Full Name:
Peter Sunny Shanthveer Markappa

Student Number:
R00208303

Subject:
Deep Learning

Assignment:
02

Data:
01-May-2022

University:
Munster Technology University

Department:
Computer Science

Course:
MSc in Artificial Intelligence

PART A: Convolutional Neural Networks:

PART A – 1:

Implementation and evaluation of baseline CNN and 3 architectures
Each models for 50 epochs

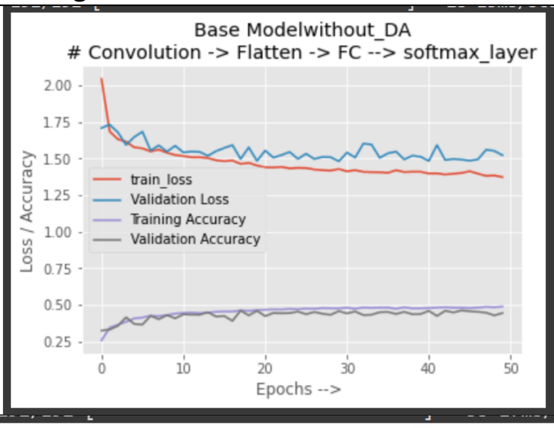
<table>
<tr><th>Model</th><th>Summary</th><th>Plotting</th></tr>
<tr><td>Implement abaseline CNN</td><td><pre>
Model: "sequential_10"
____
 Layer (type)   Output Shape   Param #
===
 conv2d_19 (Conv2D)   (None, 64, 64, 16)   448

 max_pooling2d_19 (MaxPoolin (None, 32, 32, 16)   0
 g2D)

 flatten_10 (Flatten)   (None, 16384)   0

 dense_20 (Dense)   (None, 32)   524320

 dense_21 (Dense)   (None, 9)   297

===
Total params: 525,065
Trainable params: 525,065
Non-trainable params: 0
</pre></td><td>


</td></tr>
<tr><td>Architecture – 1</td><td><pre>
Model: "sequential_11"
____
 Layer (type)   Output Shape   Param #
===
 conv2d_20 (Conv2D)   (None, 64, 64, 16)   448

 max_pooling2d_20 (MaxPoolin (None, 32, 32, 16)   0
 g2D)

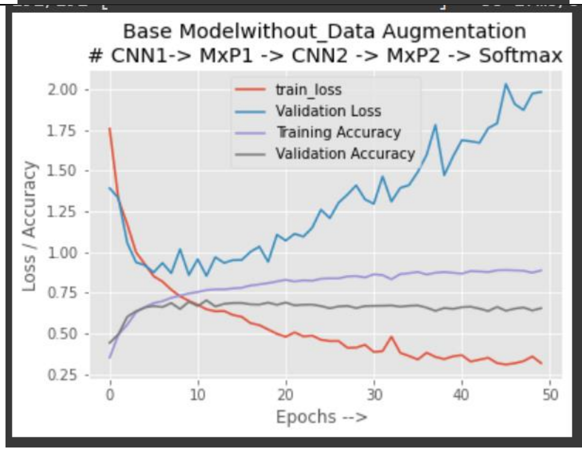
 conv2d_21 (Conv2D)   (None, 32, 32, 32)   4640

 max_pooling2d_21 (MaxPoolin (None, 16, 16, 32)   0
 g2D)

 flatten_11 (Flatten)   (None, 8192)   0

 dense_22 (Dense)   (None, 64)   524352

 dense_23 (Dense)   (None, 9)   585

===
Total params: 530,025
Trainable params: 530,025
Non-trainable params: 0
</pre></td><td>


</td></tr>
<tr><td>Architecture – 2</td><td><pre>
Model: "sequential"
____
 Layer (type)   Output Shape   Param #
===
 conv2d (Conv2D)   (None, 64, 64, 16)   448

 max_pooling2d (MaxPooling2D (None, 32, 32, 16)   0
)

 conv2d_1 (Conv2D)   (None, 32, 32, 32)   4640

 max_pooling2d_1 (MaxPooling (None, 16, 16, 32)   0
 2D)

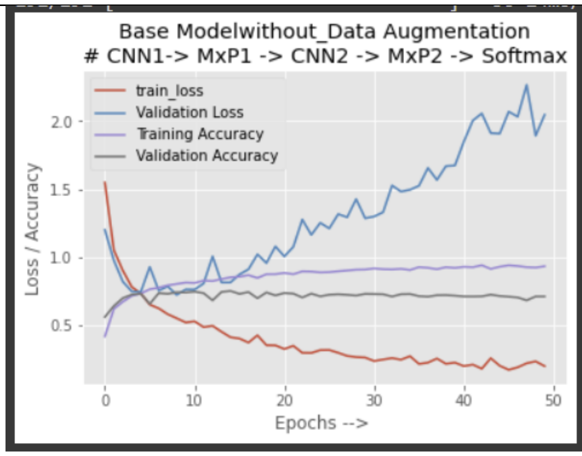
 conv2d_2 (Conv2D)   (None, 16, 16, 64)   18496

 max_pooling2d_2 (MaxPooling (None, 8, 8, 64)   0
 2D)

 flatten (Flatten)   (None, 4096)   0

 dense (Dense)   (None, 64)   262208

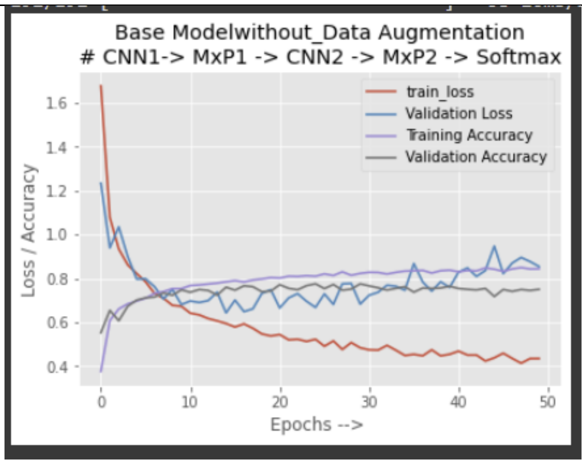
 dense_1 (Dense)   (None, 9)   585

===
Total params: 286,377
Trainable params: 286,377
Non-trainable params: 0
</pre></td><td>


</td></tr>
</table>

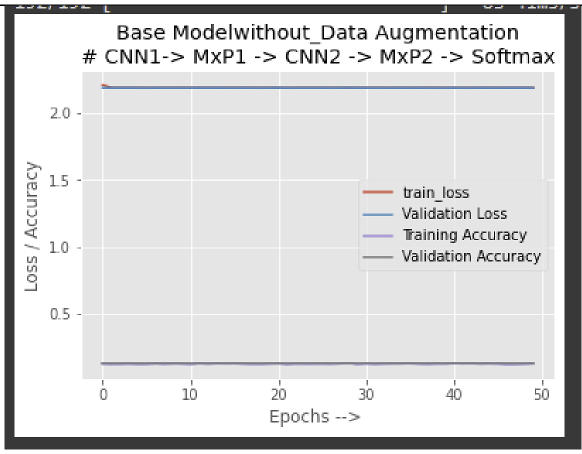
Architecture – 3

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 64, 64, 16)        448

 max_pooling2d_3 (MaxPooling  (None, 32, 32, 16)       0
 2D)

 conv2d_4 (Conv2D)           (None, 32, 32, 32)        4640

 max_pooling2d_4 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 16, 16, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_6 (Conv2D)           (None, 8, 8, 128)         73856

 max_pooling2d_6 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 flatten_1 (Flatten)         (None, 2048)              0

 dense_2 (Dense)             (None, 64)                131136

 dense_3 (Dense)             (None, 9)                 585

=================================================================
Total params: 229,161
Trainable params: 229,161
Non-trainable params: 0
```



Architecture – 4

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_7 (Conv2D)           (None, 64, 64, 16)        448

 max_pooling2d_7 (MaxPooling  (None, 32, 32, 16)       0
 2D)

 conv2d_8 (Conv2D)           (None, 32, 32, 32)        4640

 conv2d_9 (Conv2D)           (None, 32, 32, 48)        13872

 max_pooling2d_8 (MaxPooling  (None, 16, 16, 48)       0
 2D)

 conv2d_10 (Conv2D)          (None, 16, 16, 64)        27712

 conv2d_11 (Conv2D)          (None, 16, 16, 96)        55392

 max_pooling2d_9 (MaxPooling  (None, 8, 8, 96)         0
 2D)

 conv2d_12 (Conv2D)          (None, 8, 8, 128)         110720

 max_pooling2d_10 (MaxPoolin  (None, 4, 4, 128)        0
 g2D)

 flatten_2 (Flatten)         (None, 2048)              0

 dense_4 (Dense)             (None, 64)                131136

 dense_5 (Dense)             (None, 9)                 585

=================================================================
Total params: 344,505
Trainable params: 344,505
Non-trainable params: 0
```



Base line model showing the constant increasing in the accuracy where has the loss which was constant till 10 epochs is getting overfitted probably with more number of epochs it will increase.

```
loss: 1.3726 - accuracy: 0.4892 - val_loss: 1.5221 - val_accuracy: 0.4446
```

Architecture 1: validation loss is increasing constantly which means that the data is getting overfitted from 4 epochs so this architecture is discarded for data augmentation

```
loss: 0.3163 - accuracy: 0.8864 - val_loss: 1.9824 - val_accuracy: 0.6542
```

Architecture 2: even though the training loss is decreasing but the validation loss is constantly increasing also the gap between the training accuracy and validation is also increasing from epoch 2, so this architecture also overfitting the data

```
loss: 0.1982 - accuracy: 0.9322 - val_loss: 2.0462 - val_accuracy: 0.7102
```

Architecture 3: even though there is overfitting of data with this architecture when compared to baseline model, the gap between training loss and validation loss is less when compared to architecture 1 and architecture 2 also it can be observed that the accuracy has constant difference.

