# Image Classification using Transfer Learning

**Shradha Agrawal**
A53105993
sha015@eng.ucsd.edu

**Sudhanshu Bahety**
A53209213
sbahety@eng.ucsd.edu

**Jean Choi**
A53214201
jsc078@eng.ucsd.edu

**Ajitesh Gupta**
A53220177
ajgupta@eng.ucsd.edu

**Nimish Srivastava**
A53222968
n2srivas@eng.ucsd.edu

## Abstract

Transfer Learning aims at solving new tasks using knowledge gained from solving related tasks. This project deals with applying variants of Transfer Learning for the task of Image Classification. Deep architectures of Convolutional Neural Networks, with exploding number of parameters, cannot be trained on datasets of small size. It is common practice to use existing models, trained on very large scale datasets, as feature extractors or initial layers of a ConvNet that can be fine-tuned to learn the underlying model of the dataset of new tasks. In this project, we use pretrained VGG16 network and learn the corresponding softmax layer using target dataset. This network is then used for classification task on target Caltech256 and Urban Tribes dataset. We achieve a final accuracy of $78.71\%$ and $57.95\%$ on Caltech256 and Urban Tribes dataset respectively.

## 1 Description of Data

### Caltech256

Dataset consists of 256 object categories containing a total of 30,607 images with minimum 80 images per category. The original Caltech-101 was collected by choosing a set of object categories, downloading examples from Google Images and then manually screening out all images that did not fit the category. Caltech-256 is collected in a similar manner.

### Urban Tribe

Dataset consists of 11 *Urban Tribe* categories containing a total of 1,159 images. Images are collected from web searches on public websites of social venues and image searches with keywords such as "group picture" or "party picture". The dataset includes images from various social groups such as "bikers", "formal group", and etc.

## Notations

The notations defined below are used in the rest of the report.

- $y_k^n$ denotes the probability that model assigns to $n$th image as being part of the $k$th class.
- $a_i$ is the weighted sum of the inputs to unit $i$.

## Caltech256 ConvNet

### Vanilla Model Description

The base model used was VGG16 (with pre-trained weights) with the last softmax layer (containing 1,000 units) replaced by our softmax layer (containing 256 units). Only the last layer of the new model was learned taking Caltech256 dataset. Softmax Cross Entropy Loss function was taken as the energy function.

### Training

Training was done taking [2, 4, 8, 16, 24, 32] images per class using SGD optimizer with momentum=0.9 (Nesterov), decay=$10^{-6}$, $\eta = 10^{-4}$, batch size = 8 for all the experiments. Number of test samples per class were fixed to 8 in test dataset.

### Results

- In Table 1, the training loss and test loss decrease as the number of training samples increase. This is visualized in Figure 1 for different per class sample sizes taken for training. We can observe that using more samples per class decreases the loss. This is expected because providing more data helps the model not over-fit and generalize well.
- In Figure 2, the train accuracy and test accuracy increase as the number of training samples increase. In each of the experiments, train accuracy and test accuracy generally increase and then become constant as epochs pass, while train accuracy is higher than test accuracy. The explanation of increase in this accuracy is again due to more per class samples taken for training as explained above.
- In Figure 3, we provide these results in a consolidated graph, which represents accuracy of our model vs. number of per class training samples.

The first and last filter activations were visualized for Figure 4 and the outputs are displayed in Figure 5 and Figure 6.

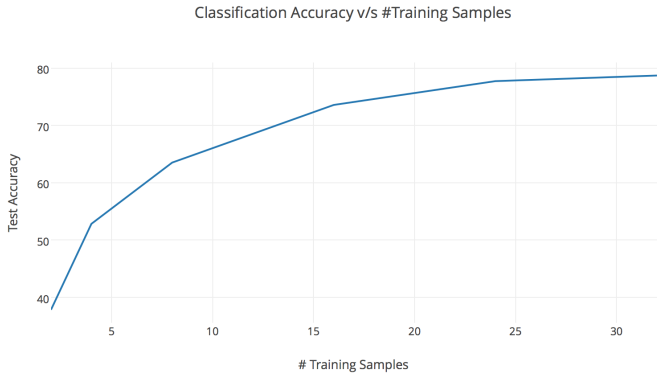


Figure 3: The accuracy increases with as the number of training examples per category increases. As number of examples increases, the model is able to learn better from unseen examples and thus able to generalize well for testing.

### Feature Extraction

Table 2 shows the results for using convolutional layers directly for classification without fully connected layers in between. In order to keep training parameters low, we connected the softmax to the pooling layer rather than the convolution layer itself. We can see that the classification performs badly. This can be attributed to two things - firstly all layers except the last layer do not have good global level features which is necessary for classification tasks especially with so many classes.

