CSE253 Programming Assignment3

Chao Yu
UCSD ECE
PID: A99049546
chy018@eng.ucsd.edu

Yau Mo Chan
UCSD CSE
AID: A53067459
ymchan@eng.ucsd.edu

Brian Whiteaker
UCSD CSME
PID: A04302767
bwhiteak@ucsd.edu

Inyoung Huh
UCSD CSME
PID: A53213515
i1huh@ucsd.edu

Abstract

We explore the capabilities of the PyTorch deep learning framework. We work through the construction of convnets by implementing convolution, pooling, activation, normalization, and drop-out layers. The modular construction of complex network architectures is shown. We make use of various optimization modules, also we use batch normalization in training. Further, we implement transfer learning using the VGG16 convnet model. The pre-trained layers are imported and a final custom layer is added. The custom layer is used in classification of the Caltech256 image data. Lastly, we use the VGG16 model for feature selection and visualization of the relevant features on an image.

1 Introduction

1.1 Deep Convolutional Network for Image Classification

In this problem, we train a deep network to classify images in the CIFAR-10 Dataset. We downloaded the CIFAR-10 data set into PyTorch dataloader. The PyTorch dataloader module allows us to make transformations to the image files such as normalization of pixel values, random flipping of images, and various other capabilities. Of particular use is the dataloader's ability to form shuffled batches of a fixed size. These are then served to the network during training on request as mini-batches.

The images are of the dimensions $32\times32\times3$, 32×32 for the image size, and 3 for the RGB channels per each image. Connected to this we make design choices of convolution layers, where we must decide on a kernel size. The kernel is slid over image taking a "snapshot" at each step, the slide is managed with a choice of stride size, allowing each snapshot to overlap the previous. This process is the convolution of the pixels set in one snapshot with the next. A strength of the convnet in image classification is its use of local structure in the image to support feature learning. This ability is rooted in the overlapping of these "snapshots" in the convolution layers.

Sandwiched between convolution layers are pooling, dropout, or normalization layers depending on design choice. Pooling layers are popularly chosen with 2×2 receptive fields. From a receptive field a representative maximum value pixel is chosen as active and is passed forward through the network, non-active pixels are effectively zeroed. Gradients backpropogate through the active pixel. This process reduces the size of the input, not necessarily the depth though. Dropout layers are used to reduce the number of parameters and help with overfitting by reducing parameters and therefore allowable complexity of the network. Convolution layers can increase the input depth to the next layer from say, 3 initially for RGB, to 64. The added depth allows for more complex features to be learned. Associated with this is an explosion of parameters. The dropout layer will choose to randomly drop some percentage of the parameters, in effect, creating a different architecture. These changing architectures give more flexibility to what may be learned by the network.

Another layer we use are the batch normalization layers. As in the standard neural networks these layers enforce that the mean and variance of the inputs to the activation function remain in some optimal range. For each batch passed through the network, it will have its own particular mean and variance, that the normalization layers adjust for.

1.2 Transfer Learning

1.2.1 Transfer Learning with VGG16

The VGG16 network won the ILSVRC competition in 2014. After winning this competition the trained network was made available to the public for transfer learning. To train a very deep network on the ImageNet data would require hours of computing time on machines not available to most users. We use the fully trained network made available through PyTorch.

The ImageNet data was comprised of 1000 image classes. Our Caltech data has 256. The idea is to take the trained network and use the features it has found to quickly train a classifier on different image data. The thought is that the structure in images from the world will not vary fundamentally for some other set of data. PyTorch allows the user to remove layers as modules and "plug" in their own layers where needed. We removed the final softmax layer, which was designed to have 1000 outputs, and plugged in our own softmax for 256 outputs. We then train this new final layer on our data and let it adjust itself to classify our images.

1.2.2 Feature Extraction

The Caltech256 data is a smaller data set than ImageNet. So it is the case that not all of the learned features of the full VGG16 are necessary for the classification problem on Caltech256. For feature extraction we dropped a large portion of the VGG16 model, and our experiments attached a fully connected layer with softmax and cross-ent loss to the third layer, and then to the fourth layer. We find that the network easily overtrains on these available features at this layer-level. We discuss this in the results section.

