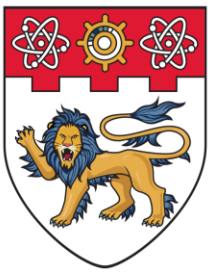


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4042 – Neural Networks and Deep Learning

Course Project Report – Material Recognition

Team members

Name	Matric Number
Mantri Raghav	U1822309B
Singh Kirath	U1822329J
Chatterjee Pramurta	U1822597G

Table of Contents

Introduction

The aim of this project is to train different models based on convolutional neural networks to classify pictures of a given dataset to one of the 10 classes. In this project, we use the MINC-2500 dataset for material recognition.

Here, we have evaluated and utilised, as well as tuned 3 different models: -

1. AlexNet
2. Inception-v3
3. VGG-16

About the Dataset

For this project, we have utilised the MINC-2500 dataset instead of the FMD dataset given in the project handout due to its larger size. It is a patch classification dataset with 2500 samples per category.

This is a subset of MINC where samples have been sized to 362 x 362 and each category is sampled evenly.

It has a total of 23 different classes of images, out of which for this project we utilise 10 classes. These classes are fabric, foliage, glass, leather, metal, paper, plastic, stone, water, and wood.

Data Augmentation

For the model to be more robust and generalize better to test images, and not overfit, data augmentation was performed where slightly modified copies of existing images were added to the training data.

- **Flipping:** The images were flipped both in the horizontal and vertical direction by 180 degrees.
- **Rotation:** The images were rotated by random degrees within the range [0,50] in both clockwise and anti-clockwise directions.
- **Width and Height Shift:** The width and height of the images were shifted by a random amount within the range of [0,20%] of the image width and height respectively, thus creating new randomly cropped images.

Train-Validation Split

In total, we have 25000 images, out of which we use 20000 for training and 5000 as the validation set. Hence, we have used a 4:1 train-validation split for our dataset.

Models

AlexNet Implementation

Introduction

AlexNet is a convolutional neural network architecture, which was first utilized in the public setting when it won the ImageNet Large Scale Visual Recognition Challenge(ILSSVRC 2012 contest). It was designed by Alex Krizhevsky in collaboration with Geoffrey Hinton and Ilya Sutskever. According to the original paper by Alex et al. [1], the depth of the model is essential for the good performance of AlexNet as compared to LeNet. This method, though computationally expensive, yielded good results on ImageNet, and can be make quicker using GPUs (Graphical Processing Units) during the training process.

AlexNet was the first CNN trained using GPUs.

Architecture

The original AlexNet architecture was trained on the ImageNet dataset. This dataset had RGB images of size 224 x 224. There are 5 convolutional layers, and 3 fully connected layers.

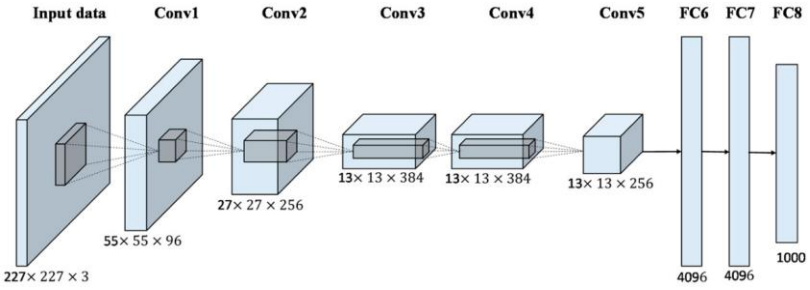


Figure 1: AlexNet architecture (Note - the dimensions given here are as per the ImageNet dataset, but the architecture remains the same)

Each convolutional layer consists of convolutional filters and a non-linear activation function ReLU.

The pooling layers are to perform max pooling (to down sample), and overall AlexNet has 60 million parameters.

