CSE 253 PA3: Transfer Learning with VGG16

Brandon Ustaris
Department of Computer Science
University of California, San Diego
La Jolla, CA 92093
bustaris@ucsd.edu

Brian Preskitt
Department of Mathematics
University of California, San Diego
La Jolla, CA 92093
bpreskit@ucsd.edu

Brian Wilcox
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, CA 92093
bpwilcox@ucsd.edu

Derek Tran
Department of Computer Science
University of California, San Diego
La Jolla, CA 92093
dtt018@ucsd.edu

John Clara
Department of Computer Science
University of California, San Diego
La Jolla, CA 92093
jclara@ucsd.edu

Abstract

This assignment was focused on doing Transfer Learning with VGG16. We utilized Keras which is a Deep Learning Library which includes the VGG16 ConvNet. VGG16 was trained on ImageNet but for this assignment, we worked with the CalTech256 and UrbanTibes dataset. To perform the transfer learning, we replaced the Softmax Layer of the VGG16 pre-trained model by our own Softmax Layer which will predict classes of our datasets. After replacing, we attempted to train it and observe its performance. Lastly, we explored the use of a temperature-based softmax regression on the last output layer of the VGG16 model as input to a softmax layer for the Caltech256.

1 CalTech256 Classification

For this section, we will use Transfer Learning in order to recognize a set of object categories from images from CalTech256.

1. **Read and prepare data**
   (a) First we read in the Caltech256 images by moving the first 16 images of each class to a training directory and the next 4 images of each class to a validation directory. We used the Keras ImageDataGenerator to read these images into memory.
   (b) We learned that the VGG16 did not perform global contrast normalization. At first, we performed samplewise contrast normalization but found that without performing the normalization we achieved much better results.
   (c) By not shuffling the ImageDataGenerator, we were able to push the training and validation data through the VGG16 net in order and create one-hot encoded targets for them easily.
   (d) After getting the features from the VGG16 net, we shuffled the training and validation output data together along with their target vectors.

2. **Create CalTech256 ConvNet**
   (a) First we created a Keras ConvNet model with weights of the VGG16 model.
   (b) Then we took off the original Softmax Layer and set all of the other layers to not be trainable.
   (c) Then we verified that the model has been updated and was appropriate by using *model.summary()*.

```
In [415]: model.summary()
```

Layer (type)	Output Shape	Param #	Connected to
input_46 (InputLayer)	(None, 224, 224, 3)	0	
block1_conv1 (Convolution2D)	(None, 224, 224, 64)	1792	input_46[0][0]
block1_conv2 (Convolution2D)	(None, 224, 224, 64)	36928	block1_conv1[0][0]
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0	block1_conv2[0][0]
block2_conv1 (Convolution2D)	(None, 112, 112, 128)	73856	block1_pool[0][0]
block2_conv2 (Convolution2D)	(None, 112, 112, 128)	147584	block2_conv1[0][0]
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0	block2_conv2[0][0]
block3_conv1 (Convolution2D)	(None, 56, 56, 256)	295168	block2_pool[0][0]
block3_conv2 (Convolution2D)	(None, 56, 56, 256)	590080	block3_conv1[0][0]
block3_conv3 (Convolution2D)	(None, 56, 56, 256)	590080	block3_conv2[0][0]
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0	block3_conv3[0][0]
block4_conv1 (Convolution2D)	(None, 28, 28, 512)	1180160	block3_pool[0][0]
block4_conv2 (Convolution2D)	(None, 28, 28, 512)	2359808	block4_conv1[0][0]
block4_conv3 (Convolution2D)	(None, 28, 28, 512)	2359808	block4_conv2[0][0]
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0	block4_conv3[0][0]
block5_conv1 (Convolution2D)	(None, 14, 14, 512)	2359808	block4_pool[0][0]
block5_conv2 (Convolution2D)	(None, 14, 14, 512)	2359808	block5_conv1[0][0]
block5_conv3 (Convolution2D)	(None, 14, 14, 512)	2359808	block5_conv2[0][0]
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0	block5_conv3[0][0]
flatten (Flatten)	(None, 25088)	0	block5_pool[0][0]
fc1 (Dense)	(None, 4096)	102764544	flatten[0][0]
fc2 (Dense)	(None, 4096)	16781312	fc1[0][0]

```
Total params: 134,260,544
Trainable params: 0
Non-trainable params: 134,260,544
```

3. **Training**
   Next, we had to train the ConvNet with a small number of training images per class. First we pushed the training and validation data through the frozen VGG16. Then we used the outputs to train a Softmax Layer. We repeated this with different numbers of samples per class.
4. **Inference**
   (a) After training, we were to plot the training and test loss vs. iterations which can be found in Figure 1. We used 16 samples per class during this training. Our training loss decreased to almost 0 pretty steadily but our test loss was only able to decrease to about 3.
   (b) Then we plotted classification accuracy vs. iterations which can be found in Figure 2. Again, we used 16 samples per class for training. Our training accuracy neared 1 but our test accuracy leveled out around .65.
   (c) Next, we needed plotted classification accuracy vs. number of samples used per class which can be found in Figure 3. The more samples we used the better our accuracy. The accuracy followed the curve from Figure 9 in Zeiler and Fergus.
   (d) From the accuracy and loss graphs we notice that the Softmax Layer seems to be learning normally in that the more iterations, the lower the loss and the higher the accuracy. However, even thought the training accuracy would get close to 100 percent, the test accuracy would cap out around 65 percent. We think that this is because we were using too few training samples.

   We notice that the more samples we use the better accuracy we can achieve. However, the more samples we use per class the more computationally expensive it becomes. Moreover, we achieve diminishing returns with each new sample added to the class. By that we mean that the accuracy increased logarithmically with the number of samples added. So if we were to up to 32 more samples we would only achieve a couple percentage points more accuracy.

   It was surprising to us that we could achieve much better accuracy without doing any samplewise centering or samplewise normalization. We suspect that the VGG16 net that Keras provides might be different that the original one.

   One way that we could improve our accuracy even further would be to unfreeze

the last couple dense layers of the VGG16 network. In this way, we can train those weight as well and fine tune them to the Caltech256 dataset instead of the Imagenet one.

(e) We then Visualized the filters from the first and last Convolution Layers of the trained model. The first layer has 64 filters in Figure 4 while the last convolution layer has 512 filters in Figure 5.

