ELEC564 – Spring 2023
Homework 4

Date: March 21th, 2023

Student's Name: Hsuan-You (Shaun) Lin / Net ID: hl116

Link to Colab notebook:
https://colab.research.google.com/drive/1IB8ZYuVEJKORxvzS5uAjogRygK6KtTrs?usp=sharing

Introduction

This assignment will introduce you to PyTorch and neural networks. We have provided a Colab notebook located here with skeleton code to get you started. Colab comes with a free GPU. To activate the GPU during your session, click Runtime on the top toolbar, followed by Change runtime type, and select GPU under hardware accelerator. You will find the GPU useful for quickly training your neural network in Problem 2.

1.0 PyTorch (10 points)

In this problem, you will perform some basic operations in PyTorch, including creating tensors, moving arrays between PyTorch and Numpy, and using autograd. **Before starting, please read the following pages:**

1. Tensor basics
2. Autograd basics, these two sections:
    a. Differentiation in Autograd
    b. Vector Calculus using autograd

1.1 Basics of Autograd (5 points)

a. In the provided notebook, fill in the function sin_taylor() with code to approximate the value of the sine function using the Taylor approximation (defined here). You can use numpy.math.factorial() to help you.

b. Create a tensor x with value $\pi/4$. Create a new tensor y = sin_taylor(x). Use y.backward() to evaluate the gradient of y at x. Is this value a close approximation to the exact derivative of sine at x?

My Answer:

The output tensor I got is [0.7071], which is a very close approximation to the exact derivative of sine at $\pi/4$.

c. Now, create a NumPy array x_npy of 100 random numbers drawn uniformly from $[-\pi, \pi]$ (use np.random.uniform). Create a tensor x from that array and place the tensor onto the GPU. Again, evaluate y = sin_taylor(x). This time, y is

a vector. If you run y.backward(), it will throw an error because autograd is meant to evaluate the derivative of a scalar output with respect to input vectors (see tutorial pages above). Instead, run either one of these two lines (they do the same thing):

```
y.sum().backward()
y.backward(gradient=torch.ones(100))
```

What is happening here? We are creating a 'dummy' scalar output (let's call it z), which contains the sum of values in y, and acts as the final scalar output of our computation graph. Due to the chain rule of differentiation, dz/dx will yield the same value as dy/dx.

My Answer:

By calling backward() on z, we can obtain the gradient of y with respect to x without modifying the sin_taylor function or manually computing the gradient of y.

d. Get the gradient tensor dz/dx and convert that tensor to a Numpy array. Plot dz/dx vs. x_npy, overlaid on a cosine curve. Confirm that the points fall on the curve and put this plot in your report.

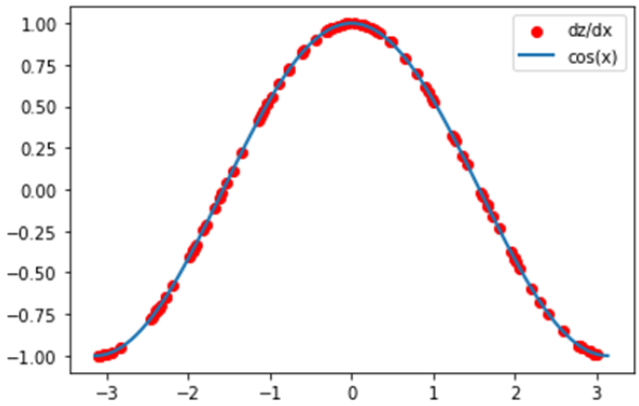


Figure 1. Plot dz/dx vs. x_npy, overlaid on a cosine curve

1.2 Image Denoising (5 points)

In this problem, you will denoise this noisy parrot image, which we denote I. To do so, you will create a denoising loss function, and use autograd to optimize the pixels of a new image J, which will be a denoised version of I.

a. In your Colab notebook, implement denoising_loss() to compute the following loss function:

$$loss = \|I - J\|_1 + \alpha \left(\left\| \frac{dJ}{dx} \right\|_1 + \left\| \frac{dJ}{dy} \right\|_1 \right)$$

The first component is a data term making sure that the predicted image J is not too far from the original image I. The second term is a regularizer which will reward J if it is smoother, quantified using J's spatial derivatives. We have provided you a function get_spatial_gradients() to compute the gradients.

b. Implement gradient descent to optimize the pixels of J using your loss function and autograd. Initialize J to be a copy of I. Try different values for the learning rate and α and find a combination that does a good job. Put the smoothed image J, along with the learning rate and α you used in your report.