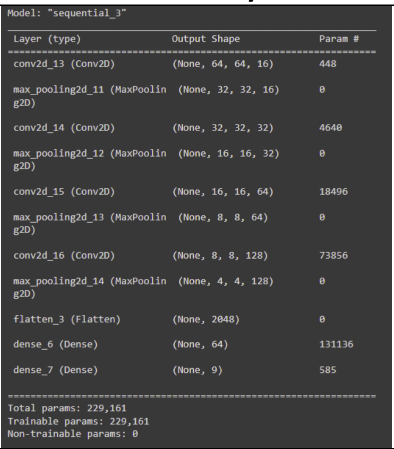
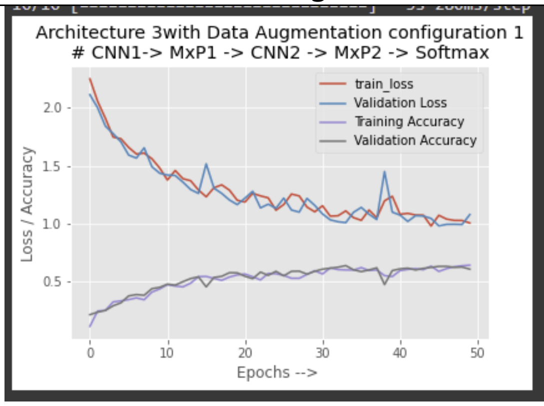
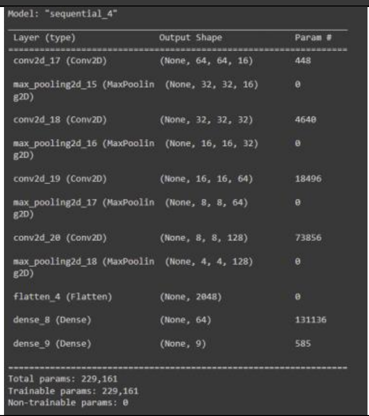
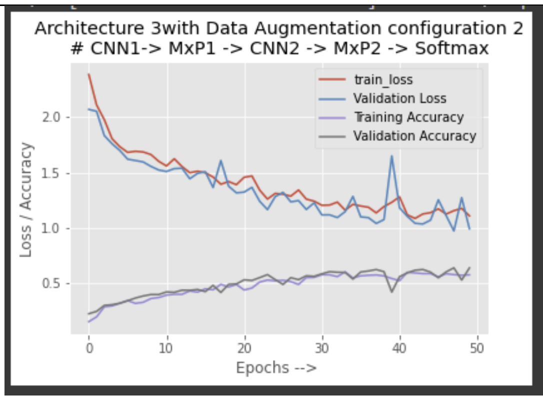
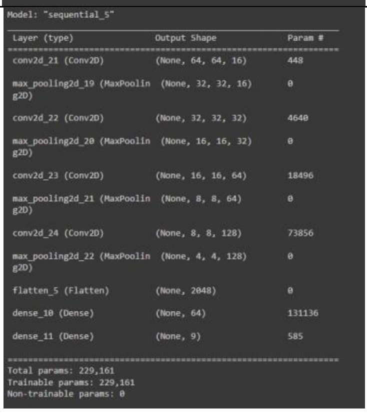
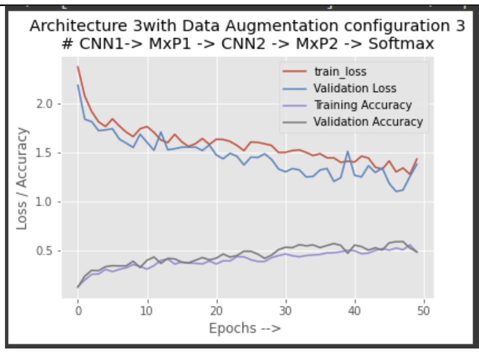
```
loss: 0.4351 - accuracy: 0.8439 - val_loss: 0.8544 - val_accuracy: 0.7504
```

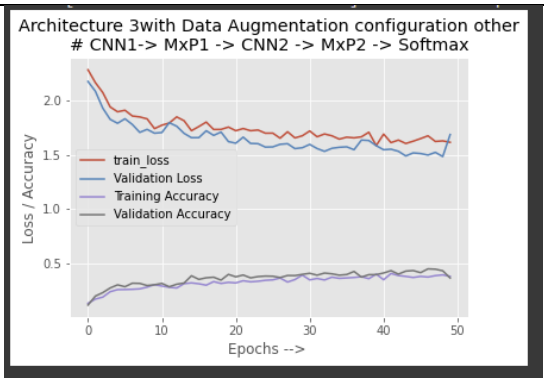
Architecture 4: This model is not performing well as everything is constant

```
loss: 2.1901 - accuracy: 0.1248 - val_loss: 2.1896 - val_accuracy: 0.1250
```

from above graph and description, the better performing model apart from baseline model is architecture 3, so this architecture will be considered for next phase that is for data augmentation. It can also be observed with result even though there is gap in the loss but the loss with this architecture is not as much with other architecture. So this architecture may perform well with data agumentation

Considering architecture 3 for the data augmentation

Configuration	Summary	Plotting
Configuration - 1	 Model: "sequential_3" Layer (type) — Output Shape — Param # conv2d_13 (Conv2D) — (None, 64, 64, 16) — 448 max_pooling2d_11 (MaxPooling2D) — (None, 32, 32, 16) — 0 conv2d_14 (Conv2D) — (None, 32, 32, 32) — 4640 max_pooling2d_12 (MaxPooling2D) — (None, 16, 16, 32) — 0 conv2d_15 (Conv2D) — (None, 16, 16, 64) — 18496 max_pooling2d_13 (MaxPooling2D) — (None, 8, 8, 64) — 0 conv2d_16 (Conv2D) — (None, 8, 8, 128) — 73856 max_pooling2d_14 (MaxPooling2D) — (None, 4, 4, 128) — 0 flatten_3 (Flatten) — (None, 2048) — 0 dense_6 (Dense) — (None, 64) — 131136 dense_7 (Dense) — (None, 9) — 585 Total params: 229,161 Trainable params: 229,161 Non-trainable params: 0	
Configuration – 2	 Model: "sequential_4" Layer (type) — Output Shape — Param # conv2d_17 (Conv2D) — (None, 64, 64, 16) — 448 max_pooling2d_15 (MaxPooling2D) — (None, 32, 32, 16) — 0 conv2d_18 (Conv2D) — (None, 32, 32, 32) — 4640 max_pooling2d_16 (MaxPooling2D) — (None, 16, 16, 32) — 0 conv2d_19 (Conv2D) — (None, 16, 16, 64) — 18496 max_pooling2d_17 (MaxPooling2D) — (None, 8, 8, 64) — 0 conv2d_20 (Conv2D) — (None, 8, 8, 128) — 73856 max_pooling2d_18 (MaxPooling2D) — (None, 4, 4, 128) — 0 flatten_4 (Flatten) — (None, 2048) — 0 dense_8 (Dense) — (None, 64) — 131136 dense_9 (Dense) — (None, 9) — 585 Total params: 229,161 Trainable params: 229,161 Non-trainable params: 0	
Configuration – 3	 Model: "sequential_5" Layer (type) — Output Shape — Param # conv2d_21 (Conv2D) — (None, 64, 64, 16) — 448 max_pooling2d_19 (MaxPooling2D) — (None, 32, 32, 16) — 0 conv2d_22 (Conv2D) — (None, 32, 32, 32) — 4640 max_pooling2d_20 (MaxPooling2D) — (None, 16, 16, 32) — 0 conv2d_23 (Conv2D) — (None, 16, 16, 64) — 18496 max_pooling2d_21 (MaxPooling2D) — (None, 8, 8, 64) — 0 conv2d_24 (Conv2D) — (None, 8, 8, 128) — 73856 max_pooling2d_22 (MaxPooling2D) — (None, 4, 4, 128) — 0 flatten_5 (Flatten) — (None, 2048) — 0 dense_10 (Dense) — (None, 64) — 131136 dense_11 (Dense) — (None, 9) — 585 Total params: 229,161 Trainable params: 229,161 Non-trainable params: 0	

<table>
<tr>
<td>Configuration - others</td>
<td>Model: "sequential_8"
Layer (type) Output Shape Param #
conv2d_33 (Conv2D) (None, 64, 64, 16) 448
max_pooling2d_31 (MaxPoolin g2D) (None, 32, 32, 16) 0
conv2d_34 (Conv2D) (None, 32, 32, 32) 4640
max_pooling2d_32 (MaxPoolin g2D) (None, 16, 16, 32) 0
conv2d_35 (Conv2D) (None, 16, 16, 64) 18496
max_pooling2d_33 (MaxPoolin g2D) (None, 8, 8, 64) 0
conv2d_36 (Conv2D) (None, 8, 8, 128) 73856
max_pooling2d_34 (MaxPoolin g2D) (None, 4, 4, 128) 0
flatten_8 (Flatten) (None, 2048) 0
dense_16 (Dense) (None, 64) 131136
dense_17 (Dense) (None, 9) 585
Total params: 229,161
Trainable params: 229,161
Non-trainable params: 0</td>
<td></td>
</tr>
</table>

From above plotting it can be observed that configuration 1 and 2 has training loss and validation loss between 1 and 1.25 so again those two configuration is chosen for more epochs to find best configuration where has other configuration has loss more than 1.5

configuration 1:

```
loss: 1.0068 - accuracy: 0.6430 - val_loss: 1.0795 - val_accuracy: 0.6087
```

configuration 2:

```
loss: 1.1072 - accuracy: 0.5740 - val_loss: 0.9908 - val_accuracy: 0.6363
```

configuration 3:

```
loss: 1.4348 - accuracy: 0.4810 - val_loss: 1.3811 - val_accuracy: 0.4858
```

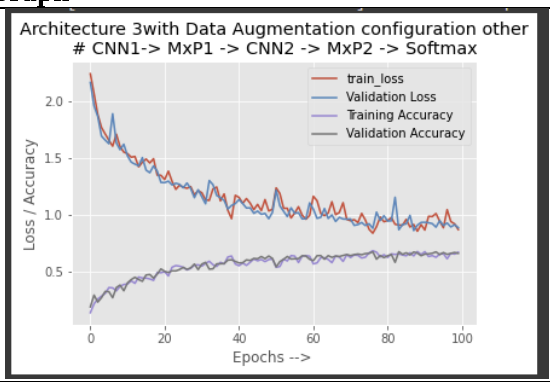
configuration other:

```
loss: 1.6170 - accuracy: 0.3810 - val_loss: 1.6883 - val_accuracy: 0.3663
```

from above loss and accuracy it can be seen that configuration 1 and configuration 2 has less overfitting of data with accuracy of more than 60% where as other configuration even though they are not overfitting still giving less accuracy.

