DEPARTMENT OF ECE, UCLA
ECE 188: THE FOUNDATIONS OF COMPUTER VISION

INSTRUCTOR: Prof. Achuta Kadambi
TA: Rishi Upadhyay

NAME: Benjamin Cruz
UID: XXX-XXX-XXX

HOMEWORK 4

PROBLEM	TYPE	TOPIC	MAX. POINTS
1	Analytical	Machine Learning Basics	10
2	Coding	Training a Classifier	15
3	Interview Questions (Bonus)	Miscellaneous	15
4	Analytical	Generative adversarial networks	5

Motivation

The problem set gives you a basic exposure to machine learning approaches and techniques used for computer vision tasks such as image classification. You will train a simple classifier network on CIFAR-10 dataset using google colab. We have provided pytorch code for the classification question, you are free to use any other framework if that's more comfortable.

The problem set consists of two types of problems:

- analytical questions to solidify the concepts covered in the class, and
- coding questions to provide a basic exposure to building a machine learning classifier using pytorch.

This problem set also exposes you to a variety of machine learning questions commonly asked in job/internship interviews

Homework Layout

The homework consists of 3 problems in total, with subparts for each problem. There are 2 types of problems in this homework - analytical and coding. All the problems need to be answered in the Overleaf document. Make a copy of the Overleaf project from here `https://www.overleaf.com/read/pqksjrzjrckj#e991de`, and fill in your answers for the questions in the solution boxes provided.

For the analytical questions you will be directly writing their answers in the space provided below the questions. For the coding problems you need to use the Jupyter notebooks from here: https://colab.research.google.com/drive/1ciRDVvVRyIx5c48yF21BWf-ttnOtx83k?usp=sharing(see the Jupyter notebook for each sub-part which involves coding). You are provided with 1 jupyter notebook for Problem 2. After writing your code in the Jupyter notebook you need to copy paste the same code in the space provided below that question on Overleaf. In some questions you are also required to copy the saved images (from Jupyter) into the solution boxes in Overleaf. For the classification question, upload the provided notebook to google colab, and change the runtime type to GPU for training the classifier on GPU. Refer to question 2 for more instructions/details.

Submission

Submission will be done via Gradescope. You will need to submit three things: (1) this PDF with answers filled out (from Overleaf), (2) Your .ipynb file, (3) a PDF printout of your .ipynb file with all cells executed. You do not need to create a folder, you will be able to upload all three files directly to gradescope.

Software Installation

You will need Jupyter to solve the homework. You may find these links helpful:

- Jupyter (https://jupyter.org/install)
- Anaconda (https://docs.anaconda.com/anaconda/install/)

1 Machine Learning Basics (10 points)

1.1 Calculating gradients (2.0 points)

A major aspect of neural network training is identifying optimal values for all the network parameters (weights and biases). Computing gradients of the loss function w.r.t these parameters is an essential operation in this regard (gradient descent). For some parameter w (a scalar weight at some layer of the network), and for a loss function L, the weight update is given by $w := w - \alpha \frac{\partial L}{\partial w}$, where α is the learning rate/step size.

Consider (a) w, a scalar, (b) $\mathbf{x}$, a vector of size $(m \times 1)$, (c) $\mathbf{y}$, a vector of size $(n \times 1)$ and (d) $\mathbf{A}$, a matrix of size $(m \times n)$. Find the following gradients, and express them in the simplest possible form (boldface lowercase letters represent vectors, boldface uppercase letters represent matrices, plain lowercase letters represent scalars):

- $z = \mathbf{x}^T\mathbf{x}$, find $\frac{dz}{d\mathbf{x}}$
- $z = Trace(\mathbf{A}^T\mathbf{A})$, find $\frac{dz}{d\mathbf{A}}$
- $z = \mathbf{x}^T\mathbf{A}\mathbf{y}$, find $\frac{\partial z}{\partial \mathbf{y}}$
- $\mathbf{z} = \mathbf{A}\mathbf{y}$, find $\frac{d\mathbf{z}}{d\mathbf{y}}$

