Name:

Darian Irani

Netid:

irani2

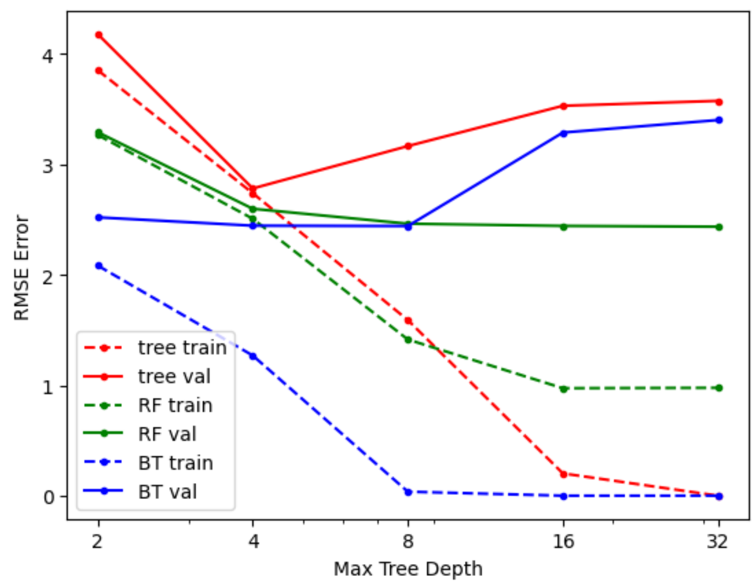
CS 441 - HW 4: Trees and MLPs

Complete the sections below. You do not need to fill out the checklist. **Do select all relevant pages in Gradescope.**

Total Points Claimed **[ ] / 170**

1. Model Complexity with Tree Regressors
    a. Depth vs. Error plot [ ] / 10
    b. Analysis [ ] / 20
2. MLPs with MNIST
    a. Loss Curves [ ] / 20
    b. Model Selection and Results [ ] / 20
3. Species Prediction
    a. Feature Analysis [ ] / 10
    b. Simple Rule [ ] / 10
    c. Model Design [ ] / 10
4. Stretch Goals
    a. Improve MNIST classification [ ] / 30
    b. A second simple rule [ ] / 10
    c. Positional encoding of RGB Image [ ] / 30

1. **Model Complexity with Tree Regressors**
   a. **Include your plot below.**



   b. **Analyze your results:**
   1. For a given max tree depth, which of regressor model (single tree, random forest, boosted tree) has the lowest bias (or most powerful)?

A low bias means that the model fits the training data more closely, hence the lowest RMSE on training data. From the graph, the BT regressor tends to have the lowest RMSE across most tree depths.

   2. For single regression trees, what tree depth achieves minimum validation error?

4

   3. A model "overfits" when increasing the complexity increases the validation error. Which model is least prone to overfitting? Why?

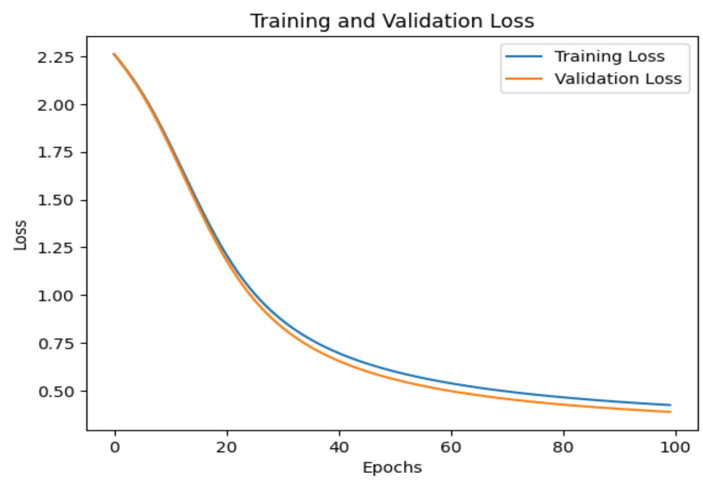
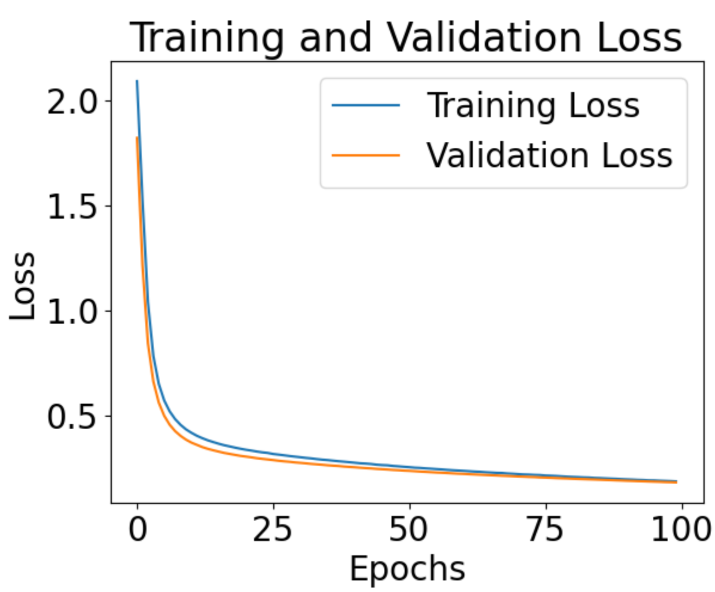
The RF model is least prone to overfitting. This is evident in the graph as this model shows the smallest increase in validation error as the complexity of the model increases. The gap between the RF training error and validation error remains relatively constant, suggesting that the model is generalizing well despite the increased complexity.

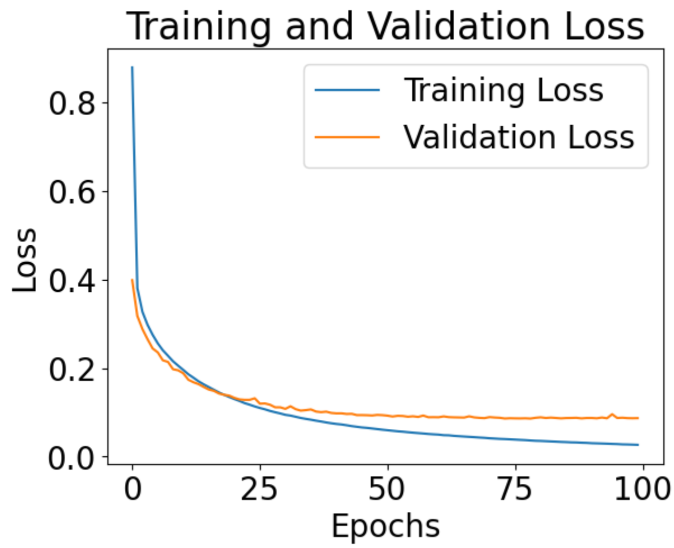
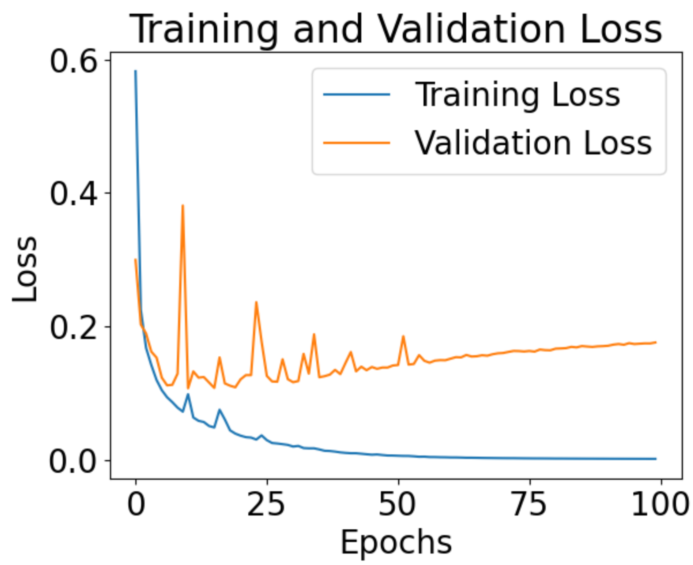
4. Do boosted trees seem to perform better with smaller or larger trees? Why?

For BT, as the tree depth increases, the training RMSE decreases and the validation RMSE increases, suggesting that the model's generalization to unseen data is getting worse (sign of overfitting). Therefore, BT perform better with smaller trees.

2. MLPs with MNIST

a. **Show the loss curves** for 3 learning rates (1E-2, 1E-1, 1E1) training for 100 epochs. An example of the loss curves is shown for LR=0.001.

Learning Rate	Loss Curve Plot
0.001	
0.01	

0.1	
1	

b. Model selection and results

Select the best hyperparameters (learning rate and number of epochs up to 100) based on minimizing the validation loss.

