CS 6910 Fundamentals of Deep Learning

Assignment 3 Analysis Report

June 15, 2020

Report by **Team 19**

ME17B084 Nishant Prabhu, **ME17B068** Soumyadeep Mondal, **ME16B177** Saye Sharan

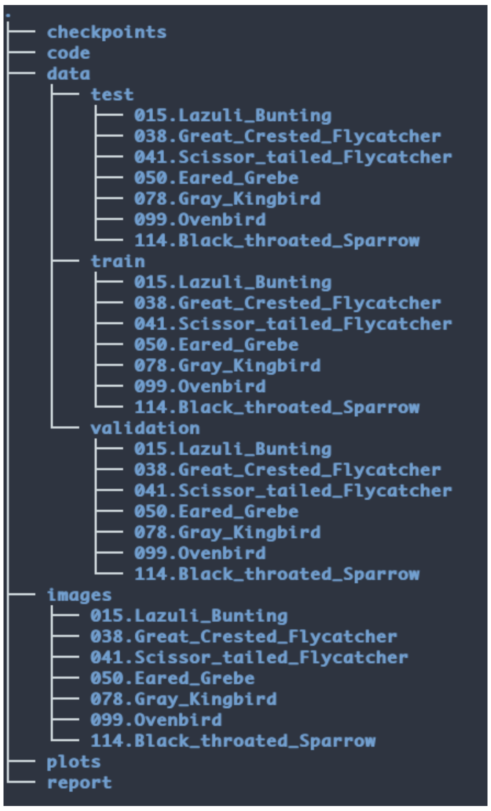
Contents

1 Submission Overview

The submission consists of the following files. PyTorch has been used to construct and train all networks for this assignment. Pretrained networks were initialized with ImageNet weights.

File	**Description**
`data_split.py`	Splits images stored in a root folder into train, test and validation sets in a format that PyTorch's `ImageFolder()` data loader class can process.
`Task1_googlenet.py`	Solution code for task 1 (image classification with pre-trained GoogleNet model)
`Task1_vgg.py`	Solution code for task 1 (image classification with pre-trained VGGNet model)
`Task2.py`	Solution code for task 2 (image classification with custom defined convolutional network)
`Task3.py`	Solution code for task 3 (image classification with custom defined convolutional network and NetVLAD)

If you wish to run the scripts as is, then please organize the directories as shown in **Figure 1**.



```
.
├── checkpoints
├── code
├── data
│   ├── test
│   │   ├── 015.Lazuli_Bunting
│   │   ├── 038.Great_Crested_Flycatcher
│   │   ├── 041.Scissor_tailed_Flycatcher
│   │   ├── 050.Eared_Grebe
│   │   ├── 078.Gray_Kingbird
│   │   ├── 099.Ovenbird
│   │   └── 114.Black_throated_Sparrow
│   ├── train
│   │   ├── 015.Lazuli_Bunting
│   │   ├── 038.Great_Crested_Flycatcher
│   │   ├── 041.Scissor_tailed_Flycatcher
│   │   ├── 050.Eared_Grebe
│   │   ├── 078.Gray_Kingbird
│   │   ├── 099.Ovenbird
│   │   └── 114.Black_throated_Sparrow
│   └── validation
│       ├── 015.Lazuli_Bunting
│       ├── 038.Great_Crested_Flycatcher
│       ├── 041.Scissor_tailed_Flycatcher
│       ├── 050.Eared_Grebe
│       ├── 078.Gray_Kingbird
│       ├── 099.Ovenbird
│       └── 114.Black_throated_Sparrow
├── images
│   ├── 015.Lazuli_Bunting
│   ├── 038.Great_Crested_Flycatcher
│   ├── 041.Scissor_tailed_Flycatcher
│   ├── 050.Eared_Grebe
│   ├── 078.Gray_Kingbird
│   ├── 099.Ovenbird
│   └── 114.Black_throated_Sparrow
├── plots
└── report
```

Figure 1: Directory tree

The folders `images` and `report` are not necessary for the code to run.

2 Deep convolutional networks for image classification

For this task, we were provided an image dataset with images belonging to one of seven classes. Classification was to be performed using pre-trained deep convolutional networks (GoogleNet and VGGNet specifically). We imported each of these networks initialized with ImageNet weights from PyTorch's `torchvision` module without the output layer (default 1000 classes for ImageNet).

All images were resized to 256, center-cropped to size 224, and normalized with means [0.485, 0.456, 0.406] and standard deviations [0.229, 0.224, 0.225] across channels RGB respectively. These parameters were found to give optimal performance in ImageNet classification, thus used here.

2.1 GoogleNet

GoogleNet generates features of size (1, 1024) for every image, which is passed on to a dense layer with 7 units and softmax activation for classification. **Figure 2** summarizes the model architecture used for this sub-task. The output layer used by us has not been shown here.