You may use the following formulae for reference:

$$\frac{\partial z}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial z}{\partial x_1} \\ \frac{\partial z}{\partial x_2} \\ \vdots \\ \frac{\partial z}{\partial x_m} \end{bmatrix}, \frac{\partial z}{\partial \mathbf{A}} = \begin{bmatrix} \frac{\partial z}{\partial A_{11}} & \frac{\partial z}{\partial A_{12}} & \cdots & \frac{\partial z}{\partial A_{1n}} \\ \frac{\partial z}{\partial A_{21}} & \frac{\partial z}{\partial A_{22}} & \cdots & \frac{\partial z}{\partial A_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial z}{\partial A_{m1}} & \frac{\partial z}{\partial A_{m2}} & \cdots & \frac{\partial z}{\partial A_{mn}} \end{bmatrix}, \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_n}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_m} & \frac{\partial y_2}{\partial x_m} & \cdots & \frac{\partial y_n}{\partial x_m} \end{bmatrix}$$

if $z = \mathbf{x}^T\mathbf{x}$, then $\frac{dz}{d\mathbf{x}} = 2\mathbf{x}$
if $z = Trace(\mathbf{A}^T\mathbf{A})$, then $\frac{dz}{d\mathbf{A}} = \mathbf{A}$
if $z = \mathbf{x}^T\mathbf{A}\mathbf{y}$, then $\frac{\partial z}{\partial \mathbf{y}} = \mathbf{A}^T\mathbf{x}$
if $\mathbf{z} = \mathbf{A}\mathbf{y}$, find $\frac{d\mathbf{z}}{d\mathbf{y}} = \mathbf{A}^T$

1.2 Deriving Cross entropy Loss (6.0 points)

In this problem, we derive the cross entropy loss for binary classification tasks. Let $\hat{y}$ be the output of a classifier for a given input x. y denotes the true label (0 or 1) for the input x. Since y has only 2 possible values, we can assume it follow a Bernoulli distribution w.r.t the input x. We hence wish to come up with a loss function $L(y, \hat{y})$, which we would like to minimize so that the difference between $\hat{y}$ and y reduces. A Bernoulli random variable (refresh your pre-test material) takes a value of 1 with a probability k, and 0 with a probability of $1 - k$.

(i) Write an expression for $p(y|x)$, which is the probability that the classifier produces an observation $\hat{y}$ for a given input. Your answer would be in terms of $y, \hat{y}$. Justify your answer briefly.

$p(y|x) = \hat{y}^{(y)}(1-\hat{y})^{1-y}$. We can compare to the Bernoulli RV that $k = \hat{y}$

(ii) Using (i), write an expression for $\log p(y|x)$. $\log p(y|x)$ denotes the log-likelihood, which should be maximized.

$\log p(y|x) = \log(\hat{y}^{(y)}(1-\hat{y})^{1-y})$
$\log p(y|x) = y\log(\hat{y}) + (1-y)\log(1-\hat{y})$

(iii) How do we obtain $L(y,\hat{y})$ from $\log p(y|x)$? Note that $L(y,\hat{y})$ is to be minimized

$L(y,\hat{y}) = \frac{1}{N}\Sigma_i - y_i \log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i)$ where i is the ith sample and there are N total samples.

1.3 Perfect Classifier (?) (2.0 points)

You train a classifier on a training set, achieving an impressive accuracy of 100 %. However to your disappointment, you obtain a test set accuracy of 20 %. For each suggestion below, explain why (or why not) if these suggestions may help improve the testing accuracy.

1. Use more training data
2. Add L2 regularization to your model
3. Increase your model size, i.e. increase the number of parameters in your model
4. Create a validation set by partitioning your training data. Use the model with highest accuracy on the validation set, not the training set.

1. This will help improve accuracy because increasing the amount of training data will make the model unable to overfit all the samples and is forced to generalize.
2. L2 regularization will help reduce the complexity of the model and this specific L2 penalty aims to minimize the squared magnitude of the weights.
3. This will not help since there is usually for underfitting and in this case, having a high training accuracy and a low test accuracy is overfitting. This will also increase the complexity of the model and not help.
4. Yes, this will help overall since it will allow us to choose a model with the best generalization.

2 Implementing an image classifier using PyTorch (15.0 points)

In this problem you will implement a CNN based image classifier in pytorch. We will work with the CIFAR-10 dataset. Follow the instructions in jupyter-notebook to complete the missing parts. For this part, you will use the notebook named PSET4_Classification. For training the model on colab gpus, upload the notebook on google colab, and change the runtime type to GPU.

