Image Processing and Deep Learning Assignment

Final Report

Convolutional Neural Networks

T S P K Sainath(6668049)

A report submitted in part fulfilment of the course of

Image Processing and Deep Learning (EEEM063)

Department of Electronic Engineering
Faculty of Engineering and Physical Sciences
University of Surrey
Guildford, Surrey, GU2 7XH, UK

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 1489

Student Name: T S P K Sainath

Date of Submission: 04 May 2021.

Table of Contents

Introduction

In this assignment we have a look at the various experiments to understand how various networks and hyperparameters affect the performance of classification using various networks.

CNN Architecture

Digits server architectures used:

- AlexNet
- GoogLeNet

Additional architectures tested(using Google Colab):

- InceptionV3
- ResNet-50V2
- ResNet-152V2
- EfficientNetB7
- VGG-16
- FlashNet(own network)

We will have a look at the various networks trained and used through the different experiments to test various parameters. I have not tried all the permutations and combinations as its pointless and a waste of resources instead we pick experiments that would cover all the various parameters and provide solid conclusions while not conducting unnecessary experiments.

Test Data

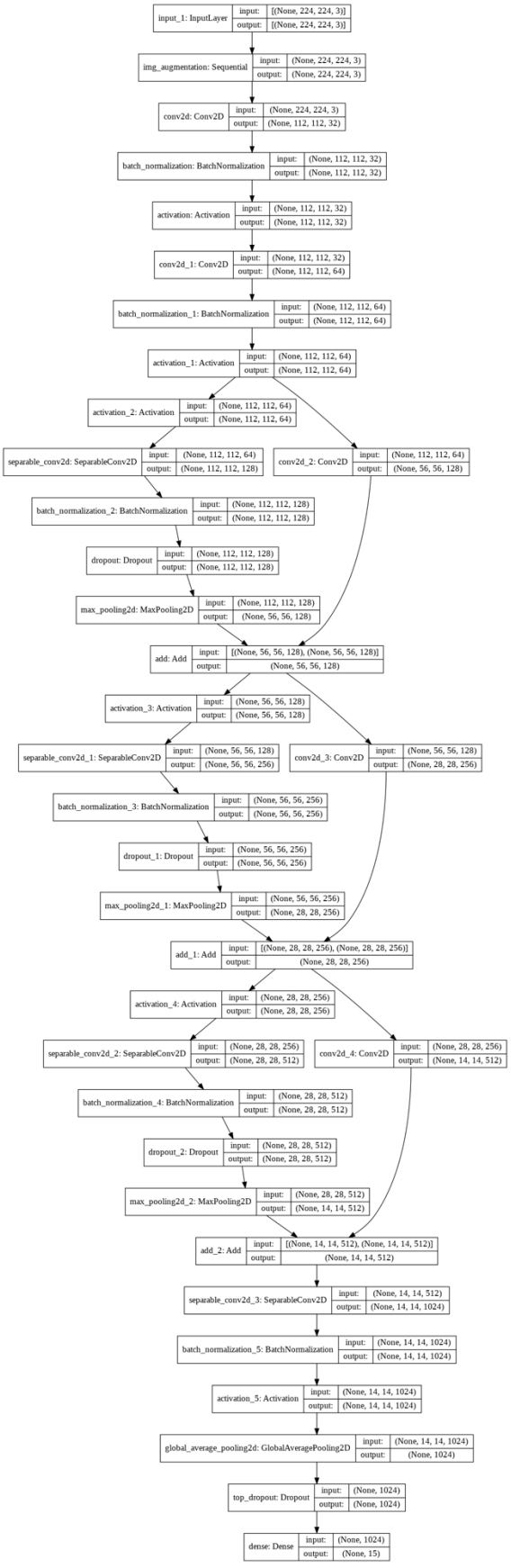
Cars dataset

Network	Epochs	Learning rate	Test Accuracy
AlexNet	45	0.01	36.2%
GoogLeNet	100	0.01 Sigmoid	70.17%
InceptionV3(Transfer learning)	45	0.01	87.28%
EfficientNetB7(Transfer learning)	40	0.01	89.76%
ResNet-50V2(Transfer learning)	30	0.01	86.15%
ResNet-152V2(Transfer learning)	20	0.01	86.48%

Plant Village dataset

Network	Epochs	Learning rate	Test Accuracy
VGG-16(Transfer learning)	45	0.01	90.55%
ResNet-50V2(No Transfer learning)	50	0.01	98.12%
ResNet-50V2(Transfer learning)	30	0.01	99.32%
FlashNet	50	0.01	66.37%

FlashNet – own implementation



We take inspiration from Xception (Chollet, 2017) network and create our own network. While not very complex, the network does offer decent classification performance. We use dropout in middle in an attempt to try and improve the performance.

