Deep Leaning Assignment 2

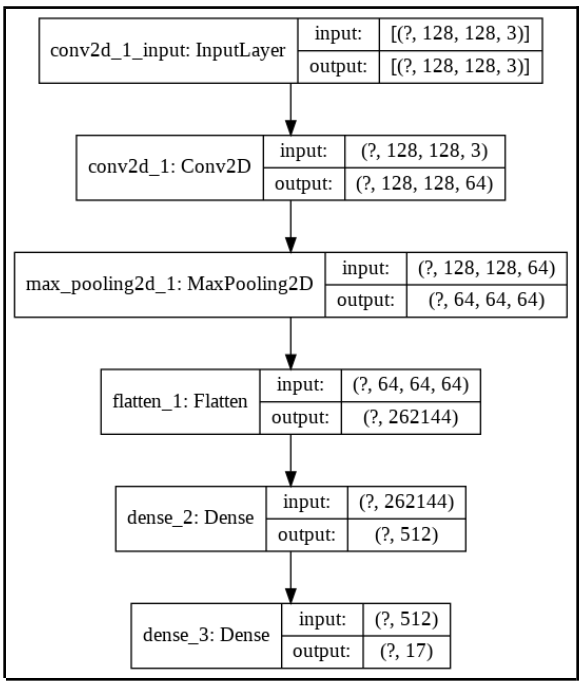
Name : Pavithra Ramasubramanian Ramachandran
Student ID : R00183771

Part A : Convolutional Neural Networks

- A total of 5 models including the mentioned baseline model are created with increasing number of layers in the Convolution Neural Network.
- Checkpointing is enabled for all the models as it helps in the augmentation part of the question.
- The number of classes in the dataset is calcultaed using np_utils.to_catgorical() function.
- Also, the height, width and depth of the images are determined using shape of trainX.

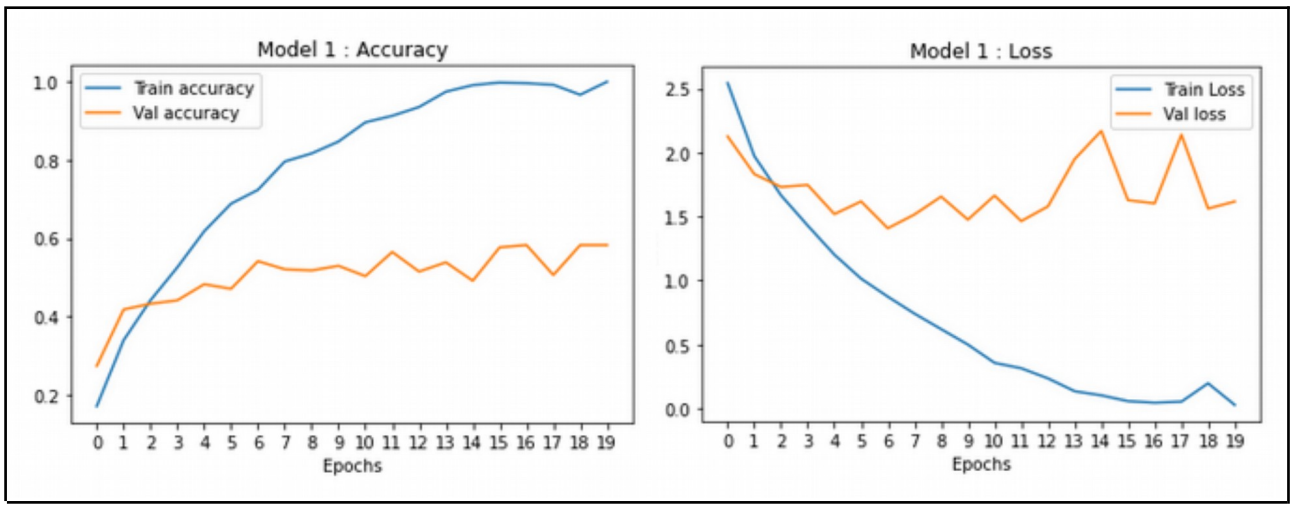
Model 1: Baseline Model

The baseline model consists of a single cnvolutional layer, a pooling layer, a fully connected layer and a softmax layer.



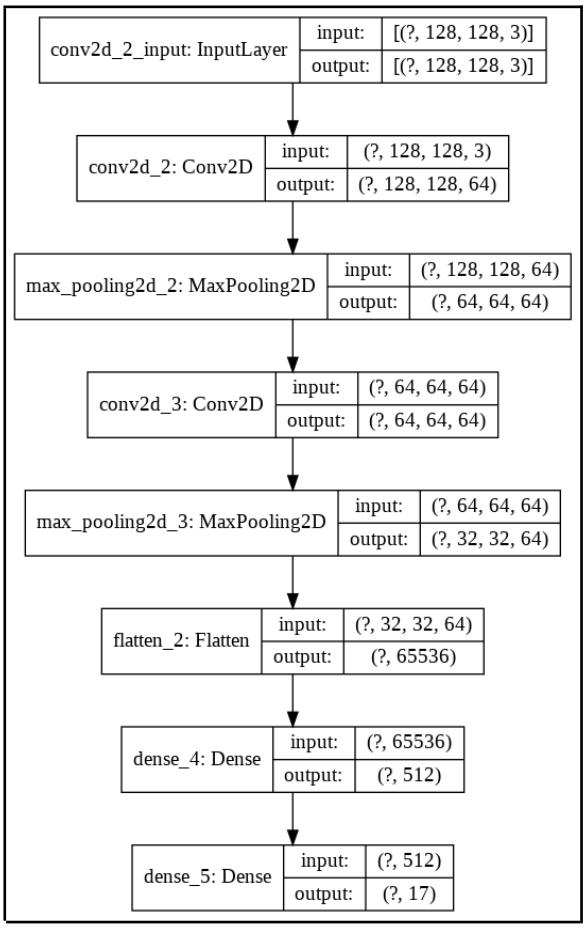
On evaluation, the model gives an accuracy of 54.11% and validation loss of 1.4073.

The below graphs depict the performance (train and validation accuracy and loss)of the model over 20 epochs. It can be observed from the graphs that the train accuracy increases rapidly after the 5 epoch. However, the validation accuracy starts becoming stable. Similary, the validation loss doe not reduce as much as the train loss. The gap between the train and validation curves clearly depict overfitting of the training data on the model. Therefore, the model most probably will not work well with any unseen data.