Model Improvements

To obtain good results and get better performance out of AlexNet, a few key changes have been done to the model and the accuracies have been examined accordingly. The key changes are :-

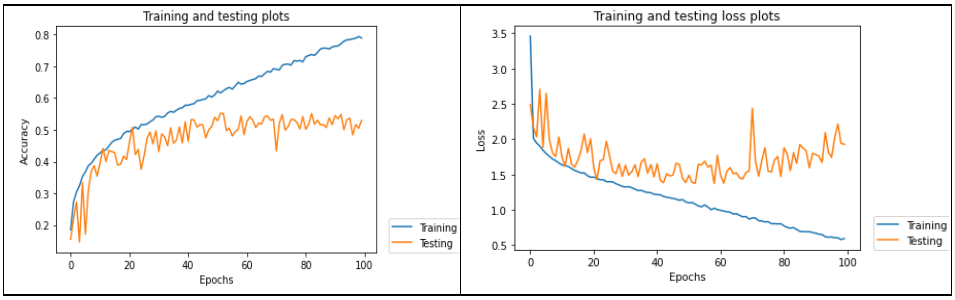
1. **Adding Batch Normalisation layers** after every convolutional layer. This helps in mitigating the effect of unstable gradients within a neural network., through introduction of an additional layer that performs operations on the inputs from the previous layer. These operations normalise and standardise input values. After this, input values are transformed using scaling and shifting operations
2. **Dropouts** - To reduce the effects of overfitting, dropouts have been introduced not only after the input layer but after the activations of each and every convolutional layer.

3. **Input shapes** have been changed from to 150x150, since this is also the dimensions of the augmented images. This helps in training the images a lot faster as compared to the original 362 x 362 dimensions.
4. **L2 regularisation** – To reduce variance so that the model generalises well, it is pivotal to do regularisation. Hence, we utilise L2 regularisation at each convolutional layer.
5. **Callbacks** – For this project for AlexNet, we have utilised 2 callbacks –
    a. **Model checkpoints** – Early stopping with patience of 10
    b. **Learning rate** - Decay rate is set to be 0.9 with 5 decay steps.

Results

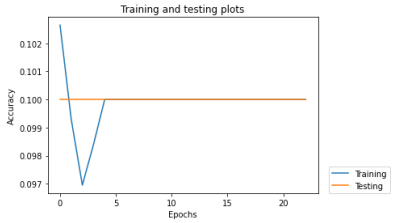
Here are the model results with using dropouts only at the input layer with batch normalisation.

The model ran on till the 100th epoch and the results are given as follows :-



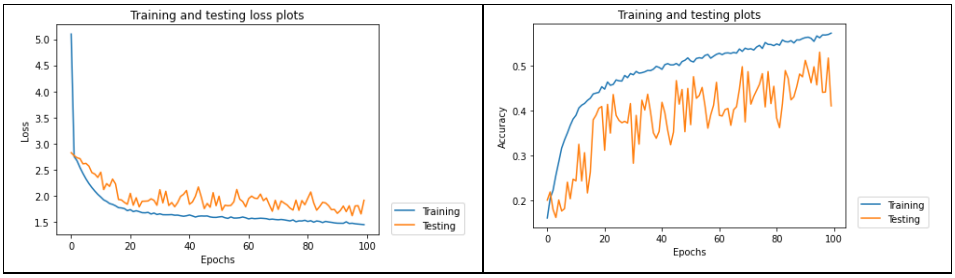
Here, we notice that our validation accuracy peaks out at around 0.51, while training keeps rising to 0.78. The model is overfitting on the training data.

Upon removing Batch normalisation to study its effects, we noticed that the test accuracy didn't change, and the model stopped training as per the call back we set.



Hence, we realised that batch normalisation is essential since it mitigates unstable gradient value effects.

Also, to avoid overfitting, we used dropouts after every layer, and our final plot of training loss versus validation loss is as follows –



We realise that despite having early stopping and regularisation and a low learning rate, there are fluctuations in accuracy values for AlexNet, and hence the convergence is not smooth. But here, overfitting is mitigated. Here, the best validation accuracy is **53.02 %**.

VGG-16 Implementation

Introduction

VGG16 is a simple and widely used Convolutional Neural Network (CNN) Architecture used for ImageNet, a large visual database project used in visual object recognition software research. The VGG16 Architecture was developed and introduced by Karen Simonyan and Andrew Zisserman from the University of Oxford, in the year 2014, through their paper “Very Deep Convolutional Networks for Large-Scale Image Recognition.” [2] ‘VGG’ is the abbreviation for Visual Geometry Group, which is a group of researchers at the University of Oxford who developed this architecture, and ‘16’ implies that this architecture has 16 layers.

