CSE 253 Homework Assignment 3
Transfer Learning

Sainan Liu
A13291871
sal131@eng.ucsd.edu

Hao-en Sung
A53204772
wrangle1005@gmail.com

Shiwei Song
A53206591
shs163@eng.ucsd.edu

Haifeng Huang
A53208823
hah086@eng.ucsd.edu

Abstract

In this assignment, we are asked to use VGG16 model[1] on Caltech256[2] and Urban Tribes[3] data. The method we use is to add a final softmax layer with the correct number of output units, and only train the last layer while maintain the weights from all previous layers. Using these method, we are able to achieve 50% accuracy with less than 4 samples per class and 73% accuracy with 32 samples per class for Caltech256 with 20 epochs. For Urban Tribes dataset, we are able to achieve 50% accuracy with 16 or 32 instances per class. Then we used the last 3 intermediate convolutional layers as input to the softmax layer respectively. With the last intermediate convolutional layer, we can still achieve 50% accuracy with 30 epochs. Finally, we trained the network without removing the last layer of the original VGG16 model and changed the activation function to a temperature-based layer. We got 50% accuracy on Caltech256 with parameter $T = 1$.

1 Caltech256 Classification

1.1 Introduction

We have trained a ConvNet with VGG16 model for Caltech256 dataset. By using existing trained weights of the VGG16 model for all previous layers, we compared the performance of the network with 2, 4, 8, 16 and 32 samples per class respectively as training samples, and evaluated the output of the last 3 intermediate convolutional layers with 32 samples per class as the training set.

1.2 Method

The Caltech256 dataset contains 257 folders. We have excluded the last folder named "clutter" in this experiment. The remaining 256 classes include a total of 29780 samples. For the ConvNet, we replaced the output layer of VGG16 model with softmax layer, which has 256 outputs. To achieve at least 50% of the accuracy, we used 2, 4, 8, 16 and 32 samples per class respectively for training. The tool we use was keras[4]. Keras function fit_generator and our own generator with a batch size of 32 were used for this experiment. Early stopping with 6 iterations of continuous non-improvement epochs was used for all categories with 24 random samples per class as validation set first. Then a maximum epoch of 20 was used for the report. For the test set, we have randomly collected 24 samples per class so that it covers 20% of the total dataset. None of the test set, validation set, and training set overlaps with each other. Training parameters included: learning rate = 0.001 with RMSprop as the optimizer.

For fine tuning, we used the last 3 intermediate convolution layers as input to the softmax layer respectively and trained the network again on 32 samples per class for Caltech256 for 30 epochs. Learning rate is 0.00001, 0.000001, and 0.0000001 respectively.

1.3 Results

(a) Loss graph for 32 samples per class

For training with 32 samples per class and testing with 24 samples per class, the training and testing loss over 20 iterations are shown in Figure 1.

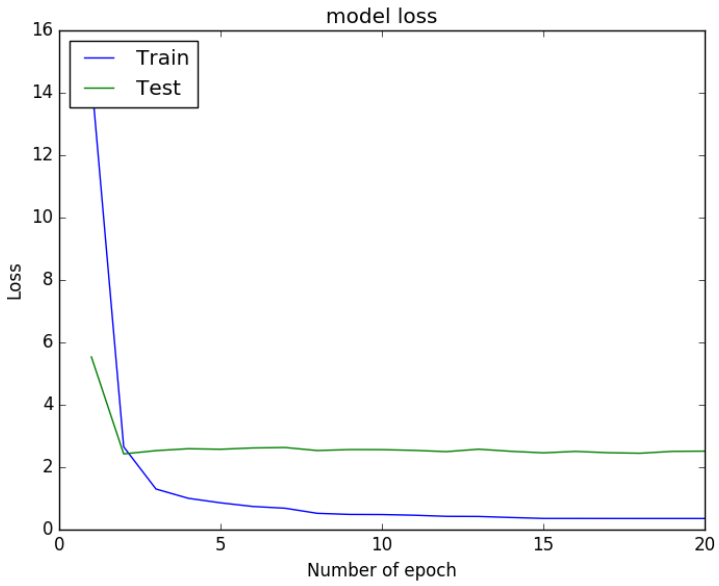


Figure 1: Caltech-256 classification loss over 20 epochs with 32 samples per class for training and 20 samples per class for validation.

(b) Accuracy graph for 32 samples per class

For training with 32 samples per class and testing with 24 samples per class, the classification train and test accuracy over 20 iterations are shown in Figure 2.

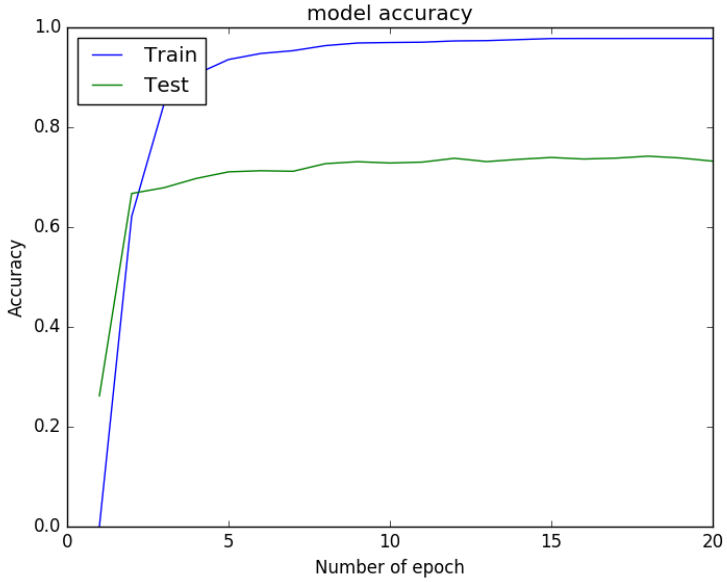


Figure 2: Caltech-256 classification accuracy over 20 iterations with 32 samples per class for training, and 10 samples per class for validation.

(c) Accuracy graph for 2, 4, 8, 16 and 32 samples per class

Classification test accuracies over 2, 4, 8, 16 and 32 samples used per class is shown in Figure 3. The final test results are shown in Table 1.

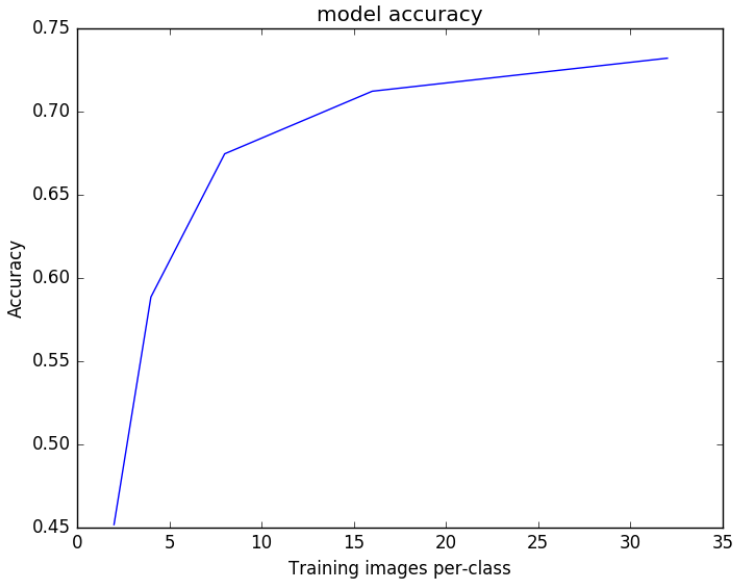


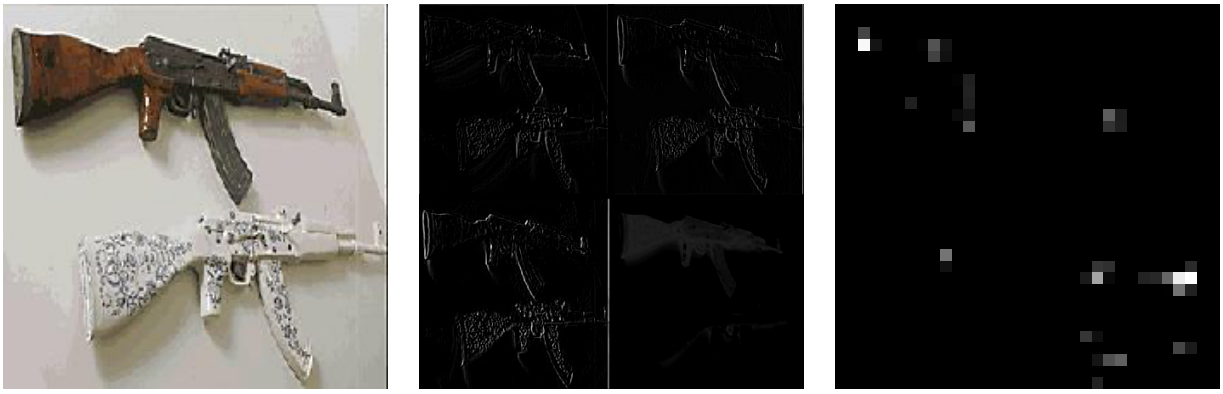
Figure 3: Caltech-256 classification test performance as the number of training images per class is varied.

Table 1: Test Accuracies for different number of samples per class in training.

# samples per class:	2	4	8	16	32
Test Accuracies:	0.4518	0.5885	0.6746	0.7112	0.7321

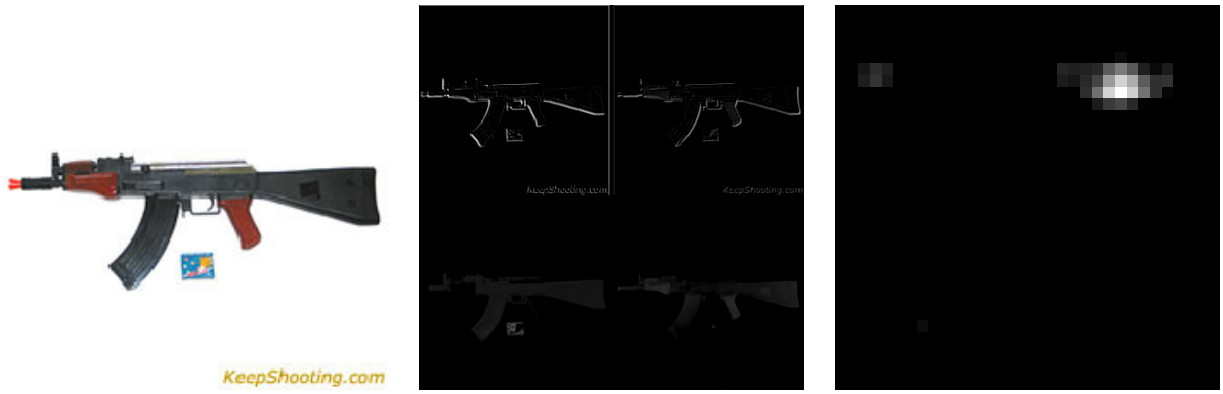
(d) Feature maps for first and last conv layers

3 randomly selected test images and 4 of their randomly selected feature maps from first and last convolution layer of the trained model are shown in Figure 4, 5 and 6.



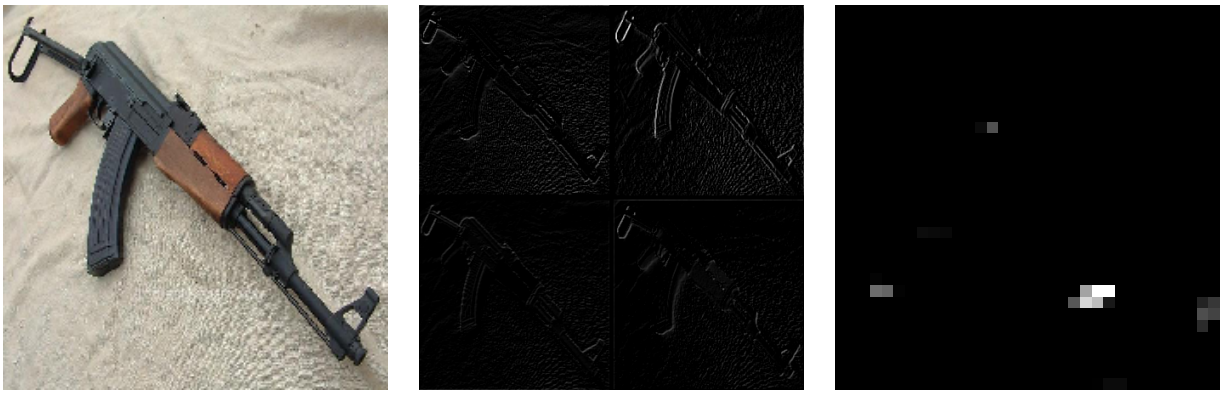
original image 1 block1_conv1 filter block5_conv3 filter

Figure 4: Selected 4 filter maps from the first and last convolution layer of the trained model for test image 1.



original image 1 block1_conv1 filter block5_conv3 filter

Figure 5: Selected 4 filter maps from the first and last convolution layer of the trained model for test image 2.



original image 1 | block1_conv1 filter | block5_conv3 filter

Figure 6: Selected 4 filter maps from the first and last convolution layer of the trained model for test image 3.