Learning Rate

0.1

Epochs

100

Report the losses and errors for the model trained with these hyperparameters:

Use scientific notation with one decimal place, eg: 1.5E-3

Training **Loss**	Validation **Loss**	
2.4E-2 ~ 0.0244	9.1E-2 ~ 0.0912	

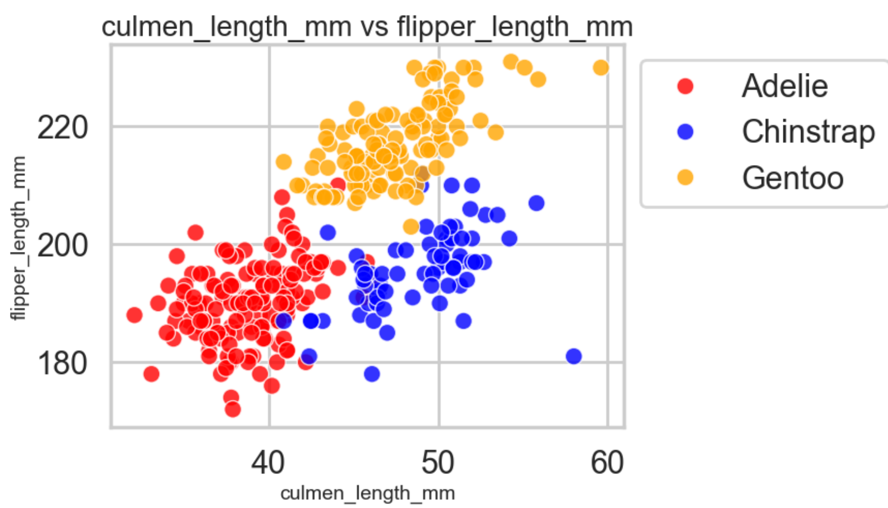
Show two decimal places for percent

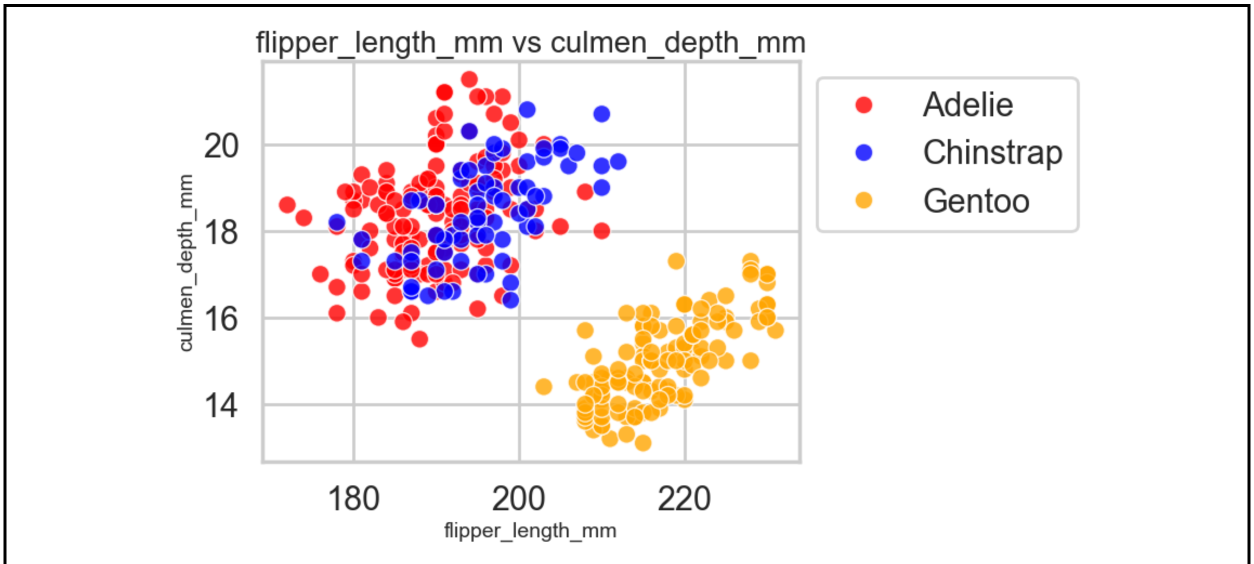
Training **Error** (%)	Validation **Error** (%)	Test **Error** (%)
0.44	2.67	2.43

3. Species Prediction

a. Visualization of Features

Include at least two scatterplots of pairs of features.

Visualization (labels should make clear which features are used)




You may extend the table if you have more results

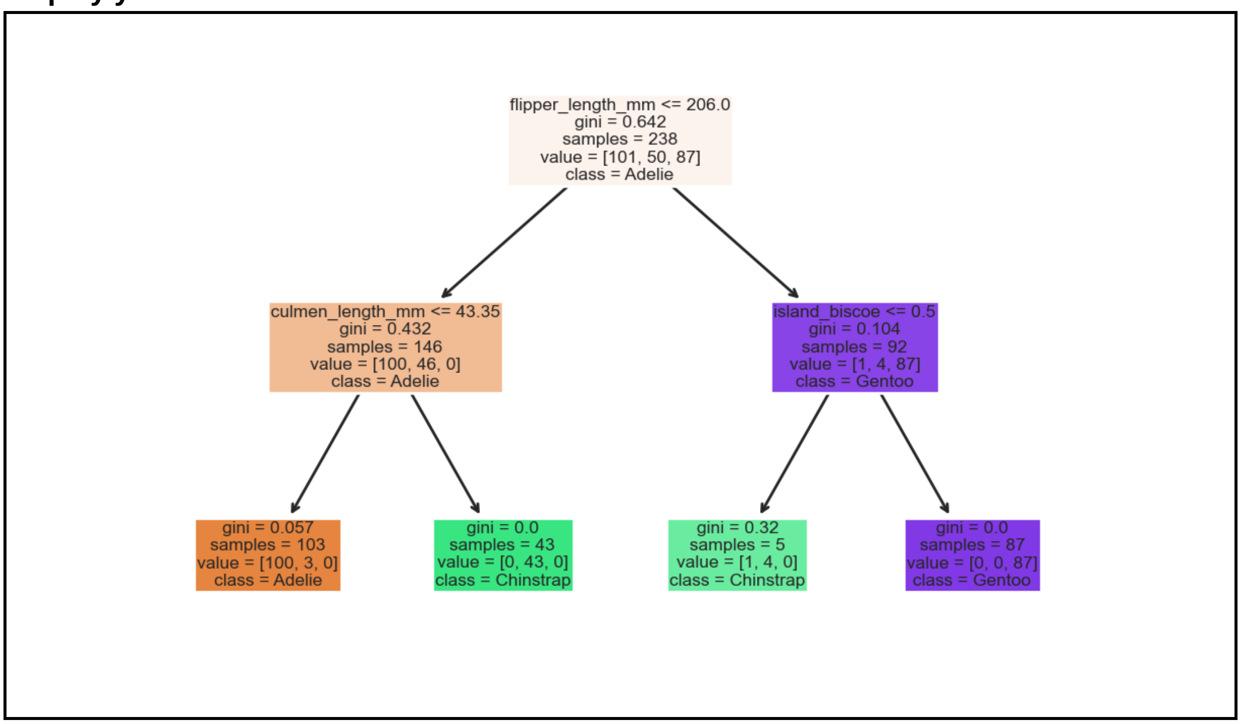
Of these three options, which two features (by themselves) are best able to classify the penguin species?

1. Culmen Depth + Flipper Length
2. Flipper Length + Culmen Length
3. Flipper Length + Body Mass

3. Flipper Length + Culmen Length

b. Simple rule to identify Gentoo

Display your decision tree with labeled features and classes.



Write down the simple two-part rule to identify Gentoo. For example, the format should be "If Mass > 3000 and Culmen Depth < 17, then species is Gentoo".

If…

flipper_length_mm <= 206

and

island_biscoe <= 0.5

then species is Gentoo.

Rule precision: fraction of penguins that satisfy this rule that are Gentoos (# gentoo predicted / # predicted)

0.89

Rule recall: fraction of all Gentoo penguins that are identified as Gentoo using this rule (# gentoo predicted / # gentoo)

0.92

c. Model Design

Describe the model that achieves best 5-fold cross-validation accuracy:

`RandomForestClassifier`, is an ensemble method that combines multiple decision trees to improve predictive accuracy and control over-fitting. Random forests perform well for classification tasks by averaging the predictions of individual trees, reducing the variance and bias. In this case, it achieved an impressive 99.4% cross-validation accuracy on the penguin dataset, indicating its strong performance in classifying Gentoo species.

5-fold Cross-Validation Accuracy: (xx.x%)

99.4%

3. Stretch Goals

a. Improve MNIST Classification Performance using MLPs

Report the classification val and test errors and details of your best method. Describe your approach and parameters. Feel free to change the MLP batch size, optimizer (e.g. try Adam), learning rate, number of epochs, hidden layer size, activation layer, or anything else.

Description and key parameters

Optimizer = Adam
Hidden layer(s) = 3
Learning rate = 0.001
Number of epochs = 300

Validation **Error** (%)	Test **Error** (%)
1.94	1.86

b. Find a second simple rule to identify Gentoo

Provide the second two-part rule here (that is substantially different from your first rule).

If…

Culmen_length_mm <= 42.35

and

Culmen_depth_mm <= 15.1

then species is Gentoo.

Rule precision: fraction of penguins that satisfy this rule that are Gentoos (# gentoo predicted / # predicted)

0.91

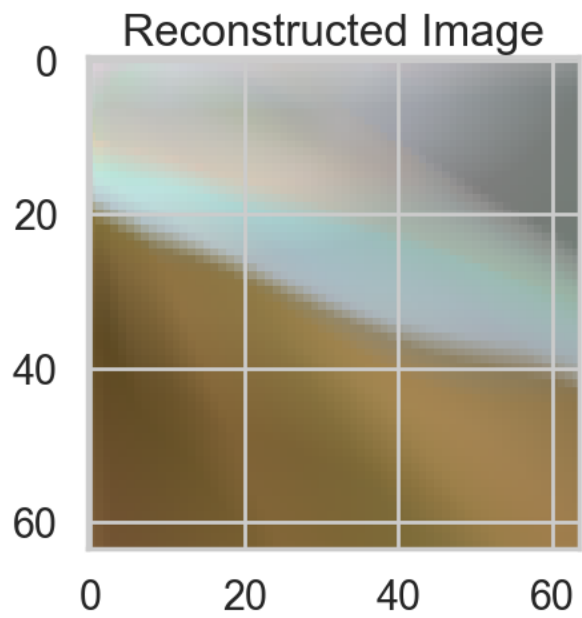
Rule recall: fraction of all Gentoo penguins that are identified as Gentoo using this rule (# gentoo predicted / # gentoo)

0.95

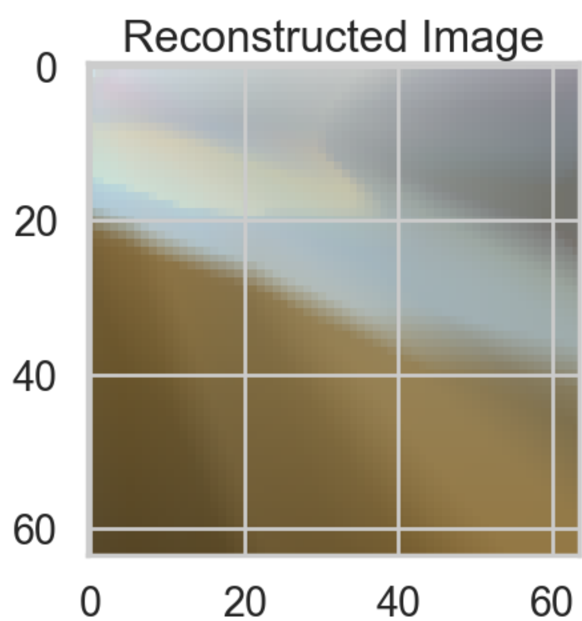
c. **Positional encoding**

Show the RGB image obtained by predicting directly from (x,y) and the image obtained by predicting from the positional encoding.

