CSCE 636 - Deep Learning - Project Report

Gundam Satyabhama Reddy
Texas A&M University

Abstract

In this project, a deep learning model to perform 10 class image classification task is implemented. The model is trained on CIFAR-10 dataset. I have tried to explore both DenseNet and ResNet architectures with slight modifications and also built an ensembled model to make more accurate predictions.

1 DenseNet

DenseNet model is a network which utilizes dense connections between the layers. That is, each layer in a dense block uses the outputs of all its previous layers within that block. This approach of preserving the feature maps is quite similar to ResNet; while ResNet sums the residual from the previous layer, DenseNet concatenates all the feature maps. Due to its heavy feature reuse, DenseNet will have very few parameters compared to ResNet.

1.1 Architecture

The initial layer in DenseNet-121 network performs a 3x3, 64 2D Convolution on the input to generate 64 channels. According to the architecture in the paper, there should be a max pool layer after this, but I skipped it as the number of dimensions is already small. The initial layer is followed by 4 **Dense Blocks** each of size 6, 12, 24, 16 respectively. The size indicates the number of **Bottleneck Blocks** used in each dense block. Between each of these dense blocks there is a **transition layer** which is responsible for downsampling. The output of the last dense block is then fed to batch normalization, ReLU, global average pooling and finally to a fully connected classifier.

1.2 Basic Blocks

Bottleneck Block : The bottleneck block comprises of Batch Normalization, ReLU and a 1x1 2D convolution with a stride of 1 and no padding, which is then followed by another Batch Normalization, ReLU and a 3x3 2D convolution with a stride of 1 and a padding of 1. This structure helps in reducing the number of parameters required in the network.

Dense Block : A dense block consists of a series of Bottleneck blocks. The output of each bottleneck block is used to concatenate to the outputs of all its succeeding layers. Therefore, if there are L bottleneck blocks, there would be L(L+1)/2 connections in the Dense Block(only those between bottleneck blocks). The dense blocks have a parameter called growth rate which determines the number of output feature maps of each block.

Transition Block : A transition block consists of Batch normalization, ReLU, a 1x1 2D convolution followed by 2D average pooling layer with a stride of 2. This helps in downsampling the results generated by the dense blocks. The transition blocks have a parameter called reduction which determines the downsampling/compression factor.

1.3 Regularization and Standardization

Dropout : To introduce regularization into the network, I introduced a dropout after each dense block with a drop rate of 0.2 and also at the end of the fully connected layer. This prevents the model from overfitting on the training data.

Weight Standardization: I also incorporated weight standardization described in [3] into the network. This has been added to every Convolutional layer used in the network. While the usual normalization techniques use the data space, this particular normalization is performed on the weights of the network. This technique helps in speeding up the training process(which can also be observed in the loss graph below 6).

Introducing these two techniques into the network increased the accuracy from $\sim$89% to $\sim$92%.

1.4 Training

The model is trained for 200 epochs with a batch size of 256. The loss function is defined to be cross entropy loss and optimizer as SGD with learning rate as 0.01, momentum as 0.9 and weight decay as 0.0005. A learning rate scheduler is used to adjust the learning rate based on the number of epochs. The growth rate for the dense layers is set to be 32 and the reduction is set to 0.5 .

2 ResNet

I used the same architecture as ResNet-110 (version 1 with standard blocks) as described in HW-2. The reason for choosing this was it had a test accuracy of 92.54% and I wanted to explore the combined effect of DenseNet and ResNet networks on predictions.

2.1 Architecture

The initial layer in ResNet-110 network performs a 3x3, 16 2D Convolution on the input to generate 16 channels which is then followed by Batch Normalization and ReLU. This is followed by 3 stacks of 18 **Standard Blocks** each. The first block of each stack has a projection shortcut as the feature map dimensions change between stacks. This is handled by performing a 1x1 convolution with stride of 2 at the beginning of stack 2 and stack 3. Finally, in the output layer, we perform average pooling, flatten the result and pass it through a fully connected classifier. The output is thus of the dimensions batch_size x 10, where 10 represents the 10 classes of the CIFAR-10 data.

