Cork Institute of Technology

COMP9067 DEEP LEARNING: ASSIGNMENT 2

PREPARED BY: MARCUS WONG YEW HON

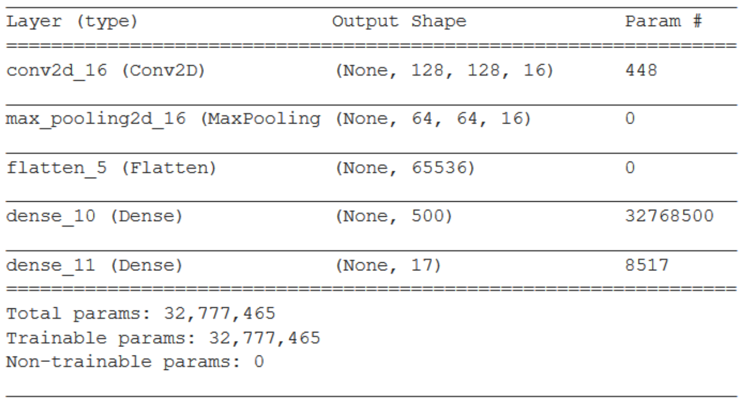
STUDENT ID: R00183595

Table of Contents

Part A: CNNs

Task 1: CNN Implementation

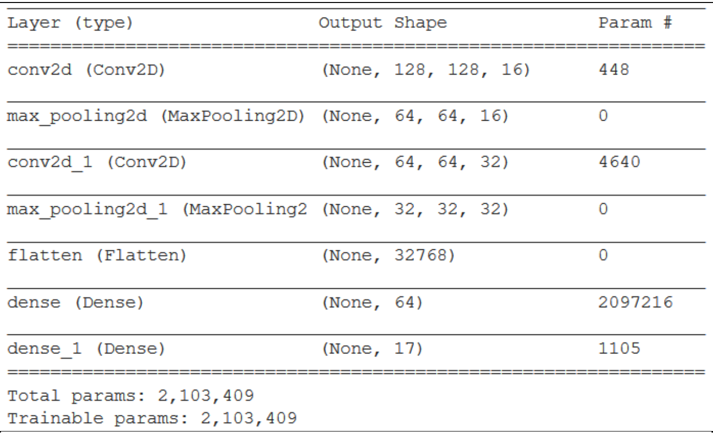
A baseline CNN network was implemented with the following configuration:



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 128, 128, 16)      448
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 64, 64, 16)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 65536)             0
_________________________________________________________________
dense_10 (Dense)             (None, 500)               32768500
_________________________________________________________________
dense_11 (Dense)             (None, 17)                8517
=================================================================
Total params: 32,777,465
Trainable params: 32,777,465
Non-trainable params: 0
_________________________________________________________________
```

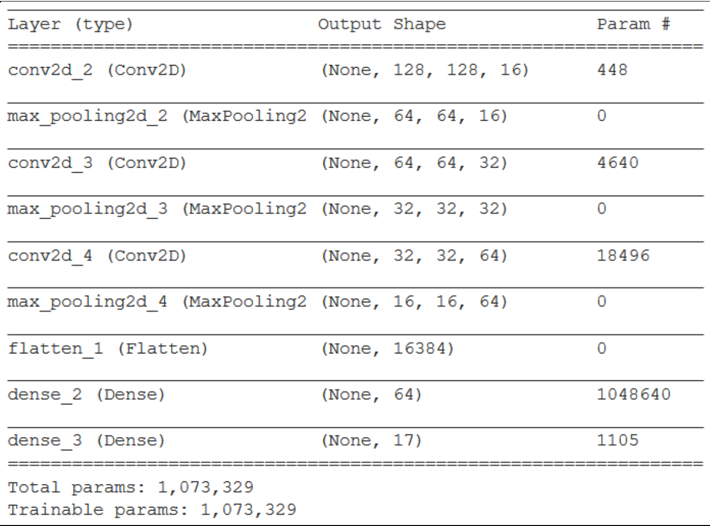
The baseline CNN configuration of one convolutional layer, one pooling layer, a fully connected layer and a SoftMax layer was implemented to serve as a benchmark for the performance of subsequent network configurations with gradually increasing complexity:

Config 1

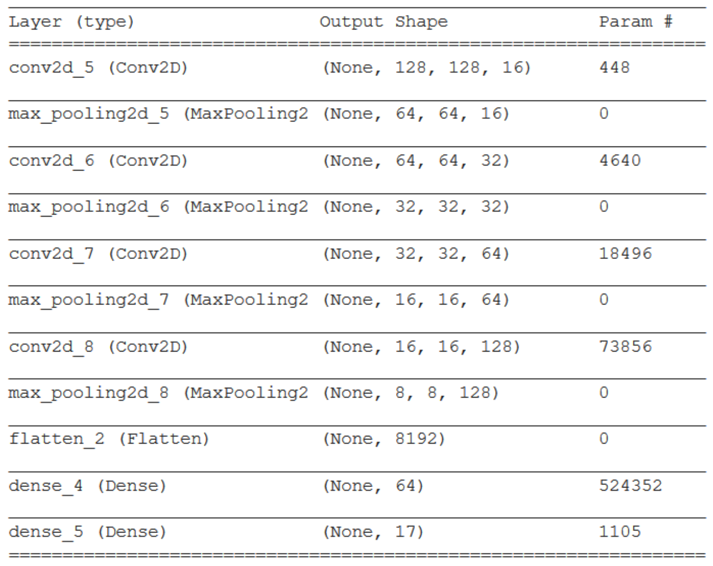