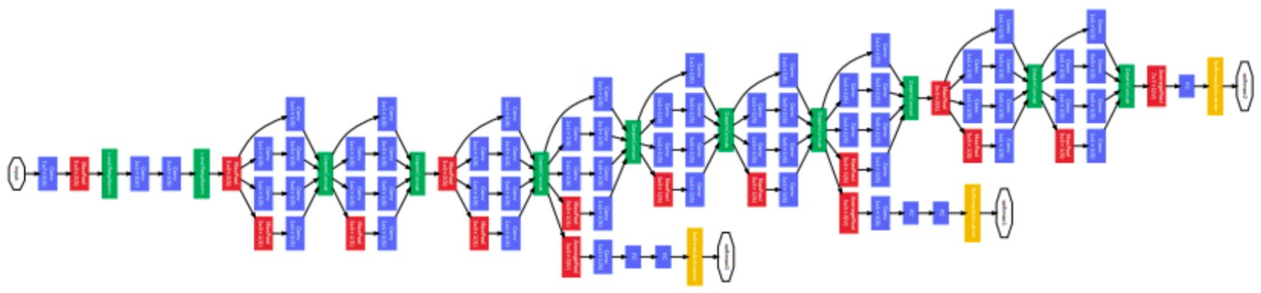


Figure 2: GoogleNet for single-label multi-class classification

Image credit: PyTorch

Here are the results after training the model for 20 epochs. The data was split 70% for training, 20% for validation and 10% for testing. The network had 7,168 trainable parameters and 291 images to train on.

	Logloss	**Top 1 accuracy**	**Top 3 accuracy**
Training	0.2897	93.47%	100.00%
Validation	0.4072	88.09%	97.62%
Testing	0.4880	86.90%	96.43%

Figures 3 shows the logloss and **Figure 4** shows the top 1 accuracy and top 3 accuracy trends as training progressed.

