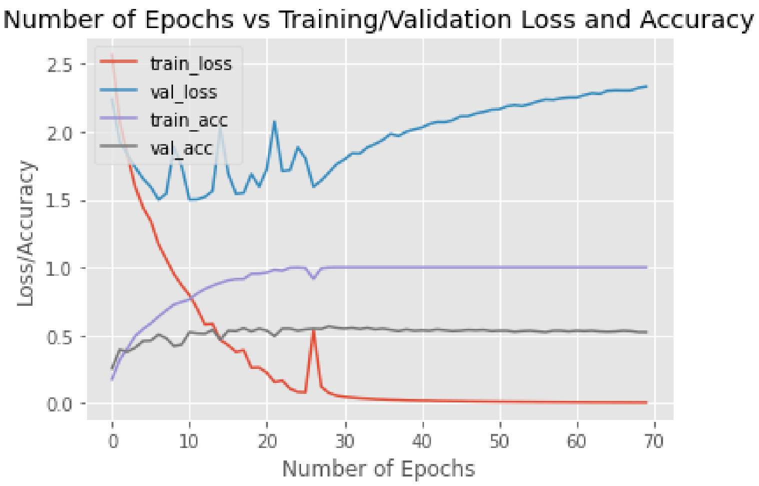
PART A: Convolutional Neural Networks

Part A – Question 1 – Development and Analysis of 4 different CNN Architectures.

Architecture 1- Evaluation of Baseline CNN Model

We trained our model for 70 epochs on the **Flower-17 dataset** with the below architecture

```
Compiling model...
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 128, 128, 32)      896
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 64, 64, 32)        0
_________________________________________________________________
flatten_8 (Flatten)          (None, 131072)            0
_________________________________________________________________
dense_13 (Dense)             (None, 17)                2228241
=================================================================
Total params: 2,229,137
Trainable params: 2,229,137
Non-trainable params: 0
```




```
11/11 [==============================] - 0s 8ms/step - loss: 2.3324 - accuracy: 0.5235
The test set loss and accuracy is
 [2.3323564529418945, 0.5235294103622437]
```

Observation of the above graph

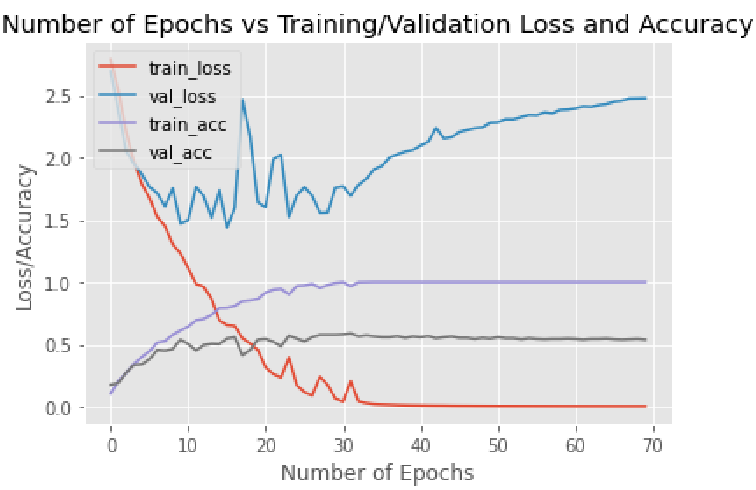
- Clearly our basic CNN model is not performing well on the test set achieving the very less test set accuracy of 52.35 %.
- Training accuracy begins to flatten out at approx. 100% from the epochs 30 and there is also very small increase in the validation accuracy from epoch 23 averaging around 53%.

- We can clearly see that the network begins to overfit aggressively on the training data from the epoch around 8 because of the increasing divergence/gap between the training loss and the validation loss at a much greater rate leading to a wider and wider gap between the two curves.
- So, we can estimate our right/optimal number of epochs to be around 8 with the basic CNN model on this flower-17 dataset.

Architecture 2- Evaluation

We trained our model for 70 epochs on the **Flower-17 dataset** with the below architecture.

```
Compiling model...
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_21 (Conv2D)           (None, 128, 128, 32)      896
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 64, 64, 64)        18496
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 32, 32, 64)        0
_________________________________________________________________
flatten_11 (Flatten)         (None, 65536)             0
_________________________________________________________________
dense_18 (Dense)             (None, 200)               13107400
_________________________________________________________________
dense_19 (Dense)             (None, 17)                3417
=================================================================
Total params: 13,130,209
Trainable params: 13,130,209
Non-trainable params: 0
_________________________________________________________________
```




```
11/11 [==============================] - 0s 12ms/step - loss: 2.4763 - accuracy: 0.5382
The test set loss and accuracy is
 [2.476296901702881, 0.5382353067398071]
```

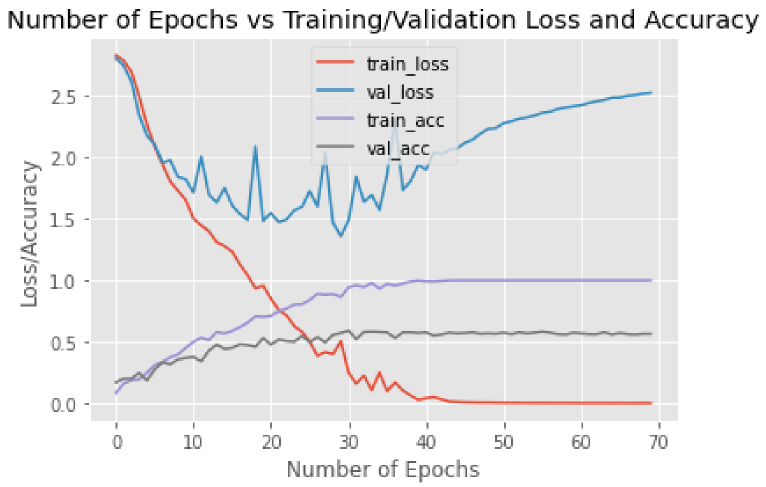
Observation of the above graph

- Clearly our CNN architecture 2 even with increased layers is not performing well on the test set achieving the very less test set accuracy of 53.82 %.
- Training accuracy begins to flatten out at approx. 100% from the epochs 35 and there is also very small increase in the validation accuracy from epoch 40 averaging around 54%.
- We can clearly see that the network begins to overfit aggressively on the training data from the epoch around 10 because of the increasing divergence/gap between the training loss and the validation loss at a much greater rate leading to a wider and wider gap between the two curves.
- So, we can estimate our right/optimal number of epochs to be around 10 with the CNN architecture 2 on this flower-17 dataset.

Architecture 3- Evaluation

We trained our model for 70 epochs on the **Flower-17 dataset** with the below architecture.

```
Compiling model...
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_26 (Conv2D)           (None, 128, 128, 32)      896
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 64, 64, 64)        18496
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 32, 32, 128)       73856
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 16, 16, 128)       0
_________________________________________________________________
flatten_13 (Flatten)         (None, 32768)             0
_________________________________________________________________
dense_23 (Dense)             (None, 400)               13107600
_________________________________________________________________
dense_24 (Dense)             (None, 200)               80200
_________________________________________________________________
dense_25 (Dense)             (None, 17)                3417
=================================================================
Total params: 13,284,465
Trainable params: 13,284,465
Non-trainable params: 0
```




```
11/11 [==============================] - 0s 14ms/step - loss: 2.5232 - accuracy: 0.5647
The test set loss and accuracy is
 [2.523185968399048, 0.5647059082984924]
```

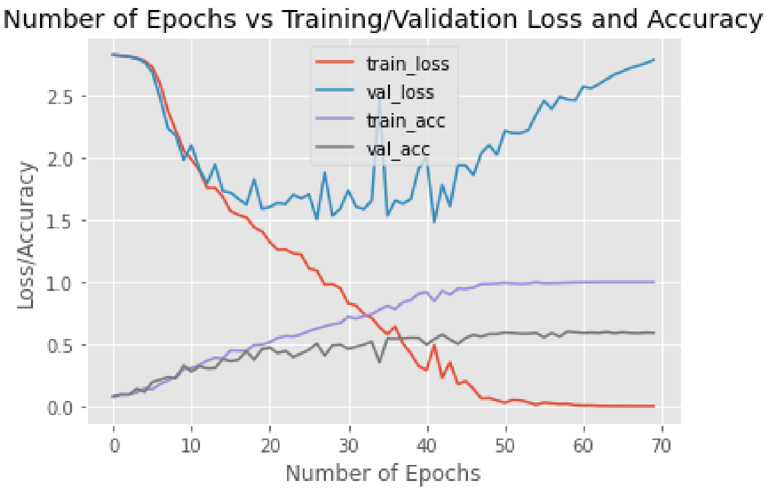
Observation of the above graph

- Clearly our CNN architecture 3 even with increased layers is not performing well on the test set achieving the very less test set accuracy of 56.47 %.
- Training accuracy begins to flatten out at approx. 100% from the epochs 45 and there is also very small increase in the validation accuracy from epoch 40 averaging around 56%.
- We can clearly see that the network begins to overfit aggressively on the training data from the epoch around 12 because of the increasing divergence/gap between the training loss and the validation loss at a much greater rate leading to a wider and wider gap between the two curves.
- So, we can estimate our right/optimal number of epochs to be around 12 with the CNN architecture 3 on this flower-17 dataset.