```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 128, 128, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 64, 64, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 64, 64, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 32768)             0
_________________________________________________________________
dense (Dense)                (None, 64)                2097216
_________________________________________________________________
dense_1 (Dense)              (None, 17)                1105
=================================================================
Total params: 2,103,409
Trainable params: 2,103,409
```

Config 2



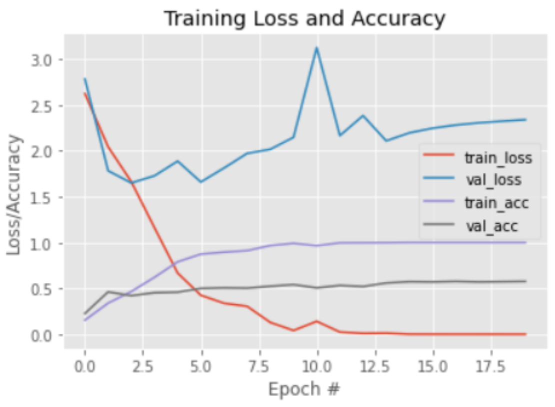
```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 128, 128, 16)      448
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 64, 64, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 64, 32)        4640
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 16384)             0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                1048640
_________________________________________________________________
dense_3 (Dense)              (None, 17)                1105
=================================================================
Total params: 1,073,329
Trainable params: 1,073,329
```

Config 3	

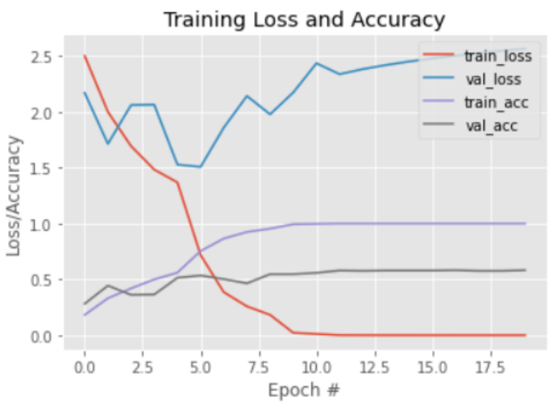
```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 128, 128, 16)      448
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 64, 64, 16)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 64, 64, 32)        4640
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                524352
_________________________________________________________________
dense_5 (Dense)              (None, 17)                1105
=================================================================
```

Performance Comparison for Networks of Different Configurations

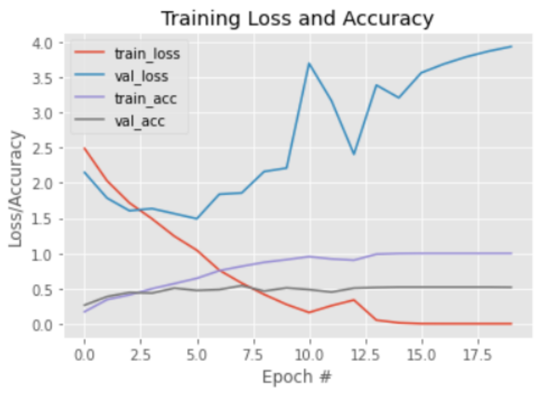
The training loss and accuracy graphs for all network configurations are as below:



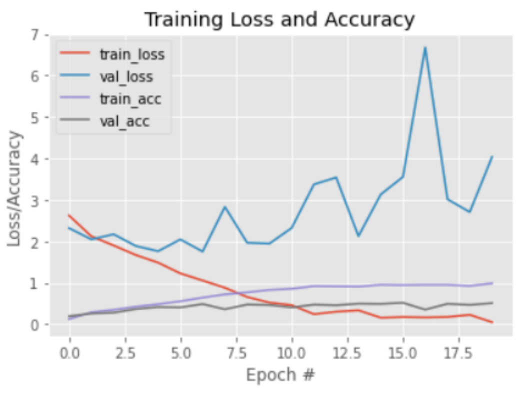
Baseline



Config 1



Config 2



Config 3

Model	Validation Loss	Validation Accuracy (%)
Baseline	2.317	57.65
Config 1	2.537	58.23
Config 2	3.911	51.76
Config 3	4.019	50.88

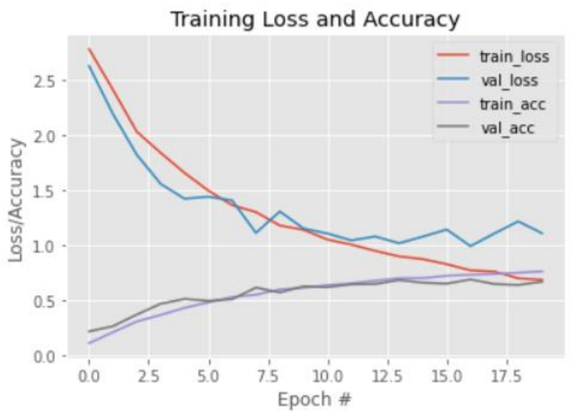
All model configurations were trained for a total of 20 epochs using the stochastic gradient descent optimizer with a learning rate of 0.01.

From the accuracy and loss graphs above, all network configurations (including the baseline configuration) exhibited severe overfitting at approximately the 5th epoch during training. This is evident from the massive gaps between the training loss and the validation loss. Not only were the gaps wide, the validation loss is gradually increasing as well nearing the end of the training sessions. Another sign of overfitting is the 100% training accuracy rate at the end of all configuration's training.

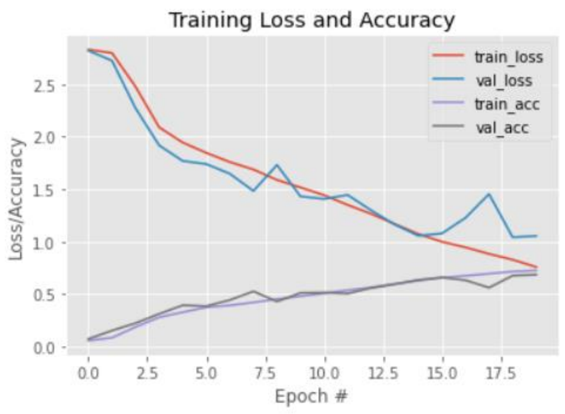
As for performance on the test dataset, *config 1* showed minor improvements in accuracy scores as compared to the baseline configuration, but dipped in loss scores. The subsequent configurations (*config 2, config 3*) reported a significant dip in accuracy scores, down roughly 6% and 7% in accuracy respectively when compared to *config 1*. This phenomenon can be attributed to the significant increase in loss values of *config 2* and *config 3* from *config 1*.

Impacts of Data Augmentation

As stated before, the high loss scores for the two deepest network configurations (*configs 2 and 3*) have caused in the reduction of accuracy rates for the model. Therefore, techniques such as data augmentation can be applied to reduce the overfitting phenomenon in training phase, which leads to improvements in loss and accuracy. This is because the CNN takes the altered input image data returned after data augmentation (through rotation, cropping, zooming etc.) as new unseen data, hence reducing chances for the network to overfit.