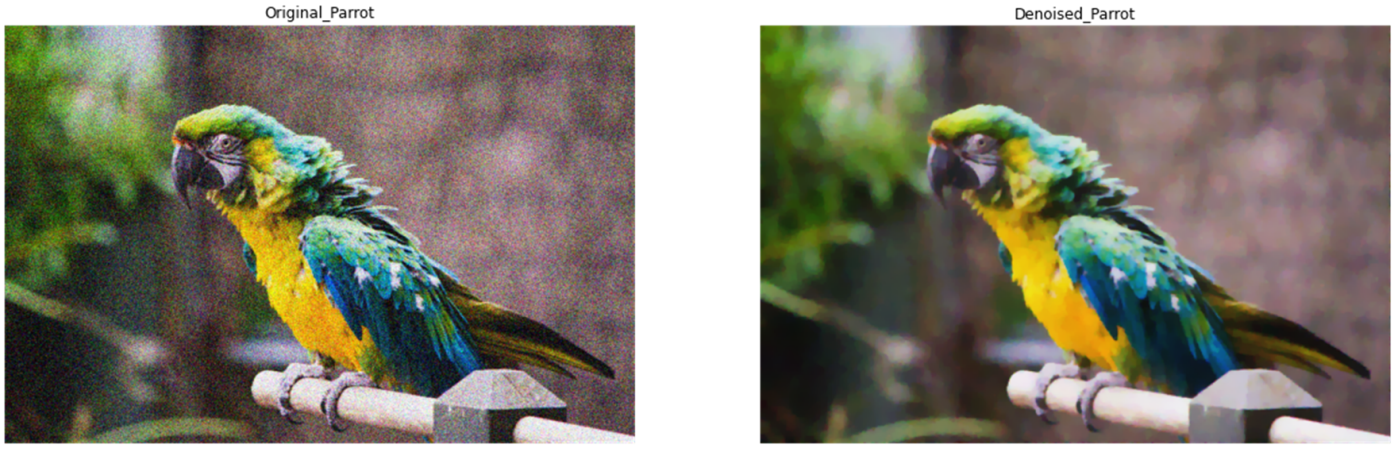


Figure 2. Original parrot image (left) and Smoothed parrot image used L1 norms (right) with (Learning rate = 0.9 / α = 1.5)

c. ELEC/COMP 546 Only: Change the loss function to use L2 norms instead of L1. Does it work better or worse? Why?