Architecture

The original VGG16 model was trained on ImageNet dataset with an input shape of 224*224*64. The training images are passed through a stack of convolution layers. There are a total of 13 convolutional layers and 3 fully connected layers in the original VGG16 architecture. VGG16 has smaller filters (3*3) with more depth instead of having large filters. This construction consists of a total of 138 million parameters.

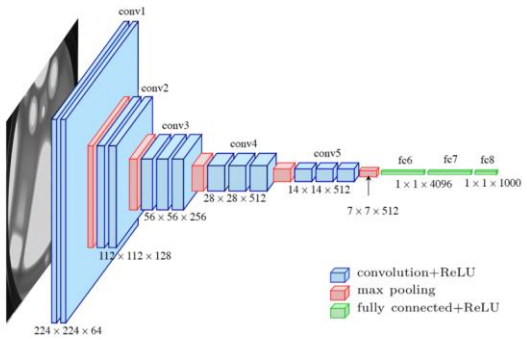


Figure 2: VGG-16 architecture

Layer walkthrough of the original architecture

- The first two layers are convolutional layers with 3*3 filters, and first two layers use 64 filters that results in 224*224*64 volume as same convolutions are used. The filters are always 3*3 with stride of 1
- After this, pooling layer was used with max-pool of 2*2 size and stride 2 which reduces height and width of a volume from 224*224*64 to 112*112*64.
- This is followed by 2 more convolution layers with 128 filters. This results in the new dimension of 112*112*128.
- After pooling layer is used, volume is reduced to 56*56*128.
- More convolution layers are added with 256 filters each followed by down sampling layer that reduces the size to 28*28*256.
- Two more stack each with 3 convolution layer is separated by a max-pool layer.
- After the final pooling layer, 7*7*512 volume is flattened into Fully Connected (FC) layer with 4096 channels and softmax output of 1000 classes.

Improvements and Modifications

In order to use the VGG16 architecture to fit our problem statement we came up with a set of changes -

1. Added a **batch normalisation** layer after every activation layer except the softmax activation layer of the last softmax activation layer as it limits effect to which updating the parameters of early layers can effect the distribution of values seen by the next layers. Moreover, batch normalisation enables the layers to learn better and more independently. [3]
2. Added dropout layers with varying dropout rates. Dropouts help us in preventing overfitting.
3. Changed the input shape to 150*150*3 as the augmented images generated from the pre-processing had a shape of 150*150*3.
4. Reduced the nodes in the output layer from 1000 to 10 as the model is supposed to make the probability predictions of 10 different class of objects as opposed to 1000.
5. Added l2 regularisation to all the convolution layers. This helps in significantly reducing the variance of our model with a weight decay of 0.0005.
6. Call-backs added -
    - Early stopping with a patience of 40 epochs by monitoring test loss.
    - Model checkpoint to save the best model during training by monitoring test accuracy..

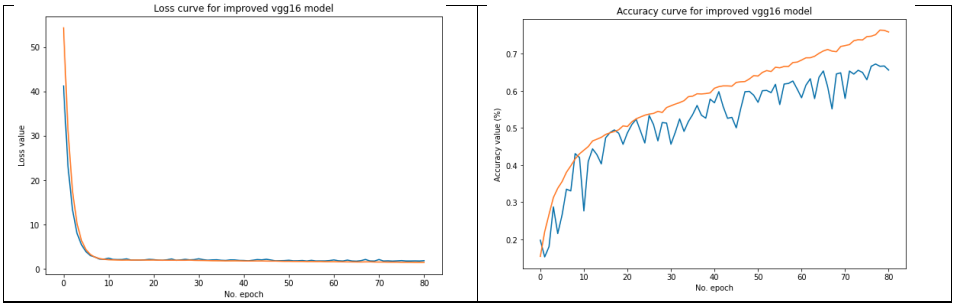
- Learning rate decay by a factor of 0.9 with patience 3, the minimum learning rate was set to 0.000001 and the initial learning rate was set to 0.1.