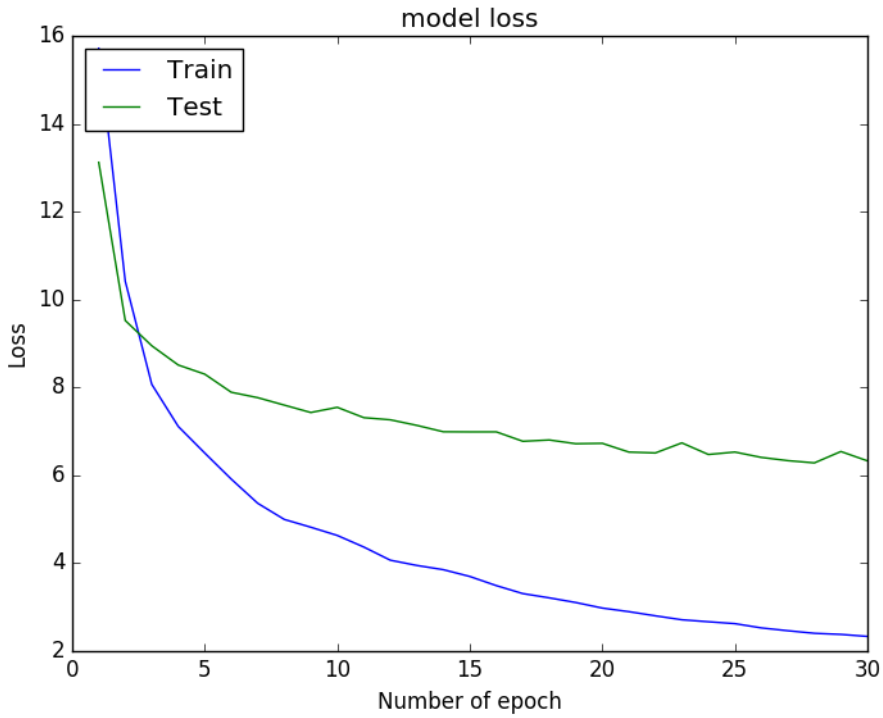
(e) Intermediate layers

The loss and accuracies over epoches using intermediate Convolutional Layer block5_conv3, block5_conv2, and block5_conv1 with 32 samples per class for training and 24 samples per class for testing over 30 epoches are shown in Figure 7, 8, and 9.

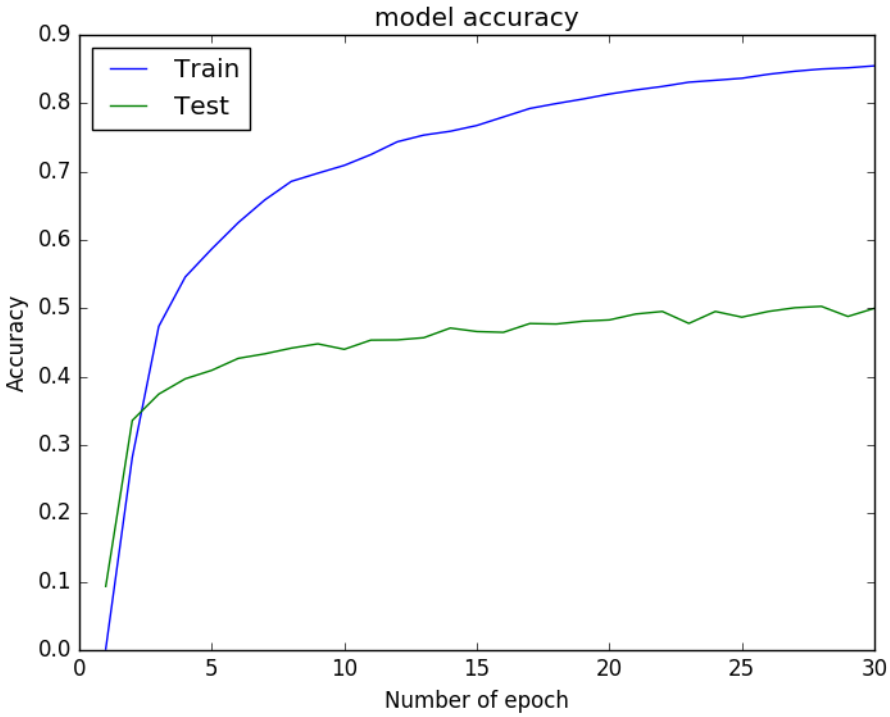
Trainable parameters for original modified model was :1,048,832. Directly getting information from intermediate layers need trainable parameters to be: 25,690,368 Non-trainable parameters dropped from 134,260,544 to 9,995,072 for block5_conv1 layer, to 12,354,880 for block5_conv2 layer, and to 14,714,688 for block5_conv3 layer.

Table 2: Test Accuracies for different intermediate layers.

# layer name:	Train Accuracy	Tain Loss	Test Accuracy	Test Loss
block5_conv3:	0.0132	15.9056	0.0107	15.9429
block5_conv2:	0.1315	13.9779	0.0916	14.5963
block5_conv1:	0.8547	2.3343	0.5000	6.3298

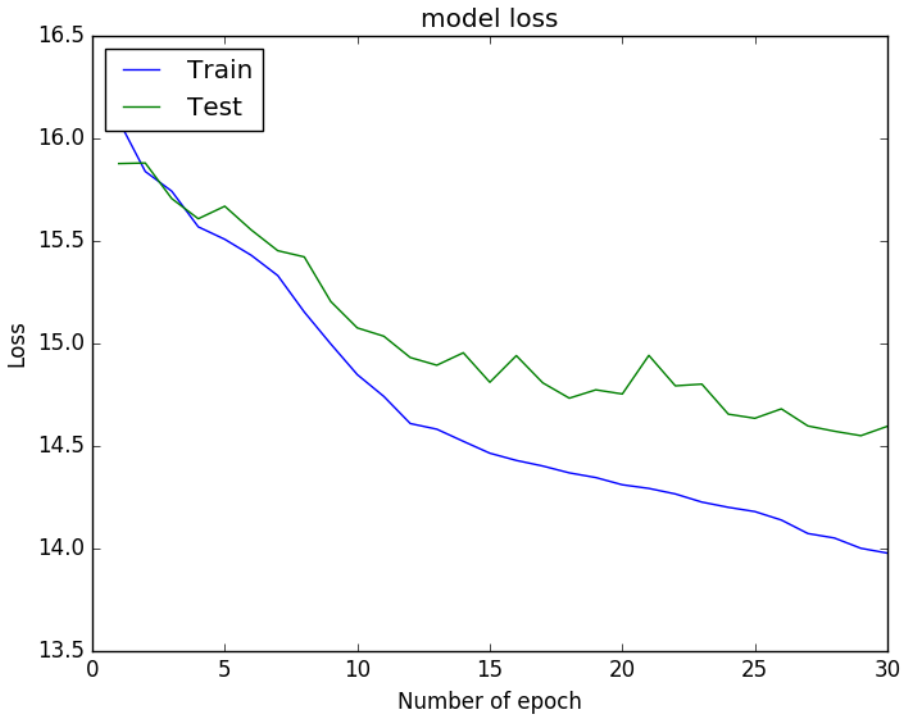


block5_conv3 loss

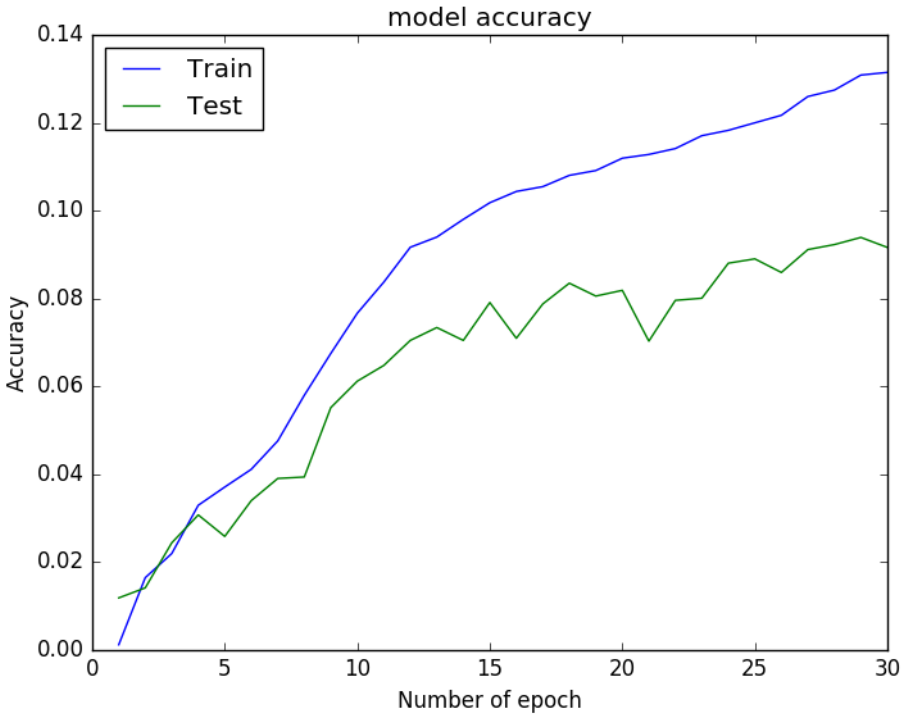


block5_conv3 accuracies

Figure 7: Caltech-256 classification loss and accuracies over 30 epoches with 32 samples per class for training and 24 samples per class for validation with the direct output of bloc5_conv3 layer.

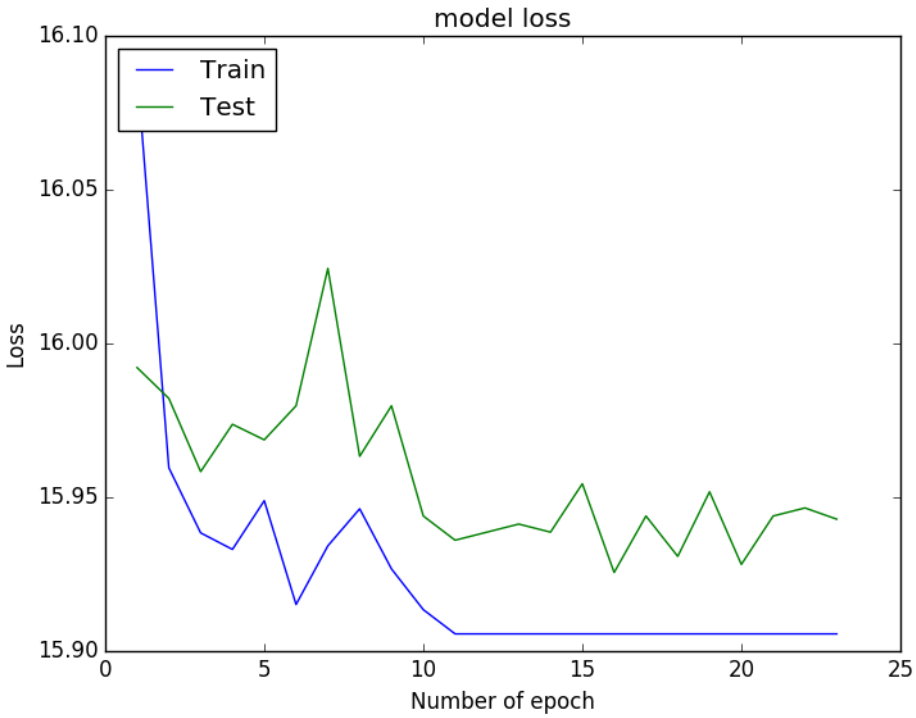


block5_conv3 loss

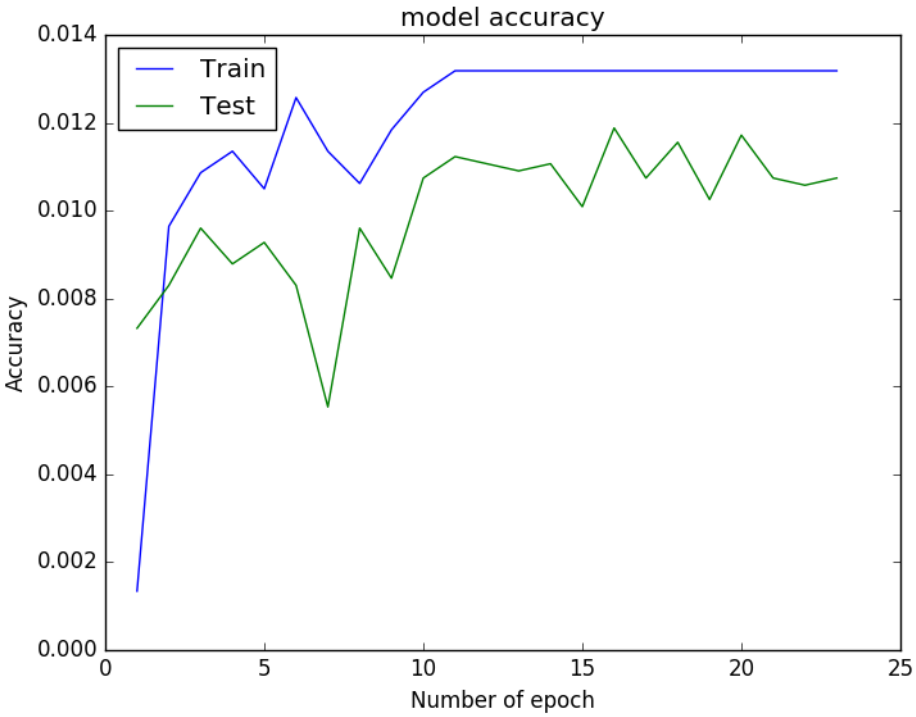


block5_conv3 accuracies

Figure 8: Caltech-256 classification loss and accuracies over 30 epoches with 32 samples per class for training and 24 samples per class for validation with the direct output of bloc5_conv2 layer.



block5_conv3 loss



block5_conv3 accuracies

Figure 9: Caltech-256 classification loss and accuracies over 30 epoches with 32 samples per class for training and 24 samples per class for validation with the direct output of bloc5_conv1 layer.