Architecture 4- Evaluation

We trained our model for 70 epochs on the **Flower-17 dataset** with the below architecture.

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_29 (Conv2D)           (None, 128, 128, 32)      896
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 64, 64, 64)        18496
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 32, 32, 128)       73856
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 16, 16, 256)       295168
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 8, 8, 256)         0
_________________________________________________________________
flatten_14 (Flatten)         (None, 16384)             0
_________________________________________________________________
dense_26 (Dense)             (None, 600)               9831000
_________________________________________________________________
dense_27 (Dense)             (None, 400)               240400
_________________________________________________________________
dense_28 (Dense)             (None, 200)               80200
_________________________________________________________________
dense_29 (Dense)             (None, 17)                3417
=================================================================
Total params: 10,543,433
Trainable params: 10,543,433
Non-trainable params: 0
```




```
11/11 [==============================] - 0s 16ms/step - loss: 2.7910 - accuracy: 0.5912
The test set loss and accuracy is
 [2.791020154953003, 0.591176450252533]
```

Observation of the above graph

- Clearly our CNN architecture 4 even with increased layers is not performing well on the test set achieving the very less test set accuracy of 59.11 %.
- Training accuracy begins to flatten out at approx. 100% from the epochs 55 and there is also very small increase in the validation accuracy from epoch 50 averaging around 59%.
- We can clearly see that the network begins to overfit aggressively on the training data from the epoch around 20 because of the increasing divergence/gap between the training loss and the validation loss at a much greater rate leading to a wider and wider gap between the two curves.
- So, we can estimate our right/optimal number of epochs to be around 20 with the CNN architecture 4 on this flower-17 dataset.

Comparative Analysis of CNN Architecture Baseline, 2, 3, 4

Now we will be performing the comparative analysis of the Baseline CNN with Architecture 2, 3 and 4 as mentioned above. This will lead to the examination of the deeper convolutional neural network to our Flower-17 classification problem.

CNN Architecture	**Test Set Accuracy**	**Overfitting Levels Start**
1) **Baseline CNN** (1 Conv layer + 1 Pooling layer) , Total layers= 4	52.35%	From Epoch=8
2) **Architecture 2 CNN** (2 Conv layers + 2 Pooling Layers) , Total layers= 7	53.82%	From Epoch=10
3) **Architecture 3 CNN** (3 Conv layers + 3 Pooling Layers), Total layers= 10	56.47%	From Epoch=12
4) **Architecture 4 CNN** (4 Conv layers + 4 Pooling Layers), Total layers= 13	59.11%	From Epoch=20

- It is clearly evident that as we increased the number of layers in our CNN, our test set accuracy gradually increased attaining 59.11% for our architecture 4.
- In all of the above architectures, there was the observed aggressive overfitting on the training data. The architecture with less layers started to overfit as early when compared with the deeper versions as evident from the table.
- **Additional Experiment** – We moreover introduced 5th set of CONV and POOL layers in our architecture to investigate further; the impact of increasing layers. But this reduced our test set accuracy to 55.29%. (PLEASE see the PartA Jupyter notebook for the detailed insights for this architecture 5.)
- Therefore we can come up with the best CNN model for our flower classification problem that is CNN Architecture 4 with 13 layers (4 sets of CONV and POOL layers).

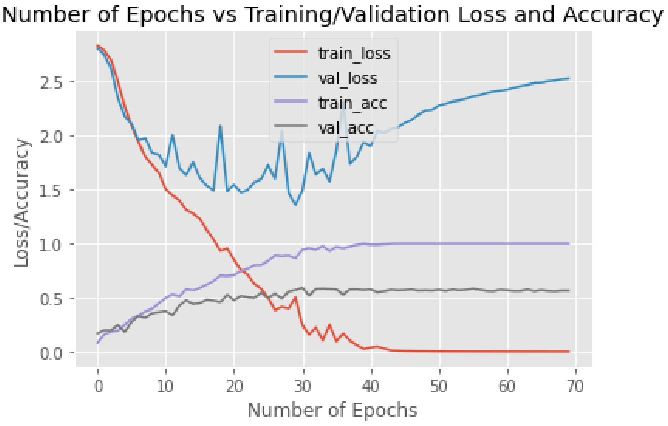
Part A – Question 1.b – Application and Analysis of Data Augmentation to Architecture 3 and Architecture 4.

- We would be choosing architecture 3 and 4 for the application of data augmentation techniques as these architectures were the deepest ones.
- We would be applying the below data augmentation configurations to each of these deep CNN architectures.

```
# --------------------- DATA AUGMENTATION CONFIGURATION 1-----------------------------------

train_data_generator1 = tf.keras.preprocessing.image.ImageDataGenerator(
    zoom_range=-0.2,
    shear_range=0.2,
    vertical_flip=False,
    rotation_range=30,
    horizontal_flip=True)

# --------------------- DATA AUGMENTATION CONFIGURATION 2-----------------------------------

train_data_generator2 = tf.keras.preprocessing.image.ImageDataGenerator(
    zoom_range=-0.2,
    shear_range=0.2,
    vertical_flip=True,
    rotation_range=30,
    horizontal_flip=False,
    height_shift_range=0.1,
    width_shift_range=0.1)
```

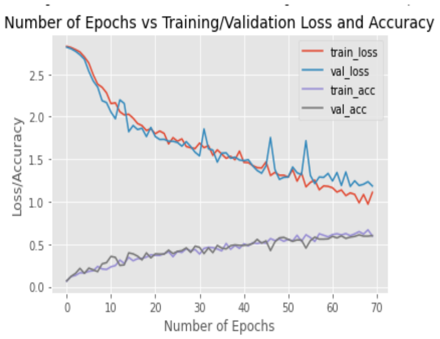
Comparative Analysis of CNN Architecture 3 with and without Data Augmentation

Comparison Metric	Architecture 3 without Data Augmentation	Architecture 3 with Data Augmentation configuration 1	Architecture 3 with Data Augmentation configuration 2
Test Accuracy	56.47%	60.0%	56.17%
Overfitting Levels	From Epoch 12	No Overfitting	No Overfitting



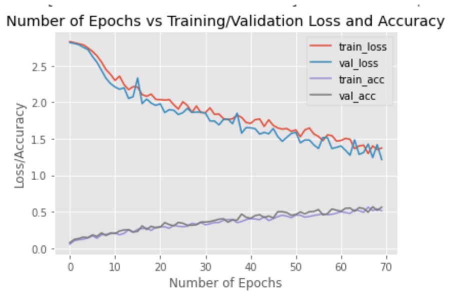

```
11/11 [==============================] - 0s 14ms/step - loss: 2.5232 - accuracy: 0.5647
The test set loss and accuracy is
 [2.523185968399048, 0.5647059082984924]
```

CNN Architecture 3 without Data Augmentation




```
11/11 [==============================] - 0s 10ms/step - loss: 1.1842 - accuracy: 0.6000
The test set loss and accuracy is
 [1.184156894683838, 0.6000000238418579]
```

CNN Architecture 3 with Data Augmentation Configuration 1



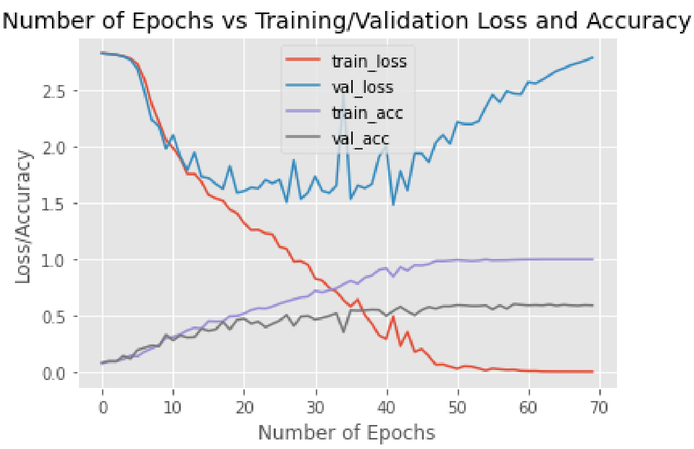

