UCLA
Dept. of Electrical and Computer Engineering
ECE 114, Fall 2019

Computer Assignment: Image Classification with Neural Networks

Introduction: Deep learning algorithms/techniques such as artificial neural networks have been widely used for image processing related applications such as content recognition and object tracking.

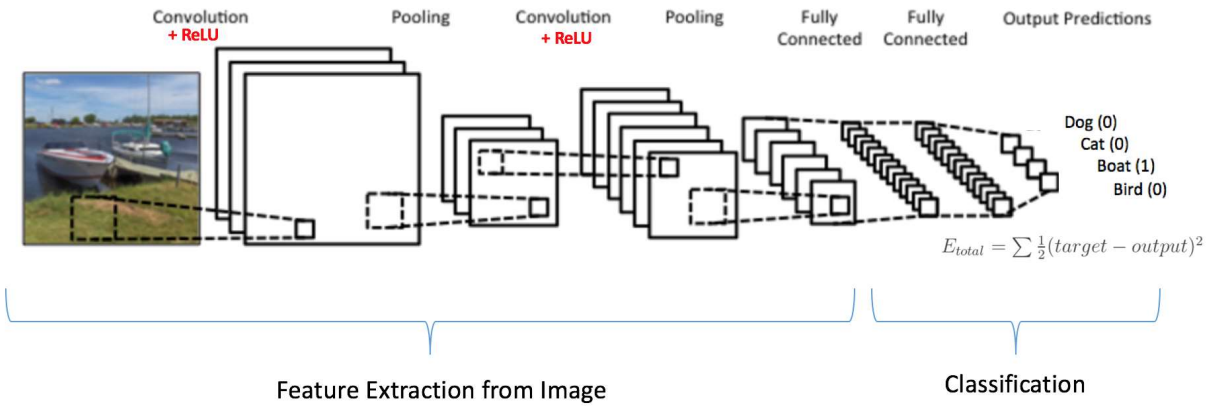
In this assignment, you will learn how to train a machine learning model to recognize different classes of images using the popular Convolutional Neural Network (CNN).

Part I. Data Preparation:

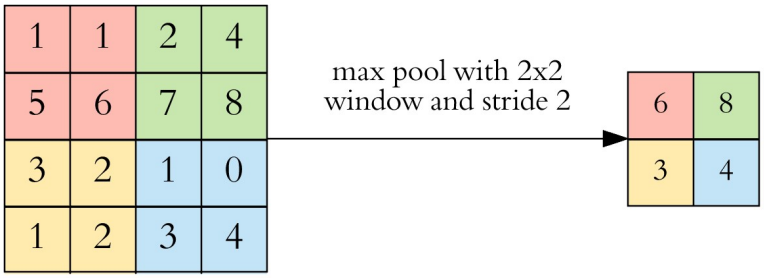
In general, the machine learning experts spend over 70% of their time on data acquisition and data processing. To save time on both, we will use the CIFAR-10 dataset (Canadian Institute for Advanced Research dataset) available through the Python tensor flow library. The CIFAR-10 dataset is a set of 60,000 32x32 images, each an example of one of ten classes: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. We will train a CNN to classify these images based on their given class.

Part II. Convolutional Neural Networks:

With the invention of GPUs (Graphics Processing Units) able to handle the computationally expensive operations performed in a convolution, CNNs became wildly popular in machine learning for image processing in the early 2000's. Innovative CNN designs such as LeNet and AlexNet broke the boundaries of what was thought possible in computer vision and sparked a movement in creating and optimizing CNNs for a variety of image classification tasks.



A CNN is a network in which the image is repeatedly convolved* with kernels and down-sampled in order to transform it into its image label. First, the convolution of the image and a kernel called a filter is performed. The values of the filter are parameters that the network must learn, often being implemented as multidimensional tensors. CNNs typically only allow "valid" convolutions as opposed to the "full" convolutions that we typically perform in signal processing. This means that the convolution is only performed over the area over which the filter and the image overlap completely. The result of the convolution is then passed through a nonlinear function and down sampled. A common way to down sample in image processing is through max pooling. In max pooling, a new image is formed from the maximum of each window in the original image (this is similar to median filtering if the median is replaced by the max). We choose the window size and stride (the amount by which we shift the window during max pooling) to achieve down sampling that preserves the distribution of pixels in the original image without including more information than necessary to make a simple classification decision.



