Sign Language Classification Using CNN

Prepared & Presented by: Habiba Hassan 18-11594, Jaida Adel 18-06393, Yasmin Amr 18-03452

Problem Statement: People who suffer from mutism are unable to blend into the community. They use hand signs to communicate, hence normal people face problems trying to understand them. AI technologies can play a crucial role in breaking down these communication barriers, it can contribute significantly to their social inclusion. In the wake of this news, our team has decided to carry this project.

Motivation: Communication is one of the basic requirements for survival in society. Our main goal is to make these people feel included, and cared for so that they can blend into the community and show their skills.

Data Description: Our data consists of 24 letters of the English sign language pictures. Their pixels were set into CSV files.

```
import os
import numpy as np #Math library
from PIL import Image #Python Imaging Library
%matplotlib inline
import matplotlib.pyplot as plt #Data Visulaization Library
from keras.utils.np_utils import to_categorical
from keras.preprocessing.image import ImageDataGenerator #Used for
from keras.applications.vgg16 import preprocess_input
import tensorflow #Framework that has the standard models
from keras.models import Sequential,Model
from tensorflow.keras.layers import Flatten,Dense,Dropout,LeakyReLU,ReLU,Conv2D, MaxPool2D,BatchNormalization
from tensorflow.keras.optimizers import Adam #Optimization Function
from sklearn.metrics import accuracy_score
from datetime import datetime
from keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import EarlyStopping
from keras.callbacks import ReduceLROnPlateau
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
```

```
#how to load data from kaggle
! pip install kaggle
! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: kaggle in /usr/local/lib/python3.8/dist-packages (1.5.12)
Requirement already satisfied: certifi in /usr/local/lib/python3.8/dist-packages (from kaggle) (2022.12.7)
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from kaggle) (2.25.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-packages (from kaggle) (4.64.1)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.8/dist-packages (from kaggle) (8.0.0)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.8/dist-packages (from kaggle) (1.24.3)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.8/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.8/dist-packages (from kaggle) (1.15.0)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.8/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests->kaggle) (4.0.0)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests->kaggle) (2.10)
mkdir: cannot create directory ‘/root/.kaggle’: File exists
```

```
!kaggle datasets download -d datamunge/sign-language-mnist -p /content/dataset
```

```
Downloading sign-language-mnist.zip to /content/dataset
 78% 49.0M/62.6M [00:00<00:00, 93.9MB/s]
100% 62.6M/62.6M [00:00<00:00, 96.6MB/s]
```

```
!unzip /content/dataset/sign-language-mnist.zip #Unzipping Data
```

```
Archive:  /content/dataset/sign-language-mnist.zip
  inflating: amer_sign2.png
  inflating: amer_sign3.png
  inflating: american_sign_language.PNG
  inflating: sign_mnist_test.csv
  inflating: sign_mnist_test/sign_mnist_test.csv
  inflating: sign_mnist_train.csv
  inflating: sign_mnist_train/sign_mnist_train.csv
```

```
train_df = pd.read_csv("/content/sign_mnist_train/sign_mnist_train.csv")#Reading the CSV files into the colab notebook
test_df = pd.read_csv("/content/sign_mnist_test/sign_mnist_test.csv")
```

```
y_train = train_df['label']
y_test = test_df['label']
del train_df['label']
del test_df['label']
```

```
from sklearn.preprocessing import LabelBinarizer
label_binarizer = LabelBinarizer()
y_train = label_binarizer.fit_transform(y_train)
y_test = label_binarizer.fit_transform(y_test)
```

```
x_train = train_df.values
x_test = test_df.values
```

```
# Normalize the data
x_train = x_train / 255 #Normalization
x_test = x_test / 255
# Reshaping the data from 1-D to 3-D as required through input by CNN's
```

```
x_train = x_train.reshape(-1,28,28,1)



# With data augmentation to prevent overfitting

#datagen = ImageDataGenerator(
#
#        rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
#        zoom_range = 0.1, # Randomly zoom image
#        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
#        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
#        horizontal_flip=False,  # randomly flip images
#        vertical_flip=False)  # randomly flip images
#datagen.fit(x_train)

datagen = ImageDataGenerator(
                                       zoom_range = 0.2,
                                       vertical_flip = True ,
                                       rotation_range=10,
                                       horizontal_flip = True,
                                width_shift_range=0.1,
                                  height_shift_range=0.1, )
datagen.fit(x_train)


learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy', patience = 2, verbose=1,factor=0.5, min_lr=0.00001)


model = Sequential()
model.add(Conv2D(75 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu' , input_shape = (28,28,1)))
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Flatten())
model.add(Dense(units = 512 , activation = 'relu'))
model.add(Dropout(0.3))
model.add(Dense(units = 24 , activation = 'softmax'))
model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics = ['accuracy'])
model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics = ['accuracy'])
history = model.fit(datagen.flow(x_train,y_train, batch_size = 128) ,epochs = 20 , validation_data = (x_test, y_test) )
```