2 Methods

2.1 Deep Convolutional Network for Image Classification

In order to find the best and creative deep networks, we applied various methods and experimented with different number of layers, kernel and filter sizes. The things we used in our networks are as follows.

1. Batch Normalization to provide any layer in a Neural Network with inputs that are zero mean/unit variance.
2. Xavier Initialization to make sure the weights keeping the signal in a reasonable range of values through many layers.
3. Adam Optimizer to compute individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.
4. Dropout to randomly zero some elements of the input tensor for regularization and preventing the co-adaptation of neurons.
5. Stochastic gradient descent performs batch gradient descent with a learning rate or learning rate scheduler attatched, and built in Nesterov momentum.

Padding and stride is 1 and kernel size is 3x3.

	$BatchNorm$	$XavierInit$	$AdamOptimizer$	$Dropout$	$BatchSize$
$ModelA$	O	X	O	X	32
$ModelB$	O	O	O	O	125
$ModelC$	O	O	O	O	125

Table 1: Summary of Methods

2.2 Transfer Learning

2.2.1 Transfer Learning with VGG16

We carried out transfer learning on VGG16. The original VGG16 is used to classify 1000 classes but Caltech256 has only 256 classes, so, we adjusted the output layer to 4096$\times$256 for the last fully connected layer.

2.2.2 Feature Extraction

For feature extraction we experiment using the third and fourth intermediate Convolutional Blocks as input. For each class in dataset, we choose batch sizes of 32 for training data and 8 for test data. We used SGD with learning rate 0.001 as an optimization function and trained 10 epochs.

3 Results and Discussion

3.1 Deep Convolutional Network for Image Classification

3.1.1 Model A

For model A in Figure 1 2, experiments were carried out starting with use of the stochastic gradient descent(SGD) optimizer of the optimizer package. Some basic parameters to SGD were learning rate and momentum options. Initially using a basic network as given by the PyTorch tutorial we experimented.

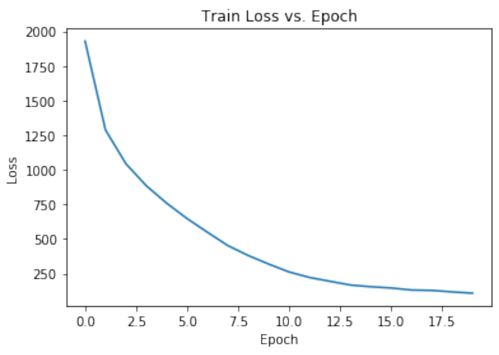
Holding momentum constant and varying the learning rate from 0.1-0.00001, the accuracy was degraded near the endpoints of this interval. Learning rate 0.001 provided the best performance. The same process applied to the momentum with a range of 1.2-.85 saw a momentum setting of 0.92 as best. The accuracy with these settings was maximum of about 63%.

Attempting a different optimizer, Adaptive Momentum (Adam) was chosen. With this choice, the performance gains were not spectacular and moved accuracy upward only a percent or two. Various other layers were added and taken away in different permutations. Dropout layers with a dropout of 0.4 were tried. This was not useful since this basic network was never close to over-fitting. Batch normalization, changes to kernel size in convolution layers and maxpooling were all implemented.

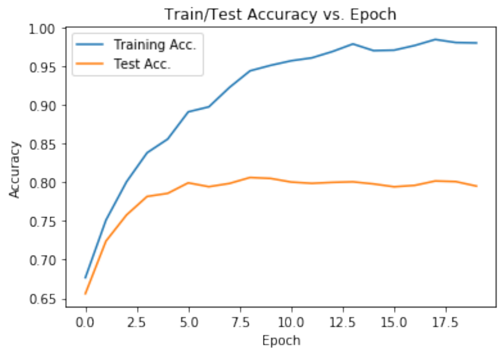
The substantial gains appeared when a second layer was added. this pushed accuracy percentages up into the low 70's. As a result, more layers were added to give the final model. The final model A mimics the VGG16 architecture. The first two convolution layers had depth 64 with kernel size 3X3 with a stride of one. The stride small stride does not reduce the volume very much at this point. These convolutions are separated by a batch normalization. The second convolution feeds into a maxpool layer with receptive area of 2X2.