Config 2



Config 3

Model	Validation Loss	Validation Accuracy (%)
Config 2	1.108	67.06
Config 3	1.068	68.53

Data augmentation was applied to the training process of *config 2* and *config 3*. From the accuracy and loss graphs above, it can be observed that overfitting for both configurations have been significantly reduced after data augmentation has been implemented. The reduction in overfitting have resulted in significant improvements for both validation loss and accuracy as well, with losses reducing from 3.911 to 1.108 and accuracy improving from 51.76% to 67.06% for *config 2*, and losses reducing from 4.019 to 1.068 and accuracy improving from 50.88% to 68.53% for *config 3.*

Now with the new results after data augmentation, the impact of number of network layers in a CNN can be seen. As *config 3* has more layers than *config 2*, naturally the final validation loss and accuracy will be marginally better than *config 2.*

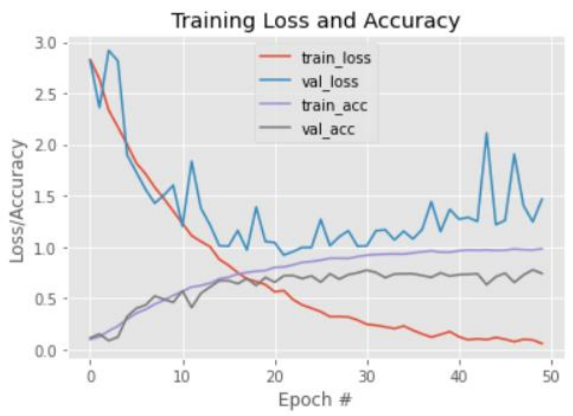
Task 2: Ensembles

For this task, an ensemble was to be implemented using a multitude of base learners. The final ensemble accuracy can be obtained by averaging the predictions of all base learners, then extracting the index of the highest average probability as the final predicted class.

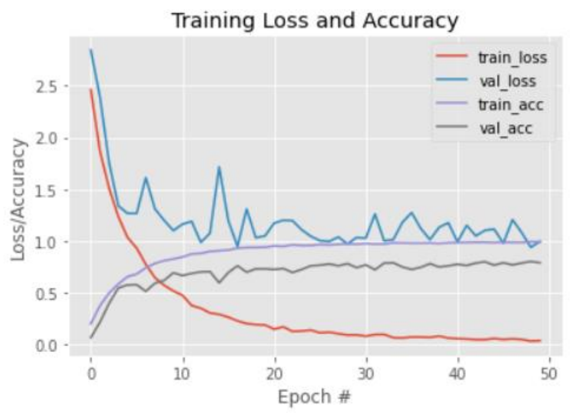
The following 6 base learners were chosen for this task. Each base learner was trained for 50 epochs with a learning rate of 0.01 as well as data augmentation applied:

- AlexNet
- InceptionV3
- LeNet5
- ResNet50
- ShallowVGG
- VGG16

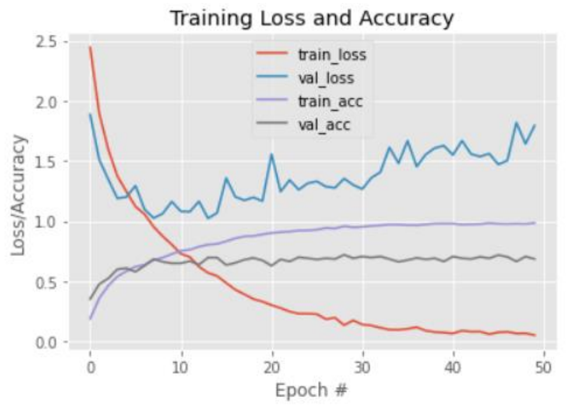
Training loss and accuracy graphs:



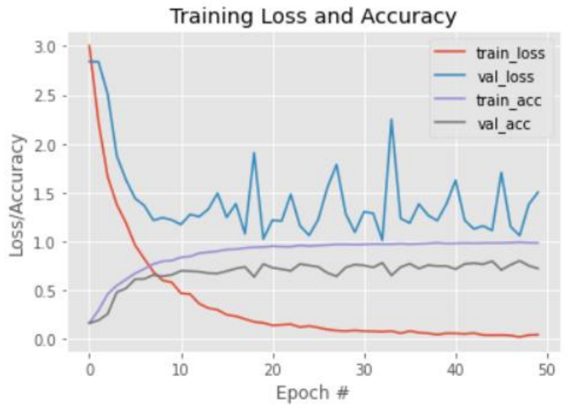
AlexNet



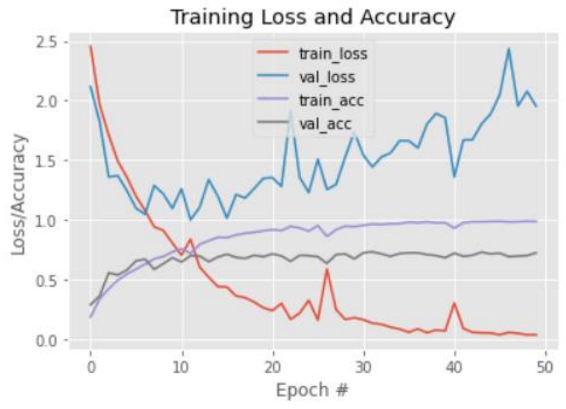
InceptionV3



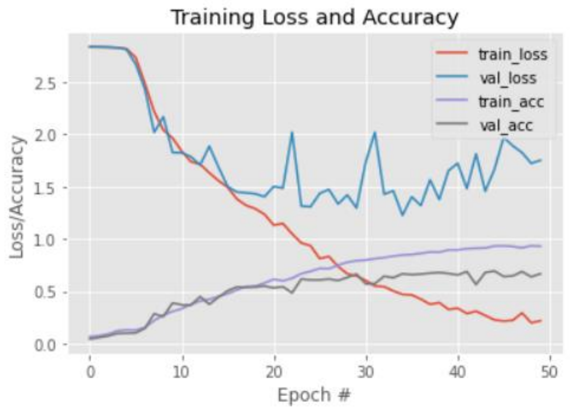
LeNet5



ResNet50



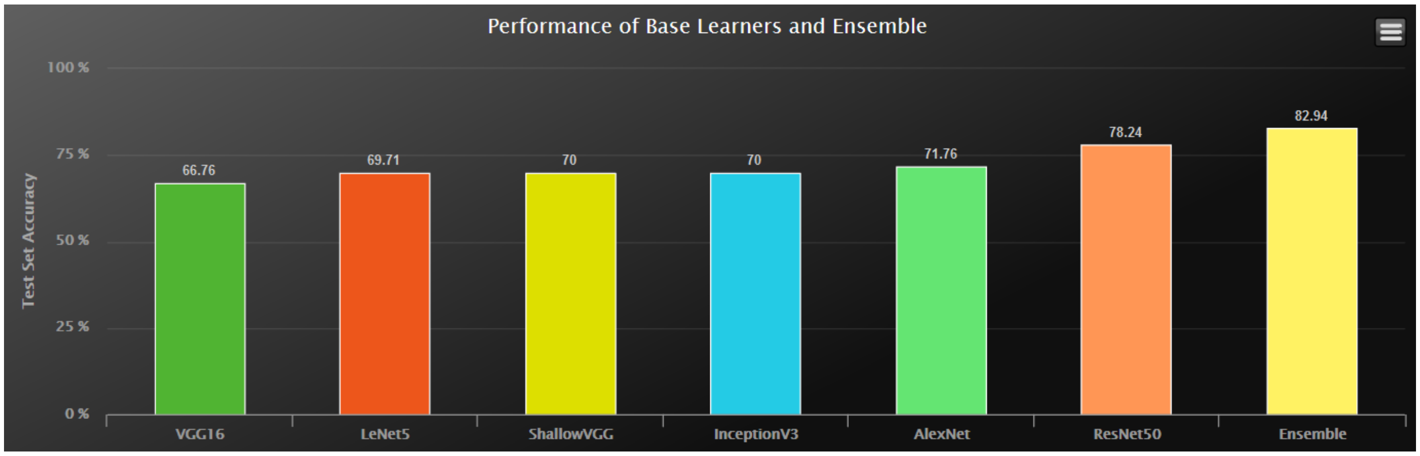
ShallowVGG



VGG16

Performance of base learners and final ensemble:

Base learner	**Best Validation Loss**	**Best Validation Accuracy (%)**
ResNet50	1.0142	78.24
ShallowVGG	0.9997	70.00
VGG16	1.2259	66.76
InceptionV3	0.9379	80.00
LeNet5	1.0232	69.71
AlexNet	0.9203	71.76
	Final Ensemble Accuracy (%)	**82.94**



The best validation loss and accuracy for each base learner was chosen as performance benchmark by monitoring and checkpointing the validation loss during training. The resulting weights are then saved and used for prediction, yielding the accuracy score as tabulated above. Therefore, the overfitting phenomenon exhibited by all models would not affect the final output of the ensemble greatly.

From the bar chart shown, it can be seen that the final ensemble ranks the highest in accuracy as compared to each of the base learner with an accuracy of almost 83% on the validation set. Meanwhile, the best accuracy given by an individual base learner was the InceptionV3 model with an accuracy of 80%, giving the ensemble an almost 3% increase in accuracy rate as compared to the best performing base learner.

Part B: Transfer Learning

Task 1: Feature Extraction

In this task, feature extraction from CNNs was performed by piping in the input image data into a pre-trained InceptionV3 (with dense classification layers removed) to obtain predicted feature data. The predicted features data obtained from InceptionV3 will then be used as new feature data for classification models to predict classes for the test dataset. Multiple classification models will be assessed in this portion of the assignment:

- Random Forest
- Support Vector Machine (SVM)
- K-Nearest Neighbors (KNN)