```
11/11 [==============================] - 0s 9ms/step - loss: 1.2161 - accuracy: 0.5618
The test set loss and accuracy is
 [1.2160934209823608, 0.5617647171020508]
```

CNN Architecture 3 with Data Augmentation Configuration 2

When we applied the Data Augmentation configurations 1 and 2 on CNN architecture 3, there was no significant improvement in the test set accuracy, but both configurations almost reduced the overfitting to zero in the architecture.

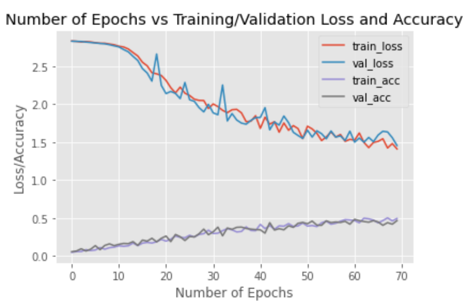
Comparative Analysis of CNN Architecture 4 with and without Data Augmentation

Comparison Metric	Architecture 4 without Data Augmentation	Architecture 4 with Data Augmentation configuration 1	Architecture 4 with Data Augmentation configuration 2
Test Accuracy	59.11%	46.76%	40.00%
Overfitting Levels	From Epoch 20	No Overfitting	No Overfitting

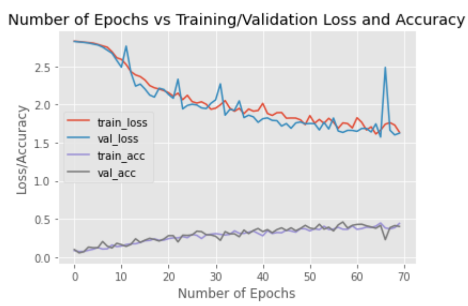



```
11/11 [==============================] - 0s 16ms/step - loss: 2.7910 - accuracy: 0.5912
The test set loss and accuracy is
 [2.791020154953003, 0.591176450252533]
```

CNN Architecture 4 without Data Augmentation




```
11/11 [==============================] - 0s 13ms/step - loss: 1.4570 - accuracy: 0.4676
The test set loss and accuracy is
 [1.457032561302185, 0.46764704585075378]
```




```
11/11 [==============================] - 0s 11ms/step - loss: 1.6251 - accuracy: 0.4000
The test set loss and accuracy is
 [1.6250592470169067, 0.4000000059604645]
```

CNN Architecture 4 with Data Augmentation Configuration 1

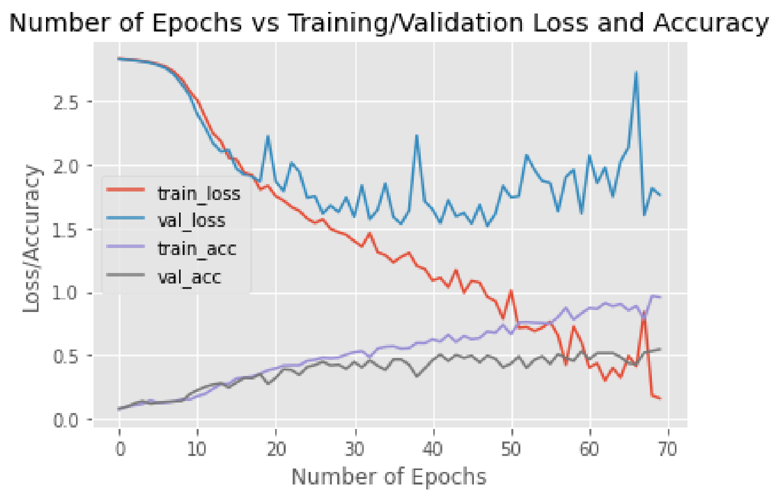
CNN Architecture 4 with Data Augmentation Configuration 2

When we applied the Data Augmentation configurations 1 and 2 on CNN architecture 4, test set accuracy decreased by more than 13%, but both configurations almost reduced the overfitting to zero in the architecture.

Part A – Question 2 – Application and Analysis of Ensemble technique on Flower-17 dataset.

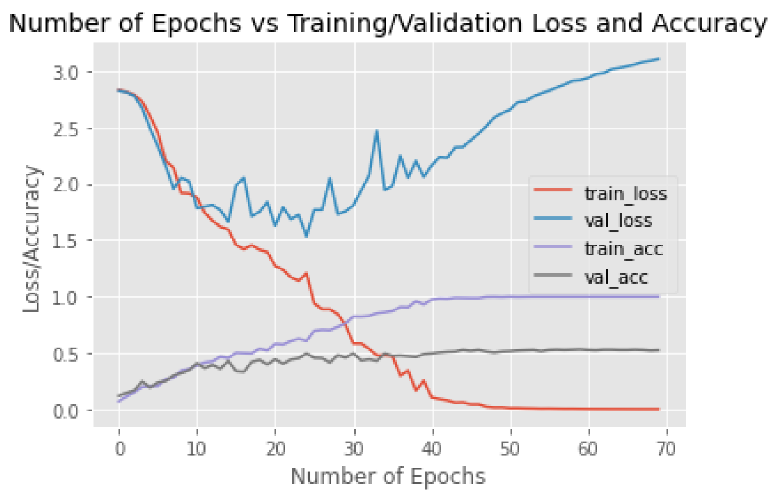
- We created the 4 CNN architectures for developing our Ensemble model.
- Below is the evaluation graph of each of the individual base learners.

Evaluation graph of Base Learner 1



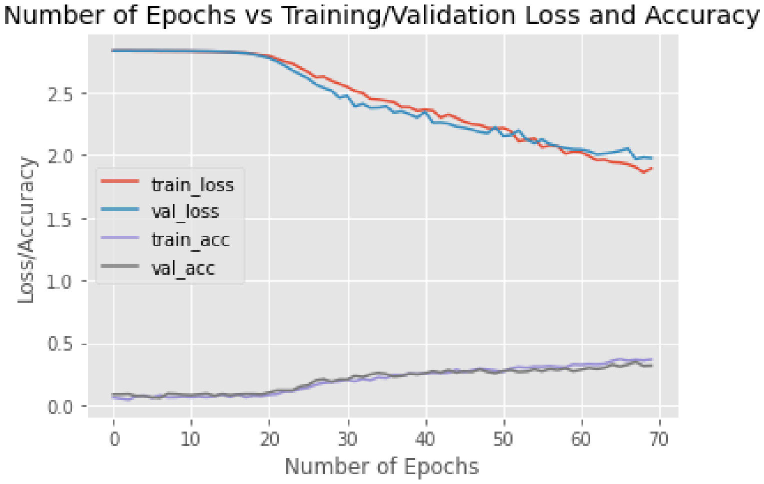

```
11/11 [==============================] - 0s 11ms/step - loss: 1.7598 - accuracy: 0.5471
The test set loss and accuracy is
 [1.7598086595535278, 0.5470588207244873]
```

Evaluation graph of Base Learner 2



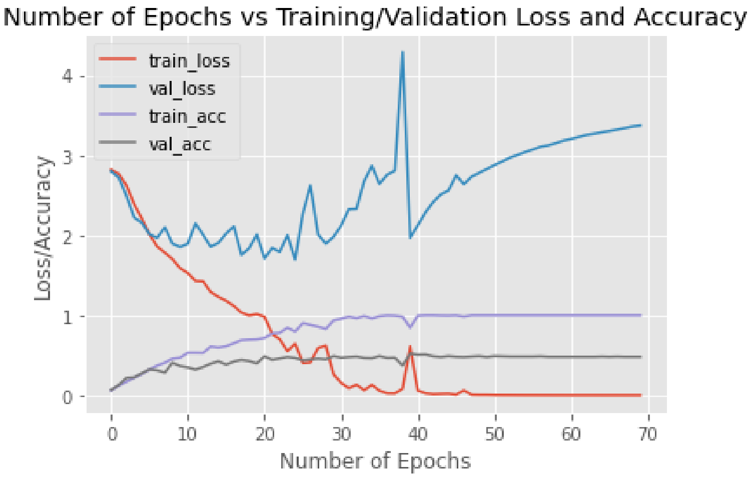

```
11/11 [==============================] - 0s 12ms/step - loss: 3.1033 - accuracy: 0.5265
The test set loss and accuracy is
 [3.103313446044922, 0.5264706015586853]
```

Evaluation graph of Base Learner 3