The next layer has the same setup of convolution, batch normalization, convolution, maxpool. The only difference here is the convolutions have depth 128. similar to how VGG16 increases by a factor of 2. For all convolutions, affine was set to true.

The resulting network gives a training accuracy of 98.008% on training data and 79.51% on unseen test data after 20 epochs.



(a) $Costerror$



(b) $Accuracy$

Figure 1: Model A

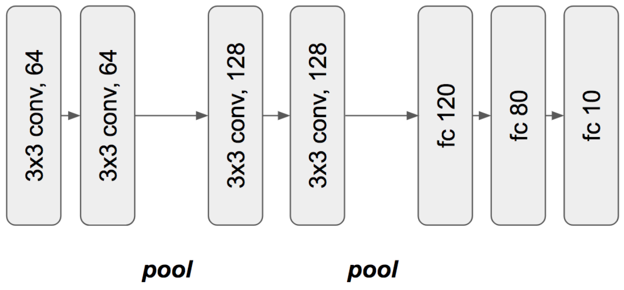


Figure 2: Model A - Diagram

3.1.2 Model B

Since we should use VGG16 model for next problem, we designed our convolution neural network similar with VGG16 in Figure 3, 4. This consist of lots of layer and designed as deep neural network. We used 1 padding, 1 stride and 3X3 kernels to design this network. Here, we can find why padding is the key to design deep neural network. Without padding, it is hard to design deep neural network due to pooling. Padding can allow us to design deeper neural network. Especially, in this problem, we should use padding because the images we used consist of small number of pixels. Also for this experiment, we applied Adams optimizer, dropout, Batch normalization and 125 mini batch. We achieved 71.64% accuracy.

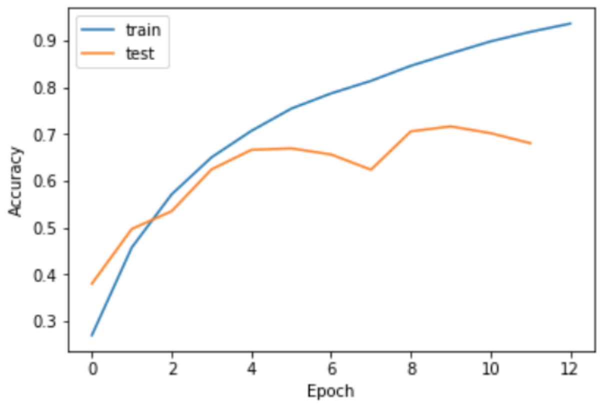


Figure 3: Model B - Accuracy

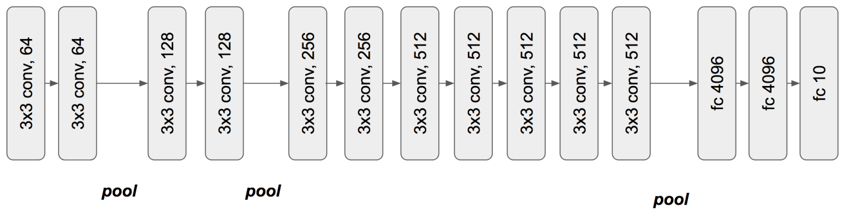


Figure 4: Model B - Diagram

3.1.3 Model C

Model C is basically similar with Model B shown in Figure 5, 6. However, we added more layers on Model B. Also model C used all the same methods with Model B (Adams optimizer, dropout, Batch normalization and 125 mini batch. This result obviously shows that we can reach out the better accuracy 77.28% through deeper layers. It requires more time to learn. If we add more layers on this Model C, we can get the better accuracy than this one. However, we always think about the balance between accuracy and time. We can get better by adding the layers but it might be slight improvement compared the time we invest.