- Decision Tree

KNN was chosen because the clustering approach taken by KNN might allow for decent classification of images based on a decent set of image features provided by the CNN convolutional layers. Decision Tree was chosen as its decision-making approach of binary choices based on possible image features could make for a decent classification result. Random forest was chosen as it is an ensemble technique, encompassing multiple decision trees in its implementation. Lastly, SVM was chosen as it provides good classification accuracy, does not make biased decisions based on data, which leads to less overfitting.

The prediction accuracy of each classification models will be assessed. The accuracy of each classification model will be compared against the baseline accuracy model, which is the original accuracy scores of the InceptionV3 model obtained from the previous subtask.

Secondary Algorithm	**Test Set Accuracy (%)**
Baseline (InceptionV3)	80.00
K-Nearest Neighbors	64.12
SVM	80.88
Decision Tree	45.88
Random Forest	78.82

As shown in the table above, it appears that the SVM classification method performed the best out of other classification algorithms with an accuracy rate of 80.88%. In fact, it marginally outperformed the baseline InceptionV3 model by 0.88%. This was to be expected of an SVM classifier, where it usually provides great classification accuracies when presented with defining input features.

The decision tree algorithm performed the worst out of all other algorithms with only 46% accuracy. However, the Random Forest algorithm, which is a tree ensemble classifier, yielded a respectable 78.82% accuracy rate. This suggests that the numerous decision trees generated by the random forest classifier is decent enough in classifying images based on given input image features.

To summarize, the extraction of features through CNN and feeding said extracted features to existing classification algorithms are an efficient approach for obtaining decent classification results, as input data would only go through the CNN once for feature outputs, which can then be directly used with classification algorithms to predict classes. This can sometimes be more desirable as opposed to training the CNN multiple times over 10s of epochs just to obtain similar or sometimes lower accuracy scores than some classification algorithm outputs.

Task 2: Fine-tuning

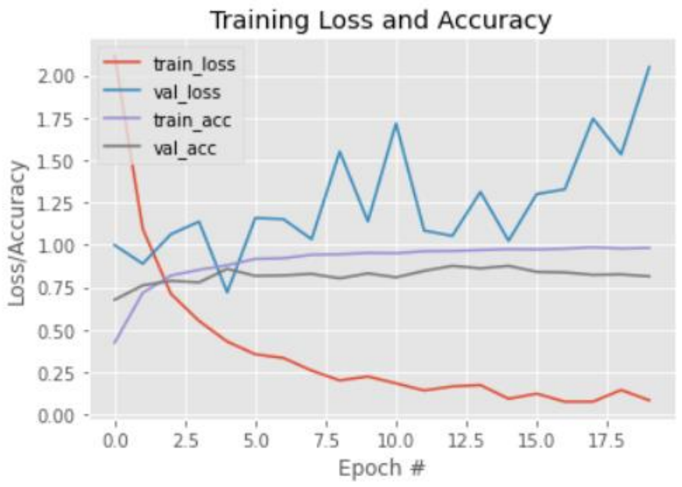
For this task, fine-tuning would be done in an attempt to obtain the best validation accuracy on the Flowers dataset. The fine-tuning process will be performed in 4 phases. Each phase will be trained using the RMSProp optimizer for 20 epochs. Each phase will be broken down in detail as below:

Phase A

In the first phase, a pre-trained VGG16 model will be initialized with its trainable status set as "False" to ensure that the weights would not be modified during training. The VGG16 model will be incorporated into a new model, appending it to a fully connected dense layer and a SoftMax output layer. A dropout layer was introduced as well to reduce overfitting. The model architecture will be illustrated as below:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_5 (Flatten)          (None, 8192)              0
_________________________________________________________________
dropout_5 (Dropout)          (None, 8192)              0
_________________________________________________________________
dense_10 (Dense)             (None, 256)               2097408
_________________________________________________________________
dense_11 (Dense)             (None, 17)                4369
=================================================================
Total params: 16,816,465
Trainable params: 2,101,777
Non-trainable params: 14,714,688
```