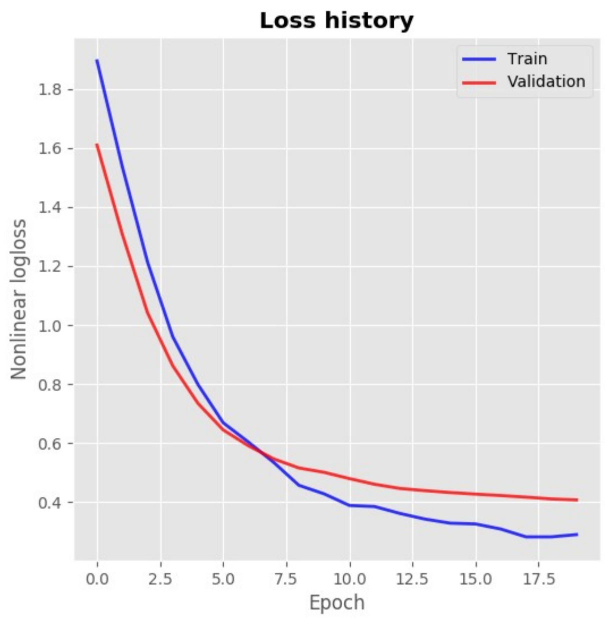


Figure 3: Logloss history for GoogleNet over 20 epochs

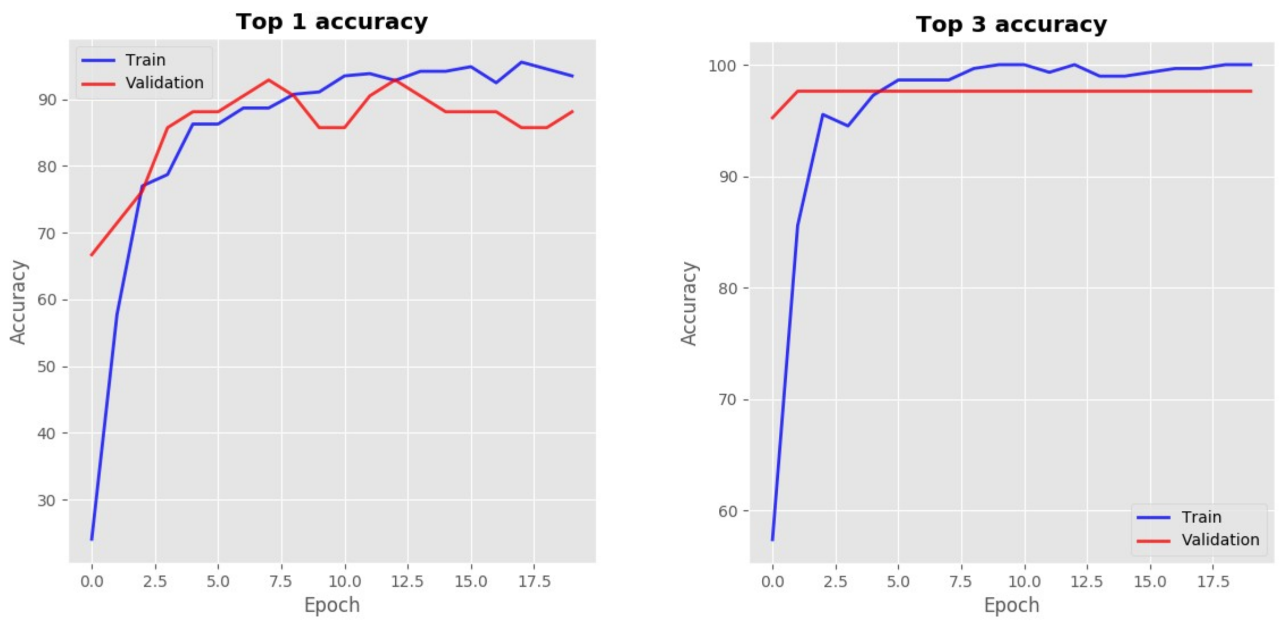


Figure 4: Top 1 accuracy and top 3 accuracy for GoogleNet over 20 epochs

2.2 VGGNet

VGGNet generates features of size (1, 4096) for every image which is passed to a dense layer with 7 units and softmax activation for classification. **Figure 5** summarizes the model architecture used for this sub-task.

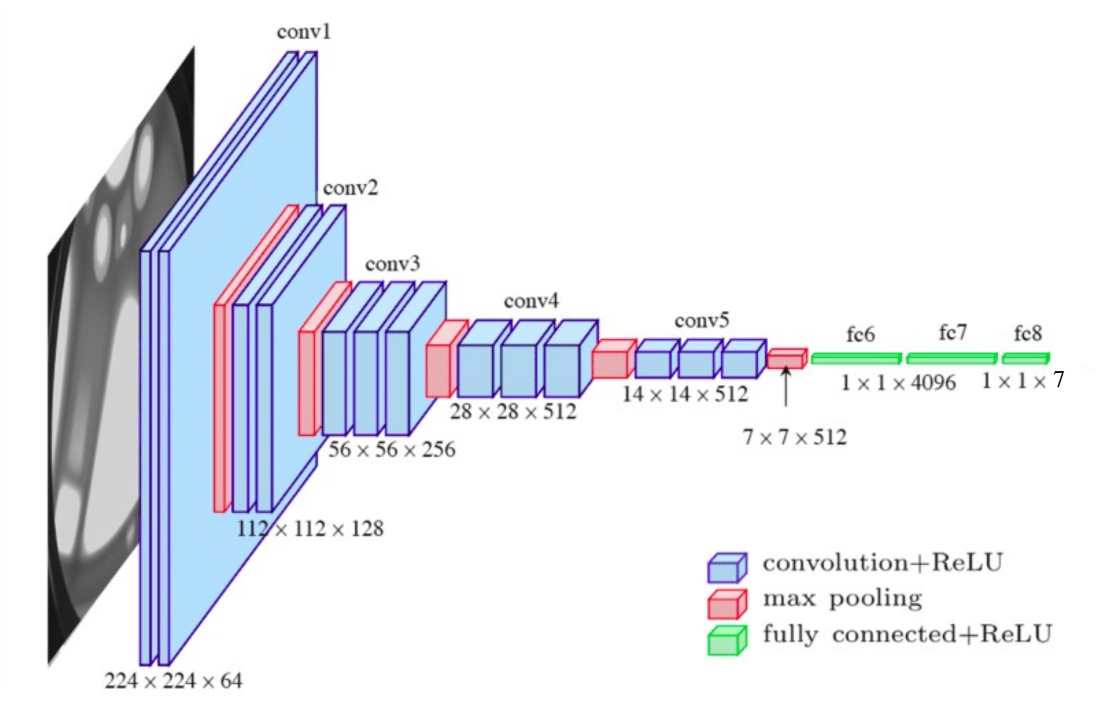


Figure 5: VGGNet for single-label multi-class classification

Image credit: ResearchGate

Here are the results after training the model for 20 epochs. The data was split 70% for training, 20% for validation and 10% for testing. The network had 28,672 trainable parameters and 291 images to train on.

	Logloss	**Top 1 accuracy**	**Top 3 accuracy**
Training	0.0005	100.00%	100.00%
Validation	1.0316	80.95%	97.62%
Testing	0.9658	77.38%	95.24%

Figure 6 shows logloss and **Figure 7** shows top 1 accuracy and top 3 accuracy trends as training progressed.