7. SGD optimiser was used with nestrov set to True and a momentum of 0.9. SGD was preferred over adam because, during our experimentation, SGD with the above-mentioned parameters was making our model converge, while on the other hand adam optimiser did not lead to convergence due to the large number of parameters in the VGG network. The reason behind this behaviour is that SGD is more locally unstable and is more likely to converge to the minima at the flat or asymmetric basins/valleys which often have better generalization performance over other type minima.[4] A study was conducted in University of California, Berkely where similar results were found.[5] The experiment showed that adaptive methods are not superior to non-adaptive methods and more often than not, finely tuned non-adaptive methods perform better than adaptive methods. The paper however stated that the reason behind this behaviour could not be determined.
8. Performed an experimentation using batch sizes in the sample space - (16,32,64,96,128,160). The best results were obtained using a batch size of 64 as the lower batch sizes had a enormous training time, while on the other hand, the validation results were not good enough for higher batch sizes.

Results

We used early stopping with a patience of 40 in order to stop the training process if we do not experience any change in validation loss. Hence our model was trained for 106 epochs and stopped thereafter due to early stopping. The test accuracy that we observed was in the ballpark of 70%. In our earlier experimentation we observed that the model usually converges at an accuracy of 70% hence we used early stopping to significantly reduce our training time.

The plots for the test loss and accuracy can be found below.



Inception-v3 Implementation

Introduction

Inception-v3 is a convolutional neural network architecture which mainly focuses on improving the model performance and consume lesser computational resources compared to its previous Inception architectures. Some notable improvements include using Label Smoothing, Factorized 7 x 7 convolutions, and the use of an auxiliary classifier to propagate label information lower down the network (along with

the use of batch normalization for layers in the sidehead). This idea was proposed in the paper Rethinking the Inception Architecture for Computer Vision [6], published in 2015. It was co-authored by Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, and Jonathon Shlens.

Architecture

The Keras implementation of the Inception-v3 was trained on ImageNet dataset with an input shape of 299*299*3. There are a total of 311 layers comprising of a series of stacks of Inception modules which are made up of Inception blocks that contain the concatenated output of the previous layer after the previous layer is passed through multiple kernel filters and max/average pooling layers.

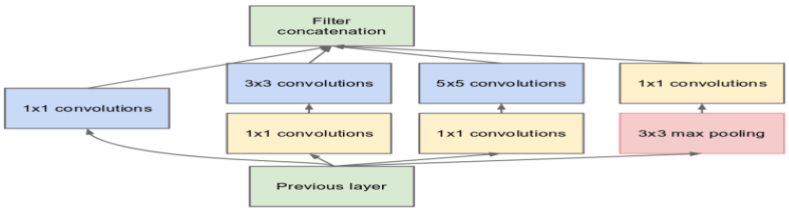


Figure 3: An inception block

The changes which were made to the original Inception architecture to further enhance its performance in Inception v3 are as follows:

- **Grid size reduction and Factorized Convolutions:** A series of sophisticated arrangements of pooling and convolutional layers help to improve the computational efficiency as it reduces the number of parameters involved in a network. It also keeps a check on the network efficiency.
- **Smaller Convolutions:** Replacing bigger convolutions with smaller convolutions leads to faster training. For example, a 5 × 5 filter has 25 parameters; replacing that with two 3 × 3 filters leads to fewer parameters (in this case 18).
- **Auxiliary Classifier:** It is a small CNN inserted between layers during training, and the loss incurred is added to the main network loss. In GoogLeNet auxiliary classifiers were used for a deeper network, whereas in Inception v3 an auxiliary classifier acts as a regularizer.