```
11/11 [==============================] - 0s 20ms/step - loss: 1.9758 - accuracy: 0.3176
The test set loss and accuracy is
 [1.9758161306381226, 0.3176470696926117]
```

Evaluation graph of Base Learner 4




```
11/11 [==============================] - 0s 16ms/step - loss: 3.3709 - accuracy: 0.4824
The test set loss and accuracy is
 [3.370889902114868, 0.48235294222831726]
```

Final Result with Ensemble

```
The accuracy with ensemble of 4 CNN models on Flower-17 dataset is 0.5323529243469238
```

Comparative Analysis of Individual Models with Ensembles

CNN Model	Test Set Accuracy
1. Model 1	54.70%
2. Model 2	52.64%
3. Model 3	31.76%
4. Model 4	48.23%
5. Ensemble Model	53.23%

PART B: Transfer Learning

Part B – Question 1 – Comparative Analysis of Feature Extraction from various pre-trained models on multiple Machine learning models.

- We extracted the discriminative features from various pre-trained CNN models like VGG16, VGG19 and InceptionV3.
- Then we created the new feature train and test datasets that can be fed into numerous Machine learning algorithms.
- Further we trained our Machine learning models (Random Forest, Logistic Regression, Decision Tree and SVM) with these powerful and discriminative features that are obtained from these pre-trained CNN models.
- Then we evaluated the performance of each of these pre-trained models on the various Machine learning algorithms.
- Below is the comparative analysis of our obtained results for each of these pre-trained CNN models with 4 Machine learning algorithms.

Pre-trained Model	Machine learning classifier	Test Set Accuracy
1. VGG16	Random Forest	61.17%
	Logistic Regression	87.94%
	Decision Tree	46.47%
	Support Vector Machine	84.41%
2. VGG19	Random Forest	65.29%
	Logistic Regression	86.76%
	Decision Tree	44.70%
	Support Vector Machine	81.17%
3. InceptionV3	Random Forest	47.35%
	Logistic Regression	84.70%
	Decision Tree	45.58%
	Support Vector Machine	84.11%

Observation of the above table

- Out of VGG16, VGG19 and InceptionV3, the **VGG16** features performed the best with **logistic regression** achieving the test set accuracy of **87.94%** on the Flower-17 dataset.
- In all the pre-trained model features, the worst performance was of **Decision Tree classifier** with VGG19 with only **44.70%** test set accuracy.

Comparative Analysis of Feature Extraction from Variants of various pre-trained models on multiple Machine learning models.

- We created the variants of VGG16 and VGG19 such that we removed all the layers from `block4_conv2` in VGG16 and `block4_conv4` in VGG19 respectively.
- We then evaluated the performance of these variants of pre-trained model with the above Machine learning models.
- Below is the comparative analysis of the results obtained.

Pre-trained Model Variant	Machine Learning Model	Test Set Accuracy
1. VGG16_Variant	Random Forest	48.52%
	Logistic Regression	82.94%
	Decision Tree	39.41%
	SVM	78.82%
2. VGG19_Variant	Random Forest	42.35%
	Logistic Regression	79.41%
	Decision Tree	38.23%
	SVM	71.17%

Observation of the above table

- Overall the VGG16_variant is performing better than VGG19_variant for all the machine learning models.
- The highest performance was achieved by the **VGG16_variant** with **logistic regression** achieving the **82.94%** on the test set.
- The initial VGG16 and VGG19 features (only fully connected layers were removed) performed better that their variants where we removed the layers from the architectures from a certain exit point.

Part B – Question 2 – Application of Fine Tuning Transfer learning on Flower-17 dataset

- Here our main goal is to come up with the best validation accuracy on the flower-17 dataset with the help of fine tuning on the pre-trained models.
- From our above analysis from feature extraction, we found that VGG16 is performing better than the other 2 pre-trained models (these are the results obtained without fine tuning the model and using the pre-trained model weights as it is for our own dataset). So we would be choosing the VGG16 model for fine tuning with our own flower-17 dataset.

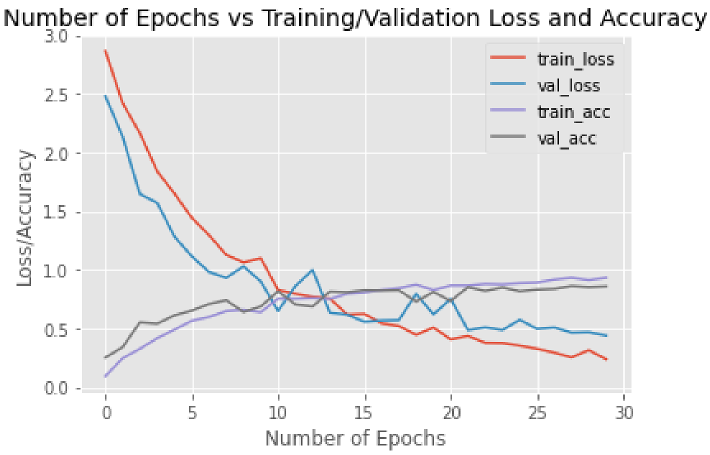
Main OBJECTIVE/GOALS of this fine tuning part in Transfer Learning

1) The best test set accuracy from feature extraction (this is without fine tuning) is reported by VGG16 with logistic regression as **87.94%.** Our main aim is to beat this accuracy with the application of fine tuning and its variations.
2) First we will apply the Phase 1 of fine tuning and evaluate the results.
3) Then we will apply atleast 3 architectural configurations in Phase 2 of fine tuning.
4) Finally we will apply the data augmentation techniques on the best obtained settings/model from phase 2 of fine tuning to check the impact on the performance.

Evaluation Graph of Phase 1 of Fine Tuning

- In phase 1, we removed the fully connected layers of VGG16 model and added our own fully connected layers where weights are initialized randomly.
- Then we freeze all the trainable parameters on all the layers except the FC layers such that weights of FC layers are only updated and not of other layers.
- Then we allow the training process which will update the weights in our created fully connected network. Below is the model we created in phase 1.

```
Model: "sequential_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_19 (Flatten)         (None, 8192)              0
_________________________________________________________________
dropout_19 (Dropout)         (None, 8192)              0
_________________________________________________________________
dense_53 (Dense)             (None, 500)               4096500
_________________________________________________________________
dense_54 (Dense)             (None, 150)               75150
_________________________________________________________________
dense_55 (Dense)             (None, 17)                2567
=================================================================
Total params: 18,888,905
Trainable params: 4,174,217
Non-trainable params: 14,714,688
_________________________________________________________________
```

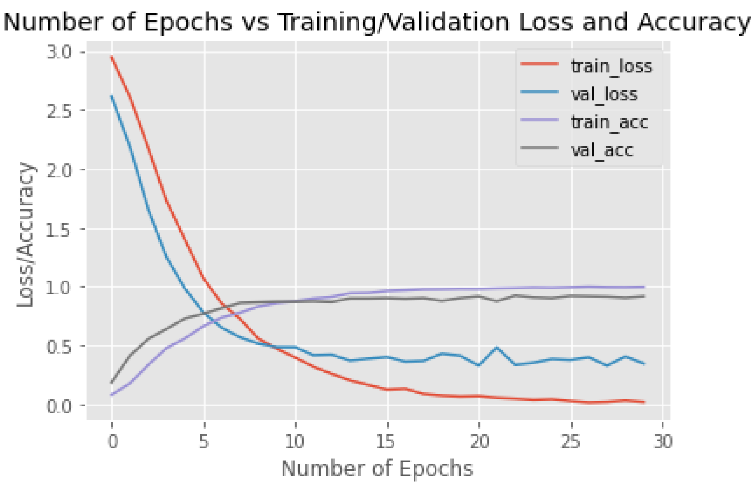



