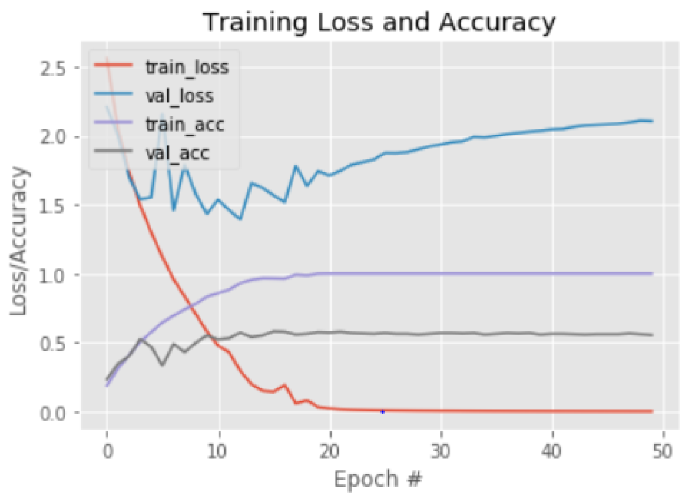
R00182510 - DL Assignment 2

Part A - Task 1

Baseline CNN

The model was implemented with a single convolutional layer and max pooling layer, a fully connected layer (ReLu) and a final SoftMax layer. Epochs = 50.



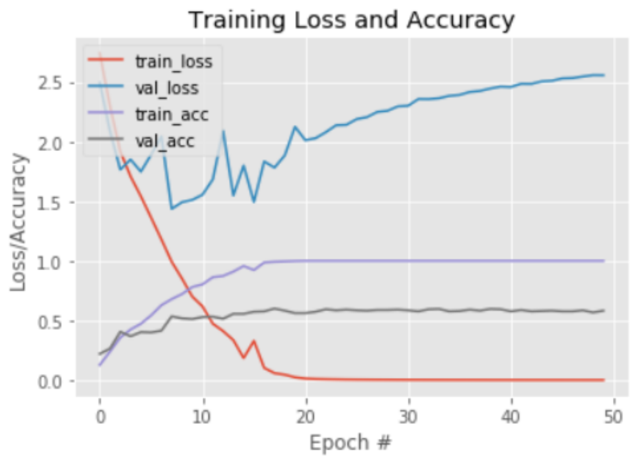
Final Training accuracy: 1.0000 **Final Training Loss:** 0. 0.0026

Final Validation accuracy: 0.5559 **Final Validation Loss:** 2.1057

The training accuracy reached 100% whereas the validation accuracy is almost halfway behind, which indicates that the model is significantly overfitting after just 4 epochs.

CNN Network 1

The baseline model was modified to include an additional convolutional and max pooling layer before the fully connected layers. Epochs = 50.



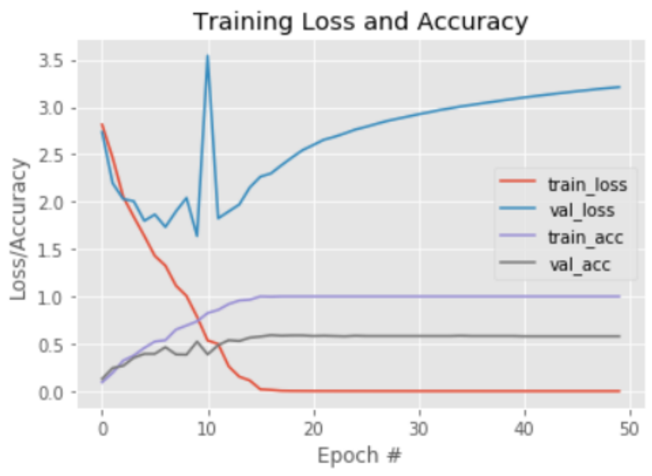
Final Training accuracy: 1.0000 **Final Training Loss:** 0.0012

Final Validation accuracy: 0.5824 **Final Validation Loss:** 2.5555

The updated model seems to still be overfitting and the validation loss has increased than what was observed in the baseline model, indicating a slightly more overfitting.

CNN Network 2

The next variation with respect to the architecture included 2 convolutional layers stacked together before both the max pooling layers from CNN Network 1. Epochs = 50.



Final Training accuracy: 1.0000 **Final Training Loss:** 2.0323e-04

Final Validation accuracy 0.5794 **Final Validation Loss:** 3.2106

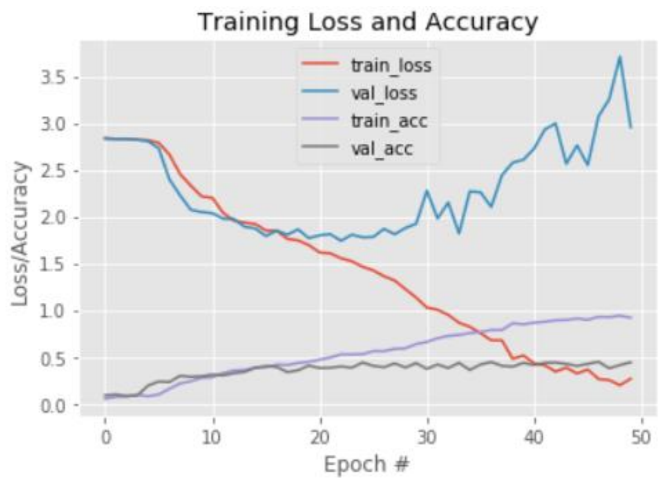
The overfitting has again increased more than the previous model.

CNN Network 3

A deeper network was studied in this model, with 3 repeated stacks of 2 convolutional layers and a max pooling layer before the fully connected layer. Epochs = 50.

Final Training accuracy 0. 9186 **Final Training Loss:** 0. 2648

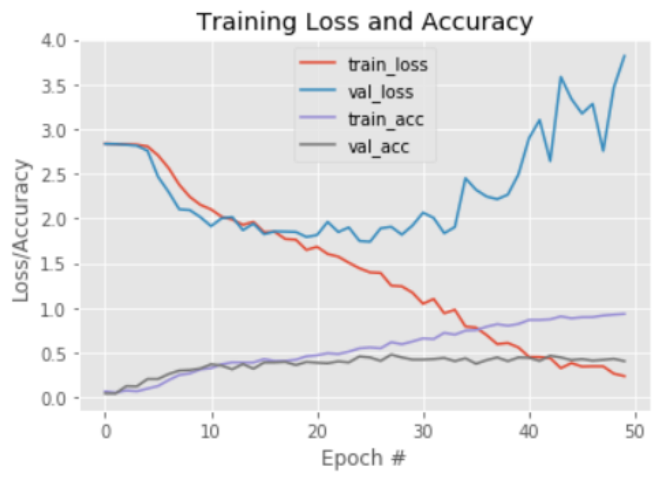
Final Validation accuracy 0.4412 **Final Validation Loss:** 2.9569



The overfitting still exists as per the observations from the plot and the accuracy and loss values. The validation accuracy has reduced even more along with the training accuracy. Another observation is that the previous models were seen to be overfitting after 3 or4 epochs, but in this model, it starts a little later at around 17 epochs. This indicates that the model started well, however still overfits as it learns.