2.1 Loading Data (2.0 points)

(i) Explain the function of transforms.Normalize() function (See the Jupyter notebook Q1 cell). How will you modify the arguments of this function for gray scale images instead of RGB images. (ii) Write the code snippets to print the number of training and test samples loaded.

Make sure that your answer is within the bounding box.

The normalize function will normalize the image in the range of [-1, 1]. For grayscale images, we can just change (0.5, 0.5, 0.5) to 0.5

```
print('The number of training samples: {}'.format(len(train_data)))
print('The number of test samples: {}'.format(len(test_data)))
```

2.2 Classifier Architecture (6.0 points)

(See the Jupyter Notebook) Please go through the supplied code that defines the architecture (cell Q2 in the Jupyter Notebook), and answer the following questions.

1. Describe the entire architecture. The description for each layer should include details about kernel sizes, number of channels, activation functions and the type of the layer.
2. What does the padding parameter control?
3. Briefly explain the max pool layer.
4. What would happen if you change the kernel size to 3 for the CNN layers without changing anything else? Are you able to pass a test input through the network and get back an output of the same size? Why/why not? If not, what would you have to change to make it work?
5. While backpropagating through this network, for which layer you don't need to compute any additional gradients? Explain Briefly Why.

1. The architecture is as follows
    1. Layer 1: Convolution layer, 3 input channels and 6 output channels, kernel size is 5
    2. Layer 2: ReLU layer, the activation function is ReLU
    3. Layer 3: Pooling layer using max pooling, kernel size is 2
    4. Layer 4: Convolution layer, 6 input channels and 16 output channels, kernel size is 5

5. Layer 5: ReLU layer, the activation function is ReLU.
6. Layer 6: Pooling layer using max pooling, kernel size is 2
7. Layer 7: Fully connected layer, input size is 16*5*5, output size is 120
8. Layer 8: ReLU layer, the activation function is ReLU.
9. Layer 9: Fully connected layer, input size is 120, output size is 84.
10. Layer 10: ReLU layer, the activation function is ReLU
11. Layer 11: Fully connected layer, input size is 84, output size is 10.

2. The padding parameter controls the additional size of zero padding.
3. The max pool layer downsamples the data
4. No, we need to change the flatten dimension before the fully connected layers and the input size of the first fully connected layer.
5. We do not need to backpropagate for ReLU. According to the definition of ReLU, the gradient of the non-negative elements are simply 1 and that of the negative elements are simply 0.

2.3 Training the network (3.0 points)

(i) (See the Jupyter notebook.) Complete the code in the jupyter notebook for training the network on a CPU, and paste the code in the notebook. Train your network for 3 epochs. Plot the running loss (in the notebook) w.r.t epochs.

```
## for reproducibility
torch.manual_seed(7)
np.random.seed(7)

## Instantiating classifier
device = torch.device("cpu")
net = Net()
net.to(device)

## Defining optimizer and loss function
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)


## Defining Training Parameters

num_epochs = 3 # 2 for CPU training, 10 for GPU training
running_loss_list = [] # list to store running loss in the code below
```

```
for epoch in range(num_epochs):  # loop over the dataset multiple times
    running_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        #================================================#
        # Fill in the training loop here.
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        #================================================#
        # print statistics
        running_loss += loss.cpu().item()
        if i % 250 == 249:    # print every 250 mini-batches
            print('[{}, {}] loss: {:.3f}'.format(epoch + 1, i + 1, running_loss / 250))
            running_loss_list.append(running_loss)
            running_loss = 0.0

print('Training Complete')
PATH = './net.pth'
torch.save(net.state_dict(), PATH)

## complete the code to plot the running loss per 250 mini batches curve

def plot_loss_curve(running_loss_list):
  ## complete code
  plt.plot(running_loss_list)
plot_loss_curve(running_loss_list)
```

(ii) (See the Jupyter notebook.) Modify your training code, to train the network on the GPU. Paste here the lines that need to be modified to train the network on google colab GPUs. Train the network for 20 epochs