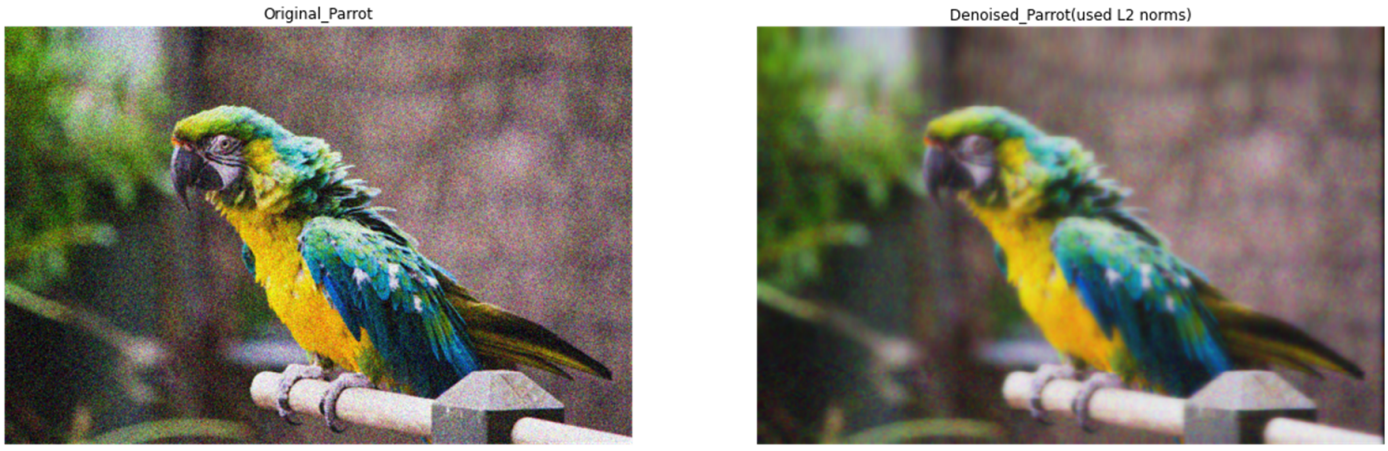


Figure 3. Original parrot image (left) and Smoothed parrot image used L2 norms (right) with (Learning rate = 0.9 / α = 1.5)

My Answer:

No, it doesn't work better, I think the L2 norm didn't improve performance because it was not well-suited to this image.

2.0 Training an image classifier (10 points)

In this problem, you will create and train your first neural network image classifier! Before starting this question, please read the following pages about training neural networks in PyTorch:

1. Data loading
2. Models
3. Training loop

We will be using the CIFAR10 dataset, consisting of 60,000 images of 10 common classes. Each image is of size 32 x 32 x 3. Download the full dataset as one .npz file here, and add it to your Google Drive. This file contains three objects: X: array of images, y: array of labels (specified as integers in [0,9]), and label_names: list of class names. Please complete the following:

a. Finish implementing the `CIFARDataset` class. See comments in the code for further instructions.

b. Add transforms: `RandomHorizontalFlip`, `RandomAffine` ([-5, 5] degree range, [0.8, 1.2] scale range) and `ColorJitter` ([0.8, 1.2] brightness range, [0.8, 1.2] saturation range). Don't forget to apply the `ToTensor` transform first, which converts a H x W x 3 image to a 3 x H x W tensor, and normalizes the pixel range to [0,1]. You will find the transform APIs in this page.

c. Implement a CNN classifier with the structure in the following table. You will find the APIs for Conv, Linear, ReLU, and MaxPool in this page. The spatial dimensions of an image should NOT change after a Conv operation (only after Maxpooling).

Layer	**Output channels for Conv/output neurons for Linear**
3 x 3 Conv + ReLU	50
MaxPool (2 x 2)	X
3 x 3 Conv + ReLU	100
MaxPool (2 x 2)	X
3 x 3 Conv + ReLU	100
MaxPool (2 x 2)	X
3 x 3 Conv + ReLU	100
Linear + ReLU	100
Linear	10

d. Implement the training loop.

e. Train your classifier for 15 epochs. The GPU, if accessible, will result in faster training. Make sure to save a model checkpoint at the end of each epoch, as you will use them in part f. Use the following training settings: batch size = 64, optimizer = Adam, learning rate = 1e-4.

f. Compute validation loss per epoch and plot it. Which model will you choose and why?

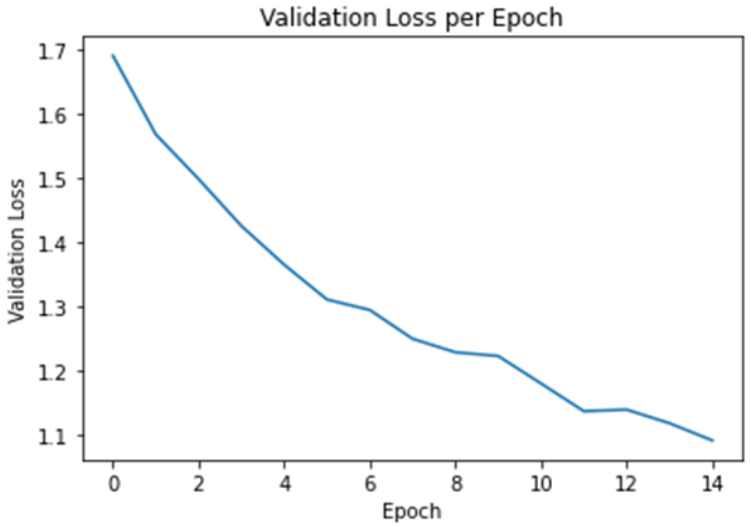


Figure 4. Validation loss per epoch

My Answer:

I chose model_14.params model as the best model. The reason for this choice is that the validation loss consistently decreased with each epoch during the training process, with the lowest loss achieved at epoch 14. This indicates that the model was able to improve its performance over time and achieved the best possible results at the end of the training process.

g. Run the best model on your test set and report:

i. Overall accuracy (# of examples correctly classified / # of examples)

```
Overall accuracy: 0.613
```

Figure 5. Overall accuracy on the best model

ii. Accuracy per class

```
Accuracy for class 0 (airplane): 0.671
Accuracy for class 1 (automobile): 0.645
Accuracy for class 2 (bird): 0.495
Accuracy for class 3 (cat): 0.342
Accuracy for class 4 (deer): 0.471
Accuracy for class 5 (dog): 0.566
Accuracy for class 6 (frog): 0.633
Accuracy for class 7 (horse): 0.725
Accuracy for class 8 (ship): 0.794
Accuracy for class 9 (truck): 0.789
```

Figure 6. Accuracy per class

iii. Confusion matrix: A 10 x 10 table, where the cell at row *i* and column *j* reports the fraction of times an example of class *i* was labeled by your model as class *j*. Please label the rows/columns by the object class name, not indices.

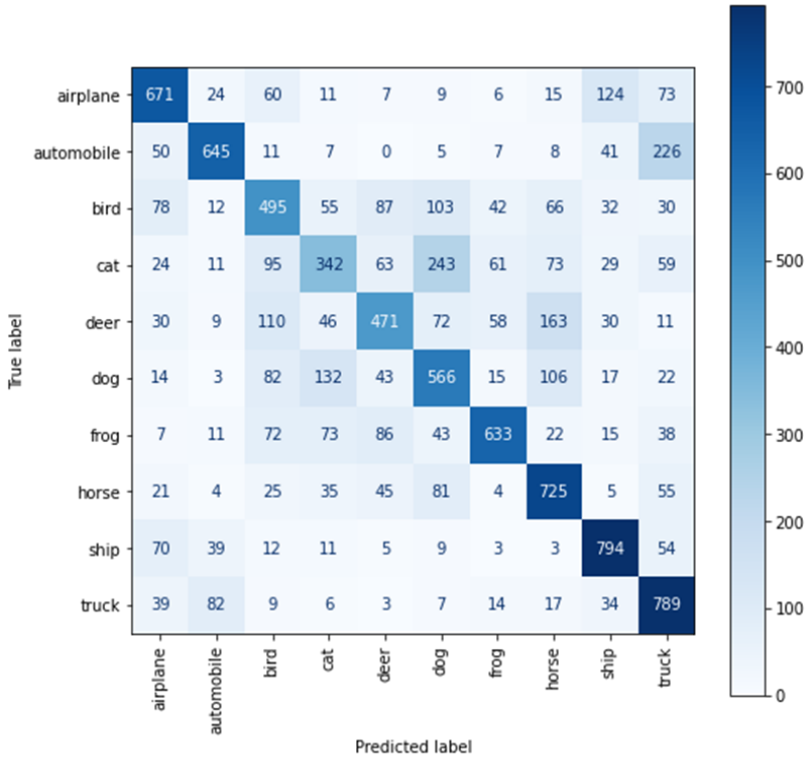


Figure 7. Confusion matrix

iv. For the class on which your model has the worst accuracy (part ii), what is the other class it is most confused with? Show 5-10 test images that your model confused between these classes and comment on what factors may have caused the poor performance.