Input to network is (x,y)



Input to network is pos_enc(x, y)



Acknowledgments / Attribution

List any outside sources for code or improvement ideas or “None”.

https://towardsdatascience.com/master-positional-encoding-part-i-63c05d90a0c3

CS441: Applied ML - HW 4

Part 1: Model Complexity and Tree-based Regressors

One measure of a tree's complexity is the maximum tree depth. Train tree, random forest, and boosted tree regressors on the temperature regression task, using all default parameters except:

- **max_depth={2,4,8,16,32}**
- **random_state=0**
- **For random forest: max_features=1/3**

Measure train and val RMSE for each and plot them all on the same plot using the provided plot_depth_error function. You should have six lines (train/val for each model type), each with 5 data points (one for each max depth value). Include the plot and answer the analysis questions in the report.

In [50]:

```
import numpy as np
%matplotlib inline
from matplotlib import pyplot as plt

# load data (modify to match your data directory or comment)
def load_temp_data():
  datadir = "/Users/darian/Desktop/UIUC/Applied ML/HW4/Code/temperature_data.npz"
  T = np.load(datadir)
  x_train, y_train, x_val, y_val, x_test, y_test, dates_train, dates_val, dates_test, feature_to_city, feature_to_day = \
  T['x_train'], T['y_train'], T['x_val'], T['y_val'], T['x_test'], T['y_test'], T['dates_train'], T['dates_val'], T['dates_test'], T['feature_to_city'], T['feature_to_day']
  return (x_train, y_train, x_val, y_val, x_test, y_test, dates_train, dates_val, dates_test, feature_to_city, feature_to_day)

# plot one data point for listed cities and target temperature
def plot_temps(x, y, cities, feature_to_city, feature_to_day, target_date):
  nc = len(cities)
  ndays = 5
  xplot = np.array([-5,-4,-3,-2,-1])
  yplot = np.zeros((nc,ndays))
  for f in np.arange(len(x)):
    for c in np.arange(nc):
      if cities[c]==feature_to_city[f]:
        yplot[feature_to_day[f]+ndays,c] = x[f]
  plt.plot(xplot,yplot)
  plt.legend(cities)
  plt.plot(0, y, 'b*', markersize=10)
  plt.title('Predict Temp for Cleveland on ' + target_date)
  plt.xlabel('Day')
  plt.ylabel('Avg Temp (C)')
  plt.show()

# load data
(x_train, y_train, x_val, y_val, x_test, y_test, dates_train, dates_val, dates_test, feature_to_city, feature_to_day) = load_temp_data()
```

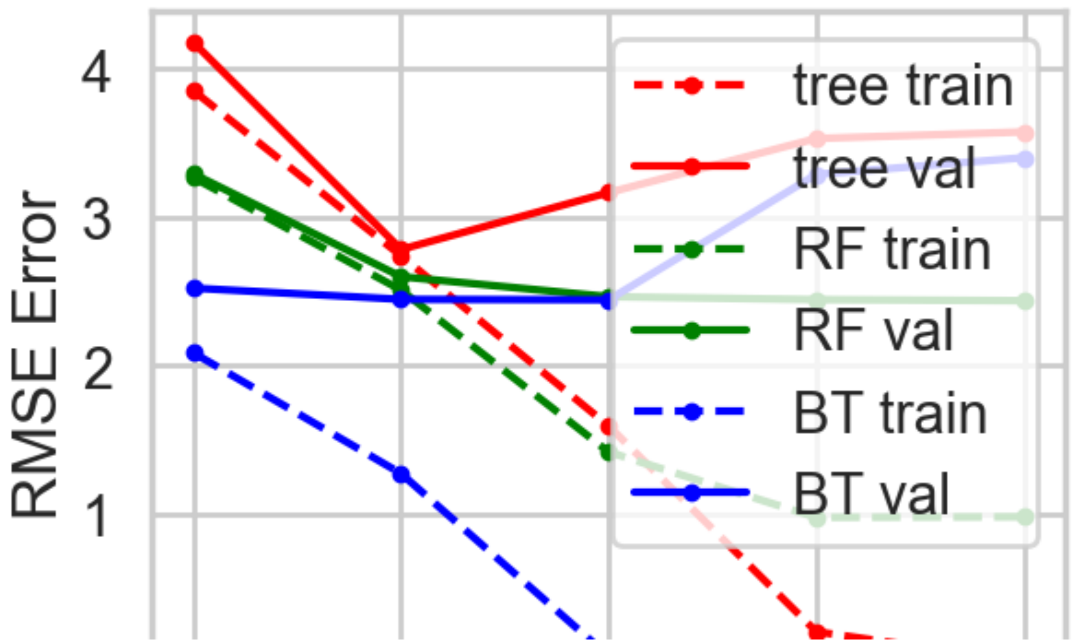
In [51]:

```
# to plot the errors
def plot_depth_error(max_depths, tree_train_err, tree_val_err, rf_train_err, rf_val_err, bt_train_err, bt_val_err):
  plt.figure()
  plt.semilogx(max_depths, tree_train_err, 'r.--',label='tree train')
  plt.semilogx(max_depths, tree_val_err, 'r.-', label='tree val')
  plt.semilogx(max_depths, rf_train_err, 'g.--',label='RF train')
```

```python
    plt.ylabel('RMSE Error')
    plt.xlabel('Max Tree Depth')
    plt.xticks(max_depths, max_depths)
    plt.legend()
    plt.rcParams.update({'font.size': 20})
    plt.show()
```

In [53]:

```python
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
import numpy as np

max_depths = [2,4,8,16,32]

tree_train_rmse, tree_val_rmse = [], []
rf_train_rmse, rf_val_rmse = [], []
bt_train_rmse, bt_val_rmse = [], []

for depth in max_depths:
    # Decision Tree
    tree_model = DecisionTreeRegressor(random_state=0, max_depth=depth)
    tree_model.fit(x_train, y_train)
    tree_train_rmse.append(np.sqrt(mean_squared_error(y_train, tree_model.predict(x_trai
n))))
    tree_val_rmse.append(np.sqrt(mean_squared_error(y_val, tree_model.predict(x_val))))

    # Random Forest
    rf_model = RandomForestRegressor(random_state=0, max_depth=depth, max_features=1/3)
    rf_model.fit(x_train, y_train)
    rf_train_rmse.append(np.sqrt(mean_squared_error(y_train, rf_model.predict(x_train)))
)
    rf_val_rmse.append(np.sqrt(mean_squared_error(y_val, rf_model.predict(x_val))))

    # Boosted Trees
    bt_model = GradientBoostingRegressor(random_state=0, max_depth=depth)
    bt_model.fit(x_train, y_train)
    bt_train_rmse.append(np.sqrt(mean_squared_error(y_train, bt_model.predict(x_train)))
)
    bt_val_rmse.append(np.sqrt(mean_squared_error(y_val, bt_model.predict(x_val))))

plot_depth_error(max_depths, tree_train_rmse, tree_val_rmse, rf_train_rmse, rf_val_rmse,
bt_train_rmse, bt_val_rmse)
```



Part 2: MLPs with MNIST

For this part, you will want to use a GPU to improve runtime. Google Colab provides limited free GPU acceleration to all users. Go to Runtime and change Runtime Type to GPU. This will reset your compute node, so do it before starting to run other cells.

See [Tips] for detailed guidance on this problem.

First, use PyTorch to implement a Multilayer Perceptron network with one hidden layer (size 64) with ReLU activation. Set the network to minimize cross-entropy loss, which is the negative log probability of the training labels given the training features. This objective function takes unnormalized logits as inputs.

Do not use MLP in sklearn for this HW - use Torch .

In [54]:

```
# initialization code
import numpy as np
from keras.datasets import mnist
%matplotlib inline
from matplotlib import pyplot as plt
from scipy import stats
import torch
import torch.nn as nn

def load_mnist():
  '''
  Loads, reshapes, and normalizes the data
  '''
  (x_train, y_train), (x_test, y_test) = mnist.load_data() # loads MNIST data
  x_train = np.reshape(x_train, (len(x_train), 28*28))  # reformat to 768-d vectors
  x_test = np.reshape(x_test, (len(x_test), 28*28))
  maxval = x_train.max()
  x_train = x_train/maxval  # normalize values to range from 0 to 1
  x_test = x_test/maxval
  return (x_train, y_train), (x_test, y_test)

def display_mnist(x, subplot_rows=1, subplot_cols=1):
  '''
  Displays one or more examples in a row or a grid
  '''
  if subplot_rows>1 or subplot_cols>1:
    fig, ax = plt.subplots(subplot_rows, subplot_cols, figsize=(15,15))
    for i in np.arange(len(x)):
      ax[i].imshow(np.reshape(x[i], (28,28)), cmap='gray')
      ax[i].axis('off')
  else:
      plt.imshow(np.reshape(x, (28,28)), cmap='gray')
      plt.axis('off')
  plt.show()
```

In [55]:

```
device = torch.device("mps")
print(device) # make sure you're using GPU instance
```

```
mps
```

2a

after the epoch is complete. The mean training loss can either be computed after the epoch is complete, or, for efficiency, computed using the losses accumulated during the training of the epoch. Plot the training and validation losses using the display_error_curves function.