1.4 Discussion

(a, b) Loss and Accuracy graph for 32 samples per class

The training and test performance (loss in Figure 1 and accuracies in Figure 2) vs iterations show that the test and training performance increased rapidly before 5 epochs, then both come to a very slow growth, where training results outperforms the test results, which is expected. The early stopping result from previous test (not shown here) indicates that overfitting happens after 20 epochs, which means that the overfitting happened after 20 epochs. It seems like with previously trained weights, the training process can overfit the data much faster than training the entire network from scratch comparing to how much time they spend on training the original VGG16 network. This may indicate that with previous weights, the units in the network already contains very useful information(features) for this image classification task, and it is generalizable for different images.

(c) Accuracy graph for 2, 4, 8, 16 and 32 samples per class

From Figure 3 and Table 1 we can see that the accuracy increases as the number of samples per class increases. This is because as the number of samples increases, more information is provided for the last layer to cater towards our classification task.

We also notice that an accuracy of 50% is reached with only 4 samples per class for this dataset with pre-trained feature extractor. This is consistent with what we see in figure 9 of the paper [5].

These results indicate that the last layer is more classification task specific, and it utilizes information from the pre-trained feature extractors. The previous layers mainly contains information of unique features for images in a more generic sense, and with a good set of pre-trained feature extractors, the classification task for new dataset becomes much easier.

(d) Feature maps for first and last conv layers

The visualizations of the feature maps in Figures 4 to 6 give us some information about what each filters do in the first and last layer of the network.

We notice that the features in the first layer of the network seem to detect different local features of the image: Some filters behave like an edge detector, whereas some filters behave like a it is detecting the dark/light colors of the image shown in the second images in Figure 4 to 6. They are not necessarily image specific locally, because they tend to activate over then entire gun shape in all filters. The features in the last layer of the network has a much smaller size, (14×14) in this case. Because there are 512 of such filters, we notice a lot of them tend to be detecting nothing, and some of them tend to light up for parts of the gun instead of the entire gun shape.

In summary, we learned that filters in the first convolutional layer usually captures much more local features, such as edge detection, or color detection, which means that different image may all have lots of activations in different units, but the number of activation may be similar based on the image. This explains why most of the randomly selected filters in the first layer has some level of activations over the entire gun object. On another hand, we learned that the filters in the last convolutional layer usually captures much more global features, such as if the gun contains a specific piece or not. This is especially obvious in the tope-left and top-right of the last image in Figure 4, the top-right of the 3rd image in Figure 5, and the bottom-right of the last image in Figure 6. These behaviours are expected because the first layer was only fed with local image patches, whereas the later layers are able to combine information from different filters from preivous layers. Utilizing these information allow the last layer to construct filters that is more global-feature targeted.

(e) Intermediate layers

From Figure 7 to 9, we can see that with all fully-connected layers removed, directly getting information from last convolutional layer reduces the accuracy from 73% to 50% after 30 epoch, and it dropped more significantly after we removed last convolutional layer. We notice that the loss of the test accuracy is still increasing at 30 epoch, which means that it has not yet been overfitted; however due to limited resources and time, we were not able to keep the program running until we see the

overfitting effect.Therefore, the final performance may be even better than this report. We also found that as we remove more convolutional layers from the network, the learning rate needs to be smaller.

We learned that with a roughly 80% non-trainable parameters reduction, and 24 times more trainable parameters, we can achieve at least 68% of the original accuracy. However, due to the large size of the trainable parameters, the training process becomes much slower, and with a bigger hardware constraint.

2 Urban Tribes Classification

2.1 Introduction

In this problem, we are going to investigate transfer learning in Urban Tribes dataset. In this dataset, there are 11 different categories of people. We are going to get the pretrained CNN models for image classification - VGG16 ConvNet, trained on ImageNet, and change the final fully connected softmax layer to our own softmax layer with 11 categorical labels. Then we use different sample sizes of Urban Tribes dataset to train our new model and evaluate the performance of this transfer learning.

2.2 Method

We are using Keras library and tensorflow as backend. Get the pretrained CNN models for image classification - VGG16 ConvNet, trained on ImageNet, and change the final fully connected softmax layer to our own softmax layer with 11 categorical labels. Then we use different sample sizes of Urban Tribes dataset to train our new model and evaluate the performance of this transfer learning.

2.3 Results and Discussion

2.3.1 Training

The more samples we use for each class, the longer it takes to train the model, and we can get lower loss and higher accuracy.

2.3.2 Inference

(a) For different number of samples used per class, i.e. 2, 4, 8, 16 or 32 samples, we plot the train and test loss vs. iterations in Figure.10 to Figure.14.

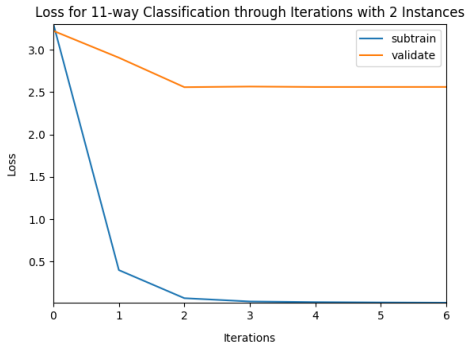


Figure 10: Model Loss through Iterations for 2 Samples per Class

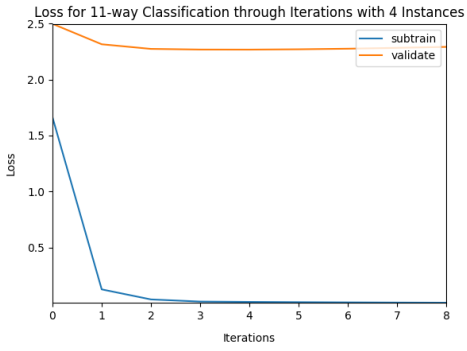


Figure 11: Model Loss through Iterations for 4 Samples per Class

(b) For different number of samples used per class, i.e. 2, 4, 8, 16 or 32 samples, we plot the train and test accuracy vs. iterations in Figure.15 to Figure.19.

(c) We plot classification accuracy vs. number of samples used per class in Figure.20.

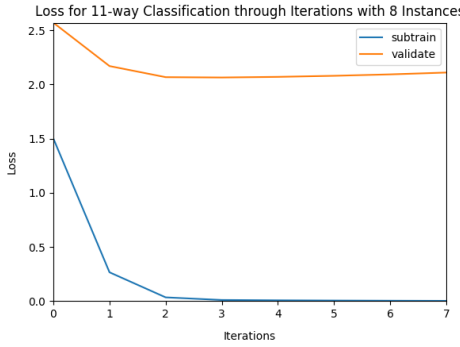


Figure 12: Model Loss through Iterations for 8 Samples per Class

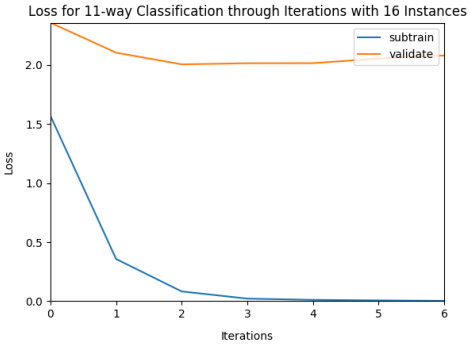


Figure 13: Model Loss through Iterations for 16 Samples per Class

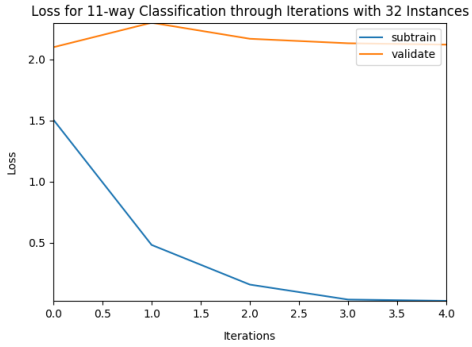


Figure 14: Model Loss through Iterations for 32 Samples per Class

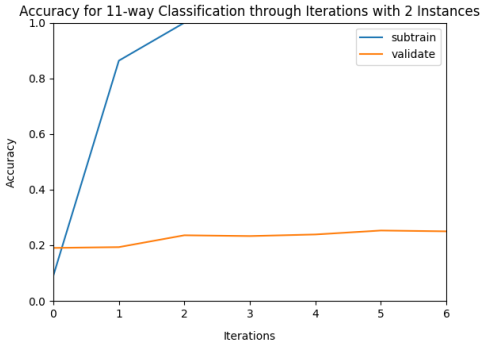


Figure 15: Model Accuracy through Iterations for 2 Samples per Class

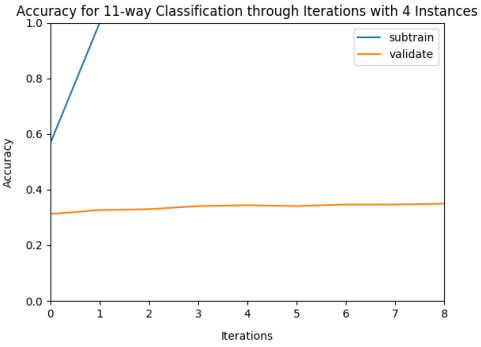


Figure 16: Model Accuracy through Iterations for 4 Samples per Class

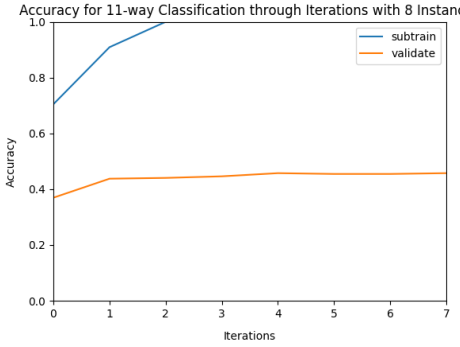


Figure 17: Model Accuracy through Iterations for 8 Samples per Class

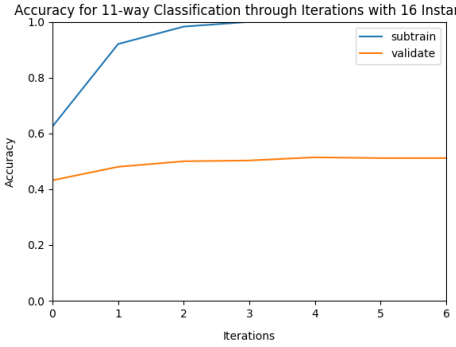


Figure 18: Model Accuracy through Iterations for 16 Samples per Class

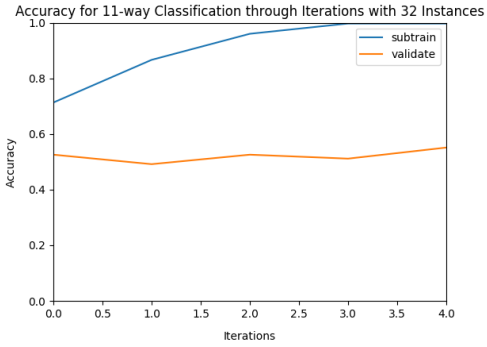


Figure 19: Model Accuracy through Iterations for 32 Samples per Class

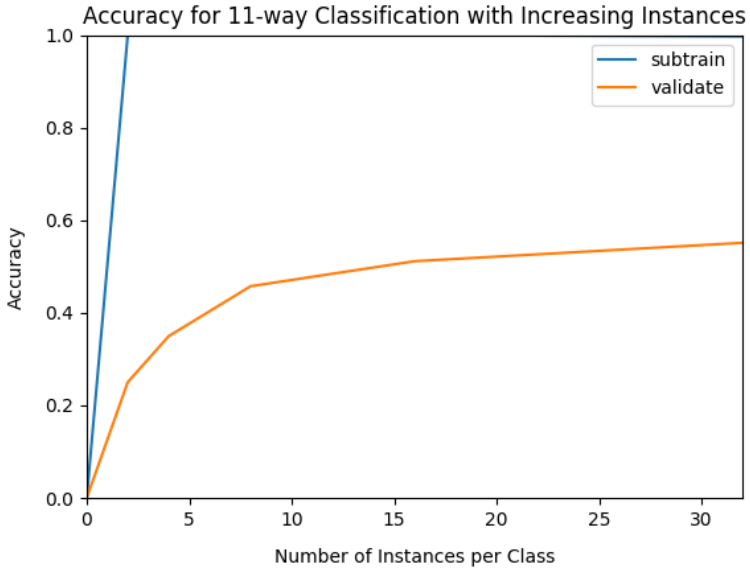


Figure 20: Classification Accuracy vs. Number of Samples Used per Class

(d) From Figure.10 to 14, we can see that after 3 iterations, the loss of training set is close to 0, and the loss of validation set (we treat it as test set) is almost unchanged after 2 iterations. The more samples we use for each class, the lower terminating loss is for validation set.

From Figure.15 to 19, we can see that after 3 iterations, the accuracy of training set is close to 1, and the accuracy of validation set (we treat it as test set) is almost unchanged after 2 iterations. The more samples we use for each class, the higher terminating accuracy is for validation set.

From Figure.20, we can see that for 2 samples per class, we can achieve accuracy 1 for training set. The more samples we use for each class, the higher terminating accuracy is for validation set.

Inference: Using more data to train the model, we can get better result. The performance of this model is worse than that of Caltech256 because the task is different between them: Caltech256 and ImageNet both use the shape to determine the category of objects, but the task of Urban Tribes is to recognize the identity of people, so it has less relation with the shape of the objects.

(e) To visualize filters, we select several images and print the activation outputs from the first and the last convolutional layers, which is shown in Figure.21 for the first convolution layer and Figure.22 for the last convolution layer.

From the output of the first convolution layer, we can see the edges of the people in the images. But form the last convolution layer, we can hardly tell any information.

Later on, we are interested in the filter performance. Hence, we build a loss function that maximizes the activation of the n-th filter of the layer considered, then compute the gradient of the input picture with this loss. We use some existing test images and run gradient ascent for 20 steps. If some filters get stuck to 0, we then just skip them. The filters that have the highest loss are assumed to be better-looking, so we will only keep the top 64 filters. The result for $block1\_conv1$ layer is in Figure.23 and the result for $block5\_conv3$ layer is in Figure.24.