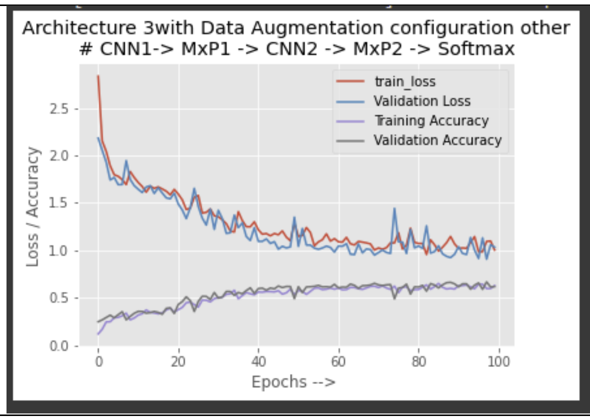
So let me consider configuration 1 and 2 for more epochs to check which of this configuration will perform best.

Running for 100 epochs for configuration 1 and configuration 2

<table>
<tr>
<th>Configuration</th>
<th>Summary</th>
<th>Graph</th>
</tr>
<tr>
<td>Configuration – 1</td>
<td>Model: "sequential_9"
Layer (type) Output Shape Param #
conv2d_37 (Conv2D) (None, 64, 64, 16) 448
max_pooling2d_35 (MaxPoolin g2D) (None, 32, 32, 16) 0
conv2d_38 (Conv2D) (None, 32, 32, 32) 4640
max_pooling2d_36 (MaxPoolin g2D) (None, 16, 16, 32) 0
conv2d_39 (Conv2D) (None, 16, 16, 64) 18496
max_pooling2d_37 (MaxPoolin g2D) (None, 8, 8, 64) 0
conv2d_40 (Conv2D) (None, 8, 8, 128) 73856
max_pooling2d_38 (MaxPoolin g2D) (None, 4, 4, 128) 0
flatten_9 (Flatten) (None, 2048) 0
dense_18 (Dense) (None, 64) 131136
dense_19 (Dense) (None, 9) 585
Total params: 229,161
Trainable params: 229,161
Non-trainable params: 0</td>
<td></td>
</tr>
</table>

Configuration - 2

```
max_pooling2d_39 (MaxPoolin  (None, 32, 32, 16)       0
g2D)

conv2d_42 (Conv2D)          (None, 32, 32, 32)       4640

max_pooling2d_40 (MaxPoolin  (None, 16, 16, 32)       0
g2D)

conv2d_43 (Conv2D)          (None, 16, 16, 64)       18496

max_pooling2d_41 (MaxPoolin  (None, 8, 8, 64)         0
g2D)

conv2d_44 (Conv2D)          (None, 8, 8, 128)        73856

max_pooling2d_42 (MaxPoolin  (None, 4, 4, 128)        0
g2D)

flatten_10 (Flatten)        (None, 2048)             0

dense_20 (Dense)            (None, 64)               131136

dense_21 (Dense)            (None, 9)                585

=================================================================
Total params: 229,161
Trainable params: 229,161
Non-trainable params: 0
```



configuration 1:

```
loss: 0.8750 - accuracy: 0.6760 - val_loss: 0.8919 - val_accuracy: 0.6656
```

configuration 2:

```
loss: 1.0023 - accuracy: 0.6270 - val_loss: 1.0324 - val_accuracy: 0.6175
```

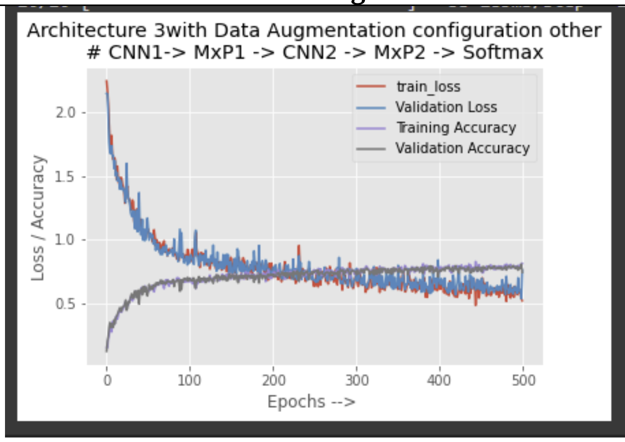
after running for 100 epochs, we can see that the data augmentation with architecture 3 and configuration 1 is performing better than the configuration 2, it can see that the configuration 1 has less training and validation loss when compared to configuration-2 and also the accuracy of training and validation of configuration 1 is more.
So, I'm considering configuration 1 of the data augmentation for 500 epochs

Running for 500 epochs for configuration 1

Summary:

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_45 (Conv2D)          (None, 64, 64, 16)        448

 max_pooling2d_43 (MaxPoolin  (None, 32, 32, 16)       0
 g2D)

 conv2d_46 (Conv2D)          (None, 32, 32, 32)        4640

 max_pooling2d_44 (MaxPoolin  (None, 16, 16, 32)       0
 g2D)

 conv2d_47 (Conv2D)          (None, 16, 16, 64)        18496

 max_pooling2d_45 (MaxPoolin  (None, 8, 8, 64)         0
 g2D)

 conv2d_48 (Conv2D)          (None, 8, 8, 128)         73856

 max_pooling2d_46 (MaxPoolin  (None, 4, 4, 128)        0
 g2D)

 flatten_11 (Flatten)        (None, 2048)              0

 dense_22 (Dense)            (None, 64)                131136

 dense_23 (Dense)            (None, 9)                 585

=================================================================
Total params: 229,161
Trainable params: 229,161
Non-trainable params: 0
```

Plotting




```
Epoch 499/500
10/10 [==============================] - 3s 283ms/step - loss: 0.5411 - accuracy:
0.7920 - val_loss: 0.5393 - val_accuracy: 0.8094
Epoch 500/500
10/10 [==============================] - 3s 285ms/step - loss: 0.5219 - accuracy:
0.8150 - val_loss: 0.7638 - val_accuracy: 0.7444
```

we can see that with 500 epochs the training loss and validation is almost reached to 0.5 and the accuracy between the training and validation is constant with more than 70%.
The final accuracy after 500 epochs for validation accuracy is 74%.

Architecture 3 with configuration 1 of data augmentation description:

```
def  CNN_configurations_3(class_label, trainX, trainY, valX, valY):
  input_s = trainX.shape[1:]

  # CNN1-> MxP1 -> CNN2 -> MxP2 -> CNN3 -> MxP3 -> CNN4 -> MxP4 -> Softmax
  architecture_3_model = Sequential()

  # 1st Layer
  architecture_3_model.add(Conv2D(16, (3,3), input_shape=input_s, activation = 'relu', padding= 'same'))
  architecture_3_model.add(MaxPooling2D(2, 2))

  # 2nd Layer
  architecture_3_model.add(Conv2D(32, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  architecture_3_model.add(MaxPooling2D(2, 2))

  # 3rd Layer
  architecture_3_model.add(Conv2D(64, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  architecture_3_model.add(MaxPooling2D(2, 2))

  # 4th Layer
  architecture_3_model.add(Conv2D(128, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  architecture_3_model.add(MaxPooling2D(2, 2))

  architecture_3_model.add(Flatten())
  architecture_3_model.add(Dense(64, activation = 'relu') )
  architecture_3_model.add(Dense(9, activation = 'softmax') )

  return architecture_3_model
```

DA Configuration1:

```
train_data = ImageDataGenerator(featurewise_center=False, samplewise_center=False,
featurewise_std_normalization=False, samplewise_std_normalization=False, zca_whiten
ing=False, zca_epsilon=1e-06,
              rotation_range=10, width_shift_range=0.1, height_shift_range=0.1,
              brightness_range=None, shear_range=0.0, zoom_range=0.0, channel_shift
_range=0.0, fill_mode='nearest', cval=0.0, horizontal_flip=False, vertical_flip=Fal
se, rescale=None, preprocessing_function=None,
              data_format=None, validation_split=0.0, dtype=None)
```

Conclusion for Part A-1:

The model build in this part gave best performance for 4 layer architecture with filters increasing from 16 to 128 and with dense layer of 64 and final with 9 by using **activation function as ReLu** and **Softmax** in the final output layer.
Data augmentation performed well with rotation of images with 45 degree keeping the horizontal and vertical shit as True

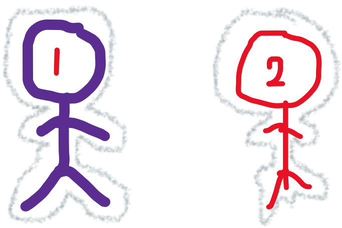
****************End of Coding of Part_A_1 ******************

Part A-1: Research

Research and describe two more recent data augmentation techniques (not currently offered by the ImageDataGenerator in Keras). Please note there is no need to implement these.

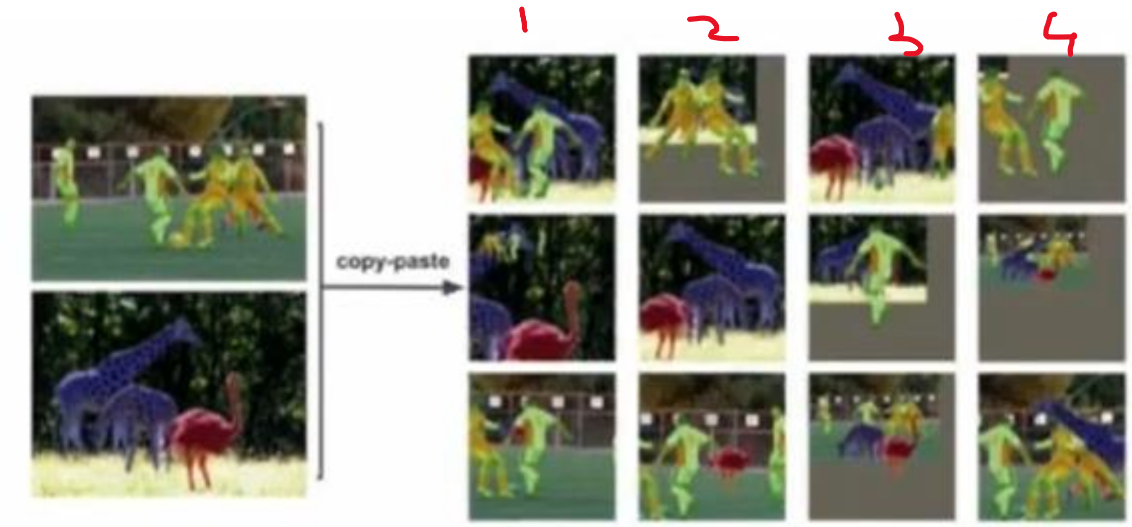
1) **Simple Copy-Paste Data Augmentation method for instance segmentation**
   https://arxiv.org/abs/2012.07177

   This is the very simple methodology that was implemented by Golnaz Ghiasi which is published in paper on 23rd June 2021. This paper explains the implementation of data augmentation for instance segmentation models in computer vision. In the below diagram description of instance segmentation is show by drawing and simple picture for better understanding of instance segmentation.



   Considering there are two images with each person in each image, and the black dotted lines show the instance segmentation of the image.
   So now the question is why do we need this? The reason for this method was Data, for example *22 worker hours were spend per 1000 instance masks for COCO segmentation*. So, it was important to develop new methods to improve data-efficiency of state-of-art instance segmentation models.

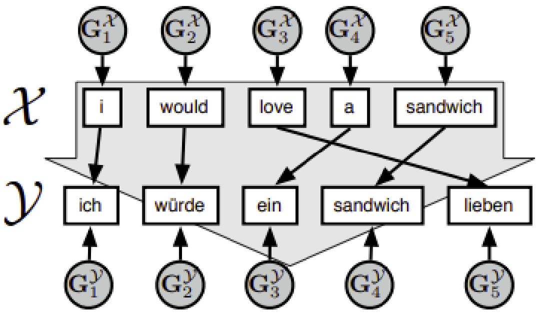
   What does this paper propose specifically? By using the image in the published paper will explain this concept



   From the above image on the left side there are two images one is football players and other wild animals, on the other hand in the right side it can be seen that on image is cropped and pasted onto another image, also it can be observed that number 3 image, if the image scale is large it is padded. This instance segmentation method does not care about the size or rotation of the images, this is very simple concept where parts of one image is cropped and pasted onto another.

2) Counterfactual Data Augmentation for Neural Machine Translation

https://aclanthology.org/2021.naacl-main.18/
https://aclanthology.org/2021.naacl-main.18.pdf



This method works by phrasal alignment and interpreting language models. It generates (path-specific) counterfactual aligned phrases to generate augmented parallel translation corpora. This is generated these by randomly sampling new source phrases from a masked language model, then randomly sampling an aligned counterfactual target phrase, keeping in mind that a translation language model can be interpreted as a GumbelMax Structural Causal Model (Oberst and Sontag, 2019)

In comparison to previous work, this method considers both context and alignment to maintain symmetry between source and target sequences. Experiments on IWSLT'15 English to Vietnamese, WMT'17 English to German, WMT'18 English to Turkish, and WMT'19 robust English to French demonstrate that the method can improve translation, backtranslation, and translation robustness.

This method of augmentation has 3 steps:

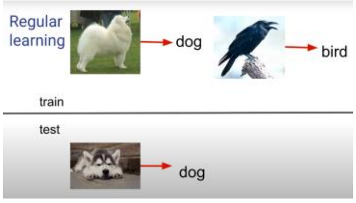
1) It usilise the unsupervised phrasal alignment (Neubig et al (2011)) and Dyer et al(2013) for getting communication between source and target phrases.
2) The source phrase will be deleted and according to the procedure of trained masked language model it will be resampled
3) Using path-specific counterfactual inference on a trained translation language model's causal model (Lample and Conneau, 2019) to resample only the aligned target phrase, given the changed source phrase.

******

A range of other solutions have been developed to facilitate building deep learning models for environments with limited data. Research and clearly describe one-shot and zero-shot learning. Given an example network and application for each category.

1) Zero-Shot Learning

It is challenge for modelling to learn without using data labelling, this learning involves little human intervention and the models will depend on the prior trained data and the additional existing data. zero-shot learning reduced the effort and time which the data labelling takes, in this learning instead of giving the trained examples, it gives high level description of the new features categories such that the machine will relate it to the existing trained categories.



To explain this more below I have used the image, this image seems not recognisable, basically we can visit to Wikipedia page and type Wampimuk which is domain ontology name with the description small, horns, furry and cute. So, the whole point of **zero shot learning** is

1) recognition of pattern with no training examples

2) this will be solved by some semantic transfer knowledge.

https://www.researchgate.net/publication/270878296_Is_this_a_wampimuk_Cross-modal_mapping_between_distributional_semantics_and_the_visual_world

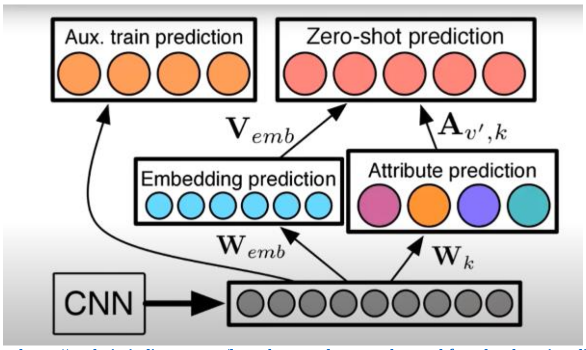
What is need for zero-shot learning in ML?

Vast and growing Number of categories

a) Collecting and annotating examples is not possible especially with deep learning scale where in real world every day new categories emerges.
   Example:
   1) Object recognition
   2) Cross-lingual dictionary induction: where every pair of languages has words and every word is a category
   3) FMRI mind reading

Application:

Zero short learning is used in critical fields like healthcare for medical imaging, chest x-ray for COVID19 diagnosis, also used in the object detection in self-driving vehicles, computer vision, natural language processing and machine perception.



https://analyticsindiamag.com/how-do-zero-shot-one-shot-and-few-shot-learning-differ/

2) One-Shot Learning

(Reference: https://analyticsindiamag.com/how-do-zero-shot-one-shot-and-few-shot-learning-differ/)
(Paper Reference: https://blog.acolyer.org/2017/01/03/matching-networks-for-one-shot-learning/)

This learning will perform the classification based on the past data which is provided to the machine. This learning is having application in **facial recognition technology** which includes facial verification and identification. **Example** for this is face recognition in mobile phones, face recognitions in the offices.
Face embedding, a rich low-dimensional feature representation, is learned by facial recognition systems. The Siamese network approach has been used in one-shot learning. Siamese networks were eventually compared to comparative loss functions, and the triplet loss function was proven to be superior, and the FaceNet system began using them. For high-quality face embeddings, which have become the foundation for modern facial recognition, the contrastive loss and triplet loss functions are now used.

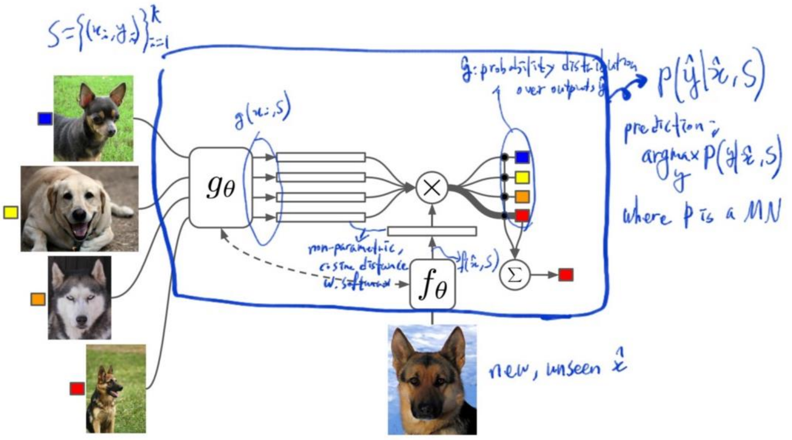


Image source: https://blog.acolyer.org/2017/01/03/matching-networks-for-one-shot-learning/
(Image is used from the above link and all other written statement was done by myself)

The author begins his concept by defining the model for one-shot learning. Given a support set S, the model defines a cs (or classifier) function for each S, i.e., a mapping S-> cs (.).

The author describes his model architecture concept in one main section and two subsections: the main section describes model definition on S and non-parametric kernel a(x, xi), and the subsections describe attention kernel a(x, xi) and full context embedding on the view of using LSTM and set S.
Note: Paper reference link is mentioned above

********************************* End of Part A – 1 **

--Next Page for Part_A_2--

PART A – 2:

Build a CNN ensemble containing a maximum of 10 base learners

4 different models were built

Model 1

Base Learner - 1

```python
# Architecture 1

def  base_learner_model_1(class_label, input_model):

  base_learner_1 = Sequential()

  # 1st Layer
  base_learner_1.add(Conv2D(16, (3,3), input_shape=input_model, activation = 'relu', padding= 'same'))
  base_learner_1.add(MaxPooling2D(2, 2))

  # 2nd Layer
  base_learner_1.add(Conv2D(32, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_1.add(Conv2D(48, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_1.add(Conv2D(64, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_1.add(MaxPooling2D(2, 2))

  # 3nd Layer
  base_learner_1.add(Conv2D(96, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))

  base_learner_1.add(Flatten())
  base_learner_1.add(Dense(32, activation = 'relu') )
  base_learner_1.add(Dense(9, activation = 'softmax') )

  return base_learner_1
```

Model 2

Base Learner - 2

```python
# Architecture 2

def  base_learner_model_2(class_label, input_model):

  base_learner_2 = Sequential()

  # 1st Layer
  base_learner_2.add(Conv2D(16, (3,3), input_shape=input_model, activation = 'relu', padding= 'same'))
  base_learner_2.add(MaxPooling2D(2, 2))

