Fall 2023 CS 4641/7641 A: Machine Learning Homework 4

Instructor: Dr. Mahdi Roozbahani

Deadline: Friday, December 1, 2023 11:59 pm EST

- No unapproved extension of the deadline is allowed. Submission past our 48-hour penalized acceptance period will lead to 0 credit.
- Discussion is encouraged on Ed as part of the Q/A. However, all assignments should be done individually.
- Plagiarism is a **serious offense**. You are responsible for completing your own work. You are not allowed to copy and paste, or paraphrase, or submit materials created or published by others, as if you created the materials. All materials submitted must be your own.
- All incidents of suspected dishonesty, plagiarism, or violations of the Georgia Tech Honor Code will be subject to the institute's Academic Integrity procedures. If we observe any (even small) similarities/plagiarisms detected by Gradescope or our TAs, **WE WILL DIRECTLY REPORT ALL CASES TO OSI**, which may, unfortunately, lead to a very harsh outcome. **Consequences can be severe, e.g., academic probation or dismissal, grade penalties, a 0 grade for assignments concerned, and prohibition from withdrawing from the class.**

Instructions for the assignment

- This assignment consists of both programming and theory questions.
- Unless a theory question explicitly states that no work is required to be shown, you must provide an explanation, justification, or calculation for your answer.
- To switch between cell for code and for markdown, see the menu -> Cell -> Cell Type
- You can directly type Latex equations into markdown cells.
- If a question requires a picture, you could use this syntax `<img src="" style="width: 300px;"/>` to include them within your ipython notebook.
- Your write up must be submitted in PDF form. You may use either Latex, markdown, or any word processing software. We will **NOT** accept handwritten work. Make sure that your work is formatted correctly, for example submit $\sum_{i=0} x_i$ instead of \ text{sum_{i=0} x_i}
- When submitting the non-programming part of your assignment, you must correctly map pages of your PDF to each question/subquestion to reflect where they appear.

Improperly mapped questions may not be graded correctly and/or will result in point deductions for the error.

- All assignments should be done individually, and each student must write up and submit their own answers.
- **Graduate Students**: You are required to complete any sections marked as Bonus for Undergrads

Using the autograder

- You will find three assignments (for grads) on Gradescope that correspond to HW4: "Assignment 4 Programming", "Assignment 4 - Non-programming" and "Assignment 4 Programming - Bonus for all". Undergrads will have an additional assignment called "Assignment 4 Programming - Bonus for Undergrads".
- You will submit your code for the autograder in the Assignment 4 Programming sections. Please refer to the Deliverables and Point Distribution section for what parts are considered required, bonus for undergrads, and bonus for all".
- We provided you different .py files and we added libraries in those files please DO NOT remove those lines and add your code after those lines. Note that these are the only allowed libraries that you can use for the homework
- You are allowed to make as many submissions until the deadline as you like. Additionally, note that the autograder tests each function separately, therefore it can serve as a useful tool to help you debug your code if you are not sure of what part of your implementation might have an issue
- **For the "Assignment 4 - Non-programming" part, you will need to submit to Gradescope a PDF copy of your Jupyter Notebook with the cells ran. See this EdStem Post for multiple ways on to convert your .ipynb into a .pdf file.** Please refer to the **Deliverables and Point Distribution** section for an outline of the non-programming questions.
- **When submitting to Gradescope, please make sure to mark the page(s) corresponding to each problem/sub-problem. The pages in the PDF should be of size 8.5" x 11", otherwise there may be a deduction in points for extra long sheets.**
- You **MUST** pass the Autograder Test to gain points for the programming section. There will not be any partial credit or manual grading for this part.

Using the local tests

- For some of the programming questions we have included a local test using a small toy dataset to aid in debugging. The local test sample data and outputs are stored in localtests.py
- There are no points associated with passing or failing the local tests, you must still pass the autograder to get points.
- **It is possible to fail the local test and pass the autograder** since the autograder has a certain allowed error tolerance while the local test allowed error may be smaller. Likewise, passing the local tests does not guarantee passing the autograder.
- **You do not need to pass both local and autograder tests to get points, passing the Gradescope autograder is sufficient for credit.**

- It might be helpful to comment out the tests for functions that have not been completed yet.
- It is recommended to test the functions as it gets completed instead of completing the whole class and then testing. This may help in isolating errors. Do not solely rely on the local tests, continue to test on the autograder regularly as well.

Deliverables and Points Distribution

Q1: Classification with Two Layer NN [80 pts; 55pts + 25pts Undergrad Bonus]

Deliverables: NN.py and Notebook Graphs

- **1.1 NN Implementation** [65pts; 50pts + 15pts **Bonus for Undergrad**] - *programming*
  - Leaky_relu [5pts]
  - Softmax [5pts]
  - Cross Entropy loss [5pts]
  - dropout [5pts]
  - forward propagation and with and without dropout [5pts + 5pts]
  - compute gradients and update weights [2.5pts + 2.5pts]
  - backward without momentum [5pt]
  - Gradient Descent [10pts]
  - Batch Gradient Descent [10pts **Bonus for Undergrad**]
  - Momentum [5pts **Bonus for Undergrad**]
- **1.2 Loss plot and CE for Gradient Descent** [5pts] - *non-programming*
- **1.3 Loss plot and CE for Batch Gradient Descent** [5pts **Bonus for Undergrad**] - *non-programming*
- **1.4 Loss plot and CE value for NN with Gradient Descent with Momentum** [5pts **Bonus for Undergrad**] - *non-programming*

Q2: CNN [25pts; 20pts Bonus for Undergrad + 5pts Bonus for All]

Deliverables: cnn.py and Written Report

- **2.1 Image Classification using Pytorch CNN** [20pts **Bonus for Undergrad**]
  - 2.1.1 Loading the Model [5pts **Bonus for Undergrad**] - *programming*

    - 2.1.3 Building the Model [5pts **Bonus for Undergrad**] - *non-programming*
    - 2.1.4 Training the Model [8pts **Bonus for Undergrad**] - *non-programming*
    - 2.1.5 Examining Accuracy and Loss [2pts **Bonus for Undergrad**] - *non-programming*
- **2.2 Exploring Deep CNN Architectures** [5pts **Bonus for All**] - *non-programming*

Q3: Random Forest [45pts; 40pts + 5pts Bonus for All]

Deliverables: random_forest.py and Written Report

- 3.1 Random Forest Implementation [35pts] - *programming*
- 3.2 Hyperparameter Tuning with a Random Forest [5pts] - *programming*
- 3.3 Plotting Feature Importance [5pts **Bonus for All**] - *non-programming*

Q4: SVM [34pts Bonus for all]

Deliverables: feature.py and Written Report

- 4.1: Fitting an SVM Classifier by hand [24pts] - *non programming*
- 4.2: Feature Mapping [10pts] - *programming*

Environment Setup

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
%cd "./drive/MyDrive/DL/HW4/hw4_code"
```

```
/content/drive/MyDrive/DL/HW4/hw4_code
```

```
import sys
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes, fetch_california_housing
from sklearn.preprocessing import MinMaxScaler

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import mean_squared_error

from collections import Counter
from scipy import stats
from math import log2, sqrt
import pandas as pd
```

```
import time
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier

from sklearn.datasets import make_moons
from sklearn.metrics import accuracy_score
from sklearn import svm
from NN import NeuralNet

from utilities.utils import get_housing_dataset

print('Version information')

print('python: {}'.format(sys.version))
print('matplotlib: {}'.format(matplotlib.__version__))
print('numpy: {}'.format(np.__version__))

%load_ext autoreload
%autoreload 2
%reload_ext autoreload
```

```
Version information
python: 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0]
matplotlib: 3.7.1
numpy: 1.23.5
```

1: Two Layer Neural Network [80 pts; 55pts + 25pts Undergrad Bonus] **[P][W]**

1.1 NN Implementation [65pts; 50pts + 15pts Bonus for Undergrad] **[P]**

In this section, you will implement a two layer fully connected neural network to perform a Classification Task. You will also experiment with different activation functions and optimization techniques. We provide two activation functions here - Leaky Relu and Softmax. You will implement a neural network where the first hidden layer has a Leaky Relu activation and the second hidden layer leads to a Softmax.

You'll also implement Gradient Descent (GD) and Batch Gradient Descent (BGD) algorithms for training these neural nets. **GD is mandatory for all. BGD is bonus for undergraduate students but mandatory for graduate students.**

In the NN.py file, complete the following functions:

- leaky_relu
- softmax

- cross_entropy_loss
- _dropout
- forward
- compute_gradients
- update_weights
- backward: Note Hint 2, if you still have issues passing the autograder make sure to address Hint 1 as well.
- gradient_descent
- batch_gradient_descent: **Mandatory for graduate students, bonus for undergraduate students.** Please batch your data in a wraparound manner. For example, given a dataset of 9 numbers, [1, 2, 3, 4, 5, 6, 7, 8, 9], and a batch size of 6, the first iteration batch will be [1, 2, 3, 4, 5, 6], the second iteration batch will be [7, 8, 9, 1, 2, 3], the third iteration batch will be [4, 5, 6, 7, 8, 9], etc...

We'll train this neural net on sklearn's California Housing dataset.

Activation Function

There are many activation functions that are used for various purposes. For this question, we use leaky ReLU and the softmax activation functions. We encourage you to explore the plethora of options, many of which are listed on Wikipedia.

Sigmoid

The sigmoid function is a non-linear function with an S-shaped curve and is regarded as a foundational activation function. Its output is in the range $(0, 1)$, making it the function to use for binary classification output. The function is expressed as

$$o = \phi(u) = \frac{1}{1+e^{-u}}$$

The derivation of the sigmoid function is given by

$$o' = \phi'(u) = \frac{1}{1+e^{-u}}\left(1 - \frac{1}{1+e^{-u}}\right) = o(1-o)$$

Note: We do not use sigmoid in this homework; it is only included for the sake of completeness.

sigmoid

Softmax

Softmax is a common activation function used in neural networks, especially for multiclass classification problems like the one we are tackling. It is used to convert a vector of raw outputs from the last layer of the Neural Network into a probability distribution over multiple classes. The softmax function takes as input a vector of real numbers and transforms them into a probability distribution, ensuring that the probabilities sum to 1.

Mathematically, given an input vector of [x1, x2, ..., xn], the softmax function calculates the probability p(y=i) for each class i as follows:

p(y=i) = $e^{xi} / \left(e^{x1} + e^{x2} + \ldots + e^{xn}\right)$

sigmoid

As discussed in class, the equation that we will use in this Neural network accounts for both the x values and the weights:

sigmoid

TODO: Implement the function softmax in NN.py.

```
from utilities.localtests import TestNN

TestNN("test_softmax").test_softmax()
```

```
test_softmax passed!
```

ReLU and Leaky ReLU

The rectified linear unit (ReLU) is the most commonly used activation function in deep learning today. It takes the form

$$o = \phi(u) = max(0, u).$$

Note that ReLU can be computed very quickly due to its simplicity. The derivative of ReLU is given by

$$o' = \phi'(u) = \begin{pmatrix} 0 & u \leq 0 \\ 1 & u > 0 \end{pmatrix}.$$

ReLU

Unfortunately, ReLU loses information for negative inputs; it always returns zero. For this reason, some researchers use a variant called leaky ReLU. Unlike ReLU, its leaky counterpart has a small slope (such as $\alpha = 0.05$) for negative inputs instead of a flat slope.

It takes the form

$$o = \phi(u) = \begin{pmatrix} \alpha u & u \leq 0 \\ u & u > 0 \end{pmatrix}$$

In this homework, we implement Leaky ReLU.

Leaky ReLU

TODO: Implement the function leaky_relu in NN.py.

```
from utilities.localtests import TestNN
```

```
TestNN("test_leaky_relu").test_leaky_relu()
TestNN("test_d_leaky_relu").test_d_leaky_relu()
```

```
test_leaky_relu passed!
test_d_leaky_relu passed!
```

Perceptron

A single layer perceptron can be thought of as a linear hyperplane as in logistic regression followed by a non-linear activation function.

$$u_i = \sum_{j=1}^{d} \theta_{ij} x_j + b_i$$

$$o_i = \phi\left(\sum_{j=1}^{d} \theta_{ij} x_j + b_i\right) = \phi\left(\theta_i^T x + b_i\right)$$

where x is a d-dimensional vector i.e. $x \in R^d$. It is one datapoint with d features. $\theta_i \in R^d$ is the weight vector for the i^{th} hidden unit, $b_i \in R$ is the bias element for the i^{th} hidden unit and $\phi(.)$ is a non-linear activation function that has been described below. u_i is a linear combination of the features in x_j weighted by θ_i whereas o_i is the i^{th} output unit from the activation layer.

Fully connected Layer

Typically, a modern neural network contains millions of perceptrons as the one shown in the previous image. Perceptrons interact in different configurations such as cascaded or parallel. In this part, we describe a fully connected layer configuration in a neural network which comprises multiple parallel perceptrons forming one layer.

We extend the previous notation to describe a fully connected layer. Each layer in a fully connected network has a number of input/hidden/output units cascaded in parallel. Let us a define a single layer of the neural net as follows: m denotes the number of hidden units in a single layer l whereas n denotes the number of units in the previous layer $l-1$.

$$u^{[l]} = \theta^{[l]} o^{[l-1]} + b^{[l]}$$

where $u^{[l]} \in R^m$ is a m-dimensional vector pertaining to the hidden units of the l^{th} layer of the neural network after applying linear operations. Similarly, $o^{[l-1]} \in R^n$ is the n-dimensional output vector corresponding to the hidden units of the $(l-1)^{th}$ activation layer. $\theta^{[l]} \in R^{m \times n}$ is the weight matrix of the l^{th} layer where each row of $\theta^{[l]}$ is analogous to θ_i described in the previous section i.e. each row corresponds to one hidden unit of the l^{th} layer. $b^{[l]} \in R^m$ is the bias vector of the layer where each element of b pertains to one hidden unit of the l^{th} layer. This is followed by element wise non-linear activation function $o^{[l]} = \phi\left(u^{[l]}\right)$. The whole operation can be summarized as,

$$o^{[l]} = \phi\left(\theta^{[l]} o^{[l-1]} + b^{[l]}\right)$$

where $o^{[l-1]}$ is the output of the previous layer.

Dropout

A dropout layer is a regularization technique used in neural networks to reduce overfitting. During training, a dropout layer looks at each input unit and randomly decide if it will be dropped (set to zero) with some given probability p. The decision for each unit is made independently. Formally, given an input of shape $N \times K$ (where N is the number of data points and K is the number of features), it samples from $\text{Bernoulli}(p)$ for each unit, resulting in an output where approximately pNK of the units are zero (in expectation). This forces the network to learn more robust and generalizable features, since it cannot rely too much on any particular input. During inference, the dropout layer is turned off, and the full network is used to make predictions.

The dropout probability p is a hyperparameter than can be tuned to adjust the strength of regularization. Setting $p=0$ is equivalent to no dropout.

Note that the derivative of $\text{dropout}(u)$ with respect to u has the same shape as u. The values of the derivative depend on the random mask.

Use this as a reference for your implementation.

Note that after applying the mask, we must scale the result by a factor of $1/(1-p)$. Why is this necessary?

TODO: Implement the _dropout function in NN.py.

```
from utilities.localtests import TestNN

TestNN("test_dropout").test_dropout()
```

```
test_dropout passed!
```

Cross Entropy Loss

Cross-Entropy Loss is a widely used loss function in machine learning and deep learning, especially for classification tasks. It measures the dissimilarity between the predicted probability distribution and the true probability distribution of a classification problem. If it is closer to zero, the better the learnt function is.

Implementation details

For classification problems as in this exercise, we compute the loss as follows:

\begin{align*} CE = -\frac{1}{N}\sum\limits_{i=1}^{N}\left(y_{i} \cdot log(\hat{y_{i}})\right) \end{align*}

where y_i is the true label and $\widehat{y}_i$ is the estimated label.

TODO: Implement the cross_entropy_loss function in NN.py.

```
from utilities.localtests import TestNN

TestNN("test_loss").test_loss()
```

```
test_loss passed!
```

Neural Network Architecture

The architecture of our neural network.

Neural Network

The above diagram shows the dimensions of the neural network you will implement, along with the relationships between the quantities. Note that the neural network consists of two linear layers, with a leaky ReLU activation in between. The logits outputted by the second linear layer are passed through the softmax function, which turns them into probability distributions over the 3 classes.

Initialization

We start by initializing the weights of the fully connected layer using Xavier initialization. (At a high level, we are using a uniform distribution for weight initialization). This is already implemented for you.

Forward Propagation

During training, we pass all data points through the network, layer by layer, using forward propagation. The equations for forward propagation are as follows: \begin{align} u^{[0]} &= x| u^{[1]}&= |theta^{[1]}u^{[0]}+b^{[1]} | o^{[1]}&= |text{Dropout}(|text{LeakyRelu}(u^{[1]})) | u^{[2]}&= |theta^{[2]}o^{[1]}+b^{[2]} | |hat{y}=o^{[2]}&= |text{Softmax}(u^{[2]}). |end{align}

We then use the output of the network to compute the loss \begin{align} CE = -|frac{1}{N}|sum| limits_{i=1}^{N}|left(y_{i} |cdot log(|hat{y_{i}})|right) |end{align}

TODO: Implement the forward function in NN.py.

```
from utilities.localtests import TestNN

TestNN("test_forward_without_dropout").test_forward_without_dropout()
TestNN("test_forward").test_forward()
```

```
test_forward_without_dropout passed!
test_forward passed!
```

Backward Propagation: Update Weights and Compute Gradients

After the forward pass, we do back propagation to update the weights and biases in the direction of the negative gradient of the loss function.

Update Weights

So, we update the weights and biases using the following formulas \begin{align} |theta^{[2]} := | theta^{[2]} - lr |times |frac{|partial l}{|partial |theta^{[2]}} | b^{[2]} := b^{[2]} - lr |times |frac{| partial l}{|partial b^{[2]}} | | |theta^{[1]} := |theta^{[1]} - lr |times |frac{|partial l}{|partial | theta^{[1]}} | b^{[1]} := b^{[1]} - lr |times |frac{|partial l}{|partial b^{[1]}} |end{align} where $l r$ is the learning rate. It decides the step size we want to take in the direction of the negative gradient.

TODO: Implement the update_weights function in NN.py with use_momentum=False.