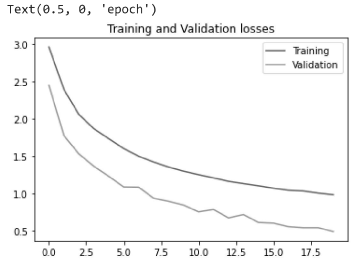
```
Epoch 1/20
215/215 [==============================] - 112s 500ms/step - loss: 2.9626 - accuracy: 0.1190 - val_loss: 2.4511 - val_accuracy: 0.2517
Epoch 2/20
215/215 [==============================] - 107s 499ms/step - loss: 2.3963 - accuracy: 0.2518 - val_loss: 1.7776 - val_accuracy: 0.4515
Epoch 3/20
215/215 [==============================] - 116s 539ms/step - loss: 2.0613 - accuracy: 0.3386 - val_loss: 1.5332 - val_accuracy: 0.5220
Epoch 4/20
215/215 [==============================] - 106s 494ms/step - loss: 1.8699 - accuracy: 0.3977 - val_loss: 1.3672 - val_accuracy: 0.5937
Epoch 5/20
215/215 [==============================] - 106s 493ms/step - loss: 1.7359 - accuracy: 0.4353 - val_loss: 1.2326 - val_accuracy: 0.6022
Epoch 6/20
215/215 [==============================] - 109s 507ms/step - loss: 1.6040 - accuracy: 0.4813 - val_loss: 1.0897 - val_accuracy: 0.6722
Epoch 7/20
215/215 [==============================] - 102s 474ms/step - loss: 1.5013 - accuracy: 0.5056 - val_loss: 1.0873 - val_accuracy: 0.6429
Epoch 8/20
215/215 [==============================] - 106s 490ms/step - loss: 1.4234 - accuracy: 0.5301 - val_loss: 0.9389 - val_accuracy: 0.7118
Epoch 9/20
215/215 [==============================] - 102s 476ms/step - loss: 1.3561 - accuracy: 0.5525 - val_loss: 0.8973 - val_accuracy: 0.7037
Epoch 10/20
215/215 [==============================] - 100s 464ms/step - loss: 1.3011 - accuracy: 0.5659 - val_loss: 0.8471 - val_accuracy: 0.7178
Epoch 11/20
215/215 [==============================] - 101s 470ms/step - loss: 1.2543 - accuracy: 0.5821 - val_loss: 0.7568 - val_accuracy: 0.7678
Epoch 12/20
215/215 [==============================] - 99s 461ms/step - loss: 1.2117 - accuracy: 0.5961 - val_loss: 0.7886 - val_accuracy: 0.7460
Epoch 13/20
215/215 [==============================] - 102s 477ms/step - loss: 1.1656 - accuracy: 0.6083 - val_loss: 0.6759 - val_accuracy: 0.7936
Epoch 14/20
215/215 [==============================] - 104s 483ms/step - loss: 1.1354 - accuracy: 0.6188 - val_loss: 0.7211 - val_accuracy: 0.7687
Epoch 15/20
215/215 [==============================] - 98s 457ms/step - loss: 1.1058 - accuracy: 0.6281 - val_loss: 0.6145 - val_accuracy: 0.8197
Epoch 16/20
215/215 [==============================] - 95s 440ms/step - loss: 1.0736 - accuracy: 0.6361 - val_loss: 0.6060 - val_accuracy: 0.8203
Epoch 17/20
215/215 [==============================] - 94s 437ms/step - loss: 1.0464 - accuracy: 0.6468 - val_loss: 0.5591 - val_accuracy: 0.8360
Epoch 18/20
215/215 [==============================] - 95s 442ms/step - loss: 1.0364 - accuracy: 0.6491 - val_loss: 0.5444 - val_accuracy: 0.8358
Epoch 19/20
215/215 [==============================] - 97s 449ms/step - loss: 1.0088 - accuracy: 0.6581 - val_loss: 0.5447 - val_accuracy: 0.8413
Epoch 20/20
215/215 [==============================] - 94s 437ms/step - loss: 0.9868 - accuracy: 0.6682 - val_loss: 0.4923 - val_accuracy: 0.8583
```

