Transfer Learning for Image Classification

Ayush Dabra
IISER Bhopal
ayushd19@iiserb.ac.in

July 19, 2020

Abstract

For the Applied Machine Learning and Data Science Program by IIT Kanpur, I have worked on the image classification project. The dataset for this project is a small scale version of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). In this project, I approached the image classification problem by using transfer learning on custom VGG16 CNN architecture. I have also applied data augmentation methods to training images, hoping to artificially create variations that help the model to generalize better. The model performed well and achieved 56.23% accuracy on the validation set.

1. Introduction

The dataset is very similar to the well known ImageNet[1] Challenge (ILSVRC). The goal is to achieve the best possible performance for the image classification problem. The dataset is a subset of the ImageNet Challenge where it contains 200 classes instead of 1000 classes. Each class has 450 training images, 50 validation images, and 50 test images. The final network is given a test image and should output a class prediction out of 200 classes. Since the problem and dataset are very similar to the ImageNet Challenge (ILSVRC), it is very natural to consider that the CNN architectures of the previous winners of the challenge can be very useful for this problem. For this purpose, I referred to the reference videos[2] on CNNs by Prof. Vipul Arora, which gave me a fair idea of the VGG16 CNN architecture.

VGG Net was introduced in the year of 2014 that has become popular due to having a simple yet deep and powerful network with small-sized filters (3×3). This allowed the network to be more efficient in terms of the number of parameters.

I focussed on the VGG16 model and modified the model by adding fully connected dense layers, batch normalization layers and dropout layers at the bottom, I also added a fully connected layer at the last and made it to output to 200 class scores and finally trained the modified network to get the predictions on the test set.

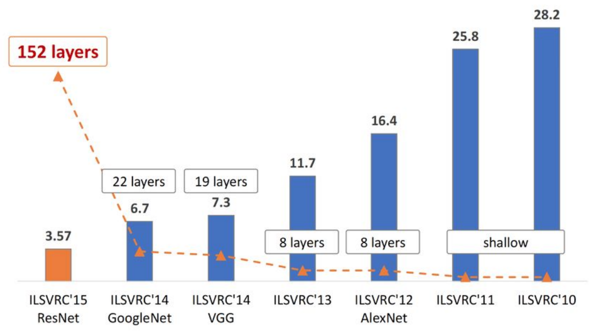

Figure 1. ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners[3]

2. Dataset

The dataset contains square images, of 64×64 pixels. Most of the images have 3 channels for color, RGB, meaning they are 64x64x3 arrays. However, some of the examples are grayscale images, i.e. 64×64×1 arrays. For the sake of simplicity, all the grayscale images are immediately converted to RGB by replicating the pixel values across the three channels. Each image belongs to exactly one out of 200 categories. The training set contains 90,000 images (450 from each category), and the validation and test sets have 10,000 images each (50 from each category).

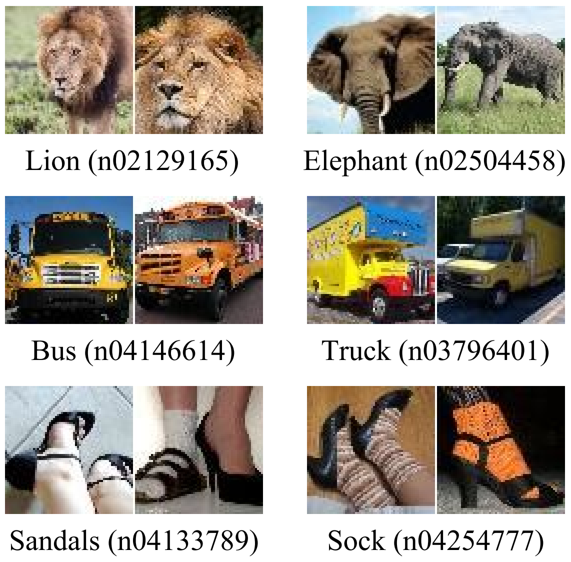

Figure 2. Example images from the training set of the dataset.