```
from utilities.localtests import TestNN

TestNN("test_update_weights").test_update_weights()
```

```
test_update_weights passed!
```

Update Weights with Momentum [Bonus for Undergrad]

Gradient descent does a generally good job of facilitating the convergence of the model's parameters to minimize the loss function. However, the process of doing so can be slow and/or noisy. **Momentum** is a technique used to stabilize this convergence.

As a reminder, vanilla gradient descent applies the following update function to the parameters:

$$\theta_{t+1} = \theta_t - \alpha \nabla f(\theta_t)$$

where θ_t represents the parameters at time t, α represents the learning rate, and f is the loss function.

Momentum proposes the following tweak to our parameter update function:

$$\begin{aligned} z_{t+1} &\leftarrow \beta z_t + \nabla f(\theta_t) \\ \theta_{t+1} &\leftarrow \theta_t - \alpha z_{t+1} \end{aligned}$$

where $\beta \in [0, 1)$ is the momentum constant and z_t represents the momentum records at time t.

You can think of momentum as taking our previous changes into consideration. If we've been moving in a certain direction recently, it's likely we should keep moving in that direction. The recurrence relation given shows that we use an exponentially-weighted average of the previous updates for our current update.

A useful analogy about momentum from this great article on Distill:

> Here's a popular story about momentum: gradient descent is a man walking down a hill. He follows the steepest path downwards; his progress is slow, but steady. Momentum is a heavy ball rolling down the same hill. The added inertia acts both as a smoother and an accelerator, dampening oscillations and causing us to barrel through narrow valleys, small humps and local minima.

TODO: Implement the update_weights function in NN.py with use_momentum=True.

HINT: z is stored in `self.change`

```
from utilities.localtests import TestNN

TestNN("test_update_weights_with_momentum").test_update_weights_with_momentum()
```

```
test_update_weights_with_momentum passed!
```

Compute Gradients

In order to compute the gradients of the loss with respect to each parameter, we use the equations that make up the forward pass: \begin{align} u_1 &= \theta_1 X + b_1 \\ o_1 &= \text{leaky_relu}(u_1) \\ u_2 &= \theta_2 o_1 + b_2 \\ o_2 &= \text{softmax}(u_2) \\ l &= \text{cross_entropy}(o_2) \end{align}

When computing gradients, we travel backwards from the loss all the way back ot the input. We first seek to obtain the derivative of the loss l with respect to the logits u_2. Note that they have the relation:

$$l = \text{cross\_entropy}\left(\text{softmax}\left(u_2\right)\right)$$

Computing the derivative of this seems very involved, but it actually has a very elegant result:

$$\frac{\partial l}{\partial u_2} = \text{softmax}\left(u_2\right) - y = \hat{y} - y\,.$$

While this is given to you, we encourage you to derive it for yourself! You can find a great explanation of the derivation in this article.

Now that we have $\frac{\partial l}{\partial u_2}$, we seek to move further back and compute $\frac{\partial l}{\partial \theta_2}$ and $\frac{\partial l}{\partial b_2}$. This is done using the chain rule: \begin{align} \frac{\partial l}{\partial \theta_2} &= \frac{\partial l}{\partial u_2} \cdot \frac{\partial u_2}{\partial \theta_2} \\ \frac{\partial l}{\partial b_2} &= \frac{\partial l}{\partial u_2} \cdot \frac{\partial u_2}{\partial b_2}. \end{align}

The quantities $\frac{\partial u_2}{\partial \theta_2}$ and $\frac{\partial u_2}{\partial b_2}$ are easy to derive from the relation $u_2 = \theta_2 o_1 + b_2$. We see that \begin{align} \frac{\partial l}{\partial \theta_2} &= \frac{\partial l}{\partial u_2} \cdot o_1 \\ \frac{\partial l}{\partial b_2} &= \frac{\partial l}{\partial u_2} \cdot 1. \end{align}

Note that the derivative involves o_1, which we computed during the forward pass. Fortunately, we saved that value in `self.cache`, so we don't need to compute it again!

The same procedure is repeated to obtain the gradients for the upstream parameters θ_1 and b_1. We must first perform the intermediate steps of computing the derivative of the loss with respect to o_1 and then u_1. These are given by \begin{align} \frac{\partial l}{\partial o_1} &= \frac{\partial l}{\partial u_2} \cdot \theta_2 \\ \frac{\partial l}{\partial u_1} &= \frac{\partial l}{\partial o_1} \cdot \frac{\partial \text{leaky_relu}}{\partial u_1}. \end{align}

In the second relation, we must consider our use of dropout! If we applied dropout on a particular neuron, it should not be adjusted. To account for this, in the case of `use_dropout=True`, we must instead use

$$\frac{\partial l}{\partial u_1} = \frac{\partial l}{\partial o_1} \cdot \frac{\partial \text{ leaky\_relu}}{\partial u_1} \cdot \text{dropout\_mask} \cdot \frac{1}{1-p},$$

where $1/(1-p)$ is the scaling factor and dropout_mask is stored in `self.cache`.

The final step! We can use these values to compute the gradients for θ_1 and b_1, using the relation $u_1 = \theta_1 X + b_1$, which are given by \begin{align} \frac{\partial l}{\partial \theta_1} &= \frac{\partial l}{\partial u_1} \cdot X \\ \frac{\partial l}{\partial b_1} &= \frac{\partial l}{\partial u_1} \cdot 1. \\ \end{align}

Implementation Tips

The above equations are given in matrix notation. When implementing these computations in code, the easiest way to make sure you are calculating the values correctly and in the right order is to check shapes. Any time you are doing a matrix/vector operation in NumPy, **check the shapes**.

Since we are computing these gradients over N data points, we must divide the gradients by N to take the *average* gradient. Make sure you are dividing by N exactly once, no more and no less!

TODO: Implement the compute_gradients function in NN.py.

```
from utilities.localtests import TestNN

TestNN(
    "test_compute_gradients_without_dropout"
).test_compute_gradients_without_dropout()
TestNN("test_compute_gradients").test_compute_gradients()
```

```
[[-0.07785091 -0.01535662 -0.06638261  0.00452876 -0.00989115]
 [ 0.06728782 -0.00192398  0.05364855  0.00106911 -0.04182411]
 [ 0.04498234  0.0061795  -0.02079071 -0.00161607  0.01896572]]
test_compute_gradients_without_dropout passed!
[[-0.10146069  0.0145839  -0.04601286  0.00931069 -0.02466569]
 [ 0.10639195 -0.01048318 -0.01250874  0.00034275 -0.0718992 ]
 [ 0.01276157  0.00178145 -0.007834   -0.00338637  0.00247914]]
test_compute_gradients passed!
```

Now that we know how to compute relevant gradients and how to update the weights of our network, we can perform the entire backwards step.

TODO: Implement the backward function in NN.py.

1.1.1 Local Test: Gradient Descent

You may test your implementation of the GD function contained in **NN.py** in the cell below. See Using the Local Tests for more details. Look at the function documentation in gradient_descent for guidance.

```
################################
### DO NOT CHANGE THIS CELL ###
################################
from utilities.localtests import TestNN

TestNN("test_gradient_descent").test_gradient_descent()
```

```
[[ 0.01275468  0.03504118 -0.00548491  0.00736414 -0.00580966
0.00049051
   0.0373363  -0.00050762  0.03568805  0.04034137 -0.00527692 -
0.00604833
   0.00795596 -0.0310148   0.03717358]
 [ 0.03170698  0.04890531 -0.00690158  0.02393657 -0.01368514 -
0.00060724
   0.04000803  0.00467808  0.03250821  0.05796458 -0.02646166 -
0.01182229
   0.00680787 -0.00961012  0.03067513]
 [ 0.00621182  0.01137051 -0.00176839  0.00436977 -0.00266881 -
0.00033843
   0.01005672  0.00052174  0.01066107  0.01280688 -0.00462363 -
0.00275664
   0.00266447 -0.00501426  0.01012803]
 [ 0.00468549  0.00558822 -0.00078394  0.00371049 -0.00200437 -
0.00043893
   0.00368699  0.00074096  0.0041001   0.00642884 -0.00458054 -
0.00197829
   0.00130368  0.00111548  0.00330559]
 [ 0.00598079  0.00986602 -0.00106489  0.0041625  -0.00261326 -
0.0004081
   0.00946052  0.00065881  0.00632364  0.0120217  -0.0041996  -
0.00285293
   0.00483731 -0.00295065  0.00808715]
 [ 0.00611429  0.00742521 -0.00094575  0.00478323 -0.00270188 -
0.00033884
   0.00521285  0.00114225  0.00295479  0.00910165 -0.00587039 -
0.00245868
   0.00167083  0.00139222  0.00338296]
 [ 0.01240247  0.01096872 -0.00314856  0.00893618 -0.00445089 -
0.001316
   0.00660045  0.00234887 -0.00150792  0.0137096  -0.01255995 -
0.0026377
   0.00178482  0.00634573  0.00808877]
 [ 0.02771434  0.04109459 -0.00435603  0.02164479 -0.01300522 -
0.00184962
```

```
   0.0330543   0.00416131  0.03317531  0.0495502  -0.02578901 -
0.01282797
   0.00988271 -0.00459041  0.02303763]]
Loss after iteration 0: 1.182135
[[ 0.01247065  0.03466323 -0.00542576  0.00713868 -0.00582018
0.00047889
   0.0369287  -0.00054463  0.03539445  0.03985684 -0.00544222 -
0.00598802
   0.00790482 -0.03090801  0.03680507]
 [ 0.03126463  0.0482188  -0.00682437  0.02359493 -0.01365888 -
0.00062168
   0.0392955   0.00461605  0.03200732  0.05711494 -0.02708587 -
0.01172842
   0.00674524 -0.00935158  0.03001981]
 [ 0.00609026  0.01118741 -0.00174231  0.00427545 -0.00265635 -
0.00034522
   0.00986564  0.0005074   0.01051566  0.01257591 -0.00472843 -
0.00271716
   0.00264258 -0.00494125  0.00995718]
 [ 0.00461553  0.00547959 -0.00077013  0.00365628 -0.0019926  -
0.00044348
   0.00357381  0.00073275  0.00401215  0.00629015 -0.00464272 -
0.00195341
   0.00129109  0.0011626   0.00320444]
 [ 0.00586872  0.00970466 -0.0010476   0.00407644 -0.00259955 -
0.00041413
   0.00929158  0.00064375  0.00620041  0.01181361 -0.0042738  -
0.00279702
   0.00480419 -0.00289517  0.00793933]
 [ 0.00602956  0.00729152 -0.00093086  0.00471889 -0.00268807 -
0.00034313
   0.0050759   0.00113046  0.00285059  0.00893382 -0.00596249 -
0.00242448
   0.00165549  0.00144429  0.00326079]
 [ 0.01219316  0.01060759 -0.00307393  0.00878691 -0.00439435 -
0.00132683
   0.00624723  0.0023244  -0.00168376  0.01327411 -0.01259949 -
0.00252444
   0.00175638  0.00651429  0.0077819 ]
 [ 0.02731999  0.04051265 -0.00430972  0.02133636 -0.01296843 -
0.00187537
   0.03243359  0.00410784  0.0325919   0.04879686 -0.02632835 -
0.01270939
   0.00980369 -0.004381    0.02247886]]
Loss after iteration 1: 1.180133
[[ 0.01219206  0.03407269 -0.00536749  0.0069177  -0.00582917
0.00046756
   0.03652983 -0.00058042  0.0351067   0.03938189 -0.00533831 -
0.00597226
```

```
   0.0078545  -0.03080319  0.03644512]
 [ 0.03083123  0.04734092 -0.00674828  0.02326052 -0.01363026 -
0.00063579
   0.03859841  0.00455574  0.03151633  0.05628188 -0.02693144 -
0.01168006
   0.00668364 -0.00909807  0.02937993]
 [ 0.00597121  0.01061484 -0.00171662  0.00418315 -0.00264329 -
0.00035184
   0.00967883  0.00049349  0.01037321  0.01234956 -0.00467414 -
0.00270256
   0.00262104 -0.00486965  0.00979049]
 [ 0.00454705  0.00504798 -0.00075652  0.00360324 -0.00198051 -
0.00044791
   0.0034632   0.00072476  0.00392602  0.00615424 -0.00460891 -
0.00194305
   0.00127872  0.00120877  0.00310579]
 [ 0.00575893  0.00952236 -0.00103057  0.00399221 -0.00258533 -
0.00042002
   0.0091264   0.00062911  0.0060797   0.01160966 -0.00421626 -
0.00278191
   0.00477163 -0.00284082  0.00779516]
 [ 0.00594658  0.00710921 -0.00091618  0.00465595 -0.00267387 -
0.00034731
   0.00494197  0.00111896  0.0027485   0.00876926 -0.00592112 -
0.00240915
   0.0016404   0.00149532  0.00314159]
 [ 0.01198839  0.01014001 -0.00300041  0.00864115 -0.00433695 -
0.00133738
   0.00590234  0.00230053 -0.00185588  0.01284751 -0.01247942 -
0.00249454
   0.00172844  0.00667925  0.00748293]
 [ 0.02693366  0.0394203  -0.00426407  0.02103444 -0.01292966 -
0.0019005
   0.03182625  0.00405581  0.03202021  0.04805811 -0.02616205 -
0.01264493
   0.00972596 -0.00417555  0.02193331]]
Loss after iteration 2: 1.178184

Your GD losses works within the expected range: True
```

1.1.2 Local Test: Batch Gradient Descent [No Points]

You may test your implementation of the BGD function contained in **NN.py** in the cell below. See Using the Local Tests for more details. Look at the function documentation in gradient_descent for guidance.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################
```

```
from utilities.localtests import TestNN

TestNN("test_batch_gradient_descent").test_batch_gradient_descent()
```

```
[[-0.00758288  0.05003193 -0.0267279  -0.0146184   0.00358087
0.00191496
   0.07215773 -0.01067301  0.09981398  0.05147914  0.02666315 -
0.00174935
   0.0041933  -0.104928    0.089539  ]
 [ 0.00583334  0.03932329 -0.01894499  0.00020931 -0.00281578
0.00102625
   0.04727932 -0.00373096  0.05862171  0.04280011  0.03587835 -
0.00572477
   0.002905   -0.05402362  0.05214382]
 [-0.00275106  0.00960691 -0.00686881 -0.00409163  0.00130584
0.00042028
   0.01489137 -0.00256085  0.01757017  0.00962791  0.00926173
0.00014176
   0.00084809 -0.0232715   0.0191948 ]
 [ 0.00035673  0.00643666 -0.0024682  -0.00055906 -0.00017536
0.00019227
   0.00821741 -0.00084692  0.00725378  0.00688712  0.00711382 -
0.00071646
   0.00049533 -0.01028557  0.00945979]
 [-0.00340772  0.00593314 -0.00257592 -0.00422197  0.00162223
0.00032894
   0.01056365 -0.00225639  0.00479336  0.00560716  0.01276008
0.00071779
   0.00058039 -0.01849374  0.01449593]
 [-0.0022825   0.00345631 -0.00238546 -0.00275446  0.00108698
0.0002037
   0.00639165 -0.00143188  0.00873024  0.00320742  0.00794336
0.00052782
   0.00034796 -0.01149064  0.00890417]
 [-0.01375716  0.00242489 -0.00615009 -0.013991    0.00656592
0.00063647
   0.01420837 -0.00580542 -0.00204469 -0.00016102  0.02926121
0.00485389
   0.00064692 -0.03738221  0.02503811]
 [ 0.01176486  0.03777313 -0.01662206  0.00631322 -0.00564634
0.00073562
   0.04048664 -0.00115046  0.04541689  0.04233512  0.0288298  -
0.00777171
   0.00258631 -0.0370321   0.04056354]]
Loss after iteration 0: 1.106816
[[-6.57741336e-03  2.01037830e-02 -5.49712348e-04 -9.40474230e-03
   2.85861040e-03 -6.16604917e-04  3.16407737e-02 -3.75548649e-03
   1.22226735e-02  1.99858982e-02 -7.78591696e-03 -2.16208333e-03
   1.80324367e-03 -5.04962941e-02  4.11724139e-02]
```

```
 [-1.23597394e-02  4.78220402e-02 -1.39867220e-03 -1.91066164e-02
   5.26177910e-03 -4.03661544e-04  7.26535127e-02 -8.14069454e-03
   3.36824586e-02  4.81924612e-02 -1.23882596e-02 -3.61047530e-03
   4.18025463e-03 -1.12206005e-01  9.28817962e-02]
 [-2.22362606e-03  6.89946770e-03 -1.89590534e-04 -3.19416301e-03
   9.65283762e-04 -6.21852826e-04  1.08320947e-02 -1.20081060e-03
   2.92684353e-03  6.86568321e-03 -5.18150122e-03 -1.53665331e-03
   6.17739883e-04 -1.72488278e-02  1.40782131e-02]
 [-1.74971231e-03  3.10074000e-03 -6.44180405e-05 -2.18101466e-03
   7.85027477e-04 -3.26246492e-04  5.46458876e-03 -4.97619955e-04
   2.11300568e-03  2.93694889e-03 -2.54404788e-03 -3.88961070e-04
   3.02562822e-04 -9.55859296e-03  7.48176464e-03]
 [-8.05386349e-03 -8.79694646e-03  5.43383494e-04 -6.74568663e-03
   3.86583492e-03 -2.97573305e-04 -5.15566047e-03 -1.03697861e-03
   5.76879096e-03 -1.09103807e-02  1.85822532e-03 -2.39124146e-04
  -4.25716974e-04 -4.22480556e-03 -1.19258835e-03]
 [-3.38117283e-03 -3.72630063e-03  2.29330756e-04 -2.82723911e-03
   1.62331759e-03 -3.55137452e-04 -2.20802402e-03 -5.88473346e-04
  -6.00880707e-03 -4.61551712e-03 -6.84547135e-04  3.28014409e-04
  -1.81338782e-04 -1.71647669e-03 -5.51898393e-04]
 [-1.76722014e-02 -1.76984869e-02  1.13391465e-03 -1.50307543e-02
   8.46506348e-03 -8.11147283e-04 -9.20516992e-03 -4.19810395e-03
  -2.78434999e-02 -2.22413546e-02  2.82256726e-03  2.36519496e-03
  -8.07680754e-04 -1.20360511e-02 -1.39114316e-04]
 [ 8.49605625e-03  4.49050447e-02 -1.87022805e-03  2.03015517e-03
  -4.46782597e-03 -3.63041335e-03  5.22433086e-02  2.35173151e-03
   4.96052733e-02  4.92339366e-02 -3.58101739e-02 -1.00652253e-02
   3.25715670e-03 -5.69627199e-02  5.62810082e-02]]
Loss after iteration 1: 1.112495
[[ 3.49526308e-03  2.16240609e-02 -8.90548535e-04  3.90446825e-04
   2.66812544e-03 -1.28978135e-03  2.55089196e-02 -1.89911301e-03
  -2.17619451e-02  2.35737681e-02  3.19752067e-02  5.49680331e-03
   1.58948774e-03 -2.87855060e-02  2.77954471e-02]
 [ 8.61746872e-02  1.09677551e-01 -6.38758841e-03  7.00238149e-02
  -1.70407250e-02 -3.72081059e-03  7.52016375e-02  1.76172491e-02
  -3.42294007e-02  1.33201198e-01  3.52768037e-02  5.50965583e-03
   5.77155021e-03  1.89155835e-02  3.61495550e-02]
 [ 3.39359234e-03  7.34141243e-03 -3.62660923e-04  2.32650049e-03
   2.22567309e-03 -1.79784314e-03  6.91339088e-03  2.33862490e-04
  -1.54734733e-02  8.44296924e-03  1.31168291e-02  3.23913955e-03
   4.65785804e-04 -4.45530537e-03  6.05657042e-03]
 [ 5.59757028e-03  8.60152047e-03 -4.69226280e-04  4.33776699e-03
  -3.99596589e-04 -9.61148479e-04  6.81651809e-03  9.19541012e-04
  -7.69515253e-03  1.02151563e-02  7.35243119e-03  1.61277088e-03
   4.91476975e-04 -1.31319927e-03  4.61275515e-03]
 [-4.86645545e-03  5.42241447e-03 -6.63529100e-05 -5.61118428e-03
   3.90393209e-03 -1.78075952e-03  1.09424706e-02 -2.76255642e-03
  -2.20312856e-02  4.76731904e-03  2.62221416e-02  5.72886390e-03
   5.90746398e-04 -2.10552618e-02  1.57655026e-02]
```

```
 [ 2.77367288e-03  5.97122470e-03 -2.95342275e-04  1.90566183e-03
   6.36189105e-04 -6.20991320e-04  5.61243829e-03  1.95571202e-04
  -1.05148460e-02  6.86986910e-03  9.90121385e-03  2.38019118e-03
   3.78402188e-04 -3.59135824e-03  4.90557721e-03]
 [ 4.06421162e-02  7.34393532e-02 -3.81082350e-03  2.99280950e-02
  -6.03649517e-03 -9.39211056e-04  6.38585115e-02  5.00461527e-03
  -1.16442173e-02  8.57922699e-02  3.81597510e-02  3.57849192e-03
   4.43544706e-03 -2.84384938e-02  5.03335219e-02]
 [-3.64090595e-03 -2.23981457e-02  9.22988716e-04 -4.24824518e-04
   7.00317245e-03 -8.90928128e-03 -2.64058279e-02  1.95892634e-03
  -7.71867836e-02 -2.44217352e-02  2.65476995e-02  1.40617978e-02
  -1.64570055e-03  2.97665068e-02 -2.87590215e-02]]
Loss after iteration 2: 1.301159

y_train input: [[0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 ...
 [0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]]
batch_y at iteration 0:  [[0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 0. 1.]]
batch_y at iteration 1:  [[1. 0. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 0. 1.]]
batch_y at iteration 2:  [[0. 0. 1.]
 [0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]]

Your BGD losses works within the expected range: True
Your batch_y works within the expected range: True
```