Model 1 Hyper Parameters:

- One Convolution Layer (Filter 3x3)
- One Pooling Layer (Filter 2x2)
- One Hidden Layer of 512 Neurons
- Output Layer of 24 Neurons For 24 Letters

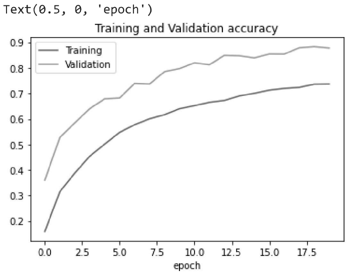
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['Training', 'Validation'])
plt.title('Training and Validation losses')
plt.xlabel('epoch')
```

```
Text(0.5, 0, 'epoch')
```



```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['Training', 'Validation'])
plt.title('Training and Validation accuracy')
plt.xlabel('epoch')
```

```
Text(0.5, 0, 'epoch')
```




```
model = Sequential()
model.add(Conv2D(75 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu' , input_shape = (28,28,1)))
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(50 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(Dropout(0.2))
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(25 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Flatten()) #Put data in an array form
model.add(Dense(units = 512 , activation = 'relu'))
model.add(Dropout(0.3))
model.add(Dense(units = 24 , activation = 'softmax'))
model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics = ['accuracy'])
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 28, 28, 75)        750

 max_pooling2d_2 (MaxPooling  (None, 14, 14, 75)       0
 2D)

 conv2d_3 (Conv2D)           (None, 14, 14, 50)        33800

 dropout_2 (Dropout)         (None, 14, 14, 50)        0

 max_pooling2d_3 (MaxPooling  (None, 7, 7, 50)         0
 2D)

 conv2d_4 (Conv2D)           (None, 7, 7, 25)          11275

 max_pooling2d_4 (MaxPooling  (None, 4, 4, 25)         0
 2D)

 flatten_2 (Flatten)         (None, 400)               0

 dense_4 (Dense)             (None, 512)               205312

 dropout_3 (Dropout)         (None, 512)               0

 dense_5 (Dense)             (None, 24)                12312

=================================================================
Total params: 263,449
Trainable params: 263,449
Non-trainable params: 0
_________________________________________________________________
```

Model 2 Hyper Parameters:

- Three Convolution Layers (Filter 3x3)
- Three Pooling Layers (Filter 2x2)
- One Hidden Layer of 512 Neurons
- Output Layer of 24 Neurons For 24 Letters

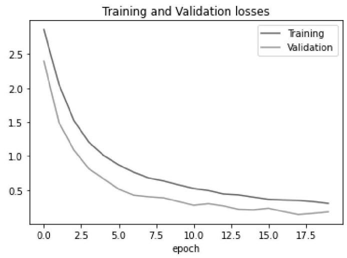
```
history = model.fit(datagen.flow(x_train,y_train, batch_size = 128) ,epochs = 20 , validation_data = (x_test, y_test) )
```