  # 2nd Layer
  base_learner_2.add(Conv2D(32, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_2.add(Conv2D(48, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_2.add(Conv2D(64, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_2.add(MaxPooling2D(2, 2))

  # 3nd Layer
  base_learner_2.add(Conv2D(96, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))

  # 4th Layer
  base_learner_2.add(Conv2D(32, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_2.add(Conv2D(64, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))

  base_learner_2.add(Flatten())
  base_learner_2.add(Dense(256, activation = 'relu') )
  base_learner_2.add(Dense(9, activation = 'softmax') )

  return base_learner_2
```

Model 3

Base Learner - 3

```python
# Architecture 3

def  base_learner_model_3(class_label, input_model):

  base_learner_3 = Sequential()

  # 1st Layer
  base_learner_3.add(Conv2D(16, (3,3), input_shape=input_model, activation = 'relu', padding= 'same'))
  base_learner_3.add(MaxPooling2D(2, 2))

  # 2nd Layer
  base_learner_3.add(Conv2D(32, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_3.add(Conv2D(48, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_3.add(Conv2D(64, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_3.add(MaxPooling2D(2, 2))

  # 3nd Layer
  base_learner_3.add(Conv2D(32, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_3.add(Conv2D(64, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_3.add(Conv2D(96, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))

  # 4th Layer
  base_learner_3.add(Conv2D(32, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_3.add(Conv2D(64, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_3.add(Conv2D(96, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_3.add(Conv2D(128, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))

  # 5th Layer
  base_learner_3.add(Conv2D(256, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))


  base_learner_3.add(Flatten())
  base_learner_3.add(Dense(64, activation = 'relu') )
  base_learner_3.add(Dense(9, activation = 'softmax') )

  return base_learner_3
```

Model 4

Base Learner - 4

```python
# Architecture 4

def  base_learner_model_4(class_label, input_model):

  base_learner_4 = Sequential()

  # 1st Layer
  base_learner_4.add(Conv2D(16, (3,3), input_shape=input_model, activation = 'relu', padding= 'same'))
  base_learner_4.add(MaxPooling2D(2, 2))

  # 2nd Layer
  base_learner_4.add(Conv2D(32, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_4.add(Conv2D(48, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_4.add(Conv2D(64, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_4.add(MaxPooling2D(2, 2))

  # 3nd Layer
  base_learner_4.add(Conv2D(32, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_4.add(Conv2D(64, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_4.add(Conv2D(96, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))

  # 4th Layer
  base_learner_4.add(Conv2D(32, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_4.add(Conv2D(64, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_4.add(Conv2D(96, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_4.add(Conv2D(128, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))

  # 5th Layer
  base_learner_4.add(Conv2D(64, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_4.add(Conv2D(128, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))


  # 6th Layer
  base_learner_4.add(Conv2D(64, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_4.add(Conv2D(96, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_4.add(Conv2D(128, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_4.add(Conv2D(256, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))
  base_learner_4.add(Conv2D(512, (3,3), input_shape=(64,64,3), activation = 'relu', padding= 'same'))


  base_learner_4.add(Flatten())
  base_learner_4.add(Dense(256, activation = 'relu') )
  base_learner_4.add(Dense(9, activation = 'softmax') )

  return base_learner_4
```

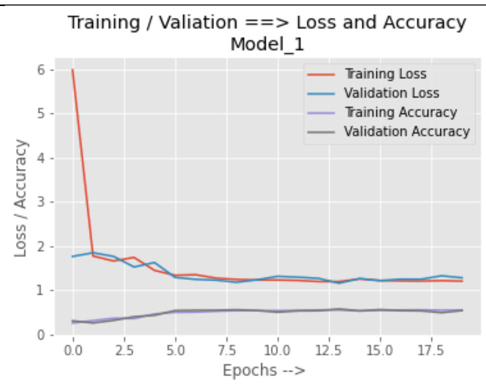
All the model were run for 20 epochs

Model_1

Model: "sequential_23"

Layer (type)	Output Shape	Param #
conv2d_203 (Conv2D)	(None, 64, 64, 16)	448
max_pooling2d_46 (MaxPooling2D)	(None, 32, 32, 16)	0
conv2d_204 (Conv2D)	(None, 32, 32, 32)	4640
conv2d_205 (Conv2D)	(None, 32, 32, 48)	13872
conv2d_206 (Conv2D)	(None, 32, 32, 64)	27712
max_pooling2d_47 (MaxPooling2D)	(None, 16, 16, 64)	0
conv2d_207 (Conv2D)	(None, 16, 16, 96)	55392
flatten_23 (Flatten)	(None, 24576)	0
dense_46 (Dense)	(None, 32)	786464
dense_47 (Dense)	(None, 9)	297

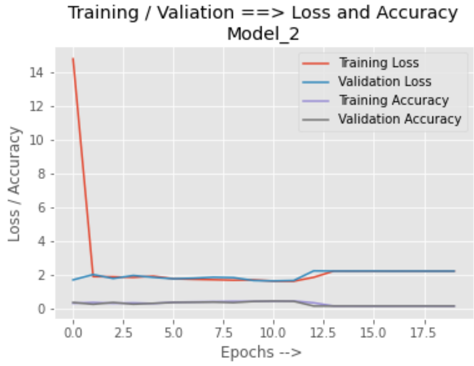
Total params: 888,825
Trainable params: 888,825
Non-trainable params: 0




```
Epoch 20: val_loss did not improve from 1.15221
600/600 [==============================] - 13s 21ms/step - loss: 1.1993 - accuracy: 0.5454 - val_loss: 1.2760 - val_accuracy: 0.5323
Valiation Loss 1.1522053480148315
 Valiation Accuracy 0.5699999928474426
Model: "sequential_24"
```

Model-2

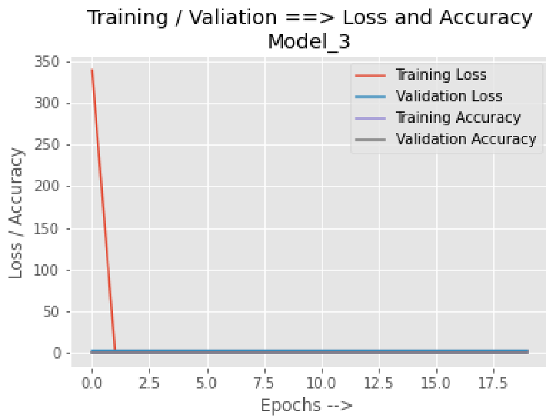
```
Model: "sequential_24"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_208 (Conv2D)         (None, 64, 64, 16)        448

 max_pooling2d_48 (MaxPoolin  (None, 32, 32, 16)       0
 g2D)

 conv2d_209 (Conv2D)         (None, 32, 32, 32)        4640

 conv2d_210 (Conv2D)         (None, 32, 32, 48)        13872

 conv2d_211 (Conv2D)         (None, 32, 32, 64)        27712

 max_pooling2d_49 (MaxPoolin  (None, 16, 16, 64)       0
 g2D)

 conv2d_212 (Conv2D)         (None, 16, 16, 96)        55392

 conv2d_213 (Conv2D)         (None, 16, 16, 32)        27680

 conv2d_214 (Conv2D)         (None, 16, 16, 64)        18496

 flatten_24 (Flatten)        (None, 16384)             0

 dense_48 (Dense)            (None, 256)               4194560

 dense_49 (Dense)            (None, 9)                 2313

=================================================================
Total params: 4,345,113
Trainable params: 4,345,113
Non-trainable params: 0
_________________________________________________________________
```




```
Epoch 20: val_loss did not improve from 1.60900
600/600 [==============================] - 15s 26ms/step - loss: 2.1920 - accuracy: 0.1203 - val_loss: 2.1922 - val_accuracy: 0.1250
Valiation Loss 1.6089953184127808
 Valiation Accuracy 0.41749998927116394
Model: "sequential_25"
```

Model-3

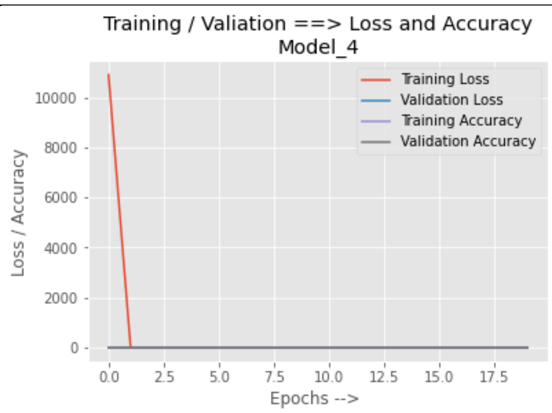
```
Model: "sequential_25"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_215 (Conv2D)         (None, 64, 64, 16)        448

 max_pooling2d_50 (MaxPoolin  (None, 32, 32, 16)       0
 g2D)

 conv2d_216 (Conv2D)         (None, 32, 32, 32)        4640

 conv2d_217 (Conv2D)         (None, 32, 32, 48)        13872

 conv2d_218 (Conv2D)         (None, 32, 32, 64)        27712

 max_pooling2d_51 (MaxPoolin  (None, 16, 16, 64)       0
 g2D)

 conv2d_219 (Conv2D)         (None, 16, 16, 32)        18464

 conv2d_220 (Conv2D)         (None, 16, 16, 64)        18496

 conv2d_221 (Conv2D)         (None, 16, 16, 96)        55392

 conv2d_222 (Conv2D)         (None, 16, 16, 32)        27680

 conv2d_223 (Conv2D)         (None, 16, 16, 64)        18496

 conv2d_224 (Conv2D)         (None, 16, 16, 96)        55392

 conv2d_225 (Conv2D)         (None, 16, 16, 128)       110720

 conv2d_226 (Conv2D)         (None, 16, 16, 256)       295168

 flatten_25 (Flatten)        (None, 65536)             0

 dense_50 (Dense)            (None, 64)                4194368

 dense_51 (Dense)            (None, 9)                 585

=================================================================
Total params: 4,841,433
Trainable params: 4,841,433
Non-trainable params: 0
_________________________________________________________________
```