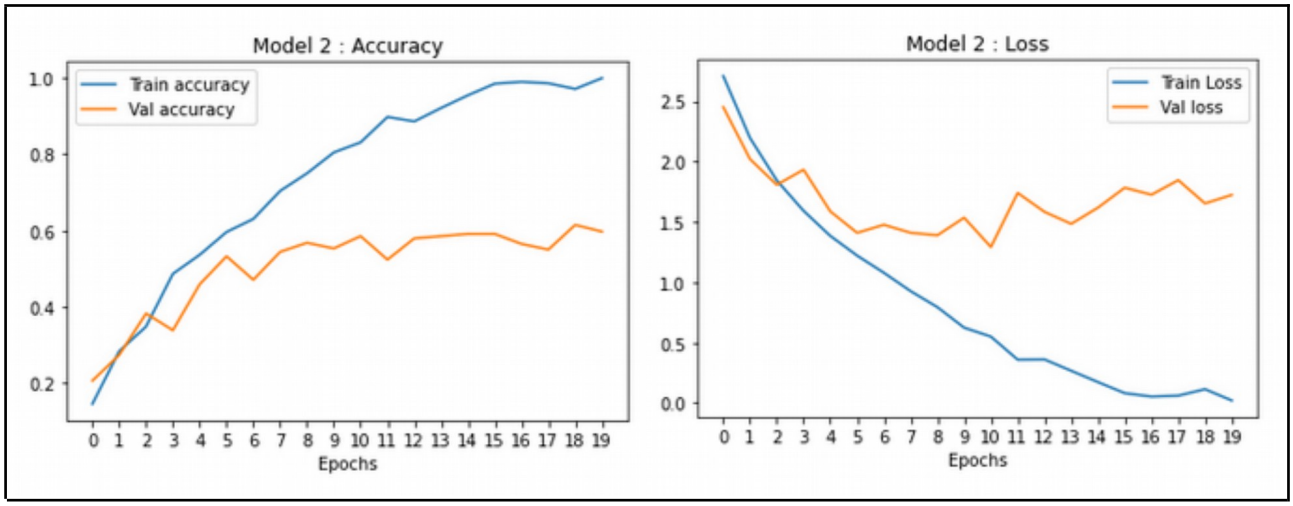
Model2

This model consists of two convolutional layers followed by pooling layers, then a fully connected layer and softmax layer.



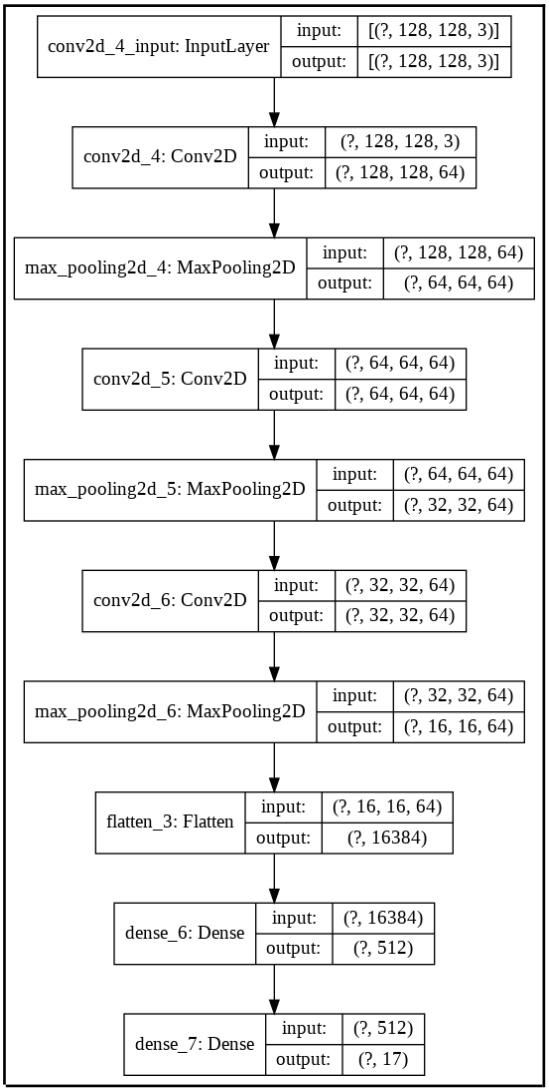
On evaluation, the model gives an accuracy of 58.29% and validation loss of 1.293.

The graphs below depict the performance of model2. Thes graphs are similar to that of model1. Both the trainand validation curves have a lot of gap between them, depecting overfitting.



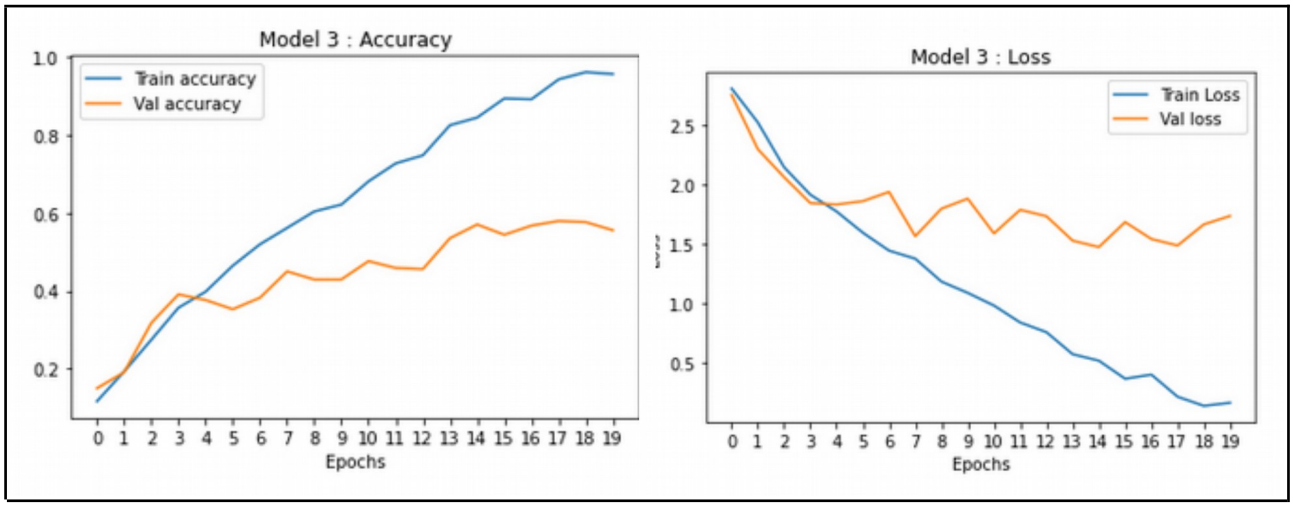
Model 3

This model consists of 3 blocks of convolutional and pooling layers, followed by a fully connected and softmax layer.



On evaluation, the model gives an accuracy of 57.05% and validation loss of 1.4726.

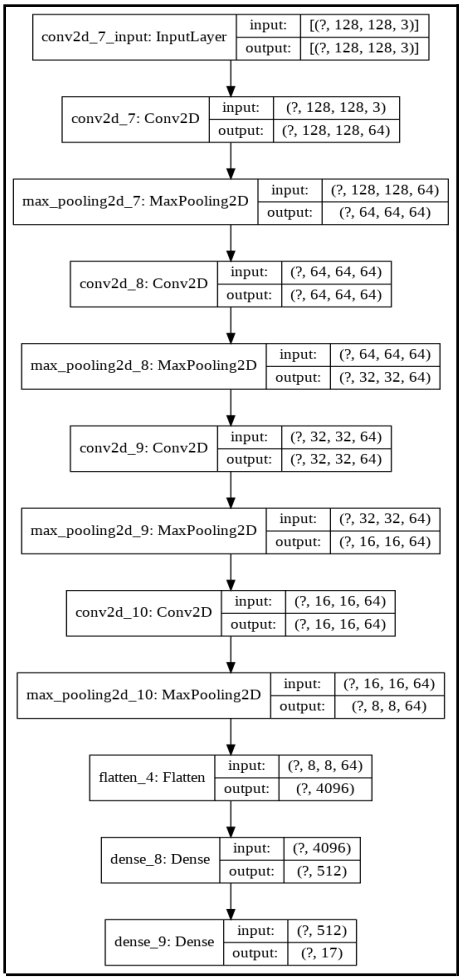
The below graphs represent the performance of the model.