We use the following parameters to train:
Split: [0.8,0.1,0.1]
Transfer Learning: False
Architecture: FlashNet
Batch_size: 128
Dataset: PlantVillage
Epochs: 50
Learning_rate: 0.01
Loss_function: categorical_crossentropy
Optimizer: SGD

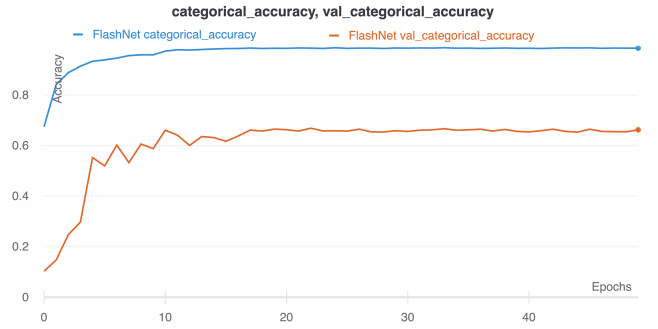


Figure 1: Training and Validation Accuracy

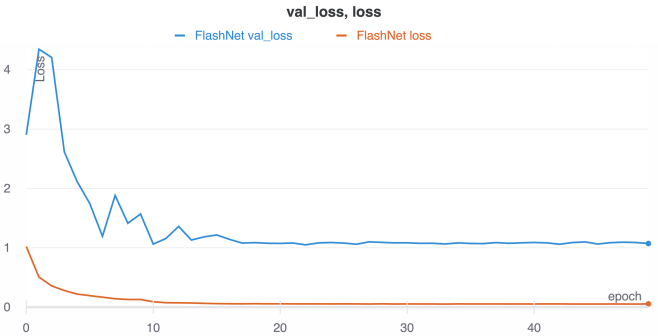


Figure 2: Training and Validation Accuracy

Best val_loss	1.05
categorical_accuracy	0.9855
val_categorical_accuracy	0.6623
test_categorical_accuracy	0.6637
loss	0.05438
val_loss	1.072
test_loss	1.052

Dataset pre-processing and preparation

We have used cars data set for all experiments on the Digits server for convenience. But since the cars dataset is not the best data set due to low image count per class, I have prepared Plant Village dataset with 15 classes and a total of 20639 Images giving much better number of images per class than the cars dataset hence better for training a network with.

The classes in Plant Village are:

- Pepper bell Bacterial spot
- Pepper bell healthy
- Potato Early blight
- Potato healthy
- Potato Late blight
- Tomato Target Spot
- Tomato mosaic virus
- Tomato YellowLeaf Curl Virus
- Tomato Bacterial spot
- Tomato Early blight
- Tomato healthy
- Tomato Late blight
- Tomato Leaf Mold
- Tomato Septoria leaf spot
- Tomato Spider mites

The dataset is available here.

Data Augmentation experiment

On the digits server I have not tried to perform data augmentation as its non-intuitive. While using Keras, we perform data augmentation as follows.

```
img_augmentation = Sequential(
    [
        preprocessing.RandomRotation(factor=0.15),
        preprocessing.RandomTranslation(height_factor=0.1,
width_factor=0.1),
        preprocessing.RandomFlip(),
        preprocessing.RandomContrast(factor=0.1),
    ],
    name="img_augmentation",
)
```

