

ESAP lab 9 and 10

Task 1 – Try training a digit recognizer on a raspberry pi.

1. Connect mouse, keyboard, and monitor to the Raspberry PI using provided material.
2. Insert SD card and start up pi,
3. Create and enter a directory called ESAP9
4. Clone the esap 9 on github.
5. Download mnist.py from the lab 9 blackboard python3 mnist.py

Allow the CNN to train a while to get a feel for the time it takes to train on a raspberry pi compared to a GPU. Let it run for at least 3 minutes.

6. Stop the program with ctrl+c, copy and paste the network's progress information and save it to a file called progress.txt
7. This code was training a digit recognizer on your raspberry pi. This would take less than minutes to fully train on a modern GPU and approximately 8 hours to finish training on your PI.

Task 2: -- Using/Deploying a deep neural network.

Since training a neural network on a raspberry pi is typically wasteful in terms of time and power consumption, we will instead use a pre-trained model. You can find pre-trained networks for a variety of tasks online and these can be deployed to a PI.

It's important to choose a model wisely as many of them are excessively large and unwise to use on embedded or mobile electronics. For this exercise we'll be using **squeezenet** [1], a binarized network that achieves similar accuracy to larger well known models on imagenet while being about 2 orders of magnitude smaller. For example, VGG16 is 528 MB and inception V3 is 92 MB whereas squeezenet is 5 MB with similar performance. Squeezenet can be deployed on mobile phones, raspberry pi's, and other resource constrained hardware.

1. Copy deeplearning.py to a new directory called squeeze
2. Enter that directory
3. Download a pretrained version of squeezenet with weights in a format tensorflow will understand

```
wget https://github.com/avoroshilov/tf-squeezenet/raw/master/sqz\_full.mat
```

4. Download class labels:

```
wget https://github.com/avoroshilov/tf-squeezenet/raw/master/synset\_words.txt
```

5. Use Bing image search to locate 5 images of everyday objects or animals. Download these images to the same directory your script is stored in and use the script to get the class predictions and confidences for each of the 5 images you download.
6. To use deeplearning.py simply give it the name of an image file as an argument.
7. Did the network perform well or poorly? Was your accuracy the same as this network on imagenet? (top 1 accuracy is ~57% and top 5 accuracy ~80%).

8. Save the class predictions, filepath, and accuracy for these 5 images that you chose in a file called predictions.txt for your TA to view later.

Task 3: -- Evaluating a Deep neural network.

Neural networks often report accuracy information on specific datasets, but these datasets may or may not be similar enough to yours to be useful for you. In this exercise, you will create a very small dataset, annotate that dataset with class labels, and evaluate squeezenet on specific classes.

1. Open the file synset_words.txt and take a moment to browse the class labels. The class labels look like this: n12267677, beside each class label is one or more corresponding words.
2. Decide on three classes that you want to use to evaluate the network.
3. Using bing search locate and download 5 images for each class by using the words that correspond to the class label.
4. Prepare a text document in the same directory that your files are in. You may format your file in CSV format like this:

```
classlabel , filename  
classlabel , filename
```


5. Write all 15 of your image file names and class labels to the file using the format above and save it as annotations.txt
6. Make a copy of the python script from Part 2 and save it as **deeplearning2.py**
7. In the function **def main():** , note the line **imgname=sys.argv[1]**. It accepts the name of an image file as its first argument. Instead we want it to accept the annotations.txt file that we made in the previous stage and to create two python lists: a list of file names and a list of classes from this annotations file. Hint: Use the python “split” function or **csvreader**.
8. You’ll notice that the code (deeplearning.py) loads just one image and reports the predicted class and confidence (probability) for that prediction. **Modify the code so that it will provide predictions for all 15 of your images.**
9. Write code to automatically check the predictions from above against the “real” or “ground truth” labels that you read from annotations.txt
10. Change the output of the program to report the total number of times the predicted labels were not the same as the true labels.

For more information on squeezenet view: <https://arxiv.org/pdf/1602.07360.pdf>

Task 4: -- Assembling a battery powered Digit recognizer

The mnist model you are using reports an accuracy of greater than 99%. But how well does it work on real data from a PI CAM? In this subsection the graph file from the first session will be used to make a hand written digit recognizer.

1. Write digit on piece of paper.

2. Assemble python tablet with picam and NCS stick. Feel free to use your battery for a power source if you'd like to be more mobile.
3. digit_recogniser.py uses the NCS to perform inference on live frames from a camera. Before we run it there are a few things you should know.
 - a. We're using the NCS 2.0 SDK, which is the latest library for the movidius neural compute sticks because it has several advantages over 1.0, including the ability to queue multiple graphs and inputs. The 2.0 SDK was released recently and is incompatible with the 1.0 SDK. If you find code written for the 1.0 version of the library, the conversion is easy. You can follow the directions here to do it: https://movidius.github.io/ncsdk/ncapi/python_api_migration.html
4. In digit_recogniser.py, change the line "graph_filename = "mnist_inference.graph"" to graph_filename = "inference.graphs"
5. Type python3 digit_recogniser.py
6. Try to get it to recognize a written digit by pointing the camera at it. Did it work as well as you thought it would? What's going wrong?
 The problem is, your network was trained on images that are 28 by 28 with each digit centered, written with white on black with non-noisy backgrounds. Neural networks are not magic, they can only work with data that is similar to information they are trained on. To get your live digit recognizer to work you'll need to use some image processing techniques make the digits you wrote look more those in the mnist dataset.
7. See if you can get your input similar to this by converting the images to white on black, cropping or using other computer vision techniques. 
8. Now that you've made these improvements, try again. Center the camera directly over the digit you want to recognize and you should get a better result.

Task 5: -- Real Time Object Detector.

In this task a battery powered, live object recognizer is developed based on the yolo algorithm.

You will set up a battery powered real time yolo based object detector on a movidius stick with a picam and a raspberry pi for object recognition. For more information about the yolo algorithm see:

<https://pjreddie.com/darknet/yolo/>

1. On your raspberry PI, open runyolo.py
2. This script won't run as is. Read through the code and locate the parts marked with "TODO" comments. Use the mnist example as a reference.
3. When you've implemented the changes, walk around the room and see what predictions your hand held object recognizer gives you for various people and objects. At what distance does it stop working? There are a variety of toys for you to test your object recognizer on. See what happens when you change perspectives and backgrounds or when you adjust the threshold.
4. Try pointing it at a youtube video playing on a laptop or phone.

Task 6: -- Combining everything

There are a limited number of classes in yolo. Wouldn't it be nice if we could get more detailed information about what each bounding box has in it?

For this task, copy the file you made from task 6 to a new file called task7.py

1. Modify task7.py so that it automatically saves each bounding box as an image every half second. Each of these images should be stored in a subdirectory called task7_images

2. Copy the program you used in task 2 to a new file called `task7_background.py`
3. `task7_background.py` should scan the `task7_images` directory every 2 seconds and print to the console the top 5 class labels of any newly added images. This means you'll need to keep track of previously read images in a list or use some other technique to ensure you don't reclassify the same images. This program should run continuously until closed and should use tensorflow on the raspberry Pi's cpu as in task 2.
4. In different terminals start both `task7_background.py` and `task7.py`
5. Walk around the room for at least 30 seconds, allowing your pi to take images of a diverse range of things automatically and test your code. You should see bounding boxes drawn on your touchscreen and detailed classes for each object in a terminal window (with quite a bit of lag). This task would be more impressive if you could do it outside or in a space with a wide variety of objects and the battery would certainly allow this. Depending on the weather a short bonus project for additional points may be announced involving this during the lab.