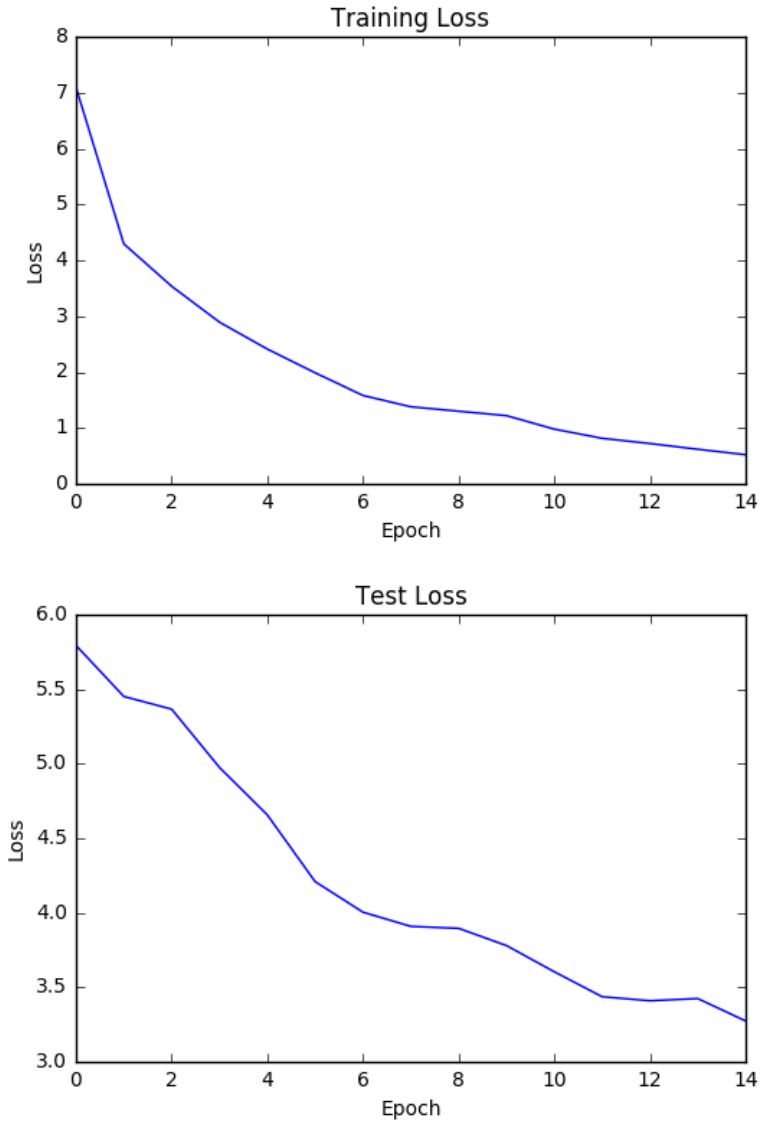


Figure 1: This plot shows loss vs. iterations.

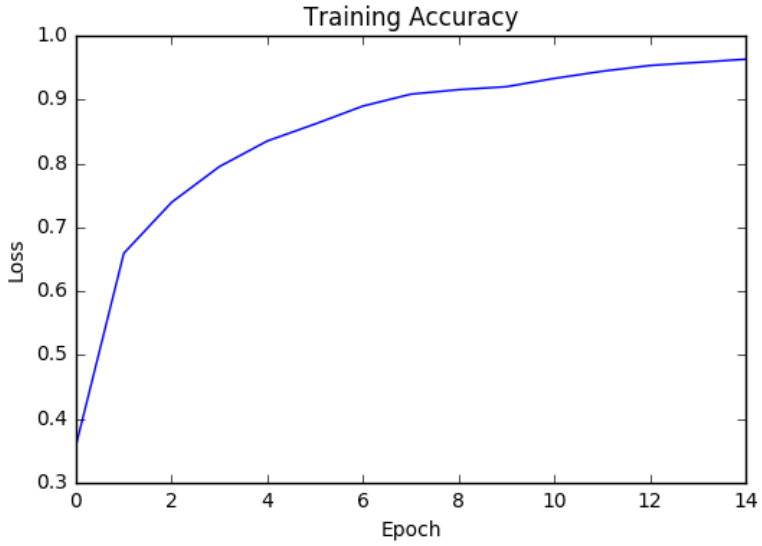


Figure 2: This plot shows training accuracy vs. iterations.

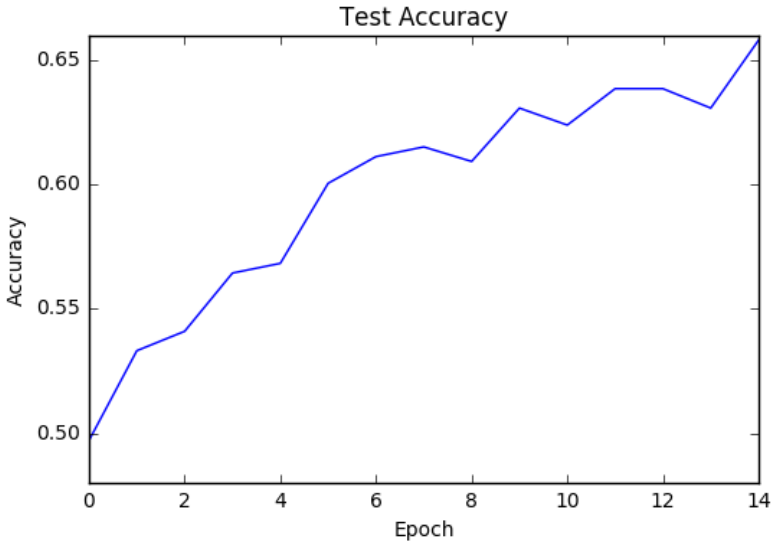


Figure 3: This plot shows test accuracy vs. iterations.

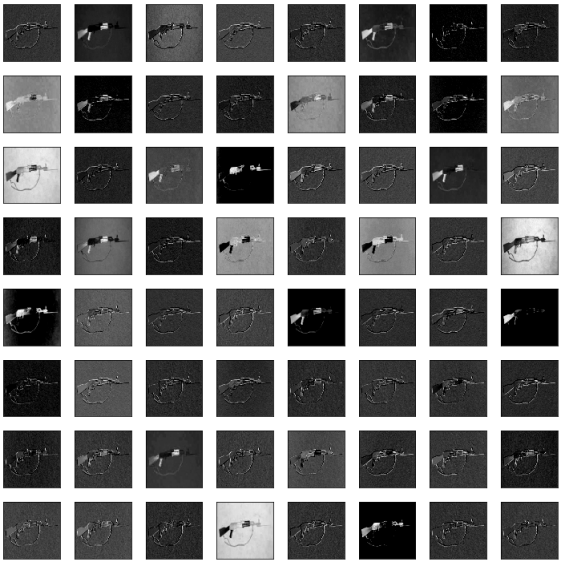


Figure 4: This is a visualization of the 64 filters which are 224 x 224 each from the first convolution later