```
11/11 [==============================] - 1s 58ms/step - loss: 0.4406 - accuracy: 0.8588
The test set loss and accuracy is
 [0.44062885642051697, 0.8588235378265381]
```

Results of Phase 2 Fine Tuning with Different configuration settings

Configuration Setting 1 – Unfreezed the weights from block4_conv1

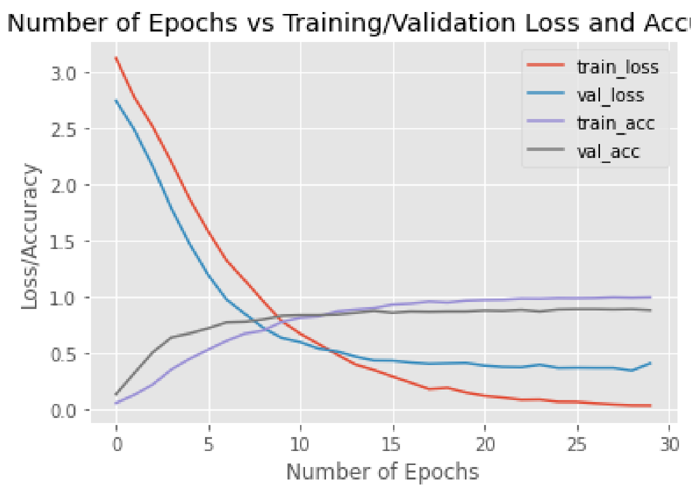
```
Compiling model...
Model: "sequential_22"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_21 (Flatten)         (None, 8192)              0
_________________________________________________________________
dropout_21 (Dropout)         (None, 8192)              0
_________________________________________________________________
dense_59 (Dense)             (None, 500)               4096500
_________________________________________________________________
dense_60 (Dense)             (None, 150)               75150
_________________________________________________________________
dense_61 (Dense)             (None, 17)                2567
=================================================================
Total params: 18,888,905
Trainable params: 17,153,417
Non-trainable params: 1,735,488
```




```
11/11 [==============================] - 1s 58ms/step - loss: 0.3448 - accuracy: 0.9147
The test set loss and accuracy is
 [0.3447767496109009, 0.9147058725357056]
```

Configuration Setting 2 - Unfreezed the weights from block5_conv1

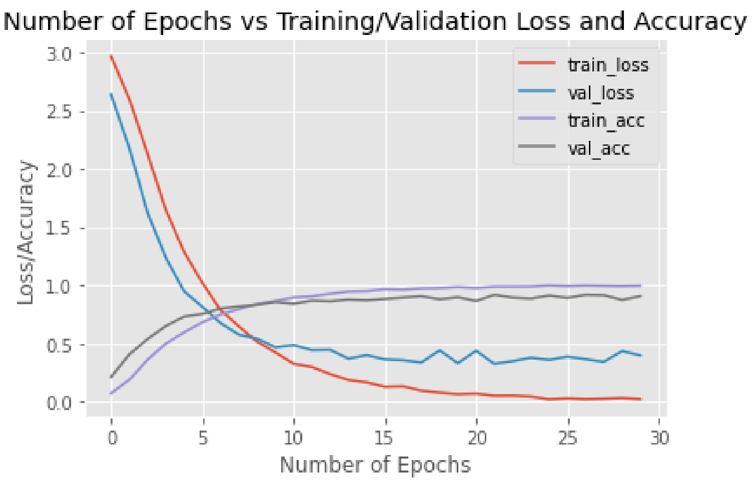
```
Model: "sequential_26"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_25 (Flatten)         (None, 8192)              0
_________________________________________________________________
dropout_25 (Dropout)         (None, 8192)              0
_________________________________________________________________
dense_71 (Dense)             (None, 500)               4096500
_________________________________________________________________
dense_72 (Dense)             (None, 150)               75150
_________________________________________________________________
dense_73 (Dense)             (None, 17)                2567
=================================================================
Total params: 18,888,905
Trainable params: 11,253,641
Non-trainable params: 7,635,264
_________________________________________________________________
```




```
11/11 [==============================] - 1s 58ms/step - loss: 0.4074 - accuracy: 0.8794
The test set loss and accuracy is
 [0.40743955969810486, 0.8794117569923401]
```

Configuration Setting 3 - Unfreezed the weights from block3_conv1

```
Compiling model...
Model: "sequential_27"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688  
_________________________________________________________________
flatten_26 (Flatten)         (None, 8192)              0         
_________________________________________________________________
dropout_26 (Dropout)         (None, 8192)              0         
_________________________________________________________________
dense_74 (Dense)             (None, 500)               4096500   
_________________________________________________________________
dense_75 (Dense)             (None, 150)               75150     
_________________________________________________________________
dense_76 (Dense)             (None, 17)                2567      
=================================================================
Total params: 18,888,905
Trainable params: 18,628,745
Non-trainable params: 260,160
_________________________________________________________________
```

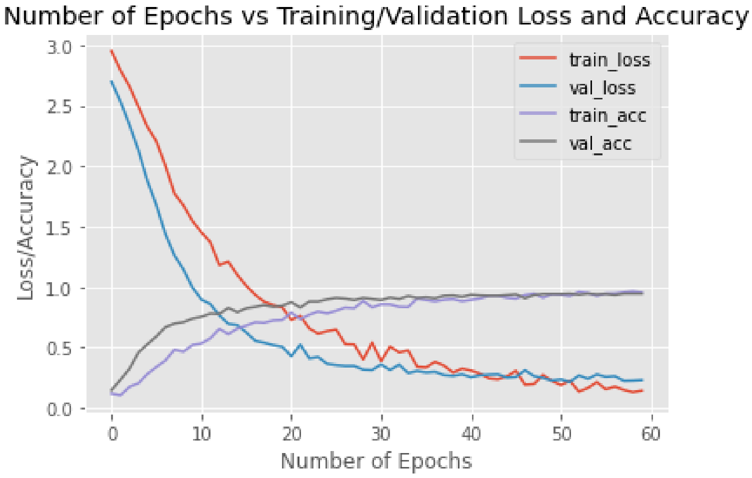



```
11/11 [==============================] - 1s 59ms/step - loss: 0.3968 - accuracy: 0.9059
The test set loss and accuracy is
 [0.3967837691307068, 0.9058823585510254]
```

Application of Data Augmentation + Fine Tuning

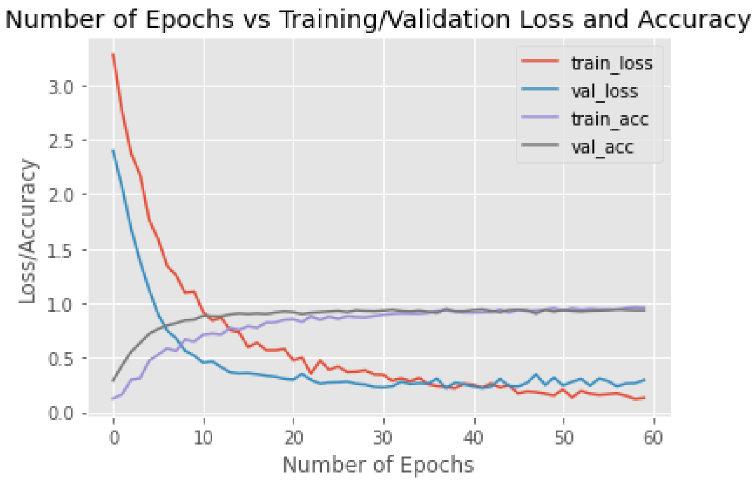
It came out that the best accuracy came out to be 91.47% with configuration setting 1, so we will apply the data augmentation techniques to this architecture with fine tuning.

We applied the 2 data augmentation settings to the best model obtained in phase 2 and the evaluation graph of each of the settings are as follows.




```
11/11 [==============================] - 0s 26ms/step - loss: 0.2295 - accuracy: 0.9441
The test set loss and accuracy is
 [0.22952333092689514, 0.9441176652908325]
```

Data Augmentation Setting 1 to the best model




