Instructions. In Practical 2, you will learn all about the convolutional neural networks. In particular, you will gain a first-hand experience of the training process, understand the architectural details, and familiarize with transfer learning with deep networks. You are allowed to do this assignment in pairs. If you wish, you may also do it individually.

Getting set up
In this assignment, you will work on convolutional neural network on artist classification. The starter code and the dataset for Part 1 can be found at the course website. The details about obtaining the data and the pretrained models for Part 2 are explained therein in detail.

Part 1 Convolutional Neural Networks

In this part, you will experiment with a convolutional neural network implementation to perform image classification. The dataset we will use for this assignment was created by Zoya Bylinskii, and contains 451 works of art from 11 different artists all downsampled and padded to the same size. The task is to identify which artist produced each image. The original images can be found in the art.data/artists directory included with the practical. The composition of the dataset and a sample painting from each artist are shown in Table 1.

Table 1: Artists and number of paintings in the dataset.

<table>
<thead>
<tr>
<th>Artist</th>
<th>#</th>
<th>Example</th>
<th>Artist</th>
<th>#</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canaletto</td>
<td>20</td>
<td><img src="image1.jpg" alt="Example" /></td>
<td>Paul Gauguin</td>
<td>37</td>
<td><img src="image2.jpg" alt="Example" /></td>
</tr>
<tr>
<td>Claude Monet</td>
<td>54</td>
<td><img src="image3.jpg" alt="Example" /></td>
<td>Paul Sandby</td>
<td>36</td>
<td><img src="image4.jpg" alt="Example" /></td>
</tr>
<tr>
<td>George Romney</td>
<td>63</td>
<td><img src="image5.jpg" alt="Example" /></td>
<td>Peter Paul Rubens</td>
<td>28</td>
<td><img src="image6.jpg" alt="Example" /></td>
</tr>
<tr>
<td>J. M. W. Turner</td>
<td>70</td>
<td><img src="image7.jpg" alt="Example" /></td>
<td>Rembrandt</td>
<td>40</td>
<td><img src="image8.jpg" alt="Example" /></td>
</tr>
<tr>
<td>John Robert Cozens</td>
<td>40</td>
<td><img src="image9.jpg" alt="Example" /></td>
<td>Richard Wilson</td>
<td>23</td>
<td><img src="image10.jpg" alt="Example" /></td>
</tr>
<tr>
<td>Paul Cezanne</td>
<td>40</td>
<td><img src="image11.jpg" alt="Example" /></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1Adapted from a homework developed for MIT 6.867 Machine Learning course, offered by Leslie P Kaelbling, Tomas Lozano-Perez, and Suvrit Sra.
Figure 1 shows an example of the type of convolutional architecture typically employed for similar image recognition problems. Convolutional layers apply filters to the image, and produce layers of feature maps. Often, the convolutional layers are interspersed with pooling layers. The final layers of the network are fully connected, and lead to an output layer with one node for each of the $K$ classes the network is trying to detect. We will use a similar architecture for our network.

![Convolutional Architecture Diagram](http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/)

The code for performing the data processing and training the network is provided in the starter pack. Again, you will use Tensorflow to implement convolutional neural networks.

In the code, the file `make_datasets.py` creates a dataset from the artists' images by downsampling them to 50x50 pixels, and transforming the RGB values to lie within the range $[-0.5, 0.5]$. The resulting dataset is saved in the file `art_data.pickle`. The code to train the network can be found in `conv.py`. You do not need to understand all of this code to complete the assignment, but you will need to make changes to the hyperparameters and network structure found in the function `define_tensorflow_graph`. Note that Tensorflow uses deferred computation; it sets up a computation graph, containing tensors – variables that do not necessarily store actual values until they are evaluated. The graph determines the flow of information through the network. Only once we are ready to train the model do we initialize a session and start feeding data into the graph.

### 1.1 Convolutional filter receptive field

First, it is important to develop an intuition for how a convolutional layer affects the feature representations that the network learns. Assume that you have a network in which the first convolutional layer applies a $5 \times 5$ patch to the image, producing a feature map $Z_1$. The next layer of the network is also convolutional; in this case, a $3 \times 3$ patch is applied to the feature map $Z_1$ to produce a new feature map, $Z_2$. Assume the stride used in both cases is 1. Let the receptive field of a node in this network be the portion of the original image that contributes information to the node (that it can, through the filters of the network, “see”). What are the dimensions of the receptive field for a node in $Z_2$? Note that you can ignore padding, and just consider patches in the middle of the image and $Z_1$. Thinking about your answer, why is it effective to build convolutional networks deeper (i.e. with more layers)?