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
net = Net()
net.to(device)
```

(iii) Explain why you need to reset the parameter gradients for each pass of the network

If we do not reset the gradients, they will accumulate the previous gradients but they will be useless in updating the network.

2.4 Testing the network (4.0 points)

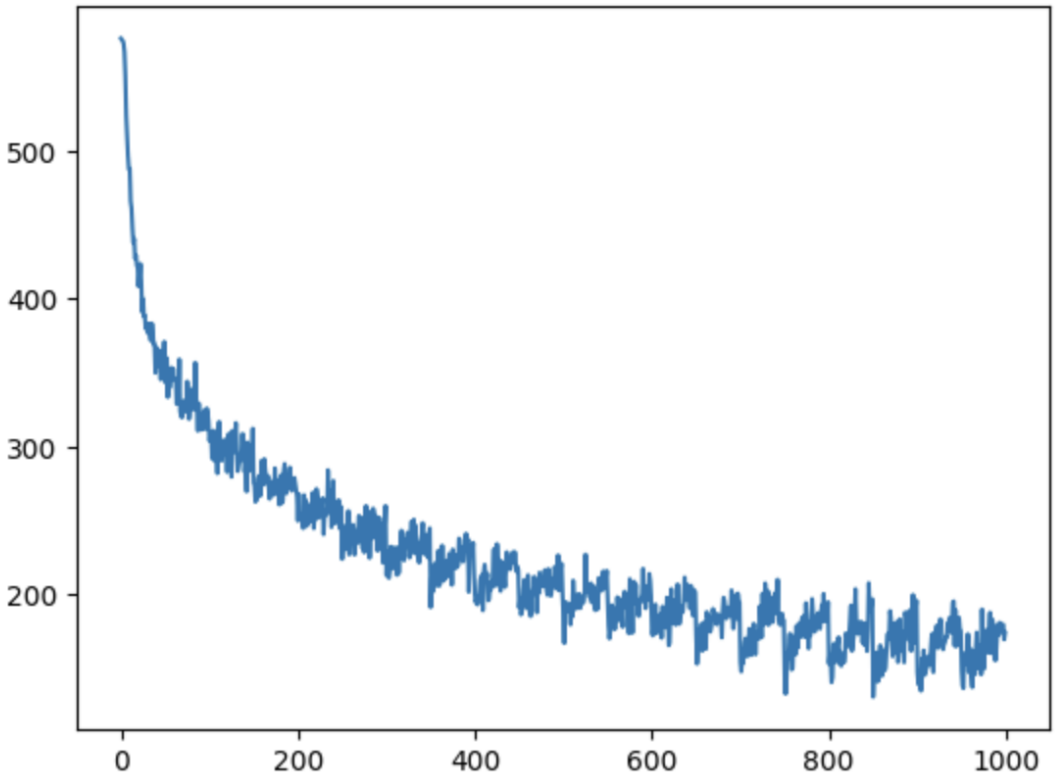
(i) (See the jupyter-notebook) Complete the code in the jupyter-notebook to test the accuracy of the network on the entire test set.