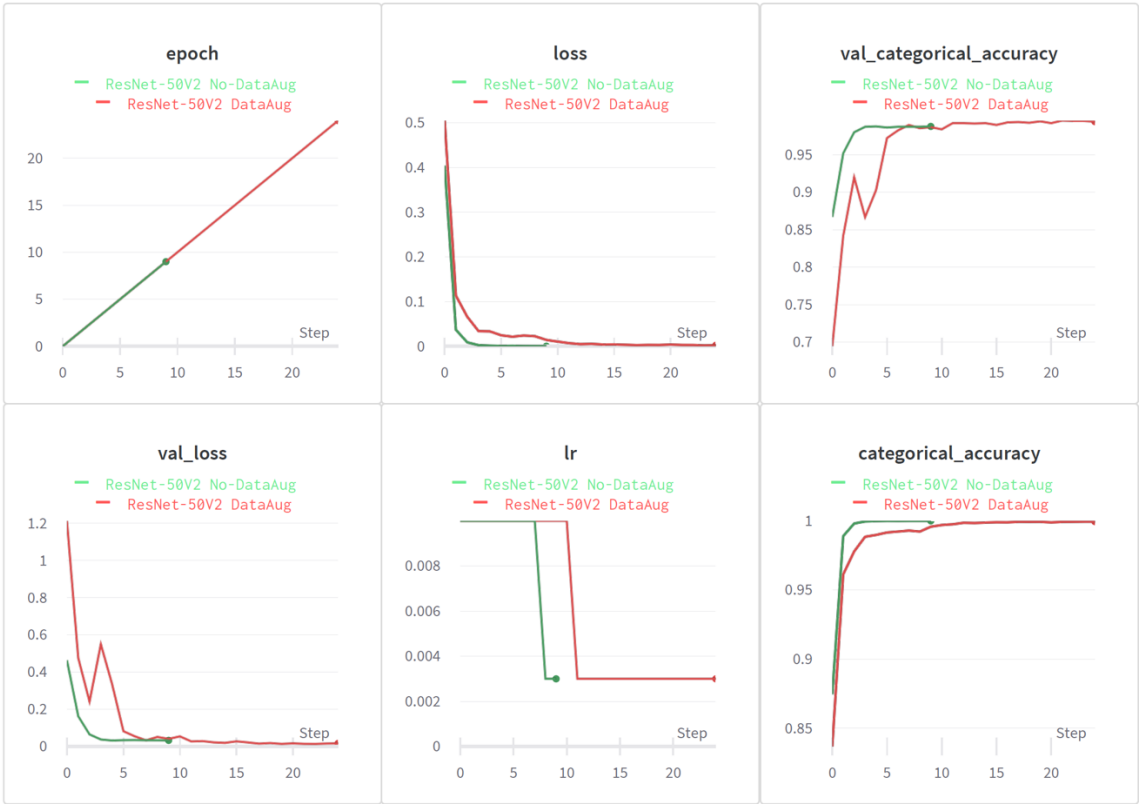


Figure 1: ResNet-50V2 with Plant Village dataset and Transfer Learning

We also perform scaling operation to normalise all values to between 0 and 1. This helps especially in cars dataset which performed really poorly without the scaling.

```
data_augmentation = keras.Sequential(
    [
        preprocessing.Rescaling(scale=1./255.)
    ]
)
```

We have used Image Augmentation for all experiments even though in our Plant Village dataset, it did not make much difference. Rescaling is done directly on all full dataset irrespective, to have better learning.

Data splits

We test 90-9-1, 80-10-10 and 70-15-15 Train, validation and test sets.