```
Epoch 20: val_loss did not improve from 2.18964
600/600 [==============================] - 24s 40ms/step - loss: 2.1906 - accuracy: 0.1263 - val_loss: 2.1899 - val_accuracy: 0.1250
Valiation Loss 2.1896419525146484
 Valiation Accuracy 0.125
```

Model-3

```
Model: "sequential_26"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_227 (Conv2D)         (None, 64, 64, 16)        448

 max_pooling2d_52 (MaxPoolin  (None, 32, 32, 16)       0
 g2D)

 conv2d_228 (Conv2D)         (None, 32, 32, 32)        4640

 conv2d_229 (Conv2D)         (None, 32, 32, 48)        13872

 conv2d_230 (Conv2D)         (None, 32, 32, 64)        27712

 max_pooling2d_53 (MaxPoolin  (None, 16, 16, 64)       0
 g2D)

 conv2d_231 (Conv2D)         (None, 16, 16, 32)        18464

 conv2d_232 (Conv2D)         (None, 16, 16, 64)        18496

 conv2d_233 (Conv2D)         (None, 16, 16, 96)        55392

 conv2d_234 (Conv2D)         (None, 16, 16, 32)        27680

 conv2d_235 (Conv2D)         (None, 16, 16, 64)        18496

 conv2d_236 (Conv2D)         (None, 16, 16, 96)        55392

 conv2d_237 (Conv2D)         (None, 16, 16, 128)       110720

 conv2d_238 (Conv2D)         (None, 16, 16, 64)        73792

 conv2d_239 (Conv2D)         (None, 16, 16, 128)       73856

 conv2d_240 (Conv2D)         (None, 16, 16, 64)        73792

 conv2d_241 (Conv2D)         (None, 16, 16, 96)        55392

 conv2d_242 (Conv2D)         (None, 16, 16, 128)       110720

 conv2d_243 (Conv2D)         (None, 16, 16, 256)       295168

 conv2d_244 (Conv2D)         (None, 16, 16, 512)       1180160

 flatten_26 (Flatten)        (None, 131072)            0

 dense_52 (Dense)            (None, 256)               33554688

 dense_53 (Dense)            (None, 9)                 2313

=================================================================
Total params: 35,771,193
Trainable params: 35,771,193
Non-trainable params: 0
```




```
Epoch 20: val_loss did not improve from 2.18961
600/600 [==============================] - 50s 84ms/step - loss: 2.1902 - accuracy: 0.1259 - val_loss: 2.1912 - val_accuracy: 0.1250
Valiation Loss 2.1896119117736816
 Valiation Accuracy 0.125
****************************************
```

```
*******************************************
ensemble_accuracy_score =  0.568125
*******************************************
```

Evaluation of Part A_2:

Model_1:
model one performance we can see that the both training loss and validation loss is maintaining constant rate it means that the data is not getting overfitted infact the accuracy has increased after 4th epoch and increasing slowly
with final validation accuracy 52%. by increasing the number of epochs it accuracy can be improved
checkpoints:

```
Epoch 1/20
598/600 [============================>.] - ETA: 0s - loss: 6.0023 - accuracy: 0.2492
Epoch 1: val_loss improved from inf to 1.75829, saving model to weights/Model_1.h5
600/600 [==============================] - 14s 23ms/step - loss: 5.9883 - accuracy: 0.2493 - val_loss: 1.7583 - val_accuracy: 0.2994
Epoch 2/20
600/600 [==============================] - ETA: 0s - loss: 1.7682 - accuracy: 0.3052
Epoch 2: val_loss did not improve from 1.75829
600/600 [==============================] - 14s 23ms/step - loss: 1.7682 - accuracy: 0.3052 - val_loss: 1.8428 - val_accuracy: 0.2519
```

```
Epoch 5: val_loss did not improve from 1.52068
600/600 [==============================] - 13s 21ms/step - loss: 1.4432 - accuracy: 0.4568 - val_loss: 1.6208 - val_accuracy: 0.4252
Epoch 6/20
598/600 [============================>.] - ETA: 0s - loss: 1.3326 - accuracy: 0.4926
Epoch 6: val_loss improved from 1.52068 to 1.28420, saving model to weights/Model_1.h5
600/600 [==============================] - 12s 21ms/step - loss: 1.3321 - accuracy: 0.4928 - val_loss: 1.2842 - val_accuracy: 0.5325
```

From above we can observe that validation loss improved only in the epoch 6 and after that it didn't improve

Model_2:
this model has also performed like the model 1 with training loss and validation but it accuracy has dropped after 11 epoch.

```
Epoch 1/20
599/600 [============================>.] - ETA: 0s - loss: 14.7861 - accuracy: 0.2896
Epoch 1: val_loss improved from inf to 1.67337, saving model to weights/Model_2.h5
600/600 [==============================] - 17s 26ms/step - loss: 14.7648 - accuracy: 0.2897 - val_loss: 1.6734 - val_accuracy: 0.3358
```

```
Epoch 20: val_loss did not improve from 1.60900
600/600 [==============================] - 15s 26ms/step - loss: 2.1920 - accuracy: 0.1203 - val_loss: 2.1922 - val_accuracy: 0.1250
Valiation Loss 1.6089953184127808
 Valiation Accuracy 0.41749998927116394
```

This model loss never improved when compared to epoch 1

Model_3:
this model has not performed well, it can been that the training loss has dropped drastically were as the validation loss is remained constant along with accuracy, this means that here the training data is not learning properly data is getting underfitted

```
Epoch 1/20
599/600 [============================>.] - ETA: 0s - loss: 339.7745 - accuracy: 0.1245
Epoch 1: val_loss improved from inf to 2.19011, saving model to weights/Model_3.h5
600/600 [==============================] - 26s 41ms/step - loss: 339.2119 - accuracy: 0.1244 - val_loss: 2.1901 - val_accuracy: 0.1250
Epoch 2/20
599/600 [============================>.] - ETA: 0s - loss: 2.1904 - accuracy: 0.1265
Epoch 2: val_loss improved from 2.19011 to 2.18983, saving model to weights/Model_3.h5
600/600 [==============================] - 25s 41ms/step - loss: 2.1905 - accuracy: 0.1264 - val_loss: 2.1898 - val_accuracy: 0.1250
Epoch 3/20
599/600 [============================>.] - ETA: 0s - loss: 2.1907 - accuracy: 0.1205
Epoch 3: val_loss improved from 2.18983 to 2.18964, saving model to weights/Model_3.h5
600/600 [==============================] - 24s 41ms/step - loss: 2.1907 - accuracy: 0.1206 - val_loss: 2.1896 - val_accuracy: 0.1250
Epoch 4/20
599/600 [============================>.] - ETA: 0s - loss: 2.1906 - accuracy: 0.1246
Epoch 4: val_loss did not improve from 2.18964
600/600 [==============================] - 24s 41ms/step - loss: 2.1906 - accuracy: 0.1246 - val_loss: 2.1904 - val_accuracy: 0.1250
```

```
Epoch 20: val_loss did not improve from 2.18964
600/600 [==============================] - 24s 40ms/step - loss: 2.1906 - accuracy: 0.1263 - val_loss: 2.1899 - val_accuracy: 0.1250
Valiation Loss 2.1896419525146484
 Valiation Accuracy 0.125
```

This model validation has improved till epoch 3 and after that it remained constant

Model_4:

this model has also not performed well just like the model 3

```
Epoch 1/20
600/600 [==============================] - ETA: 0s - loss: 10916.6025 - accuracy: 0.1256
Epoch 1: val_loss improved from inf to 2.19002, saving model to weights/Model_4.h5
600/600 [==============================] - 54s 88ms/step - loss: 10916.6025 - accuracy: 0.1256 - val_loss: 2.1900 - val_accuracy: 0.1250
Epoch 2/20
600/600 [==============================] - ETA: 0s - loss: 2.1906 - accuracy: 0.1302
```

```
Epoch 20: val_loss did not improve from 2.18961
600/600 [==============================] - 50s 84ms/step - loss: 2.1902 - accuracy: 0.1259 - val_loss: 2.1912 - val_accuracy: 0.1250
Valiation Loss 2.1896119117736816
 Valiation Accuracy 0.125
```

This model didn't performed well which kept its loss same throughout the epochs

Check points in the deep learning and machine are essentially the same thing, the way to save the present state of the experimentation of the model performance so that it can be picked up from the point where it was left.

Conclusion:

After comparison we can pick model 1 and model 2 for next phase, out of this two-model model-1 is picked for random sampling after looking into the training loss, validation loss, and accuracy.

********************************** End of Part A – 2 **

****

PART B:

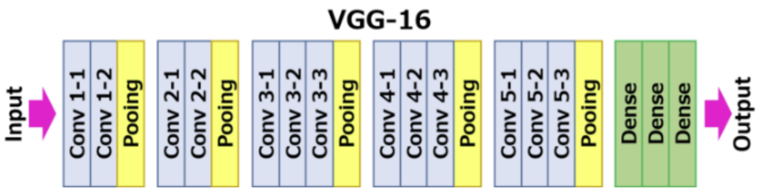
Transfer Learning

This is the era of the CNN which is in the peak in 20th century and it will raise in coming days, the main reason is increase in huge amount of data. from research it is said that in 2020, 90% of the data was unlabelled data and amongst that 40 to 50 percentage of data was in form of images, it may be from CCTV footages, images uploaded in social networking etc.,

Key features of VGG16:

1. Here number 16 refers to there are total of 16 layers which has some weights.
2. This always used 3x3 kernel for convolution and 2x2 for max pooling
3. VGG16 has about 138 parameters, this is trained on the ImageNet data
4. VGG16 has accuracy of 92.7%
5. This VGG has different versions like VGG19, with total of 19 layers with weights

Architecture of VGG16:



http://www.renom.jp/notebooks/tutorial/image_processing/neural-style-transfer/notebook.html

--Next Page--

Pre-trained VGG16 is used

```
[35] #
     VGG_16_model = VGG16(
         include_top=False,
         weights="imagenet",
         input_tensor=None,
         input_shape=(64,64,3),
         pooling=None,
         classes=9)