CNN Network 4

A spatial dropout (dropout probability = 0.5) layer after each max pooling layer was added to the previous network. Epochs = 50.



Final Training accuracy: 0.9343 **Final Training Loss:** 0. 2248

Final Validation accuracy 0.4529 **Final Validation Loss:** 3.5031

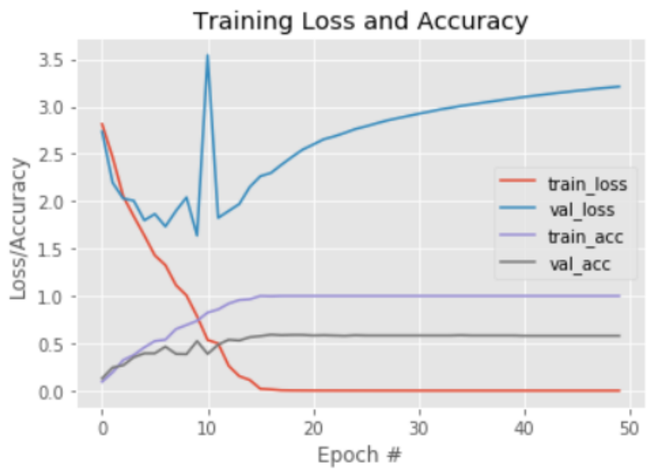
The dropout has not added much value to the overall model accuracy but is ultimately an overfitting model.

Comparison of the network performances

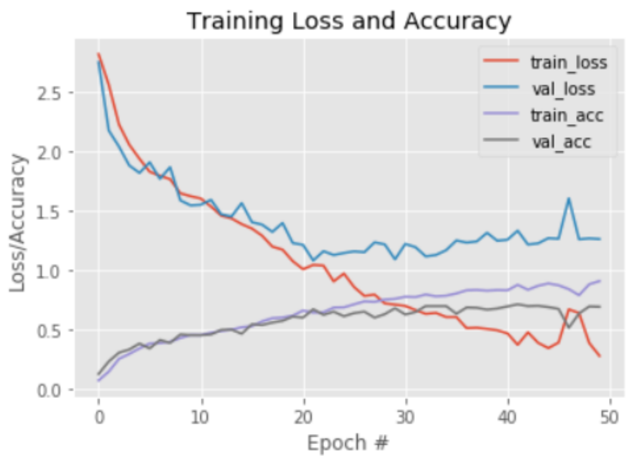
The baseline and the CNN networks 1, 2 and 3 were trained for 25 epochs and all the models resulted to be very poor performers. The deeper the network gets, the more it overfits. However, including the spatial dropout layers has an impact on delaying the overfitting, but still results in a poor performing model.

Applying Data Augmentation to CNN Network 2

Data Augmentation was applied to the CNN Network 2 without including the Spatial Dropout layers, to study its individual impact on the model. Epochs = 50.



Without Data Augmentation



With Data Augmentation

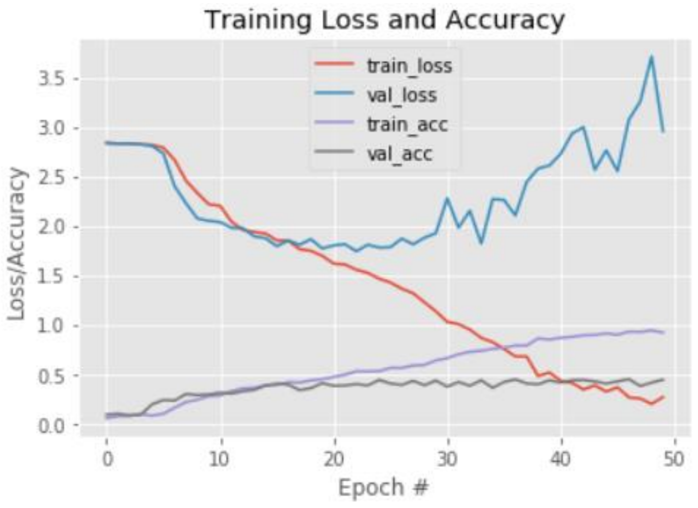
Final Training accuracy: 0.9069 **Final Training Loss** 0.2787

Final Validation accuracy 0.6912 **Final Validation Loss:** 1.2600

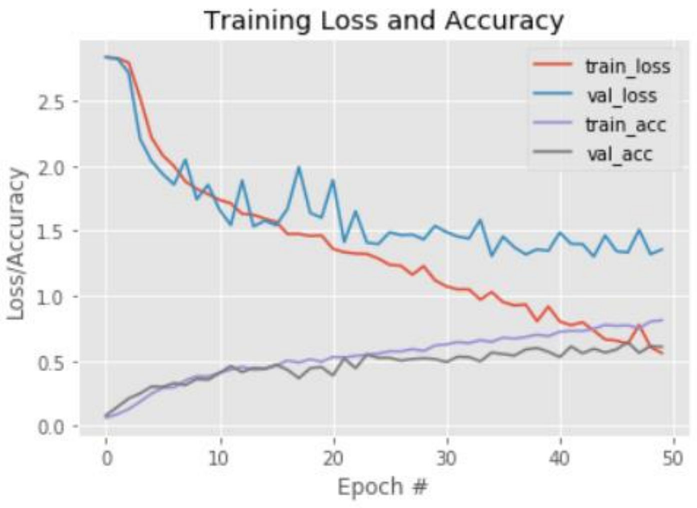
With Data augmentation, a significant improvement is observed in terms of overfitting. The validation loss value has dropped from 3.2 to 1.2 and the validation accuracy has also increased from 57 to 69. Although the module is still slightly overfitting, it only starts after 22 epochs unlike the previous model.

Applying Data Augmentation to CNN Network 3

Data Augmentation was applied to the CNN Network 3 without including the Spatial Dropout layers, to study its individual impact on the model. Epochs = 50.