```
Epoch 1/20
215/215 [==============================] - 59s 272ms/step - loss: 2.8606 - accuracy: 0.1268 - val_loss: 2.3946 - val_accuracy: 0.2639
Epoch 2/20
215/215 [==============================] - 60s 280ms/step - loss: 2.0572 - accuracy: 0.3384 - val_loss: 1.4948 - val_accuracy: 0.5481
Epoch 3/20
215/215 [==============================] - 59s 274ms/step - loss: 1.5246 - accuracy: 0.4973 - val_loss: 1.0893 - val_accuracy: 0.6638
Epoch 4/20
215/215 [==============================] - 60s 277ms/step - loss: 1.2041 - accuracy: 0.5956 - val_loss: 0.8138 - val_accuracy: 0.7474
Epoch 5/20
215/215 [==============================] - 59s 274ms/step - loss: 1.0074 - accuracy: 0.6529 - val_loss: 0.6630 - val_accuracy: 0.8067
Epoch 6/20
215/215 [==============================] - 60s 280ms/step - loss: 0.8688 - accuracy: 0.7002 - val_loss: 0.5126 - val_accuracy: 0.8366
Epoch 7/20
215/215 [==============================] - 58s 269ms/step - loss: 0.7626 - accuracy: 0.7362 - val_loss: 0.4237 - val_accuracy: 0.8793
Epoch 8/20
215/215 [==============================] - 59s 274ms/step - loss: 0.6766 - accuracy: 0.7652 - val_loss: 0.4004 - val_accuracy: 0.8829
Epoch 9/20
215/215 [==============================] - 59s 274ms/step - loss: 0.6336 - accuracy: 0.7826 - val_loss: 0.3841 - val_accuracy: 0.8843
Epoch 10/20
215/215 [==============================] - 59s 276ms/step - loss: 0.5757 - accuracy: 0.8001 - val_loss: 0.3306 - val_accuracy: 0.9106
Epoch 11/20
215/215 [==============================] - 59s 276ms/step - loss: 0.5221 - accuracy: 0.8191 - val_loss: 0.2765 - val_accuracy: 0.9134
Epoch 12/20
215/215 [==============================] - 59s 274ms/step - loss: 0.4920 - accuracy: 0.8310 - val_loss: 0.2991 - val_accuracy: 0.9108
Epoch 13/20
215/215 [==============================] - 60s 278ms/step - loss: 0.4411 - accuracy: 0.8452 - val_loss: 0.2656 - val_accuracy: 0.9204
Epoch 14/20
215/215 [==============================] - 58s 268ms/step - loss: 0.4283 - accuracy: 0.8522 - val_loss: 0.2148 - val_accuracy: 0.9283
Epoch 15/20
215/215 [==============================] - 59s 274ms/step - loss: 0.3948 - accuracy: 0.8638 - val_loss: 0.2076 - val_accuracy: 0.9332
Epoch 16/20
```