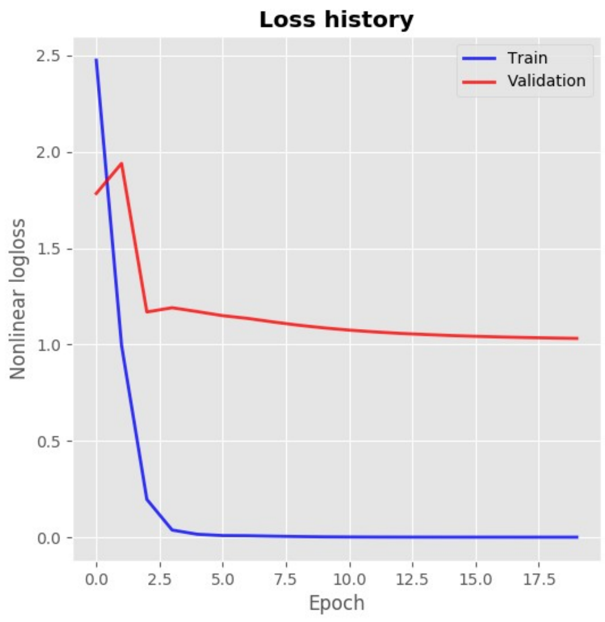


Figure 6: Logloss history for VGGNet over 20 epochs

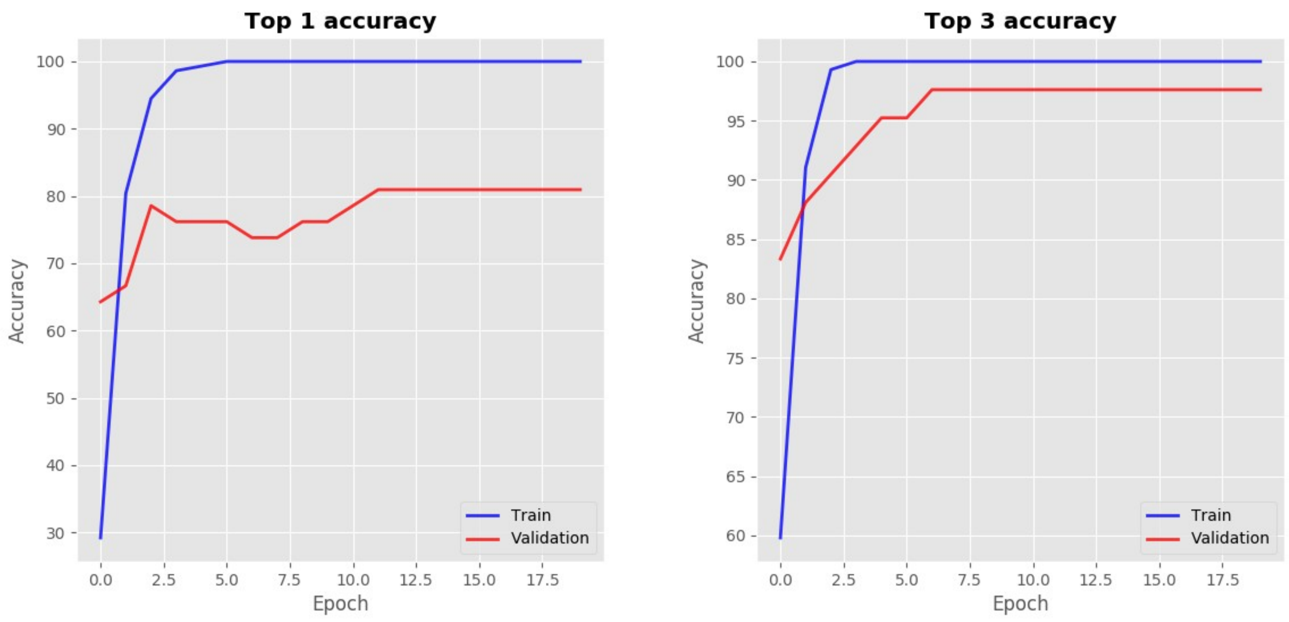


Figure 7: Top 1 accuracy and top 3 accuracy for VGGNet over 20 epochs

2.3 Comparison of results

Here are some observations we find important about training the two models above and the results that were generated by both.