1.1.3 Local Test: Gradient Descent with Momentum

You may test your implementation of the GD function with momentum contained in **NN.py** in the cell below. See Using the Local Tests for more details. Revisit your implementation for update_weights.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from utilities.localtests import TestNN

TestNN("test_gradient_descent_with_momentum").test_gradient_descent_wi
th_momentum()
```

```
[[ 0.01275468  0.03504118 -0.00548491  0.00736414 -0.00580966
0.00049051
   0.0373363  -0.00050762  0.03568805  0.04034137 -0.00527692 -
0.00604833
   0.00795596 -0.0310148   0.03717358]
 [ 0.03170698  0.04890531 -0.00690158  0.02393657 -0.01368514 -
0.00060724
   0.04000803  0.00467808  0.03250821  0.05796458 -0.02646166 -
0.01182229
   0.00680787 -0.00961012  0.03067513]
 [ 0.00621182  0.01137051 -0.00176839  0.00436977 -0.00266881 -
0.00033843
   0.01005672  0.00052174  0.01066107  0.01280688 -0.00462363 -
0.00275664
   0.00266447 -0.00501426  0.01012803]
 [ 0.00468549  0.00558822 -0.00078394  0.00371049 -0.00200437 -
0.00043893
   0.00368699  0.00074096  0.0041001   0.00642884 -0.00458054 -
0.00197829
   0.00130368  0.00111548  0.00330559]
 [ 0.00598079  0.00986602 -0.00106489  0.0041625  -0.00261326 -
0.0004081
   0.00946052  0.00065881  0.00632364  0.0120217  -0.0041996  -
0.00285293
   0.00483731 -0.00295065  0.00808715]
 [ 0.00611429  0.00742521 -0.00094575  0.00478323 -0.00270188 -
0.00033884
   0.00521285  0.00114225  0.00295479  0.00910165 -0.00587039 -
0.00245868
   0.00167083  0.00139222  0.00338296]
 [ 0.01240247  0.01096872 -0.00314856  0.00893618 -0.00445089 -
0.001316
   0.00660045  0.00234887 -0.00150792  0.0137096  -0.01255995 -
0.0026377
   0.00178482  0.00634573  0.00808877]
 [ 0.02771434  0.04109459 -0.00435603  0.02164479 -0.01300522 -
0.00184962
   0.0330543   0.00416131  0.03317531  0.0495502  -0.02578901 -
0.01282797
   0.00988271 -0.00459041  0.02303763]]
Loss after iteration 0: 1.182135
```

```
[[ 0.01247065  0.03466323 -0.00542576  0.00713868 -0.00582018
0.00047889
   0.0369287  -0.00054463  0.03539445  0.03985684 -0.00544222 -
0.00598802
   0.00790482 -0.03090801  0.03680507]
 [ 0.03126463  0.0482188  -0.00682437  0.02359493 -0.01365888 -
0.00062168
   0.0392955   0.00461605  0.03200732  0.05711494 -0.02708587 -
0.01172842
   0.00674524 -0.00935158  0.03001981]
 [ 0.00609026  0.01118741 -0.00174231  0.00427545 -0.00265635 -
0.00034522
   0.00986564  0.0005074   0.01051566  0.01257591 -0.00472843 -
0.00271716
   0.00264258 -0.00494125  0.00995718]
 [ 0.00461553  0.00547959 -0.00077013  0.00365628 -0.0019926  -
0.00044348
   0.00357381  0.00073275  0.00401215  0.00629015 -0.00464272 -
0.00195341
   0.00129109  0.0011626   0.00320444]
 [ 0.00586872  0.00970466 -0.0010476   0.00407644 -0.00259955 -
0.00041413
   0.00929158  0.00064375  0.00620041  0.01181361 -0.0042738  -
0.00279702
   0.00480419 -0.00289517  0.00793933]
 [ 0.00602956  0.00729152 -0.00093086  0.00471889 -0.00268807 -
0.00034313
   0.0050759   0.00113046  0.00285059  0.00893382 -0.00596249 -
0.00242448
   0.00165549  0.00144429  0.00326079]
 [ 0.01219316  0.01060759 -0.00307393  0.00878691 -0.00439435 -
0.00132683
   0.00624723  0.0023244  -0.00168376  0.01327411 -0.01259949 -
0.00252444
   0.00175638  0.00651429  0.0077819 ]
 [ 0.02731999  0.04051265 -0.00430972  0.02133636 -0.01296843 -
0.00187537
   0.03243359  0.00410784  0.0325919   0.04879686 -0.02632835 -
0.01270939
   0.00980369 -0.004381    0.02247886]]
Loss after iteration 1: 1.180133
[[ 0.01205187  0.03388543 -0.00533792  0.00680663 -0.00583307
0.00046179
   0.03632794 -0.00059854  0.03496056  0.0391415  -0.00528598 -
0.00596382
   0.00782898 -0.03074905  0.03626296]
 [ 0.03061334  0.04700018 -0.00670966  0.02309272 -0.01361461 -
0.00064297
   0.03824557  0.00452544  0.0312669   0.05586035 -0.02685354 -
```

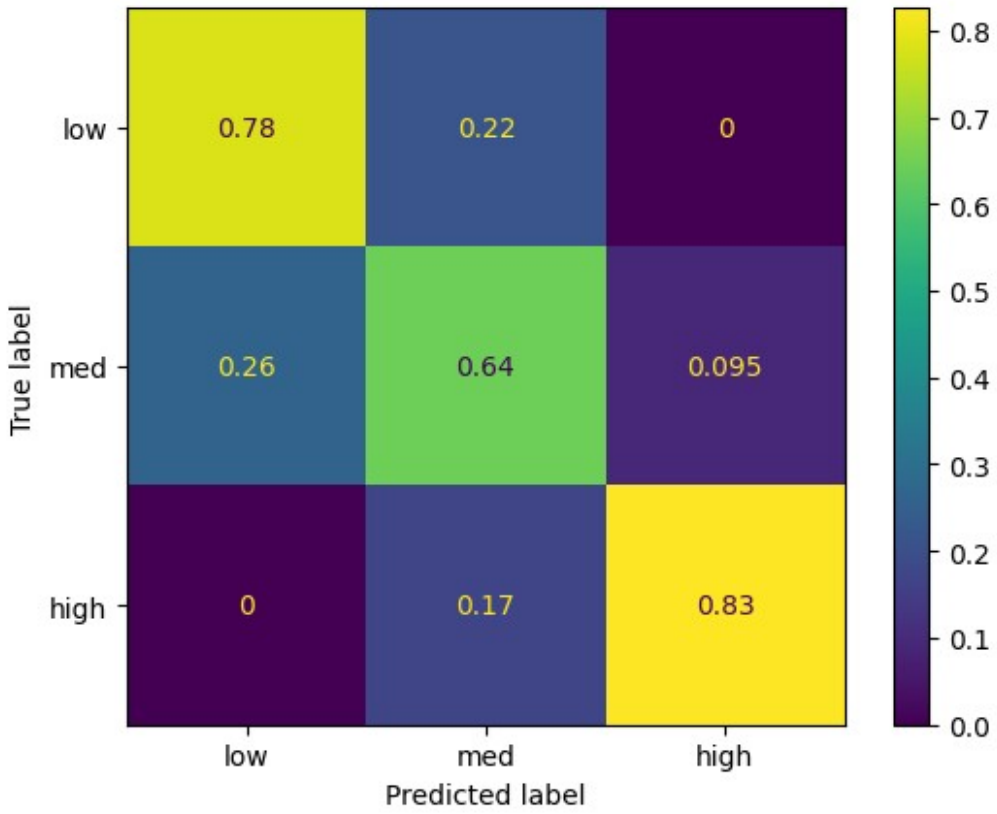
```
0.01165481
   0.00665238 -0.00896739  0.0290561 ]
 [ 0.00591139  0.01052499 -0.00170359  0.00413686 -0.00263638 -
0.0003552
   0.00958434  0.00048652  0.01030091  0.01223512 -0.00464677 -
0.00269496
   0.00261012 -0.00483279  0.00970617]
 [ 0.00451266  0.00499496 -0.00074963  0.00357666 -0.00197421 -
0.00045016
   0.00340728  0.00072079  0.00388231  0.00608556 -0.00459187 -
0.00193769
   0.00127244  0.00123254  0.0030559 ]
 [ 0.00570375  0.00944229 -0.00102194  0.00394997 -0.00257786 -
0.00042302
   0.00904285  0.00062177  0.00601841  0.01150653 -0.00418725 -
0.00277408
   0.00475512 -0.00281275  0.00772225]
 [ 0.00590489  0.00704291 -0.00090874  0.00462441 -0.00266645 -
0.00034943
   0.00487422  0.00111321  0.00269665  0.00868605 -0.00590024 -
0.00240123
   0.00163274  0.00152163  0.00308129]
 [ 0.0118856   0.00996105 -0.00296313  0.00856827 -0.0043073  -
0.00134275
   0.00572786  0.00228872 -0.0019433   0.01263186 -0.01241882 -
0.00247914
   0.00171425  0.00676419  0.00733165]
 [ 0.02673952  0.03913255 -0.00424092  0.02088299 -0.01290904 -
0.00191328
   0.03151905  0.00402969  0.03172992  0.04768456 -0.02607819 -
0.01261152
   0.00968654 -0.00406957  0.02165742]]
Loss after iteration 2: 1.177207

Your GD losses works within the expected range: True
```

1.2 Loss plot and CE value for NN with Gradient Descent [5pts] **[W]**

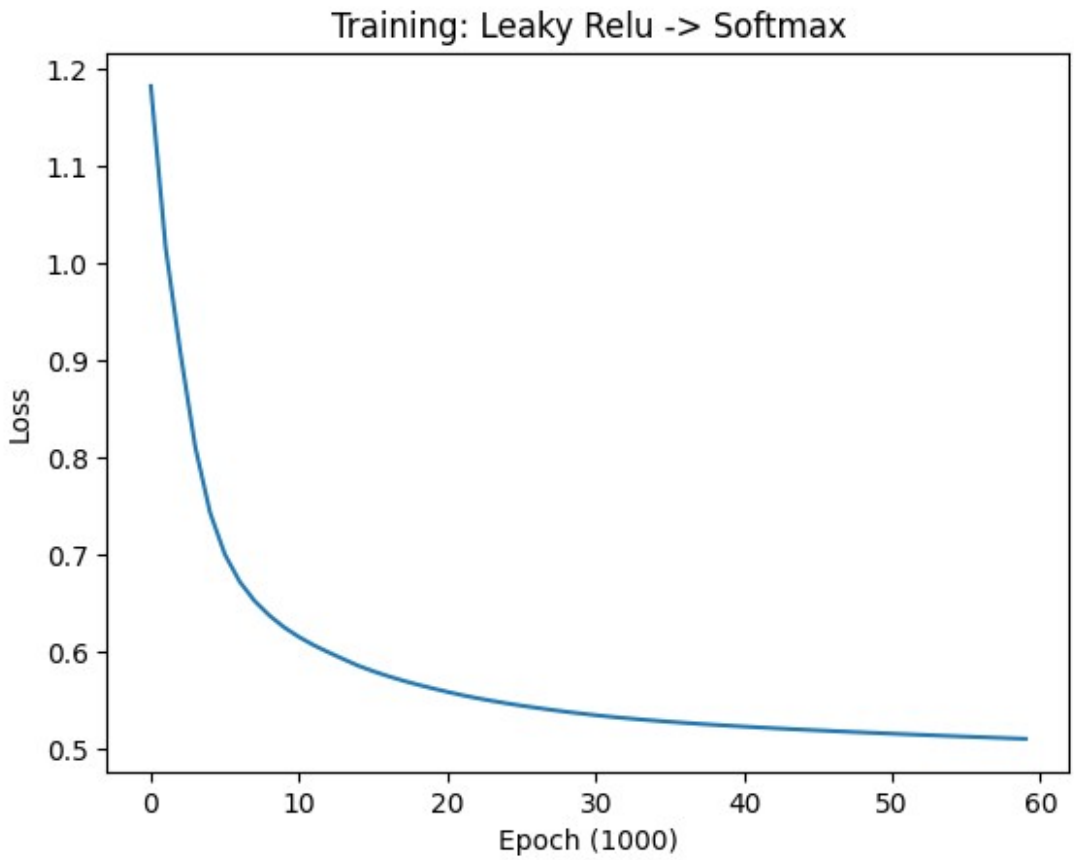
Train your neural net implementation with gradient descent and print out the loss at every 1000th iteration (starting at iteration 0). The following cells will plot the loss vs epoch graph and calculate the final test CE.

```
################################
### DO NOT CHANGE THIS CELL ###
################################
from NN import NeuralNet
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

```
x_train, y_train, x_test, y_test = get_housing_dataset()

nn = NeuralNet(
    y_train, lr=0.01, use_dropout=False, use_momentum=False
)  # initalize neural net class
nn.gradient_descent(x_train, y_train, iter=60000)  # train

# Plot confusion matrix
y_true = np.argmax(y_test, axis=1)
y_pred = nn.predict(x_test)
display_labels = ["low", "med", "high"]
ConfusionMatrixDisplay.from_predictions(
    y_true, y_pred, normalize="true", display_labels=display_labels
)
plt.show()
```




```
# Plot training loss
fig = plt.plot(np.array(nn.loss).squeeze())
plt.title(f"Training: {nn.neural_net_type}")
plt.xlabel("Epoch (1000)")
plt.ylabel("Loss")
plt.show()
```




```
# Total loss
y_hat = nn.forward(x_test, use_dropout=False)
print("Cross entropy loss:", round(nn.cross_entropy_loss(y_test,
y_hat), 3))
```

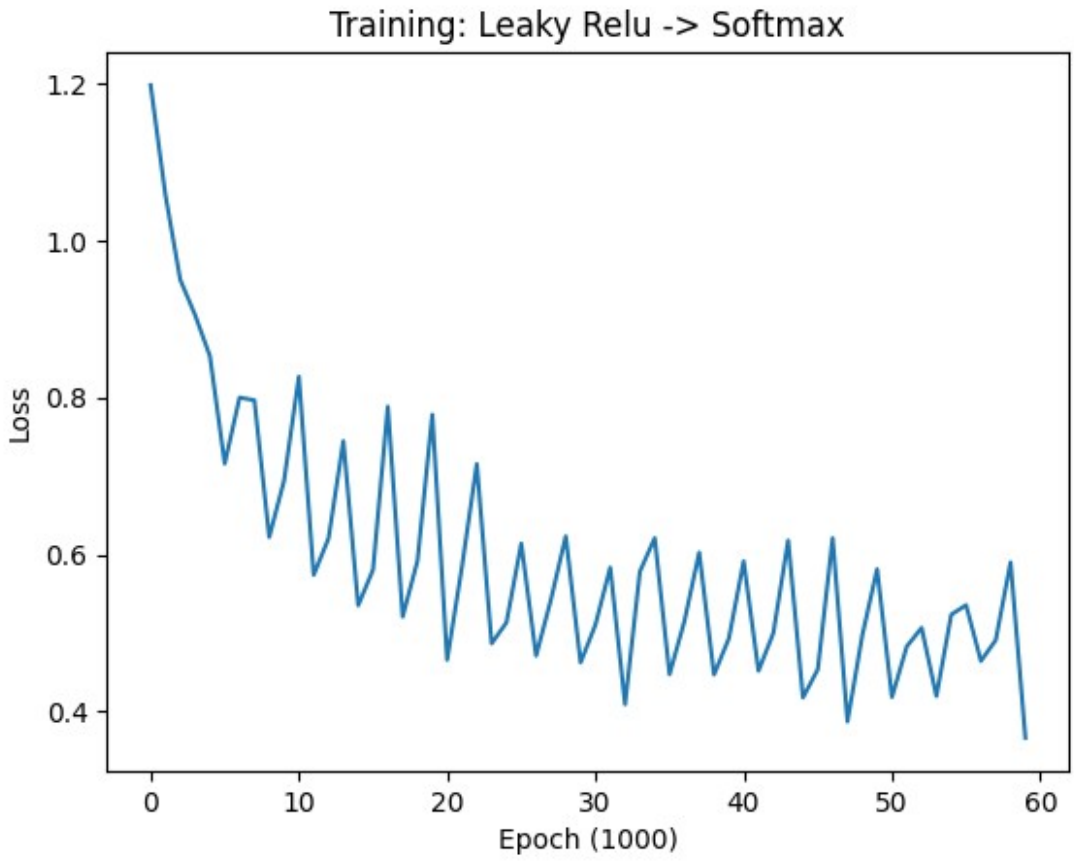
```
Cross entropy loss: 0.752
```

1.3 Loss plot and CE value for NN with BGD [5pts Bonus for Undergrad] **[W]**

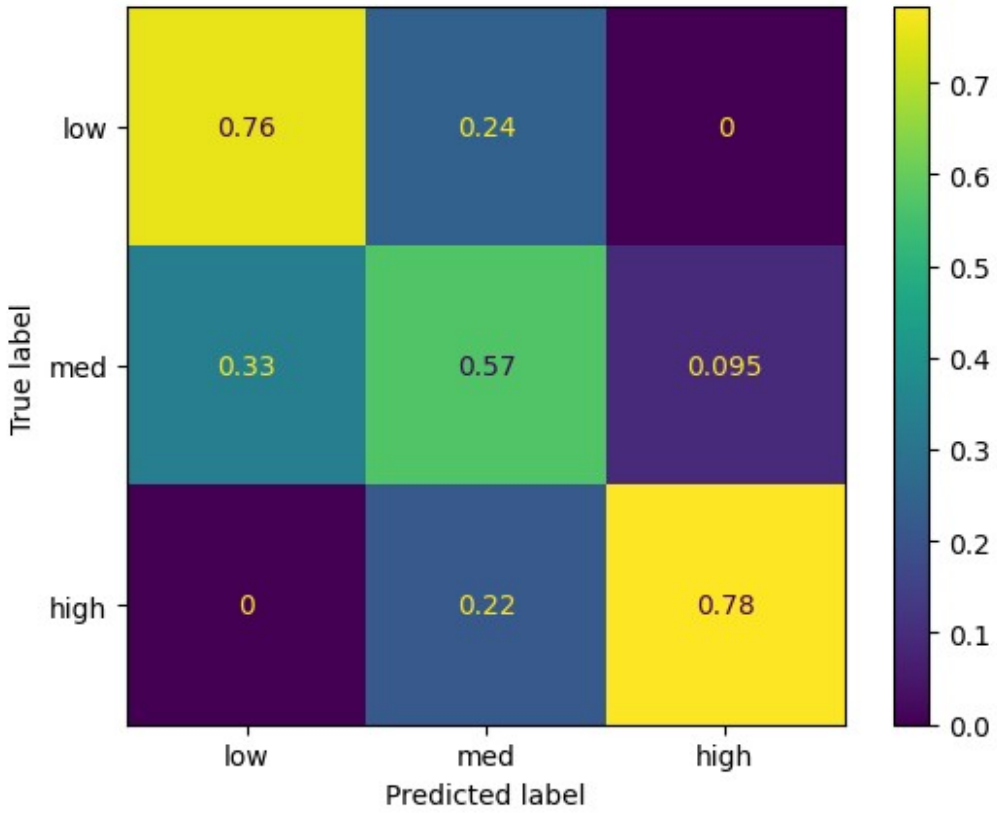
Train your neural net implementation with batch gradient descent and print out the loss at every 1000th iteration (starting at iteration 0). The following cells will plot the loss vs epoch graph and calculate the final test CE.

```
################################
### DO NOT CHANGE THIS CELL ###
################################
from NN import NeuralNet
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

x_train, y_train, x_test, y_test = get_housing_dataset()
```

```
nn = NeuralNet(
    y_train, lr=0.01, use_dropout=True, use_momentum=False
)  # initalize neural net class
nn.batch_gradient_descent(x_train, y_train, iter=60000,
use_momentum=False)

# Plot training loss
fig = plt.plot(np.array(nn.loss).squeeze())
plt.title(f"Training: {nn.neural_net_type}")
plt.xlabel("Epoch (1000)")
plt.ylabel("Loss")
plt.show()
```




```
# Plot confusion matrix
y_true = np.argmax(y_test, axis=1)
y_pred = nn.predict(x_test)
display_labels = ["low", "med", "high"]
ConfusionMatrixDisplay.from_predictions(
    y_true, y_pred, normalize="true", display_labels=display_labels
)
plt.show()
```




```
# Total loss
y_hat = nn.forward(x_test, use_dropout=False)
print("Cross entropy loss:", round(nn.cross_entropy_loss(y_test,
y_hat), 3))
```