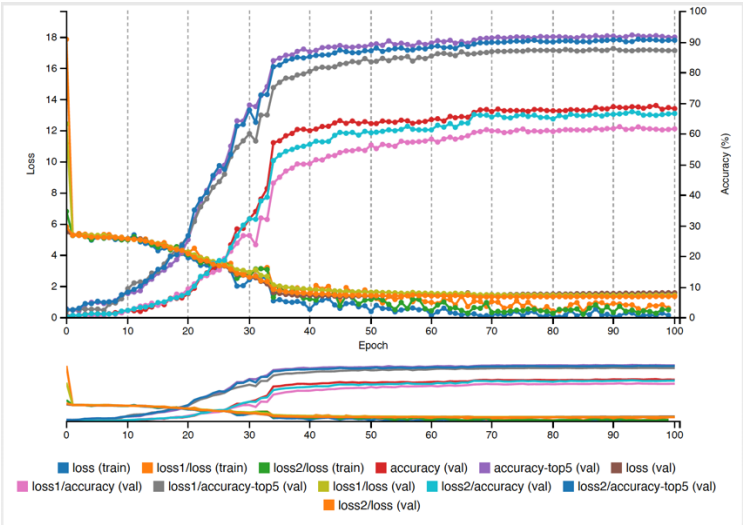


Figure 2: 90:9:1 Train:Val:Test set

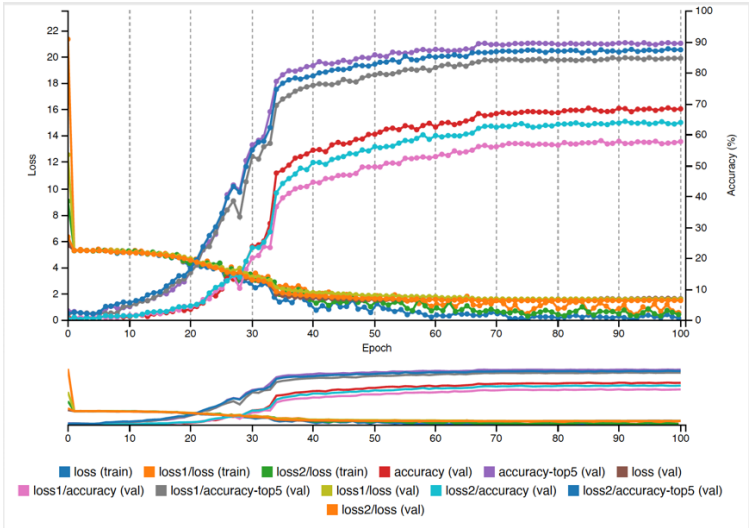


Figure 3: 80:10:10 Train:Val:Test set

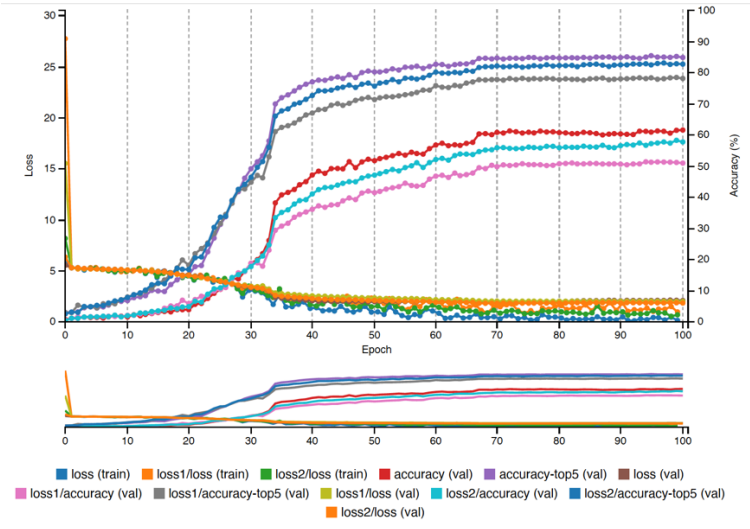


Figure 4: 70:15:15 Train:Val:Test set

Exploration of Metaparameters

The various parameters we test include:

- Learning rates
- Optimizers
- Learning rate decay
- Epochs

Learning rate

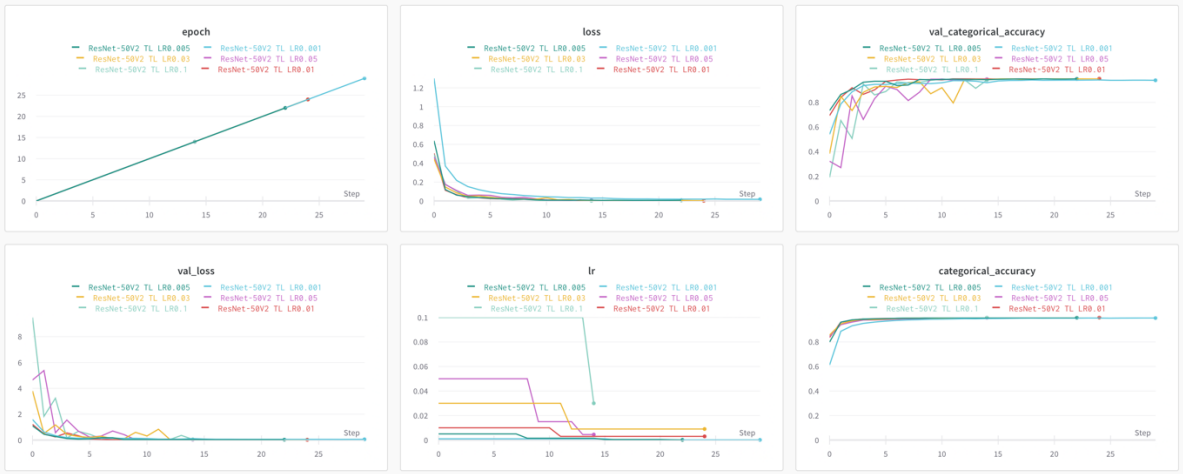


Figure 5: Learning rate on ResNet-50V2

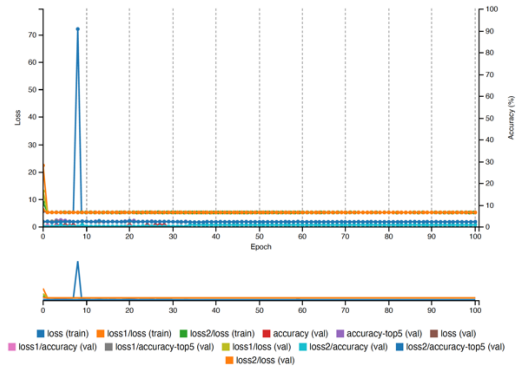
We have test LR of 0.1, 0.05, 0.03, 0.01, 0.005 and 0.001 on ResNet-50V2 on Plant-Village Dataset using transfer learning. Click here to view interactive charts.

We notice that lower learning rate takes longer(more epochs) to reach similar performance. We need to have an optimal LR to be able to prevent overfitting and to be able to train the network in time. At the same time, having too high learning rate can cause the network not to converge, leading to the loss becoming NAN.