The train and validation curves again have a considerably huge gap between them in both the accuracy and loss graphs indicating severe overfitting.

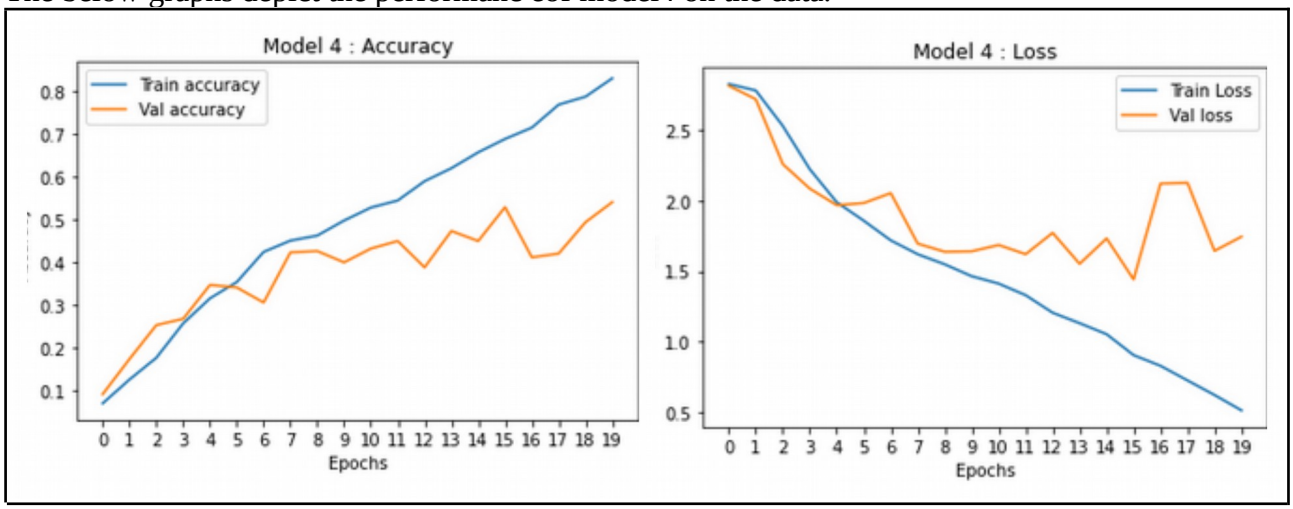
Model4

This model consists of 4 blocks of convolutional and pooling layers, followed by a fully connected and softmax layer.



On evaluation, the model gives an accuracy of 52.94% and validation loss of 1.4412.

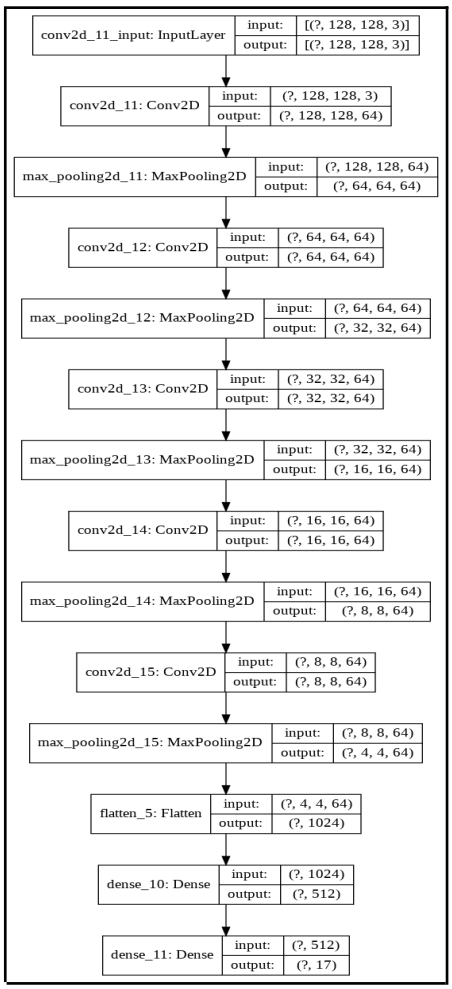
The below graphs depict the performanc eof model4 on the data.



It can be observed from the above graphs that the accuracy and loss curves for both the train and validation data tend to go in the same direction until the 9th epoch. After that, they develop a huge gap again depicting overfitting.

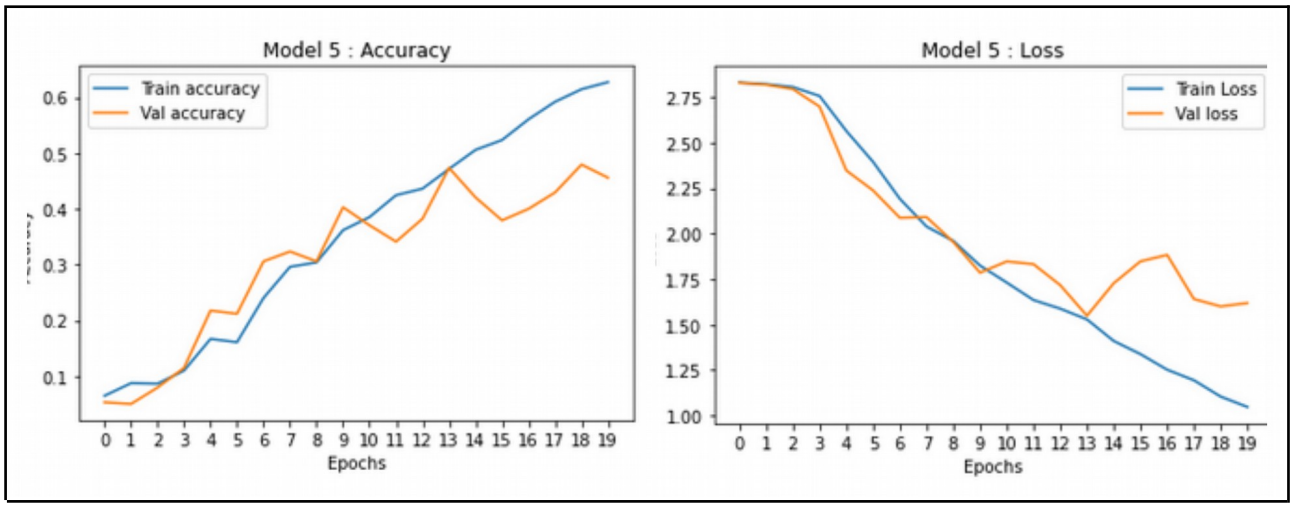
Model 5

This model consists of 5 blocks of convolutional and pooling layers, followed by a fully connected and softmax layer.