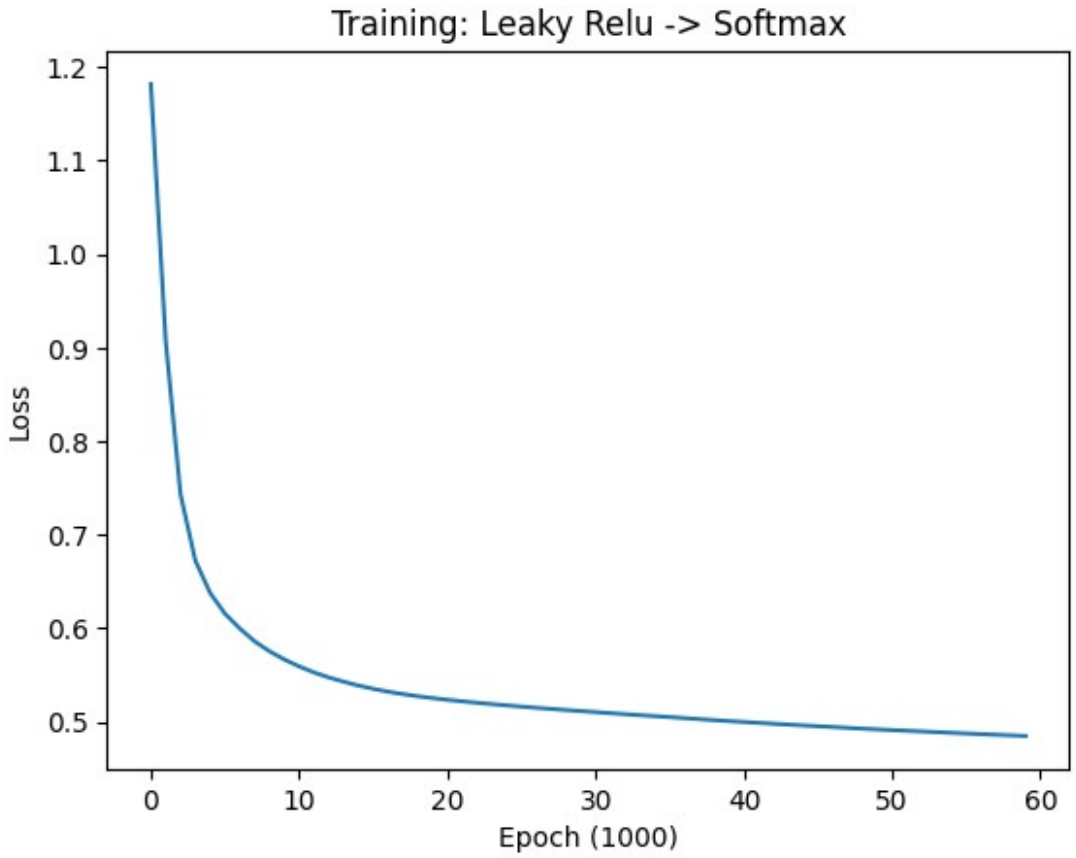
```
Cross entropy loss: 0.808
```

1.4 Loss plot and CE value for NN with Gradient Descent with Momentum [5pts Bonus for Undergrad] **[W]**

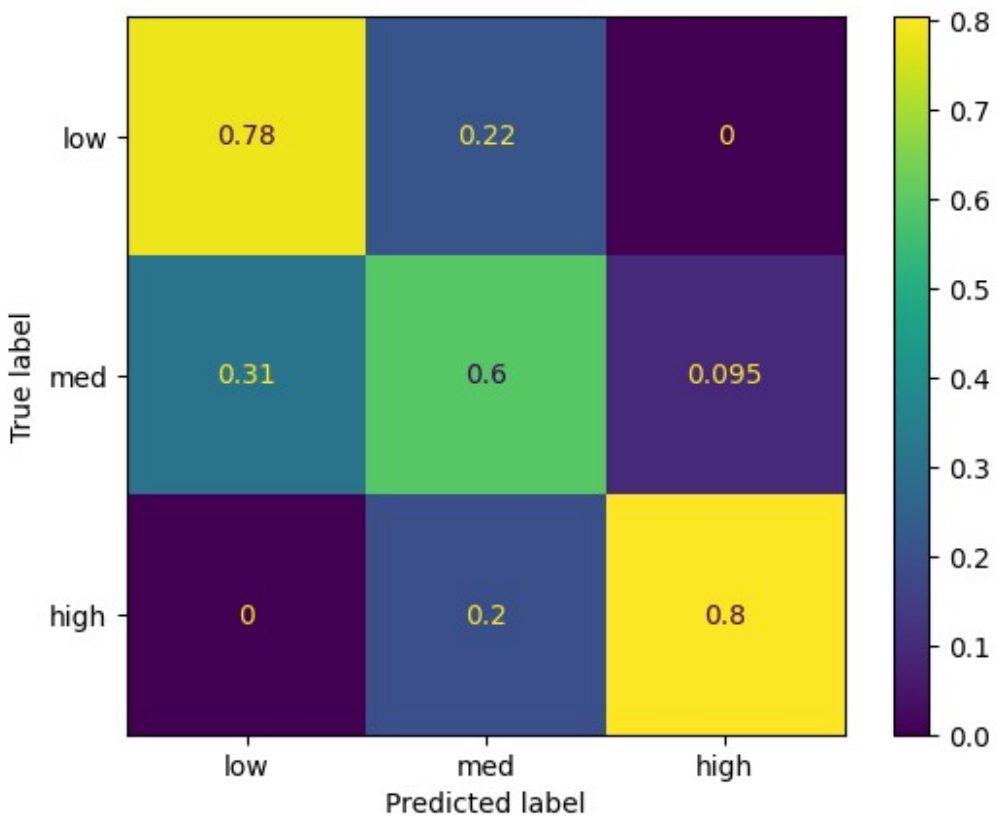
Train your neural net implementation with gradient descent with momentum and print out the loss at every 1000th iteration (starting at iteration 0). The following cells will plot the loss vs epoch graph and calculate the final test CE.

```
################################
### DO NOT CHANGE THIS CELL ###
################################
from NN import NeuralNet
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

x_train, y_train, x_test, y_test = get_housing_dataset()

nn = NeuralNet(
```

```
    y_train, lr=.01, use_dropout=False, use_momentum=True
)  # initalize neural net class
nn.gradient_descent(x_train, y_train, iter=60000, use_momentum=True)
# train

# Plot training loss
fig = plt.plot(np.array(nn.loss).squeeze())
plt.title(f"Training: {nn.neural_net_type}")
plt.xlabel("Epoch (1000)")
plt.ylabel("Loss")
plt.show()
```




```
# Plot confusion matrix
y_true = np.argmax(y_test, axis=1)
y_pred = nn.predict(x_test)
display_labels = ["low", "med", "high"]
ConfusionMatrixDisplay.from_predictions(
    y_true, y_pred, normalize="true", display_labels=display_labels
)
plt.show()
```




```
# Total loss
y_hat = nn.forward(x_test, use_dropout=False)
print("Cross entropy loss:", round(nn.cross_entropy_loss(y_test,
y_hat), 3))
```

```
Cross entropy loss: 0.733
```

2: Image Classification based on Convolutional Neural Networks [25pts; 20pts Bonus for Undergrad + 5pts Bonus for all] **[P][W]**

2.1 Image Classification using Pytorch and CNN

- Pytorch is a popular platform for machine learning.

Pytorch Description

PyTorch is a Machine Learning/Deep Learning tensor library based on Python and Torch. It uses dynamic computation graphs and is completely Pythonic. Pytorch is used for applications using GPUs and CPUs.

Helpful Links

- Install Pytorch
- Pytorch Quickstart Tutorial

Setup Pytorch

Make sure you installed pytorch and torchvision (directions here).

Please also see Pytorch Quickstart Tutorial to see how to load a data set, build a training loop, and test the model. Another good resource for building CNNs using Pytorch is here.

Environment Setup

```
from torch.utils.data import non_deterministic
import torchvision
import torch
from torchvision.transforms import v2

%load_ext autoreload
%autoreload 2
%reload_ext autoreload

The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

2.1.1 Load CIFAR-10 Dataset and Data Augmentation [5pts - Bonus for Undergrad]**[P]**

We use CIFAR-10 dataset to train our model. This is a dataset of 60,000 32x32 colour images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images. We provide code for you to download CIFAR-10 dataset below.

Data Augmentation [5pts]

Data augmentation is a technique to increase the diversity of your training set by applying random (but realistic) transformations such as image rotation and flipping the image around an axis. If the dataset in a machine learning model is rich and sufficient, the model performs better and more accurately. We will preprocess the training and testing set, but only the training set will undergo augmentation.

Go through the Pytorch torchvision.transforms.v2 documentation to see how to apply multiple transformations at once.

In the cnn_image_transformations.py file, complete the following functions to understand the common practices used for preprocessing and augmenting the image data:

- create_training_transformations
  - In this function, you are going to preprocess and augment training data.
    - PREPROCESS: Convert the given PIL Images to Tensors

        - AUGMENTATION: Apply Random Horizontal Flip and Random Rotation

- create_testing_transformations
    - In this function, you are going to only preprocess testing data.
        - PREPROCESS: Convert the given PIL Images to Tensors

Please note that the Gradescope only checks if expected preprocessing layers are existent.

References

v2.Compose()

v2.ToTensor() (Hint: Look at the warning)

v2.RandomHorizontalFlip()

v2.RandomApply()

v2.RandomRotation()

Article about performance regarding transformations

```
################################
### DO NOT CHANGE THIS CELL ###
################################

from cnn_image_transformations import create_training_transformations
from cnn_image_transformations import create_testing_transformations

# Create Transformations
training_transformations = create_training_transformations()
testing_transformation = create_testing_transformations()

# Load data
trainset = torchvision.datasets.CIFAR10(
    root="./data", train=True, download=True,
transform=training_transformations
)
testset = torchvision.datasets.CIFAR10(
    root="./data", train=False, download=True,
transform=testing_transformation
)

classes = (
    "plane",
    "car",
    "bird",
    "cat",
    "deer",
    "dog",
```

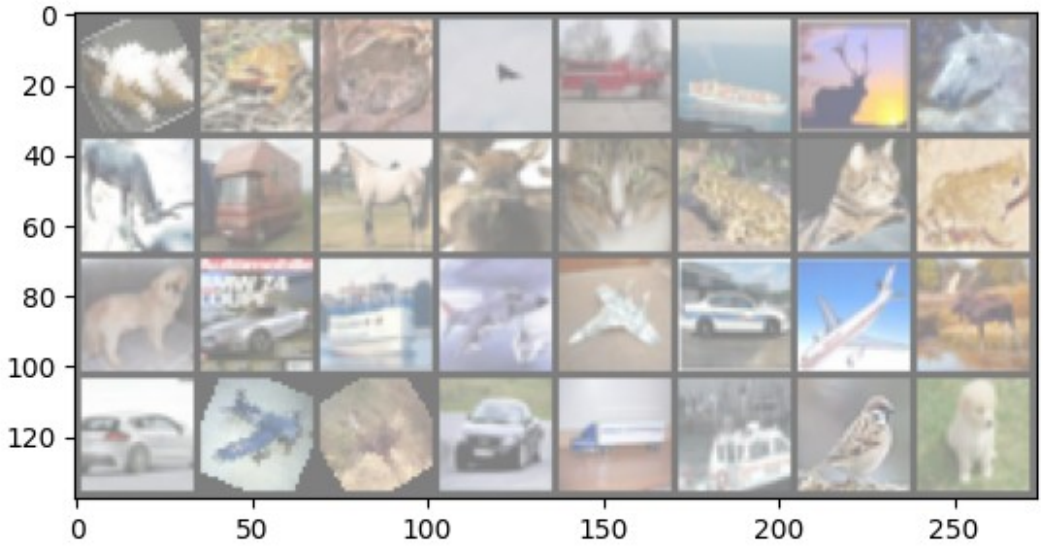
```
    "frog",
    "horse",
    "ship",
    "truck",
)

print(trainset.data.shape)
print(testset.data.shape)
```

```
Files already downloaded and verified
Files already downloaded and verified
(50000, 32, 32, 3)
(10000, 32, 32, 3)
```

2.1.2 Load some sample images from CIFAR-10 [Setup - No points]

```
################################
### DO NOT CHANGE THIS CELL ###
################################

import matplotlib.pyplot as plt
import numpy as np

trainloader = torch.utils.data.DataLoader(
    trainset, batch_size=32, shuffle=True, num_workers=2
)
testloader = torch.utils.data.DataLoader(
    testset, batch_size=32, shuffle=False, num_workers=2
)

# functions to show an image


def imshow(img):
    img = img / 2 + 0.5  # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(trainloader)
images, labels = next(dataiter)

print("Image size")
print(v2.functional.get_size(images[0]))

# show images
imshow(torchvision.utils.make_grid(images))
```

```
Image size
[32, 32]
```



As you can see from above, the CIFAR-10 dataset contains different types of objects. The images have been size-normalized and objects remain centered in fixed-size images.

2.1.3 Build convolutional neural network model [5pts] **[W]**

In this part, you need to build a convolutional neural network as described below. The architecture of the model is outlined.

In the cnn.py file, complete the following functions:

- *_init_*: See Defining Variables section
- forward: See Defining Model section

[INPUT - CONV - CONV - MAXPOOL - DROPOUT - CONV - CONV - MAXPOOL - DROPOUT - AVERAGEPOOL - FC1 - DROPOUT - FC2 - DROPOUT - FC3]

INPUT: $[32 \times 32 \times 3]$ will hold the raw pixel values of the image, in this case, an image of width 32, height 32, with 3 channels.

CONV: Conv. layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to the input volume. In our example architecture, we decide to set the kernel_size to be 3×3. For example, the output of the Conv. layer may look like $[32 \times 32 \times 8)$ if we set out_channels to be 8 and use appropriate paddings to maintain shape.

CONV: Additional Conv. layer take outputs from above layers and applies more filters. We set the kernel_size to be 3×3 and out_channels to be 32.

MAXPOOL: MAXPOOL layer will perform a downsampling operation along the spatial dimensions (width, height). With pool size of 2×2, resulting shape takes form 16×16.

DROPOUT: DROPOUT layer with the dropout rate of 0.2 to prevent overfitting.

CONV: Additonal Conv. layer takes outputs from above layers and applies more filters. We set the kernel_size to be 3×3 and out_channels to be 32. Appropriate paddings are used to maintain shape.

CONV: Additonal Conv. layer takes outputs from above layers and applies more filters. We set the kernel_size to be 3×3 and out_channels to be 64. Appropriate paddings are used to maintain shape.

MAXPOOL: MAXPOOL layer will perform a downsampling operation along the spatial dimensions (width, height).

DROPOUT: Dropout layer with the dropout rate of 0.2 to prevent overfitting.

AVERAGEPOOL: AVERAGEPOOL layer will perform a downsampling operation along the spatial dimension (width, height). Checkout AdaptiveAvgPool2d below.

FC1: Dense layer which takes output from above layers, and has 256 neurons. Flatten() operations may be useful.

DROPOUT: Dropout layer with the dropout rate of 0.2 to prevent overfitting.

FC2： Dense layer which takes output from above layers, and has 128 neurons.

DROPOUT: Dropout layer with the dropout rate of 0.2 to prevent overfitting.

FC3: Dense layer with 10 neurons, and Softmax activation, is the final layer. The dimension of the output space is the number of classes.

Activation function: Use LeakyReLU with negative_slope 0.01 as the activation function for Conv. layers and Dense layers unless otherwise indicated to build you model architecture

Note that while this is a suggested model design, you are welcome to use other architectures and experiment with different layers for better results.

The following links are Pytorch documentation for the layers you are going to use to build the CNN.

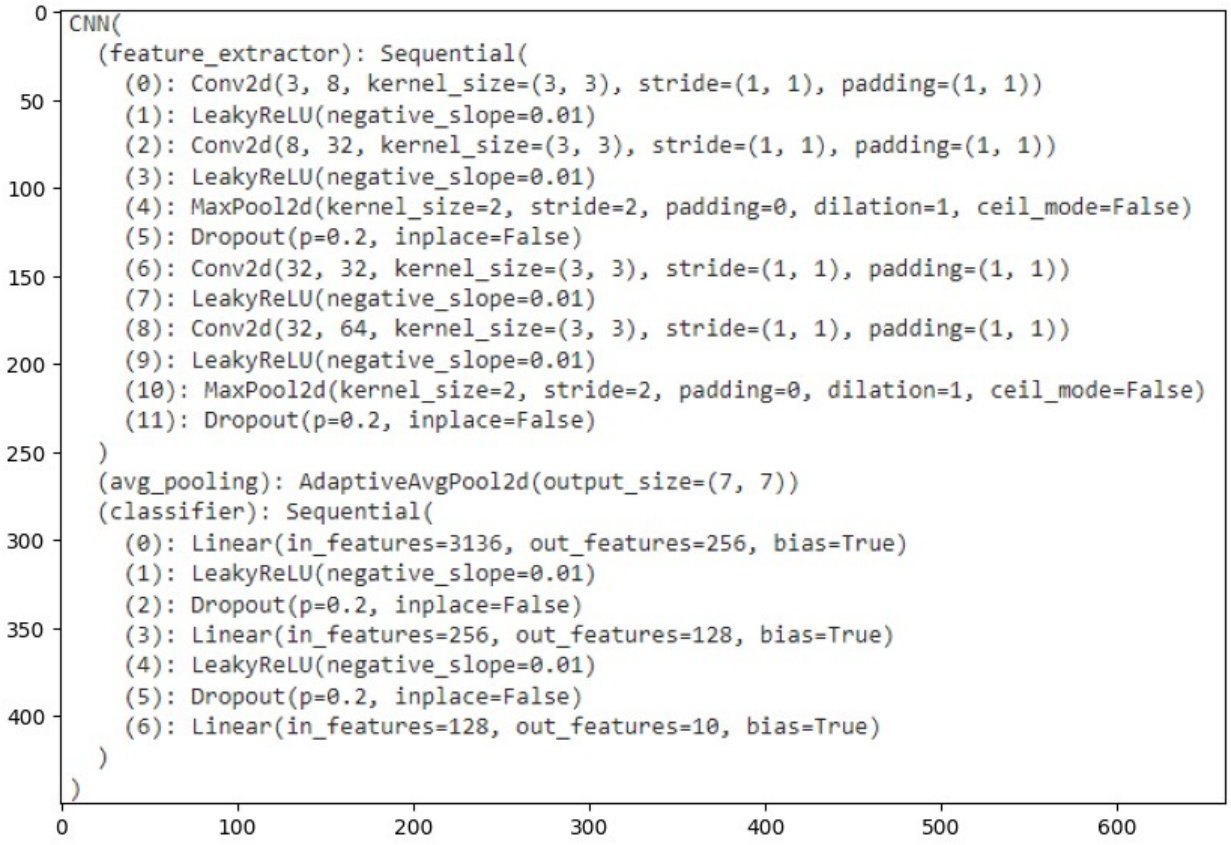
- Conv2d
- Dense
- MaxPool
- AdaptiveAvgPool2d
- Dropout
- LeakyReLU
- Flatten

Lastly, if you would like to experiment with additional layers, explore the torch.nn api.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

# Show the architecture of the model
achi = plt.imread("./data/images/Architecture.png")
```

```
fig = plt.figure(figsize=(10, 10))
plt.imshow(achi)
```

```
<matplotlib.image.AxesImage at 0x7b5c8456bd60>
```



Defining model [5pts Bonus for Undergrad]**[W]**

You now need to complete the `__init__()` function and the `forward()` function in cnn.py to define your model structure.