2.2 Basic Blocks

Standard Block : The standard block performs a 3x3, 2D Convolution on the input, followed by Batch Normalization and ReLU, and another 3x3, 2D Convolution, followed by Batch Normalization. Here the number of output channels for the 3 stacks are 16, 32 and 64 respectively. The input is then added to this output (Note: if there is a projection shortcut, then the shortcut is added.). The generated output is then passed through ReLU layer.

2.3 Training

The model is trained for 200 epochs with a batch size of 128. The loss function is defined to be cross entropy loss and optimizer as SGD with learning rate as 0.1, momentum as 0.9 and weight decay as 0.0002. The learning rate is divided by 10 after a few epochs.

3 Ensembling

Ensembling is a technique of using multiple models to make predictions. In this case, I am using both DenseNet model and ResNet model to make predictions on the image classification problem. The softmax output of each of these models is averaged(multiplied by 0.5 and $1 - 0.5$ and added) to get the final probabilities of the classes. The factor 0.5 can be chosen based on how confident we are with each model. Based on trial and error, I found that 0.5 is the best fit in this case. Using this approach increased the prediction accuracy by $\sim$2%. This shows that ensembling helps in correcting wrong predictions that would have occurred if the two models were used individually.

4 Image Augmentations

4.1 Random Crop

We try to pad the input image with a padding of 4, then select a random 32x32 section of it to train the model. This introduces generalization into the model and it also avoids giving lesser weight to edges of the image during convolution. This approach also helps in randomly changing the position of the object in the image and thereby making the model more robust.

4.2 Flip

The randomly cropped image is then subject to a random horizontal flip. Looking at the CIFAR-10 data and the test images provided, I decided not to do a vertical flip as most of the images were straight(not inverted).

4.3 Normalization

The generated image is then subject to normalization using it's mean and standard deviation. A small epsilon is introduced in the denominator along with standard deviation to avoid divisions by zero.

4.4 Other Explored Augmentations

In DenseNet, I tried to add blurring to the image for the first 25% of the epochs so that the model does not overfit to the images early on in the training. But, this did not give any significant improvements to the accuracies, probably because the images are already quite small.

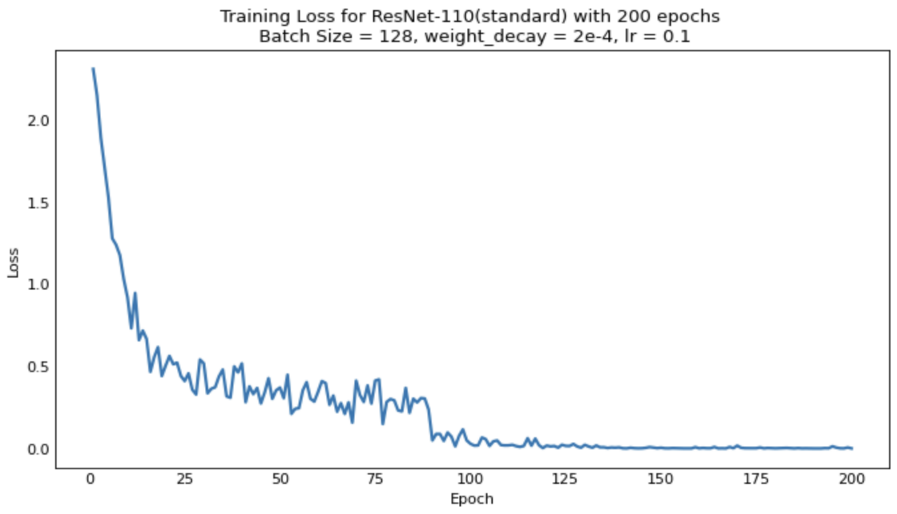
Test Time Augmentation : (test_or_validate_tta function) In ResNet, during testing, instead of passing just one image to the network, I run a few augmentations on the image and try to make predictions on all the generated images. The results generated for this set of images is then averaged out to reach the final prediction. The augmentations included blurring the image, increasing the brightness, and also the above mentioned augmentations. However, this approach increased the test time significantly and the accuracy improved only in decimals. To avoid this additional effort, I have discarded this approach.