```
11/11 [==============================] - 0s 26ms/step - loss: 0.2951 - accuracy: 0.9353
The test set loss and accuracy is
 [0.2950620651245117, 0.9352940917015076]
```

Data Augmentation Setting 2 to the best model

Comparative Analysis of Fine Tuning on Flower-17 dataset with VGG16 model

Different Configurations Settings	Test Set Accuracy	Overfitting Levels
1. Feature Extraction with Logistic Regression	87.94%	-
2. Unfreezed FC network (phase1)	85.88%	Not much overfitting
3. Unfreeze from block4_conv1 (phase2 variant)	91.47%	From Epoch 10 but not much aggressively
4. Unfreezed from block5_conv1 (phase2 variant)	87.94%	From Epoch 15 but not much aggressively
5. Unfreezed from block3_conv1 (phase2 variant)	90.58%	From Epoch 8 but not much aggressively
6. Data Augmentation setting on block4_conv1 model	**94.41%**	Overfitting dropped to almost 0.
7. Data Augmentation setting on block4_conv1 model	93.52%	Overfitting dropped to almost 0.

Best Performance Achieved on Flower-17 dataset

Observation of the above table

- We applied our fine tuning on VGG16 model in two phases. In the first phase we got the accuracy of 85.88%.
- Then we applied the 3 variants of phase 2 in which the variant 1 performed the best achieving the test set accuracy of 91.47%.
- Further to escalate the performance we experimented with the 2 data augmentation settings out of which the configuration setting 1 achieved the highest accuracy of **94.41%** on this flower-17 dataset which exceeded the performance without fine tuning that was **87.94%** with feature extraction on logistic regression.
- It was also observed that the level of overfitting almost dropped to zero with the application of data augmentation settings when compared with the fine tuning without these augmentation settings.

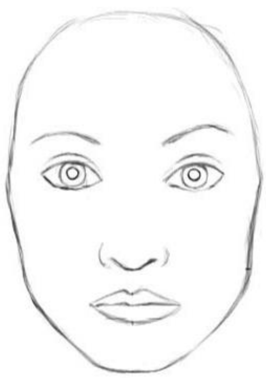
Part C: Research [20 Marks]

What are Capsule Networks?

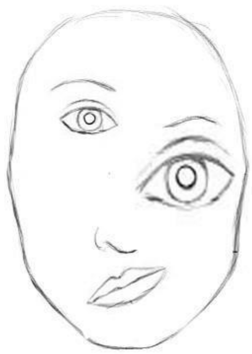
- A Capsule Neural Network often called as CapsNet is the type of one of the recent Neural Network architecture that attempts to improve the modeling of the hierarchical relationships within the Convolutional Neural Network by adding the additional components within the conventional architecture.
- The main idea behind CapsNet is to add the structures called “capsules” to a CNN. Local capsules within the network perform the internal complicated computations on their inputs and encapsulate the results into a small vector containing highly informative outputs which are probability of an observation and pose for that observation.

Description of the problem in CNN that Capsule Network Addresses

- **Working of CNN:** The main components of a CNN are the convolutional layers which are responsible for detecting the features from the pixel distribution of an image. The layers that are close to input detect the simple features such as edges and color gradients and the deep layers in the network detect the complex features such as eyes or nose by combining the simple ones from the previous layers. Finally, the dense layers combines all the features that network has learnt to make the final predictions.
- **Moving towards the actual problem:** So, everything seems fine and perfect up till here as the performance of CNN is really strong in image classification tasks. But here we will understand the big shortcoming of CNN by a small example. Consider a human face that consists of features such as oval structure, two eyes, a nose and mouth. Finally the last/deep layers in a CNN combine all these features and only the presence of these features irrespective of their location in the image, CNN will consider this as a face but it might look something as below which is not a human looking face.



Actual Human Face

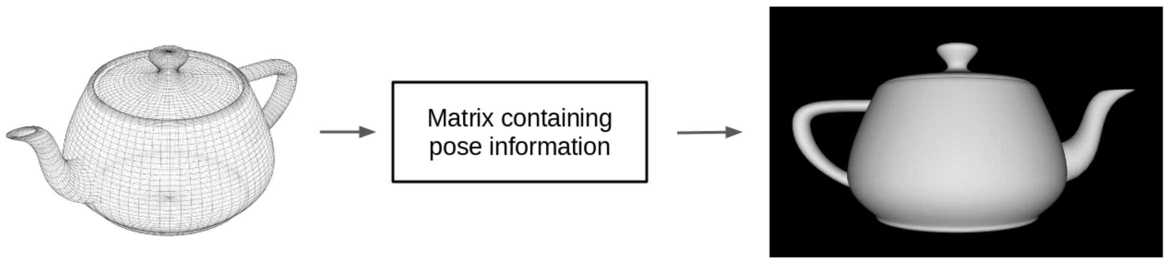


Disordered Structure containing the components of face which is a face to CNN.

- **CNN's are translation invariant:** As evident from the above picture that CNN can classify the disordered structure containing the components of a face as an actual human face. For a CNN, the Orientational and relative Spatial relationships between these components are not worth considering which can result in such misclassification. For the correct classification it is very much important to know how these objects are positioned relative to each other which CapsNet tend to preserve.
- *The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster (Geoffrey Hinton).*
- **Loose of Valuable Information in CNN:** The pooling operation in a CNN only allows the most active neurons to propagate through the deeper layers within the network. These pooling layers of a CNN results in the reduction of spatial resolution so that their outputs are invariant to small changes in the input. In the architecture for CNN, there is no way to preserve the **pose** (translational and rotational) relationship between simpler features that make up the higher level features. So we can conclude that pooling operation within the CNN results in the loss of this valuable spatial information between the layers. In the tasks such as semantic segmentation this detailed information must be preserved throughout the network.

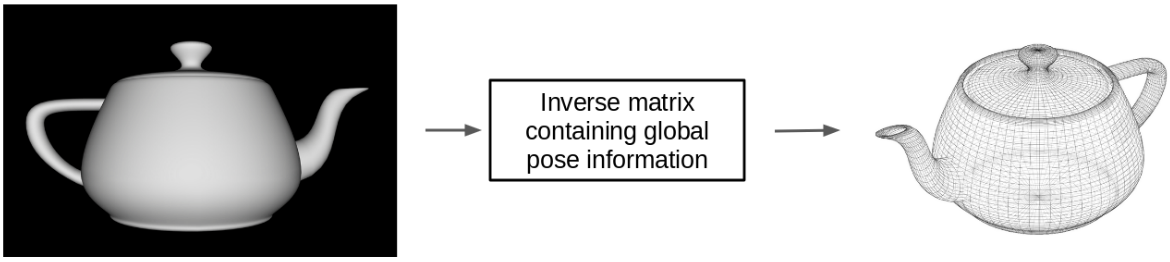
How Capsule Network Addresses the above problems in CNN

- Internal data representation of a CNN does not take important spatial hierarchies between simple and complex objects into account.
- The **main ideas** from this point are taken from the original paper on CapsNet by Geoffrey Hinton at NIPS 2017 – "*Dynamic Routing between Capsules*". This paper by Hinton was able to achieve state of the art performance on MNIST dataset.
- In computer graphics, the **rendering algorithms** are responsible converting the internal representation of hierarchical data (arrays of geometrical objects and matrices that represent relative positions and orientation of these objects) into the visual image we see on the screen.



Computer Graphics Rendering Process