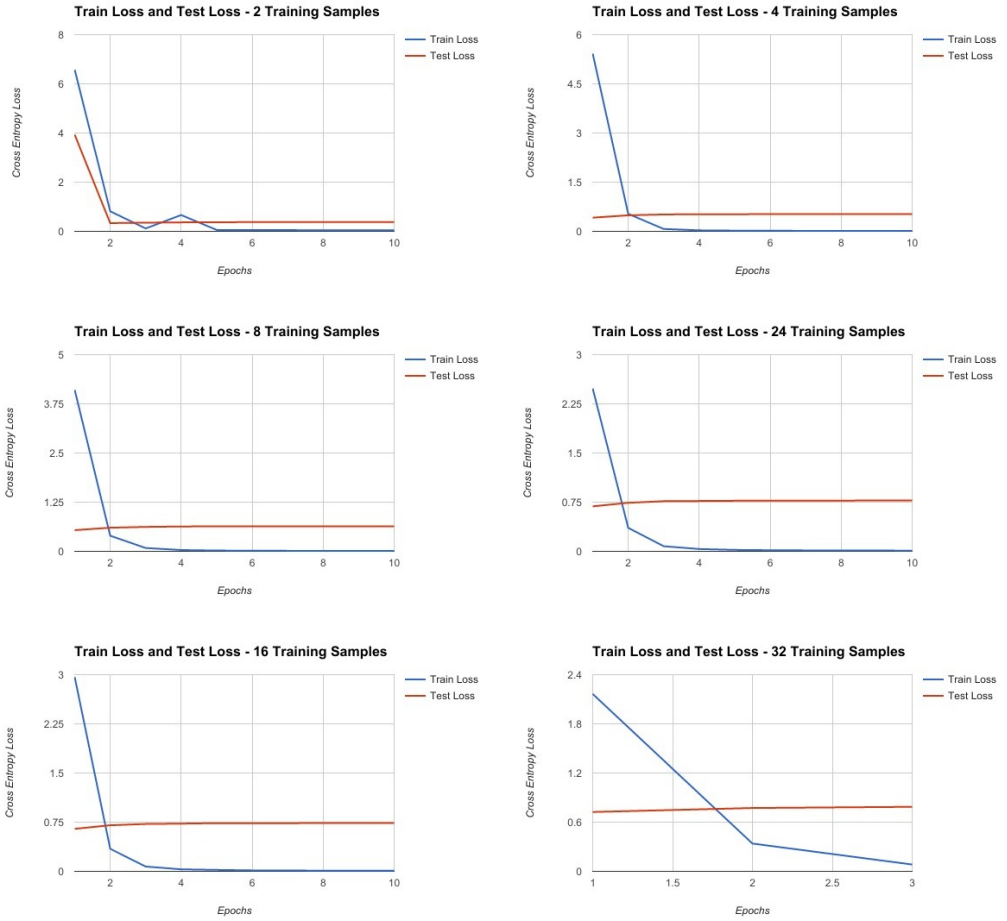


Figure 1: Plots show how the loss changes with number of epochs for different experiment settings on Caltech256 dataset. The best results can be seen for 32 examples/categories in the training set.

Table 1: Performance of Caltech256 ConvNet with different number of training examples per category. As the number of training examples per category increases, the accuracy on test set also increases. Number of samples for testing = 8, SGD for numerical optimization, Nestrov Momentum with momentum = 0.9, decay = $10^{-6}$ and batch size = 8 was used for all experiments. We were able to run only 3 epochs for 32 training samples per class due to resource constraints.

| # Samples per Category | #Epochs | Train Acc(%) | Test Acc(%) |
|---|---|---|---|
| 2 | 10 | 99.99 | 37.84 |
| 4 | 10 | 99.99 | 52.83 |
| 8 | 10 | 99.99 | 63.52 |
| 16 | 10 | 99.99 | 73.58 |
| 24 | 10 | 99.98 | 77.73 |
| 32 | 3 | 95.14 | 78.71 |

Secondly the fully connected layers are the layers which provide non-linear discriminatory powers to the network. They are akin to the hidden layer we see in normal neural networks. As such removing that makes this network like a plain softmax classifier which is being fed features extracted using the convolutional layers, which is far too weak for such a complex task. The last convolutional layer is still able to perform comparatively well due to the fact that it extracts some global level features. Between block1 and block4, block4 performs better. This is be because block4 has comparatively better global features. The GoogLeNet inception network is able to perform well without a fully

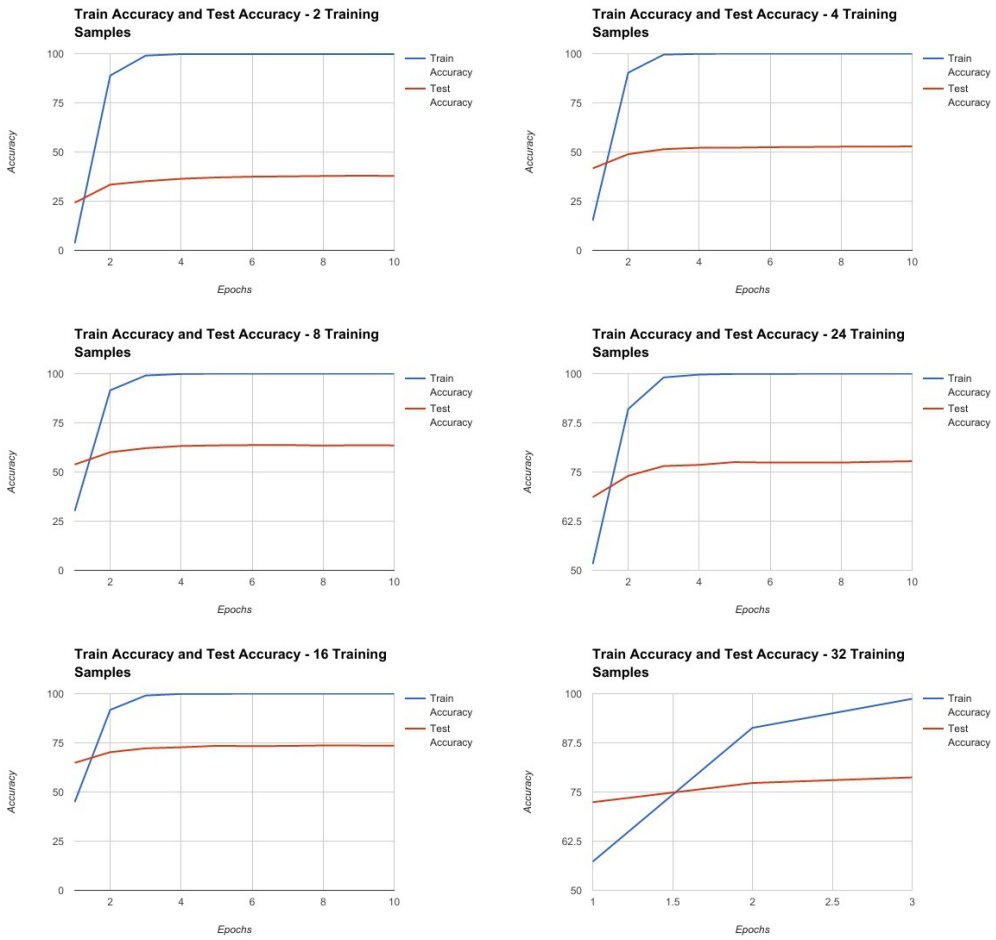


Figure 2: Plots show how the accuracy on the test set (8 samples per class) changes with number of epochs for different experiment settings on Caltech256 dataset. The best results can be seen for 32 examples per class in the training dataset.

connected layer because towards the end the filter size(5x5) becomes almost equal to the size of input(7x7) coming from previous layer. This gives the last layer filters a fully global receptive field. Besides they do multiple sized convolutions in the same layer, which gives them a variety of features. In our case the last layer filters are 3x3 whereas the input coming to them is 14x14, thus the filters have only got a local receptive field.

### Visualization

We have visualized the activations of the first and the last convolutional layers of the network by doing a forward pass of an image and monitoring the activations.

Figure 5 shows the activation at the first layer of the image in Figure 4. We can see that the filters are trying to differentiate on the basis of edges and textures. Images (1,1) and (3,3) detects the background texture. Images (3,4), (2,4), (1,2) show activation based on foreground texture detectection. Images (2,1) and (1,3) show edge detection taking place in the image. Images (1,4) and (4,4) seem to be results of texture detection and segmentation based on clothing and hair. Images (2,3), (3,3), (3,2) and (4,3) show higher activation for body/skin as compared to all other parts of the image.

Figure 6 shows the activation by the various filters in the last layer. Image (1,4) seems to be detecting the ear of buddha. Image (3,4) seems to be detecting the halo behind his head, while image (3,4) seems to be detecting the arm. Image (1,2) seems to be detecting a pair of eyes.