- *Generalization.* GoogleNet seems to generalize on the data much better than VGGNet does. This is apparent from the gap in training and validation logloss for VGGNet, which is much smaller and acceptable for GoogleNet (also, this is seen from the disparity between training, validation and testing accuracy for VGGNet). This is likely due to the smaller number of parameters to train in the case of GoogleNet (one-fourth as many as those of VGGNet).
- *Training time*. Despite being a deeper network, GoogleNet's training time is significantly smaller than that of VGGNet. This can have two reasons: (1) smaller number of parameters to train for GoogleNet, so the backpropagation step is faster, or (2) forward pass times are smaller for GoogleNet owing to the Inception module, which greatly reduces the number of calculations to be performed. The second argument, however, is weak as the difference in structure of both networks (especially depth) is significant.
- *Convergence time and stability*. VGGNet converges rapidly (within 5 epochs) but results in poor generalization and relatively unstable learning. GoogleNet has a much larger learning curve, but it is smooth and generalization is good.

Considering these, the inception module based GoogleNet seems to be a better choice for this task compared to VGGNet. InceptionV3 is a popular choice for many deep learning based image processing tasks today.

We also trained a ResNet50 model on the data (since it was a recent winner in ILSVRC), and here are the results after 20 epochs.

	Logloss	**Top 1 accuracy**	**Top 3 accuracy**
Training	0.1447	98.97%	100.00%
Validation	0.2758	95.24%	100.00%
Testing	0.3347	94.05%	97.62%

Clearly, the residual network has outperformed GoogleNet by a large margin. However it is worth noting that this model took longer to train than GoogleNet did (but not as long as VGGNet). The loss and accuracy trends (not shown here) revealed learning curves smoother than those of GoogleNet, with very less gap between training and validation metrics.

3 Shallow convolutional networks for image classification

For this task, we were asked to perform single-label multi-class classification with a shallow convolutional network, whose configuration was provided to us. **Figure 8** summarizes the architecture of the network.

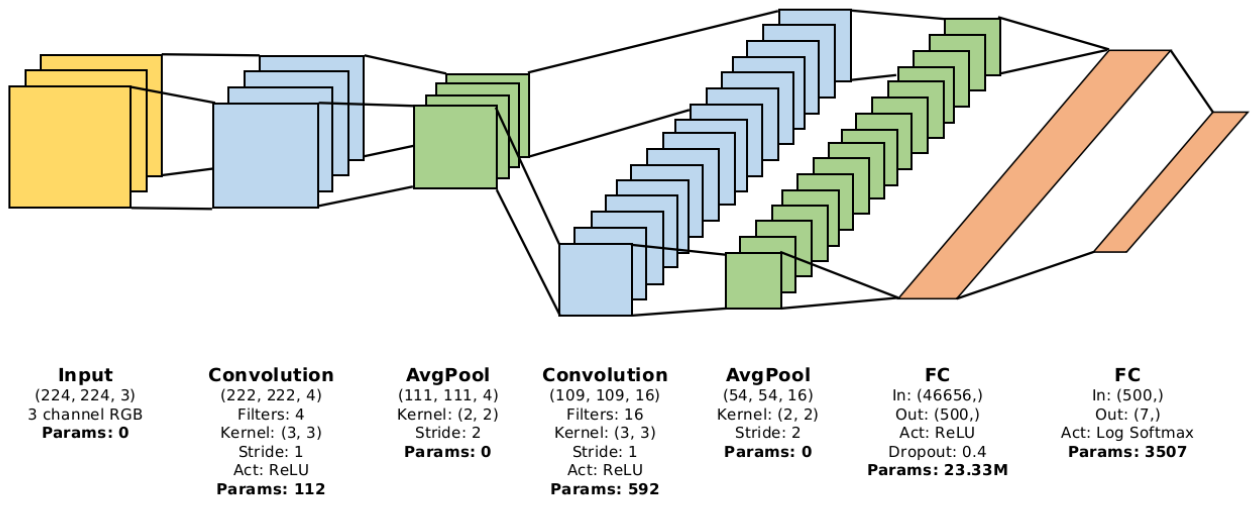


Figure 8: Model architecture for image classification with shallow convolutional network

3.1 Preprocessing

Each image was resized to (256, 256, 3), then center-cropped to (224, 224, 3) and finally mean-variance normalized with means [0.485, 0.456, 0.406] and standard deviations [0.229, 0.224, 0.225] across channels RGB respectively. These parameters were seen to give optimal results for ImageNet classification, hence used here.

3.2 Training results

Here are the results of training the model for 100 epochs.

	Logloss	**Top 1 accuracy**	**Top 3 accuracy**
Training	0.7122	84.19%	97.25%
Validation	1.4634	40.48%	73.81%
Testing	1.3038	47.62%	79.76%

Figure 9 shows logloss and **Figure 10** shows top 1 accuracy and top 3 accuracy trends of the model as training progressed.