```
215/215 [==============================] - 60s 280ms/step - loss: 0.3624 - accuracy: 0.8742 - val_loss: 0.2301 - val_accuracy: 0.9253
Epoch 17/20
215/215 [==============================] - 60s 277ms/step - loss: 0.3525 - accuracy: 0.8796 - val_loss: 0.1839 - val_accuracy: 0.9373
Epoch 18/20
215/215 [==============================] - 59s 276ms/step - loss: 0.3471 - accuracy: 0.8807 - val_loss: 0.1391 - val_accuracy: 0.9610
Epoch 19/20
215/215 [==============================] - 60s 281ms/step - loss: 0.3292 - accuracy: 0.8874 - val_loss: 0.1572 - val_accuracy: 0.9603
Epoch 20/20
215/215 [==============================] - 60s 279ms/step - loss: 0.3035 - accuracy: 0.8960 - val_loss: 0.1782 - val_accuracy: 0.9497
```

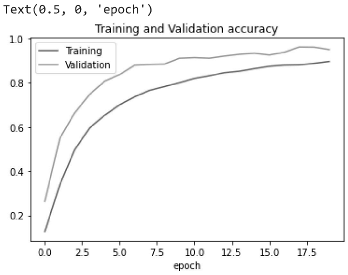
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['Training', 'Validation'])
plt.title('Training and Validation losses')
plt.xlabel('epoch')
```

```
Text(0.5, 0, 'epoch')
```




```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['Training', 'Validation'])
plt.title('Training and Validation accuracy')
plt.xlabel('epoch')
```

```
Text(0.5, 0, 'epoch')
```




```
model = Sequential()
model.add(Conv2D(75 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu' , input_shape = (28,28,1)))
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(50 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(Dropout(0.2))
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(50 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Flatten())
model.add(Dense(units = 512 , activation = 'relu'))
model.add(Dropout(0.3)) #Randomly selected neurons are ignored to improve processing & time to results
model.add(Dense(units = 256 , activation = 'relu'))
model.add(Dropout(0.3))
model.add(Dense(units = 24 , activation = 'softmax'))
model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics = ['accuracy'])
history = model.fit(datagen.flow(x_train,y_train, batch_size = 128) ,epochs = 20 , validation_data = (x_test, y_test) )
```

```
Epoch 1/20
215/215 [==============================] - 95s 440ms/step - loss: 2.8870 - accuracy: 0.1179 - val_loss: 2.2111 - val_accuracy: 0.3192
Epoch 2/20
215/215 [==============================] - 98s 454ms/step - loss: 1.9178 - accuracy: 0.3628 - val_loss: 1.1837 - val_accuracy: 0.6559
Epoch 3/20
215/215 [==============================] - 94s 438ms/step - loss: 1.2942 - accuracy: 0.5522 - val_loss: 0.7824 - val_accuracy: 0.7394
Epoch 4/20
215/215 [==============================] - 94s 435ms/step - loss: 0.9838 - accuracy: 0.6525 - val_loss: 0.5940 - val_accuracy: 0.8132
Epoch 5/20
215/215 [==============================] - 95s 441ms/step - loss: 0.8114 - accuracy: 0.7139 - val_loss: 0.4966 - val_accuracy: 0.8351
Epoch 6/20
215/215 [==============================] - 91s 422ms/step - loss: 0.6869 - accuracy: 0.7580 - val_loss: 0.3666 - val_accuracy: 0.8844
Epoch 7/20
215/215 [==============================] - 89s 415ms/step - loss: 0.5904 - accuracy: 0.7888 - val_loss: 0.4127 - val_accuracy: 0.8604
Epoch 8/20
215/215 [==============================] - 93s 434ms/step - loss: 0.5378 - accuracy: 0.8113 - val_loss: 0.3633 - val_accuracy: 0.8717
Epoch 9/20
215/215 [==============================] - 89s 415ms/step - loss: 0.4809 - accuracy: 0.8293 - val_loss: 0.2468 - val_accuracy: 0.9207
Epoch 10/20
215/215 [==============================] - 89s 413ms/step - loss: 0.4349 - accuracy: 0.8480 - val_loss: 0.2219 - val_accuracy: 0.9257
Epoch 11/20
215/215 [==============================] - 92s 426ms/step - loss: 0.3963 - accuracy: 0.8592 - val_loss: 0.1626 - val_accuracy: 0.9519
Epoch 12/20
215/215 [==============================] - 93s 434ms/step - loss: 0.3538 - accuracy: 0.8755 - val_loss: 0.1426 - val_accuracy: 0.9589
Epoch 13/20
215/215 [==============================] - 93s 434ms/step - loss: 0.3278 - accuracy: 0.8867 - val_loss: 0.1226 - val_accuracy: 0.9618
Epoch 14/20
215/215 [==============================] - 91s 422ms/step - loss: 0.3086 - accuracy: 0.8926 - val_loss: 0.0949 - val_accuracy: 0.9724
Epoch 15/20
215/215 [==============================] - 91s 424ms/step - loss: 0.2902 - accuracy: 0.8990 - val_loss: 0.1236 - val_accuracy: 0.9646
Epoch 16/20
215/215 [==============================] - 91s 424ms/step - loss: 0.2700 - accuracy: 0.9055 - val_loss: 0.0891 - val_accuracy: 0.9685
Epoch 17/20
215/215 [==============================] - 89s 415ms/step - loss: 0.2580 - accuracy: 0.9104 - val_loss: 0.0955 - val_accuracy: 0.9668
Epoch 18/20
215/215 [==============================] - 90s 417ms/step - loss: 0.2506 - accuracy: 0.9145 - val_loss: 0.0664 - val_accuracy: 0.9840
Epoch 19/20
215/215 [==============================] - 89s 413ms/step - loss: 0.2269 - accuracy: 0.9214 - val_loss: 0.0592 - val_accuracy: 0.9799
Epoch 20/20
215/215 [==============================] - 89s 415ms/step - loss: 0.2217 - accuracy: 0.9215 - val_loss: 0.0614 - val_accuracy: 0.9780
```