     VGG_16_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_10 (InputLayer)       [(None, 64, 64, 3)]       0

 block1_conv1 (Conv2D)       (None, 64, 64, 64)        1792

 block1_conv2 (Conv2D)       (None, 64, 64, 64)        36928

 block1_pool (MaxPooling2D)  (None, 32, 32, 64)        0

 block2_conv1 (Conv2D)       (None, 32, 32, 128)       73856

 block2_conv2 (Conv2D)       (None, 32, 32, 128)       147584

 block2_pool (MaxPooling2D)  (None, 16, 16, 128)       0

 block3_conv1 (Conv2D)       (None, 16, 16, 256)       295168

 block3_conv2 (Conv2D)       (None, 16, 16, 256)       590080

 block3_conv3 (Conv2D)       (None, 16, 16, 256)       590080

 block3_pool (MaxPooling2D)  (None, 8, 8, 256)         0

 block4_conv1 (Conv2D)       (None, 8, 8, 512)         1180160

 block4_conv2 (Conv2D)       (None, 8, 8, 512)         2359808

 block4_conv3 (Conv2D)       (None, 8, 8, 512)         2359808

 block4_pool (MaxPooling2D)  (None, 4, 4, 512)         0

 block5_conv1 (Conv2D)       (None, 4, 4, 512)         2359808

 block5_conv2 (Conv2D)       (None, 4, 4, 512)         2359808

 block5_conv3 (Conv2D)       (None, 4, 4, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 2, 2, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

This pretrained model takes input with shape of 64,64,3 following it has 5 blocks with 2 convolution layer in block 1, block 2 followed by 3 Convolution in block 3 with max pooling, and block 4 and block 5 has 3 convolution layer.

```
[36] featuresTrain= VGG_16_model.predict(trainX)
     featuresTrain= featuresTrain.reshape(featuresTrain.shape[0], -1)
     print("feature Training shape", featuresTrain.shape)

     featuresVal= VGG_16_model.predict(valX)
     featuresVal= featuresVal.reshape(featuresVal.shape[0], -1)
     print("features validation", featuresVal.shape)

     feature Training shape (19200, 2048)
     features validation (4800, 2048)
```

In the above code the feature is extracted using VGG16 and then they are feed into other machine learning models.

Model 1: Random Forest

```
Random_Forest Classifier

[37] randon_forest_model = RandomForestClassifier(300)
     randon_forest_model.fit(featuresTrain, trainY)

     RandomForestClassifier(n_estimators=300)

     results_randon_forest_model = randon_forest_model.predict(featuresVal)
     print (accuracy_score(results_randon_forest_model, valY))

     0.8189583333333333

printing the identified images correctly and confusion matrix to evaluate it

[39] print("Correctly identified images is ", accuracy_score(results_randon_forest_model, valY, normalize=False))
     print("\n", "Confusion matrix \n", confusion_matrix(valY, results_randon_forest_model, labels = range(0,9)))

     Correctly identified images is  3931

      Confusion matrix
      [[529   2   2  20   0  22  11   1  13]
      [  1 565   9   0   0  22   0   1   2]
      [  2  15 501  14   0  19  21  17  11]
      [ 38   0  12 318  19   4  36  12  61]
      [  8   0   2   1 456   0  15  15   3]
      [ 11  24  15   4   0 319  11   3  13]
      [ 19   1  42  24  26  12 345  15  16]
      [  0   0  15   3  12   1  10 558   1]
      [ 28   1  11  52   6  45  10   7 340]]
```

After feeding the extracted data into random forest, it gave the accuracy of 81% and this model identified 3931 images from the validation images set.

--next page--

Model 2: Logistic Regression

```
[40] logistic_regression_model = LogisticRegression()
     logistic_regression_model.fit(featuresTrain, trainY)

     LogisticRegression()

[43] results_logistic_regression_model = logistic_regression_model.predict(featuresVal)
     print (accuracy_score(results_logistic_regression_model, valY))

     0.8591666666666666

[44] print("Correctly identified images is ", accuracy_score(results_logistic_regression_model, valY, normalize=False))
     print("\n", "Confusion matrix \n", confusion_matrix(valY, results_logistic_regression_model, labels = range(0,9)))

     Correctly identified images is  4124

      Confusion matrix
      [[524   6   1  19   0  12  18   1  19]
       [  0 573   6   0   0  18   0   1   2]
       [  1  43 512  11   1   5  17   7   3]
       [ 14   1   9 394   2   6  20   5  49]
       [  4   0   1  12 445   0  21  13   4]
       [  7  42   4   3   0 323   7   0  14]
       [ 11   2  40  16   8   5 405   6   7]
       [  0   4   9   2   8   1  10 565   1]
       [ 18   3   8  47   3  29   9   0 383]]
```

This model gave the accuracy of 85% by identifying 4214 images

Model 3: Linear SVC Model

```
[45] linear_SVC_model = LinearSVC()
     linear_SVC_model.fit(featuresTrain, trainY)

     LinearSVC()

[46] results_linear_SVC_model = linear_SVC_model.predict(featuresVal)
     print (accuracy_score(results_linear_SVC_model, valY))

     0.856875

     print("Correctly identified images is ", accuracy_score(results_linear_SVC_model, valY, normalize=False))
     print("\n", "Confusion matrix \n", confusion_matrix(valY, results_linear_SVC_model, labels = range(0,9)))

     Correctly identified images is  4113

      Confusion matrix
      [[506   3   0  17   2  13  38   1  20]
       [  1 580   6   0   0   8   1   1   3]
       [  1   9 514  10   1   8  45   6   6]
       [ 14   0   7 363   4   3  48   8  53]
       [  3   1   1   7 447   0  21  19   1]
       [  8  16   4   4   0 339  16   0  13]
       [ 11   1  26  18  14   0 420   3   7]
       [  0   0   9   0   7   0  13 571   0]
       [ 13   7   8  55   2  19  20   3 373]]
```

This model gave accuracy of 85% by identifying 4113 images.

From above 3 models it can be concluded that Logistic regression performed better then other two

Implementation of other models for this extracted data

```
different_classifiers = {
    "Logistic_Regression" : LogisticRegression(solver = "lbfgs"),
    "KNearestNeighbours" : KNeighborsClassifier(n_neighbors=2),
    "Decision_Tress_Classifier" : DecisionTreeClassifier(min_samples_split=2),
    "Random_Forest_Classifier" : RandomForestClassifier(n_estimators=1000),
    "XG_Booster": XGBClassifier(),
    "Gradient_Boosted_Classifier":GradientBoostingClassifier()
}


for k, c in different_classifiers.items():
  c.fit(featuresTrain, trainY)

  y_prediction = c.predict(featuresVal)

  Training_score = cross_val_score(c, featuresTrain, trainY, cv=5)

  print("****************************************************************************************************************")
  print("Classifier = ", c.__class__.__name__, "Accuracy_score :", "-----")
  print("********************** Performance of the Test ***********************")
  print('Accuracy_Score = {:.3f} \n'.format(accuracy_score(valY, y_prediction) ))

  print("Total correctly identified images by the particular model", accuracy_score(y_prediction, valY, normalize=False),"--\n")
  print("Confusion matrix of model is : \n", confusion_matrix(valY, y_prediction, labels=range(0,9)))

  print("****************************************************************************************************************")
```

Implementation of other models:

Totally 6 classifiers were chosen to see which of this machine learning models will perform better for feature extracted data from VGG16

```
Classifier =  LogisticRegression Accuracy_score : -----
********************** Performance of the Test ***********************
Accuracy_Score = 0.859

Total correctly identified images by the particular model 4124 --

Confusion matrix of model is :
 [[524   6   1  19   0  12  18   1  19]
 [  0 573   6   0   0  18   0   1   2]
 [  1  43 512  11   1   5  17   7   3]
 [ 14   1   9 394   2   6  20   5  49]
 [  4   0   1  12 445   0  21  13   4]
 [  7  42   4   3   0 323   7   0  14]
 [ 11   2  40  16   8   5 405   6   7]
 [  0   4   9   2   8   1  10 565   1]
 [ 18   3   8  47   3  29   9   0 383]]
*********************************************************************
*********************************************************************
```

```
Classifier =  KNeighborsClassifier Accuracy_score : -----
********************** Performance of the Test **********************
Accuracy_Score = 0.728

Total correctly identified images by the particular model 3495 --

Confusion matrix of model is :
 [[529  15   5  21   0  16   9   0   5]
 [  4 584   4   1   0   6   0   1   0]
 [  0  66 482  14   1   6  26   5   0]
 [ 36   1  40 314  14  19  41   9  26]
 [  9   0   4  27 417   2  22  18   1]
 [ 17 108  44   7   0 217   6   0   1]
 [ 32   6  66  38  26  46 276   9   1]
 [  1   0  60   5  14  10  33 476   1]
 [ 29  18  33 115   8  71  23   3 200]]
*********************************************************************
*********************************************************************
```

```
Classifier =  DecisionTreeClassifier Accuracy_score : -----
********************** Performance of the Test ********************
Accuracy_Score = 0.635

Total correctly identified images by the particular model 3050 --

Confusion matrix of model is :
 [[423  10   7  40   4  22  37   3  54]
 [  6 523  26   1   0  34   1   2   7]
 [  6  32 383  32   9  30  59  33  16]
 [ 41   4  33 218  24  14  64  12  90]
 [  5   0   6  29 360   3  37  44  16]
 [ 21  39  29  14   1 231  15  10  40]
 [ 23   1  50  64  34  18 237  33  40]
 [  2   0  40  17  29   9  20 471  12]
 [ 55  13  19  83  17  45  47  17 204]]
*****************************************************************
*****************************************************************
```

```
Classifier =  RandomForestClassifier Accuracy_score : -----
********************** Performance of the Test ******************
Accuracy_Score = 0.823

Total correctly identified images by the particular model 3952 --

Confusion matrix of model is :
 [[535   2   1  16   0  21   9   1  15]
 [  0 562   9   0   0  25   0   1   3]
 [  3  17 502  13   0  17  20  17  11]
 [ 40   0  11 322  19   6  33  12  57]
 [  8   0   2   1 459   0  15  13   2]
 [  8  25  17   4   0 315  12   3  16]
 [ 20   1  39  24  26  11 348  13  18]
 [  0   0  15   2  11   1  11 559   1]
 [ 30   1   9  45   5  42  10   8 350]]
***************************************************************
***************************************************************
```

```
Classifier =  XGBClassifier Accuracy_score : -----
********************** Performance of the Test ********************
Accuracy_Score = 0.831

Total correctly identified images by the particular model 3989 --

Confusion matrix of model is :
 [[528   2   3  18   0  22  11   1  15]
 [  0 565   9   0   0  20   0   2   4]
 [  2  17 513  11   0  11  19  11  16]
 [ 29   0  15 337  13   3  32  10  61]
 [  7   0   2   7 442   0  17  20   5]
 [  7  22  16   2   0 325  12   1  15]
 [ 18   1  42  24  20  10 360  11  14]
 [  0   0  13   3  11   3  11 559   0]
 [ 25   2   9  54   4  33   9   4 360]]
*****************************************************************
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-53-39291be42f53> in <module>()
     14   y_prediction = c.predict(featuresVal)
     15
---> 16   Training_score = cross_val_score(c, featuresTrain, trainY, cv=5)
     17
     18   print("**************************************************************

──────────────────── 15 frames ────────────────────
/usr/local/lib/python3.7/dist-packages/sklearn/tree/_classes.py in fit(self, X, y, sample_weight, c
    418             )
    419
--> 420         builder.build(self.tree_, X, y, sample_weight)
    421
    422         if self.n_outputs_ == 1 and is_classifier(self):

KeyboardInterrupt:

SEARCH STACK OVERFLOW
```