If we start with a random image with some noise rather than real images, we can get the filters of the first and the last convolution layer in Figure.25 and Figure.26.

From the filters of the first convolution layer, we can see more general information of the image, and from the filters of the last convolution layer, we can see more specific inherent information of the image.

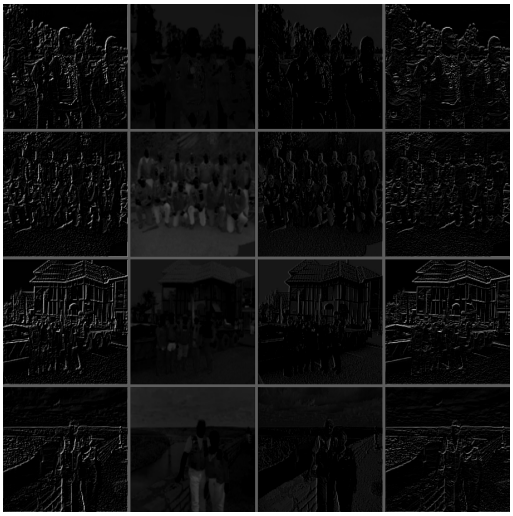


Figure 21: Output Visualization of First Convolution Layer

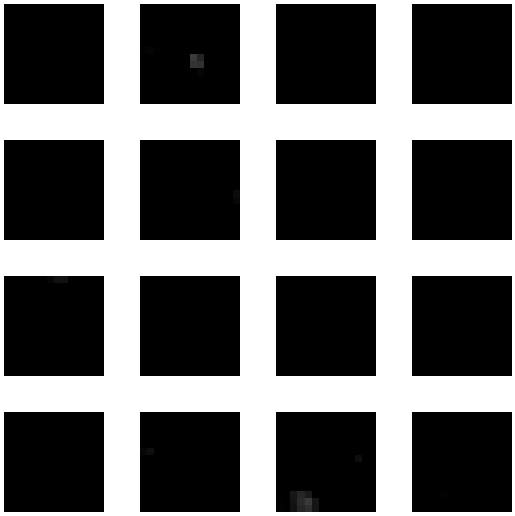


Figure 22: Output Visualization of Last Convolution Layer

Figure 23: Filter Visualization (Test Image) of First Convolution Layer

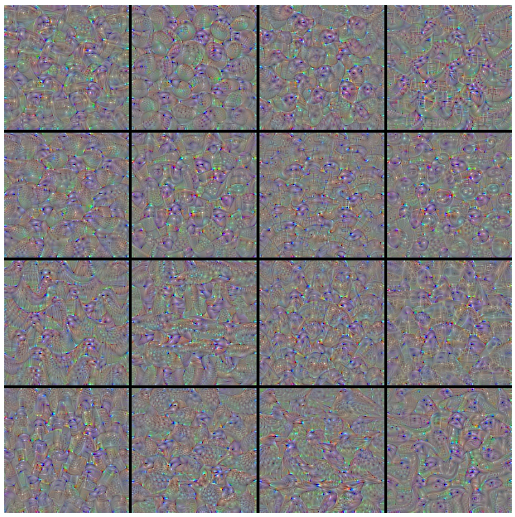


Figure 24: Filter Visualization (Test Image) of Last Convolution Layer

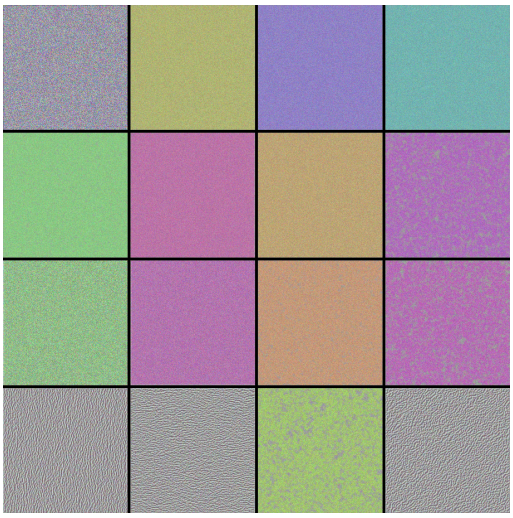


Figure 25: Filter Visualization (Random Image) of First Convolution Layer

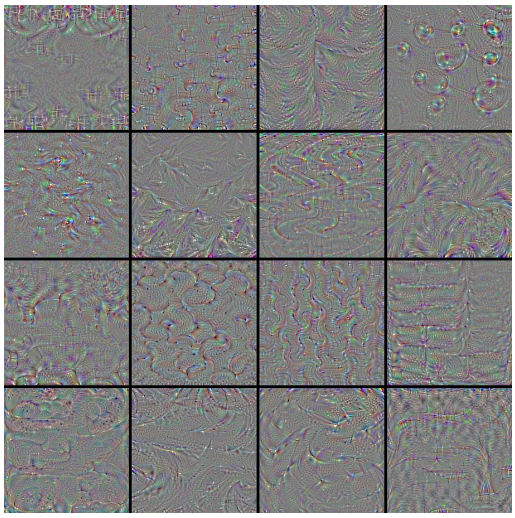


Figure 26: Filter Visualization (Random Image) of Last Convolution Layer

2.3.3 Feature Extraction

We experiment on three intermediate layers: $block5\_conv1$, $block5\_conv2$, $block5\_conv3$. After extracting those layers, we add a maxpooling layer, a flatten layer and a softmax layer after the model. Then we evaluate the accuracy of the new model on 32 images. The result is shown in Figure.27.

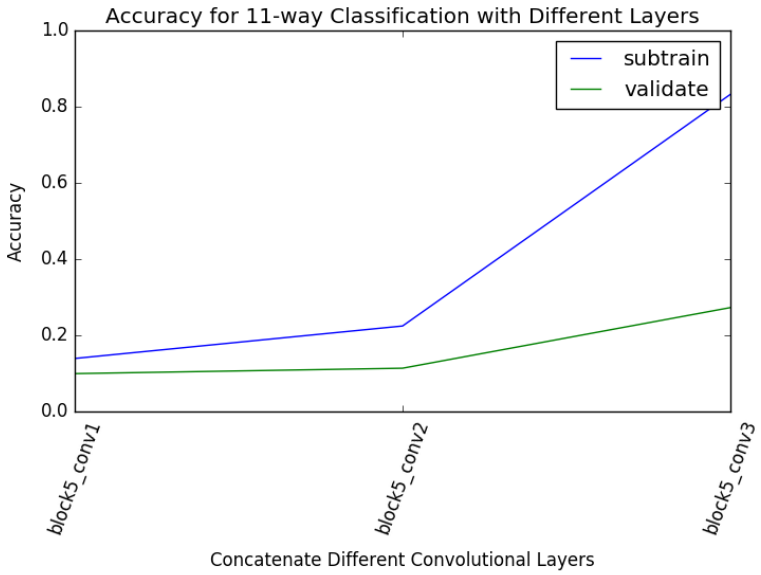


Figure 27: Intermediate Layer Performance on 32 Images

(c) We can see from Figure.27 that the more depth of the model, the high accuracy is. The reason is that the more depth of the model, the more inherent information of the image it can represent. Because the model is trained to accomplish the task of recognizing imageNet, while the task here is

to recognize the identity of people, the tasks are quite different, so we only get around 28% accuracy on test set.

3 Summary

For caltech256 data, we have trained the VGG16 model with a modified last layer and achieved 73% accuracy with 32 samples per class after 20 epochs. A 50% accuracy was achievable with only 4 samples per class with this pre-trained feature extractor which is consistant with the paper[5]. By looking at the feature maps, we learned that the first convolutional layer tends to learn more local features whereas the last convolutional layer tends to learn more global features. After connecting the softmax layer directly from the last 3 convolutional layers we found that with a 80% reduction in the non-trainable parameters, we can still achieve at least 50% accuracy when train the model with 32 samples per class, which is 68% of what it could have achieved before. However, removing more than one convolutional layers from the end seem to enlongate the training process, and with the limited time and resources we have, we found them to have a much larger sacrifies in accuracy.

4 Contributions

Sainan and Shiwei are mainly in charge of Caltech256 and bonus questions. Hao-en and Haifeng are mainly in charge of the Urban Tribes dataset.

5 Bonus: Temperature-based Softmax Regression

We modify the activation function of the VGG output layer adding a temperature parameter.

$$y_i = \frac{\exp(a_i/T)}{\sum_j \exp(a_j/T)}$$

Then we add an extra softmax layer to predict Caltech256 classes.

We try $T = 1, 2, 4, 8, 16$ under the same setting. We use 10 samples from each category and run 15 epochs with early stop for training. The result (accuracy on test set) we get is shown as follow:

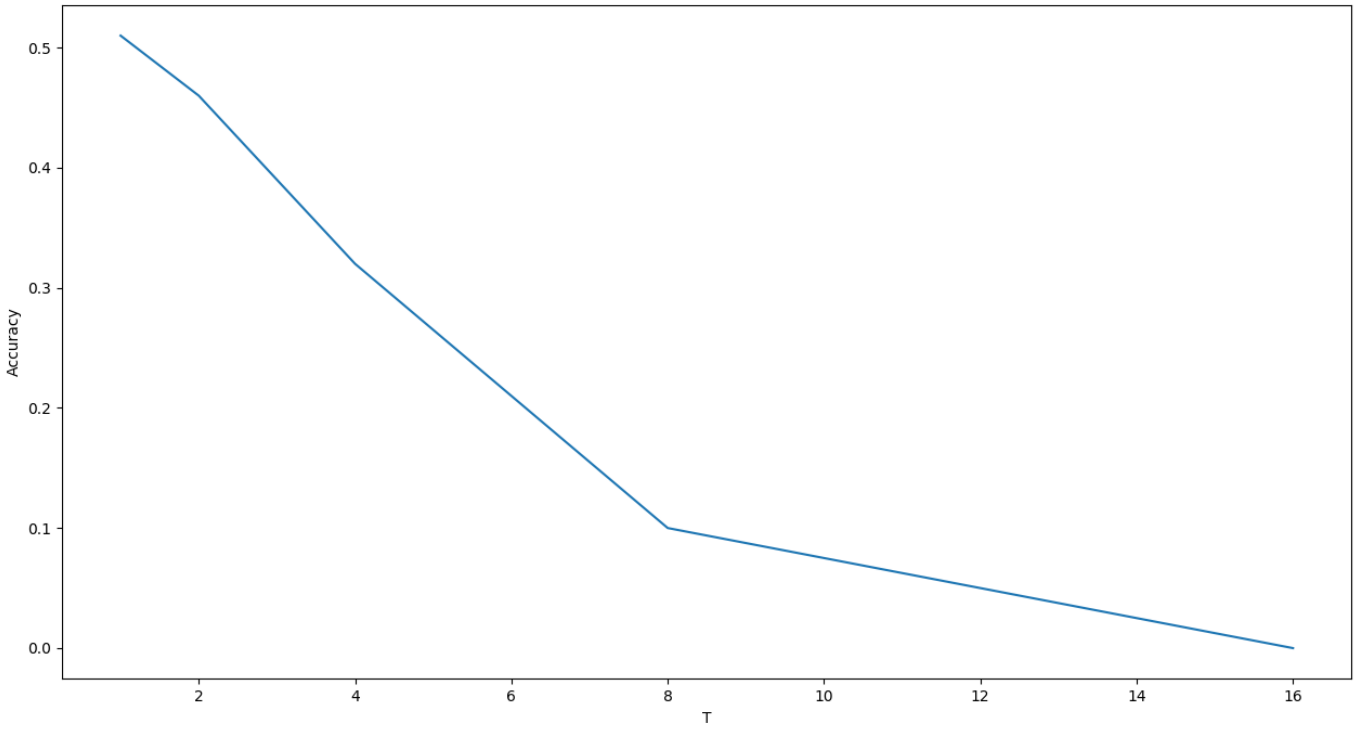


Figure 28: Accuracy on test set vs. T

Hence, the best value of T is 1. The corresponding accuracy we got is 0.51 on the test set. As T increases, the loss function will decrease slower and will lead to worse results.