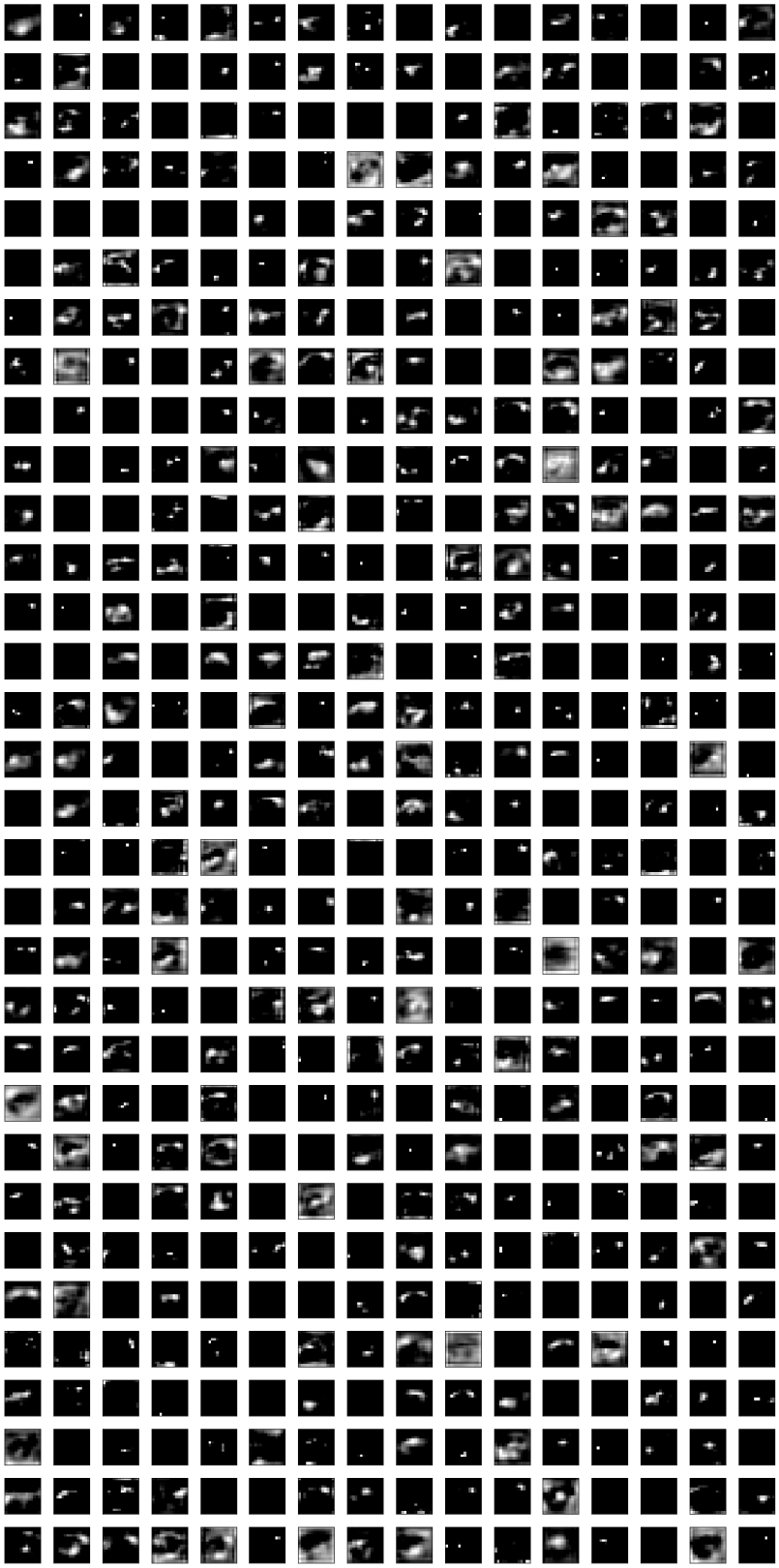


Figure 5: This is a visualization of the 512 filters which are 14 x 14 each from the last convolution later

5. **Feature Extraction**

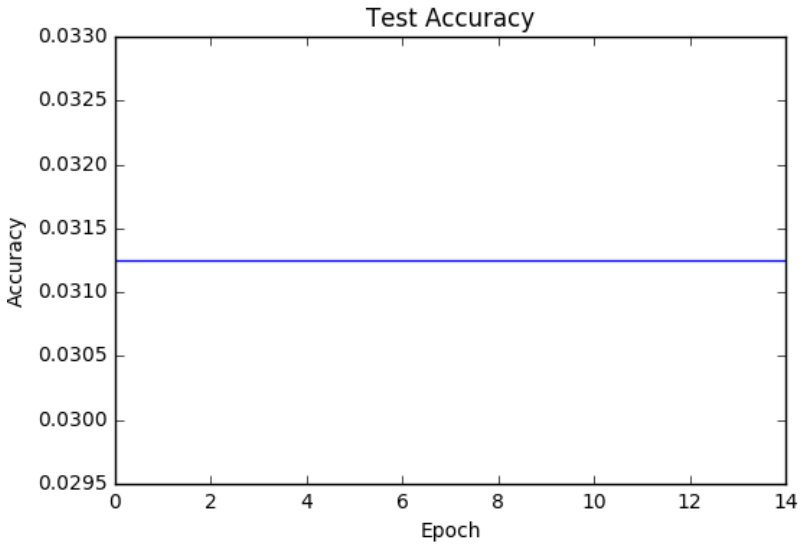
(a) At first we tried cutting the VGG16 net off at the 4th max pooling layer. This increased the dimensionality by a lot because the output dimension of the max pooling layer was 100352 instead of 4096 for the output of the 2nd dense layer. We ended up with the model:

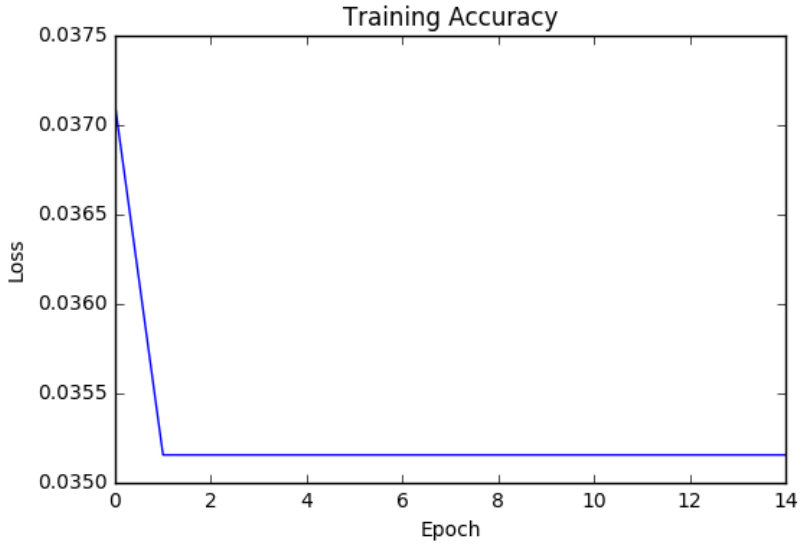
```
In [418]: model.summary()
```

Layer (type)	Output Shape	Param #	Connected to
input_47 (InputLayer)	(None, 224, 224, 3)	0	
block1_conv1 (Convolution2D)	(None, 224, 224, 64)	1792	input_47[0][0]
block1_conv2 (Convolution2D)	(None, 224, 224, 64)	36928	block1_conv1[0][0]
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0	block1_conv2[0][0]
block2_conv1 (Convolution2D)	(None, 112, 112, 128)	73856	block1_pool[0][0]
block2_conv2 (Convolution2D)	(None, 112, 112, 128)	147584	block2_conv1[0][0]
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0	block2_conv2[0][0]
block3_conv1 (Convolution2D)	(None, 56, 56, 256)	295168	block2_pool[0][0]
block3_conv2 (Convolution2D)	(None, 56, 56, 256)	590080	block3_conv1[0][0]
block3_conv3 (Convolution2D)	(None, 56, 56, 256)	590080	block3_conv2[0][0]
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0	block3_conv3[0][0]
block4_conv1 (Convolution2D)	(None, 28, 28, 512)	1180160	block3_pool[0][0]
block4_conv2 (Convolution2D)	(None, 28, 28, 512)	2359808	block4_conv1[0][0]
block4_conv3 (Convolution2D)	(None, 28, 28, 512)	2359808	block4_conv2[0][0]
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0	block4_conv3[0][0]
flatten_16 (Flatten)	(None, 100352)	0	block4_pool[0][0]

```
Total params: 7,635,264
Trainable params: 0
Non-trainable params: 7,635,264
```

(b) When training over just 32 classes with 16 samples per class, we were not able to achieve any learning for the Softmax layer.





We tried again at the 5th pooling layer and achieved the same results. We suspect that most of the usefulness of the VGG16 net is kept in the last two dense layers. We think that the convolution layers do not contain enough useful information to train a softmax layer off of.

2 Urban Tribes

For this section, we will use Transfer Learning in order to recognize the social styles of people. In particular, we will be classifying the Urban Tribes dataset into groups such as bikers, hiphop, hipster, formal, and goth.

1. **Read in the Data**
   (a) First we read in the Urban Tribes images by walking through the directories of the dataset and converting each image to an array representation of pixel values.
   (b) Then we created the labels for each array representation by using a vector of length 11 and using one-hot encoding since there are 11 classes.
   (c) The final step for preparing the data was to shuffle the entire data and prepare a hold-out set by taking a portion of the training set.
2. **Create Urban Tribes ConvNet**
   (a) First we created a Keras ConvNet model with weights of the VGG16 model based off of the provided starter file.
   (b) Then we replaced the existing Softmax Layer with a new one whose output dimension was 11 (number of classes in Urban Tribes)
   (c) We also had to ensure only this last Softmax Layer was going to be trained. So we set trainable to False for all the layers except the last Softmax Layer.
   (d) Then we verified that the model has been updated and was appropriate by using *model.summary()*.
3. **Training**
   Next, we had to train the ConvNet with a small number of training images per class in order to try to replicate the figure shown in the last lecture. We repeated this training a few times while varying the size of the subsets.
4. **Inference**
   (a) After training, we were to plot the training and test loss vs. iterations which can be found in the two figures below. Figure 6 plots training loss vs. iterations while Figure 7 plots test loss vs. iterations.

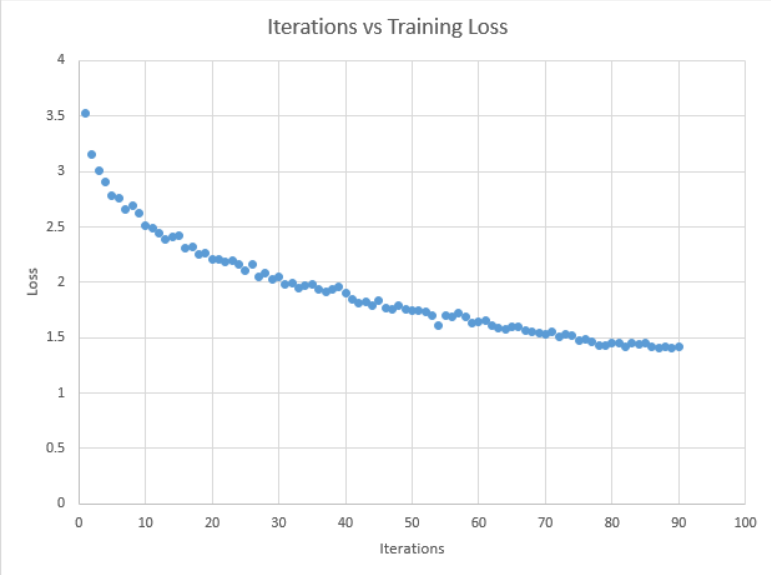


Figure 6: This plot shows training loss vs. iterations.

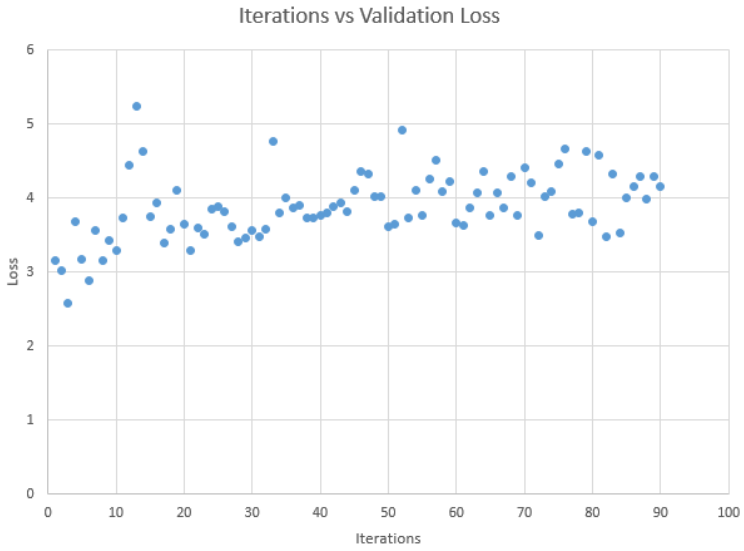


Figure 7: This plot shows validation loss vs. iterations.