- The human perception of interpreting the images is based on the idea of **Inverse Graphics** where we our brain tries to deconstruct a hierarchical representation of the world from the visual information we receive by our eyes and try to match it with already learned patterns and relationships stored in the brain. This representation of the objects in our brain does not depend on view angle. The key idea of CapsNet is to learn this brain like behaviour to retrieve important spatial information of the object representation.



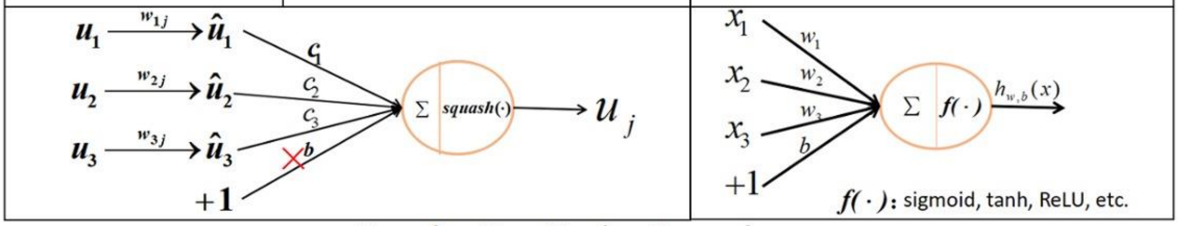
- This hierarchical relationships between 3D objects is modelled in a Neural Network with the pose information (translation + rotation) as the same way it is done in 3D graphics. This pose information is only neglected in CNN's (it only looks for features) which is preserved and explicitly modelled in these Capsule Networks that are capable of differentiating the images with variation in view angles.

Internal Working of Capsule Networks

- The main idea behind CapsNet is to implement capsule in the network (which is the group of neurons) that encode the spatial information along with the probability of an object or its part being present in the image. The network does so by encapsulating all the information in an activity vector associated with the capsule where vector length is the probability of feature existing in the image and the orientation or direction of this vector indicates the pose information.
- Therefore instead of using the pooling layers, the capsules will store all the information about the state of the feature being detected in the form of a output capsule vector as opposed to that in CNN where the output from the neuron is scalar.
- There are **4 operations** within a capsule which are explained by below figure
    1. **Matrix Multiplication of input vectors with weight matrices:** This encodes really important spatial relationships between low-level features and high-level features within the image.
    2. **Scalar weighting of input vectors:** These weights decide which higher level capsule the current capsule will send its output to. This is done through a process of dynamic routing.
    3. **Sum of weighted input vectors:** Same as we do in artificial neural network.

4. **Vector to Vector non-linearity using the squash function:** This function takes a vector and "squashes" it to have a maximum length of 1, and a minimum length of 0 while retaining its direction.

		capsule	vs. traditional neuron
Input from low-level neuron/capsule		vector(u_i)	scalar(x_i)
Operation	Affine Transformation	$\hat{u}_{j\|i} = W_{ij} u_i$ (Eq. 2)	–
	Weighting Sum	$s_j = \sum_i c_{ij} \hat{u}_{j\|i}$ (Eq. 2)	$a_j = \sum_{i=1}^{3} W_i x_i + b$
	Non-linearity activation fun	$v_j = \frac{\|s_j\|^2}{1+\|s_j\|^2} \frac{s_j}{\|s_j\|}$ (Eq. 1)	$h_{w,b}(x) = f(a_j)$
output		vector(v_i)	scalar(h)



Capsule = New Version Neuron!
vector in, vector out VS. scalar in, scalar out

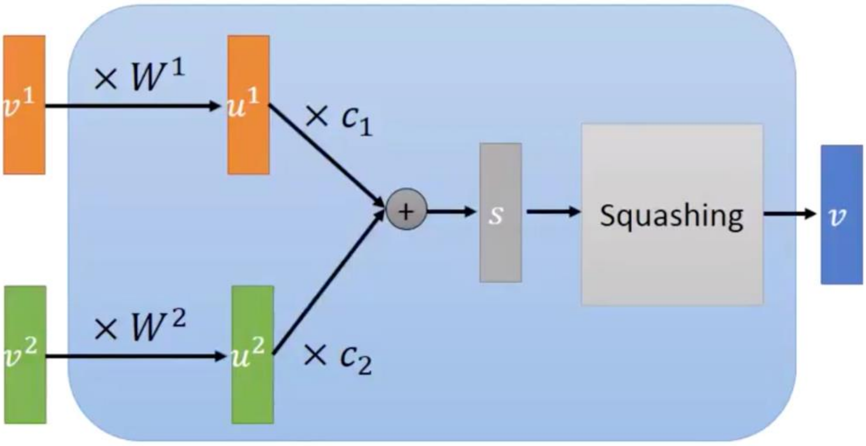
Capsule

$$u^1 = W^1 v^1 \qquad u^2 = W^2 v^2$$

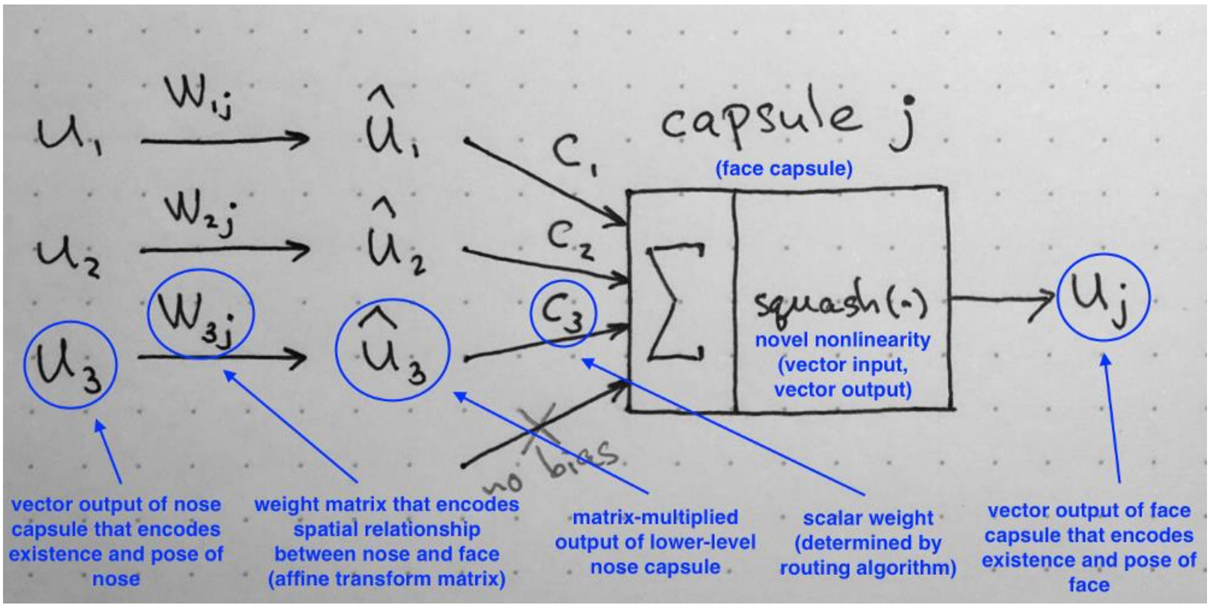
$$s = c_1 u^1 + c_2 u^2$$

$$v = Squash(s)$$

$$v = \frac{\|s\|^2}{1 + \|s\|^2} \frac{s}{\|s\|}$$



Operations in a Capsule Unit



Dynamic Routing Algorithm

- Iterative routing by mechanism is used in CapsNet to allow the training of the network.
- In this process of routing, lower level capsules send its input to higher level capsules that “agree” with its input. For each higher capsule that can be routed to, the lower capsule computes a prediction vector by multiplying its own output by a weight matrix. If the prediction vector has a large scalar product with the output of a possible higher capsule, there is top-down feedback which has the effect of increasing the coupling coefficient for that high-level capsules and decreasing it for others.
- *“A lower-level capsule prefers to send its output to higher level capsules whose activity vectors have a big scalar product with the prediction coming from the lower-level capsule” and following is the algorithm (from original paper)*

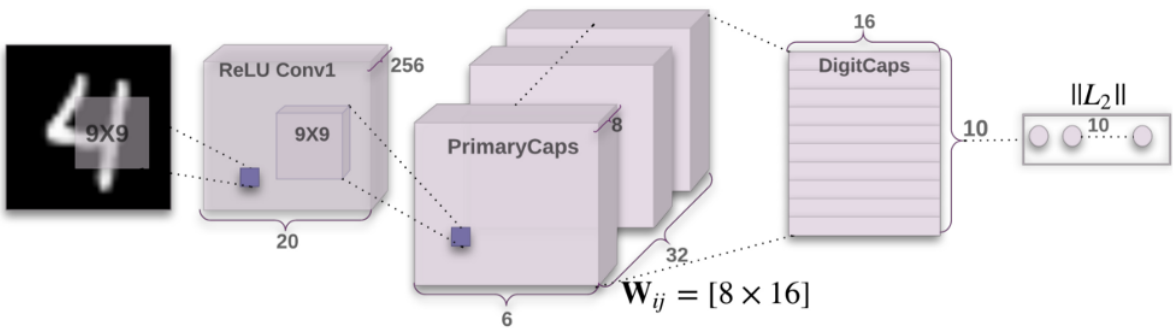
Procedure 1 Routing algorithm.

1: **procedure** ROUTING($\hat{\boldsymbol{u}}_{j|i}$, r, l)
2: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$.
3: **for** r iterations **do**
4: for all capsule i in layer l: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$ ▷ softmax computes Eq. 3
5: for all capsule j in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}$
6: for all capsule j in layer $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$ ▷ squash computes Eq. 1
7: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
return $\mathbf{v}_j$

- **Step 1:** This algorithm operates in all capsules in a lower level l and their outputs u^ and the r number of routing iterations (line 1). The algorithm will produce the output of capsule vector vj and provides the procedure to calculate the forward pass of the network.

- **Step 2:** The introduction of new coefficient bij is simply the temporary value that will be iteratively updated and is initialized to 0 at the start of the training. When the procedure is over its value will be stored in cij (line 2).
- **Step 3:** The steps in 4–7 will be repeated r times (the number of routing iterations).
- **Step 4:** Then in line 4, we calculate the value of vector ci which is all routing weights for a lower level capsule ci which is performed for all lower level capsules. Here we use softmax to ensure that each weight cij is the non-negative number and their sum is equal to 1.
- **Step 5:** After calculating all cij weights for the lower level capsules, we can move on to higher level capsules. Here we calculate the linear combination of input vectors, weighted by routing coefficients cij, determined in the previous step. Here we are simply scaling down the input vectors and adding them together to produce the vector sj which is replicated for all higher level capsules.
- **Step 6:** Then the vectors from last step are passed to the squash non-linearity which makes sure the direction of the vector is preserved and the length is set to maximum of one. This produces vj for all the higher level capsules.