On evaluation, the model gives an accuracy of 47.35% and loss of 1.5484.

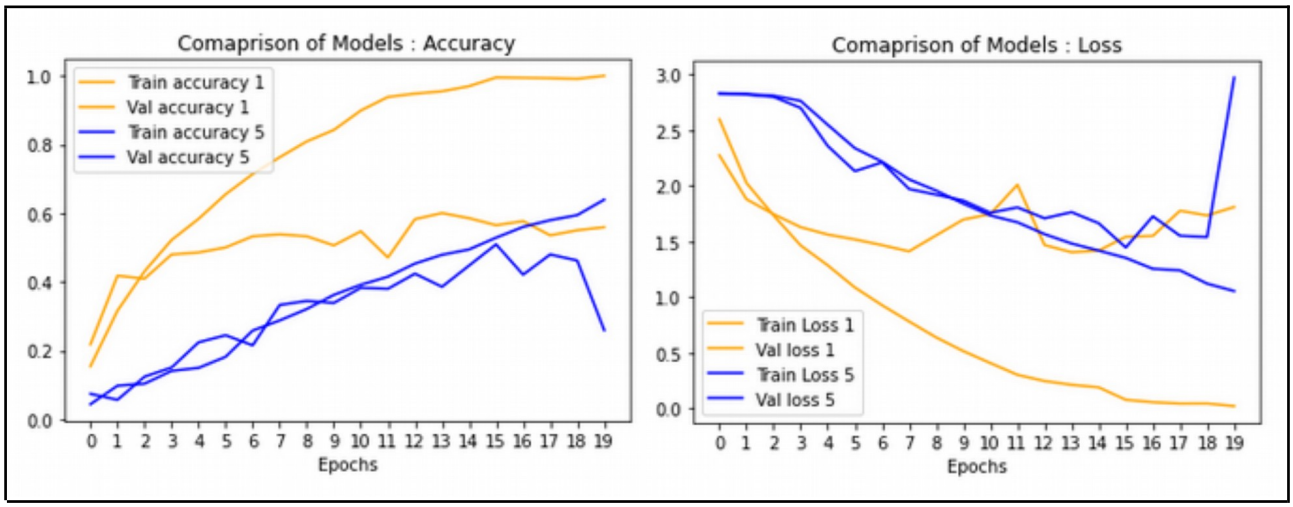
The below graphs depict the performance of model5.



It can be observed from the above graphs that in this model, overfitting starts to show up much later in the 13th epoch. Until then, the model seems to be performing pretty well in terms of being able to predict unseen data although the accuracy is really low(47%) which is not acceptable.

Overall Observations:

It can be observed that model1 started overfitting right from the beginning whereas as the number of layers(convolutional + pooling) began to increase in the models, the phenonmenon was observed a little later(after the 9th epoch) or so.



The above graphs depicts the comparison between the two models – model1 and model5. The orange curves represent model1 and the blue curves represent model5. It is so evident from the graphs that the increase in number of layers in the model has considerably reduced overfitting but at the cost of accuracy. Accuracy of model 2 was the highest(58.5%) which is still not a great number. Every model after model2 has recorded lower accuracy. This implies that the model is not efficient enough to capture all the features from the images to predict the test samples accurately.

However, it can't be concluded that increasing layers in the model would increase accuracy and reduce overfitting. The number of layers in a model should be determined based on the amount of training data and several other factors. A model with say 20 blocks of convolutional and pooling layers may fail terribly for this data.

Data Augmentation

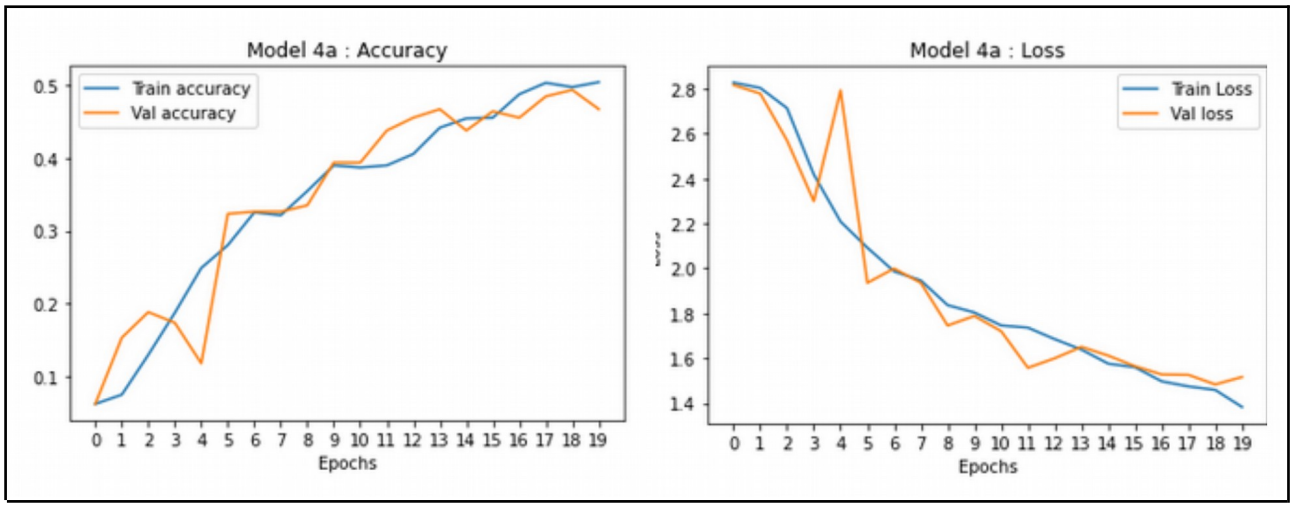
Data augmentation is one of the most recommended techniques to reduce overfitting in deep learning models. It comes in very handy when the train data is very small and may not be enough for the model to learn features from.

Two models – model4a and model5a are replications of model4 and model5 respectively with data augmentation.