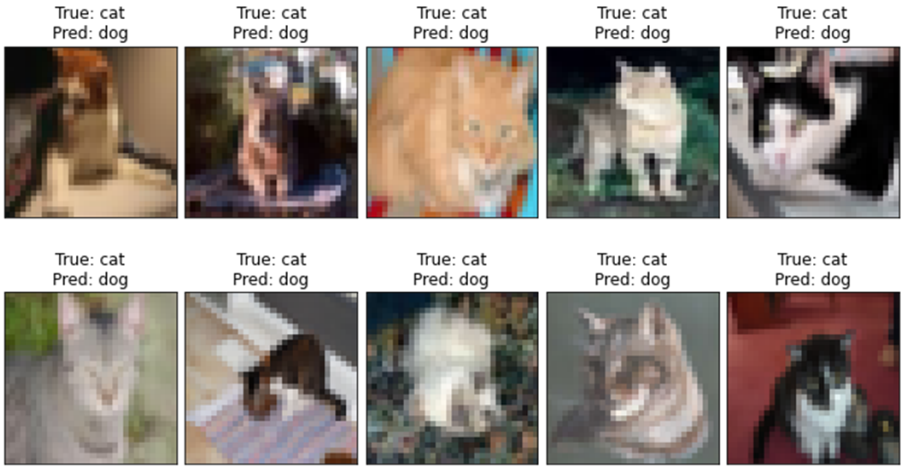
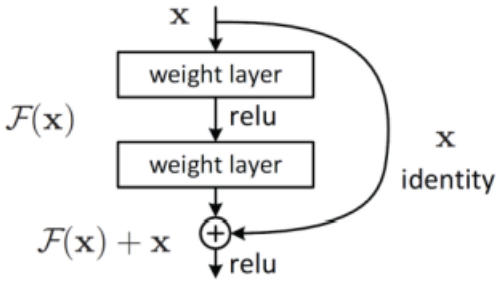


Figure 8. 10 test images that my model confused

My Answer:

Out of the 10 test images displayed above, my model seems to be most confused between the "cat" and "dog" classes. This observation is supported by the confusion matrix, which shows that 24.3% of the images belonging to the "cat" class were incorrectly classified as "dog", and 13.2% of the images belonging to the "dog" class were incorrectly classified as "cat".

h. ELEC/COMP 546 Only: Change the last two Conv blocks in the architecture to Residual blocks and report overall accuracy of the best model. Recall that a residual block has the form:



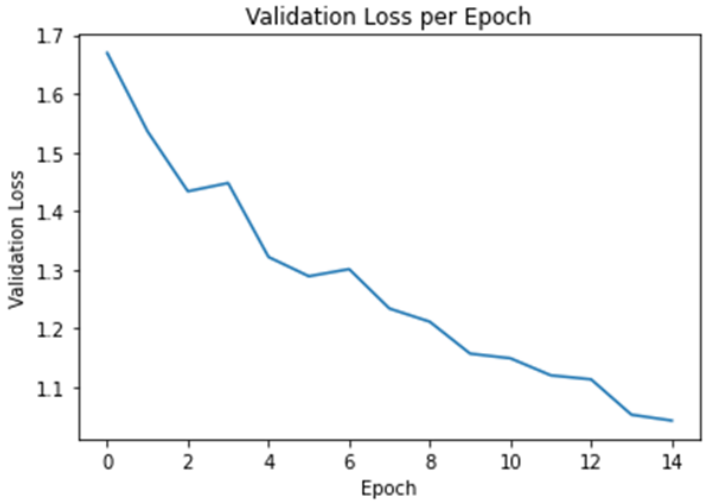


Figure 9. Validation loss per epoch

```
Overall accuracy: 0.635
```

Figure 10. Overall accuracy on the best model

My Answer:

By replacing the last two Conv blocks in the architecture with Residual blocks, the validation loss decreased consistently with each epoch during the training process, as demonstrated in Figure 9. The lowest validation loss was achieved at epoch 14, which is why I selected the residual_model_14.params model as the best model. The overall accuracy of the best model was 0.635, which is higher than the original architecture without residual blocks, where the overall accuracy was only 0.613.