Codes

Codes for Main Report

Listing 1: vgg16_model.py for cal256 dataset

```
from keras.applications import VGG16
from keras.layers import Dense, Flatten
from keras.models import Model, load_model
from keras.optimizers import RMSprop, SGD
from keras.callbacks import EarlyStopping

from os import makedirs
from os.path import isfile, isdir

from MyTensorBoard import TensorBoard

from vgg16_cal256_setup import getSampleSetupInfo, getTrainSetupInfo, getPathSetupInfo, getAn
from vgg16_cal256_utils import saveTempPkl, loadTempPkl, getCal256Info, prepData, getData, gen
from vgg16_cal256_report import savehistory, loadhistory, printpklhistory, savesummary, savefi

def getModel( output_dim, modelfile='../model/myvgg16.h5'):
    '''
        * output_dim: the number of classes (int)
        * return: compiled model (keras.engine.training.Model)
    '''
    if isfile(modelfile):
        # identical to the previous one
        tl_model = load_model(modelfile)
    else:

        vgg_model = VGG16( weights='imagenet', include_top=True )
        #Last FC layer's output
        vgg_out = vgg_model.layers[-2].output
        #Create softmax layer taking input as vgg_out
        softmax_layer = Dense(output_dim, activation='softmax')(vgg_out)
        #Create new transfer learning model
        tl_model = Model( input=vgg_model.input, output=softmax_layer )

        #Freeze all layers of VGG16 and Compile the model
        for layer in tl_model.layers:
            layer.trainable = False
        tl_model.layers[-1].trainable = True
        #Confirm the model is appropriate
        opt = RMSprop(lr=0.001)
        tl_model.compile(optimizer=opt,
                         loss='categorical_crossentropy',
                         metrics=['accuracy'])
        tl_model.summary()
        tl_model.save(modelfile)  # creates a HDF5 file

    # returns a compiled model
    return tl_model

def trainModel(trainN, valN, testN,  # samples per class count.
               tl_model, classNames, classCounts, # model and dataset info
               batchSize, maxEpoch, earlyStoppingMaxIcre, # training related
               resultDir, modelname, datasetname, verboseFlag=2, earlystopping=False):# saving
    # set up filenames for storage.
    weighth5path, historypklpath, _, partitionpklpath, accfigpath, lossfigpath, summarytxtpath

    z_train, z_valid, z_test = prepData(trainN, valN, testN, classCounts, partitionpklpath)

    # Setup callbacks.
    bd = TensorBoard(log_dir='%s/%s_logs'%(resultDir, datasetname),
```

```
60                          histogram_freq=2,
61                          write_graph=True)
62      es = EarlyStopping(
63          monitor='val_loss',
64          min_delta=0, # minimum change for improvement
65          patience = earlyStoppingMaxIcre, # number of epochs with no improvement after which tr
66          verbose = verboseFlag, # print more information
67          mode='auto') # infer min from monitoring loss, and infer max for monitoring accuracy.
68      if earlystopping:
69          callbackList = [bd, es]
70      else:
71          callbackList = [bd]
72      output_dim = len(classCounts)
73      # Train the model
74      print "----------->>Training<<----------"
75      if not isfile(weighth5path) or not isfile(historypklpath):
76          #Train the model
77          history = tl_model.fit_generator(genSample(z_train, batchSize, classNames),
78                                           samples_per_epoch=output_dim*trainN,
79                                           nb_epoch=maxEpoch, verbose = verboseFlag,
80                                           callbacks=callbackList,
81                                           validation_data=genSample(z_valid, batchSize, classNa
82                                           nb_val_samples=output_dim*valN)
83          print 'Done training.'
84          print 'Start saving weights...'
85          tl_model.save_weights(weighth5path)
86          print 'Done saving weights.'
87          savehistory(history,
88                      historypklpath,
89                      accfigpath,
90                      lossfigpath,
91                      summarytxtpath)
92
93      else:
94          print 'Found history and weights, printing...'
95          loadhistory(historypklpath)
96
97          print "----------------------------------------"
98
99  def testModel(z_test, classNames, resultDir, modelname, datasetname, batchSize,maxEpoch,earlyS
100     #Test the model
101     _, _, testresultpklpath, _, _, _, _ = getPathSetupInfo(resultDir, modelname, datasetname,
102     testN = len(z_test)
103     if ispath(weighth5path):
104         print 'Found previous weights, and load from %s'%(weighth5path)
105         tl_model = getModel(output_dim)
106         tl_model.load_weights(weighth5path)
107     output_dim = len(classNames)
108     if not isfile(testresultpklpath):
109         print "----------->>Testing<<----------"
110         print "Testing with %d samples per class with %d samples."%(testN, len(z_test))
111         testResult = tl_model.evaluate_generator(genSample(z_test, batchSize, classNames),
112                                                  val_samples=output_dim*testN)
113         zipresult = zip(testResult, tl_model.metrics_names)
114         saveTempPkl(zipresult, testresultpklpath)
115     else:
116         print "Found test result in %s"%(testresultpklpath)
117         zipresult = loadTempPkl(testresultpklpath)
118
119     for (v,n) in zipresult:
120         print(n + ": " + str(v))
121         print "----------------------------------------"
122
123 def getModelByLayer(layer_name, lr, output_dim, overwrite = False):
124     modelfile='../model/vgg16_%s.h5'%(layer_name)
```

```
125     print 'get model from %s'%(modelfile)
126     if isfile(modelfile) and not overwrite:
127         # identical to the previous one
128         tl_model = load_model(modelfile)
129 
130     else:
131         vgg_model = VGG16( weights='imagenet', include_top=True )
132         # get the symbolic outputs of each "key" layer (we gave them unique names).
133         layer_dict = dict([(layer.name, layer) for layer in vgg_model.layers[1:]])
134 
135         # Selected cnn layer output.
136         vgg_out = layer_dict[layer_name].output
137 
138         print layer_dict[layer_name].output_shape
139         flatten = Flatten()(vgg_out)
140         #Create softmax layer taking input as vgg_out
141         softmax_layer = Dense(output_dim, activation='softmax')(flatten)
142         #Create new transfer learning model
143         tl_model = Model(input=vgg_model.input, output=softmax_layer )
144 
145         #Freeze all layers of VGG16 and Compile the model
146         for layer in tl_model.layers:
147             layer.trainable = False
148         tl_model.layers[-1].trainable = True
149         #Confirm the model is appropriate
150         prop = RMSprop(lr=0.0001)
151         print "learning rate is:%0.4f"%(lr)
152         #sgd = SGD(lr=lr, decay=1e-6, momentum=0.9, nesterov=True)
153         tl_model.compile(optimizer=prop,
154                          loss='categorical_crossentropy',
155                          metrics=['accuracy'])
156         tl_model.save(modelfile)  # creates a HDF5 file
157     tl_model.summary()
158     return tl_model
159 
160 def trainByLayer(layer_name, lr):
161     classNames, classCounts = getCal256Info()
162     output_dim = len(classNames)
163     verboseFlag = 2
164     trainN, valN, testN, batchSize, maxEpoch, earlyStoppingMaxIncre, resultDir, modelname, dat
165     # get all training info.
166     if not isdir(resultDir):
167         makedirs(resultDir)
168     # get model.
169     l_model = getModelByLayer(layer_name, lr, output_dim)
170     # trainModel.
171     trainModel(trainN, valN, testN, l_model, classNames, classCounts, batchSize, maxEpoch, ear
172 
173 # Unit test.
174 if __name__ == '__main__':
175     model = getModel(256)
176     model.summary()
```

Listing 2: vgg16_cal256_utils.py for data extraction and generation

```
1 
2 # coding: utf-8
3 
4 # In[25]:
5 
6 
7 # coding: utf-8
8 
9 # In[25]:
10 
```

```
11 import cPickle as pickle
12 import numpy as np
13 import os
14 import progressbar as pb
15 import tarfile
16 import urllib
17
18 from keras.preprocessing import image
19 from keras.applications.vgg16 import preprocess_input
20 from keras.utils import np_utils
21 from random import shuffle
22
23 def downloadAndUntarFolder(url, directory, datasetname):
24     tarfilename = directory + datasetname + ".tar"
25     savefolder = directory + datasetname
26     # check if the folder exsits.
27     if not os.path.isdir(savefolder):
28         print savefolder + " does not exsit, check if file has been downloaded."
29         # download tar file first if the folder does not exist.
30         if not os.path.isfile(tarfilename):
31             print tarfilename + " does not exist."
32             # make data folder if not yet created.
33             if not os.path.isdir(directory):
34                 print directory + " does not exist, create the folder."
35                 os.mkdir(directory)
36             # download the file
37             print "downloading from: "
38             print url
39             urllib.urlretrieve(url, tarfilename)
40
41         else:
42             print tarfilename + " exists under" + directory
43         print "untar " + tarfilename
44         tar = tarfile.open(tarfilename)
45         tar.extractall(directory)
46         tar.close
47         print "done."
48
49 def saveTempPkl(dataset, pklfile):
50     f = file(pklfile, 'wb')
51     pickle.dump(dataset, f, protocol=pickle.HIGHEST_PROTOCOL)
52     f.close
53
54 def loadTempPkl(pklfile):
55     f = open(pklfile, 'rb')
56     dataset = pickle.load(f)
57     f.close
58     return dataset
59
60 def getImgInfo(imgTopDir, infotxt, infopkl, overwrite = False):
61     print "Reading information of images from " + imgTopDir
62     classNames = []
63     imgCounts = []
64     nClasses = 0
65     if not os.path.isfile(infotxt) or not os.path.isfile(infopkl) or overwrite:
66         bar = pb.ProgressBar(maxval=260).start()
67         i = 0
68         for subdir, dirs, files in os.walk(imgTopDir):
69             if i == 0:
70                 classNames = dirs
71                 classNames.sort()
72                 classNames = classNames[:-1] # drop clutters
73                 nClasses = len(classNames)
74                 imgCounts = np.zeros(nClasses)
75                 print "found ", nClasses, " classes"
```

```
76              for f in files:
77                  filepath = os.path.join(subdir, f)
78                  if not f.startswith(".") and not f.endswith(".mat") and subdir.split(".")[-1]
79                      labelpart = f.split("_")[0]
80                      try:
81                          labelidx = int(labelpart) - 1
82                          imgCounts[labelidx] += 1
83                      except:
84                          print labelpart, "returns error while being case to int, skip."
85              bar.update(i+1)
86              i += 1
87
88          bar.finish()
89          print "Done."
90          f = open(infotxt, 'w')
91          for i in range(nClasses):
92              f.write("{}\t\t\t{}\n".format(classNames[i], imgCounts[i]))
93
94          stats = classNames, imgCounts
95          saveTempPkl(stats, infopkl)
96      else:
97          print "found " + infotxt + " and " + infopkl
98          classNames, imgCounts = loadTempPkl(infopkl)
99
100     print "There are %d classes with a total of %d sample images\n"%(len(classNames), np.sum(i
101     return classNames, imgCounts
102
103 def partition(classCounts, trainNperClass, valNperClass, testNperClass):
104     print "partition image ids per class."
105     nClasses = len(classCounts)
106
107     leftTrainN = trainNperClass*nClasses
108     leftValN = valNperClass*nClasses
109     leftTestN = testNperClass*nClasses
110     z_train = []
111     z_valid = []
112     z_test = []
113
114     for i in range(nClasses):
115         # Randomize images per class.
116         classLabel = i+1
117         imgN = int(classCounts[i])
118         randIdSeq = np.random.permutation(imgN)
119
120         # take the first selectTrainN as training data. If there are less than 2 samples left
121         selectTrainN = int(min(leftTrainN, imgN-2, trainNperClass))
122         leftTrainN -= selectTrainN
123         # take the next selectValN as validation. If what is left is not enough for validation
124         selectValN = int(min(leftValN, max(1, imgN - selectTrainN), valNperClass))
125         leftValN -= selectValN
126         # take the last selectTestN as test. If what is left is not enough for testing, force
127         selectTestN = int(min(leftTestN, max(1, imgN - selectTrainN - selectValN), testNperCla
128         leftTestN -= selectTestN
129
130         temp_z_train = zip(np.ones(selectTrainN)*i, randIdSeq[:selectTrainN])
131         temp_z_valid = zip(np.ones(selectValN)*i,
132                            randIdSeq[selectTrainN:selectTrainN+selectValN])
133         temp_z_test = zip(np.ones(selectTestN)*i,
134                           randIdSeq[selectTrainN+selectValN:selectTrainN+selectValN+selectTest
135
136         z_train.extend(temp_z_train)
137         z_valid.extend(temp_z_valid)
138         z_test.extend(temp_z_test)
139
140     accTrainCount = len(z_train)
```

```
141     accValCount = len(z_valid)
142     accTestCount = len(z_test)
143 
144     print "selected %d trainIds, %d valIds, %d testIds"%(accTrainCount, accValCount, accTestCo
145     return z_train, z_valid, z_test
146 
147 def getCal256Info():
148     # check if the data has been downloaded.
149     url = "http://www.vision.caltech.edu/Image_Datasets/Caltech256/256_ObjectCategories.tar"
150     saveDir = "../data/"
151     zipFolderName = "256_ObjectCategories"
152     downloadAndUntarFolder(url, saveDir, zipFolderName)
153 
154     # get information from the image directories.
155     imgTopDir = saveDir + zipFolderName
156     infotxt = "{}_summary.txt".format(imgTopDir)
157     infopkl = "{}_summary.pkl".format(imgTopDir)
158     classNames, classCounts = getImgInfo(imgTopDir, infotxt, infopkl)
159     return classNames, classCounts
160 
161 def prepData(trainN, valN, testN, classCounts, partitionpklpath, overwrite=False):
162     if not os.path.isfile(partitionpklpath) or overwrite:
163         z_train, z_valid, z_test = partition(classCounts, trainN, valN, testN)
164         data = z_train, z_valid, z_test
165         saveTempPkl(data, partitionpklpath)
166     else:
167         z_train, z_valid, z_test = loadTempPkl(partitionpklpath)
168     return z_train, z_valid, z_test
169 
170 def getData(z, classNames, imgFolder= '../data/256_ObjectCategories', targetsize=(224, 224), c
171     nClasses = len(classNames)
172     X = np.empty((0, targetsize[0], targetsize[1], channels))
173     y = np.empty(0, dtype=np.int64)
174 
175     for i in range(len(z)):
176         classId = int(z[i][0])
177         classLabel = int(classId + 1)
178         imgId = int(z[i][1])
179         imgLabel = int(imgId+1)
180         y = np.append(y, [classId],axis=0)
181 
182         imgpath = "%s/%s/%03d_%04d.jpg"%(imgFolder, classNames[classId], classLabel, imgLabel)
183         img = image.load_img(imgpath, False, target_size = targetsize)
184         x = image.img_to_array(img)
185         x = np.expand_dims(x, axis =0)
186         x = preprocess_input(x, dim_ordering='tf')
187         X = np.append(X, x, axis=0)
188     yhot = np_utils.to_categorical(y, nClasses)
189 
190     return X, yhot
191 
192 def genSample(z, batch_size, classNames):
193     i = 0
194     while 1:
195         X, y = getData(z[i:i+batch_size], classNames)
196         i = i + batch_size
197         if i + batch_size > len(z):
198             shuffle(z)
199             i = 0
200         yield X, y
201 
202 #unit test
203 if __name__ == '__main__':
204     getCal256Info()
```