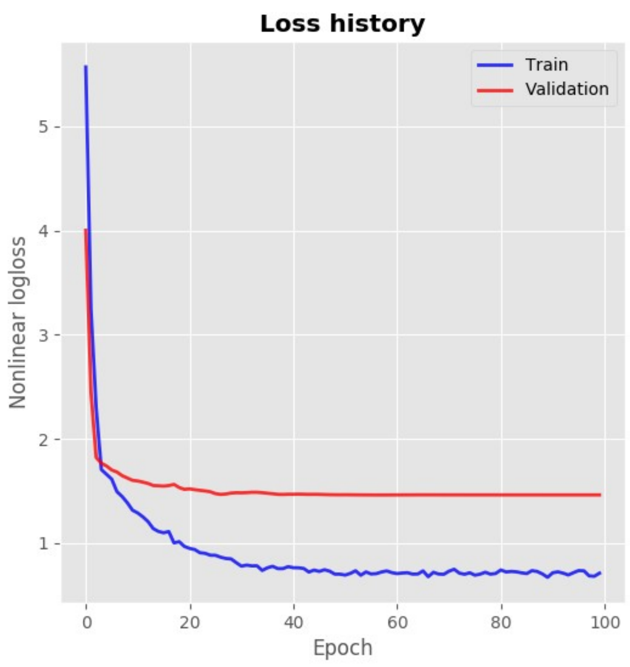


Figure 9: Logloss history for network over 100 epochs

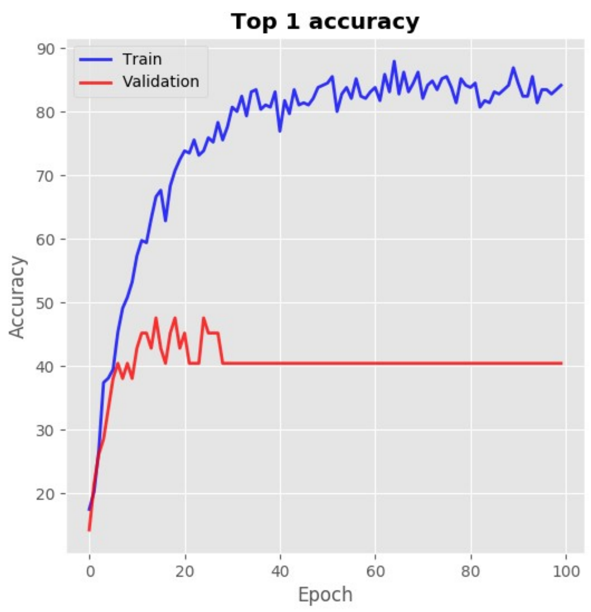
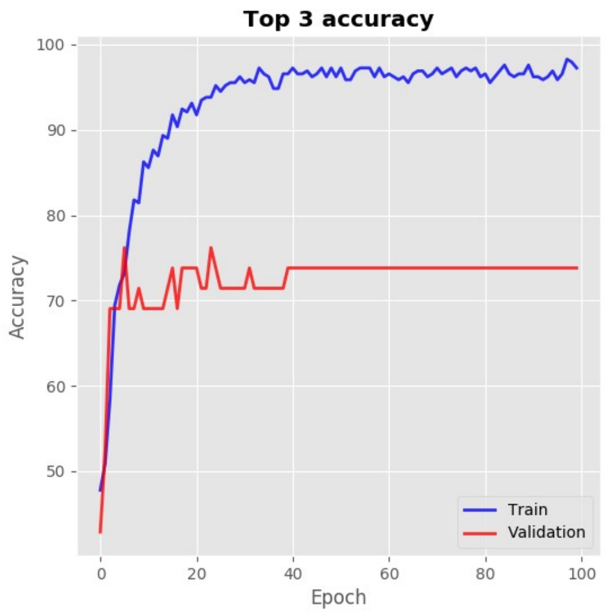


Figure 10: Top 1 accuracy and top 3 accuracy trends for network over 100 epochs

The model has clearly generalized very poorly; while the training accuracy reaches close to 90%, validation accuracy struggles to rise above 40%. This is perhaps due to the large number of parameters the model has to train (over 23 million) with only 291 images. Reducing the number of parameters (by keeping only one fully-connected layer) also does not help much, since we still have a large number of parameters to tune with a small training dataset. One way to solve this issue would be to use image augmentation, by which new images are generated by adding random shifts, rotations or skews to the target image. This model does not perform as good as the pre-trained models used earlier because its weights initialization is random and depth is small.