Figure 2 shows two examples from six distinct classes, picked from the training set of the dataset. Some of the challenges are evident. First, the images are not very highly resolved (only 64×64), which makes details hard to spot. Moreover, while the algorithm is expected to perform some relatively easy, coarse grain classification (for example, distinguishing a lion from an elephant), it is also expected to perform some very fine classification (for example, distinguishing a truck from a similar-looking bus). Thus, even though the number of classes is less than ImageNet, the classification task is still significantly challenging.

3. Approach

3.1 Data Augmentation[4]

The dataset contains 450 training images per class. Compared to other datasets like ImageNet or CIFAR-10, it is a relatively small amount of data. In order to artificially increase the amount of data and avoid overfitting, I preferred using data augmentation. Data augmentation was achieved through the following techniques:

- **Rescaling**

The images were rescaled by a factor of 1.0/255 because the original images consist of RGB coefficients in the 0-255, but such values would be too high for our model to process, so we target values between 0 and 1 instead, by scaling with a 1.0/255 factor.

- **Shear Transformation**

The shear_range argument of the ImageDataGenerator class is set to 0.2, this slants the shape of the image. This is different from rotation in the sense that in shear transformation, we fix one axis and stretch the image at a certain angle known as the shear angle. This creates a sort of 'stretch' in the image, which is not seen in rotation.

- **Zoom**
  The zoom_range argument of the ImageDataGenerator class is set to 0.2, this magnifies the image by a factor of 0.2.

- **Horizontal Flipping**
  The horizontal_flip argument of the ImageDataGenerator class is set to True, so the generator will generate images which, on a random basis, will be horizontally flipped.

- **Rotation**
  The rotation_range argument of the ImageDataGenerator class is set to 20, so the data generated is randomly rotated by an angle of +20 to -20 (in degrees).

- **Width Shifting**
  The width_shift_range argument of the ImageDataGenerator class is set to 0.2, this is the upper bound of the fraction of the total width by which the image is to be randomly shifted horizontally, either towards the left or right.

- **Height Shifting**
  The height_shift_range argument of the ImageDataGenerator class is set to 0.2, this is the upper bound of the fraction of the total height by which the image is to be randomly shifted vertically, either towards the up or down.

In order to speed up the training process, I have applied these methods in a random fashion. When an image is fed to the model, every augmentation method is applied randomly to the image. In this way, the total number of training examples is the same but I managed to have the model see slightly different but highly recognizable images at each epoch. Figure 3 gives concrete instances of the above methods.

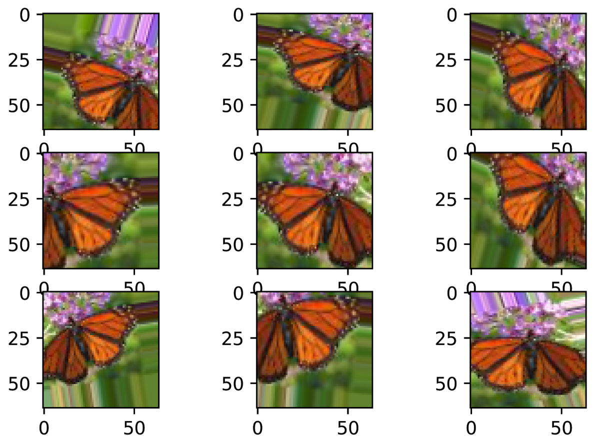

Figure 3. Augmented images of a butterfly from the training dataset.

3.2 Modified VGG16 Architecture

VGGNet was introduced by Simonyan and Zisserman in 2014[5]. This network is characterized by its simplicity. It contains 16 CONV/FC layers and features an architecture that only consists of 3×3 convolutions and 2×2 pooling from the beginning to end.

In this project, I have implemented the VGG16 CNN model with some modifications. I have added two blocks of a fully-connected layer of size 512 having ReLU non-linearity, a batch normalization layer, and a dropout layer having a dropout rate of 40%. Finally, in the end, softmax activation is applied to the output of the FC-200 layer. Figure 4 shows the modified VGG16 CNN architecture.