Listing 3: vgg16_cal256_train.py for training

```
from os.path import isdir
from vgg16_cal256_setup import getSampleSetupInfo, getTrainSetupInfo
from vgg16_cal256_utils import getCal256Info
from vgg16_model import getModel, trainModel, testModel

if __name__ == '__main__':
    # Output dim for your dataset

    # Setup parameters.
    trainNs, valNs, testNs = getSampleSetupInfo()
    classNames, classCounts = getCal256Info()
    output_dim = len(classNames)
    verboseFlag = 2

    Ns = zip(trainNs, valNs, testNs)
    for N in Ns:
        trainN = N[0]
        valN = N[1]
        testN = N[2]
        batchSize, maxEpoch, earlyStoppingMaxIcre,resultDir, modelname, datasetname = getTrain
        if not isdir(resultDir):
            makedirs(resultDir)

        print 'Start training...'
        # Load a new model every time.
        tl_model = getModel(output_dim)

        trainModel(trainN, valN, testN, tl_model, classNames, classCounts, batchSize, maxEpoch

        test = False
        if test:
            testModel(z_test, classNames, resultDir, modelname, datasetname, batchSize,maxEpoc
```

Listing 4: vgg16_cal256_setup.py for training parameters

```
from os import makedirs
from os.path import isdir

def getSampleSetupInfo():
    trainNs = [2, 4, 8, 16, 32]
    # 256 classes, 29780 sample images, 29780*20%/256 = 23.26
    valNs = [24, 24, 24, 24, 24]
    testNs = [24, 24 ,24 ,24 ,24]
    return trainNs, valNs, testNs

def getTrainSetupInfo():
    batchSize = 32
    maxEpoch = 20
    earlyStoppingMaxIncre = 6
    resultDir = '../result'
    modelname = 'vgg16'
    datasetname = 'cal256'
    #output_dim = 256 #For Caltech256

    return batchSize, maxEpoch, earlyStoppingMaxIncre, resultDir, modelname, datasetname

def getImgPathsInfo(z, classNames):
    imgFolder = '../data/256_ObjectCategories'
    paths  =[]
    for i in range(len(z)):
        classId = int(z[i][0])
        classLabel = int(classId + 1)
        imgId = int(z[i][1])
        imgLabel = int(imgId+1)
```

```
30          imgpath = "%s/%s/%03d_%04d.jpg"%(imgFolder, classNames[classId], classLabel, imgLabel)
31          paths.append(imgpath)
32      return paths
33
34  def getPathSetupInfo(topdir, modelname, datasetname,
35                       batchSize, maxEpoch, earlyStoppingMaxIncre, trainN):
36      prefix = '%s/%s_%s_result/'%(topdir,
37                                   modelname,
38                                   datasetname)
39      if not isdir(prefix):
40          makedirs(prefix)
41      postfix = '_bs%d_me%d_esi%d_train%d'%(batchSize, maxEpoch,
42                                            earlyStoppingMaxIncre, trainN)
43      weighth5path = '%sweights%s.h5'%(prefix, postfix)
44      trainresultpklpath = '%strainresult%s.pkl'%(prefix, postfix)
45      accfigpath = '%saccs%s.png'%(prefix, postfix)
46      lossfigpath = '%sloss%s.png'%(prefix, postfix)
47      summarytxtpath = '%ssummary%s.txt'%(prefix, postfix)
48      testresultpklpath = '%stestresult%s.pkl'%(prefix, postfix)
49      partitionpklpath = '%spartition%s.pkl'%(prefix, postfix)
50      return weighth5path, trainresultpklpath, testresultpklpath, partitionpklpath, accfigpath,
51
52  def getFinalReportFigInfo(resultDir, qname):
53      return '%s/vgg16_cal256_figs/%s.png'%(resultDir, qname)
54
55  def getAnalysisSetupInfo(layer_name):
56      trainN = 32
57      valN = 24
58      testN = 24
59      batchSize = 32
60      maxEpoch = 30
61      earlyStoppingMaxIncre = 6
62      resultDir = '../result'
63      modelname = 'vgg16_%s'%(layer_name)
64      datasetname = 'cal256'
65      return trainN, valN, testN, batchSize, maxEpoch, earlyStoppingMaxIncre, resultDir, modelna
66
67  def getLayerNamesInfo():
68      return ['block5_conv3', 'block5_conv2', 'block5_conv1']
```

Listing 5: vgg16_cal256_report.py for generating reports

```
1  from os.path import isfile, split, basename
2  from os import rename
3  from shutil import copy
4  import matplotlib.pyplot as plt
5  from scipy.misc import imsave
6  from keras.applications import VGG16
7  from keras.models import load_model
8  from keras import backend as K
9  from keras.preprocessing import image
10
11 import time
12 import numpy as np
13 from vgg16_cal256_setup import getSampleSetupInfo, getTrainSetupInfo, getPathSetupInfo, getFin
14 from vgg16_cal256_utils import saveTempPkl, loadTempPkl, getCal256Info, prepData, getData
15
16 def printpklhistory(historypkl):
17     keys, acc, loss, val_acc, val_loss = historypkl
18     summary_train = "last accuracy:%0.4f, last loss:%0.4f, best accuracy: %0.4f, best loss: %
19     summary_val = "last val accuracy:%0.4f, last val loss:%0.4f, best val_accuracy: %0.4f, bes
20
21     print summary_train
22     print summary_val
23     return summary_train, summary_val
```

```
24
25 def savehistory(history, historypklpath,
26                  accfigpath, lossfigpath,
27                  summarytxtpath, fig=True):
28     print 'Start saving history...'
29     acc = history.history['acc']
30     loss = history.history['loss']
31     val_acc = history.history['val_acc']
32     val_loss = history.history['val_loss']
33     pklhistory = history.history.keys(), acc, loss, val_acc, val_loss
34     saveTempPkl(pklhistory, historypklpath)
35     sum_train, sum_val = printpklhistory(pklhistory)
36     savesummary(sum_val, summarytxtpath)
37     if fig:
38         savefig([acc, val_acc],
39                 'model accuracy',
40                 'epoch', 'accuracy',
41                 ['train', 'validation'],
42                 accfigpath)
43         savefig([loss, val_loss],
44                 'model loss',
45                 'epoch',
46                 'loss',
47                 ['train', 'valication'],
48                 lossfigpath)
49
50     print 'Done saving history.'
51
52 def loadhistory(historypklpath):
53     if isfile(historypklpath):
54         print "Found previous historypkl. Loading %s."%(historypklpath)
55         pklhistory = loadTempPkl(historypklpath)
56         printpklhistory(pklhistory)
57         keys, acc, loss, val_acc, val_loss  = pklhistory
58     else:
59         print "Cannot find previous historypkl at path: %s"%(historypklpath)
60         keys = None
61         acc = None
62         loss = None
63         val_acc = None
64         val_loss = None
65     return acc, loss, val_acc, val_loss
66
67 def savesummary(summary, summarytxtpath):
68     if not isfile(summarytxtpath):
69         print "Saving %s"%(summarytxtpath)
70         with open(summarytxtpath, 'w') as f:
71             f.write(summary)
72
73 def savefig(results, title='', xlabel='', ylabel='', legends = [], savepath = '',
    Xs = [], display = False, overwrite = True):
74     if not isfile(savepath) or overwrite:
75         print "Save %s..."%(savepath)
76
77         if Xs == []:
78             Xs = range(len(results[0]))
79         print "# iterations:", len(Xs)
80         for Ys in results:
81             plt.plot(Xs, Ys)
82
83             plt.title(title)
84             plt.ylabel(ylabel)
85             plt.xlabel(xlabel)
86
87         if legends !=[]:
```

```
88              plt.legend(legends, loc='upper left')
89          plt.savefig(savepath)
90          print "Done saving acc figure."
91          if display:
92              plt.show()
93          plt.clf()
94
95  def genReportQ3():
96      # 3, 4.c
97      trainNs, _, _ = getSampleSetupInfo()
98
99      batchSize, maxEpoch, earlyStoppingMaxIcre, resultDir, modelname, datasetname = getTrainSet
100
101     final_val_accs = []
102     for trainN in trainNs:
103         _, historypklpath, testresultpklpath,_, _, _, _ = getPathSetupInfo(resultDir, modelnam
104         _, _, val_acc, _ = loadhistory(historypklpath)
105         if val_acc != None:
106             final_val_accs.append(val_acc[-1])
107     print final_val_accs
108     if final_val_accs != []:
109         title = 'model accuracy'
110         xlabel = 'Training images per-class'
111         ylabel = 'Accuracy'
112         legends = []
113         reportfig = getFinalReportFigInfo(resultDir, 'q3')
114         savefig([final_val_accs], title, xlabel, ylabel, legends, reportfig, trainNs,
    True)
115     else:
116         print "Could not find validation accuracies."
117
118 def genReportQ4ab():
119     # 4.a, 4.b
120     trainNs, _, _ = getSampleSetupInfo()
121     trainN = trainNs[-1]
122     batchSize, maxEpoch, earlyStoppingMaxIcre, resultDir, modelname, datasetname = getTrainSet
123     _, historypklpath, _, _, _, _, _ = getPathSetupInfo(resultDir, modelname, datasetname, bat
124
125     acc, loss, val_acc, val_loss = loadhistory(historypklpath)
126
127     if loss != []:
128         reportfig = getFinalReportFigInfo(resultDir,'q4_a')
129         savefig([loss, val_loss],
130                 'model loss',
131                 'Number of epoch',
132                 'Loss',
133                 ['Train', 'Test'],
134                 reportfig,
135                 range(len(loss)+1)[1:])
136         reportfig = getFinalReportFigInfo(resultDir, 'q4_b')
137         savefig([acc, val_acc],
138                 'model accuracy',
139                 'Number of epoch',
140                 'Accuracy',
141                 ['Train', 'Test'],
142                 reportfig,
143                 range(len(loss)+1)[1:])
144     else:
145         print "Could not find validation accuracies."
146
147 def normalize(x):
148     # utility function to normalize a tensor by its L2 norm
149     return x / (K.sqrt(K.mean(K.square(x))) + 1e-5)
150
151 def stitch(activations, n = 8):
```

```
152     img_width = activations.shape[0]
153     img_height = img_width
154     print '%dx%d'%(img_width, img_height)
155     # we will stich the best 64 filters on a 224 x 224 grid.
156     n = int(np.sqrt(activations.shape[2]))
157 
158     # the filters that have the highest loss are assumed to be better-looking.
159     # we will only keep the top 64 filters.
160     # activations = activations[:, :, 0:n * n]
161 
162     # build a black picture with enough space for
163     margin = 5
164     width = n * img_width + (n - 1) * margin
165     height = n * img_height + (n - 1) * margin
166     stitched_filters = np.zeros((width, height, 3))
167 
168     # fill the picture with our saved filters
169 
170     for i in range(n):
171         for j in range(n):
172             img = activations[..., i * n + j]
173             img = np.expand_dims(img, axis=2)
174             stitched_filters[(img_width + margin) * i: (img_width + margin) * i + img_width,
175                              (img_height + margin) * j: (img_height + margin) * j + img_height
176 
177     return stitched_filters
178 
179 def stitch4(activations, indices):
180     img_width = activations.shape[0]
181     img_height = img_width
182     print '%dx%d'%(img_width, img_height)
183     # we will stich the best 64 filters on a 224 x 224 grid.
184     margin = 5
185     n =2
186     width = n * img_width + (n - 1) * margin
187     height = n * img_height + (n - 1) * margin
188     stitched_filters = np.zeros((width, height, 3))
189 
190     # fill the picture with our saved filters
191 
192     count = 0
193     for i in range(n):
194         for j in range(n):
195             img = activations[..., indices[count]]
196             img = np.expand_dims(img, axis=2)
197             stitched_filters[(img_width + margin) * i: (img_width + margin) * i + img_width,
198                              (img_height + margin) * j: (img_height + margin) * j + img_height
199             count += 1
200     return stitched_filters
201 
202 def deprocessImage(x):
203     # normalize tensor: center on 0., ensure std is 0.1
204     x -= x.mean()
205     x /= (x.std() + 1e-5)
206     x *= 0.1
207 
208     # clip to [0, 1]
209     x += 0.5
210     x = np.clip(x, 0, 1)
211 
212     # convert to RGB array
213     x *= 255
214     if K.image_dim_ordering() == 'th':
215         x = x.transpose((1, 2, 0))
216     x = np.clip(x, 0, 255).astype('uint8')
```

```
217     return x
218
219 def visualizeLayer(layer_name, X, save=True):
220     # Setup model
221     model = VGG16( weights='imagenet', include_top=True )
222     model.summary()
223     input_img = model.input
224
225     # get the symbolic outputs of each "key" layer (we gave them unique names).
226     layer_dict = dict([(layer.name, layer) for layer in model.layers[1:]])
227
228     kept_filters = []
229     nFilters = layer_dict[layer_name].output_shape[-1]
230     # Get layer output
231     layer_output = layer_dict[layer_name].output
232     # create a function that returns the loss and grads given the input picture.
233     get_activation = K.function([input_img], [layer_output])
234     activations = get_activation([X])[0]
235     activations = np.squeeze(activations)
236
237     gridsize = 2
238     indices = np.random.randint(activations.shape[2], size=gridsize*gridsize)
239     stitched_filters = stitch4(activations, indices)
240     # save the result to disk
241     return 'stitched_filters_%dx%d.png' % (gridsize, gridsize), stitched_filters
242
243 def genReportQ4d():
244     # Get test info for 32 samples.
245     trainNs, valNs, testNs = getSampleSetupInfo()
246     batchSize, maxEpoch, earlyStoppingMaxIcre,resultDir, modelname, datasetname = getTrainSetu
247     classNames, classCounts = getCal256Info()
248
249     trainN = trainNs[-1]
250     valN = valNs[-1]
251     testN = testNs[-1]
252
253     _, _, _, partitionpklpath, _, _, _ = getPathSetupInfo(resultDir, modelname, datasetname, b
254     prepData(trainN, valN, testN, classCounts, partitionpklpath)
255
256     z_train, z_valid, z_test = prepData(trainN, valN, testN, classCounts, partitionpklpath)
257
258     # z_test contains the test data that were not used for training or validation.
259     X, yhot = getData(z_test[:5], classNames)
260     paths = getImgPathsInfo(z_test, classNames)
261     # Decide which layer to look at.
262     layer_name1 = 'block1_conv1'
263     layer_name2 = 'block5_conv3'
264     maxfilterpool = 200 # 512 available.
265
266     for i in range(3):
267         reportfigImg = getFinalReportFigInfo(resultDir, 'q4_d_img%d'%(i+1))
268         imgPath = paths[i]
269         img = image.load_img(imgPath, False, (224, 224))
270         x = image.img_to_array(img)
271         imsave(reportfigImg, x)
272         #imsave(reportfigImg, img)
273         reportfig1 = getFinalReportFigInfo(resultDir, 'q4_d_feature_conv1_%d'%(i+1))
274         reportfig2 = getFinalReportFigInfo(resultDir, 'q4_d_feature_conv3_%d'%(i+1))
275
276         filename, filters = visualizeLayer(layer_name1, np.expand_dims(X[i, ...], axis=0))
277         print "filters", type(filters), filters.shape
278         imsave(reportfig1, filters)
279         filename, filters = visualizeLayer(layer_name2, np.expand_dims(X[i, ...], axis=0))
280         imsave(reportfig2, filters)
281
```

```
282 def genReportQ5():
283     layer_names = getLayerNamesInfo()
284     for layer_name in layer_names:
285         trainN, _, _, batchSize, maxEpoch, earlyStoppingMaxIncre, resultDir, modelname, datase
286         _, historypklpath, _, _, _, _, _ = getPathSetupInfo(resultDir, modelname, datasetname,
287         print "try load %s"%(historypklpath)
288         acc, loss, val_acc, val_loss = loadhistory(historypklpath)
289         print type(acc)
290         if loss != []:
291             reportfig = getFinalReportFigInfo(resultDir, 'q5_a_%s'%(layer_name))
292             print "saving reports to %s"%(reportfig)
293             savefig([loss, val_loss],
294                     'model loss',
295                     'Number of epoch',
296                     'Loss',
297                     ['Train', 'Test'],
298                     reportfig,
299                     range(len(loss)+1)[1:])
300             reportfig = getFinalReportFigInfo(resultDir, 'q5_b_%s'%(layer_name))
301             print "saving reports to %s"%(reportfig)
302 
303             savefig([acc, val_acc],
304                     'model accuracy',
305                     'Number of epoch',
306                     'Accuracy',
307                     ['Train', 'Test'],
308                     reportfig,
309                     range(len(loss)+1)[1:])
310         else:
311             print "Could not find validation accuracies."
312 # Unit tests
313 if __name__ == "__main__":
314     #genReportQ3()
315     #genReportQ4ab()
316     genReportQ4d()
317     #genReportQ5()
```