Without Data Augmentation



With Data Augmentation

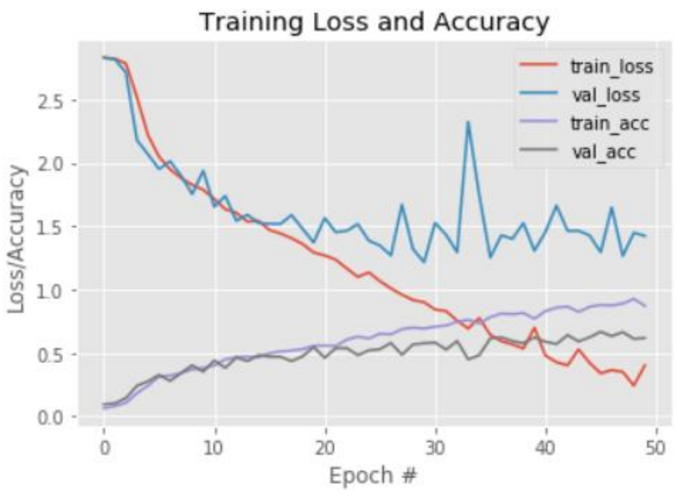
Final Training accuracy 0.8127 **Final Training Loss** 0.5629

Final Validation accuracy 0.6088 **Final Validation Loss:** 1.3567

This model again shows the impact of data augmentation as it has reduced the overfitting to a great extent. The training and validation loss values are much closer, however still experiencing some overfitting after 22 epochs, but better than the version without augmentation.

The results from data augmentation shows that it adds a lot of value when training models with small dataset as it helps to generate more data by producing images that are very close to the original images. The selection of methods such as zooming, flipping, cropping, rotations adds to the generation of new images from the augmentation process in this model.

The impact of removing one of the selection methods (zooming) was studied and the results are given below.



Final Training accuracy 0.8716 **Final Training Loss** 0.4015

Final Validation accuracy 0.6176 **Final Validation Loss:** 1.4241

It is clearly observed that removing a selection method such as zooming has an impact in the model performance. When comparing with the previous results, this model is showing an increased overfitting pattern as the gap between the training and validation loss values is larger.

Part A - Task 2

Ensemble Methdology:

When multiple neural networks are combined to jointly solve a problem, it refers to an ensemble neural network model. Initially, we build a basic neural network model and use it as an ensemble network by training the model repeatedly for 5 times. Here, the variability is achieved only through the random initialization of model weights. A more complex ensemble is built based on the following study.

The work done by Kumar, Ashnil, et al. [1] shows an ensemble network of 5 different architectures, where three networks act as feature extractors and the remaining as actual classifiers generating SoftMax probabilities. In this task, a similar ensemble model is implemented with a few variations in the chosen pre-trained networks and in the feature extraction process.

For our task, the 5 different CNN model architectures that will be used for prediction of the classes in flowers dataset are given below.

1) "A one-vs-one multi-class SVM" [1] trained using the features extracted from pre-trained Xception model
2) "A one-vs-one multi-class SVM" [1] trained using the features extracted from pre-trained VGG16 model
3) Fine-tuned Xception using SoftMax classifier
4) Fine-tuned VGG16 using SoftMax classifier

5) "A one-vs-one multi-class SVM" trained using the features extracted from pre-trained Xception and VGG16 models, where the features from both the models are stacked together and used for the SVM based classification.

Source of variability

A stand-alone CNN model has offered increased flexibility in training models by learning complex nonlinear relationships in the given data [2], however due to the stochastic training nature, these modes can be very sensitive to the specifics of the training data. This results in different set of predictions with different set of weights every time the model is trained. This creates a high variance in the model predictions.

To reduce the variance in predictions and still get accurate results, a source of variability or diversity is induced into the input data or the different parameters of the model or by using different model architectures. This helps in generating results that has less variance and also improves the overall generalization performance combined from all the ensemble models.

In this task, the variability is achieved by varying the model architectures constituting the ensemble model. In addition to this, a couple of architectures use data augmentation that increases the generalization of the base learner and the overall performance of the ensemble.

A Basic Ensemble

A baseline ensemble with a simple network model was implemented, by executing the model 5 times with different initialized weights. The results from the repeated executions of the model is given below.

Model 1 accuracy on Test Set 0.5323529411764706
Model 2 accuracy on Test Set 0.5058823529411764
Model 3 accuracy on Test Set 0.5058823529411764
Model 4 accuracy on Test Set 0.47058823529411764
Model 5 accuracy on Test Set 0.5676470588235294

The overall Ensemble network accuracy was reported to be 0.59

From the above results it can be observed that while repeating executions just by different initialized weights, has caused a lot of variance in the individual base learner accuracies. However, the overall e nsemble network accuracy seems to be higher than all the base learner accuracies, which shows that an ensemble network can perform better than standalone CNN models.

A more complex Ensemble

As discussed in the "Ensemble Methodology" section, 5 different models with varying architectures and additional techniques were implemented. A checkpointing methodology was used in all the models, where the model weights generating the lowest validation loss was used for the prediction of the test set. The model description and results from all these base learners and the ensemble is given below.

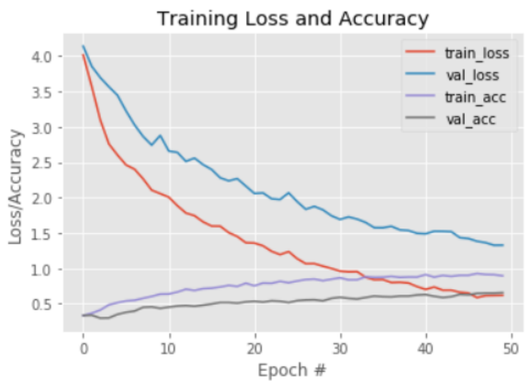
1) <u>Model 1: Xception as Feature extractor with one-vs-one SVM classifier</u>
All the top layers of the Xception model was used as a feature extractor. This set of features obtained were fed into SVM classifier which performs a one-vs-one classification for the multi class problem.

Model 1 Accuracy on Test Set: 0.7647058823529411

2) <u>Model 2: VGG16 as Feature extractor with one-vs-one SVM classifier</u>
All the top layers of the VGG16 model was used as a feature extractor. This set of features obtained were fed into SVM classifier which performs a one-vs-one classification for the multi class problem.

