# Topics in Artificial Intelligence (CPSC 532S): Assignment #2: *CNNs and Image Classification*

Due on Thursday, January 28, 2021 at 11:59pm PST

In this assignment you will experiment with VGG-16 one of the most common 16-layer Convolutional Neural Networks (CNNs) with approximately 138 million parameters. As part of the assignment you will learn how to instantiate a CNN network, load pre-trained weights, train your own network for classification, use the network to classify images, and how to utilize pre-trained network to extract generic image features.

This assignment is an adopted version of the Visual Recognition lab from Vicente Ordonez (link). However the original lab was written in Keras and TensorFlow, where as we are using PyTorch. Never the less the original lab maybe a good resource. You will also undoubtedly endup searching for PyTorch code samples and function use on the web. This is expected and is an important part of the learning experience with DNNs. However, if you do use any of the code from elsewhere you need to acknowledge the original source.

**Programming environment**

You will be using **PyTorch** for this assignment. You are welcome to use Colab, Microsoft Azure or Amazon AWS for this assignment. If you opt to do this assignment on your personal computer, it is your responsibility to install PyTorch and ensure it is running properly. A number of tutorials for doing this are available on-line.

**Data**

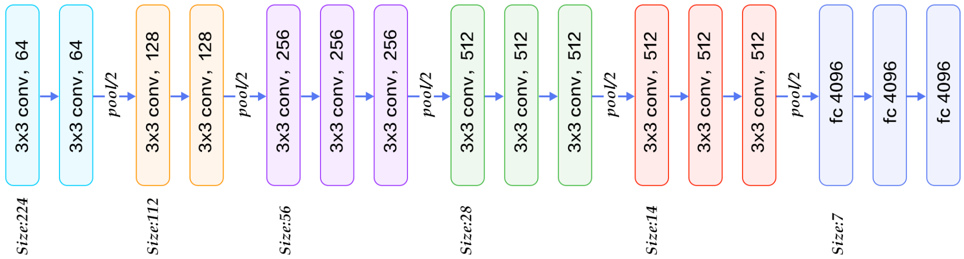
For this assignment you will use approximately 20K images from the MS-COCO dataset for training (total number of training images in MS-COCO is approximately 100K which we sub-sampled to 20K – a reasonable compromise in terms of run-time vs. realism of the assignment) and additional 100 validation images for testing the network. You will use the same data for future assignments. Each image in MS-COCO comes annotated with 5 descriptive sentences and bounding boxes (and segmentations) denoting each instance of one of the 80 object categories. For this assignment we will ignore sentences and object bounding boxes/segments. Instead we will convert annotated objects into an 80-dimensional classification vector which indicates whether a particular object class is present in the image or not. Note that in general, for most images, we will have few non-zero entries in this 80-dimensional vector.

## Assignment Instructions

You will need to follow instructions in the corresponding Jupyter notebook and submit the assignment as saved Jupiter notebook (with all the code included) when done. As part of the assignment we are providing the code to pre-process/re-size the images, to be of the required size, as well as the code that will generate 80-dimensional class label vectors from COCO annotations. The steps of the assignment are detailed below.

1. **Loading a Pre-trained Convolutional Neural Network (CNN).** We will work with the VGG-16 image classification CNN network first introduced in: Very Deep Convolutional Neural Networks

for Large-Scale Image Recognition by K. Simonyan and A. Zisserman. The network architecture is illustrated in the Figure below. The provided Jupyter notebook provides the code to instantiate this model and to pre-load the weights obtained by training the model to recognize 1,000 image categories as part of the ImageNet challenge. The illustration above is not completely accurate and the final `fc3` layer at the end of the network in the loaded model will actually contains 1,000 neurons (not 4,096).



2. **Making Predictions Using VGG-16 [10 points].** Given a pre-trained network, you will need to write the code to make predictions on the provided 100 validation images, this is equivalent of the forward pass through the network. Since the last layer of the VGG-16 (or most classification networks) is `softmax` layer, the output can be interpreted as probability of each of the 1,000 ImageNet categories being present in the image. As stated above, typically the final layer of VGG-16 is a softmax layer, however the pre-trained PyTorch model that we are using does not have softmax built into the final layer (instead opting to incorporate it into the loss function), therefore you will need to manually apply softmax to the output of the function. The code for this is provided for convenience (see notebook). For the first 10 validation images display the image itself and print (i) ground truth object classes present in the image and (ii) the top 5 predicted classes with corresponding probabilities (ordered by their probability in descending order). Depending on

