Image Classification Using Transfer Learning

Members:

- Lenord Melvix Joseph Stephen Max (A53092538)
- Thyagarajan Venkatanarayanan (A53097887)
- Srinivas Avireddy (A53101356)
- Abhishek Majumdar (A53097687)

Abstract

In this assignment, we will attempt to classify images from the Caltech256 (a dataset similar to the ImageNet) and Urban Tribe (a dataset not so similar to ImageNet) datasets using transfer learning, with the help of Keras (a Deep Learning library that provides high-level utilities to build deep networks). We will use a pre-trained convolutional neural network (CNN), VGC16, originally trained to classify images into 1000 classes. We shall replace the softmax Layer of the VGG16 pre-trained model with our own softmax Layer which will predict classes of our dataset. We will attempt to train only the softmax Layer of the new model using a small subset of examples from our dataset and observe if it performs well.

---

1. Overview

Convolutional neural networks:

We use a convolutional neural network (CNN) to perform classification on our datasets. A CNN is very similar to a regular neural network, consisting of neurons whose weights can be trained, with a differentiable cost function and we can use the usual training methods to train the CNN. The unique feature of CNNs is that the input layer in their architecture assumes that the inputs are images, because of which the network will have some unique properties. This helps in having an efficient implementation for forward propagation and also make the weight matrices more sparse.

Transfer Learning:

Transfer Learning aims at solving new tasks using knowledge gained from solving related tasks. Deep architectures of Convolutional Neural Networks, with exploding number of parameters, cannot be trained on datasets of incomparable size. We use an existing model, trained on a very large scale dataset, as feature extractors or initial layers of a ConvNet that can be fine-tuned to learn the underlying model of the dataset of our new task.

In this assignment, we take VGG16, a CNN trained on the ImageNet dataset (which contains 1.2 million images with 1000 categories), replace its final softmax layer(s) with another softmax layer with output nodes equal to the number of classes in our new dataset. We then only train this final layer with very few number of training examples, using Keras.

VGG16:

The inputs to the CNN are standard sized 224 x 224 x 3 RGB images. The image is then passed through a stack of convolutional layers, which have 3 x 3 filters (that extract low level features). The convolution stride is set to 1 pixel and the spatial padding is such that spatial resolution is preserved after convolution, i.e. the padding is

1 pixel for 3 x 3 convolutional layers. Then, spatial pooling is carried out by 5 max-pooling layers, which follow only some of the convolutional layers where max-pooling is performed over a 2x2 pixel window, with the stride set to 2.

A stack of convolutional layers of varying depth is followed by three fully-connected (FC) layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. All hidden layers are equipped with the rectification (ReLU) non-linearity. This pre-trained network (on ImageNet dataset) has been imported from Keras for classification using transfer learning

2. Caltech 256 Image Classification

Dataset used:

The Caltech-256 dataset consists of images divided into 257 classes. For our assignment, we will only use the first 256 classes, leaving out the images from the 'clutter' category.

Pre-processing input images:

Since the images in Caltech256 dataset are of different sizes, each image is reshaped to a 224 x 224 x 3 image and then pre-processed using a module from Keras. The corresponding labels for each image are one-hot encoded to outputs of dimension 256.

Training CNN:

Since the output layer in VGG-16 has 1000 nodes, we replace it with a layer consisting of 256 nodes with softmax activation function since the Caltech256 dataset has only 256 classes and we only need to train the last layer of the network.

We study the dependency of the accuracy of classification on the size of training set used. We pick K examples from each class at random and then shuffling it. The network is trained using K = 2, 4 and 8 and the test accuracy in each case is observed. Next, we use other intermediate layers as input to the softmax (output) layer and again training only the last layer of the CNN. Finally, to visualize filter activations (first and last convolutional layers) we pick a random sample from the test set and perform forward propagation in the CNN

3. Results/performance

Dependency on the number of training examples per class:

Number of examples per class	**Best test accuracy**
2	39 %
4	52 %
8	61 %
16	66 %

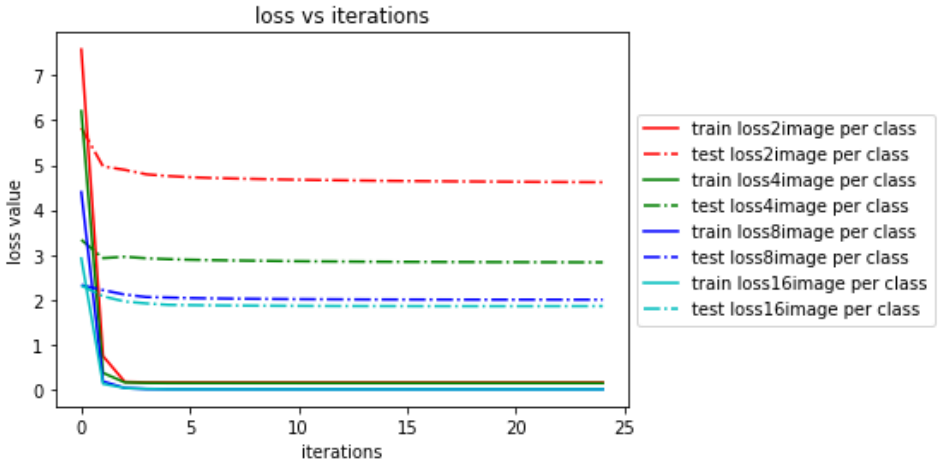

Figure 1 - Training and test loss vs number of epochs for different values of K

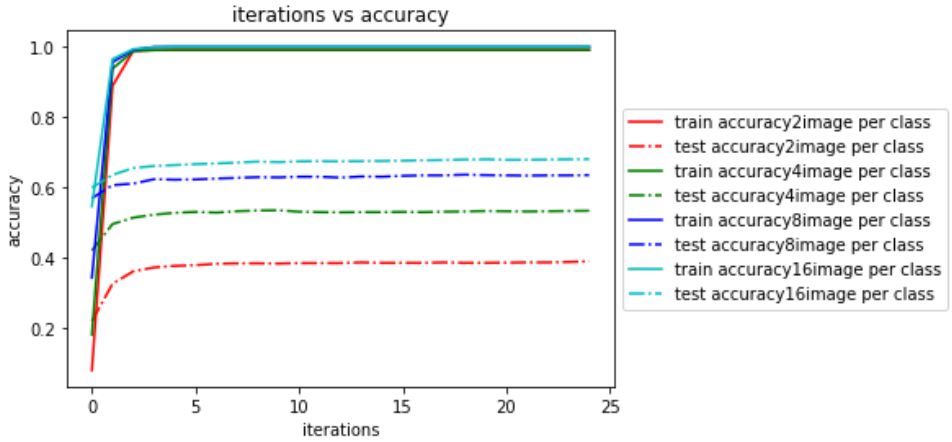

Figure 2 - Train accuracy vs epochs for different values of K

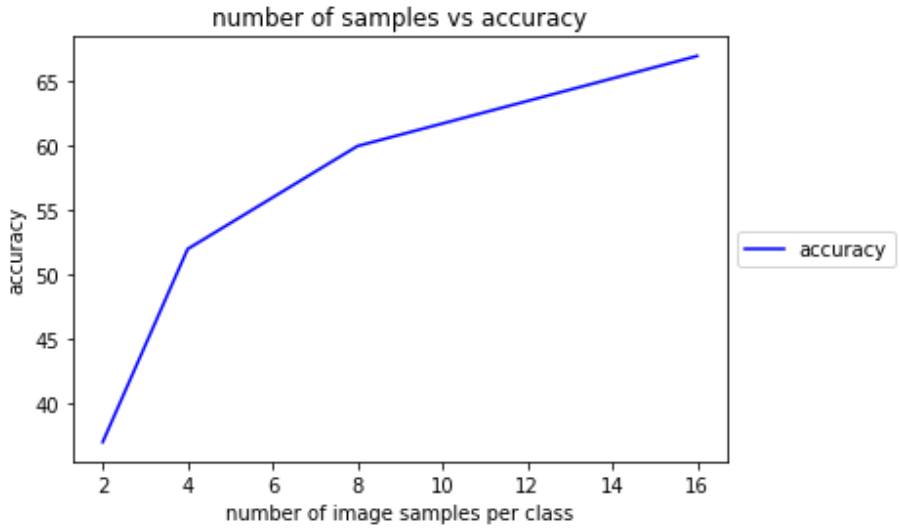

Figure 3 - Test accuracy vs number of examples per class

By training just one layer (the output layer), with the same set of weights, the network is able to learn important high level representative features from the images. As expected, the training accuracy increases and the loss function decreases monotonically with number of epochs and the test accuracy increases as the size of training set is increased. The VGG-16 network is pre-trained on the ImageNet dataset which is much larger than Caltech-256 (1.2 million images from 1000 classes vs 25k images with 256 classes) because of this, transfer learning works well here.

Visualization of filter activations in different convolutional layers:

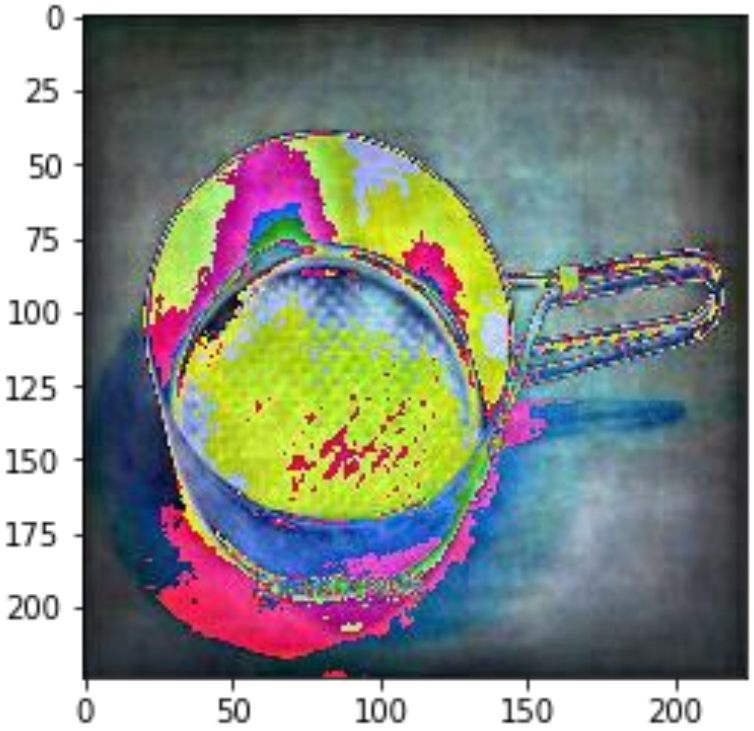

Figure 4 - Original Image

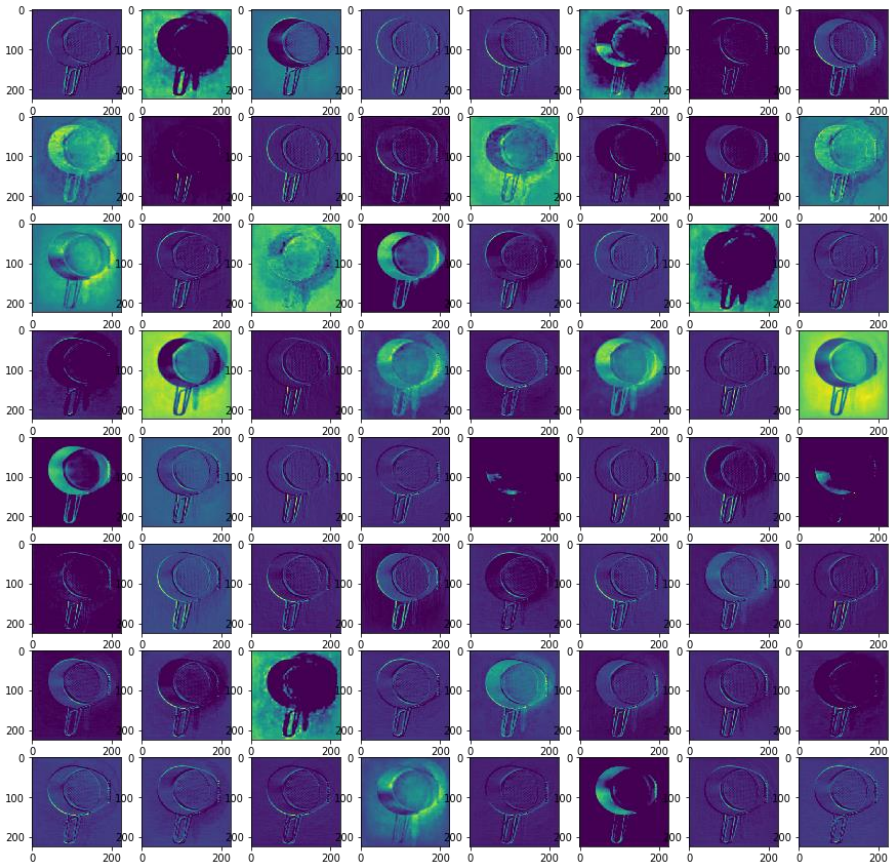

Figure 5 - Filter activations in first layer

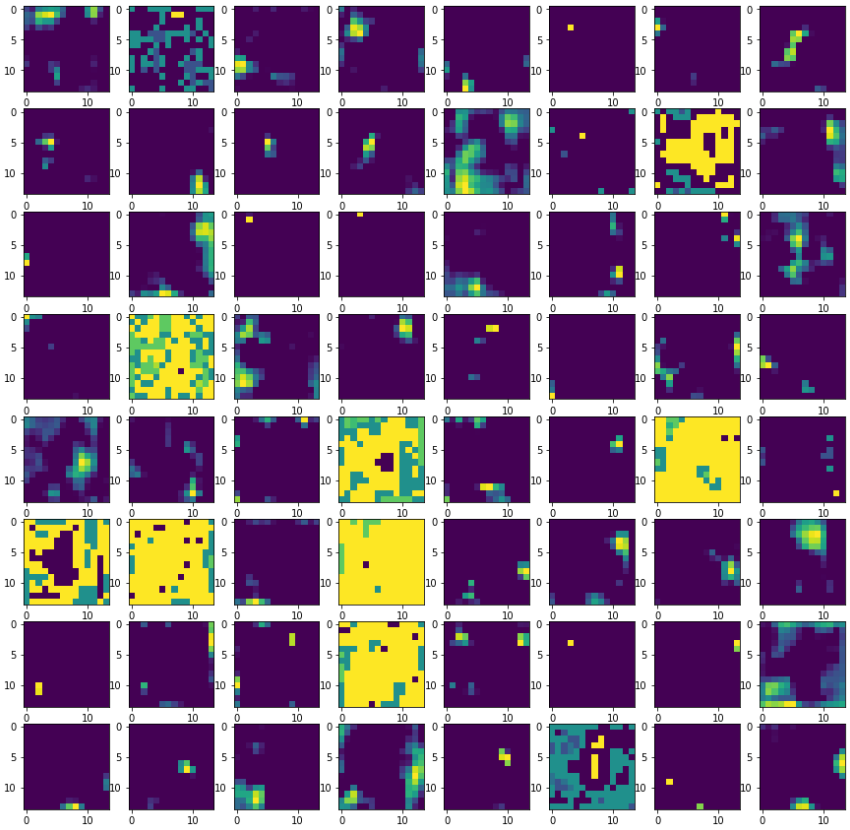

Figure 6 - Filter activations in final layer

We also visualize the filter activations of the first and last convolutional layers of VGG-16 network. For this purpose, we use 1 image from the test set. We can see that the first layer encodes the colours and edges present in the images, which are then used by the top layers to produce more complex features.

<u>Dependency on changing which intermediate layer feeds into the output layer:</u>

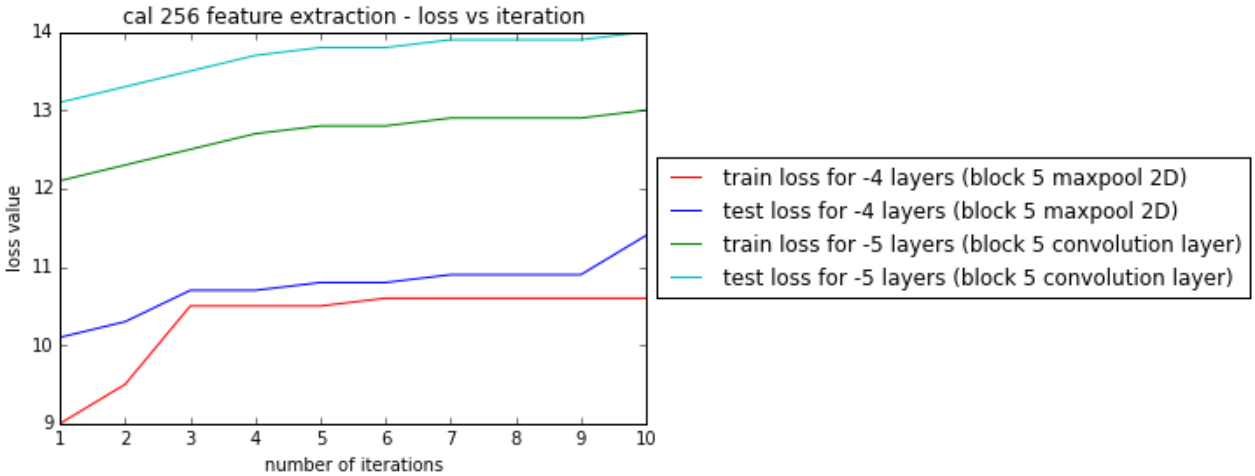

Figure 7 – Train and test loss vs number of epochs for different layer input to softmax