### 1.2 Run the Tensorflow ConvNet

Study the provided code in the `define_tensorflow_graph` function in `conv.py`. Answer the following questions about the initial implementation:

- How many layers are there? Are they all convolutional? If not, what structure do they have?

---

2Note that this code was optimized for simplicity, not best programming practices.
• Which activation function is used on the hidden nodes?
• What loss function is being used to train the network?
• How is the loss being minimized?

Now that you are familiar with the code, try training your network! This can be accomplished by simply running the command `python conv.py`. It should take between 30-60 seconds to train your network for 1500 steps. What is the training accuracy for your network after training? What is the validation accuracy? What do these two numbers tell you about what your network is doing?

1.3 Add pooling layers

We will now max pooling layers after each of our convolutional layers. This code has already been provided for you; all you need to do is switch the `pooling` flag in the hyperparameters to `True`, and choose different values for the pooling filter size and stride. After you applied max pooling, what happened to your results? How did the training accuracy vs. validation accuracy change? What does that tell you about the effect of max pooling on your network?

1.4 Regularize your network!

Because this is such a small dataset, your network is likely to overfit the data. Implement the following ways of regularizing your network. Test each one individually, and discuss how it affects your results.

• **Dropout.** In Tensorflow, this is implemented using the `tf.nn.dropout` function, which takes a value called the `keep_prob`, representing the probability that an activation will not be dropped out. This value should be between 0.5 and 1.0 during training, and 1.0 for evaluation and testing. To change the value, you can use a placeholder variable, and feed in different values for `keep_prob` at training and testing. An example of how this works is available [here](#). Dropout has already been integrated into the `conv.py` code. To activate it, simply set the `dropout_prob` hyperparameter to some value < 1.0. You should explore different values to find one that works well.

• **Weight regularization.** As in the last question, we will add an additional term to the loss function to penalize the norm of your network’s weights. Again, this has already been implemented, and you simply need to find a good value for the `weight_penalty` hyperparameter.

• **Data augmentation.** Another effective regularization technique is to augment the data with transformed versions of the original samples. The file `augmented_art_data.pickle` contains randomly flipped and distorted versions of the original training images (note that the validation images remain unchanged). Change the `DATA_FILE` constant at the top of your `conv.py` code to point to the augmented dataset, and try running your code on this new data. The dataset is 4 times as large, so you should increase the number of training steps accordingly.

• **Early stopping.** Stop training your model after your validation accuracy starts to plateau or decrease (so that you do not overtrain your model). The number of steps can be controlled through the `num_training_steps` hyperparameter.

Discuss your results. Which regularization technique was most effective?

1.5 Experiment with your architecture

All those parameters at the top of `define_tensorflow_graph` still need to be set. You cannot possibly explore all combinations; so try to change some of them individually to get some feeling for their effect (if
any). Optionally, you can explore adding more layers. Report which changes led to the biggest increases and decreases in performance. In particular, what is the effect of making the convolutional layers have (a) a larger filter size, (b) a larger stride and (c) greater depth? How does a pyramidal-shaped network in which the feature maps gradually decrease in height and width but increase in depth (as in Figure 2) compare to a flat architecture, or one with the opposite shape?

Figure 2: Pyramidal architecture with 3 convolutional layers. Figure was adapted from https://www.udacity.com/course/deep-learning--ud730

1.6 Optimize your architecture

Based on your experience with these tests, try to achieve the best performance that you can on the validation set by varying the hyperparameters, architecture, and regularization methods. You can even (optionally) try to think of additional ways to augment the data, or experiment with techniques like local response normalization layers using tf.nn.lrn or the new weight normalization using the implementation here. Report the best performance you are able to achieve, and the settings you used to obtain it.

1.7 Test your final architecture on variations of the data

Now that you have optimized your architecture, you are ready to test it on augmented data! The file invariance_art_data.pickle contains versions of the validation images that have been distorted using the transformations shown in Figure 3. To test your code on this data, run the command python conv.py invariance. Report your performance on each of the transformed datasets. Are you surprised by any of the results? Which transformations is your network most invariant to, and which lead it to be unable to recognize the images? What does that tell you about what features your network has learned to use to recognize artists’ images?