Model 2 Accuracy on Test Set: 0. 8735294117647059

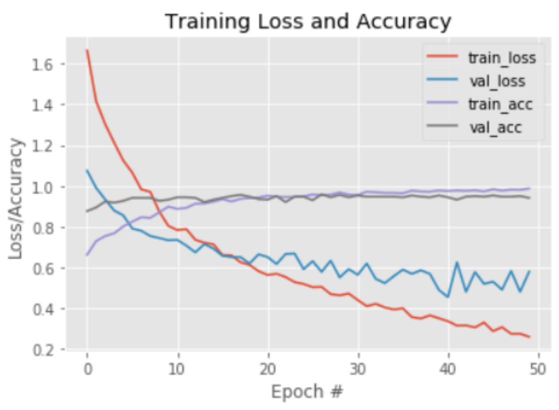
3) <u>Model 3: Fine-tuned Xception with SoftMax classifier</u>
The Xception model was trained in two phases to fine tune certain pre-trained convolutional layers from the model. Data augmentation was used in order to overcome the challenges due the dataset size and to prevent overfitting. The model was trained for 50 epochs.



Model 3 Accuracy on Test Set: 0.65

The model still overfits slightly, although dropout and regularization has been performed.

4) <u>Model 4: Fine-tuned VGG16 with SoftMax classifier</u>
The VGG16 model was trained in two phases to fine tune certain pre-trained convolutional layers from the model. Data augmentation was used in order to overcome the challenges due the dataset size and to prevent overfitting. The model was trained for 50 epochs.



Model 4 Accuracy on Test Set: 0.9441176470588235

The above model overfits slightly. However, the checkpointing method, enables us to use the model weights from the epoch that generated the lowest validation loss.

5) Model 5: Xception and VGG16 as Feature extractor with one-vs-one SVM classifier
The final model is a combination of Models 1 and 2. The top layers of the Xception model and VGG16 was used as a feature extractor separately. Both the set of features were then stacked together and provided as input to the SVM classifier which performs a one-vs-one classification for the multi class problem.

Model 5 Accuracy on Test Set: 0.8117647058823529

The overall ensemble accuracy was reported to be 0.95.

From the above results, all the base learners have shown different levels of performance ranging bet ween .65 to .94. The fine-tuned VGG model shows the highest accuracy, whereas the fine-tuned Xce ption model shows the lowest accuracy. Although so much of variance is observed between the resul ts from the base learners, the final ensemble accuracy is reported to be 0.95.
By averaging all the generated probabilities for each class, the ensemble network, combines and add s a bias to the most closer target class that was predicted and hence produces more accurate and be tter results than the stand-alone models. Both the basic ensemble model and the complex ensemble model has shown that the accuracy generated by an ensemble model is better than the ones genera ted from individual models.

Part B - Task 1

Transfer Learning as Feature Extraction

Transfer learning has been a very popular approach in recent times and there are many pre-trained models that were built with the ImageNet dataset as part of the Large Scale Visual Recognition Challenge (ILSVRC).

Rationale for choosing the variants

Variant1 – Choosing pre-trained model

Out of all the pre-trained networks the InceptionV3, VGG16 and VGG19 have been some of the most popular networks that have outperformed a lot of baseline models. However, these networks are very complex models in terms of the depth of layers and the computational power required to implement these models.

On the other hand, some smaller networks like MobileNet and AlexNet have also been able to perform well in many image classification tasks and is much less complex than the previous models.

In this task, we will consider implementing the feature extraction process with one large pre-trained network which is VGG and one small pre-trained network which MobileNet.

Variant2 – Choosing output layer for features

The pre-trained networks are built with multiple layers before the fully connected layer and has differing sets of features from varying layers. The initial set of layers focus on extracting the high level features and gets to more specific features in the lower levels. levels. We will study the impact of extracting the features from the entire set of layers and from a partial set of it, by cutting a few layers at the end and extracting the features from the remaining layers.

Variant3 – Choosing secondary classifier

The extracted features will be fed into standard machine le   arning algorithms such as KNN, Decision Tree, SVM, Random Forest and Logistic Regression etc. As suggested in [4], there are many popular machine learning algorithms out of which for the current task we will experiment with Logistic Regression, SVM and Random Forest classifier.

Implementation

Extracting features from all the top layers of VGG16. We will not be including these top layers for training as it would solely be used to extract the features from the data when pushed through it once.

S.no	Model Variants			Resulting Test Accuracy
	Pretrained model	Layers used for feature extraction	Secondary classifier	
1	VGG16	All top layers	Logistic Regression	.88
2	VGG16	All top layers	Random Forest	.84

3	VGG16	All top layers	SVM	.84
4	VGG16	Until block4_conv3	Logistic Regression	.86
5	VGG16	Until block3_conv3	Logistic Regression	.76
6	**MobileNet**	**All top layers**	**Logistic Regression**	**.94**
7	MobileNet	All top layers	Random Forest	.93
8	MobileNet	All top layers	SVM	.93
9	MobileNet	Until conv_dw_12	Logistic Regression	.92
10	MobileNet	Until conv_dw_11	Logistic Regression	.90

From the overall results, it can be observed MobileNet has a better performance than the VGG16 model with all the similar variants. With a smaller network and less computation time, MobileNet was able to achieve better results in generating feature rich data.

With respect to the performance of the secondary classifiers, Logistic Regression has outperformed the other classifiers with both VGG and MobileNet. However, SVM and Random Forest resulted in very similar accuracies throughout the experiments.

The feature extraction implemented by cutting the bottom most layer (experiments 4 and 9) hasn't shown much improvement over the performance with all the layers unchanged. Similarly, the experiments 5 and 10 has a decreased test accuracy when compared to the other variants. For this task, the results clearly that as the levels are cut off from the bottom, the features extracted become less representative of the data and hence results in lesser performing models.

The best model with respect to transfer learning + feature extraction, was a combination of MobileNet with all the top layers and Logistic regression (experiment 6). It reported an accuracy of 0.94.

Part B – Task 2

Transfer Learning with Fine tuning

With fine tuning, a pre-trained model is used with a SoftMax classifier and is first trained using all the top layers unchanged for a certain number of epochs. All the top layers are not trainable in this initial phase and we use the fixed ImageNet weights for these layers. We will only tune the fully connected layer that is designed to suit the current problem.