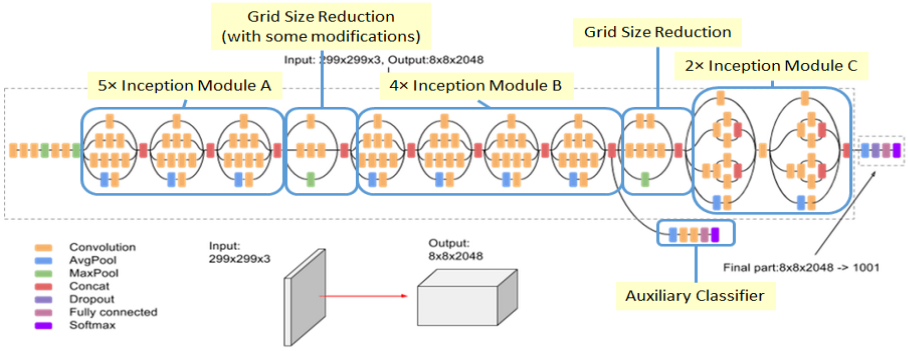


Figure 4: Inception v3 Architecture

Improvements and Modifications:

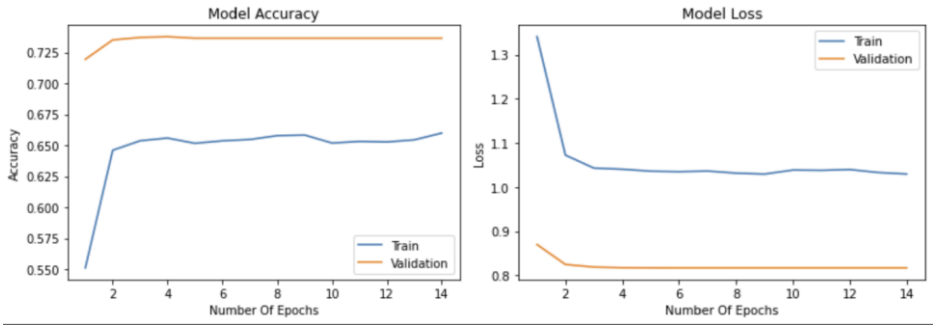
- **Freezing weights and the addition of custom dense layers:** A total of 6 experiments were performed on the Inception v3 network where the weights of some layers were frozen, and the rest were trainable; they were followed by a series of dense layers with dropouts and a SoftMax layer for class prediction:
    - **Freezing the weights of the first 50 layers:** The training accuracy was hovering 40%, so this model was rejected.
    - **Freezing the weights of the first 100 layers:** Although, the training accuracy improved from 40% to 50% the model was still underfitting, so it was rejected.
    - **Freezing the weights of the first 150 layers:** Although the training accuracy improved from 50 to 62%, the validation accuracy was below 50% which clearly indicates overfitting. So, this model was rejected.
    - **Freezing the weights of the first 200 layers:** The training accuracy hovered around 65% but the validation accuracy was still below 50%, thus leading to overfitting. Hence, this model was rejected.
    - **Freezing the weights of the first 250 layers:** The training accuracy was around 65%, but the validation accuracy climbed to around 74%. Therefore, this model was chosen as the final one.
    - **Using all the layers:** The training accuracy was around 65% but the validation accuracy dropped to 56%. Therefore, this model was rejected.

  Freezing the weights of the first 250 layers gave the best results.

- **Learning Rate Decay:** To ensure smooth convergence towards the global minima, learning rate decay was used with a decay rate of 0.9 (best value after tuning), and the number of decay steps was set to 5 as the convergence took place quickly.
- **Dropouts:** To prevent overfitting, dropout layers were added between the dense layers.

Results:

After performing 6 experiments by taking different subsets of validation data, followed by data augmentation, freezing weights of the first 250 layers of the Inception-v3 model and the addition of a few dense layers with dropouts, the model is robust, and it can generalize very well which can be clearly seen from the higher validation accuracy than training accuracy. Also, an early stopping callback was set with a patience of 10 epochs.



The best validation accuracy of the Inception-v3 model is **73.64%.**

Model Comparisons

Models	Validation Accuracy (best epoch)	Validation Loss (best epoch)	Best Epoch
AlexNet	53.03%	1.6212	100
VGG-16	70.06%	1.8057	106
Inception-v3	73.64%	0.8168	14

Testing Results

We used 190 images for testing, out of which 90 are google images and the rest are taken from the FMD dataset, all of which are evenly sampled. After testing with the best model (Inception-v3), we got a **test accuracy of 79.69%**.