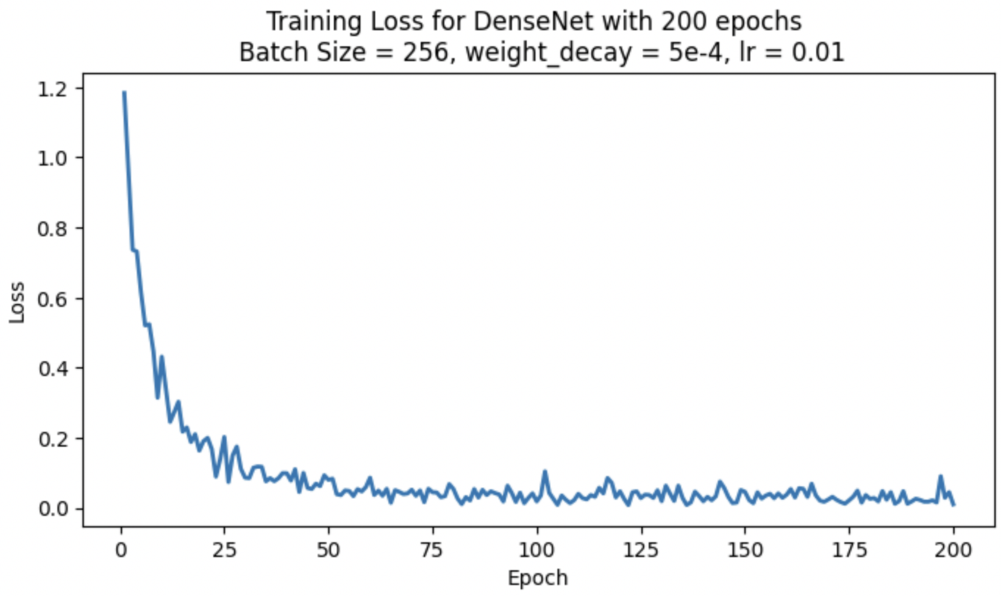
5 Project Structure

1. **main.py** : Main file used to run the project.
2. **Configure.py** : Configures the training and model parameters
3. **DataLoader.py** : Load training and testing images.
4. **ImageUtils.py** : Helper class to preprocess image and perform augmentations.
5. **Model.py**: Has the traininig ans testing functions for the DenseNet model.
6. **ResnetModel.py** : Has the traininig ans testing functions for the ResNet model.
7. **Network.py** : Has the network definition for DenseNet.
8. **ResnetNetwork.py** : Has the network definition for ResNet.
9. **Ensembled.py** : Helper class to make predictions on both DenseNet and ResNet combined.
10. **saved_models** : The saved models are present in this directory.

6 Loss Graphs

The loss graphs for training both the models is as shown in Figure 6. As can be seen, the DenseNet model converges a lot faster than ResNet model.





7 Results

Following are the test accuracies on CIFAR-10 test set for the models trained with the above mentioned configurations and augmentations(See Figure 1). As can be seen from the accuracies, while DenseNet has an accuracy of 92.06% and ResNet has 92.32%, the ensemblement of the two models generates an accuracy of 94.01%. There is nearly 2% increase in the accuracy after this arrangement.