In [56]:

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

# create train/val split
ntrain = 50000
x_val = x_train[ntrain:].copy()
y_val = y_train[ntrain:].copy()
x_train = x_train[:ntrain]
y_train = y_train[:ntrain]
```

In [57]:

```
def display_error_curves(training_losses, validation_losses):
  """
  Plots the training and validation loss curves
  training_losses and validation_losses should be lists or arrays of the same length
  """
  num_epochs = len(training_losses)

  plt.plot(range(num_epochs), training_losses, label="Training Loss")
  plt.plot(range(num_epochs), validation_losses, label="Validation Loss")

  # Add in a title and axes labels
  plt.title('Training and Validation Loss')
  plt.xlabel('Epochs')
  plt.ylabel('Loss')

  # Display the plot
  plt.legend(loc='best')
  plt.show()
```

In [58]:

```
import torch
import torch.nn as nn

class MLP(nn.Module):
    def __init__(self, input_size, output_size, hidden_size=64):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

In [59]:

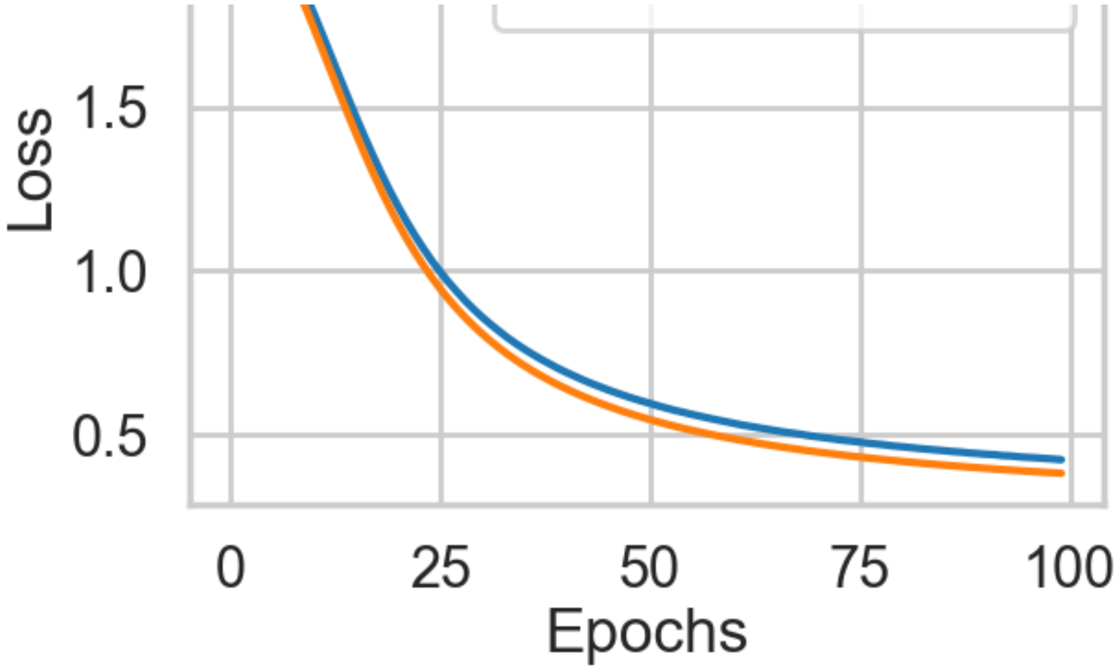
```
from torch.utils.data import DataLoader, TensorDataset
import tqdm


# This is a possible function definition for training MLP, but feel free to change it
# You may also want to create helper functions, e.g. for computing loss or prediction

def train_MLP_mnist(train_loader, val_loader, lr, num_epochs=100):
  '''
  Train a MLP
  Input: train_loader and val_loader are dataloaders for the training and
  val data, respectively. lr is the learning rate, and the network will
```

```
  input_size = 28*28
  hidden_size = 64
  output_size = 10

  mlp = MLP(input_size, output_size).to(device)
  loss_func = nn.CrossEntropyLoss()
  optimizer = torch.optim.SGD(mlp.parameters(), lr=lr)

  training_losses = []
  validation_losses = []

  for epoch in range(num_epochs):
      mlp.train()
      running_loss = 0.0
      for inputs, labels in tqdm.tqdm(train_loader):
          inputs, labels = inputs.to(device), labels.to(device)
          optimizer.zero_grad()
          outputs = mlp(inputs)
          loss = loss_func(outputs, labels)
          loss.backward()
          optimizer.step()

          running_loss += loss.item()
      training_losses.append(running_loss / len(train_loader))

      mlp.eval()
      running_val_loss = 0.0
      with torch.no_grad():
          for inputs, labels in val_loader:
              inputs, labels = inputs.to(device), labels.to(device)
              outputs = mlp(inputs)
              val_loss = loss_func(outputs, labels)
              running_val_loss += val_loss.item()
      validation_losses.append(running_val_loss / len(val_loader))

  display_error_curves(training_losses, validation_losses)

  return mlp


def evaluate_MLP(mlp, loader):
  ''' Computes loss and error rate given your mlp model and data loader'''
  N = 0
  acc = 0
  running_loss = 0
  loss_func = torch.nn.CrossEntropyLoss()
  with torch.set_grad_enabled(False):
    for i, data in enumerate(loader, 0):

      # Get inputs
      inputs, targets = data
      N += len(targets)

      # Perform forward pass
      outputs = mlp(inputs.to(device))

      # Compute sum of correct labels
      y_pred = np.argmax(outputs.cpu().numpy(), axis=1)
      y_gt = targets.numpy()
      acc += np.sum(y_pred==y_gt)

      # Compute loss
      running_loss += loss_func(outputs, targets.to(device)).item()*len(targets)

  running_loss /= N
  acc /= N
```

```python
# Code for running experiments

print(device) # make sure you're using GPU instance
torch.manual_seed(0) # to avoid randomness, but if you wanted to create an ensemble, you should not use a manual seed

# TODO (set up dataloaders, and call training function)

device = torch.device("mps")

tensor_x_train = torch.FloatTensor(x_train)  # Assuming x_train is scaled [0, 1]
tensor_y_train = torch.LongTensor(y_train)
tensor_x_val = torch.FloatTensor(x_val)
tensor_y_val = torch.LongTensor(y_val)

# Convert test data to DataLoader
tensor_x_test = torch.FloatTensor(x_test)
tensor_y_test = torch.LongTensor(y_test)
test_dataset = torch.utils.data.TensorDataset(tensor_x_test, tensor_y_test)
test_loader = DataLoader(dataset=test_dataset, batch_size=256, shuffle=False)

train_dataset = torch.utils.data.TensorDataset(tensor_x_train, tensor_y_train)
val_dataset = torch.utils.data.TensorDataset(tensor_x_val, tensor_y_val)

train_loader = DataLoader(dataset=train_dataset, batch_size=256, shuffle=True)
val_loader = DataLoader(dataset=val_dataset, batch_size=256, shuffle=False)

learning_rates = [0.001, 0.01, 0.1, 1.0]
for lr in learning_rates:
  print(f"Loss curve with lr: {lr}")
  train_MLP_mnist(train_loader, val_loader, lr=lr, num_epochs=100)
```

```
mps
Loss curve with lr: 0.001
```

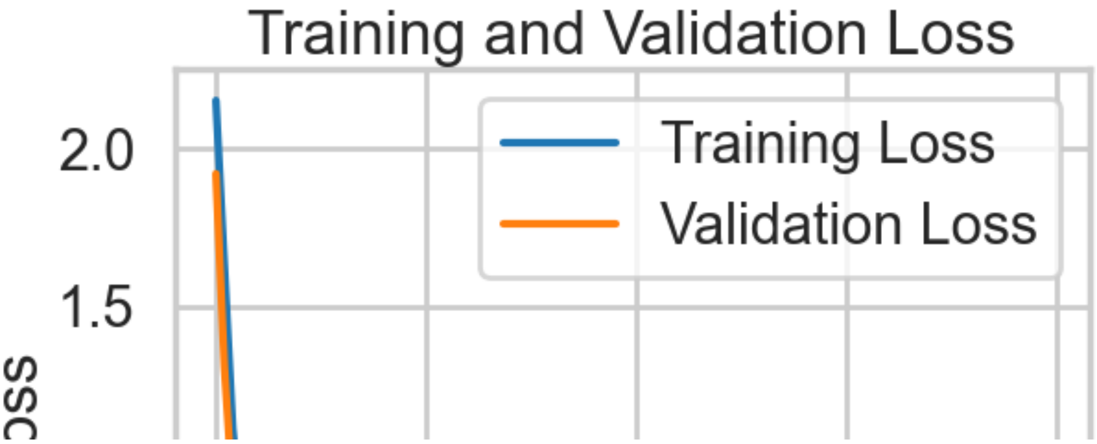
```
100%|██████████| 196/196 [00:00<00:00, 215.74it/s]
100%|██████████| 196/196 [00:00<00:00, 265.35it/s]
100%|██████████| 196/196 [00:00<00:00, 287.61it/s]
100%|██████████| 196/196 [00:00<00:00, 291.21it/s]
100%|██████████| 196/196 [00:00<00:00, 289.00it/s]
100%|██████████| 196/196 [00:00<00:00, 280.45it/s]
100%|██████████| 196/196 [00:00<00:00, 293.76it/s]
100%|██████████| 196/196 [00:00<00:00, 297.55it/s]
100%|██████████| 196/196 [00:00<00:00, 292.96it/s]
100%|██████████| 196/196 [00:00<00:00, 294.70it/s]
100%|██████████| 196/196 [00:00<00:00, 296.05it/s]
100%|██████████| 196/196 [00:00<00:00, 294.97it/s]
100%|██████████| 196/196 [00:00<00:00, 292.89it/s]
100%|██████████| 196/196 [00:00<00:00, 298.75it/s]
100%|██████████| 196/196 [00:00<00:00, 295.70it/s]
100%|██████████| 196/196 [00:00<00:00, 295.93it/s]
100%|██████████| 196/196 [00:00<00:00, 290.38it/s]
100%|██████████| 196/196 [00:00<00:00, 250.55it/s]
100%|██████████| 196/196 [00:00<00:00, 295.32it/s]
100%|██████████| 196/196 [00:00<00:00, 300.57it/s]
100%|██████████| 196/196 [00:00<00:00, 262.77it/s]
100%|██████████| 196/196 [00:00<00:00, 296.26it/s]
100%|██████████| 196/196 [00:00<00:00, 300.26it/s]
100%|██████████| 196/196 [00:00<00:00, 297.36it/s]
100%|██████████| 196/196 [00:00<00:00, 295.86it/s]
100%|██████████| 196/196 [00:00<00:00, 307.67it/s]
100%|██████████| 196/196 [00:00<00:00, 291.18it/s]
100%|██████████| 196/196 [00:00<00:00, 300.04it/s]
100%|██████████| 196/196 [00:00<00:00, 304.66it/s]
100%|██████████| 196/196 [00:00<00:00, 293.20it/s]
100%|██████████| 196/196 [00:00<00:00, 289.96it/s]
100%|██████████| 196/196 [00:00<00:00, 251.19it/s]
100%|██████████| 196/196 [00:00<00:00, 296.40it/s]
```