4 Shallow ConvNet with NetVLAD from image classification

4.1 Brief overview of NetVLAD layer

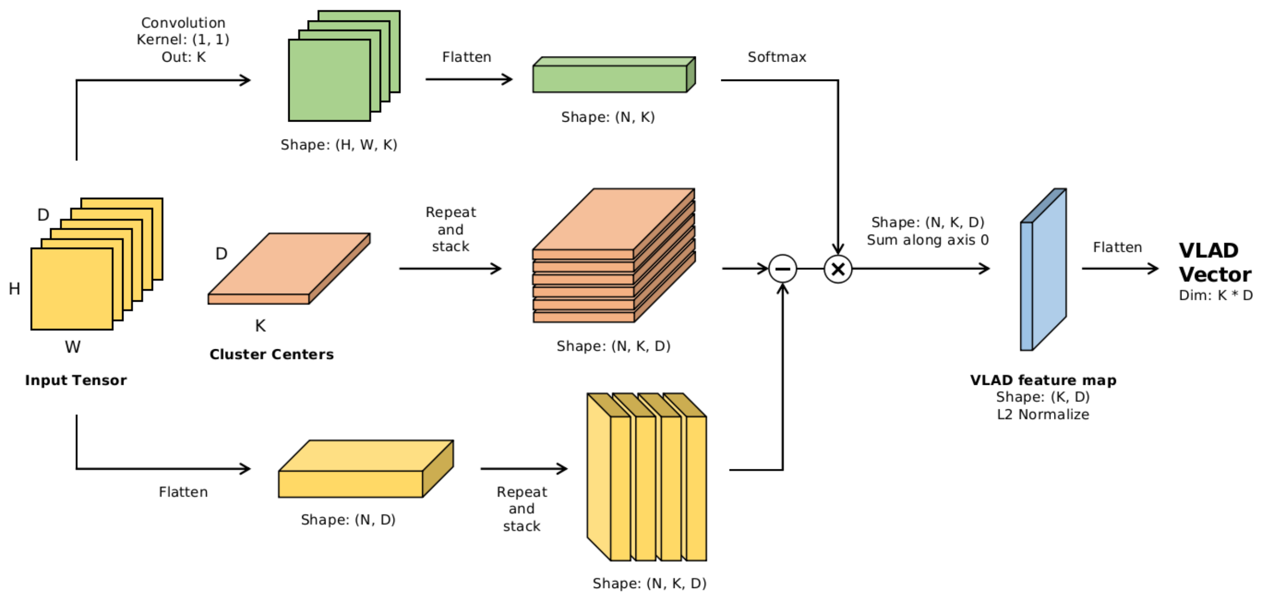


Figure 11: Functioning of the NetVLAD layer

NetVLAD takes in the output from the last convolutional (or pooling) layer in the convolutional network. In general, its input is of size (H, W, D), which can be seen as $N = H * W$ descriptors, each of dimension D. Also, it initializes some codebook vectors (K in number), which we will call *cluster centers*, each of dimension D as well. Intuitively, this layer generates features that quantify the closeness of each of these N descriptors with the K cluster centers.

$$V(j,k) = \sum_{i=1}^{N} a_k(\mathbf{x}_i)\,(x_i(j) - c_k(j))$$

NetVLAD transformation; adapted from this publication

The weights $a_k(x_i)$ are determined as shown in the equation below. Mathematically, this expression can be reduced to a convolution + softmax operation, as performed in **Figure 11**.

$$\bar{a}_k(\mathbf{x}_i) = \frac{e^{-\alpha \|\mathbf{x}_i - \mathbf{c}_k\|^2}}{\sum_{k'} e^{-\alpha \|\mathbf{x}_i - \mathbf{c}_{k'}\|^2}}$$

Soft assignment of cluster membership; adapted from this publication

The residual of each descriptor with each cluster center is computed and multiplied with its corresponding weight, as computed above. Finally, it is summed over all descriptors to obtain the final VLAD feature map. This is reduced to a form acceptable by the subsequent fully-connected layers of the network.

4.2 Preprocessing

Each image was resized to (256, 256, 3), then center-cropped to (224, 224, 3) and finally mean-variance normalized with means [0.485, 0.456, 0.406] and standard deviations [0.229, 0.224, 0.225] across channels RGB respectively. These parameters were seen to give optimal results for ImageNet classification, hence used here.

4.3 Training results

Here are the results of training the model for 100 epochs.

	Logloss	**Top 1 accuracy**	**Top 3 accuracy**
Training	1.9054	26.00%	62.31%
Validation	1.9287	30.95%	40.48%
Testing	1.9338	29.24%	40.48%

Visibly, the performance of this systems is considerably poor compared to the previous network. One would expect the addition of this layer to aid the network in classification, but there could be a number of other reasons (some of which have been state previously) that resulted in this.

- Small size of the training dataset, and more parameters to train in this network compared to the previous network.
- Small number of clusters in the NetVLAD layer than necessary for better representation of features (we tried increasing the number of clusters but it did not help much).
- Correctness (or otherwise) of our implementation of NetVLAD.

In **Figure 12** and **Figure 13**, logloss and accuracy trends of the model are shown.