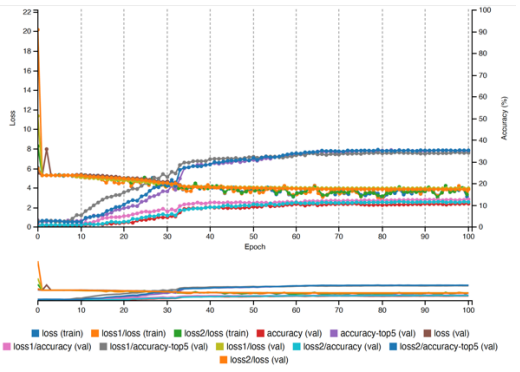
We picked the default learning rate of 0.01 for all our experiments further on. Other good learning rates are 0.005 and 0.03.

Optimizers:

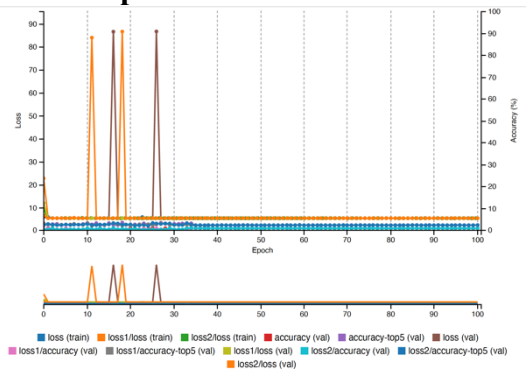
ADAM



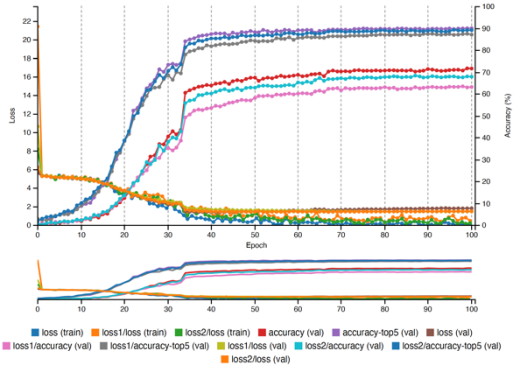
AdaGrad



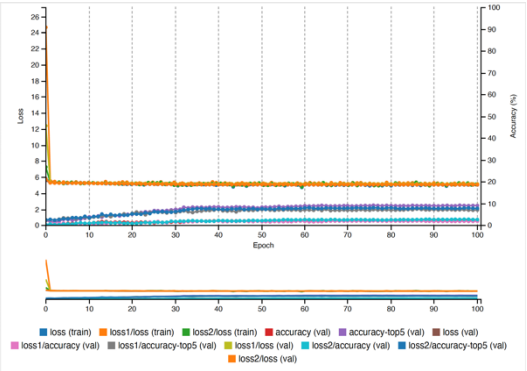
RMSProp



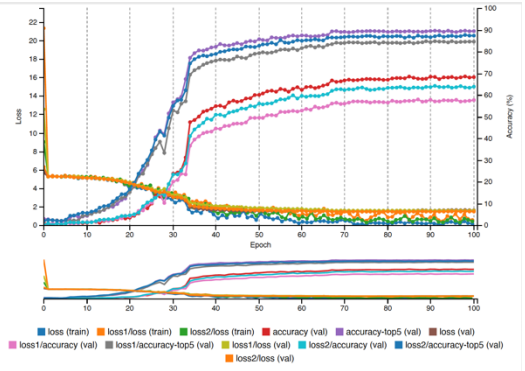
NAG



AdaDelta



SGD

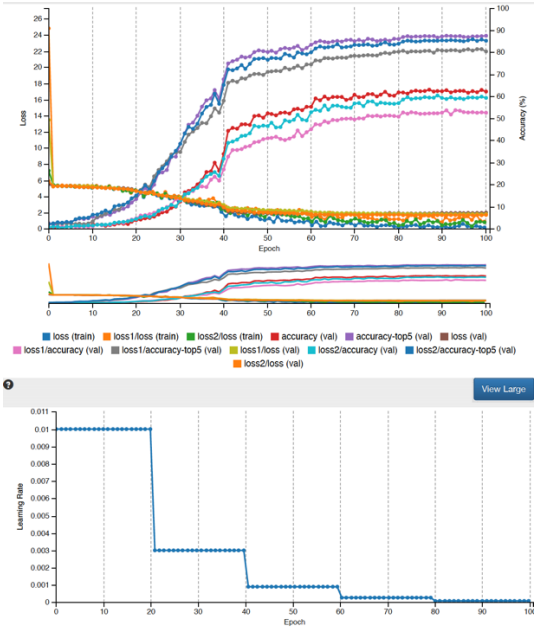


We can see that selecting the correct Optimiser is important for the network. We can see Adam, RMSProp and AdaDelta not converging at all. SGD and NAG appear to be the best optimisers for our data. We choose **SGD** for all our other experiments.

