Assignment 4

Neural Networks and Deep Learning

CSCI 5922

Fall 2017

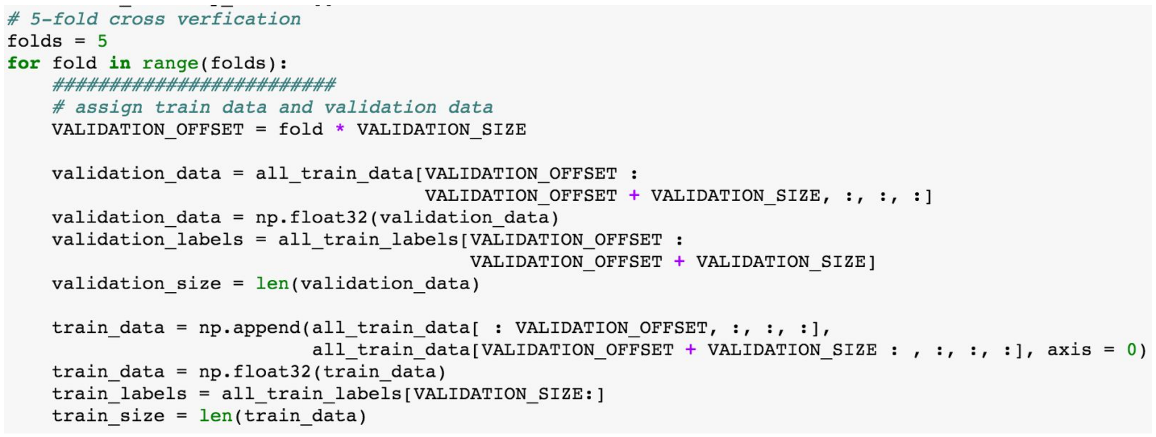
Student: **Bo Cao**

bo.cao-1@colorado.edu
boca7588@colorado.edu
Github: https://github.com/BryanBo-Cao/neuralnets-deeplearning

Part 1

(1a) Split your 50k training examples into a training set and a validation set. Decide how many examples to use for each. Describe your split into training and validation. (You may wish to do k-fold cross validation instead of just a single fold of validation, or you may want to do many-fold validation where you resample the training and validation sets each time. Simply describe the strategy you have selected.)

Me:
I split my 50k training dataset into a training set and a validation set with **4:1** proportion. I used **5-fold** cross validation to calculate



```
# 5-fold cross verfication
folds = 5
for fold in range(folds):
    #########################
    # assign train data and validation data
    VALIDATION_OFFSET = fold * VALIDATION_SIZE

    validation_data = all_train_data[VALIDATION_OFFSET :
                                     VALIDATION_OFFSET + VALIDATION_SIZE, :, :, :]
    validation_data = np.float32(validation_data)
    validation_labels = all_train_labels[VALIDATION_OFFSET :
                                         VALIDATION_OFFSET + VALIDATION_SIZE]
    validation_size = len(validation_data)

    train_data = np.append(all_train_data[ : VALIDATION_OFFSET, :, :, :],
                           all_train_data[VALIDATION_OFFSET + VALIDATION_SIZE : , :, :, :], axis = 0)
    train_data = np.float32(train_data)
    train_labels = all_train_labels[VALIDATION_SIZE:]
    train_size = len(train_data)
```

Figure 1 - Snippet of the code

Each experiment used mini-batch training with **batch size 128**.

*(1b) Use tensorflow to build a convolutional neural net for this task. You can use any of the tricks we've discussed in the course, including **pooling, max-pooling, batch normalization, data augmentation, residual networks, drop out**, etc. Of course there is a huge literature on CIFAR-10 on the web (including results from various approaches), but I ask you not to pay much attention to this literature and try to invent your own architecture from scratch. You will want to experiment with **different variations of your architecture** and the **combination of tricks** you use. I want you to report the **history of experiments** you've performed by presenting the following information for each variant:*

*Describe the **architecture in enough detail** that another member of the class could replicate the work. For example, you may vary the number of hidden layers or whether you are using dropout. But report details such as the **receptive field size** of each convolutional layer and the **stride**, and if you're using dropout, what **dropout %** you chose.*

*Report **training classification accuracy** (regardless of the loss function you use for training) and the **validation classification accuracy**.*

You will undoubtedly play with some minor variations (e.g., changing learning rates, loss function, etc.). You needn't report every such tweek. My goal is for you to convey what you believe are the most important architectural manipulations needed to get good performance on this task.

Me:
In terms of the architecture summary, the data flow is direction is from top to bottom, input layer is at the top and the output layer is on the bottom.