```
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
acc  = correct / total * 100## stores the accuracy computed in the above loop
print('Accuracy of the network on the 10000 test images: %d %%' % (acc))
```

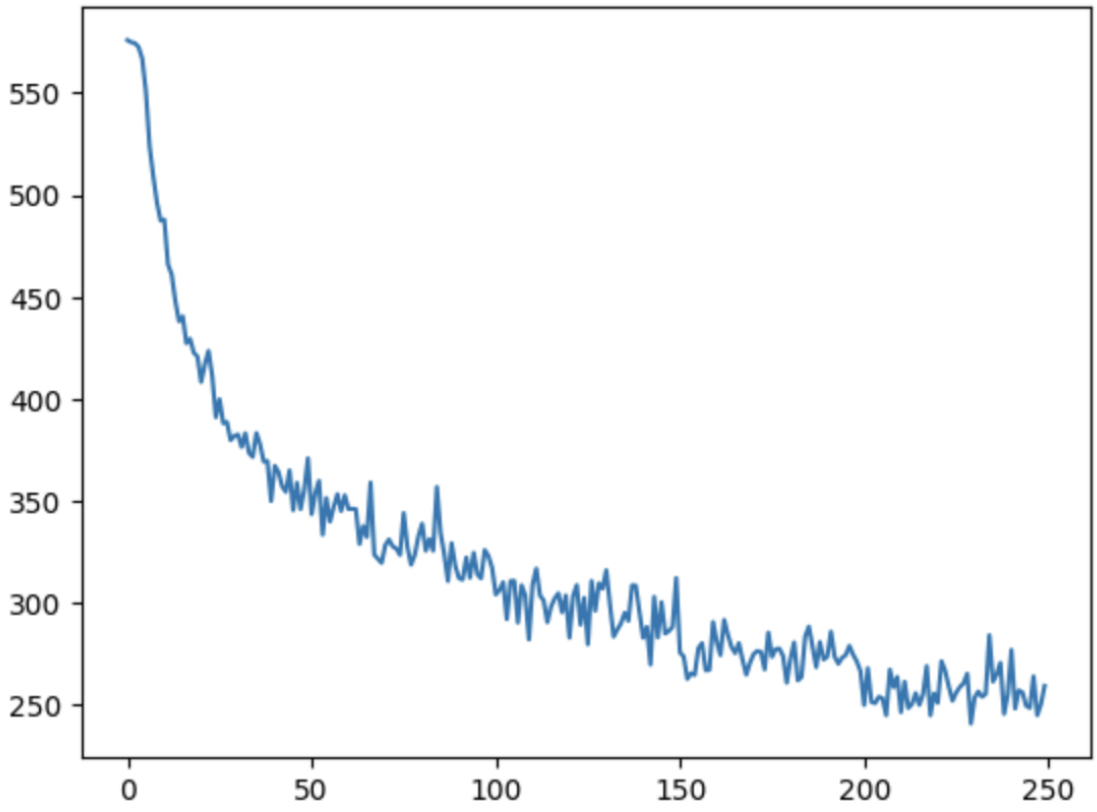
(ii) Train the network on the GPU with the following configurations, and report the testing accuracies and running loss curves -

- Training Batch Size 4, 20 training epochs
- Training Batch Size 4, 5 epochs
- Training Batch Size 16, 5 epochs
- Training Batch Size 16, 20 epochs

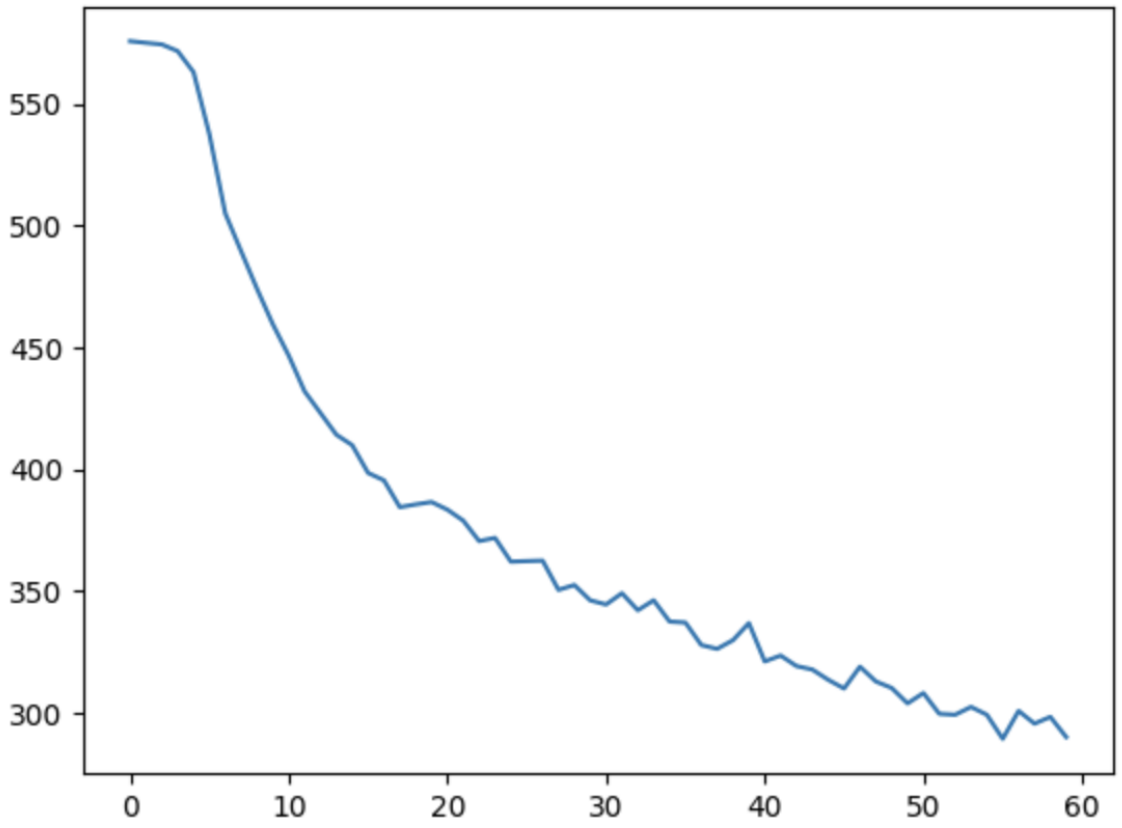
1. Training Batch Size 4, 20 epochs: accuracy 60%



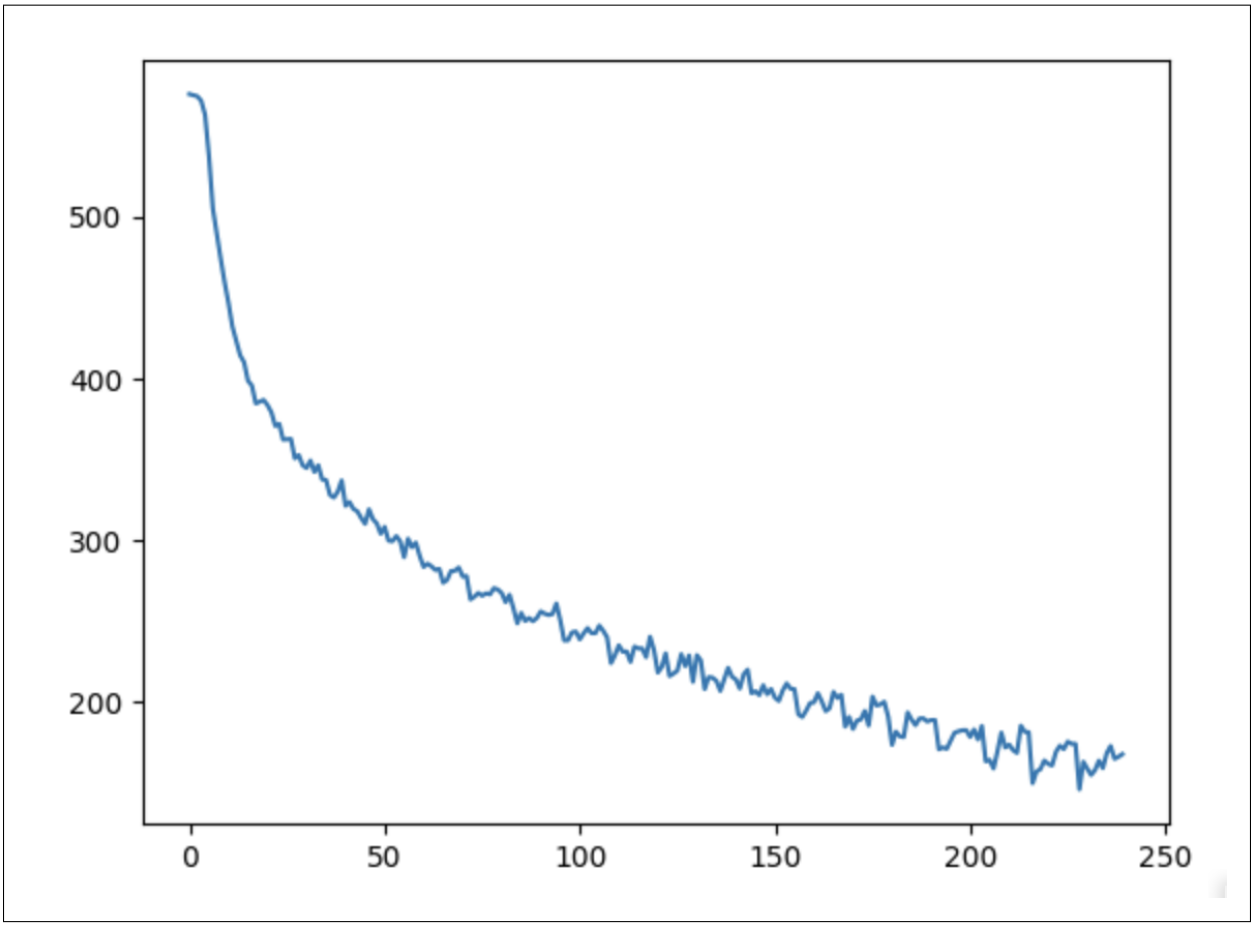
2. Training Batch Size 4, 5 epochs: accuracy 62%



3. Training Batch Size 16, 5 epochs: accuracy 57%



4. Training Batch Size 16, 20 epochs: accuracy 65%



(iii) Explain your observations in (ii)

Accuracy, batch size & epochs are not strictly positively correlated. Generally, batch size should not be too small or too large, and epochs should not be too large to avoid overfitting.

3 Interview Questions (15 points)

3.1 Batch Normalization (4 points)

Explain
(i) Why batch normalization acts as a regularizer.
(ii) Difference in using batch normalization at training vs inference (testing) time.

i. Each batch is scaled using its own mean and standard deviation so this will introduce noise and therefore giving a regularizing effect.
ii. At training time, batch normalization will see the whole batch and calculate the mean and

standard deviation and learn the parameters. During inference or testing time, this is not good since it is supposed to only see one test data point at a time. We should just use the estimated means and standard deviation which should be a good approximation.