(b) Then we plotted classification accuracy vs. iterations which can be found in the two figures below. Figure 8 plots training accuracy vs. iterations while Figure 9 plots test accuracy vs. iterations.

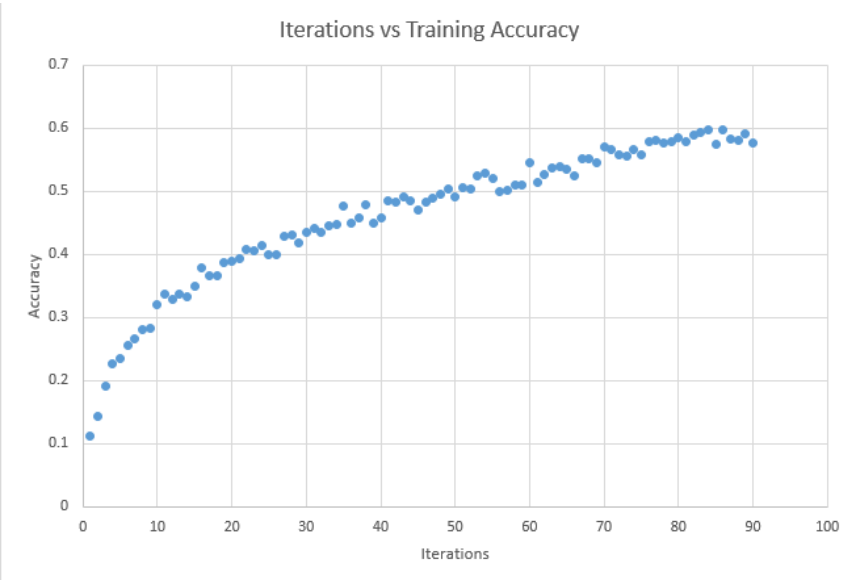


Figure 8: This plot shows training accuracy vs. iterations.

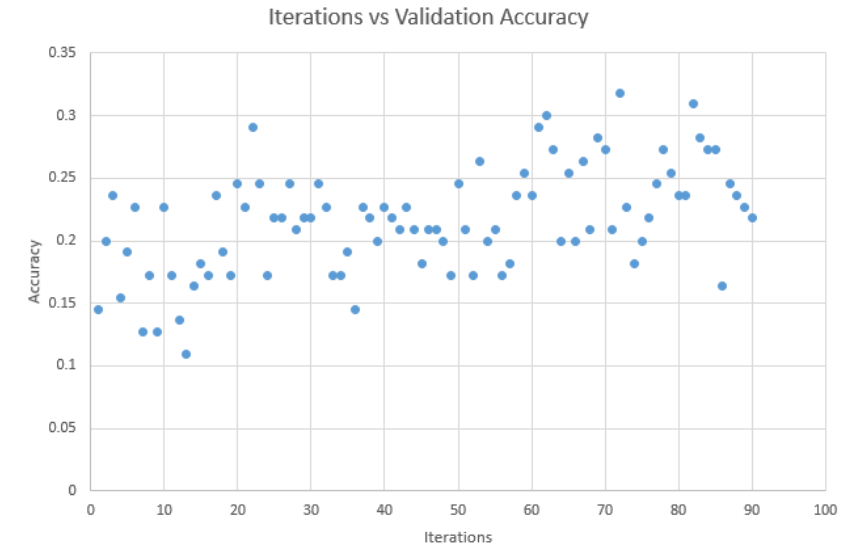


Figure 9: This plot shows validation accuracy vs. iterations.

(c) Next, we needed plotted classification accuracy vs. number of samples used per class which can be found in the two figures below. Figure 10 plots training accuracy vs. sample size while Figure 11 plots validation accuracy vs. sample size.

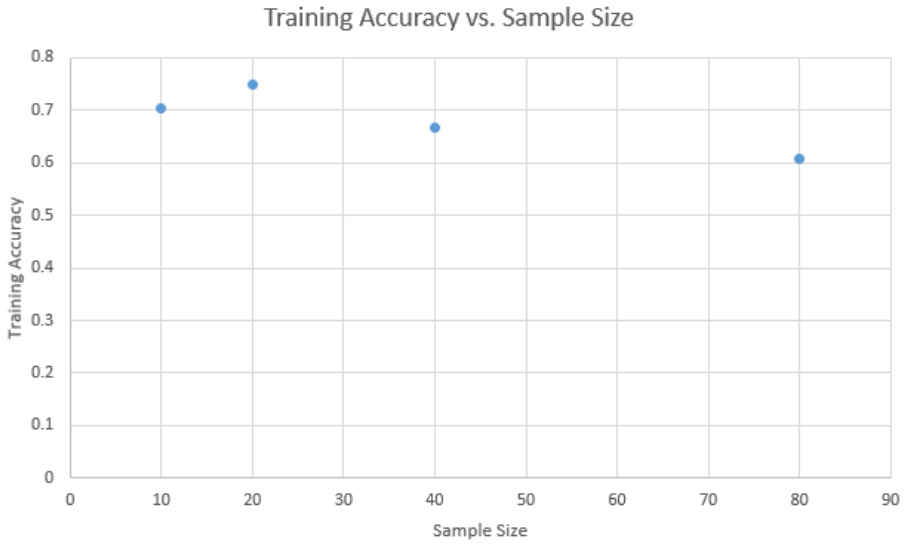


Figure 10: This plot shows training accuracy vs. sample size.

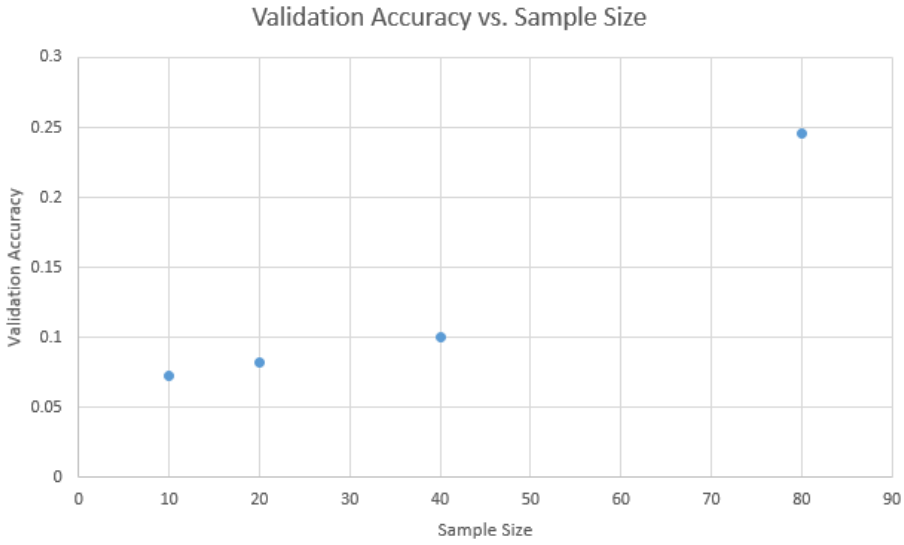


Figure 11: This plot shows validation accuracy vs. sample size.