This process is repeated several times with the size of the valid convolution shrinking on each iteration. Finally, the result of the last pooling is passed to a fully connected layer that maps the image to a C x 1 vector, V, where C is the total number of classes. V(i) is the probability decided by the network that the image belongs to the ith class. We can then use argmax(V) to decide which class is most probable for the given image. See Appendix A for more details on each layer.

*Despite the name, most CNNs use a correlation instead of a convolution for simplicity. The result is that the learned values of the filter are flipped (both up down and left right) from what they would be in a true convolution.

Assignment

You will train a Convolutional Neural Network to classify each of the test images into one of the 10 target classes using Tensorflow through the Python Keras library. Open the Image_CNN.ipybn in Google colab and run through all of the cells. You need not modify any of the code there for this assignment. **Before running, at the toolbar at the top, go to runtime>change runtime type and select GPU in Hardware Accelerator**. Training will take several hours as opposed to a few seconds if you do not do this.

Questions

1. Include a pdf print out of your completed python notebook in your report.
2. Clearly write the test accuracy of the CNN in your report. Given that there are 10 classes total, what is the probability of randomly guessing the correct class? Does your network give significantly better results than this?
3. The last cell of the Python notebook outputs a confusion matrix of the classes. A confusion matrix is a table in which the rows represent the actual class of each example and the columns represent the predicted class of each target. From this, we can see how often each class is mistaken for another class. For example, in the confusion matrix below, horse was correctly identified as horse 50 times, mistaken for cat 1 time, and mistaken for boat 2 times. Use the confusion matrix in the Python notebook to identify the classes most mistaken for each other and explain why they might be confused. Then give a detailed suggestion from the image processing techniques that you have learned or from any machine learning techniques that you have researched on how the most common confusion you see can be reduced.

Confusion Matrix			Predicted	
		horse	cat	boat
	horse	50	1	2
Actual	cat	3	47	5
	boat	7	6	43

Appendix A

Image Input Layer An imageInputLayer is where you specify the image size, which, in this case, is 28-by-28-by-1. These numbers correspond to the height, width, and the channel size. The digit data consists of grayscale images, so the channel size (color channel) is 1. For a color image, the channel size is 3, corresponding to the RGB values. You do not need to shuffle the data because trainNetwork, by default, shuffles the data at the beginning of training. trainNetwork can also automatically shuffle the data at the beginning of every epoch during training.

Convolutional Layer In the convolutional layer, the first argument is filterSize, which is the height and width of the filters the training function uses while scanning along the images. In this example, the number 3 indicates that the filter size is 3-by-3. You can specify different sizes for the height and width of the filter. The second argument is the number of filters, numFilters, which is the number of neurons that connect to the same region of the input. This parameter determines the number of feature maps. Use the 'Padding' name-value pair to add padding to the input feature map. For a convolutional layer with a default stride of 1, 'same' padding ensures that the spatial output size is the same as the input size.

Batch Normalization Layer Batch normalization layers normalize the activations and gradients propagating through a network, making network training an easier optimization problem. Use batch normalization layers between convolutional layers and nonlinearities, such as ReLU layers, to speed up network training and reduce the sensitivity to network initialization.

ReLU Layer The batch normalization layer is followed by a nonlinear activation function. The most common activation function is the rectified linear unit (ReLU).

Max Pooling Layer Convolutional layers (with activation functions) are sometimes followed by a down-sampling operation that reduces the spatial size of the feature map and removes redundant spatial information. Down-sampling makes it possible to increase the number of filters in deeper convolutional layers without increasing the required amount of computation per layer. One way of down-sampling is using a max pooling. The max pooling layer returns the maximum values of rectangular regions of inputs, specified by the first argument, poolSize. In this example, the size of the rectangular region is [2,2]. The 'Stride' name-value pair argument specifies the step size that the training function takes as it scans along the input.

Fully Connected Layer The convolutional and down-sampling layers are followed by one or more fully connected layers. As its name suggests, a fully connected layer is a layer in which the neurons connect to all the neurons in the preceding layer. This layer combines all the features learned by the previous layers across the image to identify the larger patterns. The last fully connected layer combines the features to classify the images. Therefore, the OutputSize parameter in the last fully connected layer is equal to the number of classes in the target data. In this example, the output size is 10, corresponding to the 10 classes.

Softmax Layer The softmax activation function normalizes the output of the fully connected layer. The output of the softmax layer consists of positive numbers that sum to one, which can then be used as classification probabilities by the classification layer.

Classification Layer The final layer is the classification layer. This layer uses the probabilities returned by the softmax activation function for each input to assign the input to one of the mutually exclusive classes and compute the loss.