Figure 4: We used image of Buddha for visualizing first and last layer image outputs from Caltech256 dataset.

Figure 5: The above images shows output of 16 different filters of the first layer of VGG16 model corresponding to Figure 4. It can be seen that the model is able to learn different features like object boundary{(1,1)}, color gradient{(1,4), (4,4)}, edges{(1,3)}, contrast{(2,3), (3,1), (4,3)}, sharpness{(4,2)}, shades{(3,3)}, brightness{(2,4), (3,4)}. {.} denotes the set of tuples in the above image matrix. For example, (1,3) indicates 3rd image in the 1st row.

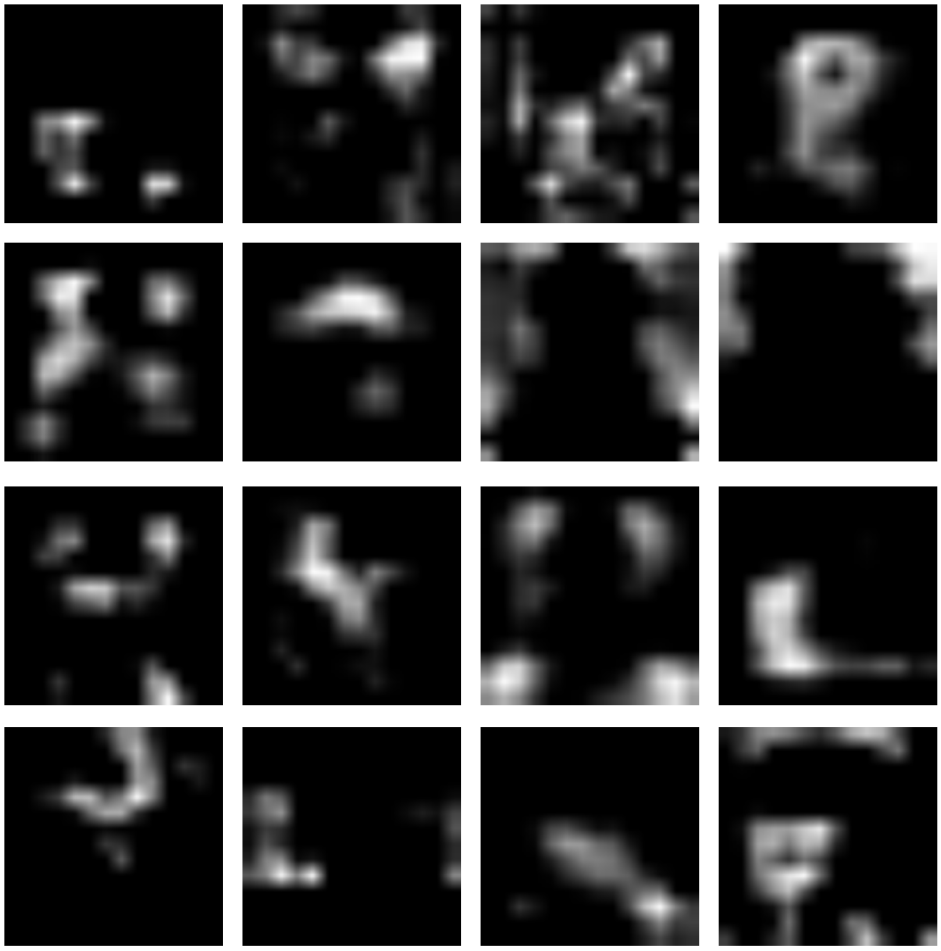

Figure 6: The above images shows output of 16 different filters of the last convolution layer of VGG16 model corresponding to Figure 4. It can be seen that as we dwell deeper in the model, it is able to identify different shapes present in the input images. For example, above we can see that {(2,3)} are similar to body outline of Buddha, {(1,2)} can be related to a pair of eyes.

Table 2: Performance of Caltech256 ConvNet with softmax output attached to different parts of the network. Number of samples per class = 8 training & 2 test, # Epochs = 10, RMSprop for numerical optimization, $\rho = 0.9$, $\eta = 0.001$, decay = $0.01$ and batch size = 8 was used for all experiments.

| **Layer before Softmax Output** | **Train Acc(%)** | **Test Acc(%)** |
|---|---|---|
| block5_pool | 80.42 | 28.97 |
| block4_pool | 0.69 | 0.98 |
| block1_pool | 0.59 | 0.39 |

For Caltech256, we studied the accuracies at intermediate layers: block5_pool, block4_pool and block3_pool. Running the experiments on layers below these gave memory errors on 2GB Nvidia GeForce 940mx GPU, and without the GPU it takes unnecessarily long time to converge. The results are displayed in Table 2

## Urban Tribe ConvNet

### Vanilla Model Description

The model used for urban tribe was same as Caltech256 ConvNet, the only difference was in the softmax layer in the output. In case of urban tribe, we used 11 units for our softmax output layer as there were 11 different classes.

### Training

The model was trained in the same way as Caltech256 ConvNet.

### Results

- In Table 3, the train accuracy and test accuracy increase as the number of training samples increase. This is visualized in Figure 8.

- In Figure 7, the train loss and test loss decrease as the number of training samples increase. This is because the more training samples we feed in, the more trained the network is. In each of the experiments, train loss generally decrease as epochs pass, while the test loss tend to be constant. As epochs pass, the network gets used to the training samples so the loss decrease.

- In Figure 8, the train accuracy and test accuracy increase as the number of training samples increase. In each of the experiments, train accuracy and test accuracy generally increase and then become constant as epochs pass, while train accuracy is higher than test accuracy. This is because the network gets used to the training samples as epochs pass.

- To see the general trend of test accuracy with respect to the number of training samples can be seen in Figure 9. As the number of training samples increases, the test accuracy increases rapidly with the input of unseen samples at first, but becomes constant as it starts to generalize.

The first and last filter activations were visualized for Figure 10 and the outputs are displayed in Figure 11 and figure 12.

Table 3: Performance of Urban Tribe ConvNet with different number of training examples per category. As the number of training examples per category increases, the accuracy on test set also increases. Number of samples for testing = 16, SGD for numerical optimization, Nestrov Momentum with momentum = 0.9, decay = $10^{-6}$ and batch size = 8 was used for all experiments.

| # Samples per Category | #Epochs | Train Acc(%) | Test Acc(%) |
|---|---|---|---|
| 4 | 20 | 99.99 | 34.09 |
| 8 | 10 | 99.99 | 43.75 |
| 16 | 10 | 99.99 | 46.59 |
| 32 | 10 | 99.99 | 57.95 |

### Feature extraction

Table 4 shows the results of removing fully connected layers and directly connecting various intermediate convolutional layers. Again to keep training parameters low we connect with the pooling layer rather than the convolutional layer itself. As with the Caltech experiments here also we see that the accuracies are not that good. The last layer again performs best with performance decreasing as we go towards the earlier convolutional layers. Again this can be attributed to the better global features towards the end of the network and the fact that it is essentially a softmax classifier being fed features from the convolutional layers.