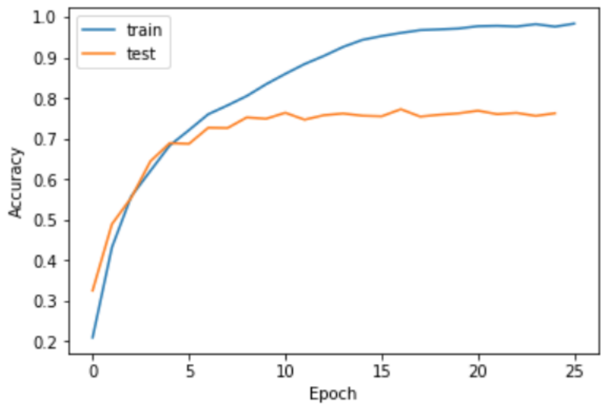


Figure 5: Original Image

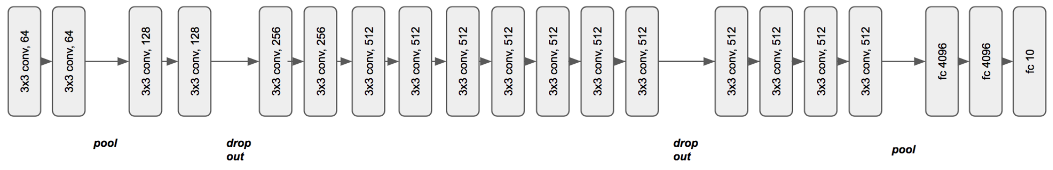
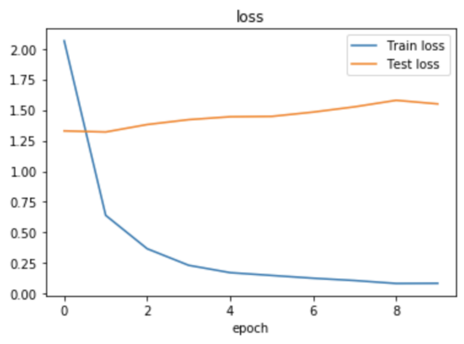


Figure 6: Original Image

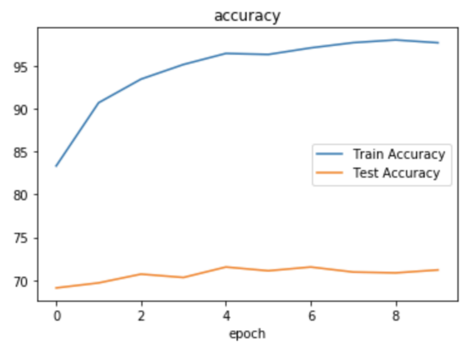
3.2 Transfer Learning

3.2.1 Transfer Learning with VGG16

From Figure 7 , we have 97.7294% for training data and 71.1914% for test data. We trained 10 epochs with SGD with learning rate 0.001. The training loss dropped rapidly and reached 0.0859 while the test loss oscillated in the training process. We observe that VGG16 has many parameters while our training set was not very large. The excessive available parameters create a flexible network, causing overfit of the model.



(a) $Costerror$



(b) $Accuracy$

Figure 7: VGG16

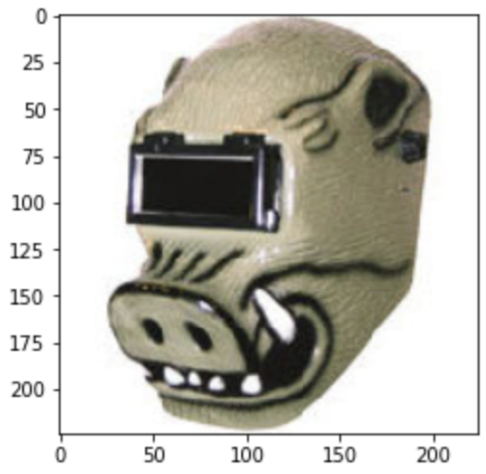
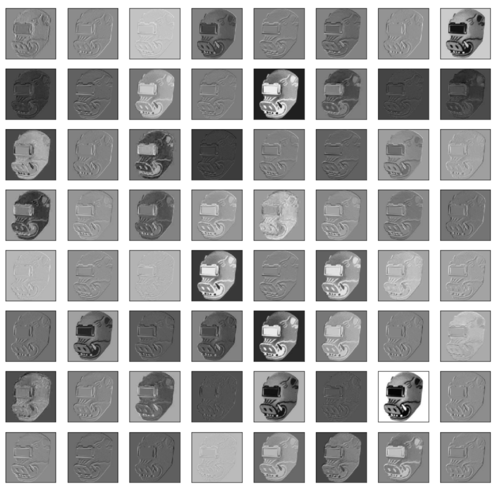