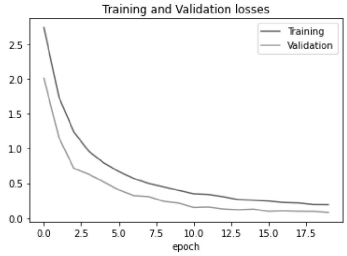
Model 3 Hyper Parameters:

- Three Convolution Layers (Filter 3x3)
- Three Pooling Layers (Filter 2x2)
- Two Hidden Layers of 512 Neurons & 256 Neurons
- Output Layer of 24 Neurons For 24 Letters

When you have training data, if you try to train your model too much, it might overfit, and when you get the actual test data for making predictions, it will not probably perform well. Dropout regularization is one technique used to tackle overfitting problems in deep learning.

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['Training', 'Validation'])
plt.title('Training and Validation losses')
plt.xlabel('epoch')
```

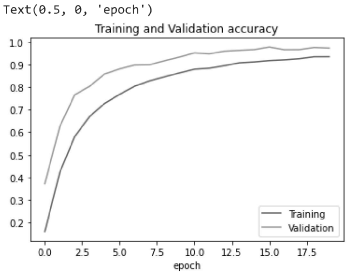
```
Text(0.5, 0, 'epoch')
```



Results Analysis: As shown above, the losses curve is decreasing with every epoch which means our model is successfully learning.

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['Training', 'Validation'])
plt.title('Training and Validation accuracy')
plt.xlabel('epoch')
```

```
Text(0.5, 0, 'epoch')
```



Results Analysis: As shown above, the accuracy curve is increasing with every epoch which means our model is successfully learning.

Comparison:

1. MODEL ONE: Although the validation accuracy increased with epochs, it only reached 0.8779.
2. MODEL TWO:Increasing the number of covolution & pooling layers has affected the validation accuracy that it jumped to 0.9497.
3. MODEL THREE: Adding more hidden layers i.e More neurons, the validation accuracy has shown greater improvement. It rose to 0.9780. This has proved that as the number of layers, and filters increase, also as different activation functions are put to trial with different types of data, the accuracy increases.

References:

- https://www.kaggle.com/
- https://www.tensorflow.org/tutorials/images/cnn
- https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7