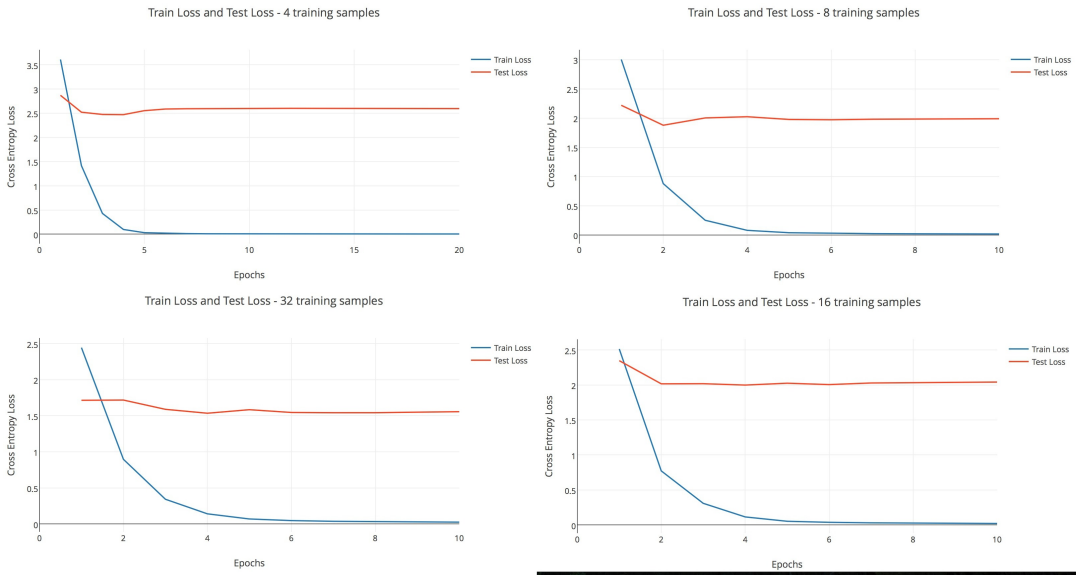


Figure 7: Plots show how the loss changes with number of epochs for different experiment settings on Urban Tribe dataset. The best results can be seen for 32 examples in the training.

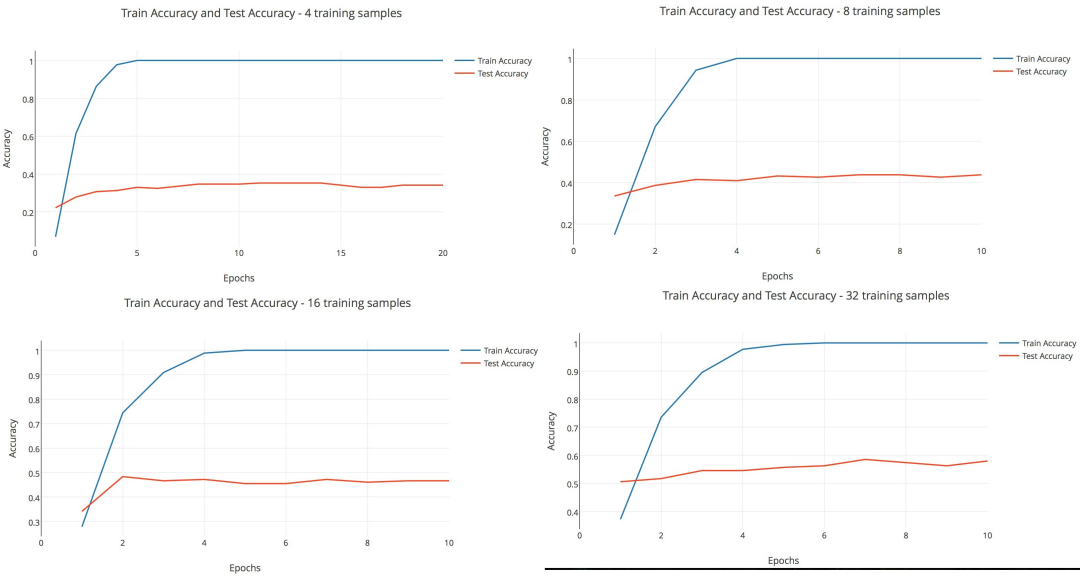


Figure 8: Plots show how accuracy on the test set (16 examples per class) changes with number of epochs for different experiment settings on Urban Tribe. The best results can be seen for 32 examples per class in the training.

## Visualization

Figure 10 shows the image we used to visualize the activations in the urban tribes trained CNN. It is a group of people belonging to the 'Formal' class.

Figure 11 shows the activation at the first convolutional layer. Images (1,1), (2,4), (3,4) show results belonging to an edge detecting filter. Image (4,3) and (1,3) shows good activation for areas belonging to clothes being done by a texture based filter. Images (1,2) and (2,3) show good activation for background.

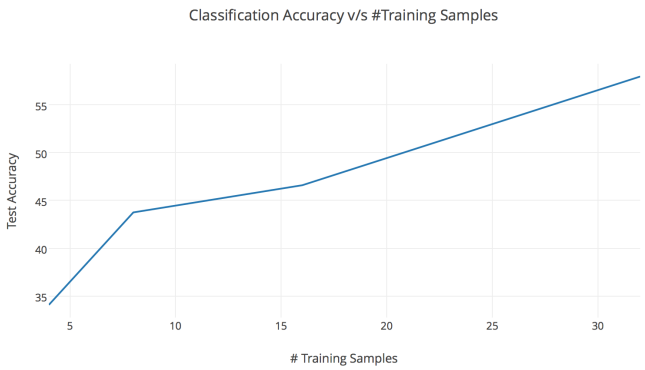


Figure 9: The accuracy increases with as the number of training examples per category increases, similar to Caltech256 dataset. As number of examples increases, the model is able to learn better from unseen examples and thus able to generalize well for testing.

Figure 12 shows the activation at the last convolutional layer. Although owing to the low resolution, its not really clear by human inspection as to what is being exactly identified by the convolution. But we believe that it is indeed identifying global features that are helping in performing the classification.

Figure 10: We used image of "formal" category for visualing first and last layer image outputs from Urban Tribe dataset.