<table>
<thead>
<tr>
<th>Original</th>
<th>Translated</th>
<th>Brightened</th>
<th>Darkened</th>
<th>High contrast</th>
<th>Low contrast</th>
<th>Flipped</th>
<th>Inverted</th>
</tr>
</thead>
</table>

Figure 3: Image transformations used in data augmentation.
Part 2  Transfer Learning with Deep Networks

In this part, you will fine-tune AlexNet model pretrained on ImageNet to recognize faces. You will work with a subset of the FaceScrub dataset. The subset of male actors is here and the subset of female actors is here. The dataset consists of URLs of images with faces, as well as the bounding boxes of the faces. The format of the bounding box is as follows (from the FaceScrub readme.txt file):

The format is x1,y1,x2,y2, where (x1,y1) is the coordinate of the top-left corner of the bounding box and (x2,y2) is that of the bottom-right corner, with (0,0) as the top-left corner of the image. Assuming the image is represented as a Python NumPy array I, a face in I can be obtained as I[y1:y2, x1:x2].

You may find it helpful to use and/or modify this script for downloading the image data. Note that you should crop out the images of the faces and resize them to appropriate size before proceeding further. Make sure to check the SHA-256 hashes, and make sure to only keep faces for which the hashes match. You should set aside 70 images per faces for the training set, and use the rest for the test and validation set.

To make your life easier, the code for running AlexNet on images are provided here. As a second alternative, you can also try Davi Frossard’s VGG-16 code/weights.

2.1 Train a multilayer perceptron

Use a fully-connected neural network with a single hidden layer. You may use this code for training a single-hidden-layer fully-connected network with TF.

In your report, include the learning curve for the test, training, and validation sets, and the final performance classification on the test set. Include a text description of your system. In particular, describe how you preprocessed the input and initialized the weights, what activation function you used, and what the exact architecture of the network that you selected was. You might get performances close to 80-85% accuracy rate.

2.2 AlexNet as a fixed feature extractor

Extract the values of the activations of AlexNet on the face images. Use those as features in order to perform face classification: learn a fully-connected neural network that takes in the activations of the units in the AlexNet layer as inputs, and outputs the name of the person. In your report, include a description of the system you built and its performance. It is recommended to start out with only using the conv4 activations. Using conv4 is sufficient here.

You should modify the AlexNet code so that the image is put into a placeholder variable.

2.3 Visualize weights

Train two networks the way you did in Part 2.1. Use 300 and 800 hidden units in the hidden layer. Visualize 2 different hidden features for each of the two settings, and briefly explain why they are interesting. A sample visualization of a hidden feature is shown in Figure 4.

Note that you probably need to use L2 regularization while training to obtain nice weight visualizations.

Adapted from a homework developed for Toronto CSC321 Introduction to Machine Learning and Neural Networks course, offered by Michael Guerzhoy.
2.4 Finetuning AlexNet

In Part 2.2, you used the (intermediate) outputs of AlexNet as inputs to a different network. You could view this as adding more layers to AlexNet (starting from e.g. conv4). Modify the handout AlexNet code so that you have an implementation of a complete network whose outputs are as in 2.2. Include the modifications that you made to AlexNet in the report. In your report, include an example of using the network (i.e., plugging in an input face image into a placeholder, and getting the label as an output.)

2.5 Visualize the gradient of the output

For a specific picture of an actor other than Gerard Butler, compute and visualize the gradient of the output of the network in Part 4 for that actor with respect to the input image. Visualize the positive parts of the gradient and the input image in two separate images. Include the code that you used for this part in your report. Use `tf.gradients`. An example with a picture of Gerard Butler is given in Figure 5. It seems that the nose eye shape and the stubble to some extent are strong cues for recognizing him.

What to turn in

You will be turning in a written report of your solutions in a single PDF file. These reports should be prepared using LaTeX using NIPS style and will be submitted via e-mail. A zip file containing the corresponding template and style files are provided at the course website. The write-up must be no more than 6 pages long. You should present your results in a clear way using well-designed tables, plots and figures. Do not include your codes.
Grading

The practical will be graded out of 100 points: 0 (no submission), 20 (an attempt at a solution), 40 (a partially correct solution), 60 (a mostly correct solution), 80 (a correct solution), 100 (a particularly creative or insightful solution). Note that the grading depends on both the content and clarity of your report.

Late Policy

You may use up to five slip days (in total) over the course of the semester for the three practicals you will take. Any additional unapproved late submission will be weighted by 0.5 and no submission after five days will be accepted.

Academic Integrity

All work on assignments must be done individually unless stated otherwise. You are encouraged to discuss with your classmates about the given practical, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in the pseudocode) will not be tolerated. In short, turning in someone else’s work, in whole or in part, as your own will be considered as a violation of academic integrity. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else.