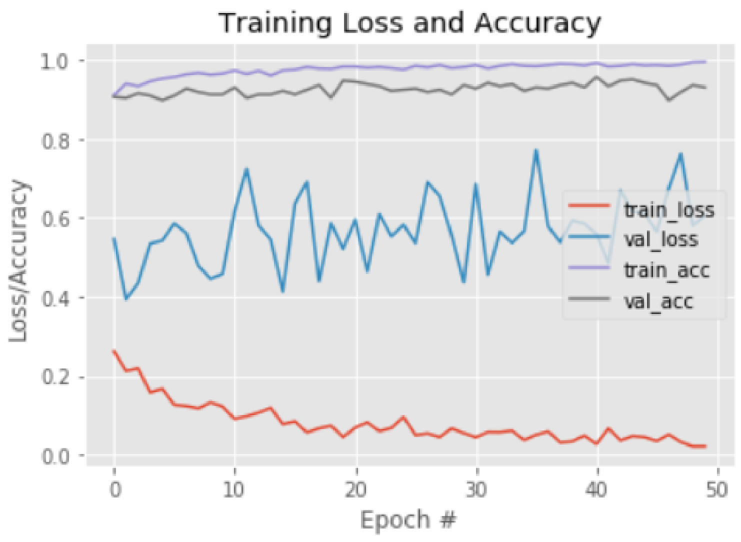
In the next phase we take the trained model from the previous phase and retrain it with a much lesser learning rate and by setting a few layers before the fully connected layer to be trainable. This is done by changing the trainable flag for the required layers to 'True'. The weights associated with these layers and the fully connected layers will be tuned through the same number of epochs. The final accuracy from the second phase model is used to asses the performance of the model.

From the previous task, MobileNet and VGG seem to be well suited models for the given data, and hence we will try to fine tune these models to achieve a better accuracy. As data augmentation has a significant impact in training the data as shown in Task 1, we will be using it across all the experiments in the current task. The selection methods and the parameter values for data augmentation will be similar to Task1. The different experiments and results obtained is shown below.

All the code for the below experiments can be referred in the associated python notebook using the same experiment name.

Exp 1 - VGG16 model

An initial model with a Phase A learning rate of 0.001 and a Phase B learning rate of 1e-5 is implemented for 50 epochs. In Phase A, the trainable flag for al the top layers are set False, and in phase 2 the trainable flag for all the layers from **'block4_conv1'** is set to trainable. The model includes 1ReLu layer and 1 SoftMax layer.

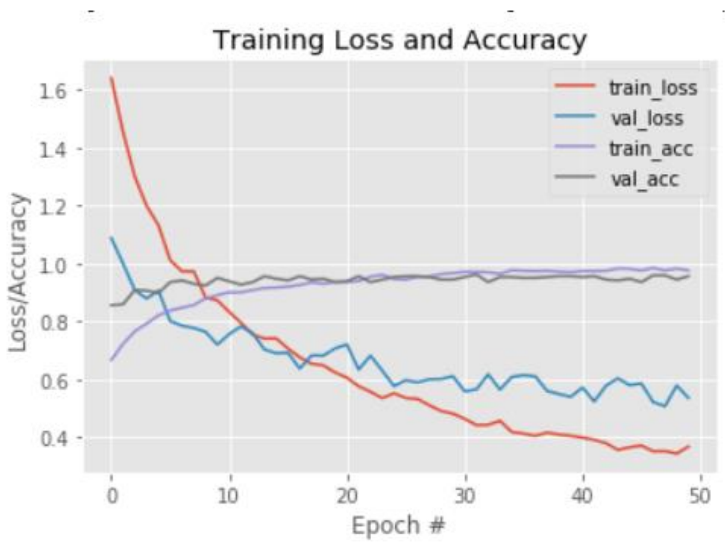


Final Training accuracy	**Final Validation accuracy**	**Final Training Loss**	**Final Validation Loss**
0.9941	0.9294	0.0220	0.6061

Although the model includes data augmentation, we still see an overfitting pattern with the results.

Exp 2 - Adding Regularization to Exp 1

To overcome the overfitting, we will apply **L2 Regularization(0.05)** and include the checkpointing function to evaluate the model for the weights that produce the lowest validation loss. We will also include a learning rate schedular to update the learning rate after every epoch.

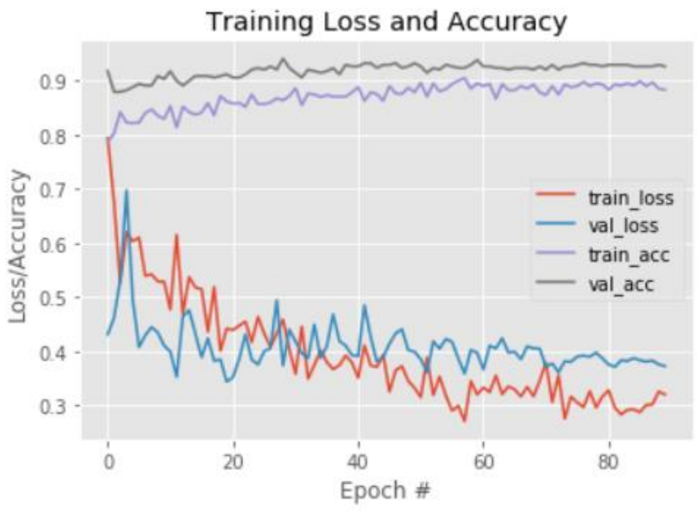


Best Validation accuracy after checkpointing	Best Validation loss after checkpointing
0.9588235	0.5068482959971709

By using regularization with checkpointing, the accuracy for the test set umps up from .92 to .96. Alt hough the model is starting to overfit after 25 epochs, the checkpointing feature allows us to record an accuracy that is obtained with the lowest validation loss at 48 epochs.

Exp 3 - VGG16 with additional ReLu layer

Building over the previous initial model without regularization, an additional ReLu layer is included. T he checkpointing is still used. The model was trained for 90 epochs.

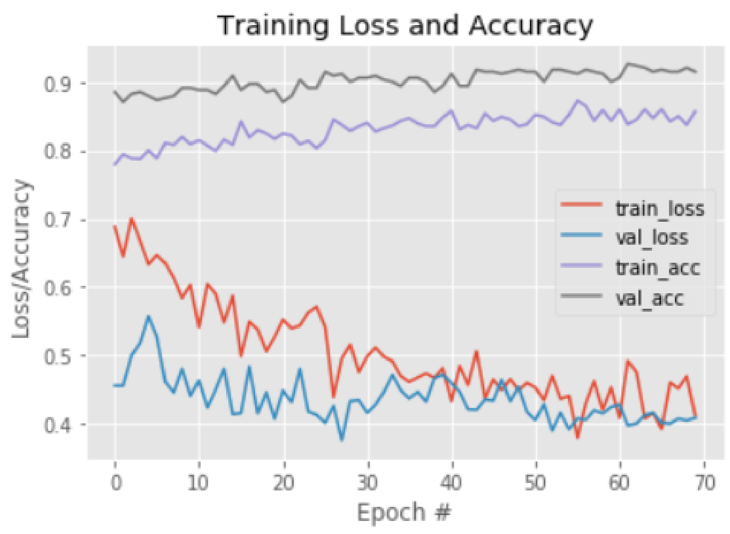


Best Validation accuracy after checkpointing	Best Validation loss after checkpointing
0.9117647	0.34314760951434864

Even without regularization, the model doesn't overfit much. But, by evaluating on the model with l owest reported loss, the accuracy is at .91 after 90 epochs.

