Using Neural Networks to Effectively Classify Hand-Written Digits of the MNIST Dataset

Chetan Gandotra
UC San Diego
9500 Gilman Drive
`cgandotr@ucsd.edu`

Rishabh Misra
UC San Diego
9500 Gilman Drive
`r1misra@ucsd.edu`

Abstract

This report discusses the second programming assignment of our course CSE 253: Neural Networks and Pattern Recognition, its solutions and the inferences we drew. A variable layer neural network was implemented from the scratch and observations were made based on various parameters and before/after adding certain trades of tricks as discussed in Yann LeCun's famous paper "Efficient BackProp". The data-set used was the famous MNIST data-set and a ten-way classification was performed on it. A test data-set accuracy in excess of 97% was achieved using various mechanisms and tricks, which is almost at par with the accuracy reported by LeCun on his website.

1 Task 3: Implementing Neural Network and Gradient Calculation

Submitted to 30th Conference on Neural Information Processing Systems (NIPS 2016). Do not distribute.

Appendix

The code consists of three files - vgg16_starter_1.py

vgg16_starter_1.py

```
from keras.applications import VGG16
from keras.models import Model
from os import walk
from os.path import join
import numpy as np
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from sklearn.utils import shuffle
import numpy
from keras.layers import Dense, Flatten
import matplotlib.pyplot as plt
from keras.optimizers import SGD, RMSprop
from keras.applications.resnet50 import preprocess_input

def f_softmax(X):
    Z = numpy.sum(numpy.exp(X), axis=1)
    Z = Z.reshape(Z.shape[0], 1)
    return numpy.exp(X) / Z

def getModel(output_dim):
    '''
        * output_dim: the number of classes (int)

        * return: compiled model (keras.engine.training.Model)
    '''
    vgg_model = VGG16( weights='imagenet', include_top=True )
    vgg_out = vgg_model.layers[18].output #Last FC layer's output
    for layer in vgg_model.layers:
        layer.trainable = False
    flatten1 = Flatten()(vgg_out)
    softmax_out = Dense(257,activation = 'softmax')(flatten1)
    #Create softmax layer taking input as vgg_out
    #Create new transfer learning model
    tl_model = Model( input=vgg_model.layers[1].input, output=softmax_out)

    #Freeze all layers of VGG16 and Compile the model
    #Confirm the model is appropriate

    return tl_model

def load_image_custom(image_path):
    img = image.load_img(image_path, target_size = (224, 224, 3))
    data = image.img_to_array(img)
    return data

def load_all_images_in_folder(dir_path, dir_name, examples_per_class,
                              validation_per_class):
    images_list = []
    images_names = []
    val_images_list = []
    val_images_names = []
    for dirpath, dirnames, filenames in walk(dir_path):
        if (len(filenames) > 0):
            np.random.shuffle(filenames)
            cnt = 0
            for file_name in filenames:
                cnt += 1
                if (cnt <= examples_per_class):
                    file_path = join(dir_path, file_name)
                    images_list.append(load_image_custom(file_path))
                    #images_names.append(dir_name)
                    images_names.append(file_path)
                elif (cnt <= examples_per_class+validation_per_class):
                    file_path = join(dir_path, file_name)
                    val_images_list.append(load_image_custom(file_path))
                    #val_images_names.append(dir_name)
                    val_images_names.append(file_path)
    return images_list, images_names, val_images_list, val_images_names

def load_all_images(base_dir_path, base_dir_name, images_list, images_names,
                    val_images_list, val_images_names, examples_per_class,
                    validation_per_class):
    for dirpath, dirnames, filenames in walk(base_dir_path):
        if (len(dirnames) > 0):
            for dir_name in dirnames:
                dir_path = join(base_dir_path, dir_name)
                images_list1, images_names1, val_images_list1, val_images_names1 = lo
                                                        dir_name, examples_per_class
                                                        validation_per_class)
                images_list = images_list + images_list1
                images_names = images_names + images_names1
                val_images_list = val_images_list + val_images_list1
                val_images_names = val_images_names + val_images_names1
    return images_list, images_names, val_images_list, val_images_names

def get_one_hot(images_names):
    all_categories = list(sorted(set(images_names)))
    C = len(all_categories)
    one_hot = np.zeros((len(images_names), C))
    for i in range(len(images_names)):
        index1 = all_categories.index(images_names[i])
        one_hot[i, index1] = 1.0
    return one_hot

def getModel3(output_dim, conv_layer):
    '''
        * output_dim: the number of classes (int)

        * return: compiled model (keras.engine.training.Model)
    '''
    vgg_model = VGG16(weights='imagenet', include_top=True)
    for layer in vgg_model.layers:
        layer.trainable = False;
    vgg_out1 = vgg_model.layers[conv_layer].output #Last FC layer's output

    #Create new transfer learning model
    tl_model = Model( input=vgg_model.input, output=vgg_out1 )

    #Freeze all layers of VGG16 and Compile the model
    #Confirm the model is appropriate

    return tl_model