The model will then be subjected to training for 20 epochs with an initial learning rate of 0.001. The accuracy/loss graph and results for the training process is as below:



Validation Loss	Validation Accuracy (%)
2.0253	81.47

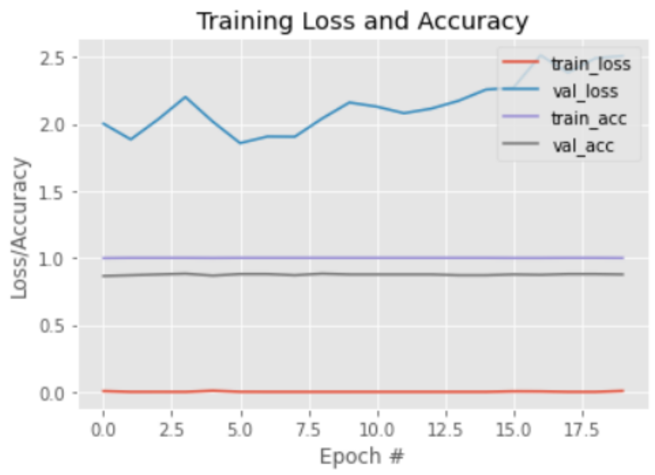
It can be seen that the resulting validation accuracy have been boosted to 81.47%, nearly a 15% increase in accuracy as compared to the base VGG16 model validation accuracy of 66.76% obtained in Part A Task 2. However, overfitting can be seen happening in the training process.