3. **Computing Generic Visual Features using CNN [10 points].** It has been shown in a number of prior papers that output of fully connected layers in a CNN trained for image classification (ImageNet) can be used as effective features for other tasks. We will validate this here. First, write the code to remove the `softmax` prediction layer from the instantiated VGG-16 network with pre-trained weights. You will then be left with `fc2` layer as the final layer of the network. Now, compute the features for all the 20K training and 100 validation images, saving them into the two corresponding files. Note that for every image $i$ you should end up with $\mathbf{x}_i \in \mathbb{R}^{4,096}$ dimensional feature vector. This amounts to a forward pass using all of the data and may take some time (depending on the GPU, an hour or more).

4. **Visual Similarity [10 points].** One way to test if a given feature representation is “good” is to see if similar images end up being close in the feature space. Given the pre-computed features, for 10 images in the validation set compute the closest image in the training set (among 20K images) using Euclidian distance as the distance metric. In other words: $\arg\min_i ||\mathbf{x}_i - \hat{\mathbf{x}}||^2$, where $\hat{\mathbf{x}}$ is a feature of the validation image and $i$ is over the training images. Display the query and the nearest neighbor retrieved images.

5. **Training a Multi-label Classification Network [30 points].** ImageNet classification task inherently assumes that only one image category is predominant in an image (single label multi-class), where as MS COCO images typically contain multiple objects. We will see if we can train a classification network that is better suited for predicting the object classes in this case. We will do so by building a simple 2-layer classification network which takes 4,096-dimensional feature vectors from above and outputs probability of the 80 categories present in MS COCO. The network will have two fully connected layers with sigmoid activations (same as in Assignment 1). The hidden dimension should be set

to 512. For the loss, we will need to use multi-label loss (`MultiLabelSoftMarginLoss` in PyTorch). Once the classification network is trained, redo the prediction experiment in Part 2, *i.e.*, for the first 10 validation images display them and print (i) ground truth object classes present in the image and (ii) the top 5 predicted classes with corresponding probabilities as given by the trained classification network (in descending order). Briefly comment on the quality of the results.

6. **End-to-End Model Fine-tuning [30 points].** Above we extracted features and then trained a classification model on top of the extracted features. This is obviously sub-optimal, so here you will need to integrates the feature extractor (VGG-16), and the classification model that predicts 80 categories. Instead of training the entire architecture from scratch we will train it starting from the ImageNet weights for the VGG-16 portion of the network. The multi-label classification network on top of VGG-16 should also be initialized with weights from the previous part of the assignment. Again, once the model is trained on 20K images, redo the prediction experiment in Part 2 of the assignment, *i.e.*, for each one of the first 10 validation images display it and print (i) ground truth object classes present in the image and (ii) the top 5 predicted classes with corresponding probabilities as given by the trained classification network (in descending order). Briefly comment on the quality of the results.

   **Note**: Our preliminary experiments indicate that 1 epoch (which is defines as one iteration over all the data) of training with batch size of 50 should take approximately 30 minutes on NVIDIA 1080Ti GPU. You will need to train for a few epochs (2-3) to get reasonable performance. I would not suggest running more than 3-4 epochs for the assignment. If you have a weaker GPU with less memory, you will need to run with smaller mini-batches, which should work just fine.

7. **Hyper-parameter Tuning [10 points].** The hyper-parameters (`learning rate`, `batch size`) are likely not the best and finding good hyper-parameters is an important step in training of neural networks. Do a 3x3 grid search over the hyper-parameters, training the model with 3 different learning rates (*e.g.*, $\frac{1}{10}$ and 10× the original) and 3 different batch sizes (*e.g.*, $\frac{1}{2}$ and 2× the original). This will result in running end-to-end learning 9 times. Each time plot the value of the loss on both training and validation sets as a function of the epochs and record the value of the training loss at the end of the training. Since you are learning on the same data the training loss values should be comparable across hyper-parameter choices and can be used to choose the best model (actually, in real experiments, you would choose the model based on validation loss, not the training loss). Briefly comment on the process and which hyper-parameters you found to be optimal.

   **Note**: It is sufficient to run for 2 epochs to perform the grid search. Meaning that the experiment would likely take 30 min/epoch x 2 epochs x 9 runs = 9 hours or more depending on a specific GPU.