Table 4: Performance of Urban Tribe ConvNet with softmax output attached to different parts of the network. Number of samples per class = 16 training & 4 tests, # Epochs = 10, RMSProp for numerical optimization, rho = 0.9 and batch size = 8 was used for all experiments.

| **Layer before Softmax Output** | **Test Acc(%)** | **Train Acc(%)** |
|---|---|---|
| block5_pool | 25.95 | 27.78 |
| block4_pool | 11.39 | 38.89 |
| block1_pool | 10.76 | 16.67 |

Figure 11: The above images shows output of 16 different filters of the first layer of VGG16 model corresponding to Figure 10. It can be seen that the model is able to learn different features like edges{(1,1), (3,1)}, contrast{(4,4)}, color of skin(all in black){(4,3)}, brightness{(4,1)}. {.} denotes the set of tuples in the above image matrix indicating the results.

For Urban Tribes, we studied the accuracies at intermediate layers: block5_pool, block4_pool and block1_pool. The results are displayed in Table 4

## Temperature based softmax regression

### Model Description

We used VGG16 as our base model and experimented with the Softmax Layer of it. The softmax layer of VGG16 was used as an input to our Softmax Layer to predict the classes of Caltech256. The idea was to use a "temperature parameter" on the softmax layer of VGG Model to reveal the similarities between the CalTech 256 objects. The temperature is inserted into the calculation of the last layer of VGG16 as shown in equation 1. $y_i$ now acts as the input to the Caltech256 Softmax layer.

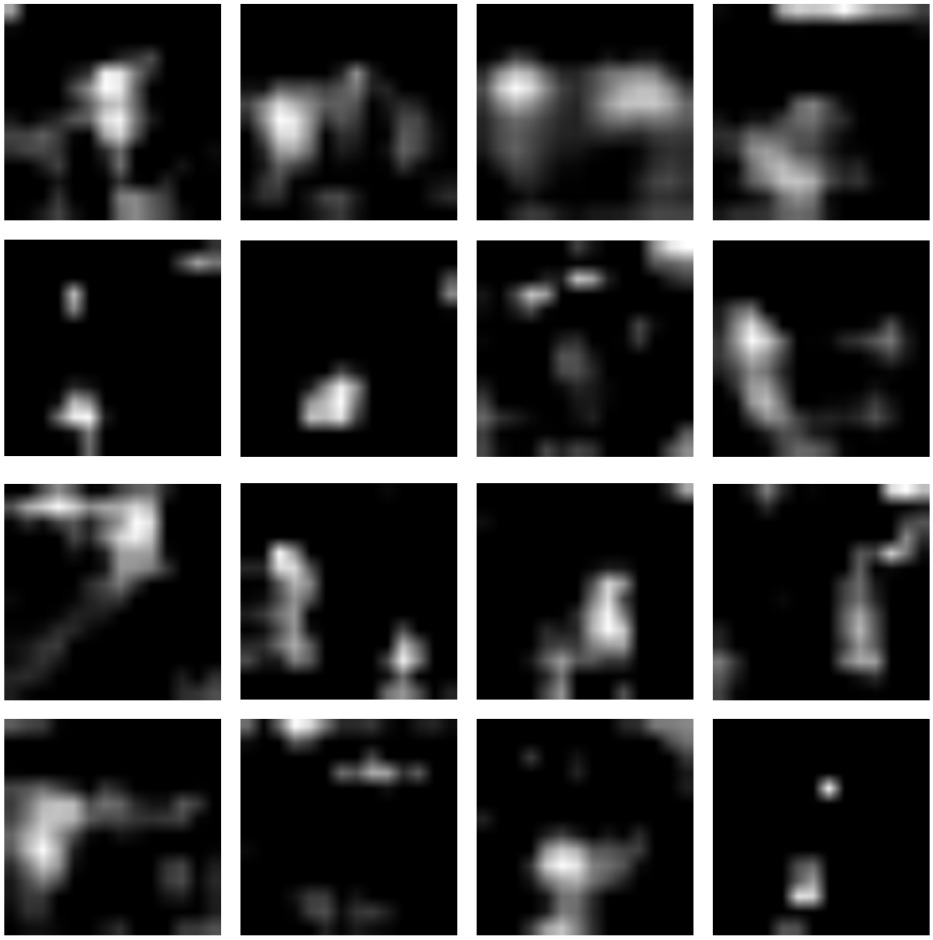

Figure 12: The above images shows output of 16 different filters of the last convolution layer of VGG16 model corresponding to Figure 10. It tries to learn more complex object shapes and other features present in the image.

**Result on Caltech256**

We experimented with different values of $T$ = [1, 2,4,8,16], and found $T = 1$ to give the best result with training accuracy of 71.00% and test accuracy of 56.25%. We used 8 training samples and 4 test samples per class. RMSProp was used as numerical optimization with $\eta = 0.01, \rho = 0.9, \epsilon = 10^{-8}$. We used 10 epochs for training and batch size of 8. Results are displayed in Table 5.
We observe that for larger values of T, specifically those more than 2, the model accuracy falls down drastically. This might be because the softmax output after annealing with temperature becomes too soft to generate a differential among the classes.

The idea of "temperature parameter" is to utilize the results of the last softmax output layer to generate "soft targets." If a model consists of many simple objects, then it will result in many "soft targets." For example, if there is a complex image which consists of people, bikes, wheels, and trees, these objects would all be potential "soft targets." The process of raising the "temperature" in the last softmax layer until the complex model results in a set of nice "soft targets", is called "distillation."

$$y_i = \frac{e^{a_i/T}}{\sum_j e^{a_j/T}} \quad \text{(Softmax Function with temperature parameter } T\text{)} \tag{1}$$

Table 5: Experiments with temperature parameter T. Number of samples per class = 16, RMSprop for numerical optimization, $\eta = 0.001, \rho = 0.9, \epsilon = 10^{-8}$ and batch size = 8 was used for all experiments.

| T | #Epochs | Train Acc(%) | Test Acc(%) |
|---|---|---|---|
| 1 | 10 | 71.00 | 56.25 |
| 2 | 10 | 67.58 | 54.49 |
| 4 | 10 | 37.01 | 38.28 |
| 8 | 10 | 5.71 | 13.28 |
| 16 | 10 | 0.59 | 1.56 |

## Team Contribution

All the team members wrote the code individually and then the best methods / practices among the group were discussed and submitted for the final report.