Figure 8: Original Image


(a) $VGG16 - Firstfilter$

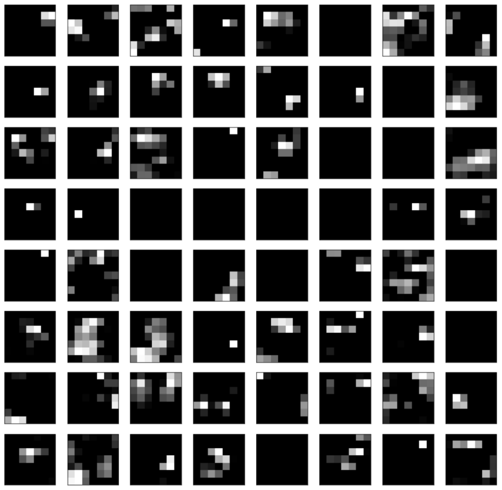

(b) $VGG16 - Lastfilter$

Figure 9: Visualize filters from layers first and last Convolution Layers

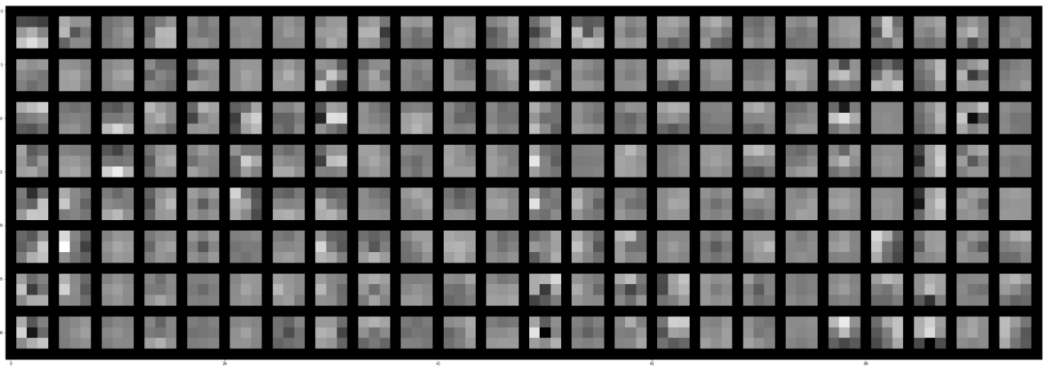


Figure 10: Weight - 3 channels

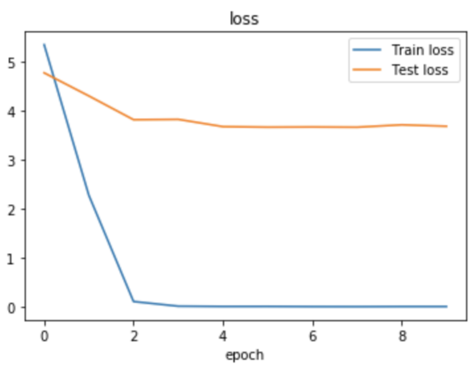
Each filter is generated by same input. However, their initial values are not identical. Hence, the values after filtering are different. If the initial values were the same, their weights would also be the same. This is the reason why we have different initial values for weights.

The first filters learn the edges in Figure 9(a), and other simple structure, eventually distinguishing color and blob. These figures show the result of random initialization of a filter after learning. It follows that with more layers, more complex information is learned in Figure 9(b). This complex information is changed from simple patterns, to the meaning of images. The more layers we have, the more abstract information we can achieve. The Figure 10 showed the weights of three channels.

