CS7-GV1 : Computer Vision Report

Amit Prasad (17318406)

February 25, 2018

1 Introduction

Convolutional Neural Networks(CNN) have proven to be excellent for the task of image classification in the recent years. They have demonstrated their capability to be comparable to humans in several visual recognition tasks. Although they have been there for many years, it was not until recent that it has been possible to exploit them due to the advances in GPU technology. The basic framework of CNNs can be easily applied towards object detection and segmentation too.

The aim of this assignment is to create and train a new architecture of Convolutional Neural Networks (CNN) for the task of image classification. The given dataset is TinyImageNet that consists of 200 classes. 500 instances of each class belong to the training set and 50 instances to the validation set. All images have size 64 x 64 x 3.

This document is organized as follows.In section 2 the theory behind CNN and techniques for image pre-processing are discussed. Section 3 discusses the approach taken towards solving the task at hand. In section 4, detailed analysis of the results have been discussed and the reasoning behind the choices made. The document finally concludes with a summary of my learnings from this assignment and future work.

2 Theoretical Background

This section will walk you through the necessary theoretical knowledge for understanding the work discussed in the further sections.

2.1 PyTorch

I have used PyTorch v0.3.1 in the implementation of my DL model. PyTorch is a deep learning framework in Python. It supports Tensor computation with strong GPU acceleration [1]. PyTorch gives us dual support of tensors that can either run on the CPU or on the GPU. For obvious reasons of performance I have used the GPU support. PyTorch uses reverse-mode auto-differentiation which allows you to make dynamic changes to your network without any overhead or lag and till date has the fastest implementation of this technique.

2.2 Image Transformations

Augmentation strategies are generally used in the case when the available data isn't sufficient. The general understanding is that, more the data available better the training. This is done by artificially inflating the training dataset with label preserving transformations. Augmentation strategies also prevent overfitting of image data [2], [3]. The most common augmentation strategy is flipping the images horizontally with some probability. Other techniques include 5-Crop, 10 Crop, Random Crop, Vertical Flip, Gray Scaling, Color Jittering and Random Rotation [4].

2.3 Non-Linearity

The purpose of using non-linear activations is to introduce non-linearity in the network. This helps the model to generalize better, otherwise the network will only be able to compute a linear function. There are several non-linear functions like sigmoid, tanh, ReLU and Leaky ReLU [5].

2.4 Pooling layers

Pooling is often referred to as downsampling. This layer helps us to reduce the parameters by upto 75% (which helps to train faster) and also prevents overfitting [6]. The intuition behind this layer is that, features have high activation values and their exact location is not as important as their relative location to other features. The general strategy used is max pooling [7]. Other strategies include average pooling and l2 norm pooling. The basic idea is to take a filter (of say size 2x2) and a stride of the same length and convolute over the input image to produce the output image.

2.5 Dropout Layers

Dropout layers play a very important role in preventing overfitting to the training image data.This layer simply 'drops out' a random set of activations in layer by setting them to zero. Inherently, this forces the network to be redundant i.e. even if some activations are dropped out, it should still be able to predict the right class label [8].

3 Implementation

3.1 Data PreProcessing

As mentioned earlier, I am using the TinyImageNet dataset that consists of 200 classes and 500 instances of each for the training operation. I have applied different augmentation techniques on different models based on the network depth and need for reducing overfitting.

1. **Horizontal Flip** with probability 0.5.
2. **Scaling** the training image to prevent the image from diminishing due to the network depth.
3. **Random Crop** with probability 0.5.

I have considered a combination of these strategies based on the network model.

3.2 Network Design

I have trained many models by varying the network design and changing the hyper parameters. I'll be comparing a few of my prominent models with state-of-the-art architectures of ResNet18, VGG19 and AlexNet.

I. [conv-relu-norm-pool]x4-conv-relu-[fc-softmax-dropout]x1-fc

II. [conv-relu-norm-pool]x6-conv-relu-[fc-softmax-dropout]x1-fc

III. [[conv-relu-norm]-[conv-relu-norm-pool]]x3-conv-relu-[fc-softmax-dropout] x 1-fc

I have experimented majorly with these 3 model design by changing the hyperparameters, output feature map size, weight initializations and augmentation strategies to get the best results from them.