Exp 4 - VGG16 with modified unfreezed pre-trained layers

Building over the previous model with 2 ReLu layers and without regularization , the model in phase B was trained by unfreezing the weights from the layer '**block5_conv1**'. The mo del was trained for 70 epochs.



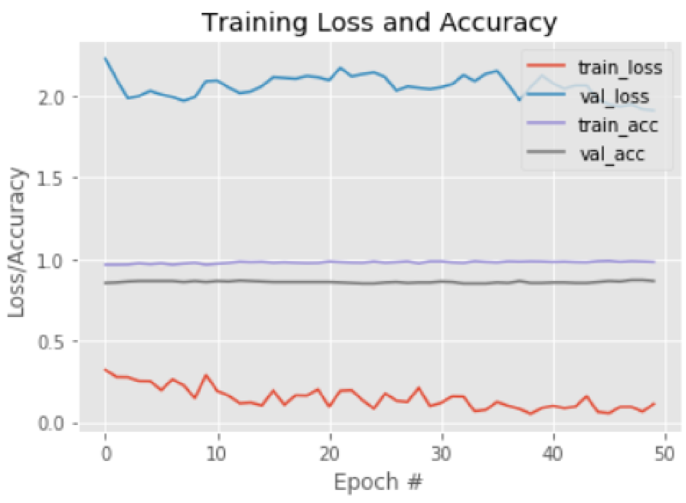
Best Validation accuracy after checkpointing	**Best Validation loss after checkpointing**
0.9117647	0.37601236140026767

The reported accuracy of .91 is similar to the previous model, but it doesn't overfit.

Exp 5 - MobileNet model

A similar set of experiments were conducted with MobileNet to see if it improved the performance.

An initial model with a Phase A learning rate of 0.001 and a Phase B learning rate of 1e-5 is implemented for 50 epochs. In Phase A, the trainable flag for all the top layers are set False, and in phase 2 the trainable flag for all the layers from '**conv_dw_12'** is set to trainable. The model includes 1ReLu layer and 1 SoftMax layer.

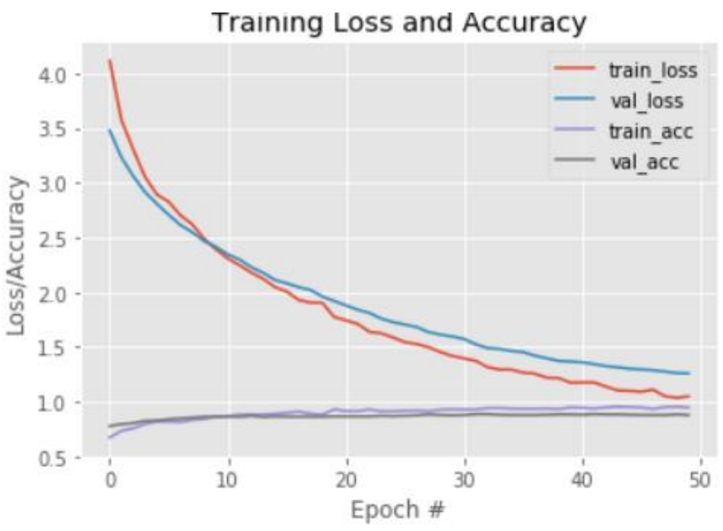


Final Training accuracy	Final Validation accuracy	Final Training Loss	Final Validation Loss
0.9814	0.8647	0.1119	1.9088

Although the model includes data augmentation, we see a significant overfitting pattern with the results.

Exp 6 – Adding Regularization to Exp 5

To overcome the overfitting, we will apply **L2 Regularization (0.07)** and include the checkpointing function to evaluate the model for the weights that produce the lowest validation loss. We will also include a learning rate schedular to update the learning rate after every epoch.

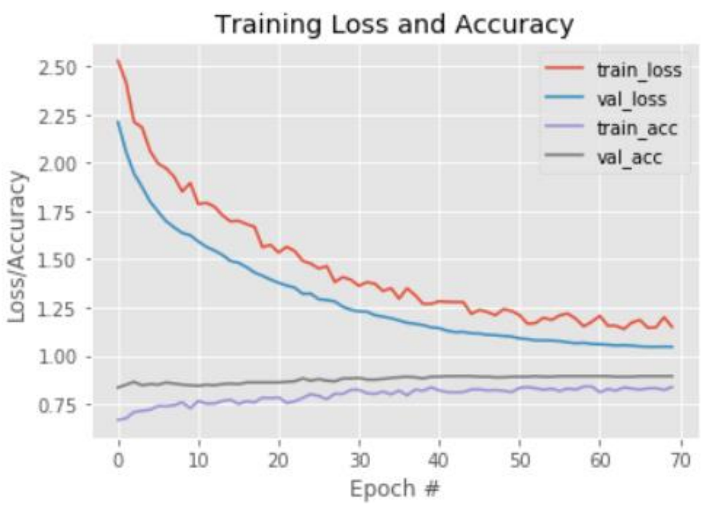


Best Validation accuracy after checkpointing	Best Validation loss after checkpointing
0.87941176	1.259915688458611

With L2 Regularization the validation accuracy jumps from .86 to .90. The model very slightly overfits after 20 epochs.

Exp 7 - MobileNet with additional ReLu layer

Building over the previous initial model with L2 regularization (0.07), an additional ReLu layer is inclu ded. The checkpointing is still used. The model was trained for 90 epochs.