(d) From the classification and error plots, we see the normal observations on the training set in that as we increase the number of iterations, the loss decreases and the classification accuracy increases. However, the values we achieved for classification accuracy and error on the training set were not very high and low respectively. The classification accuracy reached around 60%.

Classification and loss on the validation set produced even worse results. The results seemed to move almost randomly as the number of iterations increased, with the exception of a slight increase in the classification accuracy (which was our only indication that our neural network was working). However, in general there was a wide variation in the increase and decrease of the loss and classification accuracy per iteration. One conclusion from this is that the validation set may have been too small.

Simply changing the classifications of a few images due to a change in weights of our model could then lead to a drastic change in the percentage of images classified correctly/incorrectly since there are so few images in the set to begin with.

Along with the variation of increases and decreases in the results per iteration, the accuracy was quite low for the validation set, reaching to a maximum of about 30%. This may again have been to due to the small validation set, but the main reason is simply that training to the training set simply did not have a good effect on lowering the loss of the validation set. So for some reason, the training set did not accurately represent the images we had in our validation set. This could have been due to a number of things such as not enough training/validation samples or not shuffling our examples enough before splitting them into training and validation sets.

Another observation to make is that we were simply training a softmax layer based off the outputs of a different convolutional neural network. It makes sense that we would get worse results than the original network where the last layer was not replaced with our softmax layer. This is because the original network could update its weights correctly based off of the results it got from its last layer while in our case we have a two step process of first training the original network, and then training the softmax layer based off of the outputs of the already trained original network. If the original network cannot update its weights based off of the results obtained by the softmax layer, it is obvious that it should perform worse than the original convolutional neural network.

Finally is the discussion about the plots of classification accuracy vs. sample size. Changing the sample size did not seem to have much effect on the classification accuracy for the training set. As we can see from the above plot, the accuracy stays around the same values for the training accuracy. However, for the validation set, the accuracy increases as we increase the sample size. This makes sense as the more samples we use for the training set, the better we can represent the validation set correctly. Obviously if the images of the validation and training set are from the same set of images, then more images used from this set in the training set will better represent the validation set.

(e) We then Visualized the filters from the first and last Convolution Layers of the trained model. The first layer has 64 filters in Figure 12 while the last convolution layer has 512 filters in Figure 13.

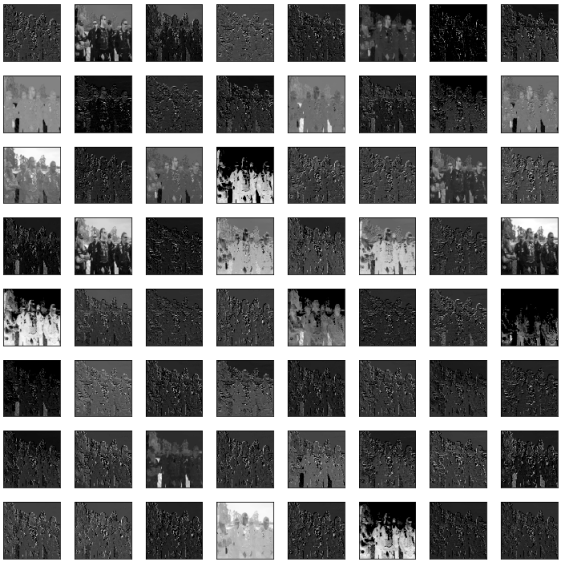


Figure 12: This is a visualization of the 64 filters which are 224 x 224 each from the first convolution later