Figure 1: Model-2 Network Design

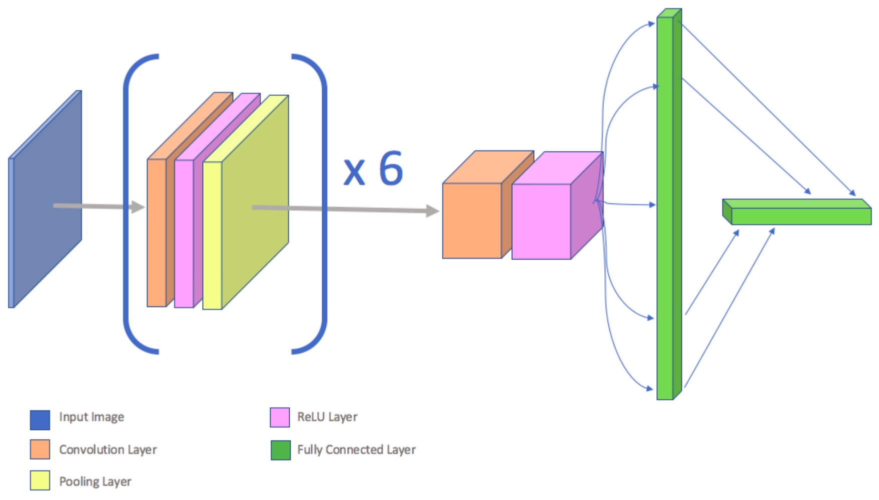


Table 1: Configuration of models

MODEL	TYPE	AUG ST	NON-LIN	KERNEL	FEAT. MAP	DROPOUT	INIT
1	I	H FLIP	ReLU	5X5	128	NO	-
2	II	RCF	Leaky ReLU	3X3	128	YES	-
3	III	RCF	Leaky ReLU	3X3	128	YES	XAVIER
4	II	RCF	Leaky ReLU	3X3	64	YES	-
5	II	RCF	Leaky ReLU	3X3	64	YES	XAVIER
6	II	-	Leaky ReLU	3X3	64	YES	-

AUG ST - Augmentation Strategy
NON-LIN - Non Linearity Strategy
FEAT-MAP - No of Feature Maps
INIT - Weight Initializations
H FLIP - Horizontal Flip
RCF - Random Crop and H Flip

Table 1 on page 4 defines the best configurations I have settled upon after experimenting with different strategies for the model designs above.

4 Experimentation and Analysis

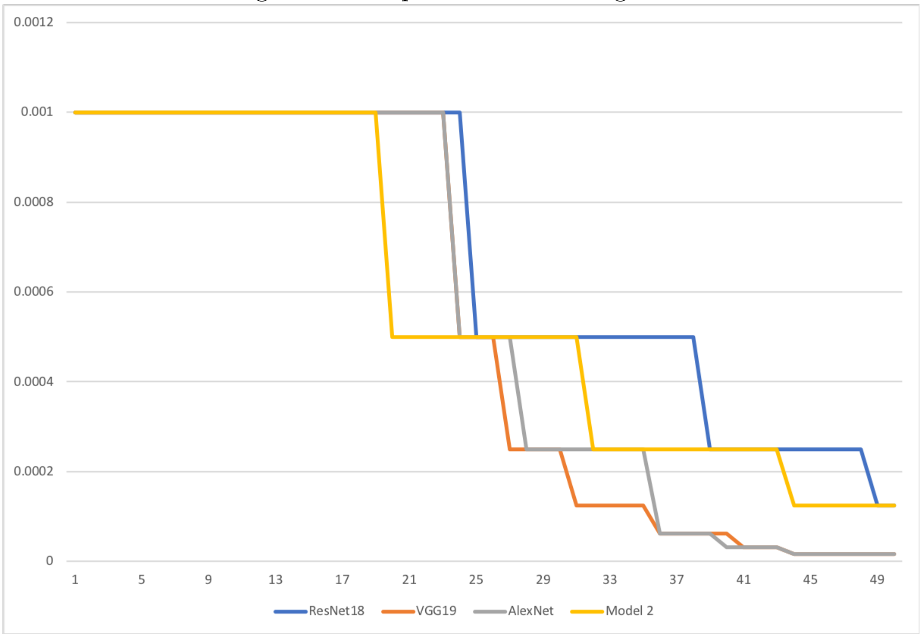
4.1 Training

I have trained my models by minimizing the Cross Entropy loss using Stochastic Gradient Descent (SGD). Other optimization algorithms like Adadelta, Adam, Averaged SGD too are available in pytorch. The basic intuition behind SGD is to imagine a rolling ball down the mountain slope, and is given by $W \leftarrow W - \alpha \Delta w L$. The momentum has been set to 0.9.