Model4a : model4 + Data augmentation

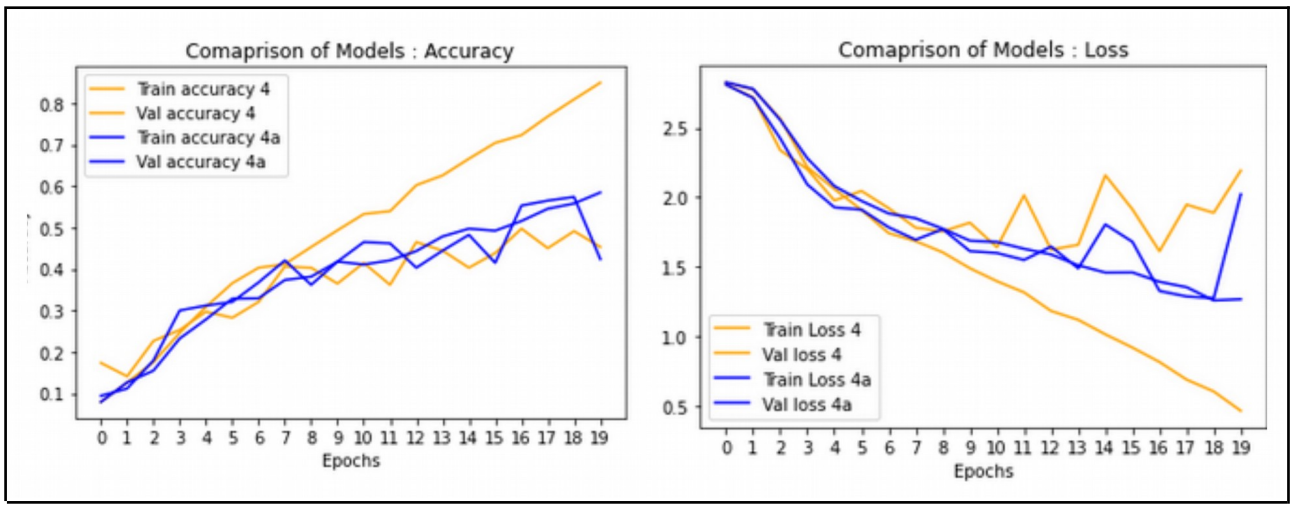
On evaluation, this model gives an accuracy of 48% and loss of 1.4853.

The below graphs depict the performance of model4a for the dataset.



It can be observed that the model does not overfit at all. Both the blue and orange curves emerge together across increasing epochs. Although accuracy of this model is much lesser that model4, it is more generalized than model4. On training further for more number of epochs, the model will definitely give better accuracy.

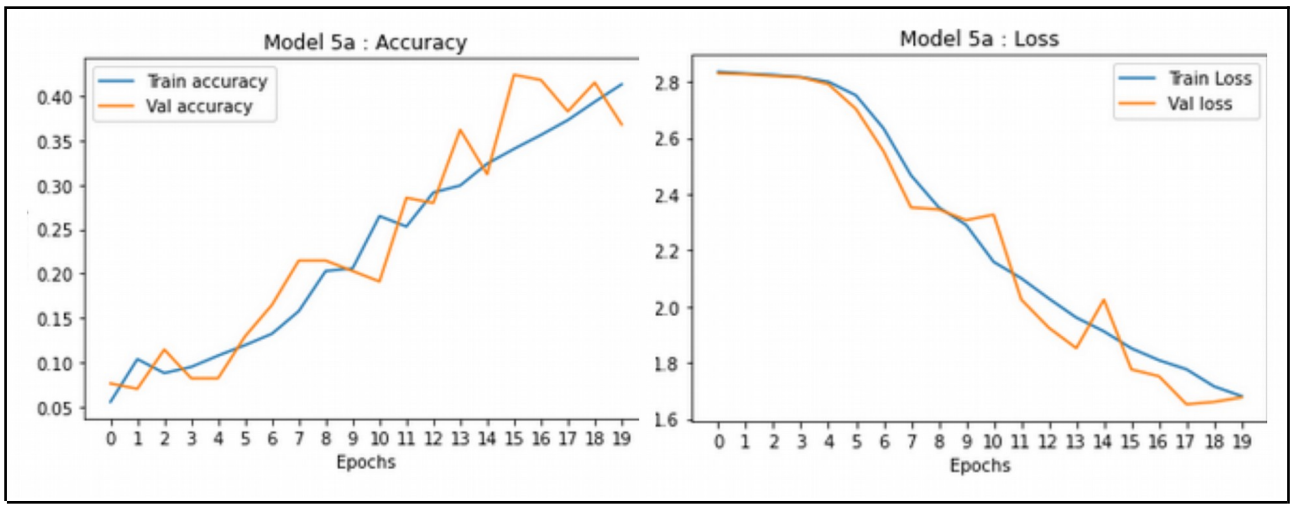
The graphs depicting the comparison between model4 and model4a is below.
Clearly, model4a does not overfit and also tends to show an increase in accuracy on training further.



Model5a : Model5 + Data Augmentation

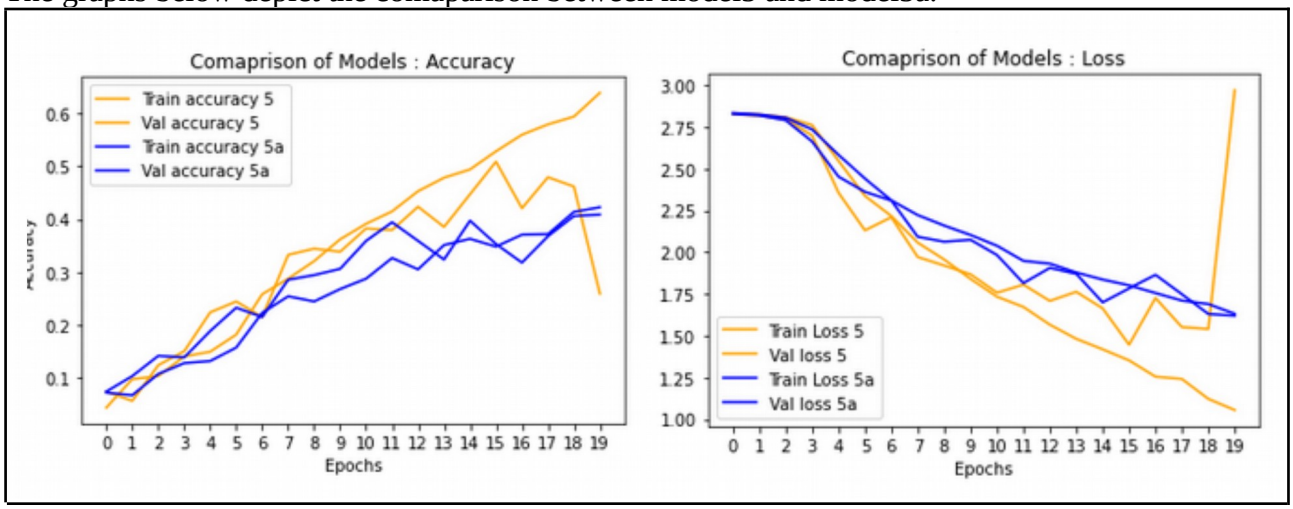
On evaluation, this model gives an accuracy of 39.7% and loss of 1.633.

The below graphs depict the performance of model5a for the dataset.



Similar to model4a, this model also does not overfit and would definitely give a better accuracy on further training.

The graphs below depict the comaparison between model5 and model5a.



Although the accuracy is lower, model5a does not overfit as early as model5.