Four layers from the back – Maxpooling over a 2 x 2 window with stride 2

Five layers from the back – convolutional layer with 512 features feeding into the softmax layer

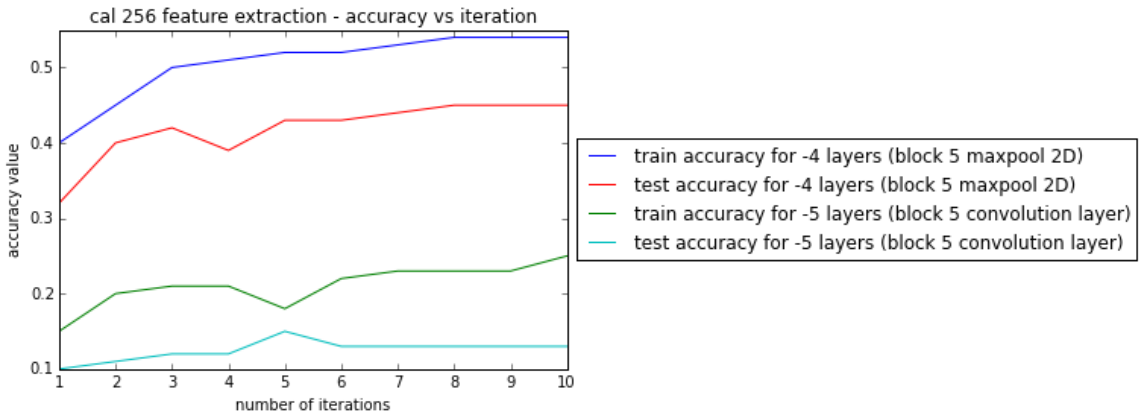

Figure 8 – Train and test accuracy vs number of epochs for different layer input to softmax

Finally, we experimented using different intermediate layers of the network as input to the softmax Layer. We observed that the classification accuracy decreased as we moved the softmax layer closer to the input layer. This is because the feature representations learned by the top layers of the CNN are more useful in classifying the images since the initial layers learn only the low level features, which will not give good accuracy.

4. Urban Tribes Classification

Dataset used:

We used the Urban Tribes dataset in the context of recognition of social styles of people, which classifies people into different social groups such as bikers, hipsters, formal, hiphop, goth etc. This dataset consists of 11 classes in total.

Pre-processing input images:

We pre-process the images the same way we did for the Caltech256 dataset, but now, the labels are one-hot encoded as outputs of dimension 11.

Training CNN:

The last layer of modified VGG-16 network will now have 11 nodes with softmax activation. Once again, we follow the same procedure that we used before and analyse the performance of the model for different sizes of training set (i.e. for different values of K) and also for different intermediate layer inputs to the convolutional layer. Also, the filter activations for the first and last convolutional layers are visualized.

5. Results/performance

Dependency on the number of training examples per class:

Number of examples per class	Best test accuracy
2	33 %
4	41 %
8	51 %
16	59 %

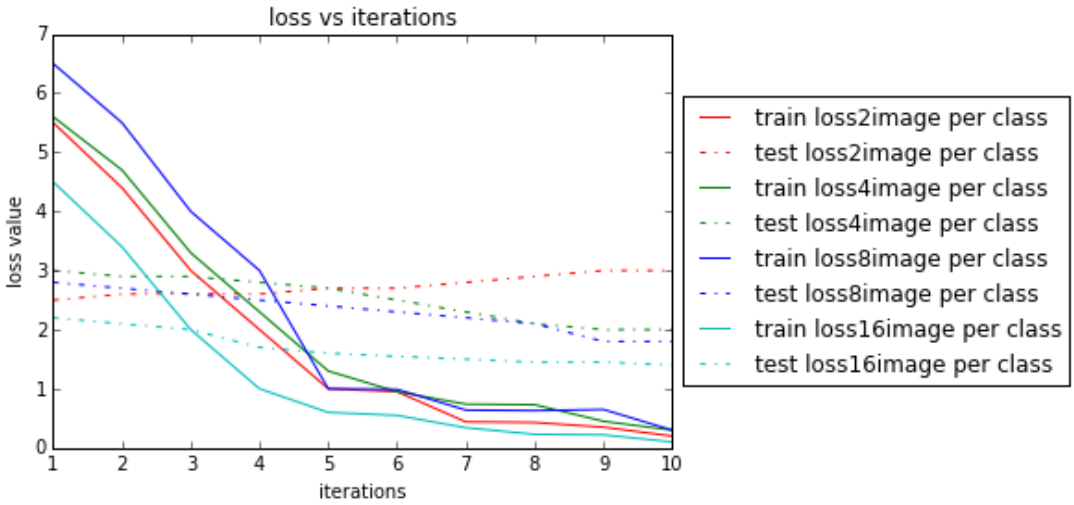

Figure 9 - Training and test loss vs epochs for different values of K

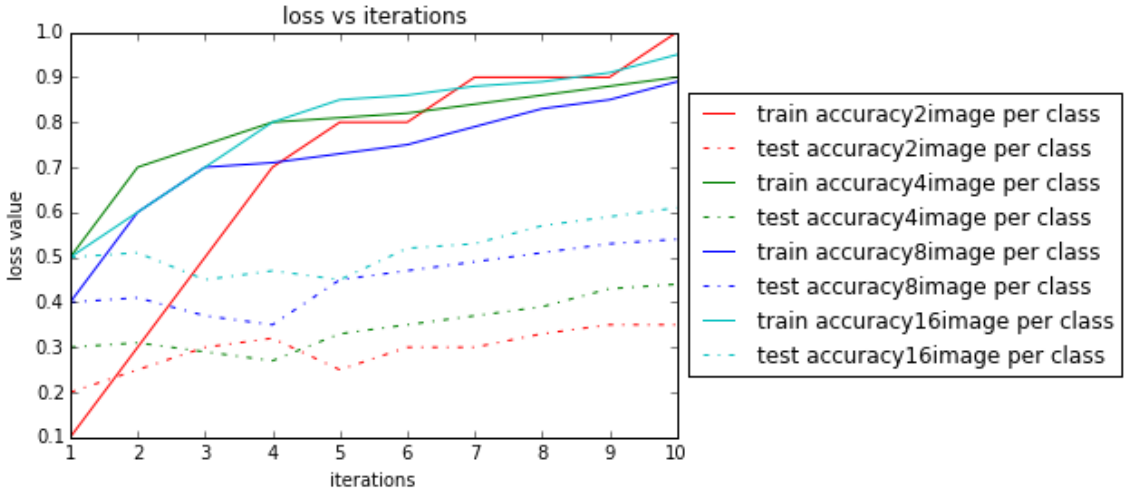

Figure 10 - Training and test accuracy vs number of epochs for different values of K

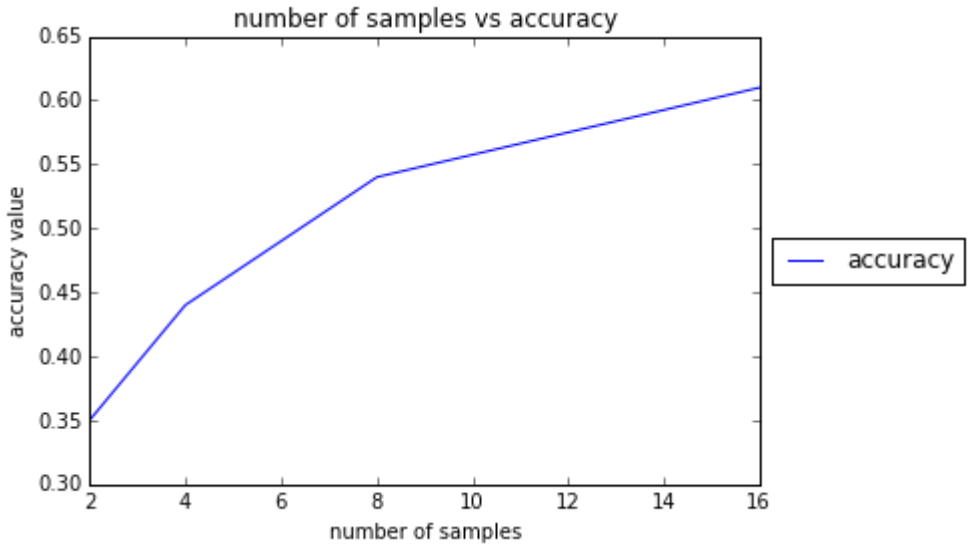

Figure 11 - Test accuracy vs number of examples per class

We observe a similar trend in training and test loss as we did earlier for Caltech256. As expected, classification accuracy increases as the size of the training set is increased.

However, we also see that for the same size of training and test sets, the accuracy of the model on UrbanTribe is lower than the corresponding numbers for Caltech256. Although the UrbanTribe dataset is much smaller in size than Caltech256 (about 1k images), we concluded that the performance is not as good because the UrbanTribe dataset is less similar to the ImageNet dataset than Caltech256. Thus, the pre-trained weights do not work as well as before.

Visualization of filter activations in different convolutional layers:

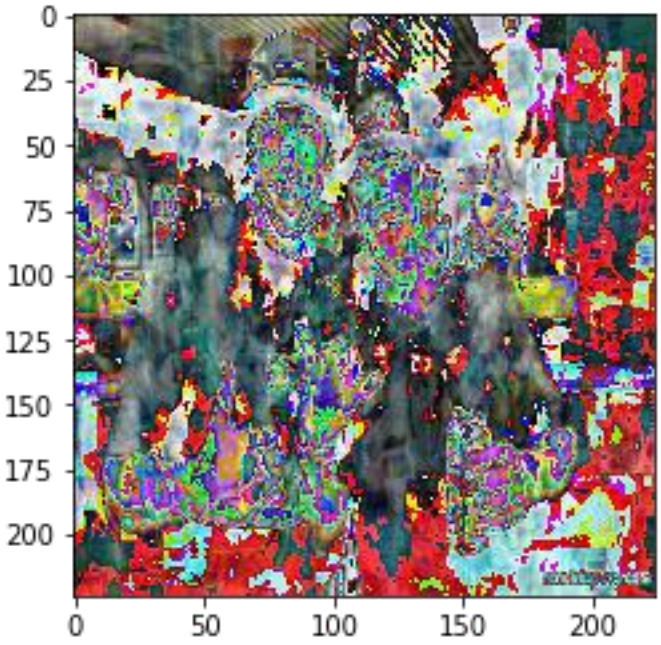

Figure 12 - Original image

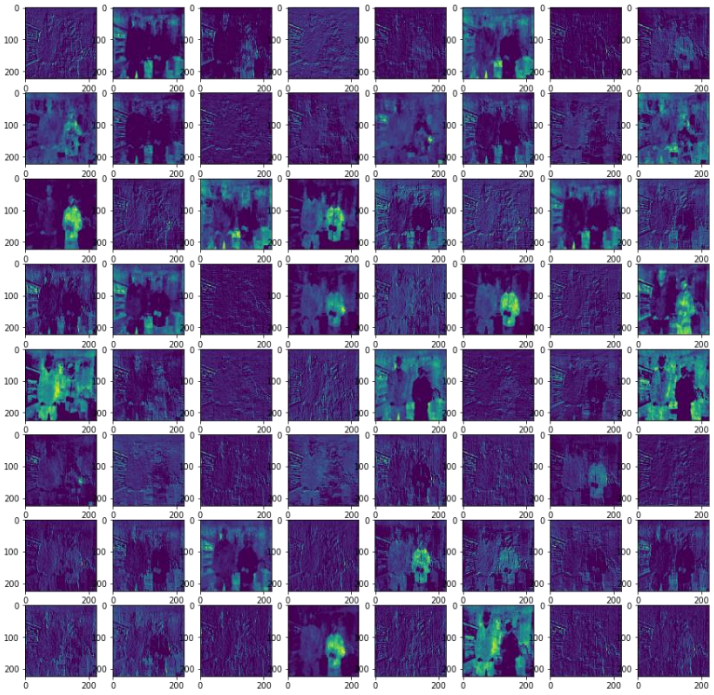

Figure 13 - Filter activations in first layer

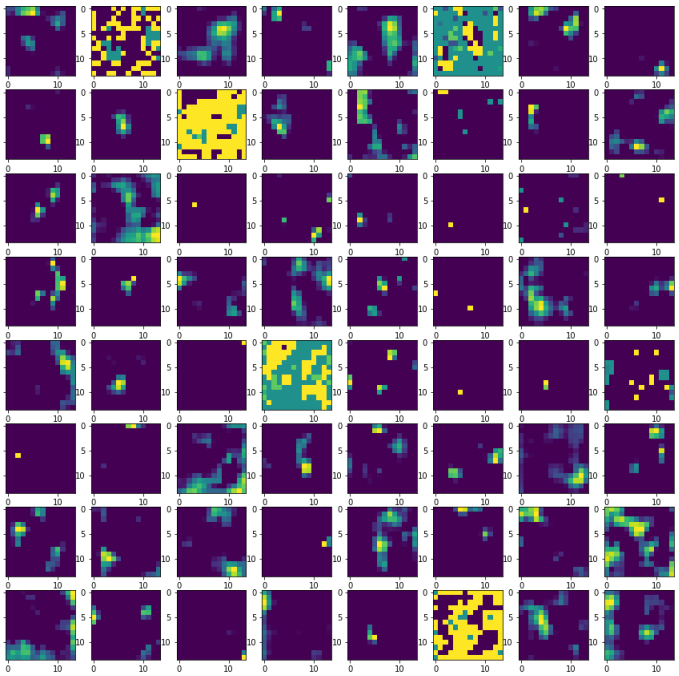

Figure 14 - Filter activations in final layer

The conclusions are the same as before.

<u>Dependency on changing which intermediate layer feeds into the output layer:</u>

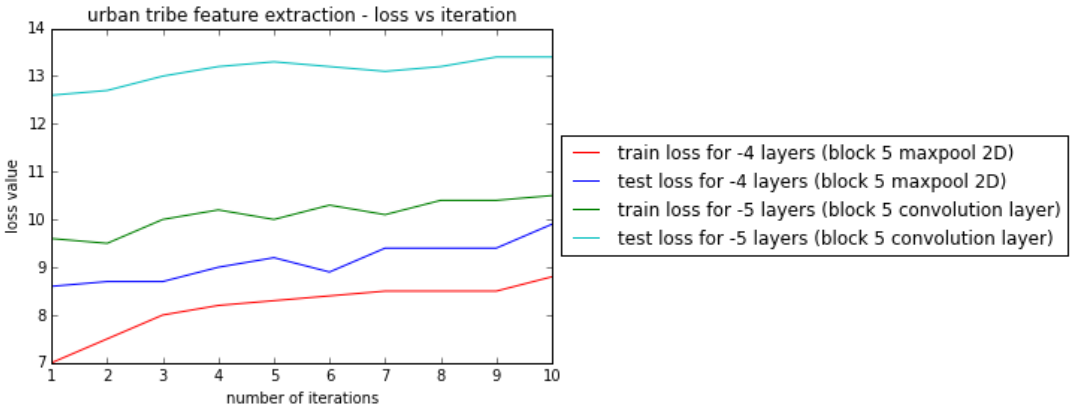

Figure 15 - Train and test loss vs number of epochs for different layer inptus to softmax

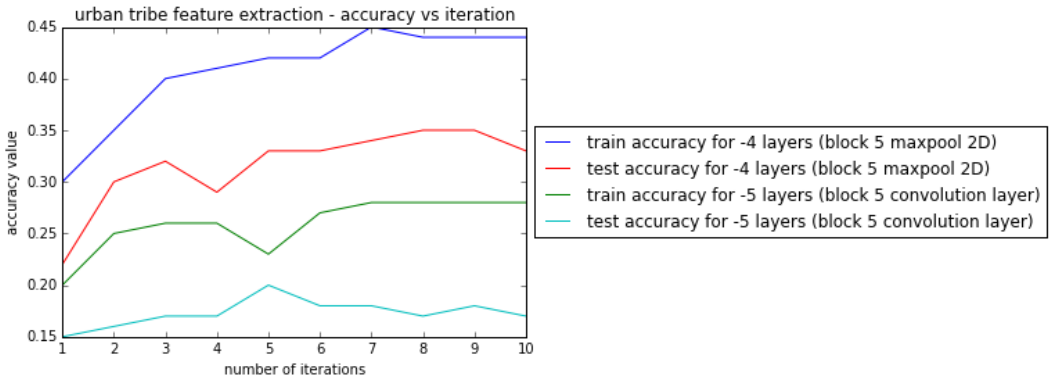

Figure 16 - Train and test accuracy vs number of epochs for different layer input to softmax

Once again, the explanations for the trend in accuracy are the same as before.