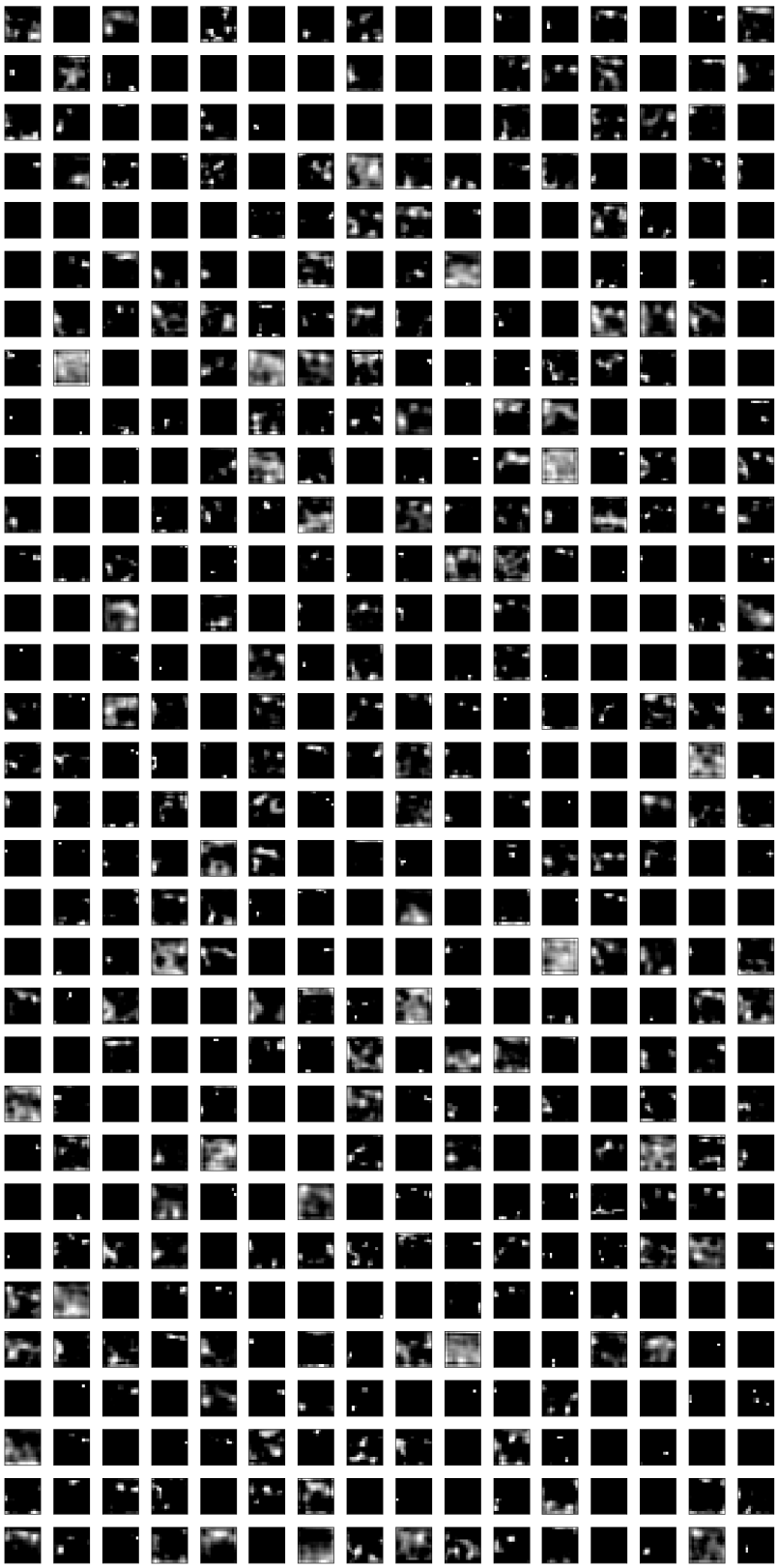


Figure 13: This is a visualization of the 512 filters which are 14 x 14 each from the last convolution later

5. **Feature Extraction**

(a) First we had to experiment using the intermediate Convolutional Layers as input to the Softmax Layer. For this case, we choose a layer right in the middle of the whole stack and went with layer 10. This layer is named "block3_pool".

(b) Then we needed train the network again on a subset of the data. In order to make our Softmax Layer be able to train, we needed to first flatten the input using a Flatten Layer.

(c) Below is our results from this experiment. The first two figures show training accuracy vs iterations (Figure 14) and validation accuracy vs iterations (Figure 15). The following two figures show training loss vs iterations and (Figure 16) validation loss vs iterations (Figure 17).

Based off of our results, we can see that our Softmax Layer barely learns anything when its input is from an intermediate Convolution Layer of VGG16. We believe the reason for this is that when we take the output of an intermediate Convolutional Layer, we are missing the other features that VGG16 learns later on that helps our Softmax Layer. Without these additional features that help tell the images apart, our Softmax Layer does a horrible job at classifying with a validation accuracy of just 9%.

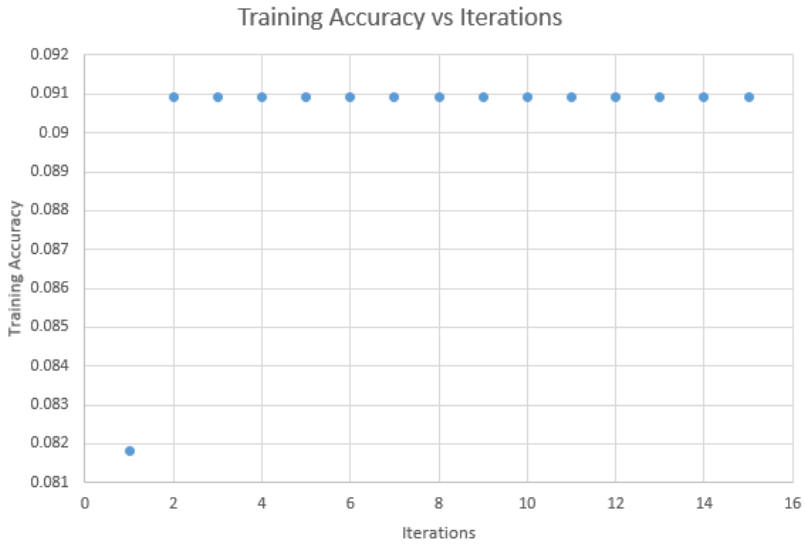


Figure 14: This plot shows training accuracy vs. iterations.

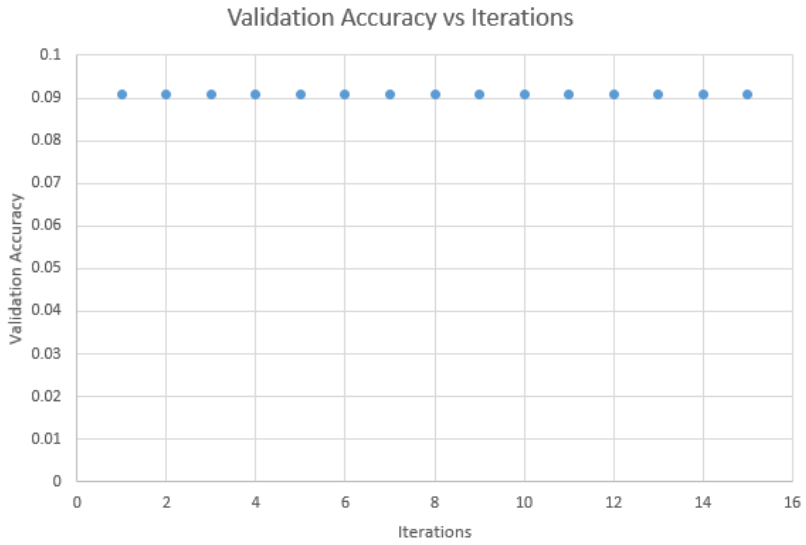


Figure 15: This plot shows validation accuracy vs. iterations.

Figure 16: This plot shows training loss vs. iterations.

Figure 17: This plot shows validation loss vs. iterations.

3 Temperature-based Softmax Regression

For this section, we will explore using the softmax layer of VGG16 as an input to our softmax layer for class prediction of Caltech256. In the previous problems, we excluded the VGG16's softmax layer from our model, because VGG16 was trained to classify images into a different set of classes than ours. In this exercise, we test whether the original VGG16 categories are similar enough to ours to where their classification scores can be used to determine proper classifications into the Caltech256 categories. In particular, we will modify the output from the VGG16 softmax layer from

$$\frac{\exp(a_i)}{\sum_j \exp(a_j)} \text{ to } \frac{\exp(a_i/T)}{\sum_j exp(a_j/T)}.$$

This somewhat perturbs or softens the classification confidence produced by the VGG16 model, so that these adjusted predictions may be used to (hopefully) separate the classes we're attempting to discern.