3.3 Transfer Learning

Transfer learning[6] consists of using models that were trained for a certain task and leveraging the knowledge that they acquired on a different, but related task. This can be

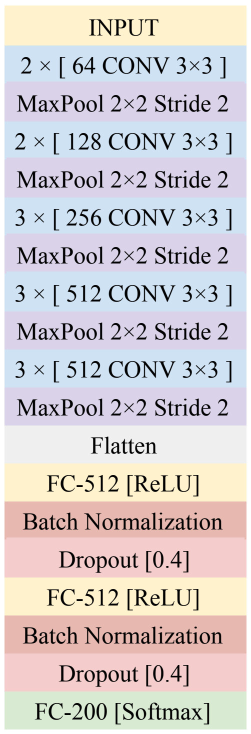

Figure 4. Modified VGG16 Architecture

highly advantageous when there is not much data available for training directly on the related task, or when it is desired to leverage powerful models that would otherwise take several weeks to properly train and cross-validate. Due to small training dataset and computational constraints, I have opted for transfer learning. The VGG16 model that I am using for the project is pre-trained on the ImageNet dataset.

4. Training Details

The architecture was coded and trained using TensorFlow and Keras. I downloaded the VGG16 model with ImageNet weights and without the fully connected output layers from the Keras API[7]. Everything was performed on Kaggle[8] Kernel, using 1 NVidia Tesla P100 GPU[9]. I have used a batch size of 64 for both the baseline model and the fine-tuned model. Adam optimizer has been used for training, with cyclical learning rate.

4.1 Baseline Model

I froze the layers of the model, so as to avoid destroying any of the information they contain during future training rounds. Then I added some new, trainable layers on top of the frozen layers, they will learn to turn the old features into predictions on a new dataset, as I have already mentioned in section 3.2. The baseline model has a total of around 16 million parameters out of which there are around 1.4 million trainable parameters.

4.2 Cyclical Learning Rate

Learning rate was the hardest hyper-parameter to tune, and every model is highly sensitive to a poorly fine-tuned learning rate. Instead, a cyclical[10] learning rate solves this problem by cyclically changing the learning rate within a reasonable bound. I have used cyclical learning rate in the range of 1e-4 to 6e-4 with a step size of 1404 for the baseline model.

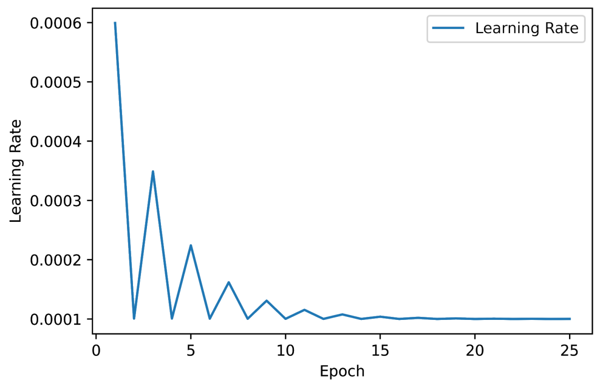

Figure 5. Cyclical learning rate from the 25 epochs of the baseline model training.

4.3 Baseline Model Training

The baseline model was trained for 25 epochs, with cyclical learning rate (as mentioned above). Early Stopping[11] and Model Checkpoint[12] callbacks were used while training, for avoiding overfitting, validation accuracy metric was being monitored by the callbacks. Each epoch took around 4 minutes to train the model on the dataset. The model achieved 33.57% accuracy and 2.85 loss on the validation set and 25.60% accuracy and 3.28 loss on the training set. Figure 6 and Figure 7 show the accuracy curve and the loss curve for the baseline model respectively.

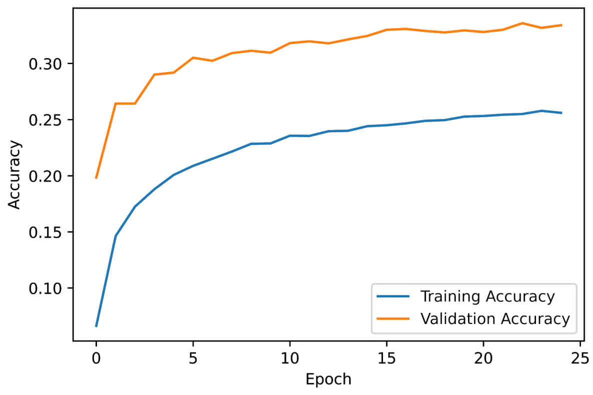

Figure 6. Accuracy curve for the baseline model.