Your model is required to have at least 2 convolutional layers and at least 2 dense layers. Ensuring that these requirements are met will earn you 5pts.

Once you have defined a model structure you may use the cell below to examine your architecture.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

# You can compare your architecture with the 'Architecture.png'

from cnn import CNN
```

```
net = CNN()
print(net)
```

```
CNN(
  (feature_extractor): Sequential(
    (0): Conv2d(3, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): LeakyReLU(negative_slope=0.01)
    (2): Conv2d(8, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (3): LeakyReLU(negative_slope=0.01)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (5): Dropout(p=0.2, inplace=False)
    (6): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (7): LeakyReLU(negative_slope=0.01)
    (8): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (9): LeakyReLU(negative_slope=0.01)
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (11): Dropout(p=0.2, inplace=False)
  )
  (avg_pooling): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=3136, out_features=256, bias=True)
    (1): LeakyReLU(negative_slope=0.01)
    (2): Dropout(p=0.2, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): LeakyReLU(negative_slope=0.01)
    (5): Dropout(p=0.2, inplace=False)
    (6): Linear(in_features=128, out_features=10, bias=True)
  )
)
```

2.1.4 Train the network [8pts total (3pts, 3pts, 2pts) Bonus for Undergrad] **[W]**

Tuning: Training the network is the next thing to try. You can set the hyperparameters in the cell below. If your hyperparameters are set properly, you should see the loss of the validation set decreased and the value of accuracy increased. It may take more than 15 minutes to train your model.

- Recommended Batch Sizes fall in the range 32-512 (use powers of 2)
- Recommended Epoch Counts fall in the range 5-20
- Recommended Learning Rates fall in the range .0001-.01

Expected Result: You should be able to achieve more than 75% accuracy on the test set to get full points. If you achieve accuracy between 60% to 69%, you will only get 3 points. An accuracy between 69% to 75% will earn an additional 3pts.

Note: If you would like to automate the tuning process, you can use a nested for loop to search for the hyperparameter that achieves the accuracy.

- 60% to 69% earns 3pts
- 69% to 75% earns 3pts more (6pts total)
- 75%+ earns 2pts more (8pts total)

Train your own CNN model

```
from cnn import CNN
from cnn_trainer import Trainer

net = CNN()

# TODO: Change hyperparameters here
num_epochs = 10
batch_size = 32
init_lr = 1e-3

# Choose best device to speed up training
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using {device} device")

trainer = Trainer(
    net,
    trainset,
    testset,
    num_epochs=num_epochs,
    batch_size=batch_size,
    init_lr=init_lr,
    device=device,
)
trainer.train()
```

```
Using cuda device

Epoch 1/10: 100%|██████████| 1563/1563 [00:35<00:00, 44.04batch/s,
accuracy=0.395, loss=1.64]

Epoch 1: Validation Loss: 1.27, Validation Accuracy: 0.541

Epoch 2/10: 100%|██████████| 1563/1563 [00:35<00:00, 44.24batch/s,
accuracy=0.559, loss=1.24]

Epoch 2: Validation Loss: 1.02, Validation Accuracy: 0.645

Epoch 3/10: 100%|██████████| 1563/1563 [00:37<00:00, 41.65batch/s,
accuracy=0.62, loss=1.08]
```

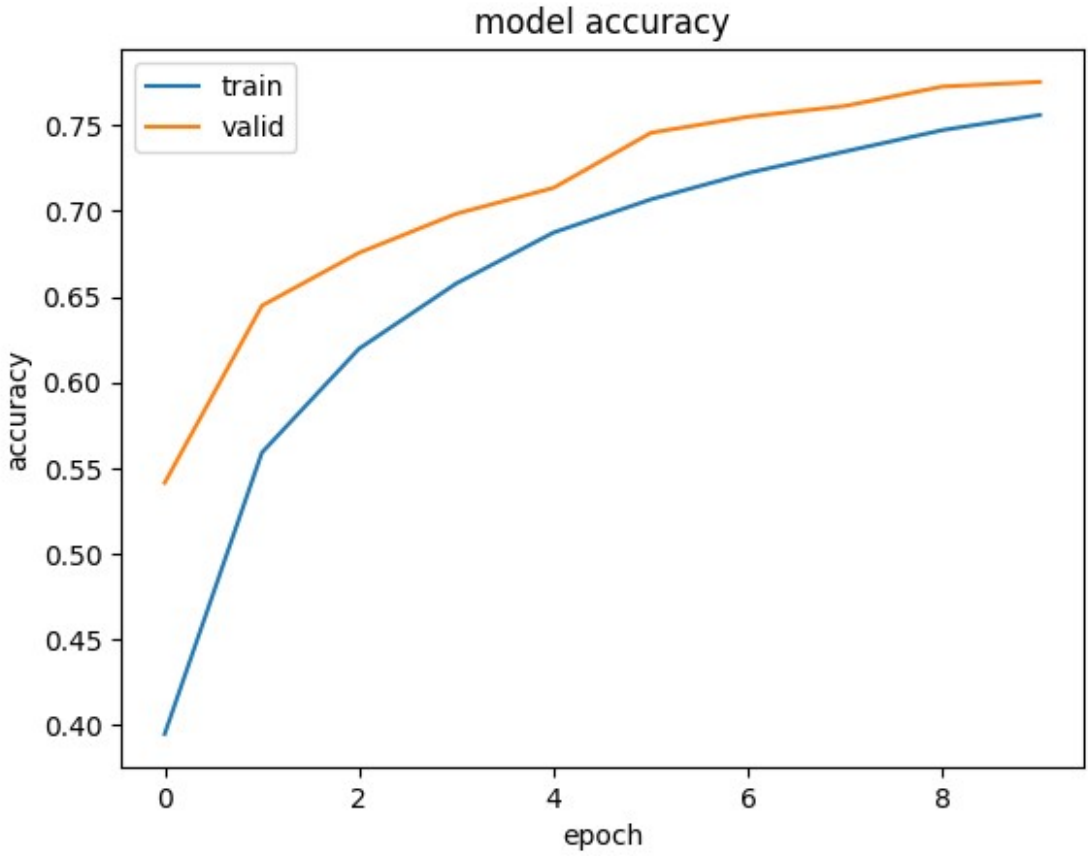
```
Epoch 3: Validation Loss: 0.93, Validation Accuracy: 0.676

Epoch 4/10: 100%|██████████| 1563/1563 [00:35<00:00, 43.76batch/s,
accuracy=0.658, loss=0.982]

Epoch 4: Validation Loss: 0.86, Validation Accuracy: 0.698

Epoch 5/10: 100%|██████████| 1563/1563 [00:35<00:00, 43.94batch/s,
accuracy=0.687, loss=0.904]

Epoch 5: Validation Loss: 0.83, Validation Accuracy: 0.714

Epoch 6/10: 100%|██████████| 1563/1563 [00:35<00:00, 44.52batch/s,
accuracy=0.707, loss=0.847]

Epoch 6: Validation Loss: 0.74, Validation Accuracy: 0.746

Epoch 7/10: 100%|██████████| 1563/1563 [00:35<00:00, 44.35batch/s,
accuracy=0.722, loss=0.801]

Epoch 7: Validation Loss: 0.72, Validation Accuracy: 0.755

Epoch 8/10: 100%|██████████| 1563/1563 [00:36<00:00, 42.44batch/s,
accuracy=0.735, loss=0.765]

Epoch 8: Validation Loss: 0.69, Validation Accuracy: 0.761

Epoch 9/10: 100%|██████████| 1563/1563 [00:37<00:00, 41.57batch/s,
accuracy=0.747, loss=0.733]

Epoch 9: Validation Loss: 0.67, Validation Accuracy: 0.773

Epoch 10/10: 100%|██████████| 1563/1563 [00:35<00:00, 44.05batch/s,
accuracy=0.756, loss=0.709]

Epoch 10: Validation Loss: 0.66, Validation Accuracy: 0.775
```

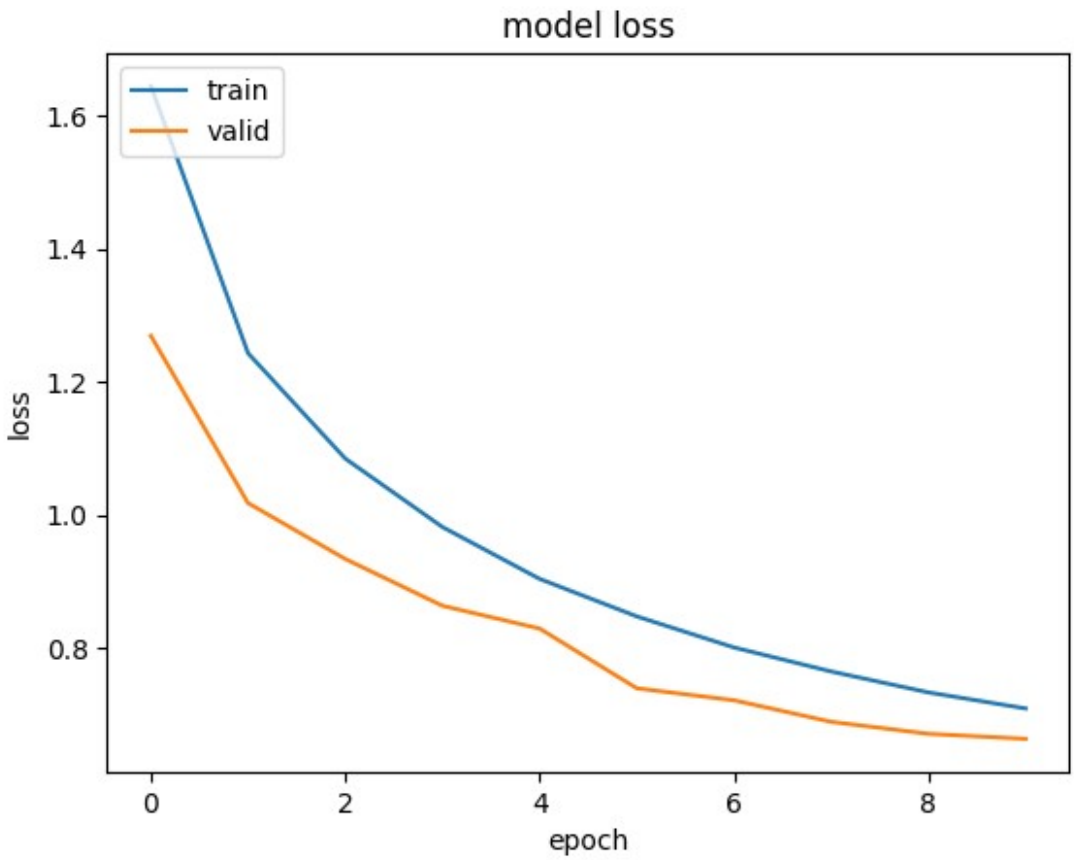
2.1.5 Examine accuracy and loss [2pts Bonus for Undergrad] **[W]**

You should expect to see gradually decreasing loss and gradually increasing accuracy. Examine loss and accuracy by running the cell below, no editing is necessary. Having appropriate looking loss and accuracy plots will earn you the last 2pts for your convolutional neural net.

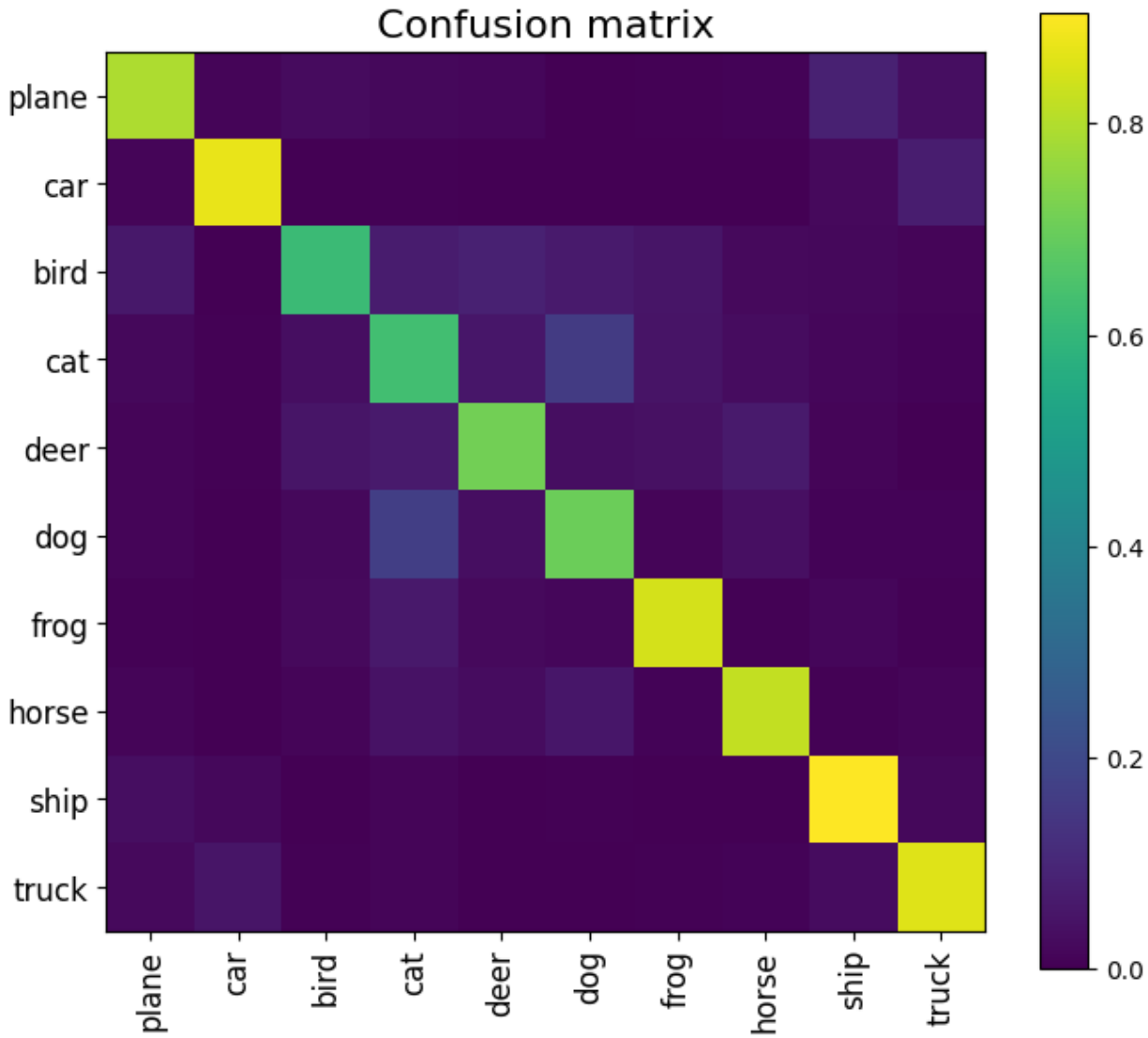
```
################################
### DO NOT CHANGE THIS CELL ###
################################

# list all data in history
train_loss, train_accuracy, valid_loss, valid_accuracy =
trainer.get_training_history()

# summarize history for accuracy and loss
```

```
plt.plot(train_accuracy)
plt.plot(valid_accuracy)
plt.title("model accuracy")
plt.ylabel("accuracy")
plt.xlabel("epoch")
plt.legend(["train", "valid"], loc="upper left")
plt.show()

plt.plot(train_loss)
plt.plot(valid_loss)
plt.title("model loss")
plt.ylabel("loss")
plt.xlabel("epoch")
plt.legend(["train", "valid"], loc="upper left")
plt.show()
```