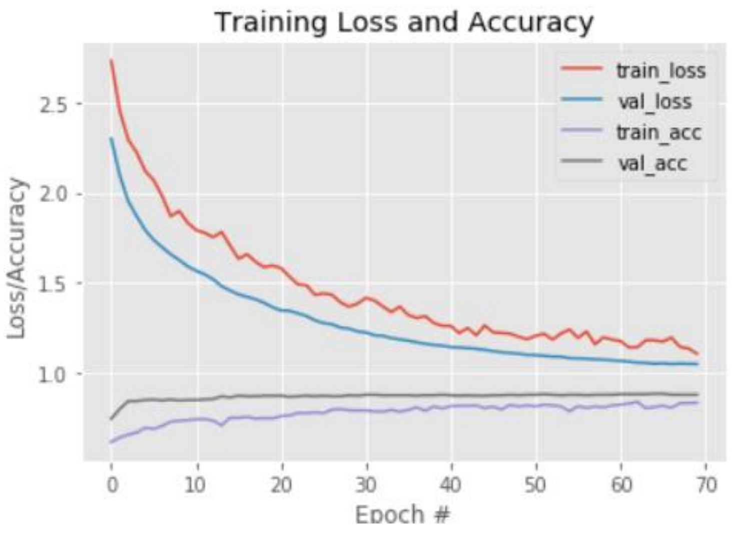


Best Validation accuracy after checkpointing	**Best Validation loss after checkpointing**
0.89411765	1.046117530149572

The model takes a very long time to converge with L2 regularization. It doesn’t converge even after 70 epochs and the accuracy is reported to be .89.

Exp 8 - MobileNet with modified unfreezed pre-trained layers

Building over the previous model with 2 ReLu layers and with L2 regularization (0.07)
, the model in phase B was trained by unfreezing the weights from the layer ‘**conv_dw_12**. The mod el was trained for 70 epochs.



Best Validation accuracy after checkpointing	**Best Validation loss after checkpointing**
0.87941176	1.049803395832286

A similar result as the previous model, is observed where it takes a long time to converge. The accuracy is even lesser. Fine tuning from ‘**conv_dw_11**hasn’t added much value to the model.

Comparison of all the fine-tuning models

Comparing all the experiments for fine-tuning, VGG16 outperformed MobileNet in all the similar experiments.

The highest accuracy was 0.96, reported in Exp -2 for the VGG16 model (1 ReLu layer+ L2 regularization-0.05+trained from layer ‘block4_conv1’ in Phase B)

Part C

Capsule Networks

Why is Capsule Networks gaining importance?

Convolutional neural networks (CNNs) has seen a lot of incremental developments over the years and has hugely contributed to the success in the field of deep learning. They have an extensive learning capacity and has been a very suitable method for a lot of image classification problems. However, they have their limits and a few fundamental drawbacks. CNNs recognize an object or pattern in an image by detecting its features as a group of pixel values. This conveys that a mere existence of an object in an image is a very strong indicator of its presence in the image and this information is used to make the final predictions from the network model. This is where the flaw lies in the system as this method solely cares about the presence of an object and ignores the spatial relations and the orientation of the object in an image. A more intuitive visualization of the problem is depicted below. From the below images, a CNN can recognize both the images on the left and right to be a face. However, this is not the expected accurate prediction for the image on the right.

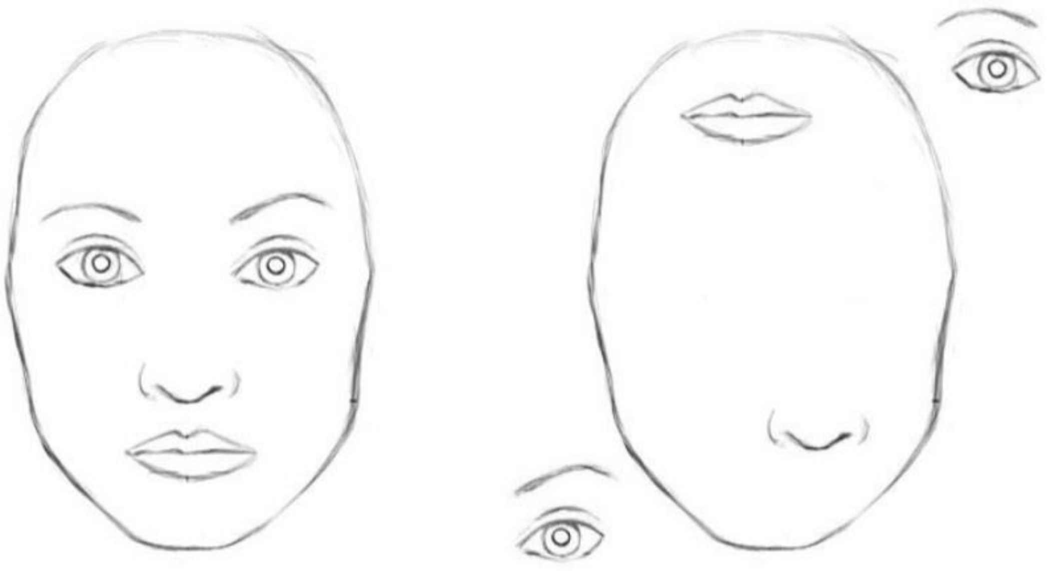


Figure 1 [7]

The CNN architecture fails to map the relationship between the different parts of the face (eyes, nose and mouth) in the prediction process. The most important constituent to this problem is in the usage of Max Pooling layers. A lot of valuable spatial information is lost in the process of max pooling layers as only the most active neurons are chosen to move further to the next layer. To overcome this issue, Hinton Et al [1] proposed a process called “routing by agreement” using capsule networks or CapsNet.

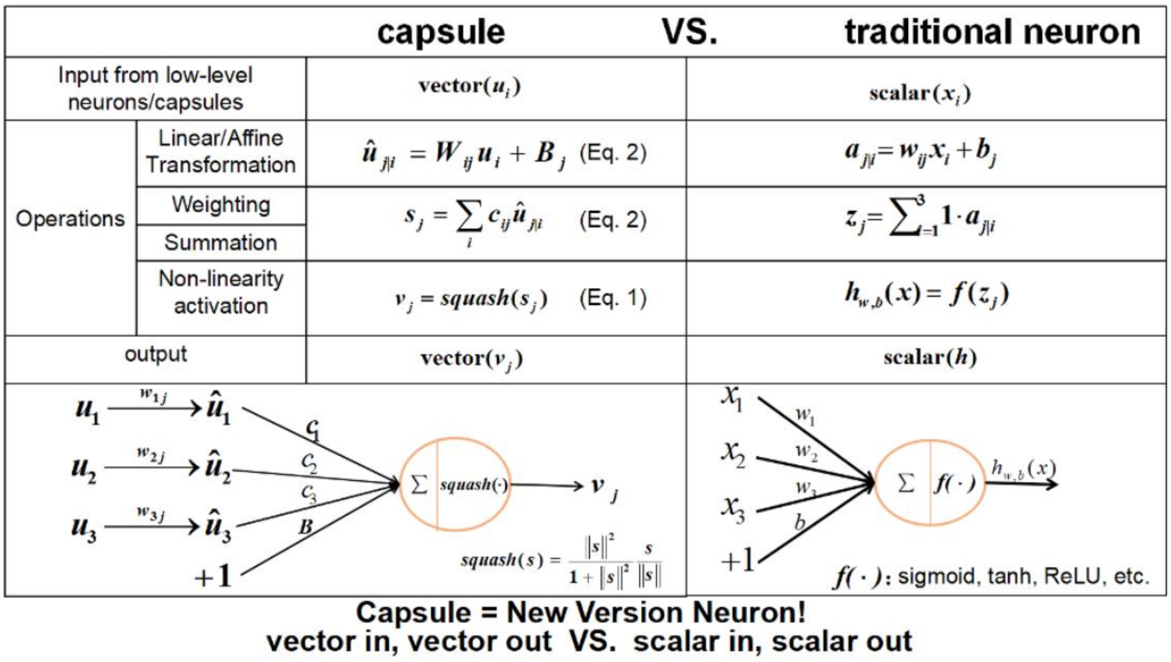
What are Capsule Networks and what is its high-level architecture?