Impact of Data Augmentation Techniques

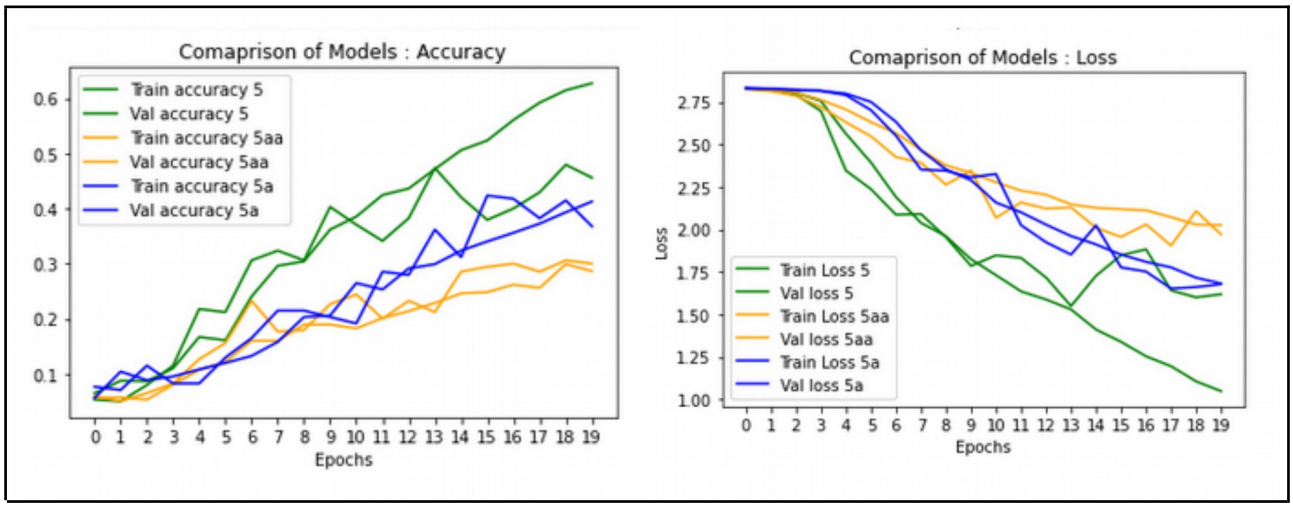
In order to illustrate the impact of data augmentation techniques used in the model, we shall use model5, model5a and model5aa.

Model5 – 5 blocks of convolution+pooling layers , no data augmentation

Model5a - 5 blocks of convolution+pooling layers , data augmentation (shear_range=0.2,
zoom_range=0.2,
rotation_range=30,
horizontal_flip=True)

Model5aa - 5 blocks of convolution+pooling layers , data augmentation (shear_range=0.2,
zoom_range=0.2,
rotation_range=30,
horizontal_flip=True,width_shift_range = 20.0,
height_shift_range = 20.0,
vertical_flip = True)

The below graph depicts the impact of data augmentation on the model.



The green lines represent accuracy and loss curves of train and validation data of model5, blue curves of model5a and orange curves of model5aa. It can be observed that model5 has the highest accuracy but begins to overfit severely after the 13th epoch. Model5a has quite a bit of noise in the graphs. Howver, model5aa does not show much noise and is smoother than model5a. This implies that having more number of variations of the same image in the train data ensures that the model can handle any variant of the image and therefore perform more stably. It also tends to imrpove the accuracy although in the above graph it does not happen so, as the number of epochs was curtailed to 20. On further training, the model5aa will begin to preform better and also will not show a tendency of overfitting.

Ensemble Models

To create an ensemble model, 8 base learners were used. The 8 models built in the previous exercise are used to create the ensemble model. The models were built with checkpoints so as to capture the best weights for each model.

Models used to create the ensemble:

Model	Validation Loss	Validation Accuracy
Model1	1.4073	54.11%
Model2	1.293	58.53%
Model3	1.726	57.05%
Model4	1.4412	52.94%
Model5	1.5484	47.35%
Model4a	1.4853	48.82%
Model5a	1.6333	39.7%
Model5aa	1.956	29.41%

Ensemble Process:

Ensembles can be created using different methodologies, namely, average, weighted average, majority voting, etc. Here, we are going to create an ensemble by using the averaging technique. The ensemble model of these 8 models was created by averaging the softmax outputs(probabilities) of the each of the train sample of each of the 8 models.

For instance,

Prediction of model1 for test sample 1: [0.3, 0.4, 0.1, 0.2]
Prediction of model2 for test sample 1 : [0.6, 0.2, 0.1, 0.1]

We average the probabilities of the sample element-wise to determine whether the sample belongs to class 1,2,3 or 4.

Average = [0.45, 0.3, 0.1, 0.15]

The class with the highest probability after averaging is outputed as the class of the test sample. Here, class 1 has the highest average probability. Therefore, the test sample will be classified as class1 sample.

This ensembling can be coded in two ways:

- Low level programming
  This can be done by using predict() on every model to predict the probabilities of every class for all the test samples. Then using np.mean(), the averages can be computed and the using np.argmax() function, the class with the highest average probability can be determined.

- Using Keras package

In this methodology, we use nested modelling to create our ensemble model. We define an input layer with the shape of the images. Then, create a nested model of the base learners with the specified input layer. The output layers will be the average of the layers of these base learner models. Using the defined input and output, anensemble model can be created, compiled and tested.

Both the techniques gave the same output : validation loss – 1.27, validation accuracy – 60.88% which is better than the performance of all the individual base learners.

Variability in Base Learners for Ensemble Model

It is essential to have a variety of models to create an ensemble. This aspect ensures that the positives of all models are captured and the negatives of the models are cancelled out by the positives of the models. Two main aspects to be considered to create an ensemble model are:

- The base learners should excel in some metric(accuracy, recall, F1 score, etc.)
  It is a good technique to choose base learners based on metrics. If one of the models can give a very good accuracy and if some other model can give a very good recall or loss score, the combination of these models can give us the best of both the models while cancelling out atleast a part of the negatives of the base learners.

- The base learners should have some correlation.
  This aspect matters when the base learners are all trained on different datasets. It is important to find the correlation betweeen the datasets that the base learners were trained on. And doing so will yield the best possible combination of base learners, and therefore a very high performance ensemble.

In order to illustrate that choosing models with good metrics could have an impact on the ensemble's performance, an ensemble of models with validation loss below 1.5 was created (models 1,2,3,4 and 4a). It turned out that the ensemble gave an accuracy score of 61.76% which is higher than the the accuracy score that the ensemble of the 8 models gave(60.88%).

Also, 2 models which had accuracy greater than 55% were used to create an ensemble model and model gave an accuracy score of 58.29% which is just 2% lesser than the ensemble with 8 base learners. Therefore, choosing a variety of base learners plays an improtant role in the performance of the model.

Performance of Base Learners vs. Ensembles

Model1:

Model	Validation Loss	Validation Accuracy	Ensemble Accuracy
Model1	1.4073	54.11%	60.88%
Model2	1.293	58.53%	
Model3	1.726	57.05%	
Model4	1.4412	52.94%	
Model5	1.5484	47.35%	
Model4a	1.4853	48.82%	

Model5a	1.6333	39.7%	
Model5aa	1.956	29.41%	

Model2 : Val loss <1.5

Model	Validation Loss	Validation Accuracy	Ensemble Accuracy
Model1	1.4073	54.11%	61.76%
Model2	1.293	58.53%	
Model3	1.726	57.05%	
Model4	1.4412	52.94%	
Model4a	1.4853	48.82%	

Model3: Val accuracy > 55%

Model	Validation Loss	Validation Accuracy	Ensemble Accuracy
Model2	1.293	58.53%	58.52%
Model3	1.726	57.05%	

Part B : Transfer Learning

Feature Extraction

Feature extraction is a process where features from pre-trained models are extracted, data is trained on these features and then a classifier is used to predict the classes of the samples. Three pre-trained CNN models were used as features extractors and 3 standard machine learning algorithms were used as classifiers.