The learning rate(LR) has been reduced using ReduceLROnPlateau algorithm. This allows dynamic change in LR by some factor after a certain number of patience epochs. In my case I have preferred to reduce the LR by 50% if there is no considerable change in Validation accuracy for 3 epochs.

Figure 2 shows the variation of learning rate. All learning rates have been initialized with 0.001 and a patience value of 3.

Figure 2: Comparison of Learning Rate



4.2 Effect of Data Augmentation

Data augmentation is another strategy to add regularization to the network. I have used techniques like Horizontal Flip, Random Cropping and Flipping with some probability as my augmentation strategy.

Adding augmentation to Model 6 increased the validation accuracy from 35.29% to 43.97% (i.e. Model 2). This is a considerable difference. The only difference between these two models is the augmentation strategy. Data augmentation strategy used in Model 2 is Random Crop and Horizontal Flip.

4.3 Effect of Dropout Layers

I have added a dropout layer after the fully connected layer to allow regularization. This is also known as dropout regularization, where a neuron is kept active with some probability *p*. In my models, this probability value is 50%. I tried experimenting with the probability value. Increasing this to a higher value resulted in underfitting.

4.4 Effect of Filters

Analysis based on 1 suggests that a higher number of features extracted from input image preserves more information and hence leads to higher accuracy (Figure 7). This is evident from the difference in validation accuracy between Model 2 and Model 4 (5%). This obviously leads to increased computation time and hence slower training.

Model 1 in Table 1 uses a filter size of 5x5 with Stride 2. This resulted in a very low model performance (41.33% top@1 accuracy). Hence for the other models I have used a 3x3 filter with Stride 1 to preserve more information.

4.5 Effect of Activation Function

Initially I trained Model 2 with activation function as ReLU and then with Leaky ReLU. There was a minor performance improvement in terms of accuracy of 0.7% and a very insignificant increase in runtime. Although, I believe that the runtime might have increased due to other processes running on the machine in parallel.

4.6 Effect of Weights Initialization

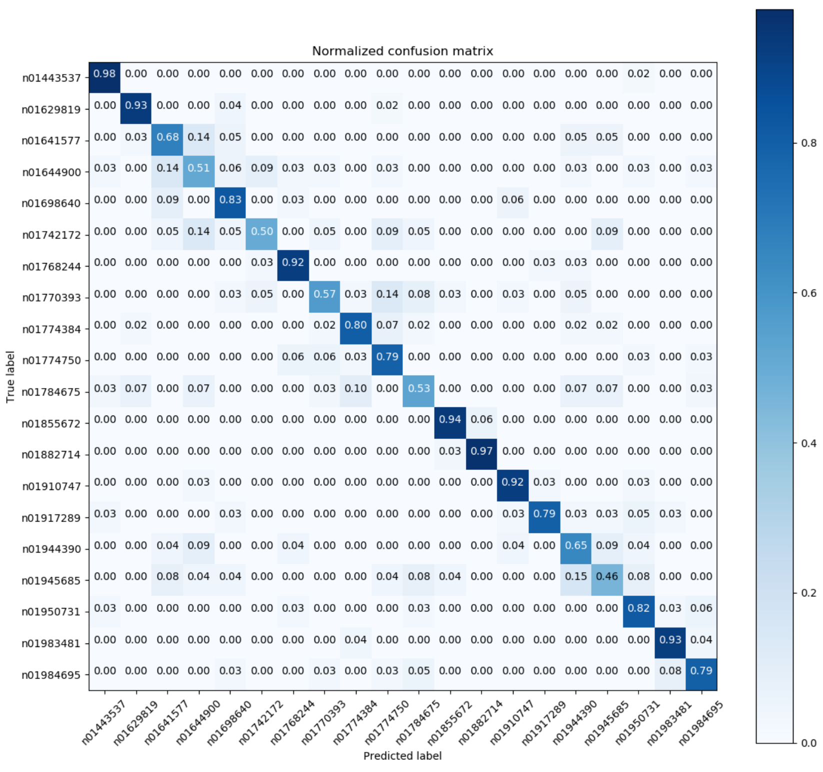
Initializing Model 4 with weights from Xavier algorithm had an adverse effect surprisingly. The accuracy dropped from 44.24% to 43.05%. Although this drop isn't much, the behavior is strange.