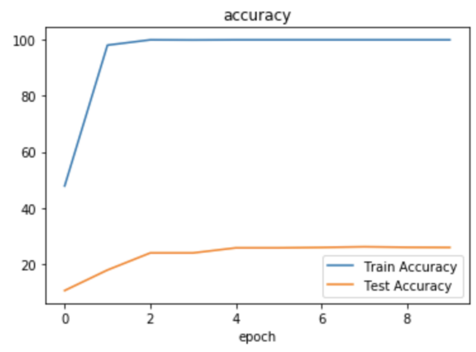
3.2.2 Feature Extraction

When using three blocks, the size of output is 28×28×256, which is 200704. Then we add a 200704×1024,ReLU and 1024×256 fully connected layer after that. The figures for accuracy and loss are shown in Figure 11. From Figure 11 , we can see that the three blocks model is very easy to overfitted. The training accuracy reaches 99.9877% accuracy while the test accuracy is only 27.4414% . The loss on training data is 0.00469 and 3.660 for test data.

When using four blocks, we add two fully connected layers with size 100352 ×1024 and 1024×256 after the fourth block. The accuracy reaches 99.9877% for training set and 5 for test set in Figure 12.



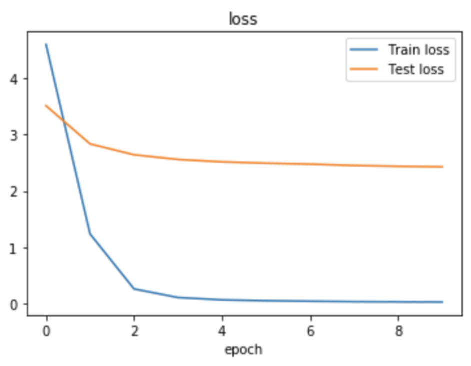
(a) $Costerror$



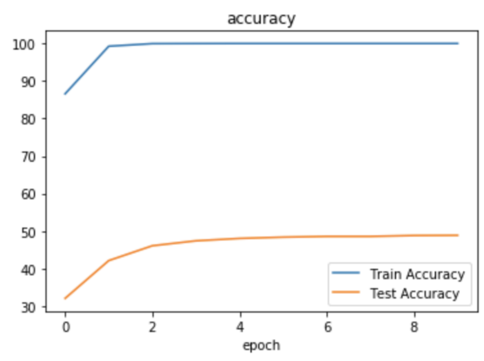
(b) $Accuracy$

Figure 11: Threeblock

The accuracy of training set and test show a huge difference. The loss was 0.00425 for training set and 2.0857 for test set. The training set loss also dropped quickly and very close to 0 while the loss of test set only dropped a little. We conclude that the model is still very overfitted.



(a) $Costerror$



(b) $Accuracy$

Figure 12: fourblock

4 Summary

This is our first experience to use neural network library 'PyTorch' and pre-trained model through PyTorch. In the first part, we designed our own convolutional neural networks and modified our model by applying various method such as Batch Normalization, Drop out, Adams optimizer, various mini-batch sizes, different size of layers and etc. From first part, we achieved around 80% accuracy. Through this problem, we can see how different methods can affect our model. In the second part, we used pre-trained model - VGG16. To understand the structure of VGG16, we looked into VGG16 by visualizing the filters and weights. With VGG16, we trained Caltech256 data and reached out 97.72% accuracy by modifying the last layers to softmax. After that, using three blocks and four blocks, we can reach out the better accuracy - 99.98%

5 Contribution

Inyoung and Brian focused on Deep Convolutional Network for Image Classification and experiment with trying different layers. Chao and Brian did the transfer learning with VGG16. Inyoung did weights and filter visualization. Chao did feature extraction. Yau Mo did separate experiment on classification.

References

[1] Christoper Bishop (1995) Neural Networks for Pattern Recognition

[2] https://github.com/pytorch/examples/blob/master/mnist/main.py

[3] https://github.com/pedrodiamel/nettutorial/blob/master/pytorch/pytorch_visualization.ipynb

[4] http://cs231n.github.io/convolutional-networks/#case

[5] http://pytorch.org/tutorials/beginner/pytorch_with_examples.html#nn-module

[6] http://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html