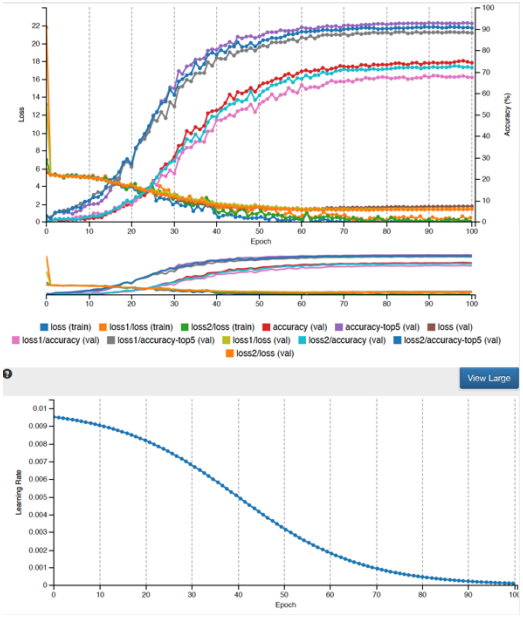
Learning Rate Decay

Learning rate has impact on performance as seen from the previous experiments. So we can utilise that to improve the learning of the network. We can start at higher learning rate and use some function to reduce the learning rate as we go into the training to improve the training of the network.

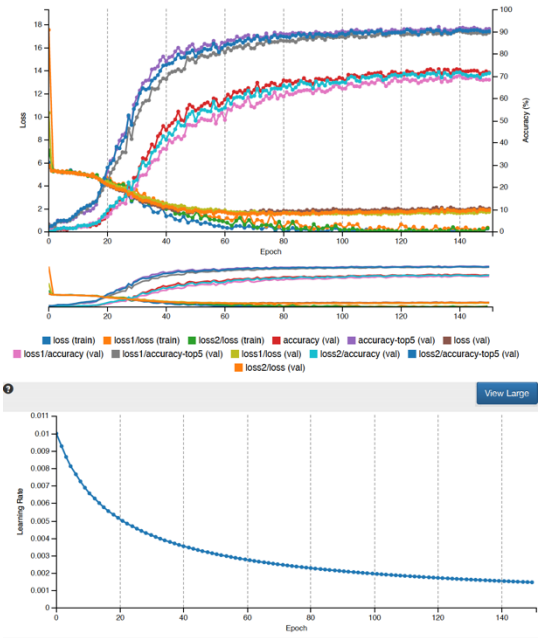
Stepwise(Step size=20%, Gamma=0.3)



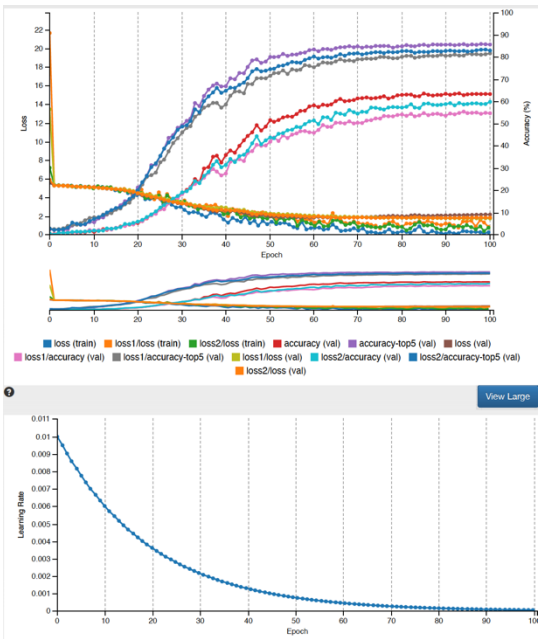
Sigmoid Decay(Step=40%, G=0.075)



Inverse Decay(Pow=0.8, Gamma=0.1)



Exponential Decay (Gamma=0.95)



Sigmoid Decay as the best performance(validation loss and validation accuracy), but on our experiments we use stepwise with conditions instead of fixed steps, we wait for val_loss to stop decreasing(patience of 3 epochs) before reducing the learning rate by a factor of 0.3(can be seen in the LR experiments).

