MODEL BUILDING

Date	**10 November 2022**
Team ID	**PNT2022TMID50713**
Project Name	**A Gesture-based Tool for Sterile Browsing of Radiology Images**

In this step, we build Convolutional Neural Networking which contains a input layer along with the convolution, maxpooling and finally a output layer.

Importing The Model Building Libraries

```
import numpy as np

from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense,Flatten

from tensorflow.keras.layers import Conv2D,MaxPooling2D

from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Initializing The Model

Sequential model is a linear stack of layers. You can create a Sequential model by passing a list ɗlayer instances to the constructor: from keras. models import Sequential from keras as follows.

```
model = Sequential()
```

Adding CNN Layers

- We are adding a convolution layer with activation function as “relu” and with a small filter size (3,3) and number of filters (32) followed by a max pooling layer.
- Maxpool layer is used to downsample the input.
- Flatten layer flattens the input. Does not affect the batch size.

```
#First Convolution Layer and Pooling
model.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))
#Second Convolution Layer and Pooling
model.add(Conv2D(32, (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))
#Flattening the Layers
model.add(Flatten())
```

Adding Dense Layers

Dense layer is deeply connected neural network layer. It is most common and frequently usedlayer.

```
#Adding a fully connected layer
model.add(Dense(units = 128, activation = 'relu'))
model.add(Dense(units = 6, activation = 'softmax'))
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 62, 62, 32)	896
max_pooling2d (MaxPooling2D)	(None, 31, 31, 32)	0
conv2d_1 (Conv2D)	(None, 29, 29, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 32)	0
flatten (Flatten)	(None, 6272)	0
e (Dense)	ıe, 128)	ı44
e_1 (Dense)	ıe, 6)	

Total params: 813,862
Trainable params: 813,862
Non-trainable params: 0

Configure The Learning Process

- The compilation is the final step in creating a model. Once the compilation is done, we can move on to training phase. Loss function is used to find error or deviation in the learning process. Keras requires loss function during model compilation process.
- Optimization is an important process which optimize the input weights by comparing the prediction and the loss function. Here we are using adam optimizer
- Metrics is used to evaluate the performance of your model. It is similar to loss function, but not used in training process.

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

Train The Model

Now, let us train our model with our image dataset.

fit_generator functions - used to train a deep learning neural network

Arguments:

steps_per_epoch: It specifies the total number of steps taken from the generator as soon as one epoch is finished and next epoch has started. We can calculate the value of steps_per_epoch as the total number of

samples in your dataset divided by the batch size.
Epochs: an integer and number of epochs we want to train our model for.
validation_data can be either:

- an inputs and targets list
- a generator
- an inputs, targets, and sample_weights list which can be used to evaluatethe loss and metrics for any model after any epoch has ended.

validation_steps: only if the validation_data is a generator then only this argument

can be used. It specifies the total number of steps taken from the generator before it is stopped at every epoch and its value is calculated as the total number of validation data pointsin your dataset divided by the validation batch size.

```
model.fit_generator(x_train, steps_per_epoch = len(x_train), epochs = 5,
   validation_data = x_test, validation_steps = len(x_test))
```

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.

```
Epoch 1/5
119/119 [==============================] - 159s 1s/step - loss: 1.4068 - accuracy: 0.4175 - val_loss: 0.7373 - val_accuracy: 0.7000
Epoch 2/5
119/119 [==============================] - 6s 47ms/step - loss: 0.6362 - accuracy: 0.7340 - val_loss: 0.6075 - val_accuracy: 0.8000
Epoch 3/5
119/119 [==============================] - 6s 47ms/step - loss: 0.4165 - accuracy: 0.8434 - val_loss: 0.5313 - val_accuracy: 0.8000
Epoch 4/5
119/119 [==============================] - 6s 47ms/step - loss: 0.3142 - accuracy: 0.8973 - val_loss: 0.4669 - val_accuracy: 0.8333
Epoch 5/5
119/119 [==============================] - 6s 47ms/step - loss: 0.3073 - accuracy: 0.8838 - val_loss: 0.3033 - val_accuracy: 0.9000
<keras.callbacks.History at 0x7f150797dc90>
```

Save The Model

The model is saved with .h5 extension as follows:

- An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

```
model.save('gesture.h5')
model_json = model.to_json()
with open("model-bw.json", "w") as json_file:
json_file.write(model_json)
```

Test The Model

Evaluation is a process during development of the model to check whether the model is best fitfor the given problem and corresponding data.