Feature Extractors :

- VGG16 model
- Inception model
- Xception model

Classifiers :

- Random Forest Classifier
- Logistic Regression Classifier
- SVC

Performance of the feature extraction models

Feature Extractor	Classifier	Train Accuracy	Validation Accuracy
VGG16	Random Forest	100%	82.05%
	Logistic Regression	100%	87.94%

	SVC	89.9%	84.41%
Inception	Random Forest	100%	81.17%
	Logistic Regression	100%	84.7%
	SVC	95.68%	84.41%
Xception	Random Forest	100%	77.35%
	Logistic Regression	100%	81.76%
	SVC	91.17%	75.58%

From the above table, it is evident that both Random forest and Logistic Regression models seem to ovefit extensively such that their training accuracy have reached a 100%. Even changing the number of decision trees in the Random Forest classifier to 20 did not reduce the overfitting problem. Only the SVC() classifier performed fairly well with all the three feature extractors, the best being with VGG16 where overfitting is minimal(train accuracy = 89.9% and validation accuracy = 84.41%). Inception and Xception models along with SVC() classifier still exhibit a considerably large amount of overfitting.

Fine Tuning

Fine tuning is the process of fine tuning the weights of a pre-trained model in order synchronize with a specific dataset.

Extensive exploration was done on three pre-trained models(VGG16, Inception and Xception models). After all the exploring, the best model that I could get was a *VGG16 model fine-tuned to the Flower Dataset + Data Augmentation with a validation accuracy of 84.12% and validation loss of 0.6793. The model also exhibited the least amount of overfitting(train loss = 0.6264).*

A few of the experiments that I carried out are enlisted below along with their performances:

VGG16 pre-trained model +Data Augmentation

- with a dropout layer(0.2), a fully connected layer(256 neurons), Loss fucntion – Sparse Categorical loss, Optimizer - SGD, Phase 1 learning rate= 0.01, layer until ‘block4_conv1’ frozen, Phase 2 learning rate = 0.00001. These parameters resullted in the best model with validation accuracy of 84.12% and train and validation loss of 0.6264 and 0.6793 respectively.
- With the same configurations as the above model but all layers until ‘block3_conv1’ frozen and phase2 learning rate = 0.000001. This model gave a validation accuracy of % and train and validation loss of 0.589 and 0.6629 respectively. The model exhibited quite a bit of overfitting in comparison with the best model.
- With the same configurations as the above model except that of the optimizer. The optimizer used was ‘RMSprop’. This model gave a very good validation accuracy of 92.94%. However, the train and validation loss was 0 and 0.34 respectively clearly indicating extensive overfitting.
- With the same configurations as the above model(RMSprop), except that the phase 2 learning rate = 0.0001. This model gave an output of 91.76% as validation accuracy but also exhibited overfitting(train accuracy = 0.222 and validation accuracy = 0.6).
- With Adam Optimizer and the same model as above, the model gave a validation accuracy of 91.76% but again exhibited overfitting(0,0.5 – train,validation loss respectively).

- Addition of extra fully connected layers or tampering the dropout parameter only reduced the accuracy and led to more overfitting.

Inception pre-trained model+Data Augmentation

- with dropout layer(0.3), a fully connected layer(512 neurons), Loss fucntion – Sparse Categorical loss, Optimizer - SGD, Phase 1 learning rate= 0.01, layer until 'conv2d_85' frozen, Phase 2 learning rate = 0.00001. These parameters resullted in the best model with validation accuracy of 83.94% and train and validation loss of 0.1316 and 0.6308 respectively. From the loss values, it is clear that this model overfits.
- With the same configuration as above but phase 2 learning rate = 0.0000001. The validation accuracy remained close = 82.94% with losses 0.1364 and 0.6308 which is very similar to the previous model.
- With same configurationa s the first model, but layers frozen until 'conv2d_50'. The model perform very similarly with some overfitting.
- With configurations similar to the 3rd model, except for the optimizer – RMSprop. The model gave a validation accuracy of 85.2% but exhibited extreme overfitting.
- With many other changes to the number of layers frozen, learning rate, optimizer, etc. None of the models exhibited both good accuracy on validation data and acceptable gap between validation and train losses.

Xception pre-trained model+Data Augmentation

- with 2 fully connected layer(512 neurons), Loss fucntion – Sparse Categorical loss, Optimizer - SGD, Phase 1 learning rate= 0.001, layer until 'block8_sepconv1' frozen, Phase 2 learning rate = 1e-8. These parameters resullted in the best model with validation accuracy of 4% and train and validation loss of 2.8 and 2.84respectively. The performace of this model is too poor.
- With the same configuration as the above model except all layers until 'block10_speconv1' frozen and phase 2 learning rate =0.00001. The model still performed very poorly.
- Many combinations of optimizers, learning rates, number of dense layers, number of frozen layers, etc. Were yielded no good results.

Part C : Research(Capsule Networks)

What is the problem?

Convolutional Neural Networks are the talk of the town in the field of deep learning. CNNs have been peforming extremely well when it comes to image recognition, image classification, etc. Several techniques have evolved using CNNs breaking barriers in the deep learning world. Howver, CNNs also have several flaws with their architectures which gives immense scope for research. In a CNN model, the basic workflow would be this. The early layers in the model would identify small features in the data and the last few layers would combine these tiny features and identify bigger, wholesome features and finally predictions would be made. In this process, one major factor remains ignored - the spatial position of the features in the image. For instance, let's take a car. A car has wheels, windows, doors, a structure to its body, etc. If the image of a car is inputed to a CNN, the early layers would identify the horizontal lines, vertical lines, the curves, etc. The middle layers, combine these tiny features to identify wheels, windows, doors, etc, and the last few layers combine the wheels, doors, etc to identify that the object in the image is a car. During this process, the CNN is almost unaware of the position of the wheel in car, position of the door in the car, etc. This sometimes leads to incorrect classification of the image.[2]

Another major aspect affecting the CNNs performance is the usage of pooling layers. Pooling layers are used in CNNs to reduce the size of the feature maps so that the computational time of the model is reduced considerably. Howver, everytime a max pooling layer is used in the network, only pixels with the highest values are chosen and the others are omitted in the pool sized filter. This process dramatically reduces the ability of the network to spatially determine the position of features. Therefore, affecting the prediction process.[2]

In order to address these issues and educate the network about the spatial orientation of the features, capsule networks were developed.

How does a capsule network operate? What are its advantages and disadvantages?

The basic operation of a capsule network is to identify low level features, and forward these features to higher layers only if these features belong to the high level features. In our car example, the low level feature rim will be forwarded to the high level feature car only; the low level feature handle will be forwarded to the high level feature door only. This is done by a using atechnique called “routing-by-agreement” in capsule networks. This is a dynamic routing feature of capsule networks which enables forwarding of low level features to higher level features only if they are specific to them and belong there.

