**COMP5623 Coursework on Image Classification and Visualizations with Convolutional Neural Networks - ImageNet10**

***Deadline***

10am, 1 March 2021

***Submission***

Submit work electronically via Minerva. This coursework is worth 25% of the credit on the module.

***Time allocation***

An indicative duration for this coursework is a total of around 17 hours.

***Motivation***

Through this coursework, you will:

* Practice building, evaluating, and finetuning a convolutional neural network on an image dataset from development to testing;
* Gain a deeper understanding of feature maps and filters by visualizing some from a pre-trained network;

***Setup and resources***

While not required for completing this coursework, having a GPU will speed up the training process, especially for Question 1.3. If you would like to use a GPU, please refer to the module website for recommended working environments with GPUs.

Please implement the coursework using Python and PyTorch, and refer to the notebooks and exercises provided in class and tutorials.  
  
This coursework will use a subset of images from the ImageNet dataset, which contains 9000 images belonging to ten object classes. We will refer to the dataset as ImageNet10. The images have been uploaded into a Git repository:

git clone <https://github.com/MohammedAlghamdi/imagenet10.git>

You are provided with some starter code for both questions which will be described below, as well as a report template.

**QUESTION I**

One challenge of building a deep learning model is to choose an architecture that can learn the features in the dataset without being unnecessarily complex. The first part of the coursework involves building a CNN and training it on ImageNet10.

We will use a method of architecture development called “single-batch training”, in which we cumulatively build a network architecture which can overfit a single training batch. A model which overfits performs very well on training data but generalises poorly on data it has not been trained on. If the model can easily overfit a single training batch, we know its architecture is complex enough to be able to learn the features present in the training data. Then we move on to training on the complete training set and adjust for the overfitting via regularisation.

The following starter code files are provided for Question 1:

* **imagenet10.py:** defines class ImageNet10 which inherits from the PyTorch Dataset class
* **config.py:** configuration variables including directories which you can set
* **starter\_code.py:** some code which builds the training and validation datasets and loaders using the ImageNet10 class

You are welcome to copy the Python code to a Jupyter or Colab Notebook if you choose. Write the foundational code which trains (both forward and backward passes) a network given a batch of training data and computes the loss on the validation set for each epoch as well.

Start with the model architecture defined below, which flattens the input image data into a 1-D tensor, and then uses a single fully-connected layer as following:

|  |  |  |
| --- | --- | --- |
| Input channels | Output channels | Layer type |
| 3 \* 128 \* 128 | 10 | fully-connected |

We suggest starting with a learning rate of 0.001 and an Adam optimizer.

**1.1**

Using **only one batch** of the training data, and part or all of the validation data, graph the training loss and validation loss over epochs (*report graph 1.1.1*). We suggest a training batch size of at least 48 if possible.

Adjust the network by adding a combination of convolutional and fully-connected layers, ReLU, max-pool, until the graph of training and validation loss show that the model is overfitting the training batch, training as many epochs as are necessary to see this clearly *(report graph 1.1.2)*. Feel free to adjust the optimization method and any other parameters as well.

Fill in a table showing your new architecture, including max-pooling and activation functions *(report table 1.1.3).*

**1.2**

Now train the model on the complete training dataset, and use the complete validation set to determine when to stop training. Consider some combination of the choice of validation set size, or how you select which data is in the validation set (or if the selection is dynamic), in order to finetune your network. Experiment with some form of regularization such as dropout or data augmentation, as your network will likely overfit the training data.

Display the graph of training and validation loss over epochs to show how you determined the optimal number of training epochs *(report graph 1.2.1).* Describe in detail your fine-tuning process of maximizing accuracy *(report section 1.2.2).*

Generate two confusion matrices, one for the training set and one for the validation set *(report figure 1.2.3).*

**1.3**

Create a test dataset using the unlabeled test data in the */test\_set* directory of the GitHub repository and generate predictions using your final model. Save all test predictions in file named *[my\_student\_username]\_test\_preds.csv*, and in the following format where each line is *image\_name, predicted\_class\_id*:

|  |
| --- |
| [my\_student\_username]\_test\_preds.csv |
| abamijvuqu.JPEG, 3 acqwaetjwm.JPEG, 8  ... |

Include this .csv file with your submission.

**QUESTION II**

The following starter code file is provided for Question II:

* **explore.py:** template code which sets up an argument parser, loads pre-trained weights into an AlexNet model and provides function headers for your work

In this question, we will visualize the filters and feature maps of a fully-trained CNN on the full ImageNet 2012 dataset.

**2.1**

First, complete the template code *explore.py* from line 50 *(report section 2.1.2)*, to read in an image located at the path stored in args.image\_path. Use this image as input for a single forward pass through the AlexNet model. For your test images, feel free to download some non-copyrighted images from Google or the ImageNet database (<http://www.image-net.org/>) from the output class categories.

Filters are the weights of the convolutional layer kernels, learned through training. Fill in function extract\_filter(conv\_layer\_idx, model), defined in the template *(report section 2.1.3)* to extract the filters from a given layer of the model.

Feature maps are the result of the filter kernels applied to the convolutional layer input. We are interested in the output after the convolution + ReLU activation. Fill in function extract\_feature\_maps(input, model), defined in the template *(report section 2.1.4)* to extract all feature maps given an input image.

|  |
| --- |
| **Tips** |
| When extracting a PyTorch tensor object, you will want to first move the tensor to CPU (if using a GPU), detach it from the gradients, make a copy, and convert to a NumPy or Python array. You can do this as follows:  tensor\_name.cpu().detach().clone().numpy() |
| The most intuitive way to get the output from a layer is to pass the input image through each layer of the network up until the target layer. If the model has a nn.Sequential list called features, we can access layer i as follows:  output\_1 = model.features[i].forward(input)  The layer activations for the subsequent layer can then be obtained by passing in the previous layer activations:  output\_2 = model.features[i+1].forward(output\_1)  Alternately, for a more elegant solution, you may choose to save layer activations by registering a hook into your network. Look for documentation on the PyTorch model.register\_forward\_hook() function to see how to use it. |

Ensure that you can retrieve filter feature-map pairs (the feature map resulting from the corresponding filter convolved on the input to the layer), and explain how you did so *(report section 2.1.5)*.

**2.2**

Write the code to visualize the outputs of extract\_filters() and extract\_feature\_maps() defined above. Please normalize filters (min-max scaling or similar method) and display one channel at a time.

For three input images belonging to different output classes, show three pairs of filters and the corresponding feature maps, each from a different layer in AlexNet. Choose one early layer, one intermediate layer, and one towards the end of the network *(report section 2.2.1)* and explore how the filters and feature maps change with depth into the network *(report section 2.2.2)*.

**Submission details**

Please submit the following:

1. Your filled-in CW1 Report Template file, named *[my\_student\_username]\_cw1.pdf* as a **PDF file.** Please do not remove any of the existing template file.
2. *[my\_student\_username]\_test\_preds.csv* containing your model’s predictions on the test set.
3. All code written for Question 1 in a single Python (.py) file, *[my\_student\_username]\_q1.py*. Please use comments to delineate the various sections of the file. If you worked in a Jupyter Notebook (.ipynb), please be sure to export it as a .py file.
4. Adjusted template code *explore.py* saved as *[my\_student\_username]\_explore.py.*

If you zip the submission files together, please use the .zip format only.