Phase B

After phase A, the fine-tuning process proceeds with unfreezing the last convolutional block layers of the VGG16 model. The layers shown below will be unfrozen for this phase:

```
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
```

The model will then again be subjected to training for 20 epochs. This time, the learning rate will be further lowered to 0.00001, this is to prevent the pre-trained weights from being altered too much too quickly as the existing weights are assumed to be relatively good already. The accuracy/loss graph and results for the training process is as below:

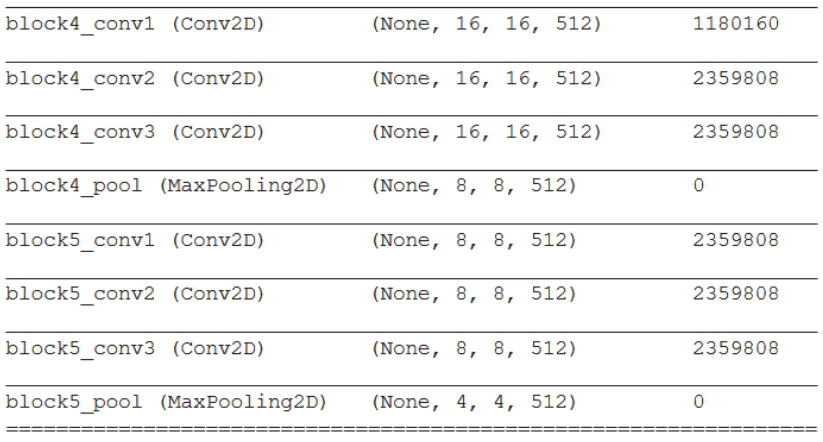


Validation Loss	Validation Accuracy (%)
2.0161	87.06

After training, the validation accuracy has further improved since phase A with an accuracy rate of 87.06%. The validation loss score has improved from phase A as well. However, just like in phase A, overfitting has occurred in the training process.

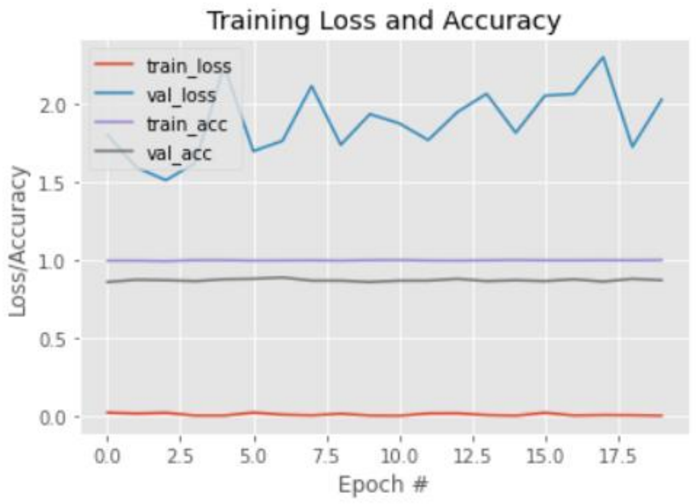
Phase C

Further unfreezing of layers will be performed in phase C. This time, all convolutional layers starting from convolutional block 4 onwards will be unfrozen and trained. The layers that are unfrozen are illustrated below:



```
block4_conv1 (Conv2D)        (None, 16, 16, 512)       1180160

block4_conv2 (Conv2D)        (None, 16, 16, 512)       2359808

block4_conv3 (Conv2D)        (None, 16, 16, 512)       2359808

block4_pool (MaxPooling2D)   (None, 8, 8, 512)         0

block5_conv1 (Conv2D)        (None, 8, 8, 512)         2359808

block5_conv2 (Conv2D)        (None, 8, 8, 512)         2359808

block5_conv3 (Conv2D)        (None, 8, 8, 512)         2359808

block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
```

Once again, the model will be re-trained for 20 epochs and the learning rate will be further reduced to 0.000001. The accuracy/loss graph and results for the training process is as below:

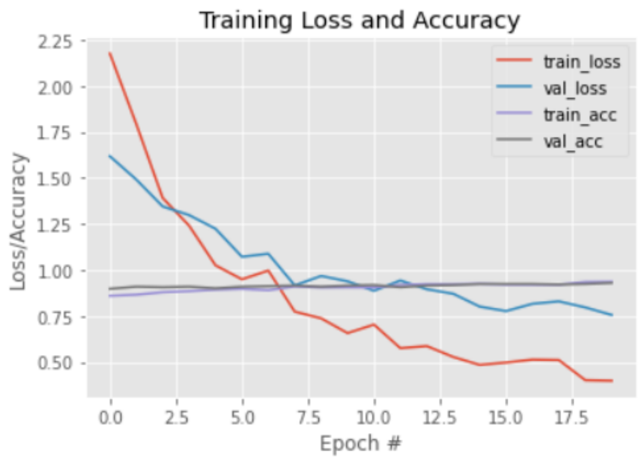


Validation Loss	Validation Accuracy (%)
2.5074	87.65

At this point, the validation accuracy has only increased marginally from 87.06% in phase B to 87.65%. The validation loss on the other hand, have increased from 2.0161 in phase B to 2.5074. This means that further unfreezing of layers and re-training the model will no longer yield significant improvements. Also, seeing that the validation loss has started to increase in this phase, additional techniques must be employed to combat this. Just like previous phases, the overfitting is getting severe.

Phase D

Seeing that performance have started to deteriorate in the previous phase, data augmentation will be implemented to the model in phase C to combat issues of overfitting and increasing validation loss. Rotation, stretching, shearing, zooming and flipping will be used for the data augmentation process. Once again, the model will be trained for 20 epochs with the learning rate remaining at 0.000001. The accuracy/loss graph and results for the training process is as below:



Validation Loss	**Validation Accuracy (%)**
0.7725	92.94

After training with data augmentation, there has been a massive reduction in overfitting. The reduction in overfitting, in turn, improved the validation accuracy to almost 93% on the Flowers dataset. The validation loss has seen huge improvements as well, going from 2.5074 in phase C to 0.7725 in phase D.