	Architecture Summary	Train Accuracy	Validation Accuracy
Exp1	Conv + relu + maxpooling	77.71%	62.29%
Exp2	Conv + relu + maxpooling Conv2 + relu2 + maxpooling2	72.45%	62.29%
Exp3	Conv1 + relu1 + maxpooling1 Hidden dropout 50% Conv2 + relu2 + maxpooling2	74.16%	61.41%
Exp4	Conv1 + relu1 + maxpooling1 Hidden1 dropout 50% Conv2 + relu2 + maxpooling2 Hidden2 dropout 50% Conv3 + relu3 + maxpooling3 Hidden3 dropout 50% Conv4 + relu4 + maxpooling4	70.13%	62.34%
Exp5	Batch normalization (BATCH_SIZE=128) Conv1 + relu1 + maxpooling1 Hidden dropout 50% Conv2 + relu2 + maxpooling2	73.81%	59.46%
Exp6	Data Augmentation (Image Enhance) Conv1 + relu1 + maxpooling1 Hidden dropout 50% Conv2 + relu2 + maxpooling2	74.73%	62.41%
Exp7	x->+relu2 Conv1 + relu1 + maxpooling1 Conv2 + relu2 + maxpooling2 (Residual Network)	[Not finished training]	[Not finished training]

The dataflow below is from left to right.

Exp1

Input	Conv(10x10 kernel, depth 16)
	strides=[1, 1, 1, 1],
	#[image index, y, x, depth]
		Relu	Maxpooling
			
			ksize=[1, 4, 4, 1],
			strides=[1, 2, 2, 1],
			padding='SAME'
				Fully connected to Output

Exp2

Input	Conv(10x10 kernel, depth 16)
	strides=[1, 1, 1, 1],
	#[image index, y, x, depth]
		Relu	Maxpooling
			
			ksize=[1, 4, 4, 1],
			strides=[1, 2, 2, 1],
			padding='SAME'
				(next row)

	Conv(10x10 kernel, depth 16)
	strides=[1, 1, 1, 1],
	#[image index, y, x, depth]
		Relu	Maxpooling
			
			ksize=[1, 2, 2, 1],
			strides=[1, 2, 2, 1],
			padding='SAME'
				Fully connected to Output

Exp3

Input	Conv(10x10 kernel, depth 16)
	strides=[1, 1, 1, 1],
	#[image index, y, x, depth]
		Relu	Maxpooling
			
			ksize=[1, 4, 4, 1],
			strides=[1, 2, 2, 1],
			padding='SAME'
				(next row)

	Hidden dropout 50%			(next row)

	Conv(10x10 kernel, depth 16)
	strides=[1, 1, 1, 1],
	#[image index, y, x, depth]
		Relu	Maxpooling
			
			ksize=[1, 2, 2, 1],
			strides=[1, 2, 2, 1],
			padding='SAME'
				Fully connected to Output

Exp4

Input	Conv(10x10 kernel, depth 16)	Relu	Maxpooling ksize=[1, 4, 4, 1],	(next row)

	strides=[1, 1, 1, 1],
	#[image index, y, x, depth]
			strides=[1, 2, 2, 1],
			padding='SAME'	

	Hidden dropout 50%			(next row)

	Conv(10x10 kernel, depth 16)
	strides=[1, 1, 1, 1],
	#[image index, y, x, depth]
		Relu	Maxpooling
			
			ksize=[1, 2, 2, 1],
			strides=[1, 2, 2, 1],
			padding='SAME'
				(next row)

	Hidden dropout 50%			(next row)

	Conv(10x10 kernel, depth 16)
	strides=[1, 1, 1, 1],
	#[image index, y, x, depth]
		Relu	Maxpooling
			
			ksize=[1, 1, 1, 1],
			strides=[1, 2, 2, 1],
			padding='SAME'
				(next row)

	Hidden dropout 50%			(next row)

	Conv(10x10 kernel, depth 16)
	strides=[1, 1, 1, 1],
	#[image index, y, x, depth]
		Relu	Maxpooling
			
			ksize=[1, 2, 2, 1],
			strides=[1, 2, 2, 1],
			padding='SAME'
				Fully connected to Output

Exp5

Input	Batch_normalization			(next row)

	Conv(10x10 kernel, depth 16)
	strides=[1, 1, 1, 1],
	#[image index, y, x, depth]
		Relu	Maxpooling
			
			ksize=[1, 4, 4, 1],
			strides=[1, 2, 2, 1],
			padding='SAME'
				(next row)

	Hidden dropout 50%			(next row)