Listing 6: vgg16_cal256_layer_analysis.py for generating data for q5

```
1 from vgg16_model import trainByLayer
2 
3 if __name__ == "__main__":
4     layer_names = ['block5_conv1', 'block5_conv2', 'block5_conv3']
5     lr = [ 0.0000001, 0.000001, 0.00001]
6     for trainpara in zip(layer_names, lr):
7         trainByLayer(trainpara[0], trainpara[1])
```

Listing 7: cal256.py for starter version for q3

```
1 import keras
2 from keras.preprocessing import image
3 from keras.applications.vgg16 import VGG16
4 from keras.applications.vgg16 import preprocess_input
5 from keras.models import Model
6 from keras.utils import np_utils
7 from keras.layers.core import Dense
8 import numpy as np
9 from random import shuffle
10 import os
11 import re
12 
13 
14 def getModel( output_dim ):
15     '''
16         * output_dim: the number of classes (int)
```

```
17
18          * return: compiled model (keras.engine.training.Model)
19      '''
20      vgg_model = VGG16( weights='imagenet', include_top=True )
21      vgg_out = vgg_model.layers[-2].output #Last FC layer's output
22      softmax_layer = Dense(256, activation='softmax')(vgg_out) #Create softmax layer taking inp
23      #Create new transfer learning model
24      tl_model = Model( input=vgg_model.input, output=softmax_layer )
25
26      #Freeze all layers of VGG16 and Compile the model
27      for l in tl_model.layers:
28          l.trainable = False
29      tl_model.layers[-1].trainable = True
30      tl_model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'
31      #Confirm the model is appropriate
32      tl_model.summary()
33      return tl_model
34
35  category = [''] * 256
36
37  def prepData(train_size=30, valid_size=3, test_size=15):
38      x_train = []
39      x_valid = []
40      x_test = []
41      y_train = []
42      y_valid = []
43      y_test = []
44      p = '256_ObjectCategories'
45      for c in os.listdir(p):
46          categ = int(c.split('.')[0]) - 1
47          if categ >= 256: break
48          category[categ] = c
49          x = []
50          for i in os.listdir(os.path.join(p, c)):
51              s = re.split('_|\.', i)
52              if len(s) != 3 or s[2] != 'jpg': continue
53              x.append(int(s[1]))
54          shuffle(x)
55          x_train += x[:train_size]
56          x_valid += x[train_size:train_size+valid_size]
57          x_test += x[train_size+valid_size:train_size+valid_size+test_size]
58          y_train += [categ] * train_size
59          y_valid += [categ] * valid_size
60          y_test += [categ] * test_size
61      z_train = list(zip(x_train, y_train))
62      shuffle(z_train)
63      z_valid = list(zip(x_valid, y_valid))
64      shuffle(z_valid)
65      z_test = list(zip(x_test, y_test))
66      shuffle(z_test)
67      return z_train, z_valid, z_test
68
69  def getData(z_data):
70      p = '256_ObjectCategories'
71      x_data = []
72      y_data = []
73      for z in z_data:
74          pa = os.path.join(p, category[z[1]])
75          pa = os.path.join(pa, str(z[1]+1).zfill(3)+'_'+str(z[0]).zfill(4)+'.jpg')
76          img = image.load_img(pa, target_size=(224,224))
77          x_data.append(image.img_to_array(img))
78          y_data.append(z[1])
79      x_data = np.array(x_data)
80      x_data = preprocess_input(x_data)
81      y_data = np_utils.to_categorical(y_data, 256)
```

```
82      return x_data, y_data
83
84  def genSample(z_train):
85      batch_size = 32
86      i = 0
87      while 1:
88          x_train, y_train = getData(z_train[i:i+batch_size])
89          i = i + batch_size
90          if i + batch_size > len(z_train):
91              shuffle(z_train)
92              i = 0
93          yield x_train, y_train
94
95
96  if __name__ == '__main__':
97      #Output dim for your dataset
98      output_dim = 256 #For Caltech256
99      tl_model = getModel( output_dim )
100
101     train_size = 30
102     z_train, z_valid, z_test = prepData(train_size)
103     x_valid, y_valid = getData(z_valid)
104     #Train the model
105     tl_model.fit_generator(genSample(z_train), samples_per_epoch=256*train_size, nb_epoch=10,
106     #Test the model
107     x_test, y_test = getData(z_test)
108     result = tl_model.evaluate(x_test, y_test)
109     print("On test set")
110     for (v,n) in zip (result, tl_model.metrics_names):
111         print(n + ": " + str(v))
```