A CapsNet is made up of capsules which are a group of neurons that learns to recognize an object with some sort of geometric information that is related to the position and orientation of the object in the image. The output from the capsules is a vector, with its length representing the probability that an object is present and its orientation (for example, in 8D space) representing the object's pose parameters such as the rotation and the precise position of the object.

Like a regular CNN, the CapsNet architecture has multiple layers of which the lowest layer is referred to as primary capsule layers. The primary capsules receive a small region of the image where it tries to locate the presence and pose of a pattern. Capsules in the higher layers referred to as routing capsules try to recognize larger and complex objects, such as ships and boats. The "routing by agreement" process comes into play between these layers, where the primary capsule layer tries to predict the outcome of the capsules in the routing layers and activates the capsules in these higher layers only if the predictions agree. This iterative process of agreement-detection and routing update has achieved a considerably better performance that the regular CNNs with complex images containing overlapping patters of data.

How does capsules work?

While a single neuron in CNN receives a scalar input, multiplies them by scalar weights and returns a scalar output from the activation function, a single capsule implements vector forms of the above steps. The following figure gives a summary of the operations involved with a capsule and traditional neuron.



		capsule VS.	traditional neuron
Input from low-level neurons/capsules		vector(u_i)	scalar(x_i)
Operations	Linear/Affine Transformation	$\hat{u}_{j\|i} = W_{ij} u_i + B_j$ (Eq. 2)	$a_{j\|i} = w_{ij} x_i + b_j$
	Weighting	$s_j = \sum_i c_{ij} \hat{u}_{j\|i}$ (Eq. 2)	$z_j = \sum_{i=1}^{3} 1 \cdot a_{j\|i}$
	Summation		
	Non-linearity activation	$v_j = squash(s_j)$ (Eq. 1)	$h_{w,b}(x) = f(z_j)$
output		vector(v_j)	scalar(h)

$u_1 \xrightarrow{w_{1j}} \hat{u}_1$
$u_2 \xrightarrow{w_{2j}} \hat{u}_2$
$u_3 \xrightarrow{w_{3j}} \hat{u}_3$
$+1$
c_1
c_2
c_3
B
Σ squash(·)
v_j
$squash(s) = \frac{\|s\|^2}{1+\|s\|^2}\frac{s}{\|s\|}$
x_1
x_2
x_3
$+1$
w_1
w_2
w_3
b
Σ $f(\cdot)$
$h_{w,b}(x)$
$f(\cdot)$: sigmoid, tanh, ReLU, etc.

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

Figure 2 [11]

From the left-hand side of the image, we can see that there are 4 different operations that happen between receiving the input and producing output of a capsule.

Step1: At first, the input received is of vector form from 3 of the previous capsules. The lengths of these vector represent the probabilities that the intended object is detected by the lower level capsules. The direction of these vectors represents the object's pose parameters. These are multiplied by weight matrices that represent the spatial relationship between lower level features such as nose, eyes etc. and higher-level features such as the face. For example, W1j would be to associate the relationship between the eyes and the face, W2j for the relationship between mouth and the face and so on.

Step2: Multiplying these weight vectors, results in a prediction that helps detecting the position of the higher-level features. For example, u1hat represents the prediction of the position the face could be detected with respect to the eyes and u2hat predicts the position of the face with respect to the mouth. These predictions are multiplied with a scalar that will help decide to which higher level capsule should the output be routed to. Unlike the scalar weights that are learnt using backpropagation, the scalar in the capsule networks is based on the dynamic routing algorithm as proposed by Hinton which involves the iterative process of agreement-detection and routing update process as mentioned before.

Step3: Once this is done, the sum of the vectors is calculated.

Step4: Finally, a novel squashing function that takes in a vector and squashes it to have a length between 0 and 1, leaving the orientation unchanged. This indicates the output length of the vector is a probability again of a feature being detected by the capsule.

Based on this iterative process, the capsules learn to associate the lower level features with respect to the higher-level features, by maintain the spatial relationships between the objects in an image.

PART A & B References

[1] Kumar, Ashnil, et al. "An ensemble of fine-tuned convolutional neural networks for medical image classification." *IEEE journal of biomedical and health informatics* 21.1 (2016): 31-40.

[2] https://machinelearningmastery.com/ensemble-methods-for-deep-learning-neural-networks/

[3] Nagahamulla, Harshani & Ratnayake, Uditha & Ratnaweera, Asanga. (2014). Selecting Most Suitable Members for Neural Network Ensemble Rainfall Forecasting Model. 10.1007/978-3-319-07692-8_56.

[4] https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/

PART C References

[5] Sabour, Sara, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic routing between capsules." *Advances in neural information processing systems*. 2017.

[6] P. Afshar, A. Mohammadi and K. N. Plataniotis, "Brain Tumor Type Classification via Capsule Networks," *2018 25th IEEE International Conference on Image Processing (ICIP)*, Athens, 2018, pp. 3129-3133, doi: 10.1109/ICIP.2018.8451379.

[7] https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b

[8] https://towardsdatascience.com/capsule-networks-the-new-deep-learning-network-bd917e6818e8

[9] https://towardsdatascience.com/capsule-neural-networks-part-2-what-is-a-capsule-846d5418929f

[10] https://www.oreilly.com/content/introducing-capsule-networks/

[11] https://github.com/naturomics/CapsNet-Tensorflow