3.2 CNN filter sizes (4 points)

Assume a convolution layer in a CNN with parameters $C_{in} = 32$, $C_{out} = 64, k = 3$. If the input to this layer has the parameters $C = 32, H = 64, W = 64$.

(i) What will be the size of the output of this layer, if there is no padding, and stride = 1
(ii) What should be the padding and stride for the output size to be $C = 64, H = 32, W = 32$

i. $H_{out} = W_{out} = \frac{64-3+2pad}{stride} + 1 = 62$ so it is $62x62x64$
ii. $32 = \frac{64-3+2pad}{stride} + 1$
$31 = \frac{61+2pad}{stride} \rightarrow pad = 16, stride = 3$

3.3 L2 regularization and Weight Decay (4 points)

Assume a loss function of the form $L(y, \hat{y})$ where y is the ground truth and $\hat{y} = f(x, w)$. x denotes the input to a neural network (or any differentiable function) $f()$ with paramters/weights denoted by w. Adding $L2$ regularization to $L(y, \hat{y})$ we get a new loss function $L'(y, \hat{y}) = L(y, \hat{y}) + \lambda w^T w$, where λ is a hyperparameter. Briefly explain why $L2$ regularization causes weight decay. Hint: Compare the gradient descent updates to w for $L(y, \hat{y})$ and $L'(y, \hat{y})$. Your answer should fit in the given solution box.

When we minimize this new loss function with the L2 term, we are also minimizing the weights at the same time. When we look at the gradient descent steps and compare them, we gain a new term $(1 - 2\alpha\lambda w)$ and this causes weight decay which is by the new factor λ. This loss function guarantees that keep the weight as small as possible while minimizing the error between the outputs and true labels.

3.4 Why CNNs? (3 points)

Give 2 reasons why using CNNs is better than using fully connected networks for image data.