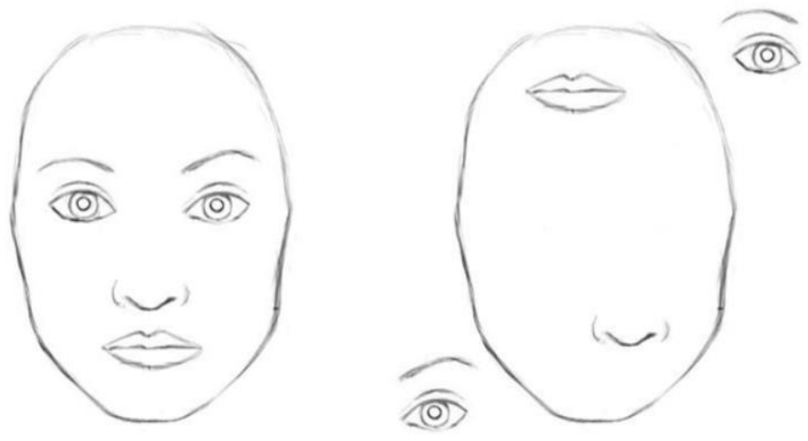
To conclude, the fine-tuning process on the pre-trained VGG16 model has dramatically improved the overall performance on the Flowers dataset, with the validation loss going from 1.2259 in the original VGG16 model to 0.775 by the end of phase D, and the validation accuracy going from 66.76% to 92.94% by the end of phase D. That is an impressive 26.18% improvement in accuracy.

Part C: Research

Deep learning as an area of research and industry interest have seen a surge in popularity the past decade and was largely due to the inception of Convolutional Neural Networks (CNNs), bringing revolutionary advancements in subfields of artificial intelligence such as neural networks and computer vision. However, CNNs does have its share of drawbacks and problems which may impede certain aspects of the field.

Problems with CNN

The core component of a CNN is its convolutional layers, which involves the convolution operation similar in machine vision terms. It is a process where various image filters loop through (or convolve) each pixel in the input image to detect key features within the image. This approach has worked wonders in the fields of machine vision and image classification as evident by AlexNet in 2012, where it outperformed other competing network architectures by 10.8% in top-5 error margin. However, it is this very approach that have led to a flaw pointed out by Geoffrey Hinton, one of the leading figures in deep learning research, whereby having the network to classify images solely by key features with no regards of individual features' spatial information or relation with one another will lead to erroneous classifications for distorted images. [1]



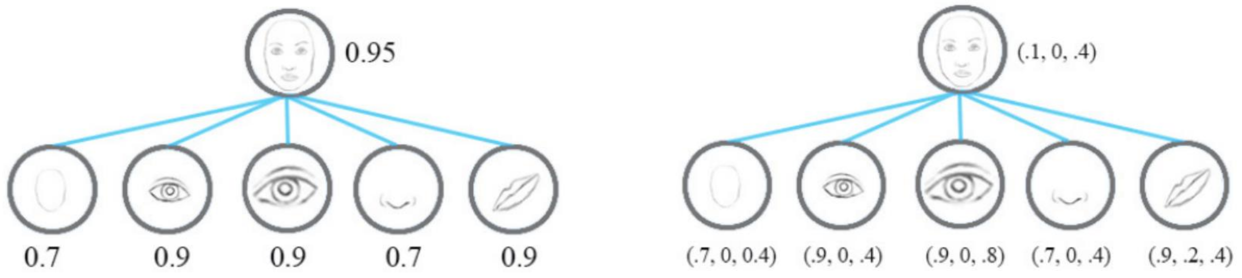
CNNs detect both images as valid faces

In the example image above, a normal CNN would be able to detect both images as faces. While this may hold true for the image on the left, the image on the right is simply not a valid face image, nor should it be detected as one. The misclassification of the right image happens because the image on the right have all key features representing a face present, which essentially tricks the CNN into believing that a face is present.

This further proves the point made by Geoffrey Hinton, whereby spatial information and relationships between key features detected are important in networks for image classification. This is where Capsule networks comes into play, where instead of only encoding scalar probability values during convolution, capsule networks utilize vectors to encode more information than just feature probability. [2]

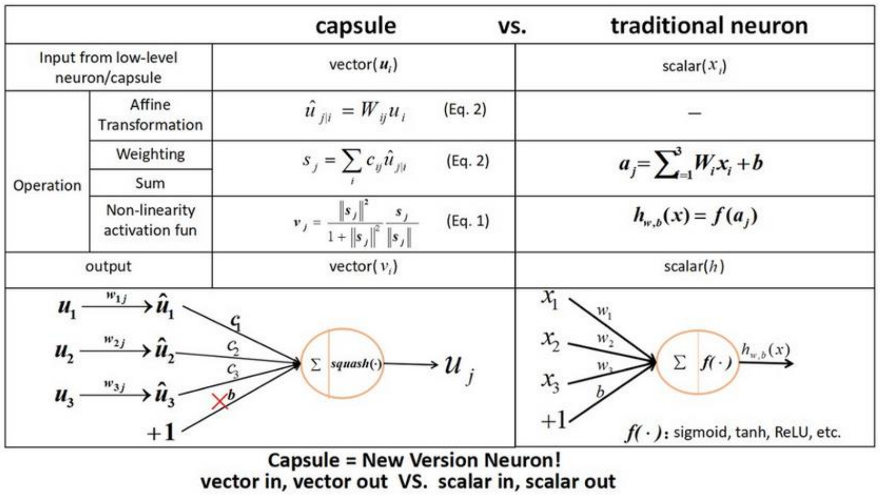
Capsule Networks

As mentioned earlier, Hinton proposed that image recognition requires spatial relationships between key features in an image to be preserved. However, due to CNNs having scalers as outputs after convolution, the required spatial information could not be retained. Therefore, a key component in capsule networks which differs from CNNs are its usage of vectors instead of scalers. Vectors are used as it allows higher dimensions of encoding, which comes in handy when retaining relational and relative information needed for spatial awareness between features. This allows capsules to output a lower likelihood for distorted images as the spatial information within the features in the distorted image are not in line with normal images. [2]



Scalar probability (left) of CNNs and vector probabilities (right) of Capsules

To be more specific, a capsule is a group of neurons whose activity vector represents the instantiation parameters of a specific type of entity such as an object or an object part. It uses the length of the activity vector to represent the probability that the entity exists and its orientation to represent the instantiation parameters (spatial information) [1]. Capsules not only encode instantiation parameters into its vectors, it attempts to predict the instantiation parameters of the detected image feature as well. Capsules exists in a hierarchical manner, consisting of lower and higher-level capsules. Therefore, active capsules of a level will make predictions of the instantiation parameters for higher level capsules via a matrix transformation operation. When multiple predictions for the instantiation parameters reaches an agreement, the higher-level capsule activates, this is also known as dynamic routing [1], and the process repeats. The figure below illustrates the overall operations of a standard capsule network in comparison with a standard CNN.