6. Summary

In this assignment, we performed a complex image classification task with the help of a pre-trained CNN by training only the output layer. We visualised the filter activations from 2 different layers to understand the advantage of transfer learning and the propagation of data through convolutional neural network. We saw that the first layer of the CNN effectively extracted low level features like edges, while the last layer helped in extracting high level features which are very useful for the final softmax classification. Experimenting with different intermediate convolutional layers as input to softmax showed that the test accuracy drops rapidly as we go deeper into the CNN, thus leading to the conclusion that most of the weights are concentrated in the top layers of the CNN. As expected, the accuracy increased when we used more number of training examples.

7. Contributions

Task	Contributors
Reading images, pre-processing	Pair Programming (Thyagarajan & Abhishek)
Training Caltech256 ConvNet	Pair Programming (Srinivas & Lenord)
Experiment with intermediate CNN layers & training examples	Pair Programming (Srinivas & Lenord)
Report Writing	Thyagarajan, Abhishek

8. APPENDIX

Python code

```
import os
import tarfile
import numpy as np
from scipy import misc
import keras
from keras.layers import Dense
from keras.models import Model
from keras.models import Sequential
from keras.applications import VGG16
import matplotlib.pyplot as plt


class ExtractFile(object):
    def __init__(self,filename):
        '''
        Initialize object to extract images from tar file
located at root
        :param filename: filename of tar file
        :return: NONE
        '''
        self.root =
os.path.splitext(os.path.splitext(filename)[0])[0]
        self.filename = filename

    def uncompress(self):
        '''
        Uncompress tar file if uncompressed file is not
available
        :return: NONE
        '''
        if os.path.isdir(self.root):
            print('Images already extracted ')
        else:
            print('Extracting data...')
            tar = tarfile.open(self.filename)
            tar.extractall(path="data")
            tar.close()

    def get_root(self):
        '''
        Get root directory where the images are
categorically stored
        :return:
        '''
        return self.root


class Preprocess(object):
    def __init__(self, image_size):
        self.image_size = image_size
        self.channel = image_size[2]

    @staticmethod
    def shuffle(data, labels):
        '''
        Randomly shuffle data for better performance
        :param data: numpy array of image dataset
        :param labels: numpy array of image labels
        :return: shuffled image and labels
        '''
        permutation =
np.random.permutation(data.shape[0])
        labels = labels[permutation]
        data = data[permutation]
        return data, labels

    @staticmethod
    def normalize_data(data):
        '''
        Normalize data by subtracting mean and dividing
```

```
by standard deviation
        :param data: numpy array of image dataset
        :return: normalized image dataset
        '''
        data -= data.mean(axis=0)
        return data/np.std(data, axis=0)

    @staticmethod
    def get_image_folders(root_path):
        '''
        Get list of directories with images
        :param root_path: root path containing image
subdirectories
        :return: list of image folders
        '''
        image_folders = [os.path.join(root_path, d) for d
in sorted(os.listdir(root_path)) if os.path.isdir(
            os.path.join(root_path, d))]
        return image_folders

    def process_all_images(self, directories,
images_per_category):
        '''
        From each directory, extact images, randomly pick
specified set of images, resize them and load into
array
        :param directories: list of subdirectories with
images
        :param images_per_category: number of images per
category to be loaded
        :return: numpy array with data and labels
        '''
        count = 0
        label = 0
        labels = np.array([])
        for directory in directories:
            images = np.array([f for f in
os.listdir(directory) if
f.lower().endswith('.jpg')])[:images_per_category]
            for f in images:
                count += 1
                labels = np.append(labels, label)
            label += 1

        # Initialize empty data array
        data = np.zeros((count, self.image_size[0],
self.image_size[1], self.channel))
        count = 0

        for directory in directories:
            images = np.array([f for f in
os.listdir(directory) if f.lower().endswith('.jpg')])
            # Randomly pick a fixed set of images from
each category
            permutation =
np.random.permutation(len(images))[:images_per_category
]

            for f in images[permutation]:
                data[count] = np.reshape(misc.imresize(
                    misc.imread(os.path.join(directory, f),
mode='RGB'),self.image_size),
                    (-1, self.image_size[0],
self.image_size[1], self.channel)
                )
                count += 1

        # Assign labels to each image
        labels = (np.arange(np.min(labels),
np.max(labels)+1) == labels[:, None]).astype(float)
        return data, labels


class DisplayStatistics(object):

    def __init__(self):
        pass

    @staticmethod
    def plot_history(model_history):
        '''
        Plot statics of model performance based on
history data
        :param model_history: list of model history
objects
        :return: NONE
        '''
        for history in model_history:
            print(history.history.keys())
            plt.plot(history.history['acc'])
            plt.plot(history.history['loss'])
            plt.plot(history.history['val_acc'])
            plt.plot(history.history['val_loss'])


class TransferLearning(object):

    @staticmethod
    def get_vgg_model(layers_to_remove):
        '''
        Generate VGG16 model with speficied number of
layers removed to implement with pre-trained weights
        :param output_dim:
        :return: VGG16 pre-trained model with last two
layers replaced with Softmax of 256 dimensions
        '''
        vgg_model =
keras.applications.VGG16(weights='imagenet',
include_top=True)

        # Use all layers except the last two
        vgg_out = vgg_model.layers[-
layers_to_remove].output

        # Build transfer learning model with predefined
layers
        tl_model = Model(input=vgg_model.input,
output=vgg_out)

        # Freeze all layers of VGG16 and Compile the
model
        for layer_idx in range(len(tl_model.layers)):
            tl_model.layers[layer_idx].trainable = False

        # Compile the configured model
        tl_model.compile(
            optimizer='rmsprop',
            loss='categorical_crossentropy',
            metrics=['accuracy'])

        # Confirm the model is appropriate
        print tl_model.summary()

        return tl_model