	Conv(10x10 kernel, depth 16)
	strides=[1, 1, 1, 1],
	#[image index, y, x, depth]
		Relu	Maxpooling
			
			ksize=[1, 2, 2, 1],
			strides=[1, 2, 2, 1],
			padding='SAME'
				Fully connected to Output

Exp6

Input	Data Augmentation (Image Enhancement)			(next row)
	Conv(10x10 kernel, depth 16) strides=[1, 1, 1, 1], #[image index, y, x, depth]	Relu	Maxpooling ksize=[1, 4, 4, 1], strides=[1, 2, 2, 1], padding='SAME'	(next row)
	Hidden dropout 50%			(next row)
	Conv(10x10 kernel, depth 16) strides=[1, 1, 1, 1], #[image index, y, x, depth]	Relu	Maxpooling ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME'	Fully connected to Output

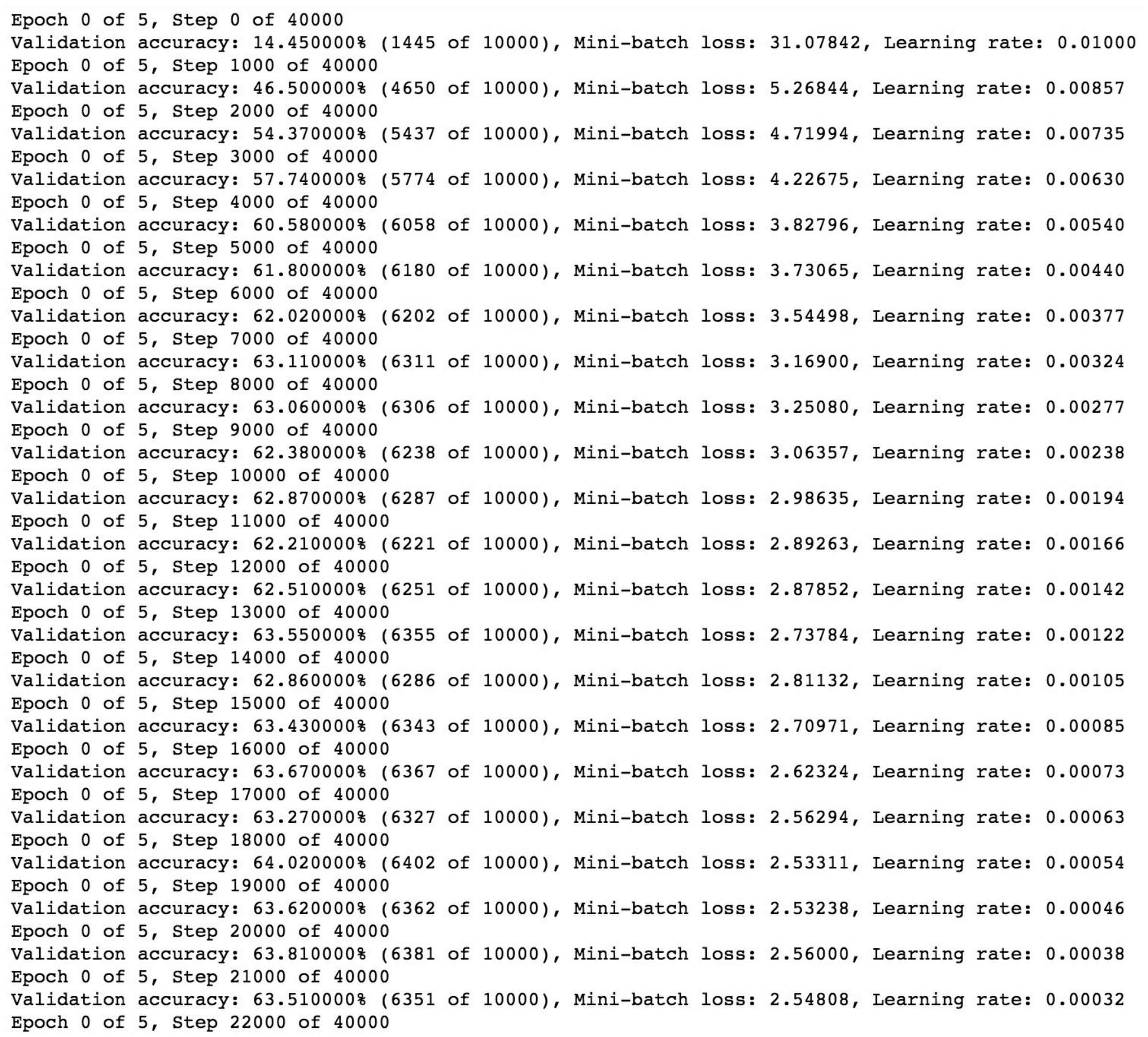
Exp7

Input	Conv(10x10 kernel, depth 16) strides=[1, 1, 1, 1], #[image index, y, x, depth]	Relu	Maxpooling ksize=[1, 4, 4, 1], strides=[1, 2, 2, 1], padding='SAME'	(next row)
	Conv(10x10 kernel, depth 16) strides=[1, 1, 1, 1], #[image index, y, x, depth]	(+Input)Relu	Maxpooling ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME'	Fully connected to Output