4.7 Analysis

From Figure 7 it is evident that Model 2 performs best in terms of Top 1 accuracy as well as Top 5 accuracy. On investigating the confusion matrix in Figure 3 for this model, interesting results were found. I have sliced the confusion matrix shown in Figure 3 for convenience as the validation set contains 200 classes.

Class label-n01443537 which represents a Fish was learned well by our model, and was predicted successfully most of the times (0.98).Scorpio (n01770393) on the other hand was not learnt well by our model. It was sometimes predicted as a Spider (n01774750). This may be due to the high similarity in colour, low resolution of training image and similar body shape, which is reasonable. Similar confusion is shown by this model for Snail(n01945685) and a Slug(n01944390). This confusion can be noticed in the sample images in Figure 4.

Figure 3: Sliced Confusion Matrix for Model-2



	n01443537	n01629819	n01641577	n01644900	n01698640	n01742172	n01768244	n01770393	n01774384	n01774750	n01784675	n01855672	n01882714	n01910747	n01917289	n01944390	n01945685	n01950731	n01983481	n01984695
n01443537	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00
n01629819	0.00	0.93	0.00	0.00	0.04	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
n01641577	0.00	0.03	0.68	0.14	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.05	0.00	0.00	0.00
n01644900	0.03	0.00	0.14	0.51	0.06	0.09	0.03	0.03	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.03	0.00	0.03
n01698640	0.00	0.00	0.09	0.00	0.83	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.00	0.00	0.00
n01742172	0.00	0.00	0.05	0.14	0.05	0.50	0.00	0.05	0.00	0.09	0.05	0.00	0.00	0.00	0.00	0.00	0.09	0.00	0.00	0.00
n01768244	0.00	0.00	0.00	0.00	0.00	0.03	0.92	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.03	0.00	0.00	0.00	0.00
n01770393	0.00	0.00	0.00	0.00	0.03	0.05	0.00	0.57	0.03	0.14	0.08	0.03	0.00	0.03	0.00	0.05	0.00	0.00	0.00	0.00
n01774384	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.02	0.80	0.07	0.02	0.00	0.00	0.00	0.00	0.02	0.02	0.00	0.00	0.00
n01774750	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.06	0.03	0.79	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.03
n01784675	0.03	0.07	0.00	0.07	0.00	0.00	0.00	0.03	0.10	0.00	0.53	0.00	0.00	0.00	0.00	0.07	0.07	0.00	0.00	0.03
n01855672	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.94	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00
n01882714	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.97	0.00	0.00	0.00	0.00	0.00	0.00	0.00
n01910747	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.92	0.03	0.00	0.00	0.03	0.00	0.00
n01917289	0.03	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.79	0.03	0.03	0.05	0.03	0.00
n01944390	0.00	0.00	0.04	0.09	0.00	0.00	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.00	0.65	0.09	0.04	0.00	0.00
n01945685	0.00	0.00	0.08	0.04	0.04	0.00	0.00	0.00	0.00	0.04	0.08	0.04	0.00	0.00	0.00	0.15	0.46	0.08	0.00	0.00
n01950731	0.03	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.82	0.03	0.06
n01983481	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.93	0.04
n01984695	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.03	0.00	0.03	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.08	0.79

Figure 5 shows the comparison of training accuracy's of different models. The graph indicates that ResNet18 and VGG19 is over-fitting the training data and will then result is lesser generalization. These models are using Horizontal Flip as their augmentation strategy.

Figure 6 shows that there was a steady decline in loss as the training progressed. Epoch 36 onwards the performance of ResNet18, VGG19 and AlexNet started saturating. Similar trend can also be observed from Figure 5 which show gradual saturation of training accuracy.

Figure 4: Sample images from Tiny-Image Dataset

5 Results

Figure 7 gives a comparison of top@1 and top@5 accuracy of my models with the state-of-the-art. I have used untrained state of the art models and then trained them on Tiny ImageNet dataset to give a fair comparison. From the results obtained, Model 2, 4 and 5 perform better than VGG19 and AlexNet. Model 2 peforms even better than ResNet18.

Pretrained models of VGG19, AlexNet, and ResNet18 perform much better than the untrained ones and hence I haven't included those results. ResNet18 performs the best amongst the pre-trained models. This again supports the statement, *"More the training data, better the model accuracy"*.