Prediction

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
model = load_model("gesture.h5")
path = "/content/drive/MyDrive/Dataset/test/0/1.jpg"
```

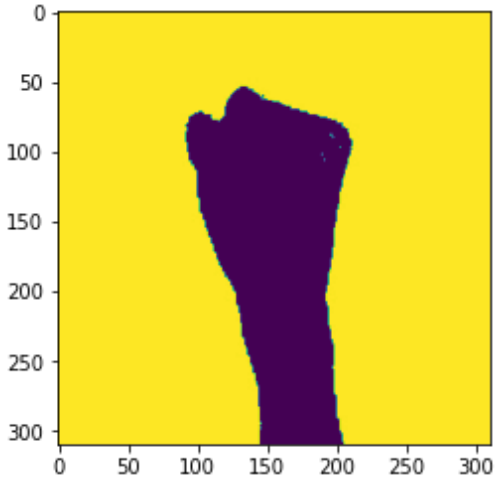
Plotting Images

```
%pylab inline
```

```
import matplotlib.pyplot as plt
```

```
import matplotlib.image as mpimgimgs = mpimg.imread(path)
imgplot = plt.imshow(imgs)plt.show()
```

Populating the interactive namespace from numpy and matplotlib



Loading the image

```
img = image.load_img(path, color_mode='grayscale’, target_size= (64,64))
```

```
x = image.img_to_array(img)#image to array
x.shape
type(x)
```

numpy.ndarray

Changing the shape

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

(1, 64, 64, 1)

Predicting the result

```
#predicting the classes

y_predict = np.argmax(model.predict(x_test), axis=1)
```

4/4 [==============================] - 0s 29ms/step

```
index=['0','1','2','3','4','5']

result=str(index[y_predict[0]]
)result
```

3

```
import numpy as np
p = []

for i in range(0,6):

   for j in range(0,5):

            img = image.load_img(path, color_mode = "grayscale", grayscale=True, target_size=(64,64,3))
            x = image.img_to_array(img)

            x = np.expand_dims(x,axis = 0)

            y_predict = np.argmax(model.predict(x_test))

            y_predict

print(p)
```

4/4 [==============================] - 0s 26ms/step
4/4 [==============================] - 0s 27ms/step
4/4 [==============================] - 0s 31ms/step
4/4 [==============================] - 0s 27ms/step
4/4 [==============================] - 0s 30ms/step
4/4 [==============================] - 0s 28ms/step
4/4 [==============================] - 0s 31ms/step
4/4 [==============================] - 0s 29ms/step
4/4 [==============================] - 0s 29ms/step
4/4 [==============================] - 0s 26ms/step
4/4 [==============================] - 0s 29ms/step
4/4 [==============================] - 0s 30ms/step
4/4 [==============================] - 0s 29ms/step
4/4 [==============================] - 0s 29ms/step
4/4 [==============================] - 0s 26ms/step
4/4 [==============================] - 0s 31ms/step
4/4 [==============================] - 0s 24ms/step
4/4 [==============================] - 0s 24ms/step
4/4 [==============================] - 0s 27ms/step
4/4 [==============================] - 0s 25ms/step
4/4 [==============================] - 0s 30ms/step
4/4 [==============================] - 0s 29ms/step
4/4 [==============================] - 0s 30ms/step
4/4 [==============================] - 0s 26ms/step
4/4 [==============================] - 0s 27ms/step

```
4/4 [==============================] - 0s 31ms/step
4/4 [==============================] - 0s 26ms/step
4/4 [==============================] - 0s 27ms/step
4/4 [==============================] - 0s 25ms/step
4/4 [==============================] - 0s 30ms/step
[]
```

```
result = []

index=['0','1','2','3','4','5']

for i in p:

  result.append(index[i[0]])
print(result)
```