Now the question is how to determine to which high level feature does a low level feature belong to? We have come across many engineering subjects that relate to computer aided drawing, geometric positioning of objects, etc. In these subjects, the task was to correctly place an object in a space based on parameters like distance from a point, angle of orientation, etc. Based on these parameters, we can place that object exactly in the position that it is required to be. Keeping this in mind, can we also determine these parameters by looking at an object in space? We can determine the distance of the object from specific points, their orientation with respect to some reference points. This is exactly what capsule networks do. They derive the postion of the objects in the image. This is called “inverse rendering”. Capsules not only identify features in an image but also derive the spatial data of the features, therefore enabling it to be able to be able to route the features to the right higher level features.

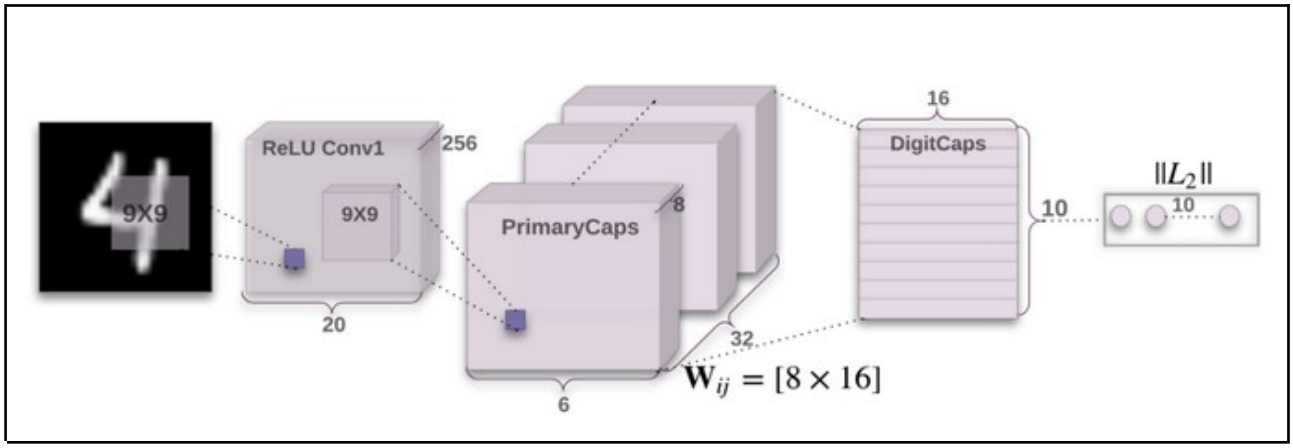
How is this done? Groups of neurons in the network capture both the probability of the existence of a feature in the network and also the spatial information of the feature. The length of a capsule vector is the probability of the feature existing in the image and the direction of the vector would represent its pose information. The 'Dynamic Routing between capsules’ paper by Geoffrey Hinton proposed the use of two loss functions as opposed to just one – one for the existence of features in the network and one for the spatial information of the feature in the network. The two loss functions are to create equivariance between capsules, that is, the loss function for the spatial information of a feature would keep changing as we move across the image and would be least where the feature exists and the loss function for the existence of a feature would remain the same throughout the image.[2]

Architecture of Capsule Network on MNIST

Capsule networks comprise of two main components :

- Encoder
- Decoder

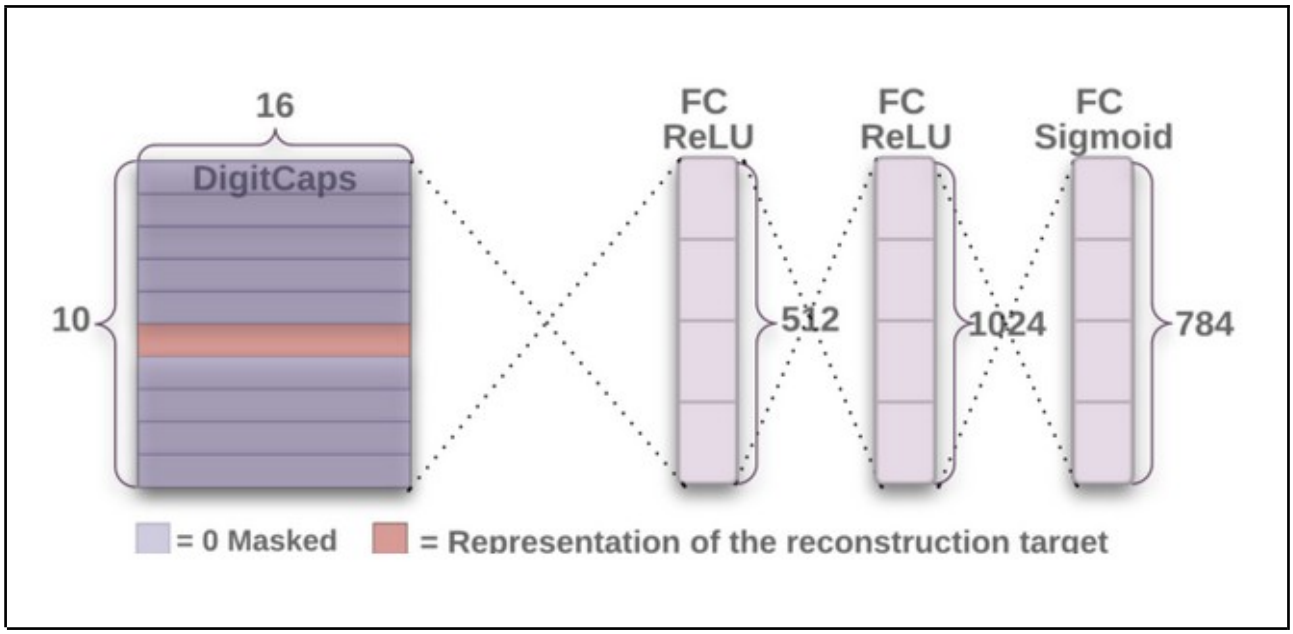
Encoder



The above diagram depicts the architecture of a capsule network's encoder. The encoder does three main jobs:

- It takes the image as an input and extracts features from it using the Convolutional Layer.
- Primary Capsule layer : There are 32 capsules. Each of these capsules capture not just the probability of existence of features in the image but also capture the direction(orientation) of the features and output a 4D vector. The 4th dimension is the direction.
- Digit Capsule Layer : The primary capsule layer determines the spatial information of the low level features and re-routes the features to the high level features in the Digit capsule layer. This layer then outputs a 16D vector which has all the parameters required by the model to reconstruct the image using the features.[3]

Decoder



The above diagram depicts the architecture of the Decoder. The decoder consists of fully connected layers. It takes the 16D vectors from the Digit Capsule layers(enough information to reconstruct the image), uses Euclidean distance as a loss function to ensure correct spatial orientaion of the image features and then it eventually classifies the image.[3]

Disadvantages :

Sometimes, these networks get to complex when there are complex images as there is so much information to record and forward. It becomes a computationally expensive process.[3]

References:

1. Dr. Ted Scully' s lecture notes
2. Sara Sarbour and Nicholar Frosst, " Dynamic Routing between Capsules", 2017.
3. https://towardsdatascience.com/capsule-networks-the-new-deep-learning-network-bd917e6818e8