Figure 8 shows the comparison of validation accuracy over 50 epochs. AlexNet and VGG show a similar growth behavior.The same can be observed from Figure 6 that shows the change in log loss over time.

Figure 10 shows the growth behavior of my best performing Model-2. It seems that the model would start to over-fit, if it were trained for a few more epochs. But looking at the loss and slope of validation accuracy, it doesn't seem to be the case in the current trained model.

I have uploaded the results of Model 2 on the Kaggle Competition Page [9]. I have secured the last position (rank 12) with a score of 0.13353.

Figure 5: Training Accuracy of Model 2 and State of the art

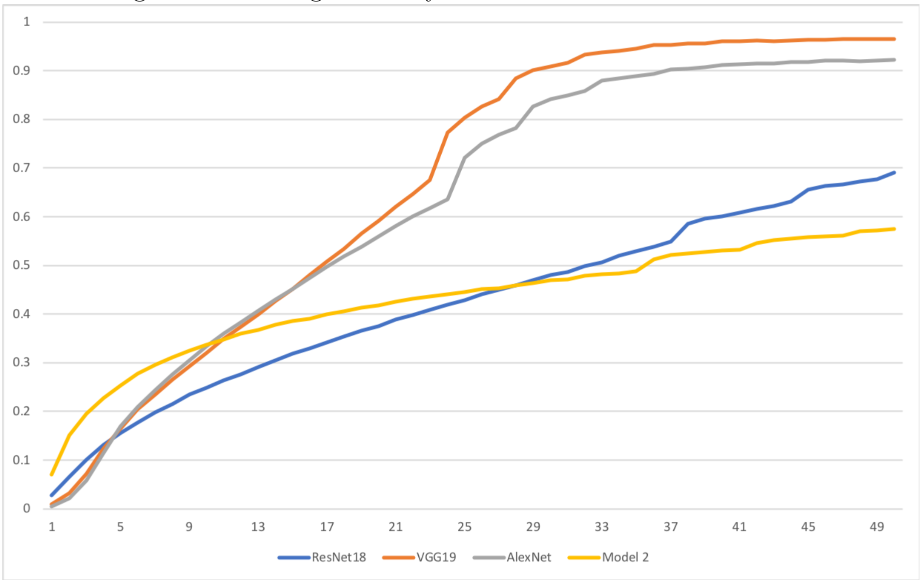


Figure 6: Training Loss

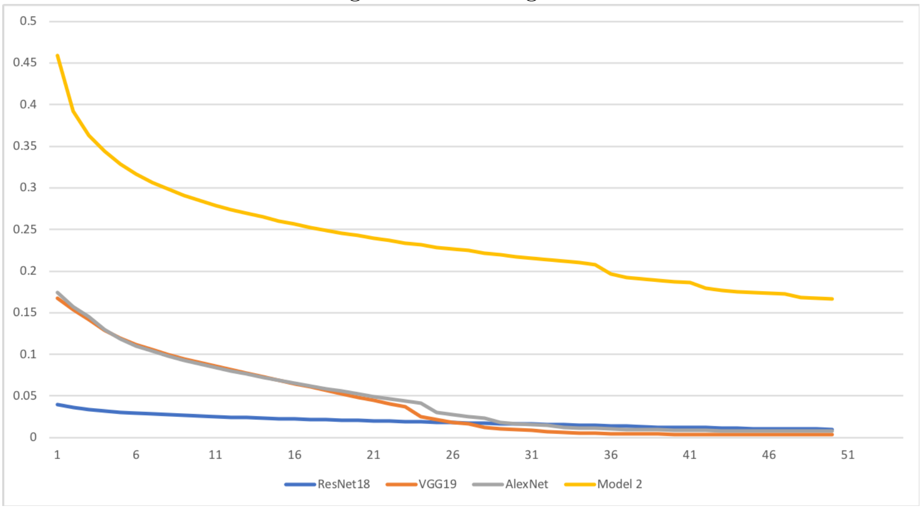


Figure 7: Top Accuracy Comparison of Different Models

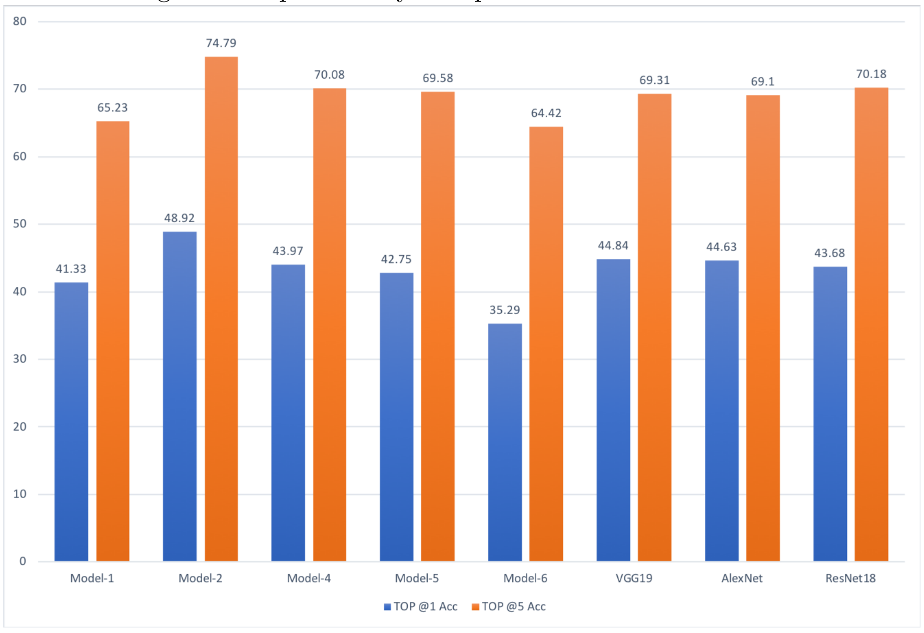


Figure 8: Validation Accuracy of Model-2 vs State-of-the-Art

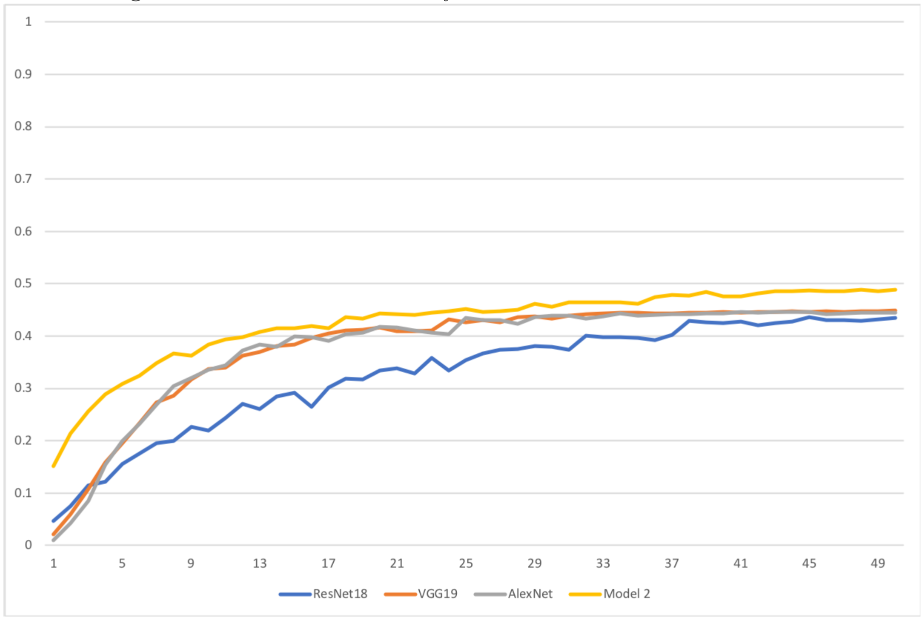


Figure 9: Validation Accuracy of Models

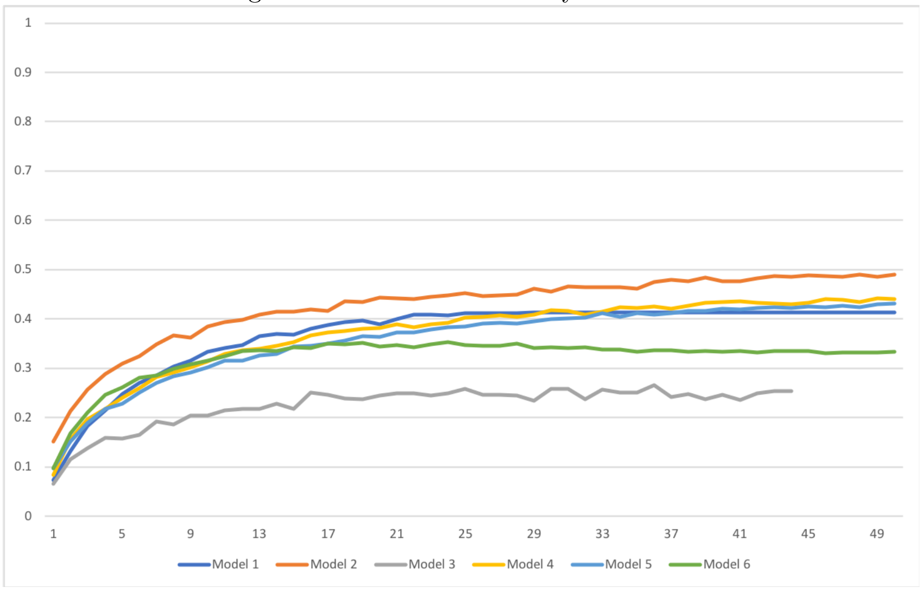
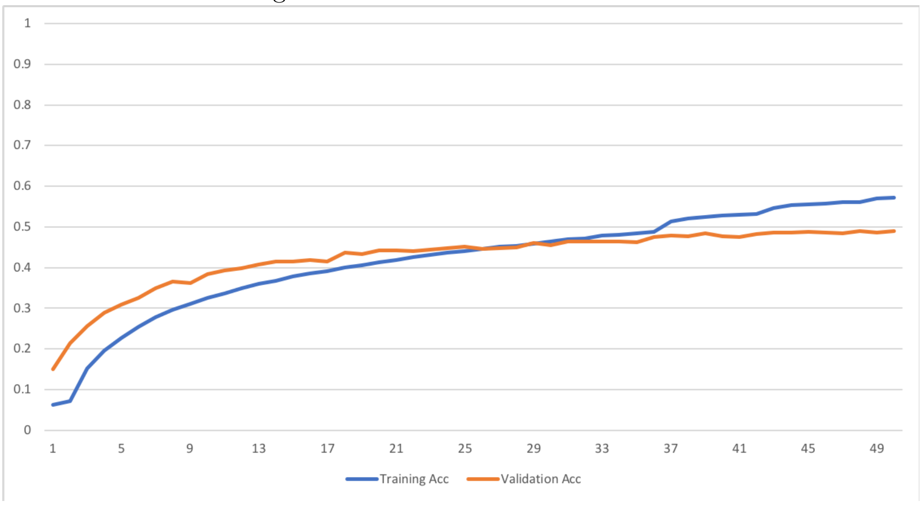