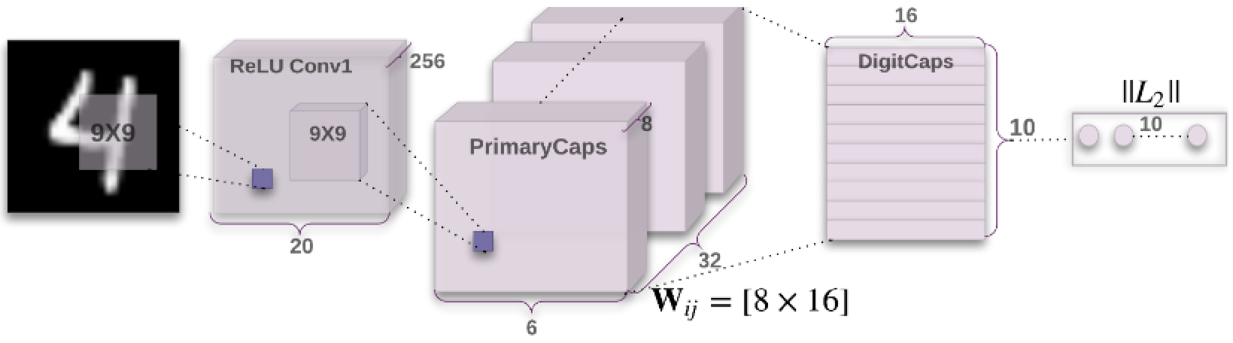
		capsule	vs. traditional neuron
Input from low-level neuron/capsule		vector(u_i)	scalar(x_i)
Operation	Affine Transformation	$\hat{u}_{j\|i} = W_{ij} u_i$ (Eq. 2)	–
	Weighting	$s_j = \sum_i c_{ij} \hat{u}_{j\|i}$ (Eq. 2)	$a_j = \sum_{i=1}^{3} W_i x_i + b$
	Sum		
	Non-linearity activation fun	$v_j = \frac{\|s_j\|^2}{1+\|s_j\|^2}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

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

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

From the chart above, the affine transformation refers to the matrix multiplication between input vectors and weights, which is used to predict instantiation parameters for higher level capsules as stated before. Capsules also undergo weighting and sum operations like traditional NNs, except the weighting in Capsules refer to the decision of which higher level capsules the predictions should be sent to, which is done through dynamic routing. Lastly, the non-linearity activation function in capsules uses the squash function, which compresses the vector to a maximum size of 1 while still retaining the instantiation parameter information.

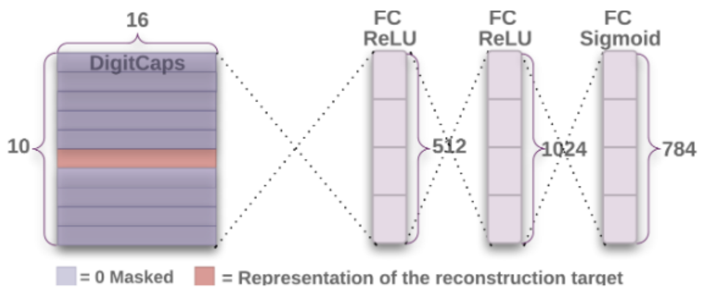
Capsule Network Architecture



The capsules network architecture is made up of a ReLu activated convolutional layer, a primary capsule layer (lower level capsule) and a digit capsule layer (higher level capsule) [1]. The ReLu convolutional layers consists of 256 9x9 filters. The ReLu convolutional layer behaves like an ordinary convolutional layer in CNNs, where it detects key features in input images and then sends it to the primary capsules for further processing.

The primary capsules then receive the key features and uses it to predict instantiation parameters. This is done by the 32 capsules within the layer, where it each applies eight 9x9x256 filters back to the first convolutional layer to predict new instantiation parameters in the form of a 4D vector [1]. This prediction process is essentially a reverse rendering process of images [1], where instead of rendering images based on input, it renders images based on predictions.

The predictions are then sent to the digit capsule layer through dynamic routing, where multiple predictions must first be agreed upon in order to proceed. The higher-level capsules within this layer will then output a 16D vector output of instantiation parameters to be sent to the decoder [2].



Decoder structure connected to fully connected dense layers

The decoder receives the 16D vector input and feeds it into a simple forward pass fully connected neural layer to learn how to reconstruct the image object it is detecting [2]. The reconstructed image object will then be compared with the actual training feature in terms of similarity. This ensures that the capsules will only retain information that will benefit in classifying its input image class within its vectors.

References

[1] Sabour, S., Frosst, N. and Hinton, G., 2020. *Dynamic Routing Between Capsules*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1710.09829> [Accessed 4 May 2020].

[2] Misra, A., 2020. *Capsule Networks: The New Deep Learning Network*. [online] Medium. Available at: <https://towardsdatascience.com/capsule-networks-the-new-deep-learning-network-bd917e6818e8> [Accessed 4 May 2020].