Batch Size

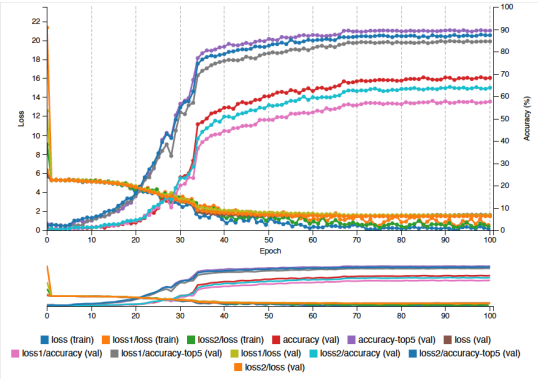


Figure 6: Batch size-32

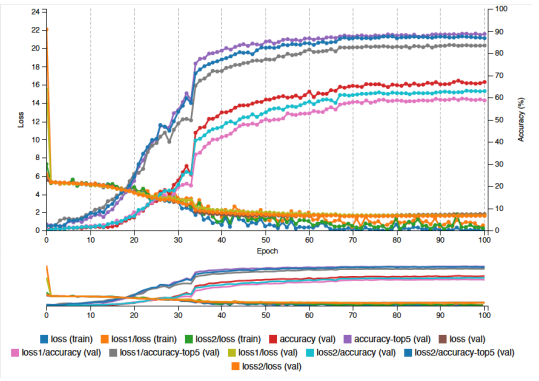


Figure 7: Batch size-16

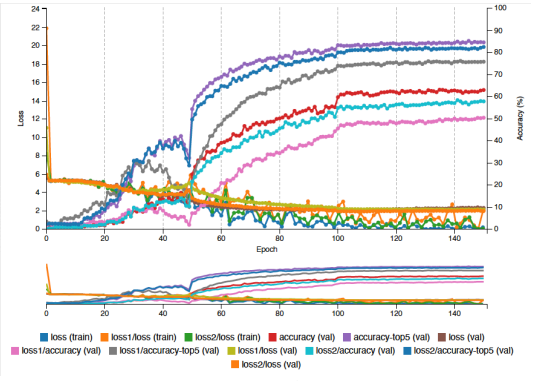


Figure 8: Batch size-8

We observe that batch size doesn't really play as much role in training, but we can see that 32 batch size performed the best, this is due to larger batches at once gives more information to learn per iteration.

Lower batch size must only be used if you run out of memory on GPU, since lower batch sizes take longer to train due to the memory access penalty.

For the rest of the experiments, we have used Batch sizes as large as 196 as we used Telsa V100 SXM2 16GB instead of the RTX 2080 or the Titan Xp on the Digits server which are limited by the VRAM capabilities.

Tesla V100 has HBM2 memory at bandwidth of 900 GB/s hence utilizing the maximum possible VRAM gives us the best performance from it.

Extra work

We try and utilise transfer learning to train our deep neural networks.

Transfer learning: Transfer learning is a technique used in deep neural networks where we take a pretrained network (ResNet-50V2) on Imagenet dataset. We then remove the top of the network and reinitiate them with our own fully connected layer and then start the training. Few people freeze the network and only train the top and then unfreeze and train the whole. This technique of freezing is mainly used due to limitation in compute power and the requirement of lower epochs, but since compute power isn't a problem due to the use of Titan V100, we go ahead unfreeze the whole network and train it for a few epochs.

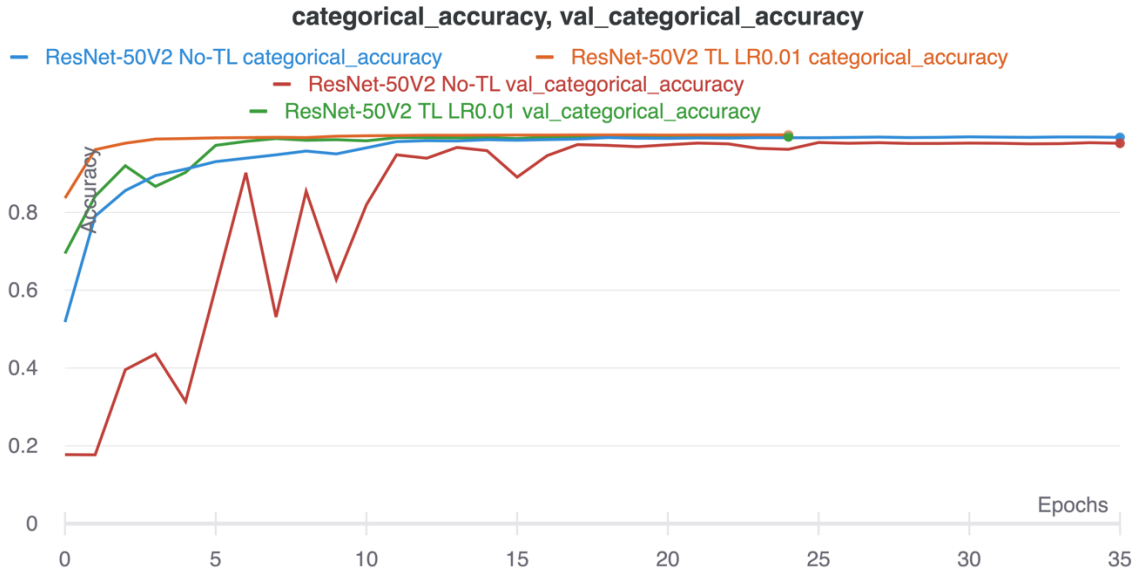


Figure 11: Training and Validation Accuracy on No-TL(No transfer learning) vs TL(Transfer learning)