Conclusion

After obtaining our results, we realise that our best test accuracy is 79.69% that can be achieved by the Inception-v3 model.

We have used a larger dataset rather than the one provided in the project handout, since the images in the other dataset are quite few, and neural networks require substantial amounts of training data to not just generalize well but also maintain decent training accuracy.

Commented [#R1]: @#CHATTERJEE PRAMURTA# can you update this to the test accuracy you got

Commented [#R2R1]:

Challenges

In this project, we faced numerous challenges, which are mainly attributed to the way neural networks train and the resources required by CNNs to run efficiently:-

1. **Large training time** – Our models took at least 4-5 hours to train on our dataset, and sometimes even more as layers were added. For AlexNet, it took around 9 hours to run 100 epochs.
2. **Limited GPU usage** – Due to lack of GPUs in our own machines, we relied on Colaboratory by Google, which has a maximum quota of 12 hours for GPU at a time. Hence, we faced issues when our model couldn't train within the stipulated quota time.
3. **Large number of hyperparameters** – For our project we noticed that there are a lot of hyperparameters can be tuned, and given the training time, it was challenging for selecting the best set of parameters to tune in order to obtain results on time, since we had to judiciously use online GPUs.
4. **High variance in models** – Due to high variance in our models, we had to find larger datasets than the one recommended.

Commented [#R3]: @#SINGH KIRATH# @#CHATTERJEE PRAMURTA# Feel free to add/edit the challenges written here

Learning points

This project gave us a great opportunity to put to test our skills and knowledge obtained in the deep learning and neural networks lectures and tutorials.

Our key learning points were:-

1. **Convolutional neural networks** – how to implement them in TensorFlow, the different hyperparameters that we can tune, what is max pooling and why we need to do batch normalisation, etc.
2. **Knowledge about State-of-the-Art Models** – AlexNet, VGG-16 and Inception are high quality state of the art models used in numerous applications around the globe. It was a great opportunity to put them to test on our work and learn about these industry-relevant models, and how they work.
3. **The benefit of GPUs** – Highly efficient and fast computational resources are highly useful for operations involving neural networks. Hence, we found out about online tools, and used Google Collaboratory for our project.
4. **Teamwork and patience** – This project required meetings to convey to one another our ideas and knowledge, and to help one another in times of need. We also required a lot of patience since our models took magnanimous amounts of time to train.

Scope of Improvement

Despite improvements on the results that we got after hyperparameter tuning as compared to vanilla architectures, there is still room for improvement: -

1. Masking of images for training and validation – to eradicate noise from the background, masking and segmentation could be done for the model to train and focus on only the relevant parts of the image
2. Potentially combining and training different architectures like an ensemble could be another way of yielding high accuracies
3. In order to generalise the model even further after using a larger dataset, we could scrape images from the internet, and utilise them for training and testing. This could make the models more robust to different kinds of images.

References

1. A. Krizhevsky, I. Stustekver and G. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks”, 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf [Accessed: 12-Nov-21]
2. K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv.org*, 10-Apr-2015. [Online]. Available: https://arxiv.org/abs/1409.1556. [Accessed: 12-Nov-2021].
3. S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” 02-Mar-2015. [Online]. Available: https://arxiv.org/pdf/1502.03167.pdf%27. [Accessed: 12-Nov-2021].
4. P. Zhou, J. Feng, C. Ma, C. Xiong, S. HOI, and W. E, “Towards Theoretically Understanding Why SGD Generalizes Better Than ADAM in Deep Learning,” 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/f3f27a324736617f20abbf2ffd806f6d-Paper.pdf. [Accessed: 12-Nov-2021].
5. A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, “The Marginal Value of Adaptive Gradient Methods in Machine Learning,” 22-May-2018. [Online]. Available: https://arxiv.org/pdf/1705.08292.pdf. [Accessed: 12-Nov-2021].
6. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” arXiv.org, 11-Dec-2015. [Online]. Available: https://arxiv.org/abs/1512.00567. [Accessed: 12-Nov-2021].