## Appendix

The code below includes all the utility functions used.

```
import numpy as np
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from keras.utils.np_utils import *
from keras.callbacks import *
from scipy.misc import imresize
from scipy.misc import imsave
from keras.models import Model
from matplotlib import pyplot as plt
import pdb

def get_early_stopping_callback():
        return EarlyStopping(monitor='val_loss', min_delta=1e-6, patience
            =3, verbose=0, mode='auto')


def save_all_activations(array, size_percentage, filepath):
        for i in range(0, len(array)):
                filename = filepath+str(i)+".png"
                print filename
                imsave(filename, imresize(array[i], size_percentage))


def visualize_filters(model, input_img):
        imsave("input_image.png", np.squeeze(input_img))

        #first convolution layer
        first_conv_model = Model(input=model.input, output=model.
            get_layer('block1_conv1').output)
        first_conv_output = first_conv_model.predict(input_img)

        first_conv_output = np.rollaxis(np.squeeze(first_conv_output), 2)
        save_all_activations(first_conv_output, 100, "../results/
            first_conv/")

        #last convoltution layers
        last_conv_model = Model(input=model.input, output=model.get_layer
            ('block5_conv3').output)
        last_conv_output = last_conv_model.predict(input_img)

        last_conv_output = np.rollaxis(np.squeeze(last_conv_output), 2)
```

```
        save_all_activations(last_conv_output, 800, "../results/last_conv
            /")


def subtract_mean(images):
        mean = np.mean(images) #, axis = 0)
        processed_images = images - mean

        return processed_images

def vgg_preprocessing(images):
        return preprocess_input(images)

def one_hot_encoding(labels, n_classes):
        return to_categorical(labels, nb_classes=n_classes)

def plot_temperature_graphs(best_T, train_acc, test_acc):
        legend = ['training accuracy', 'test accuracy']
        labels = ['epochs', 'percent accuracy']
        title = "Temperature based softmax regression: Best T = ", best_T

        t = np.linspace(1, len(train_acc), len(train_acc))

        plt.gca().set_color_cycle(['red', 'blue'])
        plt.plot(t, train_acc, t, test_acc)
        plt.xlabel(labels[0])
        plt.ylabel(labels[1])
        plt.title(title)
        plt.legend(legend, loc='best')
        plt.show()
```

The code below includes all the experiments used.

```
import numpy as np
import math
import utilities as util
import custom_model as model
from keras.callbacks import History
import gc
import pdb

def train(model, train_images, train_labels):
        early_stopping = util.get_early_stopping_callback()
        hist = History()

        model.fit(train_images, train_labels, batch_size=16, nb_epoch=10,
             verbose=1, callbacks=[hist],
                validation_split=0.1, validation_data=None, shuffle=True,
                     class_weight=None, sample_weight=None, initial_epoch
                    =0)

        print hist.history
        return hist

def test(model, test_images, test_labels):
        test_eval = model.evaluate(test_images, test_labels, batch_size
            =16, verbose=1)

        #predict labels
        predicted = model.predict(test_images, batch_size=16, verbose=1)

        predicted_labels = np.argmax(predicted, axis=1)
        print predicted_labels

        return test_eval[1]
```

```
def fc_model(output_dim, train_images, train_labels, test_images,
    test_labels):
        caltech_model = model.get_model(output_dim)

        #visualize filters
        # img = test_images[5]
        # img = img.reshape((1,)+img.shape)
        # util.visualize_filters(caltech_model, img)

    #Train the model
        train(caltech_model, train_images, train_labels)

    #Test the model
        print test(caltech_model, test_images, test_labels)


def convolution_model(output_dim, layer_name, train_images, train_labels,
     test_images, test_labels):
        caltech_model = model.get_convolution_model(output_dim,
            layer_name)

        #Train the model
        train(caltech_model, train_images, train_labels)

    #Test the model
        print test(caltech_model, test_images, test_labels)



def temperature_model(output_dim, train_images, train_labels, test_images
    , test_labels):

        T = [1.0, 2.0, 4.0, 8.0, 16.0]
        best_model = None
        best_loss = None
        best_T = 1

        train_accuracy = []
        test_accuracy = []

        for temp in T:
                print "------------------T = ", temp, "
                    ------------------------------------------------------
                    "
                caltech_model = model.get_temperature_model(output_dim,
                    temp)
                #Train the model
                hist = train(caltech_model, train_images, train_labels)
                hist = hist.history

                if best_loss == None or hist['loss'][len(hist['loss'])-1]
                     < best_loss:
                        best_loss = hist['loss'][len(hist['loss'])-1]
                        best_model = caltech_model
                        best_T = temp

                train_accuracy.append(hist['acc'][len(hist['loss'])
                    -1]*100.0)

                #Test the model
                test_acc = test(best_model, test_images, test_labels)
                    *100.0
```

```
                test_accuracy.append(test_acc)



        print "
            ----------------------------------------------------------------------------
            "
        print "best T found", best_T

        util.plot_temperature_graphs(best_T, train_accuracy,
            test_accuracy)

        return best_T, train_accuracy, test_accuracy
```

The code below consists of snippet for creating different models used in the project.

```
import keras
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Lambda
from keras.optimizers import SGD
from keras.optimizers import RMSprop
import pdb

def get_sgd_optimizer():
        return SGD(lr=0.001, momentum=0.5, decay=0.0, nesterov=True)

def get_rmsp_optimizer():
        return RMSprop(lr=0.001, rho=0.9, epsilon=1e-8, decay=0.0)

def get_model(output_dim):
        vgg_model = VGG16(weights='imagenet', include_top=True)

        #print vgg_model.summary()

        #now take ouput of all but last layer
        vgg_out = vgg_model.layers[-2].output

        #now lets add a softmax layer tailored for our task
        softmax_layer = Dense(output_dim, activation='softmax', name='
            softmax_256')(vgg_out)

        # this is the model we will train
        model = Model(input=vgg_model.input, output=softmax_layer)

        #Freeze all layers of base VGG16 model
        for layer in vgg_model.layers:
                layer.trainable = False

        # compile the model (should be done *after* setting layers to non
            -trainable)
        rmsp = get_rmsp_optimizer()
        model.compile(optimizer=rmsp, metrics=['accuracy'], loss='
            categorical_crossentropy')

        #verify that model is updated and appropiate
        #print model.summary()

        return model

def get_convolution_model(output_dim, layer_name):
        vgg_model = VGG16(weights='imagenet', include_top=False,
            input_shape=(224, 224, 3))
```

```
        #now take ouput of all but last layer
        vgg_out = vgg_model.get_layer(layer_name).output

        vgg_out = Flatten()(vgg_out)

        #now lets add a softmax layer tailored for our task
        softmax_layer = Dense(output_dim, activation='softmax', name='
            softmax_256')(vgg_out)

        # this is the model we will train
        model = Model(input=vgg_model.input, output=softmax_layer)

        #Freeze all layers of base VGG16 model
        for layer in vgg_model.layers:
                layer.trainable = False

        # compile the model (should be done *after* setting layers to non
            -trainable)
        rmsp = get_rmsp_optimizer()
        model.compile(optimizer=rmsp, metrics=['accuracy'], loss='
            categorical_crossentropy')

        #verify that model is updated and appropiate
        print model.summary()

        return model

def get_temperature_model(output_dim, temperature):
        vgg_model = VGG16(weights='imagenet', include_top=True)

        #now take ouput of all but last layer
        vgg_out = vgg_model.layers[-2].output

        #now add a lambda layer for temperature variable
        vgg_out = Lambda(lambda x: x * 1.0/ temperature)(vgg_out)

        #add back the softmax layer of VGG
        vgg_out = vgg_model.layers[-1](vgg_out)

        #now lets add a softmax layer tailored for our task
        softmax_layer = Dense(output_dim, activation='softmax', name='
            softmax_256')(vgg_out)

        # this is the model we will train
        model = Model(input=vgg_model.input, output=softmax_layer)

        #Freeze all layers of base VGG16 model
        for layer in vgg_model.layers:
                layer.trainable = False

        # compile the model (should be done *after* setting layers to non
            -trainable)
        rmsp = get_rmsp_optimizer()
        model.compile(optimizer=rmsp, metrics=['accuracy'], loss='
            categorical_crossentropy')

        #verify that model is updated and appropiate
        print model.summary()

        return model
```