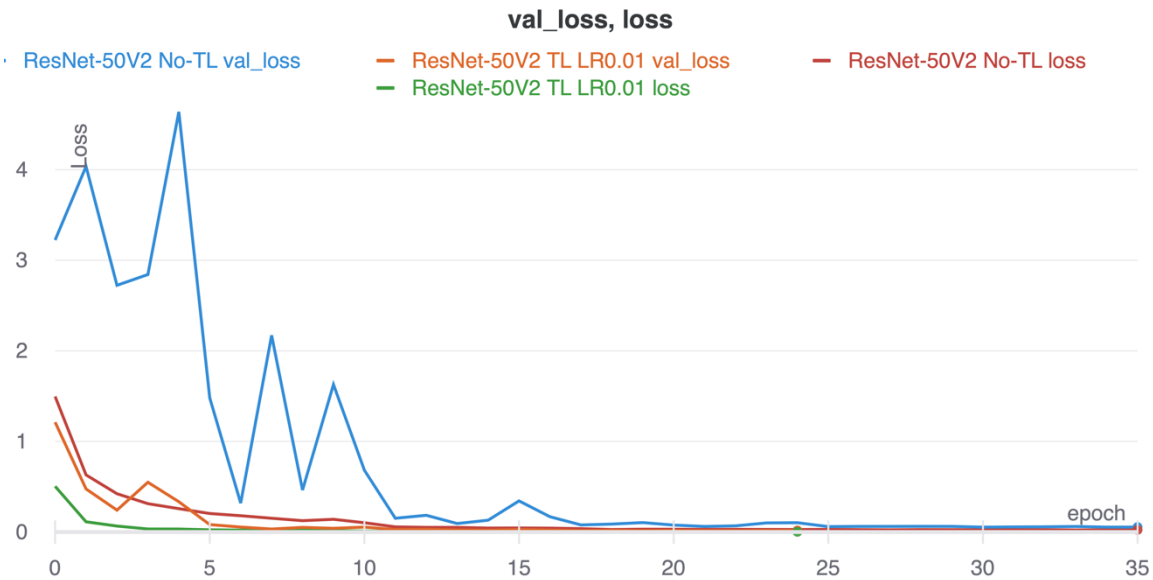


Figure 12: Training and Validation loss on No-TL(No transfer learning) vs TL(Transfer learning)

We can see that the network that uses transfer learning, reaches the peak performance much sooner than the one without transfer learning. Transfer learning can speed up your network by a huge margin.

I would also like to note that when trying the experiment of no Transfer learning on cars dataset it would not train above a certain extent (validation loss and accuracy were stuck), this suggests the network found a local minimum, but the issue was non-existent in case of transfer learning, Finding the right parameters for training the network can be hard and hence transfer learning could be a useful tool that would help.

Conclusion

To conclude, we learnt how to optimise the various parameters and understand its importance.

We chose the 80-10-10 split to prevent very small testing dataset and since the cars data set doesn't have enough images to begin with.

Learning rate was chosen to 0.01 since it worked just fine for all our experiments and we used stepwise reduction to improve the training of the network. The learning rate decay is seen to have a huge impact especially when network seems to stop learning. On google colab we used keras callbacks to take care of the reduction in LR when there was no change in validation loss for more than 3 epochs. We used early stop as well to prevent overfitting of data with a patience of 5-7 epochs depending on the network.

We learnt that having a good dataset is as important as all the parameters. The cars dataset had very few images per class while Plant village had enough and as seen all the networks performed better with that dataset.

We also learnt the importance of rescaling of data to 0 and 1. Since the network not even start to train without it. The other data augmentations did not have as much impact, but it is good to use it in the cars dataset to help generalise the network better.

We saw how different optimisers had impact on the training of the networks. Few of the optimisers don't work at all on the datasets. We however got the best result from SGD so continued to use that for all the networks we tested.

The batch size did have an impact but since we were using the V100 for most of the training we were able to push the batch sizes to higher limits allowing us not only to train faster but also with higher accuracy.

Bibliography

Chollet, F. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. *arXiv*, 1610.02357.