- **Step 7:** This step is responsible for the weight update and captures the essence of routing algorithm. This step looks at each higher level capsule j and then examines each input and updates the corresponding weight bij according to the formula. The formula says that the new weight value equals to the old value plus the dot product of current output of capsule j and the input to this capsule from a lower level capsule i. The dot product looks at similarity between input to the capsule and output from the capsule. Also, remember from above, the lower level capsule will sent its output to the higher level capsule whose output is similar. This similarity is captured by the dot product. After this step, the algorithm starts over from step 3 and repeats the process r times.
- After r times, all outputs for higher level capsules were calculated and routing weights have been established. The forward pass can continue to the next level of network.

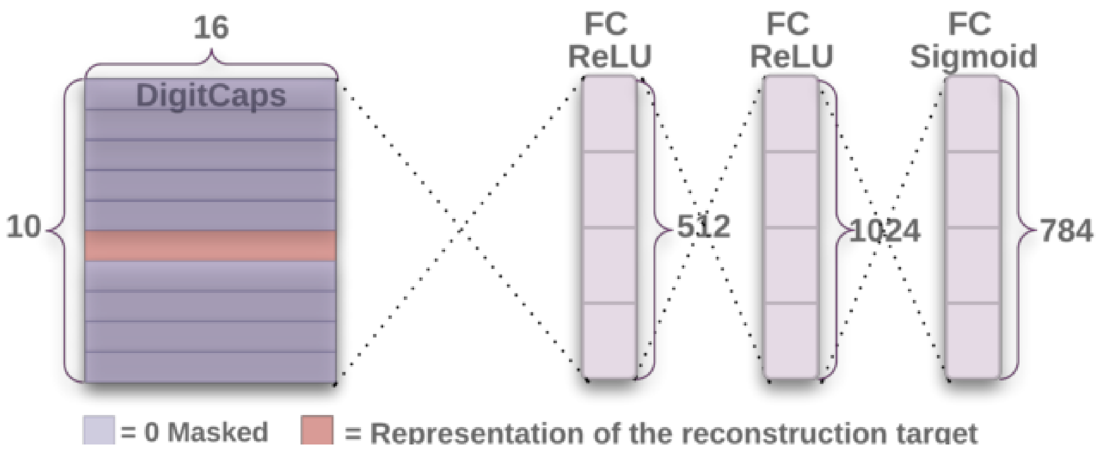
Proposed Architecture of CapsNet in Original Paper



Architecture of Encoder

Layers in Encoder

1) Layer 1 - **Convolutional layer:** Detects features that are later analyzed by the capsules. As proposed in the paper, contains 256 kernels of size 9x9x1.
2) Layer 2 - **PrimaryCaps layer:** This layer is the lower level capsule layer which I described previously. It contains 32 different capsules and each capsule applies eighth 9x9x256 convolutional kernels to the output of the previous convolutional layer and produces a 4D vector output.
3) Layer 3 - **DigitCaps layer:** This layer is the higher level capsule layer which the Primary Capsules would route to(using dynamic routing). This layer outputs 16D vectors that contain all the instantiation parameters required for rebuilding the object.



Architecture of Decoder

Layers in Decoder

The decoder takes the 16D vector from the Digit Capsule and learns how to decode the instantiation parameters given into an image of the object it is detecting. The decoder is used with a Euclidean distance loss function to determine how similar the reconstructed feature is compared to the actual feature that it is being trained from. This makes sure that the Capsules only keep information that will benefit in recognizing digits inside its vectors. The decoder is a really simple feed-forward neural net that is described below.

1) **Layer 4- Fully connected #1**
2) **Layer 5- Fully connected #2**
3) **Layer 6- Fully connected #3**

CapsNet Loss Function

calculated for correct DigitCap

calculated for incorrect DigitCaps

loss term for one DigitCap

L2 norm

L2 norm

$$\boxed{L_c} = \boxed{\boxed{T_c}\boxed{\max(0, m^+ - ||\mathbf{v}_c||)^{\boxed{2}}}} + \boxed{\boxed{\lambda}\boxed{(1 - T_c)}\boxed{\max(0, ||\mathbf{v}_c|| - m^-)^{\boxed{2}}}}$$

1 when correct DigitCap, 0 when incorrect

zero loss when correct prediction with probability greater than 0.9, non-zero otherwise

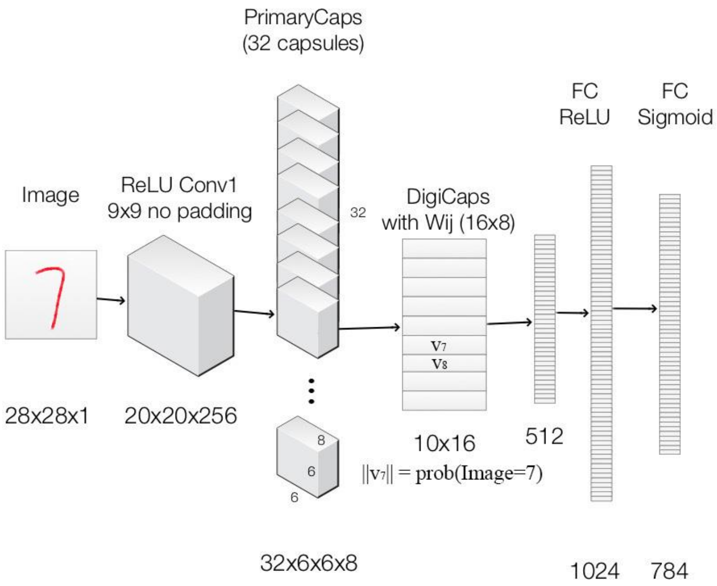
0.5 constant used for numerical stability

1 when incorrect DigitCap, 0 when correct

zero loss when incorrect prediction with probability less than 0.1, non-zero otherwise

Note: correct DigitCap is one that matches training label, for each training example there will be 1 correct and 9 incorrect DigitCaps

CapsNet Model Summary for MNIST



Layer Name	Apply	Output shape
Image	Raw image array	28x28x1
ReLU Conv1	Convolution layer with 9x9 kernels output 256 channels, stride 1, no padding with ReLU	20x20x256
PrimaryCapsules	Convolution capsule layer with 9x9 kernel output 32x6x6 8-D capsule, stride 2, no padding	6x6x32x8
DigiCaps	Capsule output computed from a W_{ij} (16x8 matrix) between u_i and v_j (i from 1 to 32x6x6 and j from 1 to 10).	10x16
FC1	Fully connected with ReLU	512
FC2	Fully connected with ReLU	1024
Output image	Fully connected with sigmoid	784 (28x28)

Potential Advantages of Capsule Networks over CNN's

- ✓ **Require Less Training data:** CapsNet can generalize well on the less training data and can achieve state of the art performance. This can be thought of the brain like behaviour as humans also don't much data to learn a particular pattern. But CNN's require much data to perform well on the unseen data which seems like brute force approach which our brain does not follows.
- ✓ **Require less parameters:** As CapsNet consists of capsules which are the group of neurons so the connection between the layers require fewer parameters.
- ✓ **Handle Ambiguity well:** CapsNet perform well on the crowded scenes so they can handle ambiguity much better than CNN's.
- ✓ **Preserve important information:** As much information is lost in CNN because of pooling operations, on the other hand CapsNet don't have pooling layers so detailed pose information such as object position, rotation, thickness is preserved in the network. This results into achieving equivariance which means a small change in input will result in small change in output – this property was missing in CNN's.
- ✓ **Viewpoint Invariant:** Because a CapsNet stores the pose matrices to recognize the objects, they are able to differentiate the same object which has different viewpoints.
- ✓ **No requirement of extra components:** CapsNet preserve the hierarchy of spatial information in the image so it is very easy to locate the object to which a particular part belongs to. This was not possible in CNN as it requires to add the additional components.
- ✓ **Defense against white box adversarial attacks:** FGSM (Fast Gradient Sign Method) is the technique which attacks the CNN by changing the pixel distributions and maximizing the loss. This can drop the performance of CNN drastically whereas CapsNet tends be unaffected from this condition.

References

[1] Original Paper: http://www.csri.utoronto.ca/~hinton/absps/transauto6.pdf

[2] CapsNet Paper: https://papers.nips.cc/paper/6975-dynamic-routing-between-capsules.pdf

[3] Wikipedia link to Capsule Nets: https://en.wikipedia.org/wiki/Capsule_neural_network

[4] CapsNet by Aurélien Géron: https://www.oreilly.com/content/introducing-capsule-networks/

[5] A detailed blog: https://towardsdatascience.com/capsule-networks-the-new-deep-learning-network-bd917e6818e8