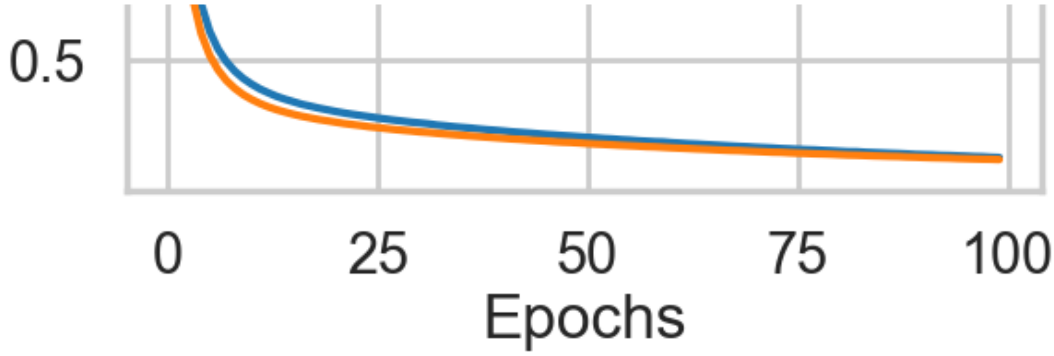


Loss curve with lr: 0.01

```
100%|██████████| 196/196 [00:00<00:00, 306.89it/s]
100%|██████████| 196/196 [00:00<00:00, 299.11it/s]
100%|██████████| 196/196 [00:00<00:00, 254.83it/s]
100%|██████████| 196/196 [00:00<00:00, 306.50it/s]
100%|██████████| 196/196 [00:00<00:00, 304.39it/s]
100%|██████████| 196/196 [00:00<00:00, 303.36it/s]
100%|██████████| 196/196 [00:00<00:00, 306.48it/s]
100%|██████████| 196/196 [00:00<00:00, 306.89it/s]
100%|██████████| 196/196 [00:00<00:00, 303.95it/s]
100%|██████████| 196/196 [00:00<00:00, 308.17it/s]
100%|██████████| 196/196 [00:00<00:00, 309.80it/s]
100%|██████████| 196/196 [00:00<00:00, 306.90it/s]
100%|██████████| 196/196 [00:00<00:00, 305.85it/s]
100%|██████████| 196/196 [00:00<00:00, 309.62it/s]
100%|██████████| 196/196 [00:00<00:00, 314.20it/s]
100%|██████████| 196/196 [00:00<00:00, 305.45it/s]
100%|██████████| 196/196 [00:00<00:00, 255.34it/s]
100%|██████████| 196/196 [00:00<00:00, 298.26it/s]
100%|██████████| 196/196 [00:00<00:00, 296.89it/s]
100%|██████████| 196/196 [00:00<00:00, 309.47it/s]
100%|██████████| 196/196 [00:00<00:00, 305.50it/s]
100%|██████████| 196/196 [00:00<00:00, 308.35it/s]
100%|██████████| 196/196 [00:00<00:00, 302.46it/s]
100%|██████████| 196/196 [00:00<00:00, 293.82it/s]
100%|██████████| 196/196 [00:00<00:00, 307.23it/s]
100%|██████████| 196/196 [00:00<00:00, 306.79it/s]
100%|██████████| 196/196 [00:00<00:00, 304.12it/s]
100%|██████████| 196/196 [00:00<00:00, 306.24it/s]
100%|██████████| 196/196 [00:00<00:00, 313.15it/s]
100%|██████████| 196/196 [00:00<00:00, 311.27it/s]
100%|██████████| 196/196 [00:00<00:00, 311.89it/s]
100%|██████████| 196/196 [00:00<00:00, 256.08it/s]
100%|██████████| 196/196 [00:00<00:00, 303.10it/s]
100%|██████████| 196/196 [00:00<00:00, 303.93it/s]
100%|██████████| 196/196 [00:00<00:00, 300.04it/s]
100%|██████████| 196/196 [00:00<00:00, 297.70it/s]
100%|██████████| 196/196 [00:00<00:00, 304.25it/s]
100%|██████████| 196/196 [00:00<00:00, 291.95it/s]
100%|██████████| 196/196 [00:00<00:00, 299.05it/s]
100%|██████████| 196/196 [00:00<00:00, 307.88it/s]
100%|██████████| 196/196 [00:00<00:00, 308.42it/s]
100%|██████████| 196/196 [00:00<00:00, 301.62it/s]
100%|██████████| 196/196 [00:00<00:00, 302.58it/s]
```

```
100%|██████████| 196/196 [00:00<00:00, 303.15it/s]
100%|██████████| 196/196 [00:00<00:00, 307.96it/s]
100%|██████████| 196/196 [00:00<00:00, 309.39it/s]
100%|██████████| 196/196 [00:00<00:00, 311.79it/s]
100%|██████████| 196/196 [00:00<00:00, 303.79it/s]
100%|██████████| 196/196 [00:00<00:00, 307.25it/s]
100%|██████████| 196/196 [00:00<00:00, 308.00it/s]
100%|██████████| 196/196 [00:00<00:00, 300.06it/s]
100%|██████████| 196/196 [00:00<00:00, 300.78it/s]
100%|██████████| 196/196 [00:00<00:00, 304.21it/s]
100%|██████████| 196/196 [00:00<00:00, 306.68it/s]
100%|██████████| 196/196 [00:00<00:00, 305.82it/s]
100%|██████████| 196/196 [00:00<00:00, 251.82it/s]
100%|██████████| 196/196 [00:00<00:00, 308.93it/s]
100%|██████████| 196/196 [00:00<00:00, 301.89it/s]
100%|██████████| 196/196 [00:00<00:00, 295.63it/s]
100%|██████████| 196/196 [00:00<00:00, 299.43it/s]
100%|██████████| 196/196 [00:00<00:00, 285.99it/s]
100%|██████████| 196/196 [00:00<00:00, 304.77it/s]
100%|██████████| 196/196 [00:00<00:00, 302.32it/s]
100%|██████████| 196/196 [00:00<00:00, 301.66it/s]
100%|██████████| 196/196 [00:00<00:00, 301.44it/s]
100%|██████████| 196/196 [00:00<00:00, 305.63it/s]
100%|██████████| 196/196 [00:00<00:00, 305.08it/s]
100%|██████████| 196/196 [00:00<00:00, 302.71it/s]
100%|██████████| 196/196 [00:00<00:00, 303.21it/s]
100%|██████████| 196/196 [00:00<00:00, 305.26it/s]
100%|██████████| 196/196 [00:00<00:00, 308.87it/s]
100%|██████████| 196/196 [00:00<00:00, 304.88it/s]
100%|██████████| 196/196 [00:00<00:00, 303.40it/s]
100%|██████████| 196/196 [00:00<00:00, 294.02it/s]
100%|██████████| 196/196 [00:00<00:00, 298.13it/s]
100%|██████████| 196/196 [00:00<00:00, 298.74it/s]
100%|██████████| 196/196 [00:00<00:00, 297.03it/s]
100%|██████████| 196/196 [00:00<00:00, 304.23it/s]
100%|██████████| 196/196 [00:00<00:00, 303.21it/s]
100%|██████████| 196/196 [00:00<00:00, 303.58it/s]
100%|██████████| 196/196 [00:00<00:00, 299.41it/s]
100%|██████████| 196/196 [00:00<00:00, 307.76it/s]
100%|██████████| 196/196 [00:00<00:00, 302.89it/s]
100%|██████████| 196/196 [00:00<00:00, 301.13it/s]
100%|██████████| 196/196 [00:00<00:00, 258.87it/s]
100%|██████████| 196/196 [00:00<00:00, 303.40it/s]
100%|██████████| 196/196 [00:00<00:00, 307.50it/s]
100%|██████████| 196/196 [00:00<00:00, 300.45it/s]
100%|██████████| 196/196 [00:00<00:00, 306.64it/s]
100%|██████████| 196/196 [00:00<00:00, 305.05it/s]
100%|██████████| 196/196 [00:00<00:00, 306.68it/s]
100%|██████████| 196/196 [00:00<00:00, 297.51it/s]
100%|██████████| 196/196 [00:00<00:00, 295.98it/s]
100%|██████████| 196/196 [00:00<00:00, 308.63it/s]
100%|██████████| 196/196 [00:00<00:00, 307.92it/s]
100%|██████████| 196/196 [00:00<00:00, 309.56it/s]
```