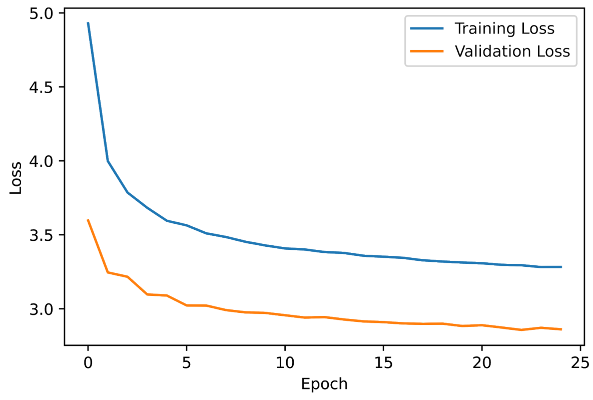

Figure 7. Loss curve for the baseline model.

4.4 Fine Tuning

I unfroze the base model and retrained the whole model end-to-end for 15 epochs for the cycles of learning rate range 1e-5 to 6e-5 with a step size of 1200. Model checkpoint callback was used for saving the best model, the validation accuracy metric was being monitored by the callback. Each epoch took around 3.5 minutes to train the model. The validation accuracy saturated around 53-54%. The accuracy on the validation set got a massive increase of 21%. The model achieved an accuracy of 54.66% and 1.85 loss on the validation set.

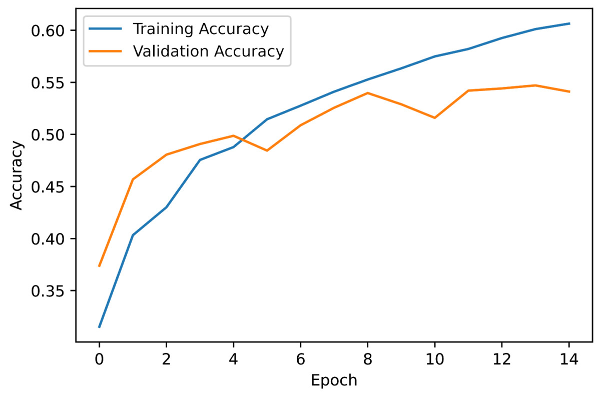

Figure 8. Accuracy curve for the fine-tuned model (Cyclical LR- 1e-5 to 6e-5).

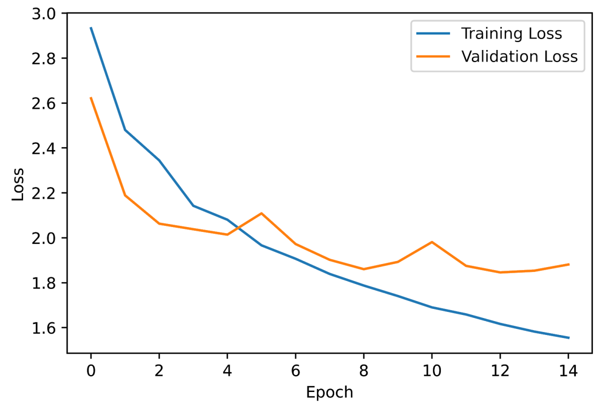

Figure 9. Loss curve for the fine-tuned model (Cyclical LR- 1e-5 to 6e-5).

Further, I trained the model again for 15 more epochs with a reduced cyclical learning rate in the range 1e-6 to 6e-6 with a step size of 1200. Now, the validation accuracy saturated around

55-56%, and the accuracy on the validation set increased by 1.57%. The training accuracy reached 64.88%, so I did not train it further to avoid overfitting. I selected the best model from the training using Model Checkpoint callback which gave an accuracy of 56.23% on the validation dataset. The training details are summarized in Table 1 on the next page.