```
################################
### DO NOT CHANGE THIS CELL ###
################################

# make predictions
y_pred, y_pred_classes, y_gt_classes = trainer.predict(testloader)
y_pred_prob = torch.max(y_pred, dim=1).values

from sklearn.metrics import confusion_matrix, accuracy_score

plt.figure(figsize=(8, 7))
plt.imshow(confusion_matrix(y_gt_classes, y_pred_classes,
normalize="true"))
plt.title("Confusion matrix", fontsize=16)
plt.xticks(np.arange(10), classes, rotation=90, fontsize=12)
plt.yticks(np.arange(10), classes, fontsize=12)
plt.colorbar()
plt.show()
```



2.2 Exploring Deep CNN Architectures [5pts Bonus for All] **[W]**

The network you have produced is rather simple relative to many of those used in industry and research. Researchers have worked to make CNN models deeper and deeper over the past years in an effort to gain higher accuracy in predictions. While your model is only a handful of layers deep, some state of the art deep architectures may include up to 150 layers. However, this process has not been without challenges.

One such problem is the problem of the vanishing gradient. The weights of a neural network are updated using the backpropagation algorithm. The backpropagation algorithm makes a small change to each weight in such a way that the loss of the model decreases. Using the chain rule, we can find this gradient for each weight. But, as this gradient keeps flowing backwards to the initial layers, this value keeps getting multiplied by each local gradient. Hence, the gradient

becomes smaller and smaller, making the updates to the initial layers very small, increasing the training time considerably.

Many tactics have been used in an effort to solve this problem. One architecture, named ResNet, solves the vanishing gradient problem in a unique way. ResNet was developed at Microsoft Research to find better ways to train deep networks. Take a moment to explore how ResNet tackles the vanishing gradient problem by reading the original research paper here: https://arxiv.org/pdf/1512.03385.pdf (also included as PDF in papers directory).

Question: In your own words, explain how ResNet addresses the vanishing gradient problem in 1-2 sentences below: (Please type answers directly in the cell below.)

Answer: Residual networks uses short paths which can carry gradient throughout the extent of very deep networks and therefore avoids the vanishing gradient problem.

3: Random Forests [45pts; 40pts + 5pts Bonus for All] **[P] [W]**

NOTE: Please use sklearn's ExtraTreeClassifier in your Random Forest implementation. You can find more details about this classifier here.

For context, the general difference between an extra tree and decision tree classifier is that the decision tree optimizes which feature to reduce entropy on and at what value to split, while an extra tree randomly splits on the features given.

3.1 Random Forest Implementation [35pts] **[P]**

The decision boundaries drawn by decision or extra trees are very sharp, and fitting a tree of unbounded depth to a list of examples almost inevitably leads to **overfitting**. In an attempt to decrease the variance of an extra tree, we're going to use a technique called 'Bootstrap Aggregating' (often abbreviated 'bagging'). This stems from the idea that a collection of weak learners can learn decision boundaries as well as a strong learner. This is commonly called a Random Forest.

We can build a Random Forest as a collection of extra trees, as follows:

1. For every tree in the random forest, we're going to

   a) Subsample the examples with replacement. Note that in this question, the size of the subsample data is equal to the original dataset.

   b) From the subsamples in part a, choose attributes at random without replacement to learn on in accordance with a provided attribute subsampling rate. Based on what it was mentioned in the class, we randomly pick features in each split. We use a more general approach here to make the programming part easier. Let's randomly pick some features (65% percent of features) and grow the tree based on the pre-determined randomly selected features. Therefore, there is no need to find random features in each split.

c) Fit an extra tree to the subsample of data we've chosen to a certain depth.

Classification for a random forest is then done by taking a majority vote of the classifications yielded by each tree in the forest after it classifies an example.

In the random_forest.py file, complete the following functions:

- _bootstrapping: this function will be used in `bootstrapping()`
- fit: Fit the extra trees initialized in `__init__` with the datasets created in `bootstrapping()`. You will need to call `bootstrapping()`.

NOTES:

1. In the Random Forest Class, X is assumed to be a matrix with num_training rows and num_features columns where num_training is the number of total records and num_features is the number of features of each record. y is assumed to be a vector of labels of length num_training.
2. Look out for TODO's for the parts that need to be implemented
3. If you receive any `SettingWithCopyWarning` warnings from the Pandas library, you can safely ignore them.
4. Hint: when bootstrapping, set replace = False while creating col_idx

3.2 Hyperparameter Tuning with a Random Forest [5pts] [P]

In machine learning, hyperparameters are parameters that are set before the learning process begins. The max_depth, num_estimators, or max_features variables from 3.1 are examples of different hyperparameters for a random forest model. Let's first review the dataset in a bit more detail.

Dataset Objective

Imagine that we are a team of researchers working to track and document various information related to dry beans for a machine learning model that predicts what type of bean is represented. We know that there are multiple things to keep track of, such as the shapes and sizes that differentiate different types of beans. We will use the information we track and document in order to publish it for the general public.

After much reflection within the research team, we come to the conclusion that we can use past observations on bean images to create a model.

We will use our random forest algorithm from Q3.1 to predict the bean type.

You can find more information on the dataset here.

The barbunya bean, also known as the cranberry bean, was first bred in Colombia.

A barbunya bean

Loading the dataset

The dataset that the company has collected has the following features:

There were 16 features used in this dataset.

Inputs:

1. Area The area of a bean zone and the number of pixels within its boundaries
2. Perimeter: Bean circumference is defined as the length of its border
3. MajorAxisLength: The distance between the ends of the longest line that can be drawn from a bean
4. MinorAxisLength: The longest line that can be drawn from the bean while standing perpendicular to the main axis
5. AspectRatio: Defines the relationship between MajorAxisLength and MinorAxisLength
6. Eccentricity: Eccentricity of the ellipse having the same moments as the region
7. ConvexArea: Number of pixels in the smallest convex polygon that can contain the area of a bean seed
8. EquivDiameter Equivalent diameter, the diameter of a circle having the same area as a bean seed area
9. Extent Feature: The ratio of the pixels in the bounding box to the bean area
10. Solidity: Also known as convexity. The ratio of the pixels in the convex shell to those found in beans.
11. Roundness: Calculated with the following formula: (4piA)/(P^2)
12. Compactness: Measures the roundness of an object
13. ShapeFactor1
14. ShapeFactor2
15. ShapeFactor3
16. ShapeFactor4

Output:

1. Target value:
   - Seker
   - Barbunya
   - Bombay
   - Cali
   - Dermosan
   - Horoz
   - Sira

Your random forest model will try to predict this variable.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################
```

```
from sklearn import preprocessing
import pandas as pd
import numpy as np

dry_bean_dataset = "./data/Dry_Bean_Dataset.csv"
df = pd.read_csv(dry_bean_dataset)

label_encoder = preprocessing.LabelEncoder()

X = df.drop(["Class"], axis=1)
y = label_encoder.fit_transform(df["Class"])

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42
)
X_test = np.array(X_test)
X_train, y_train, X_test, y_test = (
    np.array(X_train),
    np.array(y_train),
    np.array(X_test),
    np.array(y_test),
)

################################
### DO NOT CHANGE THIS CELL ###
################################
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
assert X_train.shape == (9119, 16)
assert y_train.shape == (9119,)
assert X_test.shape == (4492, 16)
assert y_test.shape == (4492,)
```

```
(9119, 16) (9119,) (4492, 16) (4492,)
```

In the following codeblock, train your random forest model with different values for max_depth, n_estimators, or max_features and evaluate each model on the held-out test set. Try to choose a combination of hyperparameters that maximizes your prediction accuracy on the test set (aim for 85%+).

In **random_forest.py**, once you are satisfied with your chosen parameters, update the following function:

- **select_hyperparameters**: change the values for `max_depth`, `n_estimators`, and `max_features`to your chosen values

Submit this file to Gradescope. You must achieve at least a **85% accuracy** against the test set in Gradescope to receive full credit for this section.

```
################################
### DO NOT CHANGE THIS CELL ###
```

```
################################
from utilities.localtests import TestRandomForest

"""
Once you have implemented Random forest, you can run this cell. If you
implemented _bootStrapping correctly,
then this cell should execute without any errors.
"""
TestRandomForest("test_bootstrapping").test_bootstrapping()
```

```
test_bootstrapping passed!
```

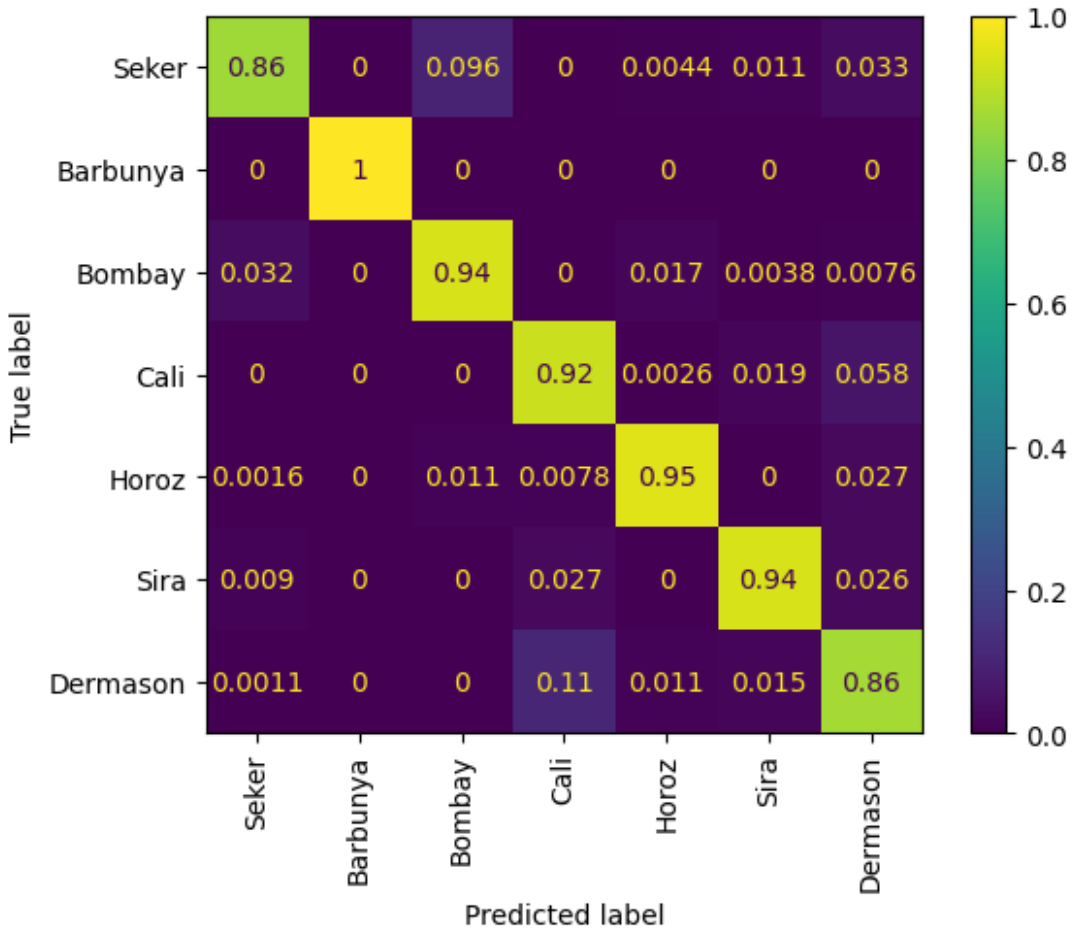
```
"""
TODO:
n_estimators defines how many Extra trees are fitted for the random
forest.
max_depth defines a stop condition when the tree reaches to a certain
depth.
max_features controls the percentage of features that are used to fit
each extra tree.

Tune these three parameters to achieve a better accuracy. n_estimators
and max_depth must both
be at least 3 in value for moderately reliable answers. While you can
use the provided test set
to evaluate your implementation, you will need to obtain 85% on the
test set to receive full
credit for this section.
"""
from random_forest import RandomForest
from sklearn import preprocessing
import sklearn.ensemble

################## DO NOT CHANGE THIS RANDOM SEED ####################
student_random_seed = 4641 + 7641
######################################################################

################## CHANGE THESE VALUES ################################
n_estimators = 11  # Hint: Consider values between 3-15.
max_depth = 12  # Hint: Consider values betweeen 3-15.
max_features = 0.6  # Hint: Consider values betweeen 0.3-1.0.
######################################################################
random_forest = RandomForest(
    n_estimators, max_depth, max_features,
random_seed=student_random_seed
)
random_forest.fit(X_train, y_train)
accuracy = random_forest.OOB_score(X_test, y_test)
print("accuracy: %.4f" % accuracy)
```

```
accuracy: 0.8769
```

DON'T FORGET: Once you are satisfied with your chosen parameters, change the values for `max_depth`, `n_estimators`, and `max_features` in the `select_hyperparameters()` function of your RandomForest class in `random_forest.py` to your chosen values, and then submit this file to Gradescope. You must achieve at least a **85% accuracy** against the test set in Gradescope to receive full credit for this section.

Below is a code block that plots a confusion matrix for the classifier's predictions on the test set. A few things to think about: What are some trends seen in the matrix? Why do they happen?

```
from sklearn.metrics import ConfusionMatrixDisplay

pred = random_forest.predict(X_test)
labels = ["Seker", "Barbunya", "Bombay", "Cali", "Horoz", "Sira",
"Dermason"]
ConfusionMatrixDisplay.from_predictions(
    y_test, pred, display_labels=labels, normalize="true",
xticks_rotation="vertical"
)
plt.show()
```



3.3 Plotting Feature Importance [5pts Bonus for All] **[W]**

While building tree-based models, it's common to quantify how well splitting on a particular feature in an extra tree helps with predicting the target label in a dataset. Machine learning practicioners typically use "Gini importance", or the (normalized) total reduction in entropy brought by that feature to evaluate how important that feature is for predicting the target variable.

Gini importance is typically calculated as the reduction in entropy from reaching a split in an extra tree weighted by the probability of reaching that split in the extra tree. Sklearn internally computes the probability for reaching a split by finding the total number of samples that reaches it during the training phase divided by the total number of samples in the dataset. This weighted value is our feature importance.

Let's think about what this metric means with an example. A high probabiity of reaching a split on feature A in an extra tree trained on a dataset (many samples will reach this split for a decision) and a large reduction in entropy from splitting on feature A will result in a high feature importance value for feature A. This could mean feature A is a very important feature for predicting the probability of the target label. On the other hand, a low probability of reaching a split on feature B in an extra tree and a low reduction in entropy from splitting on feature B will result in a low feature importance value. This could mean feature B is not a very informative

feature for predicting the target label. **Thus, the higher the feature importance value, the more important the feature is to predicting the target label.**

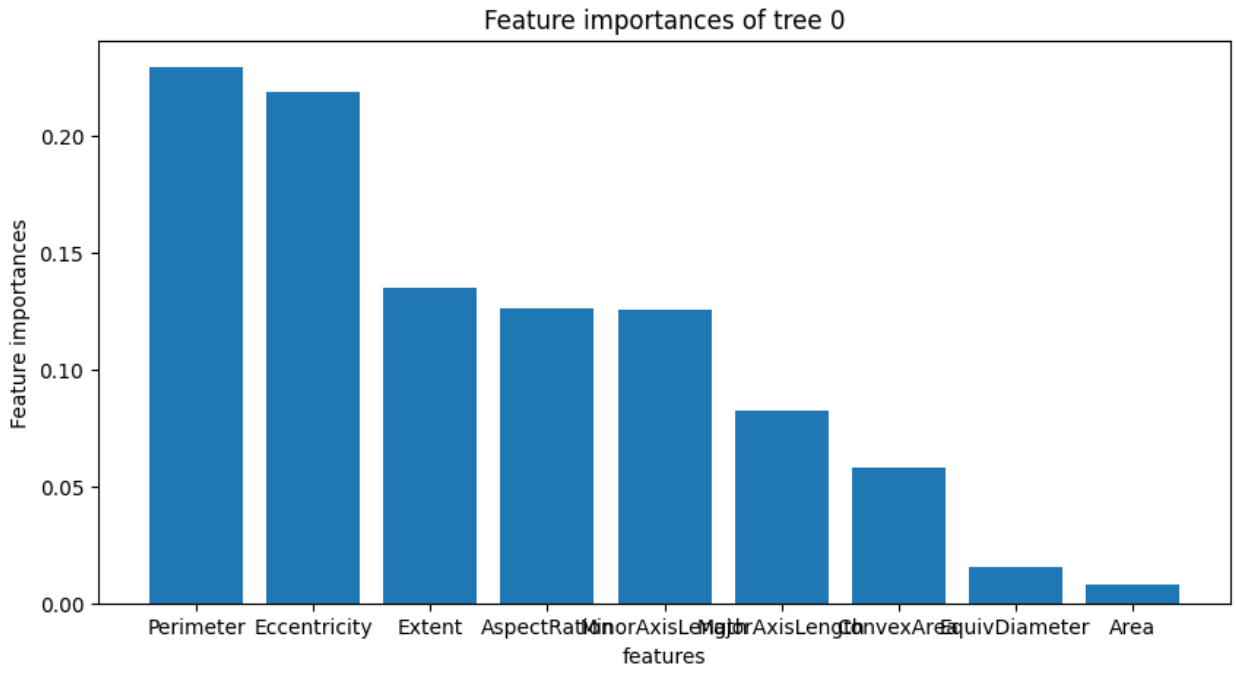
Fortunately for us, fitting a sklearn.ExtraTreeClassifier to a dataset auomatically computes the Gini importance for every feature in the extra tree and stores these values in a **feature_importances_** variable. Review the docs for more details on how to access this variable

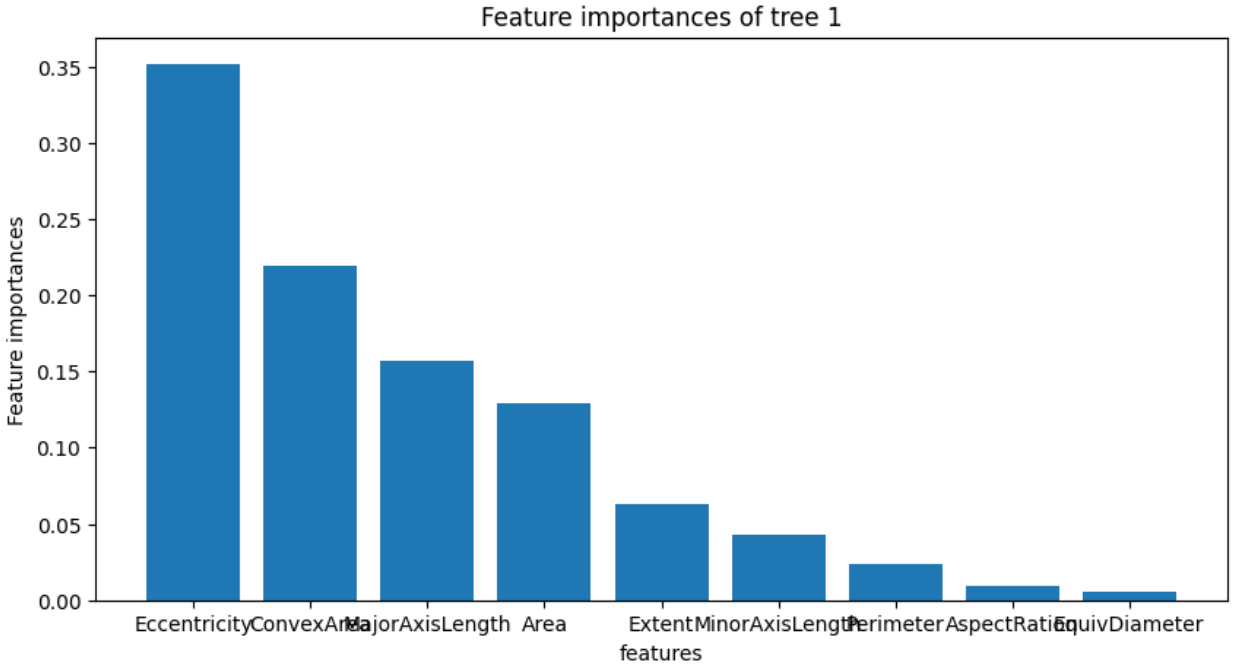
In the random_forest.py file, complete the following function:

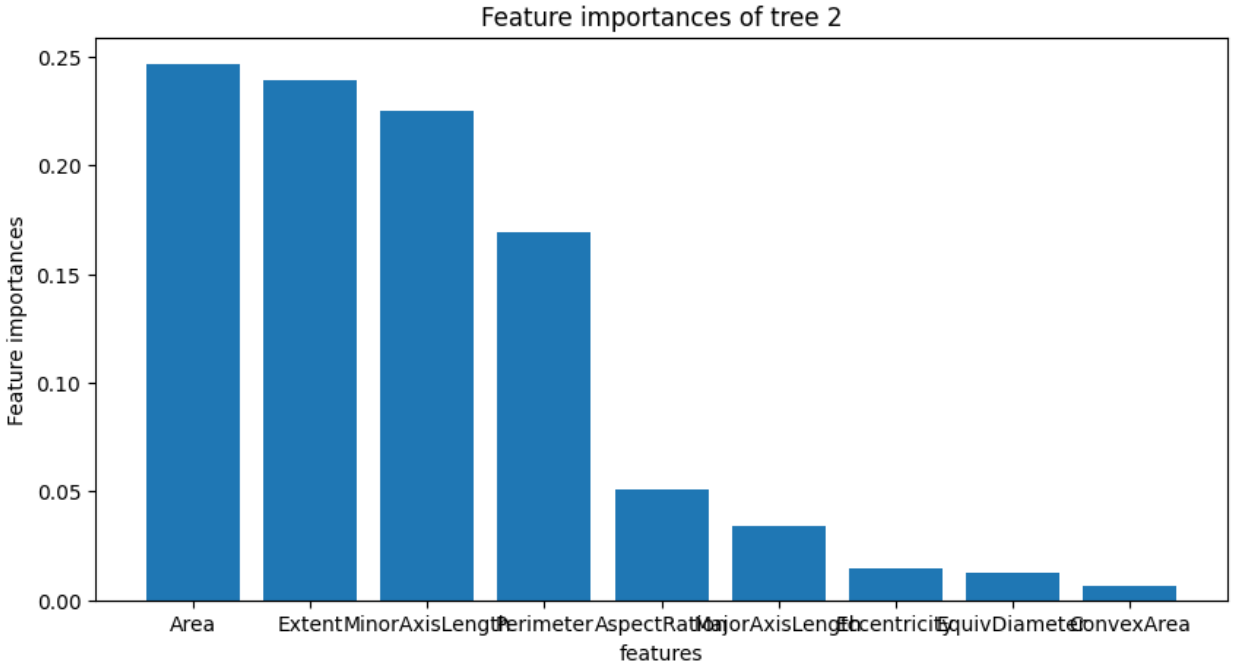
- plot_feature_importance: Make sure to sort the bars in descending order and remove any features with feature importance of 0