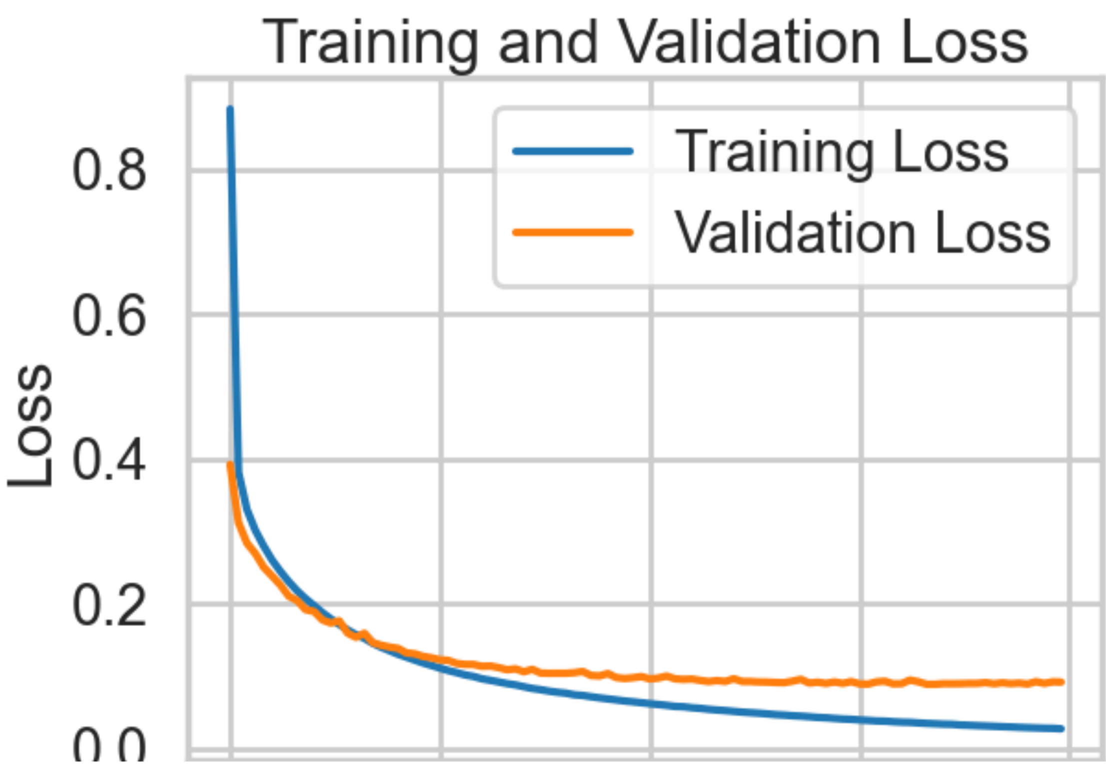




Loss curve with lr: 0.1

```
100%|██████████| 196/196 [00:00<00:00, 301.33it/s]
100%|██████████| 196/196 [00:00<00:00, 299.09it/s]
100%|██████████| 196/196 [00:00<00:00, 257.28it/s]
100%|██████████| 196/196 [00:00<00:00, 308.75it/s]
100%|██████████| 196/196 [00:00<00:00, 298.75it/s]
100%|██████████| 196/196 [00:00<00:00, 304.33it/s]
100%|██████████| 196/196 [00:00<00:00, 302.61it/s]
100%|██████████| 196/196 [00:00<00:00, 319.54it/s]
100%|██████████| 196/196 [00:00<00:00, 308.33it/s]
100%|██████████| 196/196 [00:00<00:00, 311.97it/s]
100%|██████████| 196/196 [00:00<00:00, 310.83it/s]
100%|██████████| 196/196 [00:00<00:00, 313.43it/s]
100%|██████████| 196/196 [00:00<00:00, 312.17it/s]
100%|██████████| 196/196 [00:00<00:00, 310.02it/s]
100%|██████████| 196/196 [00:00<00:00, 319.31it/s]
100%|██████████| 196/196 [00:00<00:00, 311.26it/s]
100%|██████████| 196/196 [00:00<00:00, 263.52it/s]
100%|██████████| 196/196 [00:00<00:00, 297.98it/s]
100%|██████████| 196/196 [00:00<00:00, 299.42it/s]
100%|██████████| 196/196 [00:00<00:00, 303.32it/s]
100%|██████████| 196/196 [00:00<00:00, 307.73it/s]
100%|██████████| 196/196 [00:00<00:00, 310.93it/s]
100%|██████████| 196/196 [00:00<00:00, 307.64it/s]
100%|██████████| 196/196 [00:00<00:00, 303.08it/s]
100%|██████████| 196/196 [00:00<00:00, 313.23it/s]
100%|██████████| 196/196 [00:00<00:00, 306.00it/s]
100%|██████████| 196/196 [00:00<00:00, 306.69it/s]
100%|██████████| 196/196 [00:00<00:00, 304.50it/s]
100%|██████████| 196/196 [00:00<00:00, 313.46it/s]
100%|██████████| 196/196 [00:00<00:00, 305.28it/s]
100%|██████████| 196/196 [00:00<00:00, 251.21it/s]
100%|██████████| 196/196 [00:00<00:00, 309.71it/s]
100%|██████████| 196/196 [00:00<00:00, 304.74it/s]
100%|██████████| 196/196 [00:00<00:00, 286.97it/s]
100%|██████████| 196/196 [00:00<00:00, 311.94it/s]
100%|██████████| 196/196 [00:00<00:00, 305.90it/s]
100%|██████████| 196/196 [00:00<00:00, 300.87it/s]
100%|██████████| 196/196 [00:00<00:00, 299.69it/s]
100%|██████████| 196/196 [00:00<00:00, 297.73it/s]
100%|██████████| 196/196 [00:00<00:00, 301.73it/s]
100%|██████████| 196/196 [00:00<00:00, 278.62it/s]
100%|██████████| 196/196 [00:00<00:00, 298.57it/s]
100%|██████████| 196/196 [00:00<00:00, 299.44it/s]
100%|██████████| 196/196 [00:00<00:00, 300.97it/s]
100%|██████████| 196/196 [00:00<00:00, 297.12it/s]
100%|██████████| 196/196 [00:00<00:00, 298.65it/s]
100%|██████████| 196/196 [00:00<00:00, 291.00it/s]
100%|██████████| 196/196 [00:00<00:00, 304.57it/s]
100%|██████████| 196/196 [00:00<00:00, 297.40it/s]
100%|██████████| 196/196 [00:00<00:00, 302.94it/s]
100%|██████████| 196/196 [00:00<00:00, 297.25it/s]
100%|██████████| 196/196 [00:00<00:00, 295.56it/s]
100%|██████████| 196/196 [00:00<00:00, 305.00it/s]
100%|██████████| 196/196 [00:00<00:00, 298.27it/s]
```

```
100%|██████████| 196/196 [00:00<00:00, 294.70it/s]
100%|██████████| 196/196 [00:00<00:00, 254.93it/s]
100%|██████████| 196/196 [00:00<00:00, 290.89it/s]
100%|██████████| 196/196 [00:00<00:00, 309.43it/s]
100%|██████████| 196/196 [00:00<00:00, 297.59it/s]
100%|██████████| 196/196 [00:00<00:00, 300.48it/s]
100%|██████████| 196/196 [00:00<00:00, 301.64it/s]
100%|██████████| 196/196 [00:00<00:00, 300.51it/s]
100%|██████████| 196/196 [00:00<00:00, 298.57it/s]
100%|██████████| 196/196 [00:00<00:00, 296.97it/s]
100%|██████████| 196/196 [00:00<00:00, 308.69it/s]
100%|██████████| 196/196 [00:00<00:00, 294.27it/s]
100%|██████████| 196/196 [00:00<00:00, 307.36it/s]
100%|██████████| 196/196 [00:00<00:00, 300.01it/s]
100%|██████████| 196/196 [00:00<00:00, 302.85it/s]
100%|██████████| 196/196 [00:00<00:00, 245.03it/s]
100%|██████████| 196/196 [00:00<00:00, 301.60it/s]
100%|██████████| 196/196 [00:00<00:00, 299.65it/s]
100%|██████████| 196/196 [00:00<00:00, 301.91it/s]
100%|██████████| 196/196 [00:00<00:00, 301.49it/s]
100%|██████████| 196/196 [00:00<00:00, 307.31it/s]
100%|██████████| 196/196 [00:00<00:00, 299.48it/s]
100%|██████████| 196/196 [00:00<00:00, 296.84it/s]
100%|██████████| 196/196 [00:00<00:00, 298.85it/s]
100%|██████████| 196/196 [00:00<00:00, 306.80it/s]
100%|██████████| 196/196 [00:00<00:00, 299.40it/s]
100%|██████████| 196/196 [00:00<00:00, 297.93it/s]
100%|██████████| 196/196 [00:00<00:00, 301.89it/s]
100%|██████████| 196/196 [00:00<00:00, 287.92it/s]
100%|██████████| 196/196 [00:00<00:00, 258.69it/s]
100%|██████████| 196/196 [00:00<00:00, 302.28it/s]
100%|██████████| 196/196 [00:00<00:00, 299.91it/s]
100%|██████████| 196/196 [00:00<00:00, 300.00it/s]
100%|██████████| 196/196 [00:00<00:00, 301.50it/s]
100%|██████████| 196/196 [00:00<00:00, 300.91it/s]
100%|██████████| 196/196 [00:00<00:00, 299.31it/s]
100%|██████████| 196/196 [00:00<00:00, 299.10it/s]
100%|██████████| 196/196 [00:00<00:00, 299.67it/s]
100%|██████████| 196/196 [00:00<00:00, 299.93it/s]
100%|██████████| 196/196 [00:00<00:00, 301.00it/s]
100%|██████████| 196/196 [00:00<00:00, 296.41it/s]
100%|██████████| 196/196 [00:00<00:00, 302.30it/s]
```

