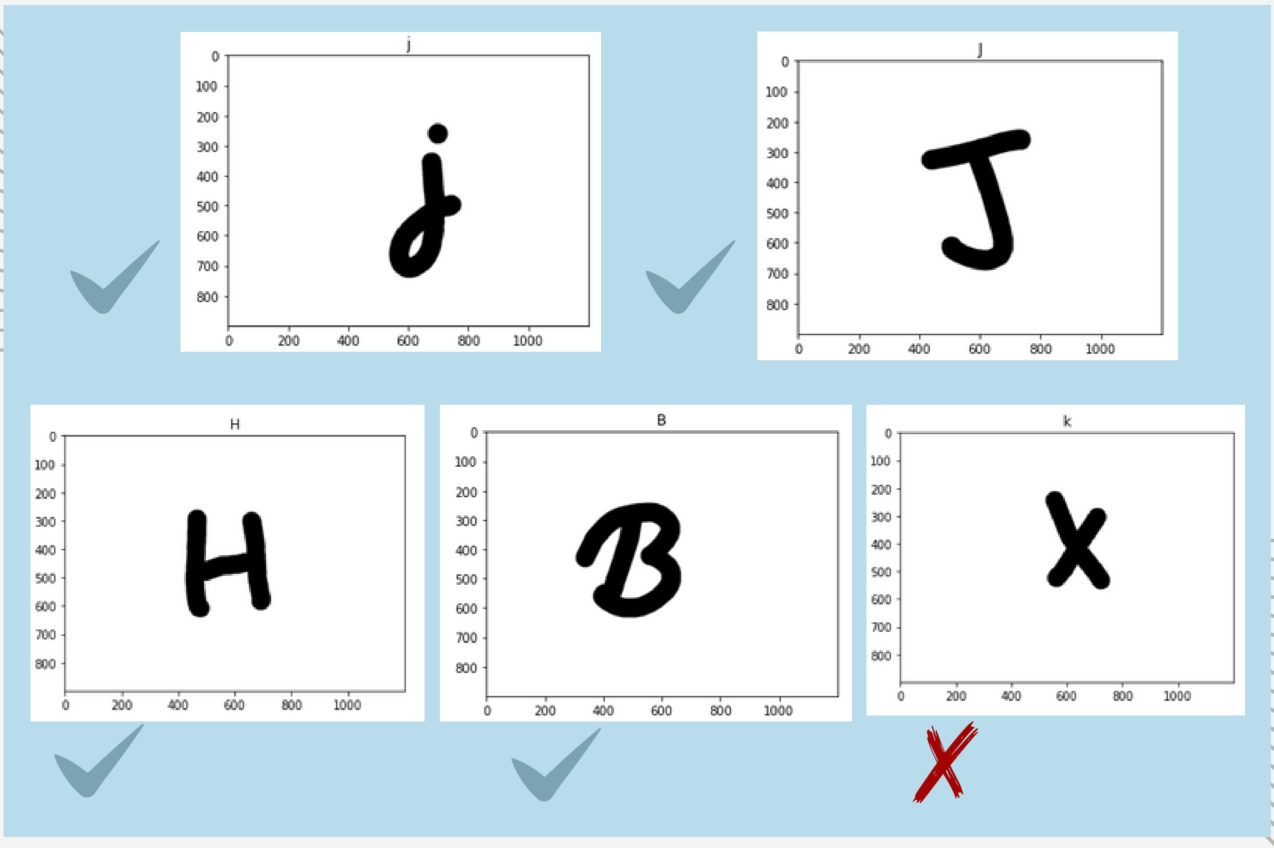
best loss

best accuracy

best validation accuracy

final results

After we applied the model many times, we did not get wrong results except in very few percentages



can say that the model worked correctly as we got a satisfactory result and only it got confused one time in similar letters in each test dataset

We have successfully developed Handwritten character recognition (Text Recognition) with Python, Tensorflow, and Machine Learning libraries. Handwritten characters have been recognized with more than 78% accuracy.

Conclusion

There are different styles of handwriting for the same language, which makes it more difficult to recognize some of the characters. But with the proper algorithm and the right tools and methods, we probably are able to develop the right model that can be able to recognize any set of characters that were written in some style. Even though the occurrence of handwritten with similar shapes and styles might challenge the model in detecting, and classifying them.

Before the process of building our model, we found out that the dataset consists of scanned processed images and the characters were isolated each character had its own set of images. The set images are images of different writing styles.