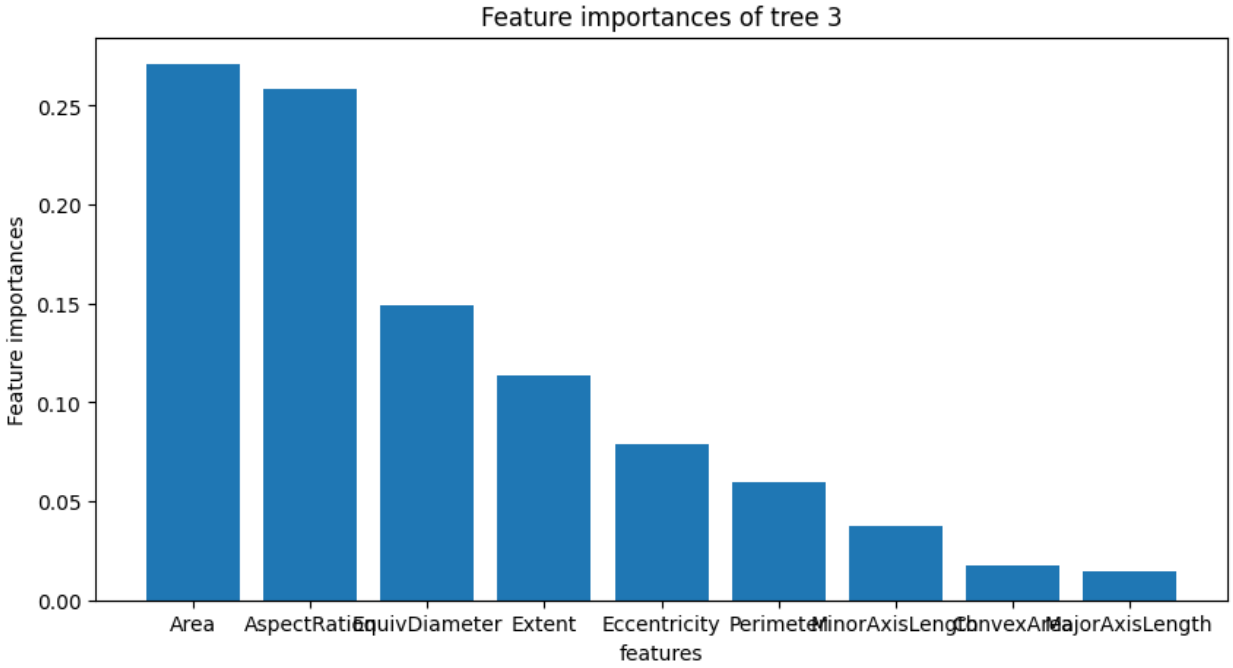
In the cell below, call your implementation of `plot_feature_importance()` and display a bar plot that shows the feature importance values for at least one extra tree in your tuned random forest from Q3.2.

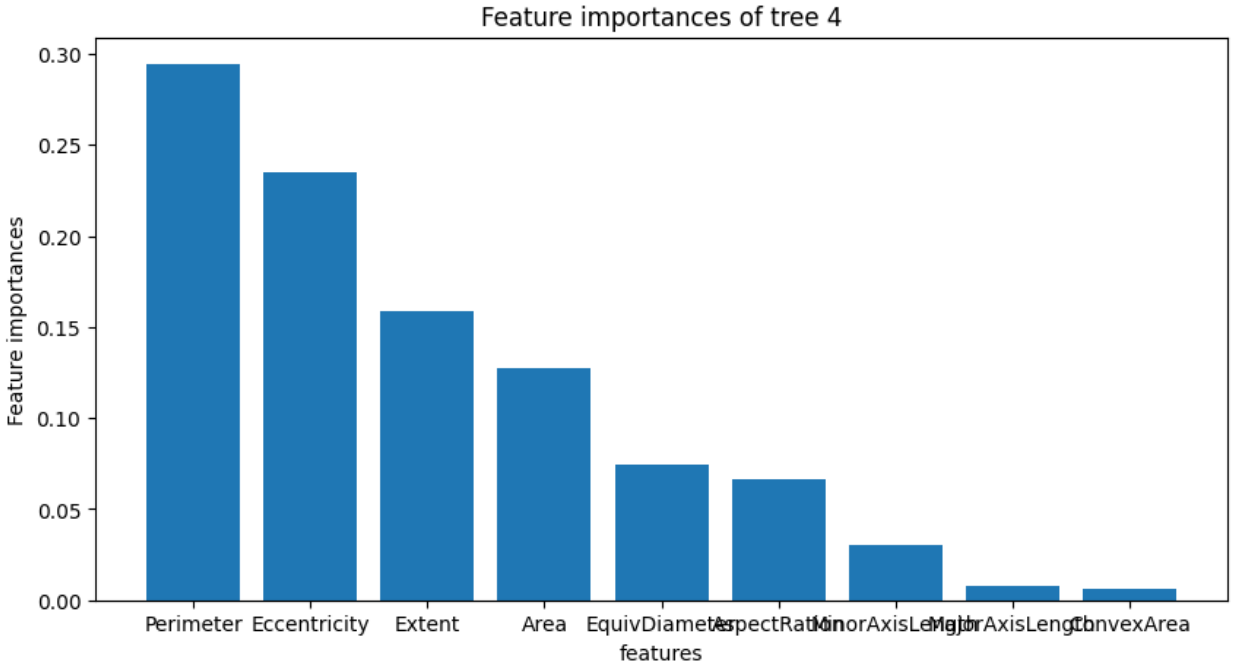
```
# TODO: Complete plot_feature_importance() in random_forest.py

random_forest.plot_feature_importance(X)
```

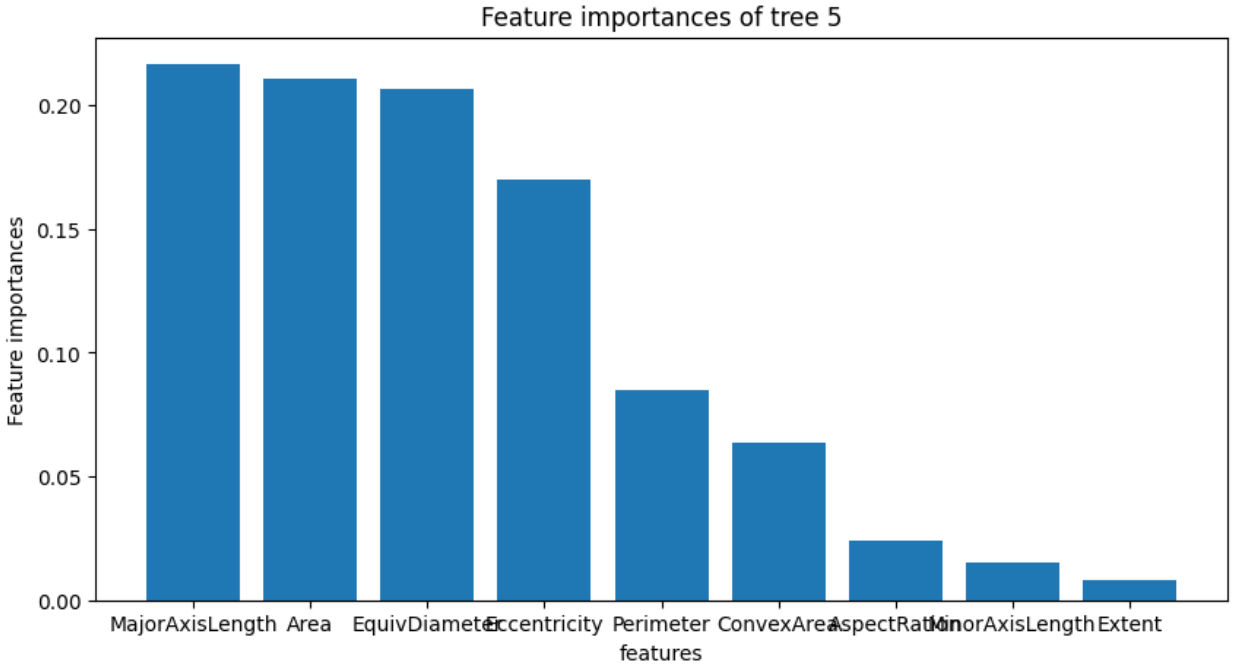


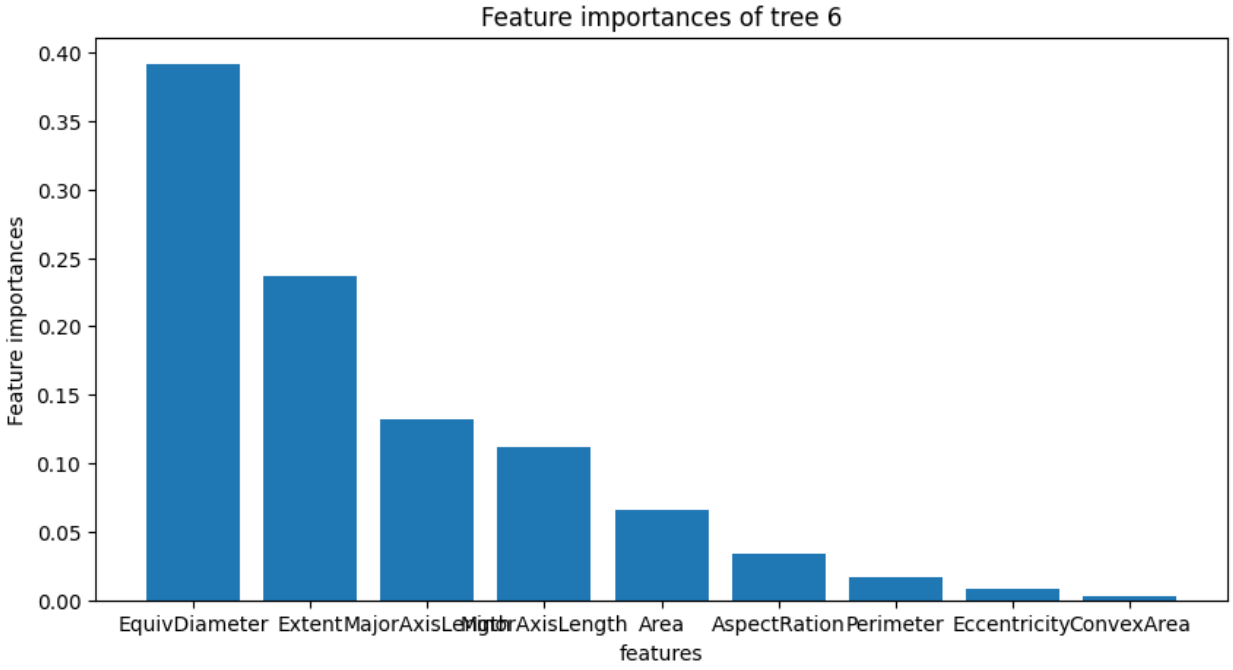

Feature importances of tree 1
Feature importances
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Eccentricity
ConvexArea
MajorAxisLength
Area
Extent
MinorAxisLength
Perimeter
AspectRatio
EquivDiameter
features

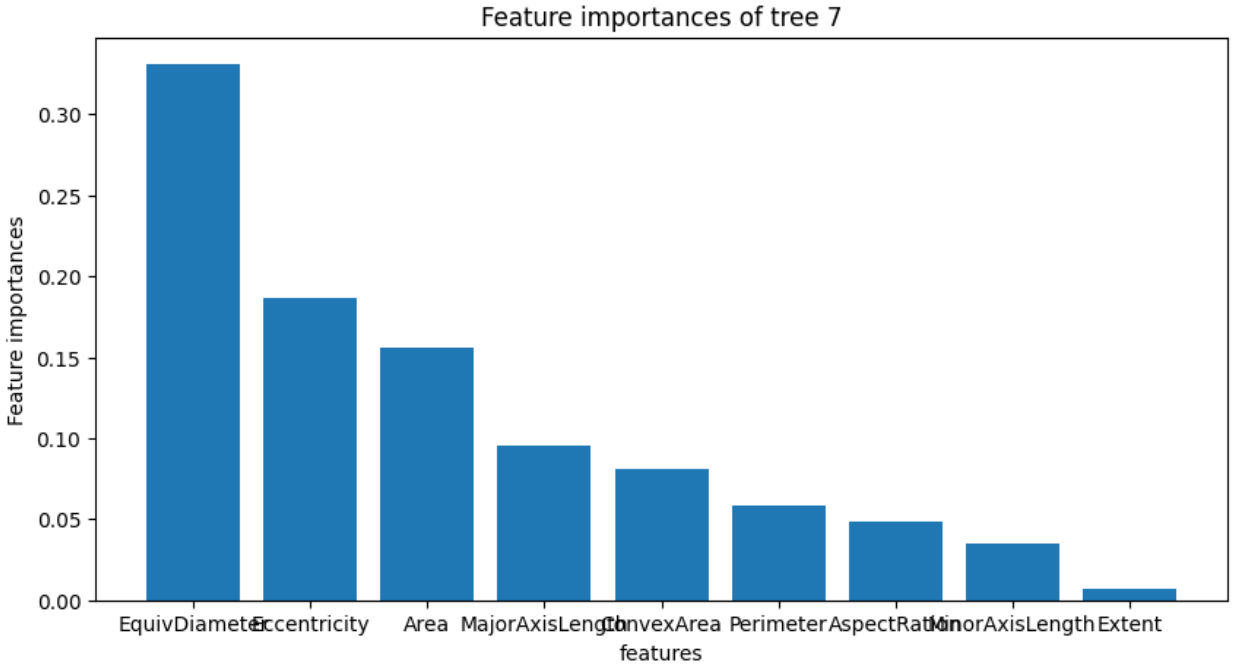

Feature importances of tree 2
Feature importances
0.25
0.20
0.15
0.10
0.05
0.00
Area
Extent
MinorAxisLength
Perimeter
AspectRatio
MajorAxisLength
Eccentricity
EquivDiameter
ConvexArea
features

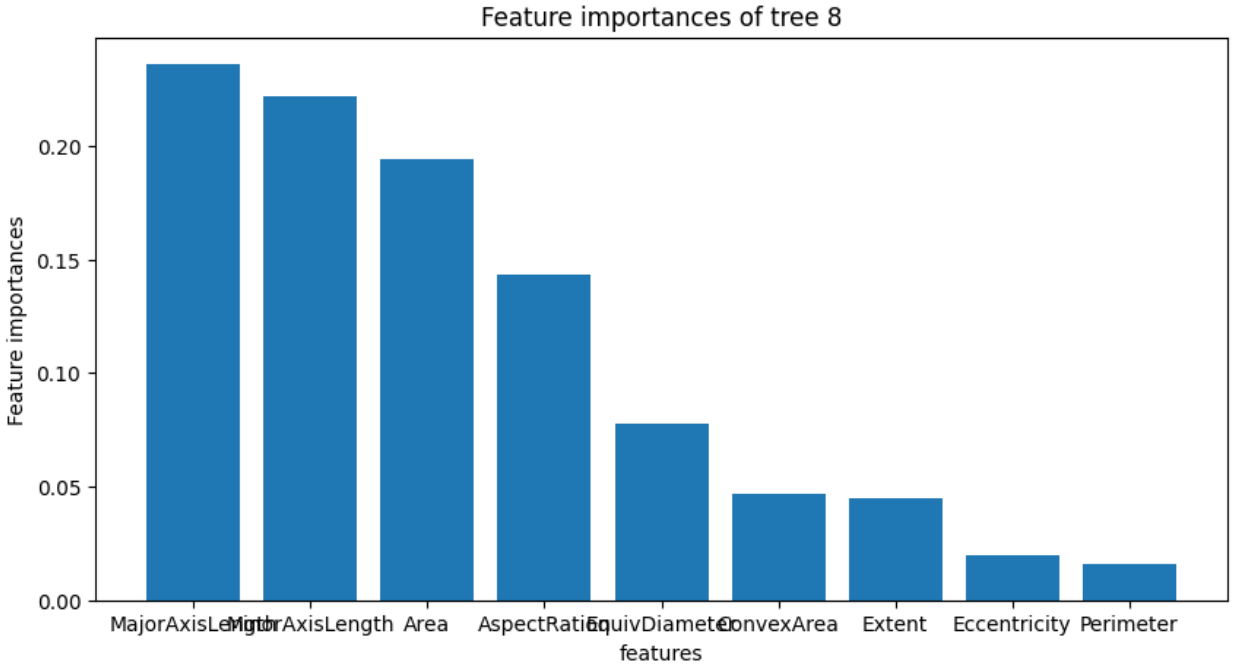

Feature importances of tree 3
Feature importances
0.25
0.20
0.15
0.10
0.05
0.00
Area
AspectRatio
EquivDiameter
Extent
Eccentricity
Perimeter
MinorAxisLength
ConvexArea
MajorAxisLength
features

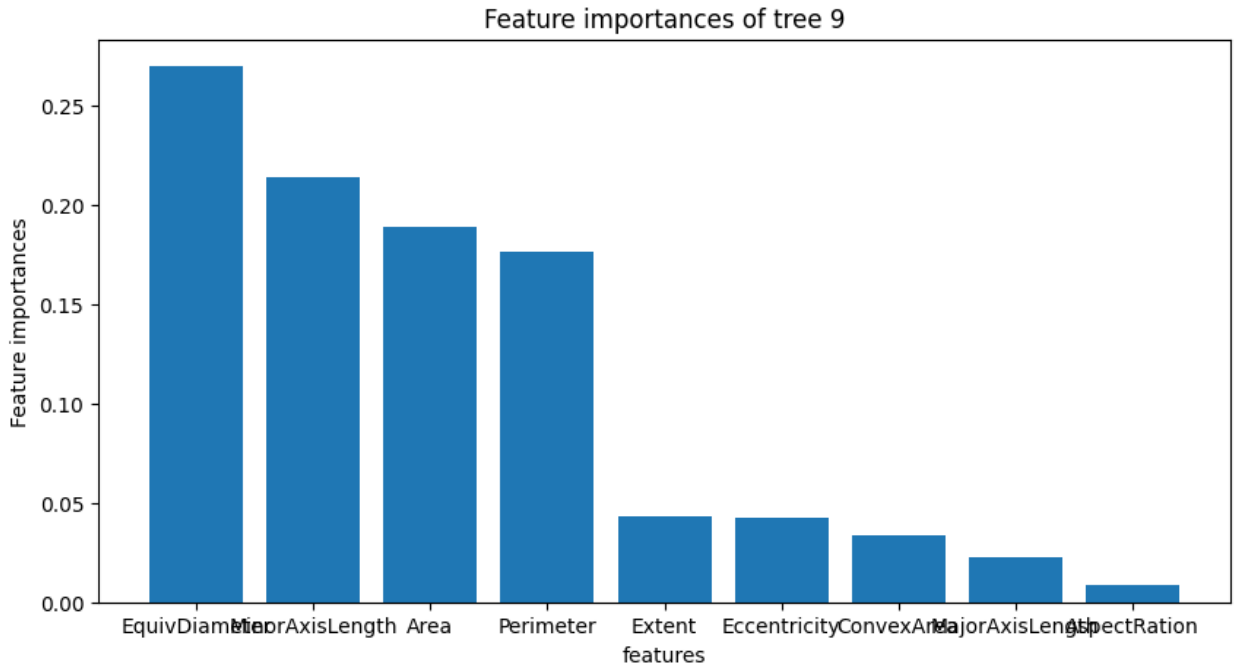

Feature importances of tree 4
Feature importances
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Perimeter
Eccentricity
Extent
Area
EquivDiameter
AspectRatio
MinorAxisLength
MajorAxisLength
ConvexArea
features