    @staticmethod
    def softmax_model(vgg16_output, output_dim):
        soft_model = Sequential()
        soft_model.add(Dense(output_dim=output_dim,
activation='softmax', input_dim=vgg16_output.shape[1]))
        soft_model.compile(
            optimizer='rmsprop',
            loss='categorical_crossentropy',
            metrics=['accuracy']
        )
        print soft_model.summary()
        return soft_model


class GetData(object):

    def __init__(self, root_path, images_per_category,
to_extract):
        self.filename = root_path
        self.image_size = [224, 224, 3]
        self.num_images_per_category =
images_per_category
        self.to_extract = to_extract

    def __extract_file(self):
        extracter = ExtractFile(self.filename)
        extracter.uncompress()
        print('Uncompression Complete !')
        return extracter.get_root()

    def __preprocess_images(self, root_path):

        preprocessor = Preprocess(self.image_size)
```

```
        # Get list of image folders
        image_folders =
preprocessor.get_image_folders(root_path)

        # Convert all images into tuple of numpy data
array and labels
        data, labels =
preprocessor.process_all_images(image_folders,
self.num_images_per_category)

        # Normalize data and shuffle the dataset
        data = preprocessor.normalize_data(data)
        data, labels = preprocessor.shuffle(data, labels)

        return data, labels

    def get_processed_data(self):

        # Uncompress tar file and load images
        if self.to_extract == True:
            root = self.__extract_file()
        else:
            root = self.filename

        # Preprocess images and load as numpy array for
training
        data, labels = self.__preprocess_images(root)

        return root,data, labels


def transferlearn_caltech256():

    num_of_classes = 257
    num_of_epochs = 25
    batch_size = 32
    validation_split = 0.3
    to_extract = True
    layers_to_remove = [2]
    num_images_per_category = [2]
    display_history = DisplayStatistics()

    # Array to store Train Validation Statistics for
Model fitting
    caltech_model_fit_history = []

    for img_per_category in num_images_per_category:

        #Extract and process dataset with specified
images per category
        caltech256 = GetData(
            'data/256_ObjectCategories.tar',
            img_per_category,
            to_extract
        )
        root, caltech_data, caltech_label =
caltech256.get_processed_data()
        tl = TransferLearning()

        # For each model with layers removed at tail end,
forward propagate input
        # through ImageNet and store output of network in
numpy array file
        for to_remove in layers_to_remove:

            # Get VGG16 model wit layers removed
            tl_model = tl.get_vgg_model(to_remove)
            numpy_array_file = "caltech256_" +
str(to_remove) + ".npy"

            if os.path.isfile(numpy_array_file):
                print str(numpy_array_file) + " exists --
Skipping forward propagation"
                continue

            print "Propagating CalTech data through VGG16
NN with " + str(to_remove) + " layers removed"
            # Get output for dataset
            vgg16_output = tl_model.predict(caltech_data)
            print "Forward Propagation Complete"

            # Store in file for future processing
            # TODO: Comment out this code block once
complete. It is a one time action

            np.save(numpy_array_file,vgg16_output)
            print "Stored numpy array in file : " +
str(numpy_array_file)

        for to_remove in layers_to_remove:

            # Load forward propagated output for each
model
            numpy_array_file = "caltech256_" +
str(to_remove) + ".npy"
            loaded_vgg16_output =
np.load(numpy_array_file)
            print "Loaded forward propagated output from
VGG16 for " + str(to_remove) + " removed network"

            # Train Softmax model with VGG16's output as
input
            vgg16_tl_model =
tl.softmax_model(loaded_vgg16_output, num_of_classes)
            history = vgg16_tl_model.fit(
                loaded_vgg16_output,
                caltech_label,
                nb_epoch=num_of_epochs,
                batch_size=batch_size,
                validation_split=validation_split)

            caltech_model_fit_history.append(history)


display_history.plot_history(caltech_model_fit_history)


def transferlearn_urbantribe():

    num_of_classes = 11
    num_of_epochs = 25
    batch_size = 32
    validation_split = 0.3
    layers_to_remove = [2]
    num_images_per_category = [2]
    to_extract = False
    display_history = DisplayStatistics()

    # Array to store Train Validation Statistics for
Model fitting
    urbantribe_model_fit_history = []

    for img_per_category in num_images_per_category:

        #Extract and process dataset with specified
images per category
        urbantribe = GetData(
            'data/pictures_all',
            img_per_category, to_extract
        )
        root, caltech_data, caltech_label =
urbantribe.get_processed_data()
        tl = TransferLearning()

        # For each model with layers removed at tail end,
forward propagate input
        # through ImageNet and store output of network in
numpy array file
        for to_remove in layers_to_remove:

            # Get VGG16 model wit layers removed
            tl_model = tl.get_vgg_model(to_remove)
            numpy_array_file = "urbantribe_" +
str(to_remove) + ".npy"

            if os.path.isfile(numpy_array_file):
                print str(numpy_array_file) + " exists --
Skipping forward propagation"
                continue
            print "Propagating UrbanTribe data through
VGG16 NN with " + str(to_remove) + " layers removed"
            # Get output for dataset
            vgg16_output = tl_model.predict(caltech_data)
            print "Forward Propagation Complete"

            # Store in file for future processing
            # TODO: Comment out this code block once
```

```
complete. It is a one time action

        np.save(numpy_array_file,vgg16_output)
        print "Stored numpy array in file : " +
str(numpy_array_file)

    for to_remove in layers_to_remove:

        # Load forward propagated output for each
model
        numpy_array_file = "urbantribe_" +
str(to_remove) + ".npy"
        loaded_vgg16_output =
np.load(numpy_array_file)
        print "Loaded forward propagated output from
VGG16 for " + str(to_remove) + " removed network"
```

```
        # Train Softmax model with VGG16's output as
input
        vgg16_tl_model =
tl.softmax_model(loaded_vgg16_output, num_of_classes)
        history = vgg16_tl_model.fit(
           loaded_vgg16_output,
           caltech_label,
           nb_epoch=num_of_epochs,
           batch_size=batch_size,
           validation_split=validation_split)

        urbantribe_model_fit_history.append(history)


display_history.plot_history(urbantribe_model_fit_histo
ry)
```