1. As per section 1, we read in and prepared the Caltech256 dataset of images and labels.
2. Then, as per the previous sections, we created a Keras ConvNet model with weights from the VGG16 model.
3. Then, we modified the softmax with temperature parameter by adding a Lambda layer before predictions softmax layer of the VGG16 model. The Lambda layer scaled the outputs by $\frac{1}{T}$ before being inserted into the pre-trained VGG16 prediction layer.
4. Next, we added our softmax layer to predict class for Caltech256. The summary of our model is shown below:

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_2 (InputLayer)             (None, 4096)          0
____________________________________________________________________________________________________
lambda_1 (Lambda)                (None, 4096)          0           input_2[0][0]
____________________________________________________________________________________________________
predictions (Dense)              (None, 1000)          4097000     lambda_1[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 256)           256256      predictions[1][0]
====================================================================================================
Total params: 4,353,256
Trainable params: 256,256
Non-trainable params: 4,097,000
```

5. We tested this model for various values of the temperature parameter T using 16 training examples and 4 validation examples per class. Figures 18 and 19 show the validation set accuracy per epoch for predicting 64 classes and 256 classes, respectively.

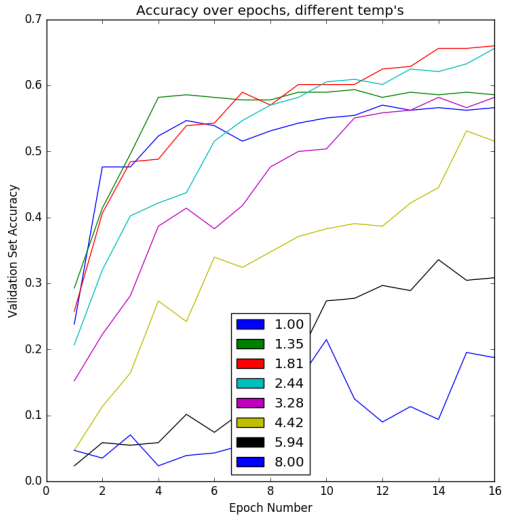


Figure 18: This plot shows validation accuracy vs. number of epochs for varying temperature T using 64 classes.

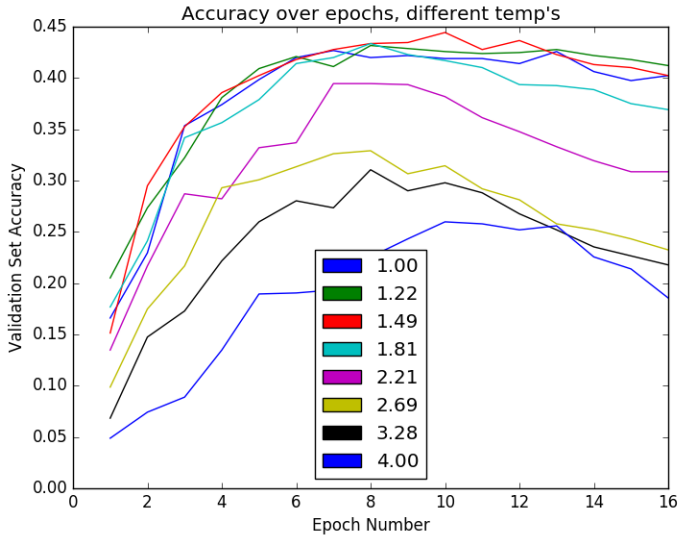


Figure 19: This plot shows validation accuracy vs. number of epochs for varying temperature T using 256 classes.

We plot the max and final accuracy with varying temperature T for 64 and 256 classes in figures 20 and 21, respectively, to better observe its affect on classification performance.

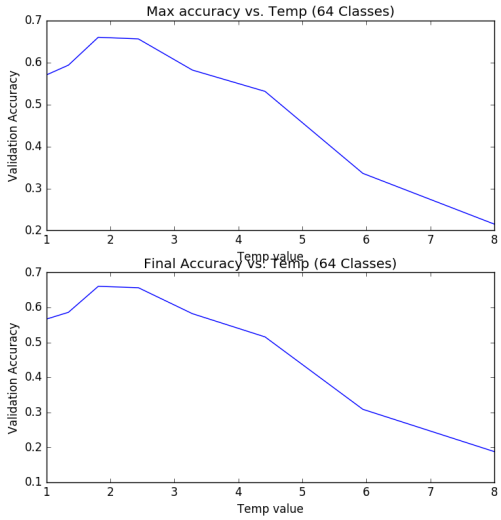


Figure 20: This plot shows max and final accuracy vs Temperature T for prediction of 64 classes.

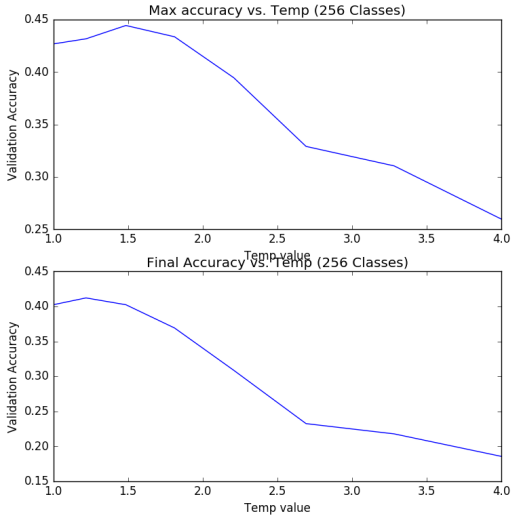


Figure 21: This plot shows max and final accuracy vs Temperature T for prediction of 256 classes.

6. According to figure 20, a temperature parameter T of 1.81 was observed to be the best case for our classification of CalTech256 with only 64 classes, while with 256 classes, shown in figure 21, a temperature parameter of 1.49 gave the max accuracy of almost 45%, and a temperature of 1.22 gave the best final accuracy after 16 epochs. In general, we see that for 64 classes, a temperature around 2 helps, however, for 256 classes, the temperatures after around 1.5 hurt the performance. Overall, this implementation of the temperature-based softmax using the VGG16 softmax layer as input to our softmax provided worse classification results than replacing the original VGG16 softmax layer.

   This result is not so surprising – the softmax layer from the VGG16 model is specifically trained to classify images into categories *different from* the categories specified by the CalTech256 dataset. Therefore, using those classification scores as a means to predict

CalTech256 classes seems intuitively to be throwing away some information in the hopes that the previous categories have some bearing on the classes at hand. Indeed, in our experiment, we found that a small tweak (that is, implementing the temperature parameter) gets somewhat usable predictions, but using the somewhat more ”raw” features that appear in the VGG16 model prior to the final prediction layer is more effective.