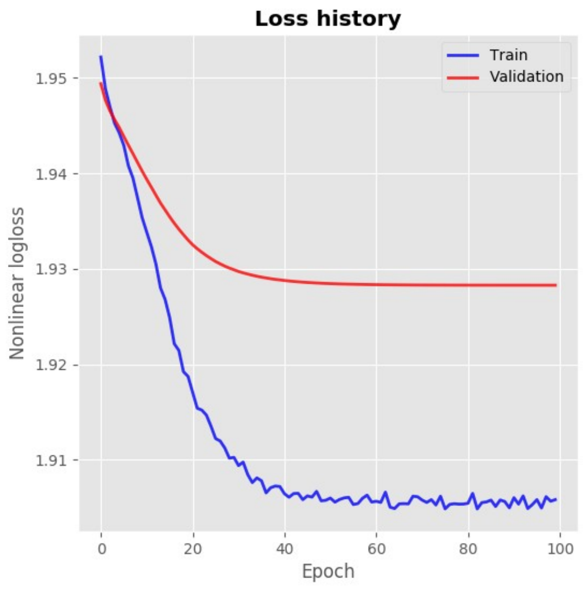
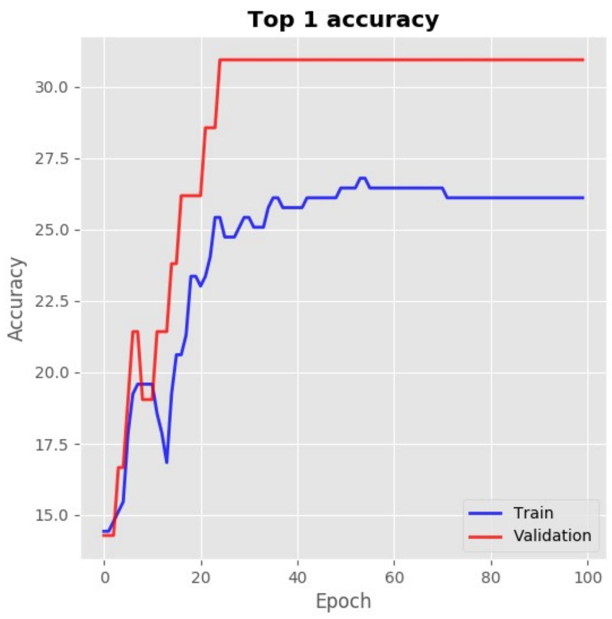


Figure 12: Logloss trends for network over 100 epochs



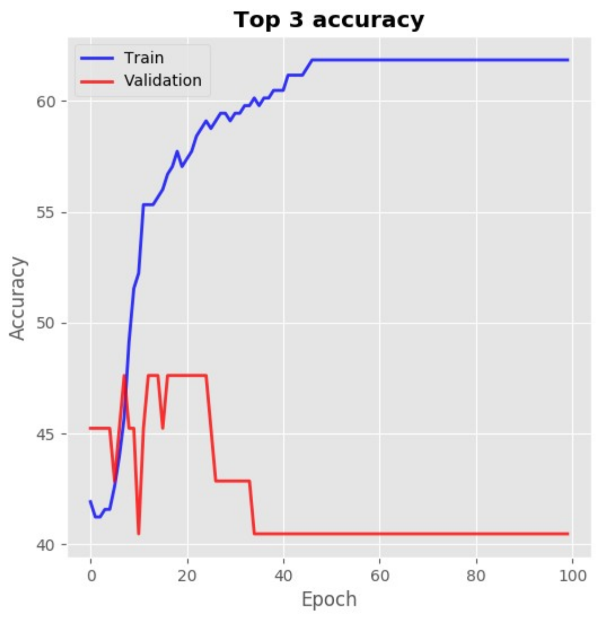


Figure 13: Top 1 accuracy and top 3 accuracy trends for network over 100 epochs

The plots also indicate towards model overfitting, which reinforces one of the points that we raised earlier. It is interesting to note (and probably a fluke) that validation accuracy of the system was far above training accuracy.

5 End note

Pretrained models were our best performers on the data, with ResNet50 performing better than GoogleNet, both of which did better than VGGNet. This is in line with their standings in ILSVRC across the years. As for custom convolutional networks, they did not perform anywhere close to how pre-trained models did, but that was expected. Addition of the NetVLAD layer caused a significant drop in performance for the network, which is something we will investigate in the future (no promises).

6 References

- Using pre-trained GoogleNet in PyTorch, **PyTorch**
- Using pre-trained VGGNet in PyTorch, **PyTorch**
- Using pre-trained ResNet in PyTorch, **PyTorch**
- NetVLAD: CNN architecture for weakly supervised place recognition, **arXiv**
- Implementing NetVLAD in PyTorch, ideas taken from **Nanne's GitHub**