EE588 Advanced Image Processing
Project #5 Convolutional Neural Networks
(due Friday, Dec. 7, 2018, 5:00:00 pm)

Matlab commands are specified in the text and in `blue Courier font` and python commands in parenthesis and in `red Courier font`; variables and filenames (or commands that are consistent between Matlab and python) are specified in black Courier font. In the following python syntax, I have used `import matplotlib.pyplot as plt` and `import theano`; additionally, you probably want to use `%matplotlib inline` to include figures inline in your jupyter notebook. You will also need the `keras-vis` package installed (`pip install keras-vis`) and to `import vis`.

Your code should be well commented, and your submission should include enough narration to lead me through your solution. You should be briefly discussing your decisions, your approach to coding, and your results as you progress through the code. Generally this will mean, for each sub-instruction, a brief narrative, your commented code, and the requested output (printed or images displayed). If it is easier, you may upload a separate pdf document for each part (please name each file according to the part so that I can easily navigate—e.g., part_a.pdf). Please also include the last page of this assignment as a cover page to assist me in grading. You may fill that out and upload it as a separate document, or include it as the first page of your merged pdf.

In this project we will use the deep learning toolbox (keras deep learning library) for this project.

- Matlab users: You will need the R2017a or later version of Matlab in order to have the deep learning toolbox. You can download Matlab from https://software.nmsu.edu.
- python users: You will need to install keras (https://keras.io/) along with a backend. I installed theano (http://deeplearning.net/software/theano/install.html#install) as my backend using just conda with no issues. Keras can also use tensorflow (https://www.tensorflow.org/install/) as the backend, but tensorflow can be more finicky to install and get working, particularly on Windows machines. Your use of keras **should** be independent of the backend and your OS, but fair word of warning that I will be using theano on ubuntu as my backend to test these instructions.

(a) **MNIST Tutorial**

Complete the MNIST tutorial:

- Matlab users: https://www.mathworks.com/examples/computer-vision/community/36153-deep-learning-for-classification-on-the-mnist-dataset. Note that you will need to download and save the `prepareData.m` function from https://www.mathworks.com/matlabcentral/fileexchange/67074-code-examples-from-deep-learning-ebook?focused=c010192c-0d97-eaf9-3d23-18c51eb2eed7&tab=example before you will be able to successfully run this tutorial. You can skip the step that begins `load MNISTmodel`.
- python users: https://elitedatascience.com/keras-tutorial-deep-learning-in-python. You may need to specify `data_format='channels_first'` when specifying your first convolutional layer. Print the accuracy of the trained model applied to the test data (the last step in the tutorial). As a note, `print(model.metrics_names)` will display the names of the metrics returned by `model.evaluate`.

Note—depending on your hardware, the training step of this tutorial may take a while to complete. If you find that the training stage will take too long based on how many seconds each iteration/epoch takes, you may decrease the number of epochs for training at the expense of a lower accuracy on your test data. Include tutorial output in your submitted report.

Compute a display the confusion matrix associated with the final trained network. I am leaving you on your own to figure out the way to do this. There are likely several ways. Google is your friend.

(b) **Visualize Tutorial Architecture Trained on MNIST**

There are various ways to visualize a CNN architecture and its response to inputs. We'll explore some of them here.

(b-i) Display a basic text summary of the CNN architecture with `net.Layers` which gives the layer number, layer name, layer type, and a description (`model.summary()` which gives the layer name, layer type, output shape, and number of parameters). Print a layer-by-layer summary of the architecture with columns Layer Name, Layer Type, # Filters, Filter Size, # Parameters. Recall that the total number of parameters in a convolutional layer include the filter weights and the filter biases. Some suggestions to help you with this follow:

- Matlab users: `layer=net.Layers(i)` will return the `i`th CNN layer as a class. There are various attributes of this class that will likely be of interest, e.g., `layer.Name`, `class(layer)`, `layer.Weights`, `layer.Bias`.
- python users: `layer=model.layers[i]` will return the `i`th CNN layer as an object. There are various attributes of this object that will likely be of interest, e.g., `layer.name`, `type(layer)`, `layer.get_weights()`, `layer.count_params()`.

(b-ii) The learned filter weights for each layer can be accessed via `layer.Weights` (`layer.get_weights()[0]`). For each convolutional layer, display the learned filters in subplots of a figure window. Choose an arrangement that is approximately square, e.g., 10x10 subplots. Since each filter operates on a grayscale image (i.e., number of channels is 1), display the filters as grayscale images.

(b-iii) The responses (activations) for each filter in a layer can be computed via `act=activations(net,im,'layer')` where `net` is the CNN model you trained, `im` is the image that you want to send through the CNN, and `'layer'` is the name of the layer for which you want to compute the activations (`get_activations=theano.function([model.layers[0].input], model.layers[i].output, allow_input_downcast=True); act=get_activations(x))` where `model` is the CNN that you trained and `x` is the 1x28x28x1 tensor image that you want to send through the CNN and you will get output for the `i`th layer.). The output `act` is a multi-dimensional matrix of activations for the specified layer. For each convolutional layer, display the activations in subplots of a figure window. Choose an arrangement that is approximately square. Since each activation is a convolution applied to a grayscale image, the activation itself is a grayscale image, so display them as grayscale images. Display activations for the first test image (which should be an image of the digit 7).