Note: Gradiend Booster was taking more time and due to limitation of colab GPu the 6th model was interrupted and didn't complete the execution

Evaluation of other 5 machine learning models:

Model Name	Accuracy	Identified Images
Logistic Regression	85%	4214
K Neighbours classifier	72%	3495
Decision classifier	63%	3050
Random forest classifier	82%	3952
XG Classifier	83%	3989

From the above classifier logistic regress performed much better than any other model with 85% and by identifying 4214 images even it has outperformed the XGClassifier

So now we will evaluate out data with how much it will perform with VGG16 network by performing fine tuning it

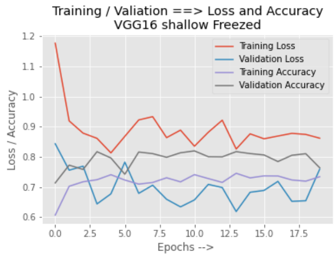
Freezing the layers by adding new fully connected layers:

Summary of new fully connected layers

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_10 (InputLayer)       [(None, 64, 64, 3)]       0

 block1_conv1 (Conv2D)       (None, 64, 64, 64)        1792

 block1_conv2 (Conv2D)       (None, 64, 64, 64)        36928

 block1_pool (MaxPooling2D)  (None, 32, 32, 64)        0

 block2_conv1 (Conv2D)       (None, 32, 32, 128)       73856

 block2_conv2 (Conv2D)       (None, 32, 32, 128)       147584

 block2_pool (MaxPooling2D)  (None, 16, 16, 128)       0

 block3_conv1 (Conv2D)       (None, 16, 16, 256)       295168

 block3_conv2 (Conv2D)       (None, 16, 16, 256)       590080

 block3_conv3 (Conv2D)       (None, 16, 16, 256)       590080

 block3_pool (MaxPooling2D)  (None, 8, 8, 256)         0

 block4_conv1 (Conv2D)       (None, 8, 8, 512)         1180160

 block4_conv2 (Conv2D)       (None, 8, 8, 512)         2359808

 block4_conv3 (Conv2D)       (None, 8, 8, 512)         2359808

 block4_pool (MaxPooling2D)  (None, 4, 4, 512)         0

 block5_conv1 (Conv2D)       (None, 4, 4, 512)         2359808

 block5_conv2 (Conv2D)       (None, 4, 4, 512)         2359808

 block5_conv3 (Conv2D)       (None, 4, 4, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 2, 2, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
```

First convolution 1 of Block 4 was blocked

```python
# Freezing block 4 conv 1 layer

value = False
VGG_16_model.trainable =not(value)
trainable = value
for layer in VGG_16_model.layers:
  if layer.name in ['block4_conv1']:
    trainable = not(value)
  # if trainable:
  #   layer.trainable = not(value)
  # else:
  #   layer.trainable = value
  layer.trainable = not(value) if trainable else layer.trainable = value


VGG_16_model.summary()
```



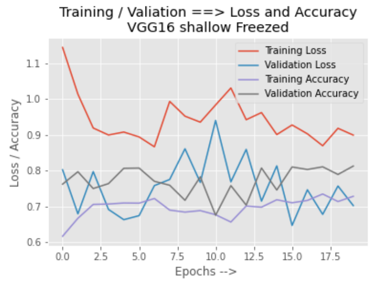

```
Epoch 19/20
600/600 [==============================] - 27s 46ms/step - loss: 0.8742 - accuracy: 0.7191 - val_loss: 0.6542 - val_accuracy: 0.8102
Epoch 20/20
600/600 [==============================] - 28s 46ms/step - loss: 0.8617 - accuracy: 0.7339 - val_loss: 0.7589 - val_accuracy: 0.7635
```

The training loss as drastically fallen after 1st epoch gradually increased after 3rd epoch where as the validation is showing completely variation in every epoch. So we cannot say that data is getting overfitted by looking at the accuracy we can observe that even it is showing some fluctuation, so this model can be train for more epochs to get the better result.

Second convolution 1 of block 5 was blocked

```python
# Freezing block 5 conv 1 layer

value = False
VGG_16_model.trainable =not(value)
trainable = value
for layer in VGG_16_model.layers:
  if layer.name in ['block5_conv1']:
    trainable = not(value)
  # if trainable:
  #   layer.trainable = not(value)
  # else:
  #   layer.trainable = value
  layer.trainable = not(value) if trainable else layer.trainable = value


VGG_16_model.summary()
```

```
Epoch 19/20
600/600 [==============================] - 28s 46ms/step - loss: 0.9184 - accuracy: 0.7134 - val_loss: 0.7568 - val_accuracy: 0.7892
Epoch 20/20
600/600 [==============================] - 28s 46ms/step - loss: 0.8994 - accuracy: 0.7280 - val_loss: 0.7019 - val_accuracy: 0.8127
```



After fine tuning this VGG16 by blocking the CNN for block 4 and block 5, the performance of this model is better in blocking 4 cnn1.

We cannot finally conclude that this model cannot perform better, by using more epochs and even by blocking it performance can be improved, as I have run this model only for 20 models, from the graph and observing the loss and accuracy values more epochs may improve it.

Conclusion:
From the above model 5 machine learning models and VGG16 (after tuning), the machine learning model performed better than the VGG16, with better performance by logistic Regression, this doesn't mean that VGG16 wont perform well, still many experiments could be done to improve it, due to time constraint this could not be achieved.

********************************* End of Part B ***************************************
****

PART C: Research

Self-supervised Learning: A category of machine learning that can learn from unlabelled data.

How do human beings learn anything so quickly?

The answer for the above question is, humans learn by adopting both unsupervised and supervised learning. We are multifaceted. Human can learn through supervised learn by being taught by our parents, teachers, friends also by doing some experiments and make conclusion of them.

On contrary to supervised learning we the humans also learn through unsupervised learning by looking or acquiring the minimum amount of data. when we look babies not all the things are taught to them are by parents but in fact they learn by observation in the surrounding environment and with little interaction in the initial stage.

This multifaceted approach of learning is easy for human, but this not the case for machine. Even though many robust Deep-Learning systems has been built which performed tasks of natural processing and image recognition, performing the complex task has remained challenging till now.

This is problem that **Self-Supervised learning** is trying to address. This is a form of learning which does not require the human intervention to label the data. The results which are generated by the models which will analyze the data, label and categorize the information without the need of human intervention. The difference with un-supervised learning and self-supervised learning is that this will not group and cluster the data as done in unsupervised learning.

This type of learning allows machines to examine a portion of a data example in order to determine the remainder. Self-supervised learning, in a nutshell, learns from un-label data to fill in the blanks for missing pieces. This information can take the form of text, audio, video or may be images.

In videos, for example, the machine can predict the missing portion of a video given only a video section. Missing frames in a video can also be predicted using videos.

Self-supervised learning aims to improve the data efficiency of deep learning models. This means that it helps to reduce the over-reliance on massive amounts of data in order to produce good models.

Applications:

1. **Computer Vision:**

   SimCLR is a framework that uses self-supervised learning to learn visual representations in images. The framework is responsible for two major tasks: a pretext task and a downstream (real) task. In the pretext task, self-supervised learning is used.
   It entails performing simple augmentation tasks on input images, such as random cropping, random color distortions, and random Gaussian blur. This process allows the model to learn more accurate representations of the input images. These results are fed into the downstream task module, which handles the main tasks like detection and classification.

2. **Natural Language Processing:**
   Self-supervised learning aids in predicting missing words in a text. This is accomplished by presenting text segments to a massive neural network with billions of parameters, such as OpenAI's GPT-3 or Google's BERT. Where in 15% of the text is masked to force the network to predict the missing word fragments.

Summary:

- Self-supervised learning uses unlabeled data to generate labels. This eliminates the need for data labeling by hand, which is a time-consuming process.
- Creating supervised tasks like pretext tasks that teach meaningful representation in order to perform downstream tasks like detection and classification.
- This type of learning assists in filling in the blanks. In NLP, for example, they can help predict missing words.

Conclusion:

Self-supervised learning has aided in the development of AI systems that can learn with fewer samples or trials. This has been demonstrated in the field of Natural Language Processing (NLP) with GPT-3 and BERT, which can learn from very few examples.

This network seeks good representations from unlabeled data rather than learning from labeled data. This reduces the need to rely on massive amounts of data, as in supervised learning.

Self-supervised learning is still in its early stages at the moment. Machines cannot yet learn or understand everything that humans can, but it appears to be an exciting and promising step forward.

Reference for this section:
https://www.section.io/engineering-education/what-is-self-supervised-learning/

**************** END *************

Note: all the references for images and the content are mentioned at respective places and for coding it is mentioned at the end of the coding