The code below consists of main functions caltech256.

```
import numpy as np
```

```
import math
import utilities as util
import custom_model as model
from keras.callbacks import History
from experiments_model import *
import gc, os
import pdb

os.system('ulimit -s unlimited')


def pick_n_examples(train_images, train_labels, test_images, test_labels,
    n):
        relevant_examples = []
        for i in range(0, len(train_labels), 16):
                for j in range(i, i+n):
                        relevant_examples.append(j)


        train_images = train_images[relevant_examples]
        train_labels = train_labels[relevant_examples]

        #for test
        # test_n = int(math.ceil(n*20.0/100.0))
        # relevant_examples = []
        # for i in range(0, len(test_labels), 4):
        #       for j in range(i, i+test_n):
        #               relevant_examples.append(j)


        # test_images = test_images[relevant_examples]
        # test_labels = test_labels[relevant_examples]

        return train_images, train_labels, test_images, test_labels


def load_data(output_dim, example_class):
        train_images = np.load('../data/train_images.npy')
        train_labels = np.load('../data/train_labels.npy')
        test_images = np.load('../data/test_images.npy')
        test_labels = np.load('../data/test_labels.npy')

        #pick only some examples per class
        train_images, train_labels, test_images, test_labels =
            pick_n_examples(train_images, train_labels, test_images,
            test_labels, example_class)

        #preprocess
        train_images = util.vgg_preprocessing(train_images)
        train_labels = util.one_hot_encoding(train_labels, output_dim)
        test_images = util.vgg_preprocessing(test_images)
        test_labels = util.one_hot_encoding(test_labels, output_dim)


        return train_images, train_labels, test_images, test_labels




if __name__ == '__main__':
        #Output dim for our dataset
        output_dim = 256 #For Caltech256

        #load data
```

```
        train_images, train_labels, test_images, test_labels = load_data(
            output_dim, 4)

        #convoltion model
        #Experiments with last convolutional layer (softmax is attached
            after so as to reduce trainable parameters)
        #convolution_model(output_dim, "block5_pool", train_images,
            train_labels, test_images, test_labels)

        #Experiments with middle convolutional layer (softmax is attached
             after so as to reduce trainable parameters)
        #convolution_model(output_dim, "block3_pool", train_images,
            train_labels, test_images, test_labels)


        #fully connected model
        #fc_model(output_dim, train_images, train_labels, test_images,
            test_labels)

        #temperature based model
        temperature_model(output_dim, train_images, train_labels,
            test_images, test_labels)
```

The code below consists of main functions urban tribes.

```
import numpy as np
import math
import utilities as util
import custom_model as model
from keras.callbacks import History
from experiments_model import *
import pdb
import os
import gc
import scipy.misc

os.system('ulimit -s unlimited')


def pick_n_examples(train_images, train_labels, test_images, test_labels,
    n, unique_labels):
        trImgs = []
        trLbls = []
        perClassCount = np.zeros((1,len(unique_labels)))

        for i in range(np.size(train_labels,axis=0)):
            curr_label_idx = np.where(train_labels[i]==1)[0][0]
            if perClassCount[0][curr_label_idx]<n:
                    perClassCount[0][curr_label_idx] += 1
                    trImgs.append(train_images[i])
                    trLbls.append(train_labels[i])
                    #fname = "./test/file_" + str(i) + ".jpg"
                    #scipy.misc.imsave(fname, train_images[i])
        train_images = np.asarray(trImgs)
        train_labels = np.asarray(trLbls)
        #print np.shape(train_labels)

        return train_images, train_labels, test_images, test_labels


def load_data(output_dim, example_class):
        train_images = np.load('../dataUT/train_images.npy')
        train_labels = np.load('../dataUT/train_labels.npy')
        test_images = np.load('../dataUT/test_images.npy')
        test_labels = np.load('../dataUT/test_labels.npy')
```

```
        classes = np.load('../dataUT/label_decode.npy')
        #pick only some examples per class
        train_images, train_labels, test_images, test_labels =
            pick_n_examples(train_images, train_labels, test_images,
            test_labels, example_class, classes)
        #preprocess
        train_images = util.vgg_preprocessing(train_images)
        #train_labels = util.one_hot_encoding(train_labels, output_dim)
        test_images = util.vgg_preprocessing(test_images)
        #test_labels = util.one_hot_encoding(test_labels, output_dim)


        return train_images, train_labels, test_images, test_labels



if __name__ == '__main__':
        #Output dim for our dataset
        output_dim = 11 #For Urban tribes

        #load data
        train_images, train_labels, test_images, test_labels = load_data(
            output_dim, 16)

        #convoltion model
        #Experiments with last convolutional layer (softmax is attached
            after so as to reduce trainable parameters)
        #convolution_model(output_dim, "block5_pool", train_images,
            train_labels, test_images, test_labels)

        #Experiments with middle convolutional layer (softmax is attached
             after so as to reduce trainable parameters)
        #convolution_model(output_dim, "block3_pool", train_images,
            train_labels, test_images, test_labels)


        #fully connected model
        #fc_model(output_dim, train_images, train_labels, test_images,
            test_labels)
```