Each experiment experiment's code was named as "assign4_exp*.ipynb". For instance, exp1's code is named as "assign4_exp1.ipynb". Each code runs through part1 to part3.

Learning Rate

At first, I used an exponential schedule for learning rate starting at 0.01, but in the later stage the learning rate decreased to a very small number, shown in the picture below, so later I used fixed learning rates.



```
Epoch 0 of 5, Step 0 of 40000
Validation accuracy: 14.450000% (1445 of 10000), Mini-batch loss: 31.07842, Learning rate: 0.01000
Epoch 0 of 5, Step 1000 of 40000
Validation accuracy: 46.500000% (4650 of 10000), Mini-batch loss: 5.26844, Learning rate: 0.00857
Epoch 0 of 5, Step 2000 of 40000
Validation accuracy: 54.370000% (5437 of 10000), Mini-batch loss: 4.71994, Learning rate: 0.00735
Epoch 0 of 5, Step 3000 of 40000
Validation accuracy: 57.740000% (5774 of 10000), Mini-batch loss: 4.22675, Learning rate: 0.00630
Epoch 0 of 5, Step 4000 of 40000
Validation accuracy: 60.580000% (6058 of 10000), Mini-batch loss: 3.82796, Learning rate: 0.00540
Epoch 0 of 5, Step 5000 of 40000
Validation accuracy: 61.800000% (6180 of 10000), Mini-batch loss: 3.73065, Learning rate: 0.00440
Epoch 0 of 5, Step 6000 of 40000
Validation accuracy: 62.020000% (6202 of 10000), Mini-batch loss: 3.54498, Learning rate: 0.00377
Epoch 0 of 5, Step 7000 of 40000
Validation accuracy: 63.110000% (6311 of 10000), Mini-batch loss: 3.16900, Learning rate: 0.00324
Epoch 0 of 5, Step 8000 of 40000
Validation accuracy: 63.060000% (6306 of 10000), Mini-batch loss: 3.25080, Learning rate: 0.00277
Epoch 0 of 5, Step 9000 of 40000
Validation accuracy: 62.380000% (6238 of 10000), Mini-batch loss: 3.06357, Learning rate: 0.00238
Epoch 0 of 5, Step 10000 of 40000
Validation accuracy: 62.870000% (6287 of 10000), Mini-batch loss: 2.98635, Learning rate: 0.00194
Epoch 0 of 5, Step 11000 of 40000
Validation accuracy: 62.210000% (6221 of 10000), Mini-batch loss: 2.89263, Learning rate: 0.00166
Epoch 0 of 5, Step 12000 of 40000
Validation accuracy: 62.510000% (6251 of 10000), Mini-batch loss: 2.87852, Learning rate: 0.00142
Epoch 0 of 5, Step 13000 of 40000
Validation accuracy: 63.550000% (6355 of 10000), Mini-batch loss: 2.73784, Learning rate: 0.00122
Epoch 0 of 5, Step 14000 of 40000
Validation accuracy: 62.860000% (6286 of 10000), Mini-batch loss: 2.81132, Learning rate: 0.00105
Epoch 0 of 5, Step 15000 of 40000
Validation accuracy: 63.430000% (6343 of 10000), Mini-batch loss: 2.70971, Learning rate: 0.00085
Epoch 0 of 5, Step 16000 of 40000
Validation accuracy: 63.670000% (6367 of 10000), Mini-batch loss: 2.62324, Learning rate: 0.00073
Epoch 0 of 5, Step 17000 of 40000
Validation accuracy: 63.270000% (6327 of 10000), Mini-batch loss: 2.56294, Learning rate: 0.00063
Epoch 0 of 5, Step 18000 of 40000
Validation accuracy: 64.020000% (6402 of 10000), Mini-batch loss: 2.53311, Learning rate: 0.00054
Epoch 0 of 5, Step 19000 of 40000
Validation accuracy: 63.620000% (6362 of 10000), Mini-batch loss: 2.53238, Learning rate: 0.00046
Epoch 0 of 5, Step 20000 of 40000
Validation accuracy: 63.810000% (6381 of 10000), Mini-batch loss: 2.56000, Learning rate: 0.00038
Epoch 0 of 5, Step 21000 of 40000
Validation accuracy: 63.510000% (6351 of 10000), Mini-batch loss: 2.54808, Learning rate: 0.00032
Epoch 0 of 5, Step 22000 of 40000
```

Note that Train Accuracy and Validation Accuracy are calculated when training is finished. In addition, train and test data values are converted from [0, 255] to [-0.5, 0.5] at the beginning. Loss function used softmax_cross_entropy.