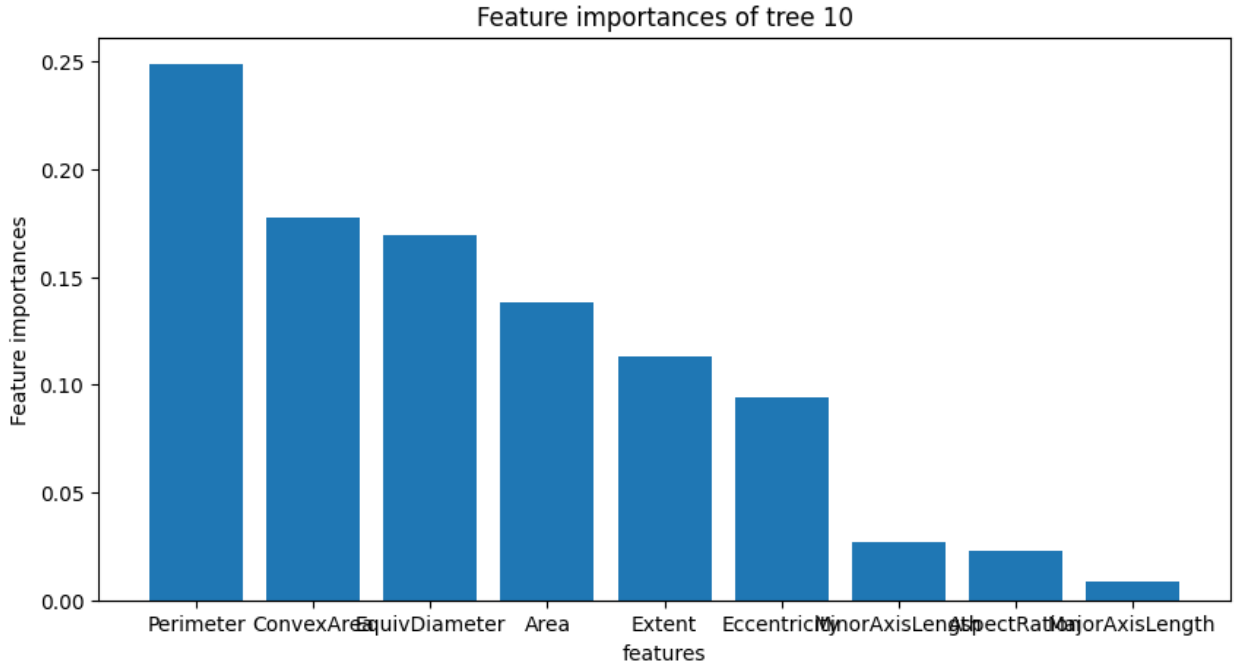

Feature importances of tree 5
Feature importances
0.00
0.05
0.10
0.15
0.20
MajorAxisLength
Area
EquivDiameter
Eccentricity
Perimeter
ConvexArea
AspectRation
MinorAxisLength
Extent
features


Feature importances of tree 6
Feature importances
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
EquivDiameter
Extent
MajorAxisLength
MinorAxisLength
Area
AspectRation
Perimeter
Eccentricity
ConvexArea
features


Feature importances of tree 7
Feature importances
0.30
0.25
0.20
0.15
0.10
0.05
0.00
EquivDiameter
Eccentricity
Area
MajorAxisLength
ConvexArea
Perimeter
AspectRatio
MinorAxisLength
Extent
features


Feature importances of tree 8
Feature importances
0.20
0.15
0.10
0.05
0.00
MajorAxisLength
MinorAxisLength
Area
AspectRatio
EquivDiameter
ConvexArea
Extent
Eccentricity
Perimeter
features






```
11
```

Note that there isn't one "correct" answer here. We simply want you to investigate how different features in your random forest contribute to predicting the target variable.

Also note that: the number of features can be different if you change max_features value since it ends up changing the number of features considered in bootstrapped datasets.

4: (Bonus for All) SVM [34 pts] **[W] [P]**

4.1 Fitting an SVM classifier by hand [24 pts] **[W]**

Consider a dataset with the following points in 2-dimensional space:

x_1	x_2	y
-1	-1	-1
-1	1	-1
1	-1	-1
2	2	1
2	3	1
1	3	1

Here, x_1 and x_2 are features and y is the label.

The max margin classifier has the formulation,

$$min||\theta||^2$$

$$s.t.\ y_i(x_i\theta+b) \geq 1\ \forall i$$

Hint: x_i are the suppport vectors. Margin is equal to $\frac{1}{||\theta||}$ and full margin is equal to $\frac{2}{||\theta||}$. You might find it useful to plot the points in a 2D plane. When calculating the θ you don't need to consider the bias term.

(1) Are the points linearly separable? Does adding the point $x=(2,1)$, $y=1$ change the separability? (2 pts)

(2) According to the max-margin formulation, find the separating hyperplane. Do not consider the new point from part 1 in your calculations for this current question or subsequent parts. (You should give some kind of explanation or calculation on how you found the hyperplane, you may solve this question graphically.) (4 pts)

(3) Find a vector parallel to the optimal vector θ. (Hint: Recall whether the optimal vector is parallel or perpendicular to the separating hyperplane.) (4 pts)

(4) Calculate the value of the margin (single-sided) achieved by this θ? (4 pts)

(5) Solve for θ, given that the margin is equal to $1/||\theta||$. (4 pts)

(6) If we remove one of the points from the original data the SVM solution might change. Find all such points which change the solution. (2 pts)

(7) Consider the optimization formulation stated above. Why do we want to optimzie $||\theta||^2$ instead of $||\theta||$? (2 pts)

(8) Plot the features x_1 and x_2, based on label y (use different color for different label), ignoring the hypothetical point mentioned in part (1). Please also included the separating hyperplane in the plot (4 pts)

Responses:

(1) Yes, the points are linearly separable. Adding the point (2, 1) does not change that

(2) $x + y$

(3)

(4)

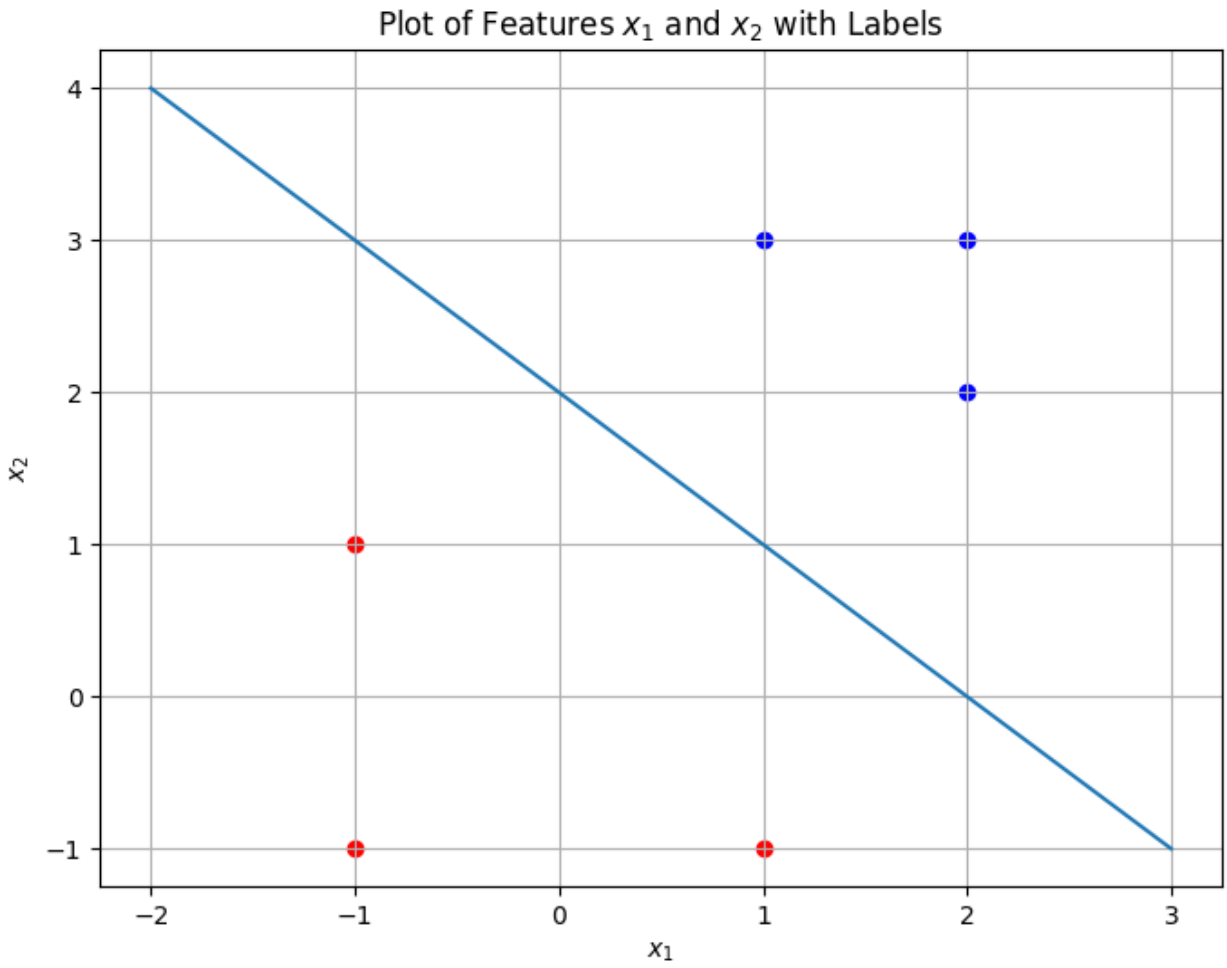
(5)

(6)

(7)

```
# TODO (question 8): plot the points listed in the table
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC

data = np.array([[-1, -1, -1],
                 [-1, 1, -1],
                 [1, -1, -1],
                 [2, 2, 1],
                 [2, 3, 1],
                 [1, 3, 1]])

# Separate features and labels
X = data[:, :-1]
y = data[:, -1]

# Fit the SVM model to get the separating hyperplane
model = SVC(kernel='linear')
model.fit(X, y)

# Plotting the points
plt.figure(figsize=(8, 6))

# Plotting points based on labels
plt.scatter(X[y == -1][:, 0], X[y == -1][:, 1], color='red')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='blue')

# Plotting the separating hyperplane
w = model.coef_[0]
b = model.intercept_[0]
x0 = np.linspace(-2, 3)
x1 = -w[0] / w[1] * x0 - b / w[1]
```

```
plt.plot(x0, x1)

plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.title('Plot of Features $x_1$ and $x_2$ with Labels')
plt.grid(True)
plt.show()
```



4.2 Feature Mapping [10 pts] **[P]**

Let's look at a dataset where the datapoint can't be classified with a good accuracy using a linear classifier. Run the below cell to generate the dataset.

We will also see what happens when we try to fit a linear classifier to the dataset.

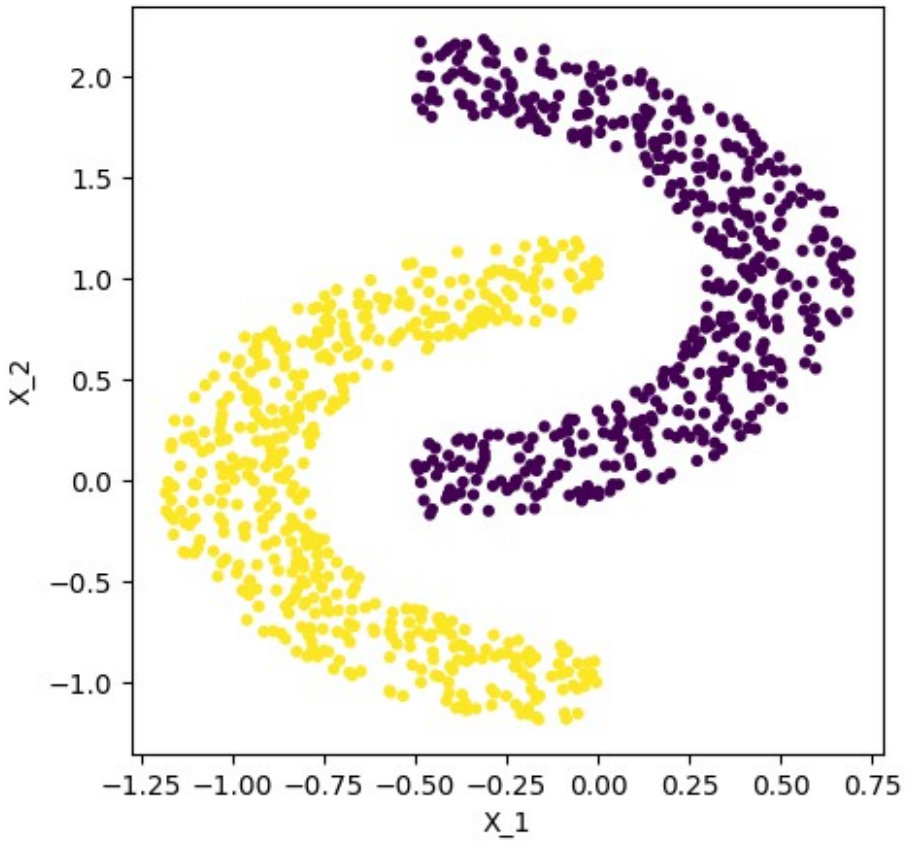
there are some suggestion readings:

https://see.stanford.edu/materials/aimlcs229/cs229-notes3.pdf

https://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf

https://www.sjsu.edu/faculty/guangliang.chen/Math251F18/lec6svm.pdf

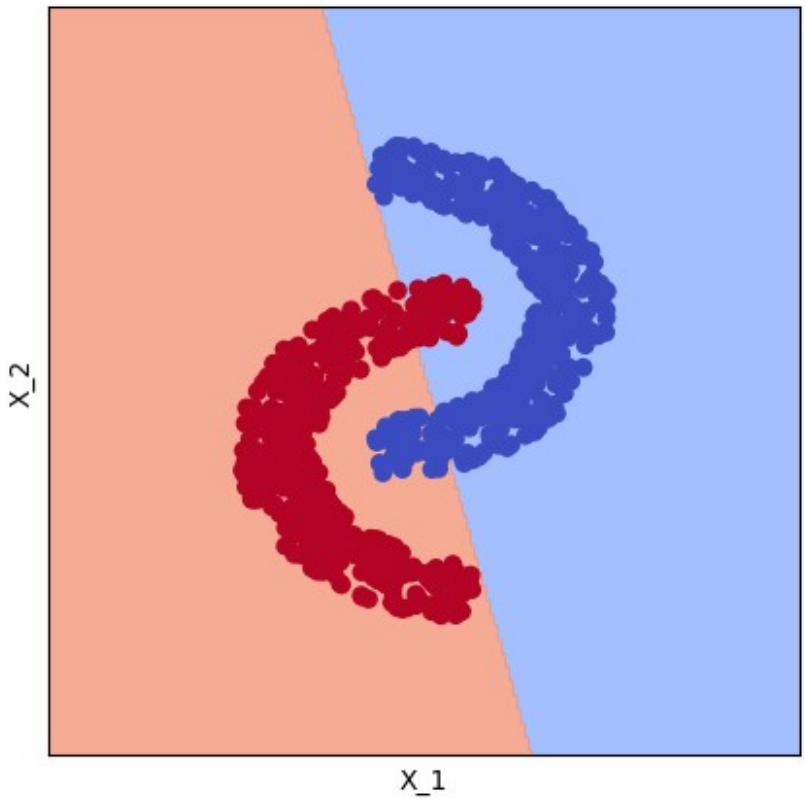
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################
# Generate dataset

random_state = 1

np.random.seed(0)
theta = np.linspace(0, 2 * np.pi, 1000)
r = np.random.uniform(0.8, 1.2, 1000)
X = np.column_stack([r * np.cos(theta), r * np.sin(theta)])
y = np.logical_or(theta < np.pi, theta >= 2 * np.pi)
X[y == 0, 0] += 1
X[y == 0, 1] += 0.5

R = np.array([[0, -1], [1, 0]])

X_rotated = X.dot(R.T)

X_train, X_test, y_train, y_test = train_test_split(
    X_rotated, y, test_size=0.20, random_state=random_state
)

f, ax = plt.subplots(nrows=1, ncols=1, figsize=(5, 5))
plt.scatter(X_rotated[:, 0], X_rotated[:, 1], c=y, marker="o", s=12)
plt.xlabel("X_1")
plt.ylabel("X_2")
plt.show()
```




```
################################
### DO NOT CHANGE THIS CELL ###
################################


def visualize_decision_boundary(X, y, feature_new=None, h=0.02):
    """
    You don't have to modify this function

    Function to vizualize decision boundary

    feature_new is a function to get X with additional features
    """
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx_1, xx_2 = np.meshgrid(np.arange(x1_min, x1_max, h),
np.arange(x2_min, x2_max, h))

    if X.shape[1] == 2:
        Z = svm_cls.predict(np.c_[xx_1.ravel(), xx_2.ravel()])
    else:
        X_conc = np.c_[xx_1.ravel(), xx_2.ravel()]
        X_new = feature_new(X_conc)
        Z = svm_cls.predict(X_new)
```

```
    Z = Z.reshape(xx_1.shape)

    f, ax = plt.subplots(nrows=1, ncols=1, figsize=(5, 5))
    plt.contourf(xx_1, xx_2, Z, cmap=plt.cm.coolwarm, alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
    plt.xlabel("X_1")
    plt.ylabel("X_2")
    plt.xlim(xx_1.min(), xx_1.max())
    plt.ylim(xx_2.min(), xx_2.max())
    plt.xticks(())
    plt.yticks(())

    plt.show()

################################
### DO NOT CHANGE THIS CELL ###
################################
# Try to fit a linear classifier to the dataset

svm_cls = svm.LinearSVC()
svm_cls.fit(X_train, y_train)
y_test_predicted = svm_cls.predict(X_test)

print("Accuracy on test dataset: {}".format(accuracy_score(y_test,
y_test_predicted)))

visualize_decision_boundary(X_train, y_train)
```

```
Accuracy on test dataset: 0.865
```



We can see that we need a non-linear boundary to be able to successfully classify data in this dataset. By mapping the current feature x to a higher space with more features, linear SVM could be performed on the features in the higher space to learn a non-linear decision boundary. In feature.py, modify create_nl_feature() to add additional features which can help classify in the above dataset. After creating the additional features use code in the further cells to see how well the features perform on the test set.

Note: You should get a test accuracy above 85%

Hint: Think of the shape of the decision boundary that would best separate the above points. What additional features could help map the linear boundary to the non-linear one? Look at this for a detailed analysis of doing the same for points separable with a circular boundary

TODO: Implement the create_nl_feature function in feature.py. There are many possible solutions to producing a decision boundary; think creatively!

```
################################
### DO NOT CHANGE THIS CELL ###
################################
from feature import create_nl_feature

X_new = create_nl_feature(X_rotated)
X_train, X_test, y_train, y_test = train_test_split(
    X_new, y, test_size=0.20, random_state=random_state
)
```

```
################################
### DO NOT CHANGE THIS CELL ###
################################
# Fit to the new features and vizualize the decision boundary
# You should get more than 90% accuracy on test set

svm_cls = svm.LinearSVC()
svm_cls.fit(X_train, y_train)
y_test_predicted = svm_cls.predict(X_test)

print("Accuracy on test dataset: {}".format(accuracy_score(y_test,
y_test_predicted)))

visualize_decision_boundary(X_train, y_train, create_nl_feature)
```

```
Accuracy on test dataset: 0.925
```