(b-iv) We can compute synthetic images to maximize activations of each of the filters via `maxact=deepDreamImage(net,layer,channels,'PyramidLevels',1)` where `net` is the CNN that you trained, `layer` is the layer number you want to visualize, `channels` is a vector of indices into the filters within layer that you want to visualize, and you set `'PyramidLevels'` to `1` to avoid resizing the maximum activation images (`maxact=vis.visualization.visualize_activation(model,layer_idx,f)` where `model` is the CNN that you trained, `layer_idx` is the layer number you want to visualize, and feature_idx is the index of the filter for which you want to compute the maximum activation). The output `maxact` is a multi-dimensional matrix of maximum activations for the specified layer (layer and filter). For each convolutional layer, display the maximum activations in subplots of a figure window. Choose an arrangement that is approximately square. Since the network was designed to operate on grayscale images, the max activations themselves are grayscale images, so display them as grayscale images.

(c) **Train Tutorial Architecture on CalTech101 Dataset**

Using what you learned in completing the MNSIT tutorial, train the same architecture on the CalTech101 dataset from Project #4. Use the same dataset split (first 90% as training and last 10% as test) as you did for

Project #4. Here is a (likely non-comprehensive) list of things you may want to think about:

- The CalTech1010 images are 3 channels, not 1 like MNIST
- Some of the CalTech101 images are actually 1 channel, but you can't just mix those with 3 channel images in the tensor (multi-D arrays) specification of CNNs.
- We will not need to use the annotation files from CalTech101 at all
- The CalTech101 images are all different sizes, pixel-wise and CNNs expect a consistent size input. Since 28x28 as in MNIST is a bit small for the CalTech101 images, I would suggest using something larger and using `imresize` (`skimage.transform.resize`).
- The CalTech101 dataset has a total of 101 classes, not 10 like MNIST

Print your final accuracy and confusion matrix. Normalize the rows of the confusion matrix (divide each row by the sum of the row) and display as an image. You can find benchmark performances for CalTech101 classification accuracy at its website (http://www.vision.caltech.edu/Image_Datasets/Caltech101/). How does your accuracy compare?

(d) **VGG16 Architecture and Transfer Learning**

The VGG CNN architectures are described in the `simonyan2014.pdf` paper and are some of the common "standard" CNN architectures for image classification and which have been pre-trained on ImageNet. In this part, we will be using the VGG16 architecture. A handy list of the 1000 ImageNet classes can be found at https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a. Note that VGG16 assumes that input images are 224x224x3 pixels. The

(d-i) Matlab users: You will need to install the vgg16 support package through the Add-Ons button at the top of your Matlab command window. This will also require you to login (or create and then login) with your Mathworks account. Load (which will require a download, so best done with an internet connection) the VGG16 network pre-trained on image net as `net=vgg16` (`model=keras.applications.vgg16.VGG16(include_top=True,weights='imagenet',input_tensor=None,input_shape=None,pooling=None,classes=100)`).

(d-ii) Display a basic text summary of the VGG16 architecture with `net.Layers` (`model.summary()`). Print a layer-by-layer summary of the architecture with columns Layer Name, Layer Type, # Filters, Filter Size, # Parameters. How does this architecture compare to the MNIST tutorial architecture in terms of number of layers, types of layers used, and total number of trained parameters?

(d-iii) Display the maximum activations of each of the filters in the first convolutional layer of VGG16. How do these maximum activations compare to those you obtained with the MNIST tutorial architecture?

(d-iv) The classes specified when VGG16 was trained in ImageNet are not exactly the same classes as specified for the CalTech101 dataset. However, you can see what labels are assigned to the CalTech101 images by resizing them appropriately (to 224x224x3 pixels) and using VGG16 to predict the class. Use the instructions provided at https://www.mathworks.com/matlabcentral/fileexchange/61733-deep-learning-toolbox-model-for-vgg-16-network (https://machinelearningmastery.com/use-pre-trained-vgg-model-classify-objects-photographs/) to write code to classify the CalTech images. Classify the first image from each of the 101 CalTech101 categories and print out the actual label and the VGG16 label. How does VGG trained on ImageNet perform on CalTech101 images?

(d-v) In transfer learning, we "freeze" early layers in a network architecture that were already learned on another dataset and learn only the latter portion of it. You can freeze layers by specifying `layer.BiasLearnRateFactor=0; layer.BiasL2Factor=0; layer.WeightLearnRateFactor=0; layer.WeightL2Factor=0;` (`model.layer[i].trainable=False`) for each layer that you want to freeze. Matlab users: Note that you will need to pull off the `i`-th layer using `layer=net.Layers(i)` and then train another network with those frozen layers, i.e., you cannot just take the trained network `net` and freeze the layers from there. Freeze all the convolutional layers and learn only the fully connected (dense) layers of VGG16 for CalTech101. Note that since CalTech101 has only 101 categories, the size of your final softmax layer will need to differ. Report accuracy and display a normalized confusion matrix for VGG16 with the fully connected layers learned on CalTech101.

EE588 Advanced Image Processing
Project #5 Convolutional Neural Networks
(due Friday, Dec. 7, 2018, 5:00:00 pm)

Confidence (number 0 to 100% about your confidence in your performance on this project and optional statement about areas where you are particularly confidence or not):

Difficulty (number 0 to 100%):

Time Spent (hours):

Problem	**Points**
(a) MNIST Tutorial	/20
(b) Visualize MNIST	/25
(c) CalTech101 Training	/25
(d) Transfer learning	/30
TOTAL	/100