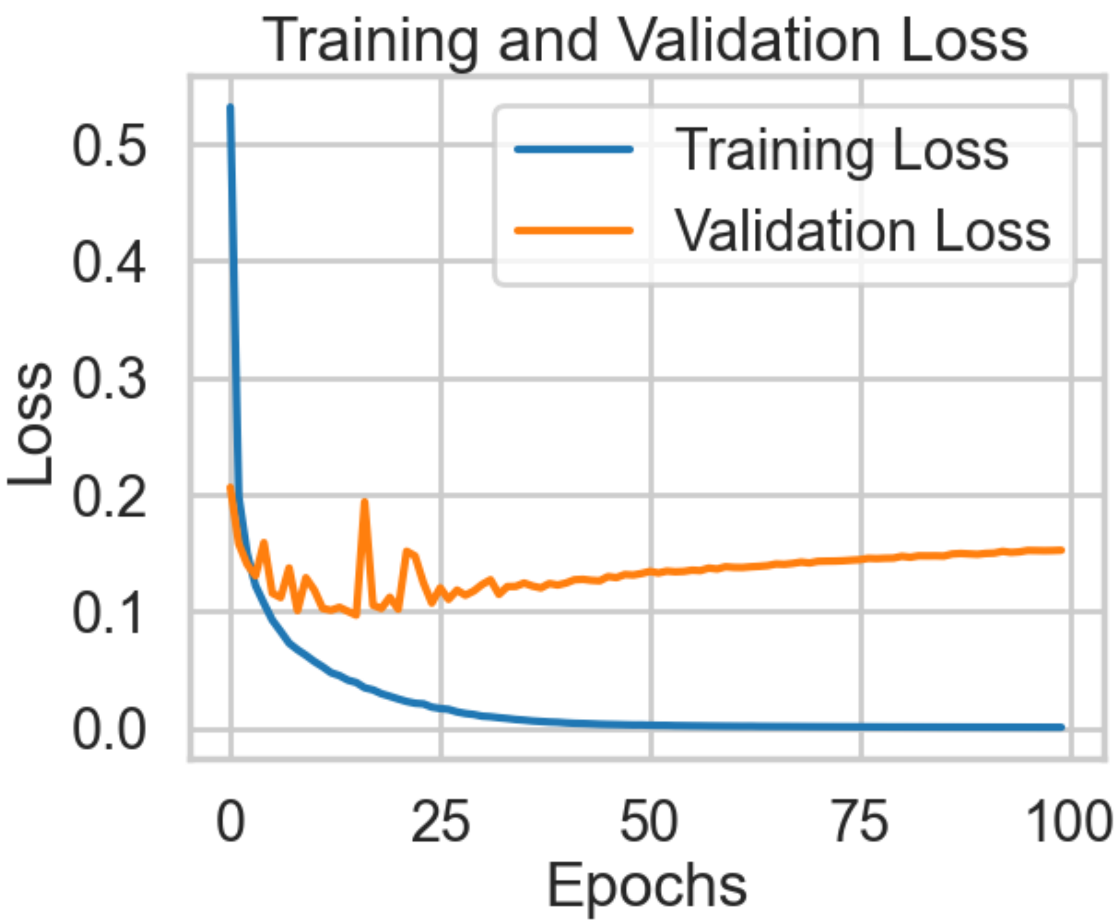


Epochs

Loss curve with lr: 1.0

```
100%|██████████| 196/196 [00:00<00:00, 280.82it/s]
100%|██████████| 196/196 [00:00<00:00, 301.00it/s]
100%|██████████| 196/196 [00:00<00:00, 301.71it/s]
100%|██████████| 196/196 [00:00<00:00, 302.36it/s]
100%|██████████| 196/196 [00:00<00:00, 288.72it/s]
100%|██████████| 196/196 [00:00<00:00, 296.85it/s]
100%|██████████| 196/196 [00:00<00:00, 305.16it/s]
100%|██████████| 196/196 [00:00<00:00, 302.04it/s]
100%|██████████| 196/196 [00:00<00:00, 303.07it/s]
100%|██████████| 196/196 [00:00<00:00, 308.24it/s]
100%|██████████| 196/196 [00:00<00:00, 308.39it/s]
100%|██████████| 196/196 [00:00<00:00, 301.01it/s]
100%|██████████| 196/196 [00:00<00:00, 295.56it/s]
100%|██████████| 196/196 [00:00<00:00, 300.06it/s]
100%|██████████| 196/196 [00:00<00:00, 306.27it/s]
100%|██████████| 196/196 [00:00<00:00, 292.69it/s]
100%|██████████| 196/196 [00:00<00:00, 246.21it/s]
100%|██████████| 196/196 [00:00<00:00, 223.54it/s]
100%|██████████| 196/196 [00:00<00:00, 246.72it/s]
100%|██████████| 196/196 [00:00<00:00, 294.04it/s]
100%|██████████| 196/196 [00:00<00:00, 294.74it/s]
100%|██████████| 196/196 [00:00<00:00, 302.12it/s]
100%|██████████| 196/196 [00:00<00:00, 296.97it/s]
100%|██████████| 196/196 [00:00<00:00, 303.13it/s]
100%|██████████| 196/196 [00:00<00:00, 300.90it/s]
100%|██████████| 196/196 [00:00<00:00, 305.88it/s]
100%|██████████| 196/196 [00:00<00:00, 295.63it/s]
100%|██████████| 196/196 [00:00<00:00, 303.36it/s]
100%|██████████| 196/196 [00:00<00:00, 312.06it/s]
100%|██████████| 196/196 [00:00<00:00, 300.89it/s]
100%|██████████| 196/196 [00:00<00:00, 256.50it/s]
100%|██████████| 196/196 [00:00<00:00, 297.89it/s]
100%|██████████| 196/196 [00:00<00:00, 308.19it/s]
100%|██████████| 196/196 [00:00<00:00, 293.39it/s]
100%|██████████| 196/196 [00:00<00:00, 300.87it/s]
100%|██████████| 196/196 [00:00<00:00, 299.06it/s]
100%|██████████| 196/196 [00:00<00:00, 290.25it/s]
100%|██████████| 196/196 [00:00<00:00, 302.00it/s]
100%|██████████| 196/196 [00:00<00:00, 301.05it/s]
100%|██████████| 196/196 [00:00<00:00, 289.06it/s]
100%|██████████| 196/196 [00:00<00:00, 298.42it/s]
100%|██████████| 196/196 [00:00<00:00, 299.02it/s]
100%|██████████| 196/196 [00:00<00:00, 297.48it/s]
100%|██████████| 196/196 [00:00<00:00, 303.91it/s]
100%|██████████| 196/196 [00:00<00:00, 245.87it/s]
100%|██████████| 196/196 [00:00<00:00, 305.33it/s]
100%|██████████| 196/196 [00:00<00:00, 294.36it/s]
100%|██████████| 196/196 [00:00<00:00, 307.71it/s]
100%|██████████| 196/196 [00:00<00:00, 283.11it/s]
100%|██████████| 196/196 [00:00<00:00, 303.67it/s]
100%|██████████| 196/196 [00:00<00:00, 299.58it/s]
100%|██████████| 196/196 [00:00<00:00, 296.20it/s]
100%|██████████| 196/196 [00:00<00:00, 305.00it/s]
100%|██████████| 196/196 [00:00<00:00, 291.48it/s]
100%|██████████| 196/196 [00:00<00:00, 293.67it/s]
100%|██████████| 196/196 [00:00<00:00, 305.16it/s]
100%|██████████| 196/196 [00:00<00:00, 297.43it/s]
100%|██████████| 196/196 [00:00<00:00, 300.17it/s]
100%|██████████| 196/196 [00:00<00:00, 302.32it/s]
100%|██████████| 196/196 [00:00<00:00, 297.03it/s]
100%|██████████| 196/196 [00:00<00:00, 294.52it/s]
100%|██████████| 196/196 [00:00<00:00, 301.19it/s]
100%|██████████| 196/196 [00:00<00:00, 298.65it/s]
100%|██████████| 196/196 [00:00<00:00, 293.73it/s]
```

```
100%|██████████| 196/196 [00:00<00:00, 299.48it/s]
100%|██████████| 196/196 [00:00<00:00, 302.26it/s]
100%|██████████| 196/196 [00:00<00:00, 296.49it/s]
100%|██████████| 196/196 [00:00<00:00, 297.15it/s]
100%|██████████| 196/196 [00:00<00:00, 301.74it/s]
100%|██████████| 196/196 [00:00<00:00, 251.13it/s]
100%|██████████| 196/196 [00:00<00:00, 280.73it/s]
100%|██████████| 196/196 [00:00<00:00, 251.31it/s]
100%|██████████| 196/196 [00:00<00:00, 267.16it/s]
100%|██████████| 196/196 [00:00<00:00, 292.62it/s]
100%|██████████| 196/196 [00:00<00:00, 273.06it/s]
100%|██████████| 196/196 [00:00<00:00, 286.15it/s]
100%|██████████| 196/196 [00:00<00:00, 263.53it/s]
100%|██████████| 196/196 [00:00<00:00, 278.71it/s]
100%|██████████| 196/196 [00:00<00:00, 290.02it/s]
100%|██████████| 196/196 [00:00<00:00, 286.19it/s]
100%|██████████| 196/196 [00:00<00:00, 289.54it/s]
100%|██████████| 196/196 [00:00<00:00, 289.32it/s]
100%|██████████| 196/196 [00:00<00:00, 291.52it/s]
100%|██████████| 196/196 [00:00<00:00, 230.11it/s]
100%|██████████| 196/196 [00:00<00:00, 262.53it/s]
100%|██████████| 196/196 [00:00<00:00, 279.09it/s]
100%|██████████| 196/196 [00:00<00:00, 287.61it/s]
100%|██████████| 196/196 [00:00<00:00, 275.16it/s]
100%|██████████| 196/196 [00:00<00:00, 295.81it/s]
100%|██████████| 196/196 [00:00<00:00, 287.86it/s]
100%|██████████| 196/196 [00:00<00:00, 287.99it/s]
100%|██████████| 196/196 [00:00<00:00, 292.50it/s]
100%|██████████| 196/196 [00:00<00:00, 289.32it/s]
100%|██████████| 196/196 [00:00<00:00, 290.03it/s]
100%|██████████| 196/196 [00:00<00:00, 261.72it/s]
100%|██████████| 196/196 [00:00<00:00, 286.55it/s]
```



2b

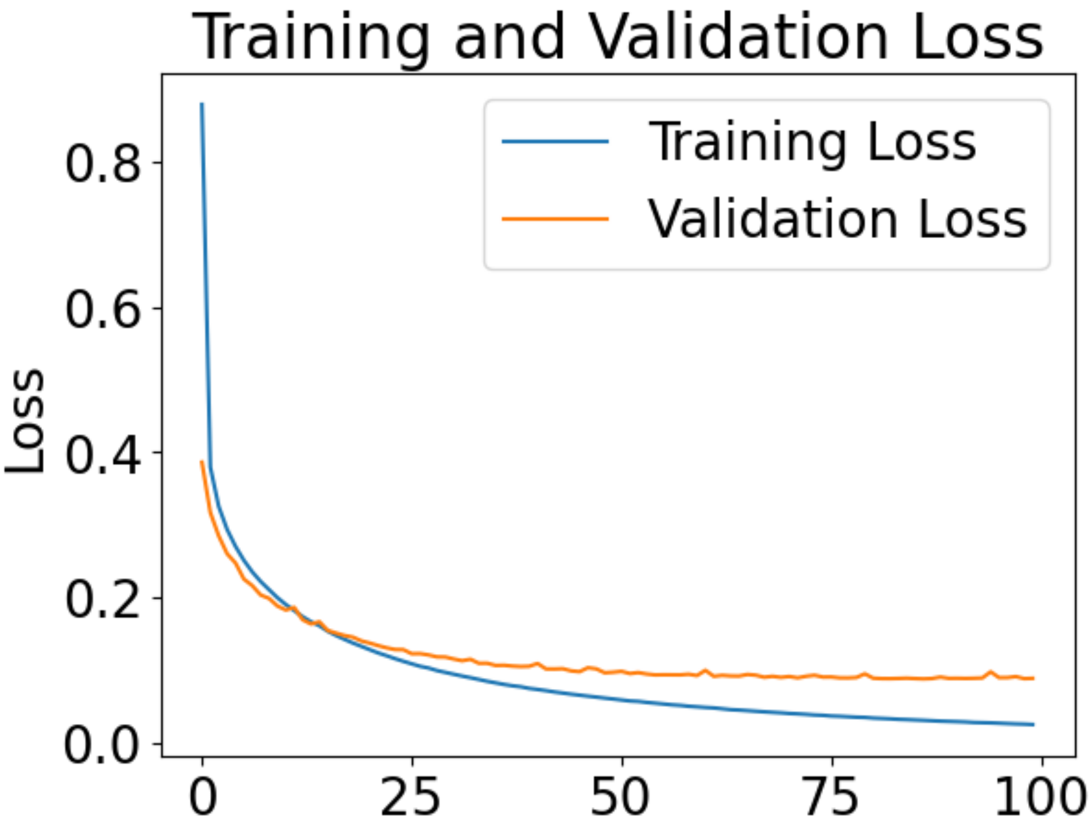
Based on the loss curves, select the learning rate and number of epochs that minimizes the validation loss