Part 2

(2a) For the variation that obtained the best performance in Part 1, compute error on the test set (the 10k examples you had set aside until now).

	Architecture Summary	Test Accuracy
Exp1	Conv + relu + maxpooling	66.04%

Exp2	Conv + relu + maxpooling Conv2 + relu2 + maxpooling2	63.26%
Exp3	Conv1 + relu1 + maxpooling1 Hidden dropout 50% Conv2 + relu2 + maxpooling2	64.61%
Exp4	Conv1 + relu1 + maxpooling1 Hidden1 dropout 50% Conv2 + relu2 + maxpooling2 Hidden2 dropout 50% Conv3 + relu3 + maxpooling3 Hidden3 dropout 50% Conv4 + relu4 + maxpooling4	62.24%
Exp5	Batch normalization (BATCH_SIZE=128) Conv1 + relu1 + maxpooling1 Hidden dropout 50% Conv2 + relu2 + maxpooling2	61.38%
Exp6	Data Augmentation (Image Enhance) Conv1 + relu1 + maxpooling1 Hidden dropout 50% Conv2 + relu2 + maxpooling2	60.28%
Exp7	x->+relu2 Conv1 + relu1 + maxpooling1 Conv2 + relu2 + maxpooling2 (Residual Network)	[Not finished training]

Part 3 (Extra Credit)

It might be interesting to pull some examples of images from the 80 million tiny images data set from classes other than the 10 classes you trained on, e.g., buildings or goats or mailboxes. Run these images through your trained network and see whether you can use the ***output entropy*** *(or some similar measure) to determine whether the* ***test image*** *can be* ***rejected as not belonging to any object category****. If you pick an entropy threshold, you will not only reject some of the out-of-class examples, but you will also falsely reject some in-class examples.*

(3a) Describe the ***rejection criterion*** *you use, and for that criterion, report on the* ***%*** *of out-of-class examples you've correctly* ***rejected****, but also the % of* ***in-class examples*** *(from the test set) that you* ***erroneously rejected****.*

Me:
I run through the images, calculate the softmax cross entropy, and output the prediction. I set a threshold of the probability of 0.5 first, if the highest of the prediction is less than 50% and the difference between the highest and the second highest probability of the prediction is less than 0.1, then this image is seen as out-of-class image. The reason using this is that, for a new image, if the network is not confident enough to put the image into a specific classification (<50%), in addition, if the network is confused at least two classifications, which is the second condition, then this image has a higher probability that does not belong to any of the 10 classes.

The test dataset contains 50 images, which were collected manually randomly online. 33 of the 50 images are out of the 10 classes.

	Architecture Summary	Correct Reject Rate	Incorrect Reject Rate
Exp1	Conv + relu + maxpooling	14%	6%
Exp2	Conv + relu + maxpooling Conv2 + relu2 + maxpooling2	10%	10%
Exp3	Conv1 + relu1 + maxpooling1 Hidden dropout 50% Conv2 + relu2 + maxpooling2	14%	12%
Exp4	Conv1 + relu1 + maxpooling1 Hidden1 dropout 50% Conv2 + relu2 + maxpooling2 Hidden2 dropout 50% Conv3 + relu3 + maxpooling3 Hidden3 dropout 50% Conv4 + relu4 + maxpooling4	16%	12%
Exp5	Batch normalization (BATCH_SIZE=128) Conv1 + relu1 + maxpooling1 Hidden dropout 50% Conv2 + relu2 + maxpooling2	8%	4%
Exp6	Data Augmentation (Image Enhance) Conv1 + relu1 + maxpooling1 Hidden dropout 50% Conv2 + relu2 + maxpooling2	10%	8%
Exp7	x->+relu2 Conv1 + relu1 + maxpooling1 Conv2 + relu2 + maxpooling2	[Not finished training]	[Not finished training]

	(Residual Network)		

Present objects from an untrained class.

This is a tiger image that does not belong to any of the 10 classes.