Listing 8: Code for Urban Tribes

```
1   import keras
2   from keras.applications import VGG16
3   from keras.applications.vgg16 import preprocess_input
4   from keras.models import Model
5   from keras.layers.core import Activation, Dense, Flatten
6   from keras.layers.pooling import MaxPooling2D
7   from keras.preprocessing.image import ImageDataGenerator, array_to_img, \
8       img_to_array, load_img
9   from keras.utils import np_utils
10  from keras import backend as K
11
12  from os import listdir
13  from os.path import isfile, join
14
15  import numpy as np
16  import scipy
17  from scipy.misc import imsave
18
19  from collections import defaultdict
20  import matplotlib.pyplot as plt
21
22
23  datapath = '../data/pictures_all/'
24  #datapath = '../data/urban_tribe/'
25
26  np.random.seed(514)
27
28
29  def getModel(output_dim):
30      vgg_model = VGG16(weights='imagenet', include_top=True)
31      vgg_out = vgg_model.layers[-2].output #Last FC layer's output
32      softmax_layer_output = Dense(output_dim, activation='softmax', name='predictions') \
```

```
33      (vgg_out)
34
35      # create new transfer learning model
36      tl_model = Model(input=vgg_model.input, output=softmax_layer_output)
37
38      # freeze all layers of VGG16
39      for layer in vgg_model.layers[:-1]:
40          layer.trainable = False
41
42      # compile the model
43      tl_model.compile(loss='categorical_crossentropy',
44                optimizer='rmsprop',
45                metrics=['accuracy'])
46
47      # confirm the model is appropriate
48      print tl_model.summary()
49
50      return tl_model
51
52
53  def loadData():
54      # create label dictionary
55      label = ['biker', 'clubber', 'country', 'formal', 'goth', 'heavy', 'hiphop', 'hipster', \
56       'others', 'ravers', 'surfer']
57      map_label_idx = dict()
58      for l, idx in zip(label, range(len(label))):
59          map_label_idx[l] = idx
60
61      # load and categorize each file with its label
62      labelData = defaultdict(list)
63      files = [f for f in listdir(datapath) if isfile(join(datapath, f))]
64      for f in files:
65          if f == '.DS_Store':
66              continue
67          img = load_img(join(datapath, f), target_size=(224, 224))
68          img = img_to_array(img)
69          img = np.expand_dims(img, axis=0)
70          img = preprocess_input(img)
71          lb = map_label_idx[f.split('_')[0]]
72          labelData[lb].append(img[0])
73
74      # shuffle and split data
75      lstData = []
76      for lb, data in labelData.iteritems():
77          data = list(np.random.permutation(data))
78          lstData.append(data)
79      return lstData
80
81
82  def splitData(lstData, num_inst=2):
83      stn_X = []
84      stn_y = []
85      vld_X = []
86      vld_y = []
87      for lb in xrange(len(lstData)):
88          data = lstData[lb]
89          stn_X += data[:num_inst]
90          stn_y += [lb for i in xrange(num_inst)]
91          vld_X += data[-32:]
92          vld_y += [lb for i in xrange(32)]
93
94      stn_y = np_utils.to_categorical(stn_y, len(lstData))
95      vld_y = np_utils.to_categorical(vld_y, len(lstData))
96      stn_X, stn_y = np.array(stn_X), np.array(stn_y)
97      vld_X, vld_y = np.array(vld_X), np.array(vld_y)
```

```
98      return stn_X, stn_y, vld_X, vld_y
99
100
101 # util function to convert a tensor into a valid image
102 def deprocess_image(img):
103     # normalize tensor: center on 0., ensure std is 0.1
104     img -= img.mean()
105     img /= (img.std() + 1e-5)
106     img *= 0.1
107
108     # clip to [0, 1]
109     img += 0.5
110     img = np.clip(img, 0, 1)
111
112     # convert to RGB array
113     img *= 255
114     img = np.clip(img, 0, 255).astype('uint8')
115     return img
116
117
118 def visualize(img, layer_dict, layer_name, size):
119     filters = []
120
121     input_img = layer_dict['input_1'].input
122     layer_output = layer_dict[layer_name].output
123
124     for filter_idx in range(size):
125         # build a loss function that maximizes the activation
126         # of the nth filter of the layer considered
127         loss = K.mean(layer_output[:, :, :, filter_idx])
128
129         # compute the gradient of the input picture wrt this loss
130         grads = K.gradients(loss, input_img)[0]
131
132         # normalization trick: we normalize the gradient
133         grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)
134
135         # this function returns the loss and grads given the input picture
136         iterate = K.function([input_img], [loss, grads])
137
138         # step size for gradient ascent
139         step = 1.
140
141         # we start from a gray image with some noise
142         #input_img_data = np.random.random((1, 224, 224, 3))
143         #input_img_data = (input_img_data - 0.5) * 20 + 128
144         input_img_data = np.float32(np.expand_dims(img, axis=0))
145
146         # run gradient ascent for 20 steps
147         for i in range(20):
148             loss_value, grads_value = iterate([input_img_data])
149             input_img_data += grads_value * step
150
151             # some filters get stuck to 0, we can skip them
152             if loss_value <= 0.:
153                 break
154
155         if loss_value > 0:
156             # deprocess
157             img = deprocess_image(input_img_data[0])
158             filters.append((img, loss_value))
159
160     n = 4
161     # the filters that have the highest loss are assumed to be better-looking.
162     # we will only keep the top 64 filters.
```

```
163     filters.sort(key=lambda x: x[1], reverse=True)
164     filters = filters[:n * n]
165
166     margin = 5
167     width = n * 224 + (n - 1) * margin
168     height = n * 224 + (n - 1) * margin
169     stitched_filters = np.zeros((width, height, 3))
170
171     for i in range(n):
172         for j in range(n):
173             img, loss = filters[i * n + j]
174             stitched_filters[(224 + margin) * i: (224 + margin) * i + 224,
175                              (224 + margin) * j: (224 + margin) * j + 224, :] = img
176
177     imsave('../result/%s_filter_new_%dx%d.png' % (layer_name, n, n), stitched_filters)
178
179
180 if __name__ == '__main__':
181     # retrieve model
182     output_dim = 11
183     tl_model = getModel(output_dim)
184     print 'finish loading model'
185
186     # load data
187     lstData = loadData()
188     print 'finish loading data'
189
190
191     ## 4. Inference (a), (b), (c)
192     num_inst_lst = [2, 4, 8, 16, 32]
193     stn_acc_lst = []
194     vld_acc_lst = []
195     for num_inst in num_inst_lst:
196         print('\nNow is using ' + str(num_inst) + ' instances')
197
198         # split dataset
199         stn_X, stn_y, vld_X, vld_y = splitData(lstData, num_inst)
200
201         # preprocessing
202         datagen = ImageDataGenerator()
203         datagen.fit(stn_X)
204
205         # train model
206         earlyStopping=keras.callbacks.EarlyStopping(monitor='val_loss', patience=3, \
207             verbose=0, mode='auto')
208         hist = tl_model.fit_generator(datagen.flow(stn_X, stn_y, batch_size=11), \
209                 samples_per_epoch=stn_X.shape[0], \
210                 nb_epoch=100, \
211                 callbacks=[earlyStopping], \
212                 validation_data=datagen.flow(vld_X, vld_y, batch_size=11), \
213                 nb_val_samples=vld_X.shape[0])
214
215
216         # loss history
217         stn_loss = hist.history['loss']
218         vld_loss = hist.history['val_loss']
219         print stn_loss, vld_loss
220
221         plt.figure()
222         plt.plot(stn_loss, label='subtrain')
223         plt.plot(vld_loss, label='validate')
224         plt.xlabel('Iterations', labelpad=10)
225         plt.ylabel('Loss', labelpad=10)
226         plt.margins(0)
227         plt.title('Loss for 11-way Classification through Iterations with ' + \
```

```
228             str(num_inst) + ' Instances')
229         plt.legend(loc=1)
230         plt.subplots_adjust(bottom=0.15)
231         plt.savefig('../result/loss_' + str(num_inst) + '.png')
232 
233         # accuracy history
234         stn_acc = hist.history['acc']
235         stn_acc_lst.append(stn_acc[-1])
236         vld_acc = hist.history['val_acc']
237         vld_acc_lst.append(vld_acc[-1])
238         print stn_acc, vld_acc
239 
240         plt.figure()
241         plt.plot(stn_acc, label='subtrain')
242         plt.plot(vld_acc, label='validate')
243         plt.xlabel('Iterations', labelpad=10)
244         plt.ylabel('Accuracy', labelpad=10)
245         plt.ylim(0, 1)
246         plt.margins(0)
247         plt.title('Accuracy for 11-way Classification through Iterations with ' + \
248             str(num_inst) + ' Instances')
249         plt.legend(loc=1)
250         plt.subplots_adjust(bottom=0.15)
251         plt.savefig('../result/acc_' + str(num_inst) + '.png')
252 
253     # accuracy history
254     plt.figure()
255     plt.plot([0] + num_inst_lst, [0] + stn_acc_lst, label='subtrain')
256     plt.plot([0] + num_inst_lst, [0] + vld_acc_lst, label='validate')
257     plt.xlabel('Number of Instances per Class', labelpad=10)
258     plt.ylabel('Accuracy', labelpad=10)
259     plt.ylim(0, 1)
260     plt.margins(0)
261     plt.title('Accuracy for 11-way Classification with Increasing Instances')
262     plt.legend(loc=1)
263     plt.subplots_adjust(bottom=0.15)
264     plt.savefig('../result/acc_instances.png')
265 
266 
267     ## 4. Inference (e)
268     layer_dict = dict([(layer.name, layer) for layer in tl_model.layers])
269 
270     # split dataset
271     #stn_X, stn_y, vld_X, vld_y = splitData(lstData, 1)
272 
273     # visualize filters
274     visualize(vld_X[0], layer_dict, 'block1_conv1', 64) # 64 filters
275     visualize(vld_X[0], layer_dict, 'block5_conv3', 512) # 512 filters
276 
277     # visualize output
278     vgg_model = VGG16(weights='imagenet', include_top=True)
279         margin = 5
280         width1 = 4 * 224 + 3 * margin
281         height1 = 4 * 224 + 3 * margin
282         width2 = 4 * 14 + 3 * margin
283         height2 = 4 * 14 + 3 * margin
284         block1_conv1_filter = np.ones((height1, width1)) * 255
285         block5_conv3_filter = np.ones((height2, width2)) * 255
286         for m in range(4):
287                 img = load_img("../data/urban_tribe/biker_group_pic0000%d.jpg" % \
288             (m + 1), target_size=(224, 224))
289                 img = img_to_array(img)
290                 img = np.expand_dims(img, axis=0)
291                 img = preprocess_input(img)
292                 for i in range(2):
```

```
293                     if i == 0:
294                         layer_name = 'block1_conv1'
295                         side = 224
296                     else:
297                         layer_name = 'block5_conv3'
298                         side = 14
299                     intermediate_layer_model = Model(input=vgg_model.input,
300                                                      output=vgg_model.get_layer(layer_name).output)
301                     intermediate_output = intermediate_layer_model.predict(img)[0]
302                     for j in range(4):
303                         if i == 0:
304                                 block1_conv1_filter[(side + margin) * m: \
305                     (side + margin) * m + side,
306                                                     (side + margin) * j: (side + margin) * j + side] \
307                                   = intermediate_output[:,:,j]
308                         else:
309                                 block5_conv3_filter[(side + margin) * m: \
310                     (side + margin) * m + side,
311                                                     (side + margin) * j: (side + margin) * j + side] \
312                                   = intermediate_output[:,:,j]
313         imsave('../result/block1_conv1_filter.png', block1_conv1_filter)
314         imsave('../result/block5_conv3_filter.png', block5_conv3_filter)
315
316
317     ## 5. Feature Extraction (a)
318     '''
319     layer_names = ['block1_conv1', 'block1_conv2',
320                    'block2_conv1', 'block2_conv2',
321                    'block3_conv1', 'block3_conv2', 'block3_conv3',
322                    'block4_conv1', 'block4_conv2', 'block4_conv3',
323                    'block5_conv1', 'block5_conv2', 'block5_conv3']
324     '''
325     layer_names = ['block5_conv1', 'block5_conv2', 'block5_conv3']
326
327     stn_acc_lst = []
328     vld_acc_lst = []
329
330     # split dataset
331     stn_X, stn_y, vld_X, vld_y = splitData(lstData, 32)
332
333     for layer_name in layer_names:
334         # create new transfer learning model
335         layer_output = layer_dict[layer_name].output
336         pooling_output = MaxPooling2D(pool_size=(2, 2), strides=None, \
337             border_mode='valid')(layer_output)
338         flatten_layer_output = Flatten()(pooling_output)
339         softmax_layer_output = Dense(output_dim, activation='softmax', name='predictions') \
340             (flatten_layer_output)
341         intermediate_layer_model = Model(input=tl_model.input, output=softmax_layer_output)
342
343         # define learning rate
344         rmsprop = keras.optimizers.RMSprop(lr=0.00001, rho=0.9, epsilon=1e-08, decay=0.0)
345
346         # compile the model
347         intermediate_layer_model.compile(loss='categorical_crossentropy',
348                 optimizer=rmsprop,
349                 metrics=['accuracy'])
350
351
352         # confirm the model is appropriate
353         print intermediate_layer_model.summary()
354
355         print('\nNow is working on ' + layer_name)
356
357         # preprocessing
```

```
358             datagen = ImageDataGenerator()
359             datagen.fit(stn_X)
360
361             # train model
362             earlyStopping=keras.callbacks.EarlyStopping(monitor='val_loss', patience=6, \
363                 verbose=0, mode='auto')
364             hist = intermediate_layer_model.fit_generator(datagen.flow(stn_X, stn_y, \
365                     batch_size=11), \
366                     samples_per_epoch=stn_X.shape[0], \
367                     nb_epoch=100, \
368                     callbacks=[earlyStopping], \
369                     validation_data=datagen.flow(vld_X, vld_y, batch_size=11), \
370                     nb_val_samples=vld_X.shape[0])
371
372             stn_acc = hist.history['acc']
373             stn_acc_lst.append(stn_acc[-1])
374             vld_acc = hist.history['val_acc']
375             vld_acc_lst.append(vld_acc[-1])
376
377
378         # accuracy history
379         plt.figure()
380         plt.plot(range(len(layer_names)), stn_acc_lst, label='subtrain')
381         plt.plot(range(len(layer_names)), vld_acc_lst, label='validate')
382         plt.xticks(range(len(layer_names)), layer_names, rotation=70)
383         plt.xlabel('Concatenate Different Convolutional Layers', labelpad=10)
384         plt.ylabel('Accuracy', labelpad=10)
385         plt.ylim(0, 1)
386         plt.title('Accuracy for 11-way Classification with Different Layers')
387         plt.legend(loc=1)
388         plt.subplots_adjust(bottom=0.3)
389         plt.margins(0)
390         plt.savefig('../result/acc_layers.png')
```

Codes for Bonus Question

Listing 9: bonus.py

```
1  import keras
2  from keras.preprocessing import image
3  from keras.applications.vgg16 import VGG16
4  from keras.applications.vgg16 import preprocess_input
5  from keras.models import Model
6  from keras.utils import np_utils
7  from keras.layers.core import Dense
8  from keras.layers.core import Activation
9  from keras import backend as K
10 from keras.callbacks import EarlyStopping
11 import numpy as np
12 from random import shuffle
13 import os
14 import re
15 import gc
16
17 def temperature_activation(a):
18     T = 8
19     return K.softmax(a/T)
20
21 def getModel( output_dim ):
22     '''
23         * output_dim: the number of classes (int)
24
25         * return: compiled model (keras.engine.training.Model)
26     '''
27     vgg_model = VGG16( weights='imagenet', include_top=True )
```

```
28      vgg_out = vgg_model.layers[-2].output #Last FC layer's output
29      fc_layer = Dense(1000, activation=temperature_activation, weights=vgg_model.layers[-1].get
30      softmax_layer = Dense(256, activation='softmax')(fc_layer) #Create softmax layer taking in
31      #Create new transfer learning model
32      tl_model = Model( input=vgg_model.input, output=softmax_layer )
33
34      #Freeze all layers of VGG16 and Compile the model
35      for l in tl_model.layers:
36          l.trainable = False
37      tl_model.layers[-1].trainable = True
38      tl_model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'
39      #Confirm the model is appropriate
40      tl_model.summary()
41      return tl_model
42
43  category = [''] * 256
44
45  def prepData(train_size=30, valid_size=3, test_size=10):
46      x_train = []
47      x_valid = []
48      x_test = []
49      y_train = []
50      y_valid = []
51      y_test = []
52      p = '256_ObjectCategories'
53      for c in os.listdir(p):
54          categ = int(c.split('.')[0]) - 1
55          if categ >= 256: break
56          category[categ] = c
57          x = []
58          for i in os.listdir(os.path.join(p, c)):
59              s = re.split('_|\.', i)
60              if len(s) != 3 or s[2] != 'jpg': continue
61              x.append(int(s[1]))
62          shuffle(x)
63          x_train += x[:train_size]
64          x_valid += x[train_size:train_size+valid_size]
65          x_test += x[train_size+valid_size:train_size+valid_size+test_size]
66          y_train += [categ] * train_size
67          y_valid += [categ] * valid_size
68          y_test += [categ] * test_size
69      z_train = list(zip(x_train, y_train))
70      shuffle(z_train)
71      z_valid = list(zip(x_valid, y_valid))
72      shuffle(z_valid)
73      z_test = list(zip(x_test, y_test))
74      shuffle(z_test)
75      return z_train, z_valid, z_test
76
77  def getData(z_data):
78      p = '256_ObjectCategories'
79      x_data = []
80      y_data = []
81      for z in z_data:
82          pa = os.path.join(p, category[z[1]])
83          pa = os.path.join(pa, str(z[1]+1).zfill(3)+'_'+str(z[0]).zfill(4)+'.jpg')
84          img = image.load_img(pa, target_size=(224,224))
85          x_data.append(image.img_to_array(img))
86          y_data.append(z[1])
87      x_data = np.array(x_data)
88      x_data = preprocess_input(x_data)
89      y_data = np_utils.to_categorical(y_data, 256)
90      return x_data, y_data
91
92  def genSample(z_train):
```

```
    batch_size = 32
    i = 0
    while 1:
        x_train, y_train = getData(z_train[i:i+batch_size])
        i = i + batch_size
        if i + batch_size > len(z_train):
            shuffle(z_train)
            i = 0
        yield x_train, y_train


if __name__ == '__main__':
    #Output dim for your dataset
    output_dim = 256 #For Caltech256
    tl_model = getModel( output_dim )

    train_size = 10
    z_train, z_valid, z_test = prepData(train_size)
    x_valid, y_valid = getData(z_valid)
    #Train the model
    early_stopping = EarlyStopping(monitor='val_loss', min_delta=0.001, patience=0)
    tl_model.fit_generator(genSample(z_train), samples_per_epoch=256*train_size, nb_epoch=15,
    #Test the model
    gc.collect()
    x_test, y_test = getData(z_test)
    result = tl_model.evaluate(x_test, y_test)
    print("On test set")
    for (v,n) in zip (result, tl_model.metrics_names):
        print(n + ": " + str(v))
```

References

[1] VGG16 model *https://gist.github.com/baraldilorenzo/07d7802847aaad0a35d3*

[2] Griffin, G., Holub, A. & Perona, P. (2007) *Caltech-256 Object Category Dataset.* California Institute of Technology . (Unpublished) http://resolver.caltech.edu/CaltechAUTHORS:CNS-TR-2007-001

[3] Urban Tribes dataset *https://www.dropbox.com/s/u3pozl1fo9ahxrq/urban_tribe.zip?dl=0*

[4] Keras software, online resource *https://keras.io/applications/*

[5] Zeiler, M.D.& Fergus, R. (2013) *Visualizing and Understanding Convolutional Networks*, https://arxiv.org/pdf/1311.2901v3.pdf