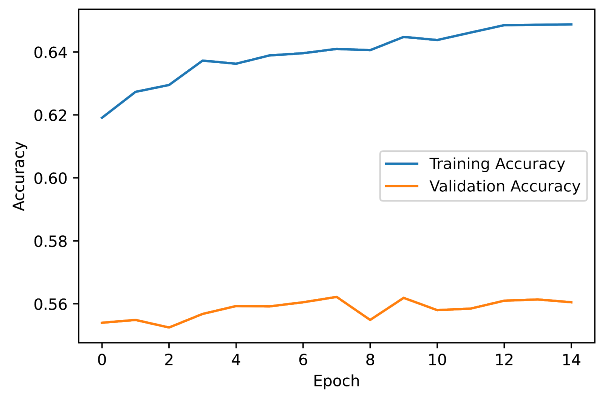

Figure 10. Accuracy curve for the fine-tuned model (Cyclical LR- 1e-6 to 6e-6).

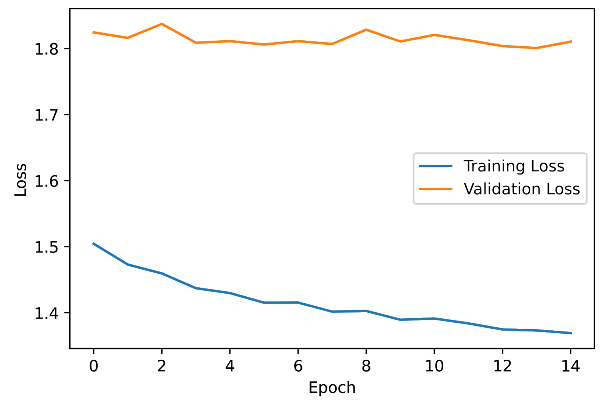

Figure 11. Loss curve for the fine-tuned model (Cyclical LR- 1e-6 to 6e-6).

4.5 Filter Visualization[13]

Visualizing the output of the machine learning model is a great way to see how it's progressing, be it a tree-based model or a large neural network. For understanding how the deep CNN model is able to classify the input image, we need to understand how our model sees the input image by looking at the output of its layers. By doing so, we are able to learn more about the working of these layers.