CNN's are better than FC's because they have better computational tractability and they give an explicit hierarchical representation of features.

4 GAN (Bonus) (5.0 points)

4.1 Understanding GANs- Loss function (2.0 points)

Mathematically express the overall GAN loss function being used. For a (theoretically) optimally trained GAN: (a) what is the ideal behavior of the discriminator, and (b) what is the value of the overall loss function?

4.2 Understanding GANs- Gradients (2.0 points)

Assume that you are working with a GAN having the following architecture:

Generator: Input: $\mathbf{x}$ shape $(2,1)$ $\rightarrow$ Layer: $\mathbf{W_g}$ shape $(5,2)$ $\rightarrow$ ReLU $\rightarrow$ Output: $\mathbf{y}$ shape $(5,1)$
Discriminator: Input: $\mathbf{z}$ shape $(5,1)$ $\rightarrow$ Layer: $\mathbf{W_d}$ shape $(1,5)$ $\rightarrow$ Sigmoid $\rightarrow$ Output: b shape $(1,1)$

Therefore, the generator output is given by, $\mathbf{y} = ReLU(\mathbf{W_g x})$, and the discriminator output is given by $b = Sigmoid(\mathbf{W_d z})$. Express the gradient of the GAN loss function, with respect to the weight matrices for the generator and discriminator.

You may use the following information:

$$Sigmoid(x) = \frac{1}{1+e^{-x}}$$

$$ReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$$

Hint: Remember that the input here is a vector, not an image.

4.3 Understanding GANs- Input distributions (1.0 points)

While training the GAN, the input is drawn from a normal distribution. Suppose in a hypothetical setting, each time the input is chosen from a different, randomly chosen probability distribution. How would this affect the training of the GAN? Justify your answer mathematically.