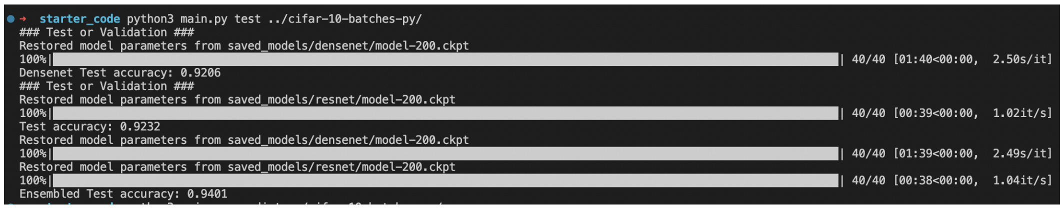

```
● ➜  starter_code python3 main.py test ../cifar-10-batches-py/
  ### Test or Validation ###
  Restored model parameters from saved_models/densenet/model-200.ckpt
  100%|████████████████████████████████████████████████████| 40/40 [01:40<00:00,  2.50s/it]
  Densenet Test accuracy: 0.9206
  ### Test or Validation ###
  Restored model parameters from saved_models/resnet/model-200.ckpt
  100%|████████████████████████████████████████████████████| 40/40 [00:39<00:00,  1.02it/s]
  Test accuracy: 0.9232
  Restored model parameters from saved_models/densenet/model-200.ckpt
  100%|████████████████████████████████████████████████████| 40/40 [01:39<00:00,  2.49s/it]
  Restored model parameters from saved_models/resnet/model-200.ckpt
  100%|████████████████████████████████████████████████████| 40/40 [00:38<00:00,  1.04it/s]
  Ensembled Test accuracy: 0.9401
```

Figure 1: Testing accuracies for DenseNet, ResNet and Ensembled Models.

A sample of the predictions generated by the ensembled approach is also shown in Figure 2

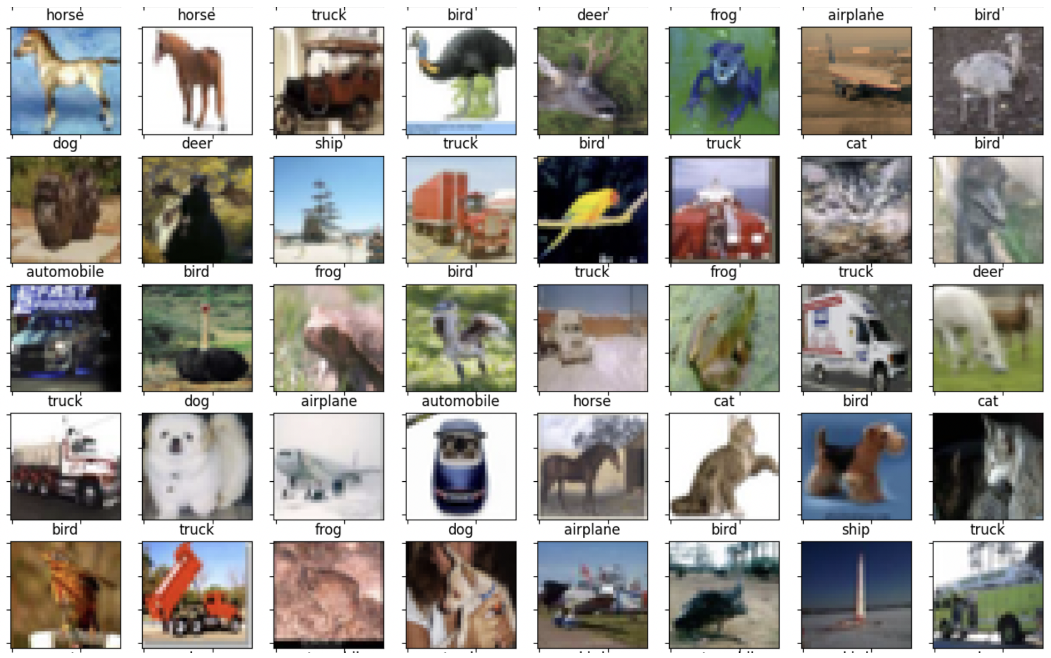


Figure 2: Predictions generated by Ensembled ResNet and DenseNet models.

The results on the private test set is generated and included in the submission as predictions.npy.

8 Conclusion

Various image augmentation techniques, both during training and testing were explored. Two models for image classification, DenseNet and ResNet, were explored and implemented. In DenseNet, techniques like Dropout and weight standardization were incorporated into the model. The best of both models was achieved through ensembling.

References

[1] Huang, Gao, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. "Densely connected convolutional networks." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4700-4708. 2017.

[2] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.

[3] Qiao, Siyuan, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. "Micro-batch training with batch-channel normalization and weight standardization." arXiv preprint arXiv:1903.10520 (2019).