The code below implements reading caltech dataset.

```
import os
from keras.preprocessing import image
import numpy as np
import pdb

def read_image(filename, target_size):
        img = image.load_img(filename, target_size=target_size)
        img = image.img_to_array(img)
        return img.reshape((1,)+img.shape)

def concatenate_images(images, labels, img, label):
        if images is not None:
                images = np.concatenate((images, img))
                labels = np.concatenate((labels, label))
        else:
                images = img
                labels = label

        return images, labels

def read_caltech256(root, target_size):
        train_images = None
        train_labels = None
```

```
        test_images = None
        test_labels = None

        i = -1

        for dirname, subdirs, fileList in os.walk(root):
                #we should find 256 directories inside root
                print "Found directory", dirname

                if "clutter" in dirname:
                        continue

                if len(fileList) >= 20:
                        np.random.shuffle(fileList)
                        i += 1
                else:
                        continue

                #get training and test images
                for j in range(0, 20):
                        filename = fileList[j]
                        if filename.endswith(".jpg") or filename.endswith
                            (".jpeg"):
                        # store in image matrix
                                img = read_image(os.path.join(dirname,
                                    filename), target_size)
                                label = np.array([i])

                                if j < 16:
                                        train_images, train_labels =
                                            concatenate_images(
                                            train_images, train_labels,
                                            img, label)
                                        print "training data", i, j,
                                            train_images.shape, img.shape
                                else:
                                        test_images, test_labels =
                                            concatenate_images(
                                            test_images, test_labels, img
                                            , label)
                                        print "test data", i, j,
                                            test_images.shape, img.shape


        return train_images, train_labels, test_images, test_labels

if __name__ == "__main__":
        target_size=(224, 224)
        train_images, train_labels, test_images, test_labels =
            read_caltech256('../data/256_ObjectCategories', target_size)

        #save
        np.save('../data/train_images.npy', train_images)
        np.save('../data/train_labels.npy', train_labels)
        np.save('../data/test_images.npy', test_images)
        np.save('../data/test_labels.npy', test_labels)
```

The code below consists of snippet for reading urban tribe dataset.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 17 17:47:33 2017

@author: nimish
```

```
"""

import os
from keras.preprocessing import image
import numpy as np

os.system('ulimit -s unlimited')

def read_image(filename, target_size):
        img = image.load_img(filename, target_size=target_size)
        img = image.img_to_array(img)
        return img.reshape((1,)+img.shape)

def concatenate_images(images, labels, img, label):
        if images is not None:
                images = np.concatenate((images, img))
                labels = np.concatenate((labels, label))
        else:
                images = img
                labels = label

        return images, labels

def read_cal256(path,target_size,trainSize, testSize):
    train_images = None
    train_labels = None
    test_images = None
    test_labels = None
    unique_labels = []
    for dirName, subDirs, files in os.walk(path):
        for label in subDirs:
            if(unique_labels==None or label not in unique_labels and '.'
                in label and label != 'clutter'):
                    unique_labels.append(label)

    perClassCount = np.zeros((1,len(unique_labels)))
    perClassCountTest = np.zeros((1,len(unique_labels)))


    #for dirName, subDirs, files in os.walk(path):
    i=0
    for subDir in unique_labels:
        print i
        i +=1
        dirDone = False
        for dirName, sdir, files in os.walk(os.path.join(path,subDir)):
            np.random.shuffle(files)
            for filename in files:
                if filename.endswith(".jpg") or filename.endswith(".jpeg"
                    ):
                    label = subDir
                    lblIdx = unique_labels.index(label)

                    if(perClassCount[0][lblIdx] < trainSize):
                        perClassCount[0][lblIdx] += 1
                        img = read_image(os.path.join(dirName,filename),
                            target_size)
                        enc = np.zeros((1,len(unique_labels)))
                        enc[0][lblIdx] = 1
                        train_images, train_labels = concatenate_images(
                            train_images, train_labels, img, enc)

                    elif(perClassCountTest[0][lblIdx] < testSize):
                        perClassCountTest[0][lblIdx] += 1
```

```
                        img = read_image(os.path.join(dirName,filename),
                            target_size)
                        enc = np.zeros((1,len(unique_labels)))
                        enc[0][lblIdx] = 1
                        test_images, test_labels = concatenate_images(
                            test_images, test_labels, img, enc)
                    if(perClassCount[0][lblIdx]==trainSize and
                        perClassCountTest[0][lblIdx]==testSize):
                        dirDone = True
                        break
                if dirDone == True:
                    break

    return train_images, train_labels, test_images, test_labels,
        unique_labels

def read_UT(path,target_size,trainSize, testSize):
    train_images = None
    train_labels = None
    test_images = None
    test_labels = None
    unique_labels = []
    for dirName, subDirs, files in os.walk(path):
        for filename in files:
            if filename.endswith(".jpg") or filename.endswith(".jpeg"):
                label = filename.partition('_')[0]
                if(unique_labels==None or label not in unique_labels):
                    unique_labels.append(label)

    perClassCount = np.zeros((1,len(unique_labels)))
    perClassCountTest = np.zeros((1,len(unique_labels)))


    for dirName, subDirs, files in os.walk(path):
        np.random.shuffle(files)
        for filename in files:
            if filename.endswith(".jpg") or filename.endswith(".jpeg"):
                label = filename.partition('_')[0]
                lblIdx = unique_labels.index(label)

                if(perClassCount[0][lblIdx] < trainSize):
                    perClassCount[0][lblIdx] += 1
                    img = read_image(os.path.join(dirName,filename),
                        target_size)
                    enc = np.zeros((1,len(unique_labels)))
                    enc[0][lblIdx] = 1
                    train_images, train_labels = concatenate_images(
                        train_images, train_labels, img, enc)

                elif(perClassCountTest[0][lblIdx] < testSize):
                    perClassCountTest[0][lblIdx] += 1
                    img = read_image(os.path.join(dirName,filename),
                        target_size)
                    enc = np.zeros((1,len(unique_labels)))
                    enc[0][lblIdx] = 1
                    test_images, test_labels = concatenate_images(
                        test_images, test_labels, img, enc)


    return train_images, train_labels, test_images, test_labels,
        unique_labels

if __name__ == "__main__":
    target_size=(224, 224)
```

```
    train_images, train_labels, test_images, test_labels, oneHotDecode =
        read_UT('./pictures_all', target_size,16,4)
    np.save('./dataUT/train_images.npy', train_images)
    np.save('./dataUT/train_labels.npy', train_labels)
    np.save('./dataUT/test_images.npy', test_images)
    np.save('./dataUT/test_labels.npy', test_labels)
    np.save('./dataUT/label_decode.npy', oneHotDecode)
    print 'urban tribes ready to load'
    del train_images, train_labels, test_images, test_labels,
        oneHotDecode

    train_images, train_labels, test_images, test_labels, oneHotDecode =
        read_cal256('./256_ObjectCategories', target_size,16,4)
    np.save('./data256/train_images.npy', train_images)
    np.save('./data256/train_labels.npy', train_labels)
    np.save('./data256/test_images.npy', test_images)
    np.save('./data256/test_labels.npy', test_labels)
    np.save('./data256/label_decode.npy', oneHotDecode)
    print 'cal256 ready to load'
    del train_images, train_labels, test_images, test_labels,
        oneHotDecode
```