Figure 10: Val-Train Acc of Model 2



6 Conclusion

The proposed name for my best performing model i.e. Model 2 is *CassataNet*. The convolutional layers of this model are stacked beautifully just like the ice-cream with a final sprinkle choco chips analogous to the fully connected layer.

The aim of this assignment is to provide preliminary understanding of deep learning with focus on CNNs. I have thoroughly explored various state-of-the-art models and different techniques used by the authors to improve network performance. Experimenting with different hyper parameters and the effects of width & depth of the network has given me practical experience and aroused further interest in CNNs.

7 Future Work

It will be interesting to see how Model 2 performs when trained on the complete ImageNet dataset. We can also observe the change in model's performance by combining the learning's of two models.

Another interesting approach is to use object bounding boxes in the given data set to exclude the images in which the actual object occupies a very low percentage of the image. The reasoning here is that, if the object occupies a low area in the image, then the network might learn the wrong features instead of features from the object under consideration.

References

[1] Pytorch webpage. `http://pytorch.org/about/`. Accessed: 2018-2-17.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, pages 1–9, 2012.

[3] Andrew G. Howard. Some Improvements on Deep Convolutional Neural Network Based Image Classification. *arXiv preprint arXiv:1312.5402*, pages 1–6, 2013.

[4] Andrew G. Howard. Some Improvements on Deep Convolutional Neural Network Based Image Classification. *arXiv preprint arXiv:1312.5402*, pages 1–6, 2013.

[5] Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on Machine Learning*, (3):807–814, 2010.

[6] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323, 1998.

[7] Dan C. Cireşan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. *IJCAI International Joint Conference on Artificial Intelligence*, pages 1237–1242, 2011.

[8] Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on Machine Learning*, (3):807–814, 2010.

[9] Kaggle competition page. `https://www.kaggle.com/c/cs7-gv1-2017/leaderboard`. Accessed: 2018-2-25.