COMP5623 Coursework on Image Classification with Convolutional Neural Networks - ImageNet10

Deadline

10am 2nd March 2020.

Submission

Submit work electronically via Minerva. This coursework is worth 20% 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:

- Use convolutional neural networks, from development to training and testing;
- Gain a deeper understanding of feature maps and filters and their evolution and function during the training process and testing;
- Explore methods of improving performance on a network.

Setup and resources

One of the following working environments is suggested: (1) Conda + Jupyter on school Linux machine; (2) Colab via any machine; or (3) custom setup on a personal laptop or machine. It is not required for completing this coursework, but having a GPU will speed up the training process. You may use one of the GPUs on the DEC-10 machines or in the cloud via Colab.

Please implement the coursework using Python and PyTorch, and refer to the "Intro to PyTorch" notebook provided in lab.

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 a Jupyter Notebook file "COMP5623 CW1 Starter.ipynb" with code which will load the data, designate labels, and provides a class `ImageNet10` which inherits from the PyTorch `Dataset` class.

Building the network

The first part of the assignment is to build a CNN and train it on ImageNet10. Start by building a CNN with the following depth and parameters:

Input channels	Output channels		Kernel size
3	16	convolution	3 x 3
16	24	convolution	4 x 4
24	32	convolution	4 x 4
*	512	fully-connected	
512	10	fully-connected	

(*) Calculate the input size, which is the flattened previous layer.

Follow each convolutional layer by ReLU and a max-pool operation with kernel size 2 x 2.

After each sequence of convolution, ReLU and max-pool, add a dropout operation with p=0.3. Dropout is a regularisation technique where layer outputs are randomly (with likelihood of p) dropped out, or ignored. This helps the neural network to avoid overfitting to noise in the training data.

Use a learning rate of 0.001, momentum 0.9, and stochastic gradient descent to train the network. Training for 10 epochs should give you an accuracy in the 40th percentile on the test set. Remember that for ten classes, random performance is 10%.

1. Experiments

Now, run two experiments testing various network architectures:

1. How does the number of layers affect the training process and test performance? Try between 2 and 5 layers.
2. Choose one more architectural element to test (filter size, max-pool kernel size, number/dimensions of fully-connected layers, etc.).

Generate and display the confusion matrices for the various experiments.

2. Filter visualisation

Now, choose the best-performing network architecture and visualise the filters from the first layer of the network. Compare the filters before training (random weight initialisation), halfway during training, and after training is completed. Normalise the filters before displaying as an image. The filters are 3D, but are easier to visualise when displayed in grayscale. You can do this using matplotlib.pyplot's colour map argument:

Commented [RS1]: plt.imshow(filter, cmap="gray")

```
plt.imshow(filter, cmap="gray")
```

3. Feature map visualisation

Feature maps are the filters applied to the layer input; i.e., they are the activations of the convolutional layers. Visualise some of the feature maps (four from each layer) from the fully-trained network, by passing images forward through the network. As with the filters, display them in grayscale.

Choose two input images of different classes and compare the resulting feature maps, commenting on how they change with the depth of the network.

Tips

When visualising 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 array. You can do this as follows:

```
tensor_name.cpu().detach().clone().numpy()
```

The most intuitive way to get layer activations is to directly access them in the network.

```
conv1_activs = model.conv_layer1.forward(input_image)
```

The layer activations for the subsequent layer can then be obtained by passing in the previous layer activations:

```
conv2_activs = model.conv_layer2.forward(conv1_activs)
```

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 `model.register_forward_hook()` function to see how to use it.

4. Improving network performance

Consider and implement at least two adjustments which you anticipate will improve the model's ability to generalise on unseen data. While you may completely change the network architecture, please do not use any additional data to train your model other than the ImageNet10 training data provided. Changes related to data loading, transforms, and the training process and parameters are also valid adjustments as long as you justify your choices and provide some form of validation.

Optional

If you would like to see how well your improved model generalises on an unseen test set, you are welcome to generate predictions on an independent test set of the same ten classes, provided in the **test_set/** directory of the Git repository.

You will need to load the unseen test set, and run your model on it to generate predictions. Create a **test_predictions.csv** file with one row for every test image, containing two columns: image_name, predicted_class (0 through 9 as used above). For example, the first two rows in your file may be:

abamijvuqu.jpg, 8
acqwaetjwm.jpg, 3

Your model's performance on the unseen test set will not affect marking and is for your benefit only. We will provide the accuracy and confusion matrix along with the coursework feedback.

Deliverable and Assessment

You should submit a report of **not more than 8 pages** (excluding Python code) in which you:

1. Present the results of the network architecture experiments and comment on these results, presenting confusion matrices (max. 25 marks).
2. Present and discuss the results of the filter visualisations at the three checkpoints (max. 15 marks).
3. Present and comment on some of feature maps extracted from the different layers of the final trained network for some selected test images (max. 15 marks).
4. Describe the two adjustments you made to your network and justify why you believe they will improve the network's ability to generalise on unseen data, as well as any experiments you have used to validate your approach (max. 20 marks).
5. Python code for all your work as an annex (max. 25 marks).

Your submission should include the following files zipped together (.zip only): one .py or .ipynb file containing your code, one PDF file for your report, and optionally, one test_predictions.csv file containing your predictions on the unseen test set.