```

```
if __name__ == '__main__':
    #Output dim for your dataset
    output_dim = 257 #For Caltech256
    images_list = []
    images_names = []
    val_images_list = []
    val_images_names = []
    examples_per_class = 3
    validation_per_class = 1
    images_list, images_names, val_images_list, val_images_names = load_all_images(
                    'C:/Users/Chetan/Documents/CSE253/PA3/256_ObjectCategories',
                    '256_ObjectCategories',
                    images_list, images_names, val_images_list, val_images_names,
                    examples_per_class, validation_per_class)
    # Normalization
    images_list=preprocess_input(np.array(images_list))
    val_images_list=preprocess_input(np.array(val_images_list))
    images_list = images_list/255.0
    val_images_list = val_images_list/255.0

    # Get one hot representation
    image_category = get_one_hot(images_names)
    val_image_category = get_one_hot(val_images_names)
    # Shuffle
    X_train, y_train = shuffle(images_list, image_category)
    tl_model = getModel(output_dim)
    tl_model.summary()
    #Train the model
    tl_model.compile(loss='categorical_crossentropy', optimizer=RMSprop(lr=0.001,
                                                                    decay=1e-2,
                                                                    rho=0.9,
                                                                    epsilon=1e-08),
              metrics=['acc'])

    tl_model.fit(X_train, y_train, batch_size=10, nb_epoch=10,
          validation_data=(val_images_list, val_image_category), shuffle=True)

    # Code from here on is to pick one image and visualize it at layer 1 and 17
    X_train = images_list
    y_train = images_names
    lst = np.empty([1,224,224,3])
    lst[0] = X_train[101]
    print (lst.shape)
    print (images_names[101])
    model_50 = getModel3(output_dim,17)#1
    model_50.compile(loss = "categorical_crossentropy", optimizer = "sgd",
                     metrics = ["acc"])
    #model_50.summary()
    predict_50 = model_50.predict(lst)
    print (predict_50.shape)
    w, h = 14, 14#224, 224 #14, 14
    data = np.zeros((h, w, 1), dtype=np.uint8)
    for i in range(1):
        title='layer17'+'.png'
        data=np.empty([h, w])
        for r in range(h):
            for c in range(w):
                data[r][c] = predict_50[0][r][c][i]
        print (data.shape)

    plt.imsave(title,  data, cmap=plt.cm.gray)
    plt.imshow(data, cmap=plt.cm.gray)
    model_50 = getModel3(output_dim,1)#1
    model_50.compile(loss = "categorical_crossentropy", optimizer = "sgd",
                     metrics = ["acc"])
    #model_50.summary()
    predict_50 = model_50.predict(lst)
    print (predict_50.shape)
    w, h = 224, 224#224, 224 #14, 14
    data = np.zeros((h, w, 1), dtype=np.uint8)
    for i in range(1):
        title='layer1'+'.png'
        data=np.empty([h, w])
        for r in range(h):
            for c in range(w):
                data[r][c] = predict_50[0][r][c][i]
        print (data.shape)

    plt.imsave(title,  data, cmap=plt.cm.gray)
```