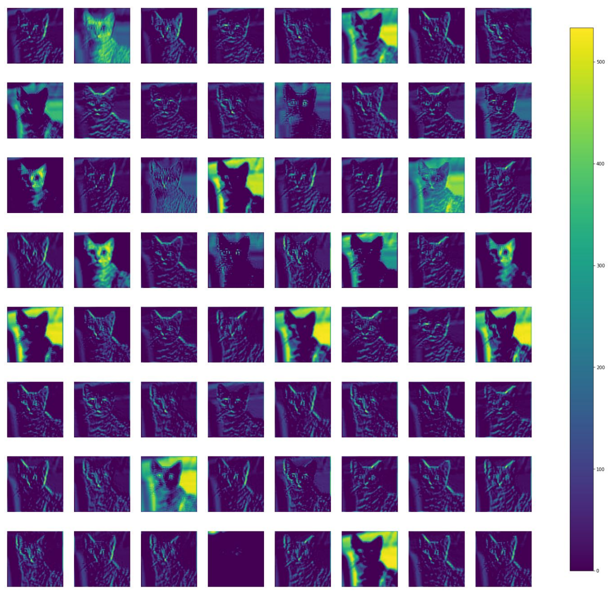

Figure 12. Visualization of first Conv layer

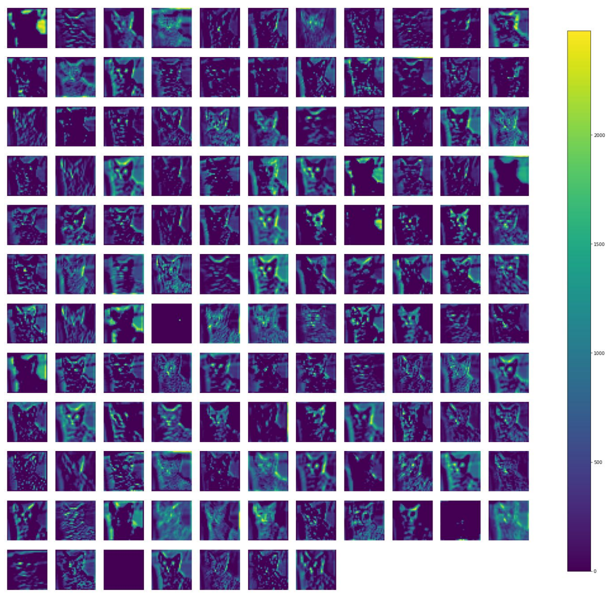

Figure 13. Visualization of third Conv layer

The above figures show the Class Activation Map[14] (CAM) visualization of the filters from the first and third convolution layers of the model. I captured these images by running the trained model on one of the images from the dataset. If we take a look at the different images from convolution layers filters, it is pretty clear to see how different filters in different layers are trying to highlight or activate different parts of the image. Some filters are acting as edge detectors, others are detecting a particular region of the image like its central portion and still others are acting as background detectors. It is easier to see this behavior of convolution layers in starting layers because as we go deeper the pattern captured by the convolution kernel becomes more and more sparse.

5. Results

The final model has achieved an accuracy of 56.23% on the validation set and 64.10% on the training set. The loss is 1.8069 and 1.4016 on the validation and training set respectively. Cyclical Learning Rate helped in the early attainment of higher validation accuracy and steeply bringing down the validation loss. It is also distinctly visible that validation accuracy has increased steeply after fine-tuning, with a cyclical learning rate in range 1e-5 to 6e-5, the model has reached a validation accuracy of 54.66% from 33.57% (a massive increase of 21%). Also, after reducing the cyclical learning rate further, the validation accuracy saturated around 55-56%. Finally, the model has achieved an accuracy of 56.23% on the validation set. The training details are summarized in Table 1.

I also observed that heavy data augmentation, batch normalization, and dropout layers have played a very crucial role in training the model. These techniques made the model more robust to slight variations and hence prevented the model from overfitting, it also helped the model to generalize better.

Base LR	Max LR	Step Size	Start Epoch	End Epoch	Max. Val. Accuracy
1E-04	6E-04	1404	0	25	33.57
1E-05	6E-05	1200	25	40	54.66
1E-06	6E-06	1200	40	55	56.23

Table 1. Training Summary

6. Conclusions

In the present paper, I described my approach to maximizing the accuracy on the image dataset. Training a classifier from scratch is challenging due to the low resolution and low quantity of training images, also because of very high constraints and scarce computation resources, I opted for transfer learning. I designed a custom VGG16 CNN model for the dataset, the model was initialized by the ImageNet weights, except the fully connected layers at the end of the network. For maximizing the accuracy, I have used several data augmentation techniques in order to artificially increase the amount of data and avoid overfitting. I also implemented cyclical learning rate while training, I have used different ranges of cyclical learning rates for the baseline model and fine-tuned model. After implementing all these things, I was able to achieve an accuracy of 56.23% on the validation set.

Efficiently training such a network would require greater computing power, so configuring the network to run on distributed GPUs would be very helpful to reduce training time. For further studies, I will focus on the architecture of the network and aim for achieving higher accuracy.

References

1. ImageNet
   `http://www.image-net.org/`
2. CNN Architectures, by Prof. Vipul Arora
   `https://www.youtube.com/watch?v=P5OgPz_h9CE`
3. Figure 1. ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners
   `https://www.researchgate.net/figure/The-evolution-of-the-winning-entries-on-the-ImageNet-Large-Scale-Visual-Recognition_fig1_321896881`
4. Data Augmentation
   `https://towardsdatascience.com/exploring-image-data-augmentation-with-keras-and-tensorflow-a8162d89b844`
5. K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, arXiv:1409.1556, 2014.
6. Transfer Learning
   `https://keras.io/guides/transfer_learning/`
7. Keras API
   `https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5`
8. Kaggle
   `https://www.kaggle.com/`
9. Kaggle Nvidia Tesla P100 GPU
   `https://www.kaggle.com/docs/efficient-gpu-usage`
10. L. N. Smith, Cyclical Learning Rates for Training Neural Networks, arXiv: 1506.01186v6, 2017
11. Early Stopping
    `https://keras.io/api/callbacks/early_stopping/`
12. Model Checkpoint
    `https://keras.io/api/callbacks/model_checkpoint/`
13. Filter Visualization
    `https://towardsdatascience.com/understanding-your-convolution-network-with-visualizations-a4883441533b`
14. Class Activation Map (CAM)
    `https://towardsdatascience.com/demystifying-convolutional-neural-networks-using-class-activation-maps-fe94eda4cef1`