In [ ]:

```python
mlp = train_MLP_mnist(train_loader, val_loader, lr=0.1, num_epochs=100)

train_loss, train_error = evaluate_MLP(mlp, train_loader)
val_loss, val_error = evaluate_MLP(mlp, val_loader)
test_loss, test_error = evaluate_MLP(mlp, test_loader)

print(f"Training Loss: {train_loss}, Training Error: {train_error}")
print(f"Validation Loss: {val_loss}, Validation Error: {val_error}")
print(f"Test Loss: {test_loss}, Test Error: {test_error}")
```

```
100%|██████████| 196/196 [00:01<00:00, 175.33it/s]
100%|██████████| 196/196 [00:00<00:00, 256.50it/s]
100%|██████████| 196/196 [00:00<00:00, 263.43it/s]
100%|██████████| 196/196 [00:00<00:00, 327.75it/s]
100%|██████████| 196/196 [00:00<00:00, 321.80it/s]
100%|██████████| 196/196 [00:00<00:00, 343.98it/s]
100%|██████████| 196/196 [00:00<00:00, 324.17it/s]
100%|██████████| 196/196 [00:00<00:00, 336.69it/s]
100%|██████████| 196/196 [00:00<00:00, 343.51it/s]
100%|██████████| 196/196 [00:00<00:00, 346.18it/s]
100%|██████████| 196/196 [00:00<00:00, 339.05it/s]
100%|██████████| 196/196 [00:00<00:00, 344.35it/s]
100%|██████████| 196/196 [00:00<00:00, 333.81it/s]
100%|██████████| 196/196 [00:00<00:00, 341.05it/s]
100%|██████████| 196/196 [00:00<00:00, 285.51it/s]
100%|██████████| 196/196 [00:00<00:00, 347.02it/s]
100%|██████████| 196/196 [00:00<00:00, 344.78it/s]
100%|██████████| 196/196 [00:00<00:00, 346.55it/s]
100%|██████████| 196/196 [00:00<00:00, 344.01it/s]
100%|██████████| 196/196 [00:00<00:00, 343.85it/s]
100%|██████████| 196/196 [00:00<00:00, 343.78it/s]
100%|██████████| 196/196 [00:00<00:00, 342.95it/s]
100%|██████████| 196/196 [00:00<00:00, 347.25it/s]
100%|██████████| 196/196 [00:00<00:00, 342.82it/s]
100%|██████████| 196/196 [00:00<00:00, 346.11it/s]
100%|██████████| 196/196 [00:00<00:00, 344.59it/s]
100%|██████████| 196/196 [00:00<00:00, 348.18it/s]
100%|██████████| 196/196 [00:00<00:00, 347.53it/s]
100%|██████████| 196/196 [00:00<00:00, 287.78it/s]
100%|██████████| 196/196 [00:00<00:00, 342.84it/s]
100%|██████████| 196/196 [00:00<00:00, 347.59it/s]
100%|██████████| 196/196 [00:00<00:00, 345.33it/s]
100%|██████████| 196/196 [00:00<00:00, 318.12it/s]
100%|██████████| 196/196 [00:00<00:00, 334.90it/s]
100%|██████████| 196/196 [00:00<00:00, 340.04it/s]
100%|██████████| 196/196 [00:00<00:00, 350.29it/s]
100%|██████████| 196/196 [00:00<00:00, 348.58it/s]
100%|██████████| 196/196 [00:00<00:00, 337.84it/s]
100%|██████████| 196/196 [00:00<00:00, 345.23it/s]
100%|██████████| 196/196 [00:00<00:00, 346.81it/s]
100%|██████████| 196/196 [00:00<00:00, 343.56it/s]
100%|██████████| 196/196 [00:00<00:00, 348.14it/s]
100%|██████████| 196/196 [00:00<00:00, 287.58it/s]
100%|██████████| 196/196 [00:00<00:00, 349.40it/s]
100%|██████████| 196/196 [00:00<00:00, 343.03it/s]
100%|██████████| 196/196 [00:00<00:00, 346.38it/s]
100%|██████████| 196/196 [00:00<00:00, 344.88it/s]
100%|██████████| 196/196 [00:00<00:00, 344.60it/s]
100%|██████████| 196/196 [00:00<00:00, 345.48it/s]
100%|██████████| 196/196 [00:00<00:00, 345.12it/s]
100%|██████████| 196/196 [00:00<00:00, 305.57it/s]
100%|██████████| 196/196 [00:00<00:00, 353.86it/s]
100%|██████████| 196/196 [00:00<00:00, 331.03it/s]
100%|██████████| 196/196 [00:00<00:00, 349.44it/s]
100%|██████████| 196/196 [00:00<00:00, 339.71it/s]
100%|██████████| 196/196 [00:00<00:00, 344.75it/s]
```

```
100%|██████████| 196/196 [00:00<00:00, 343.20it/s]
100%|██████████| 196/196 [00:00<00:00, 342.72it/s]
100%|██████████| 196/196 [00:00<00:00, 347.83it/s]
100%|██████████| 196/196 [00:00<00:00, 348.03it/s]
100%|██████████| 196/196 [00:00<00:00, 343.99it/s]
100%|██████████| 196/196 [00:00<00:00, 347.72it/s]
100%|██████████| 196/196 [00:00<00:00, 345.58it/s]
100%|██████████| 196/196 [00:00<00:00, 348.10it/s]
100%|██████████| 196/196 [00:00<00:00, 347.12it/s]
100%|██████████| 196/196 [00:00<00:00, 344.14it/s]
100%|██████████| 196/196 [00:00<00:00, 288.61it/s]
100%|██████████| 196/196 [00:00<00:00, 344.34it/s]
100%|██████████| 196/196 [00:00<00:00, 339.22it/s]
100%|██████████| 196/196 [00:00<00:00, 348.46it/s]
100%|██████████| 196/196 [00:00<00:00, 347.74it/s]
100%|██████████| 196/196 [00:00<00:00, 344.83it/s]
100%|██████████| 196/196 [00:00<00:00, 346.60it/s]
100%|██████████| 196/196 [00:00<00:00, 346.69it/s]
100%|██████████| 196/196 [00:00<00:00, 342.85it/s]
100%|██████████| 196/196 [00:00<00:00, 346.12it/s]
100%|██████████| 196/196 [00:00<00:00, 347.73it/s]
100%|██████████| 196/196 [00:00<00:00, 335.85it/s]
100%|██████████| 196/196 [00:00<00:00, 348.23it/s]
100%|██████████| 196/196 [00:00<00:00, 349.45it/s]
100%|██████████| 196/196 [00:00<00:00, 346.35it/s]
100%|██████████| 196/196 [00:00<00:00, 327.12it/s]
100%|██████████| 196/196 [00:00<00:00, 323.55it/s]
100%|██████████| 196/196 [00:00<00:00, 347.10it/s]
100%|██████████| 196/196 [00:00<00:00, 343.86it/s]
100%|██████████| 196/196 [00:00<00:00, 347.35it/s]
100%|██████████| 196/196 [00:00<00:00, 346.29it/s]
100%|██████████| 196/196 [00:00<00:00, 348.15it/s]
100%|██████████| 196/196 [00:00<00:00, 345.34it/s]
100%|██████████| 196/196 [00:00<00:00, 341.01it/s]
100%|██████████| 196/196 [00:00<00:00, 346.32it/s]
100%|██████████| 196/196 [00:00<00:00, 347.73it/s]
100%|██████████| 196/196 [00:00<00:00, 350.04it/s]
100%|██████████| 196/196 [00:00<00:00, 215.34it/s]
100%|██████████| 196/196 [00:00<00:00, 232.39it/s]
100%|██████████| 196/196 [00:00<00:00, 305.25it/s]
```



```
Validation Loss: 0.09121068652272224, Validation Error: 0.026699999999999946
Test Loss: 0.08075617408510298, Test Error: 0.024299999999999999
```

Part 3: Predicting Penguin Species

Include all your code for part 3 in this section.

In [ ]:

```
import numpy as np
%matplotlib inline
from matplotlib import pyplot as plt
import pandas as pd
import seaborn as sns
#styling preferences for sns
sns.set_style('whitegrid')
sns.set_context('poster')
datadir = datadir = "/Users/darian/Desktop/UIUC/Applied ML/HW4/Code/penguins_size.csv" #
TO DO: modify this to your directory
df_penguins = pd.read_csv(datadir)
df_penguins.head(10)

# convert features with multiple string values to binary features so they can be used by
sklearn
def get_penguin_xy(df_penguins):
  data = np.array(df_penguins[['island', 'culmen_length_mm', 'culmen_depth_mm', 'flipper
_length_mm', 'body_mass_g', 'sex']])
  y = df_penguins['species']
  ui = np.unique(data[:,0]) # unique island
  us = np.unique(data[:,-1]) # unique sex
  X = np.zeros((len(y), 10))
  for i in range(len(y)):
    f = 0
    for j in range(len(ui)):
      if data[i, f]==ui[j]:
        X[i, f+j] = 1
    f = f + len(ui)
    X[i, f:(f+4)] = data[i, 1:5]
    f=f+4
    for j in range(len(us)):
      if data[i, 5]==us[j]:
        X[i, f+j] = 1
  feature_names = ['island_biscoe', 'island_dream', 'island_torgersen', 'culmen_length_mm
', 'culmen_depth_mm', 'flipper_length_mm', 'body_mass_g', 'sex_female', 'sex_male', 'sex
_unknown']
  X = pd.DataFrame(X, columns=feature_names)
  return(X, y, feature_names, np.unique(y))
```

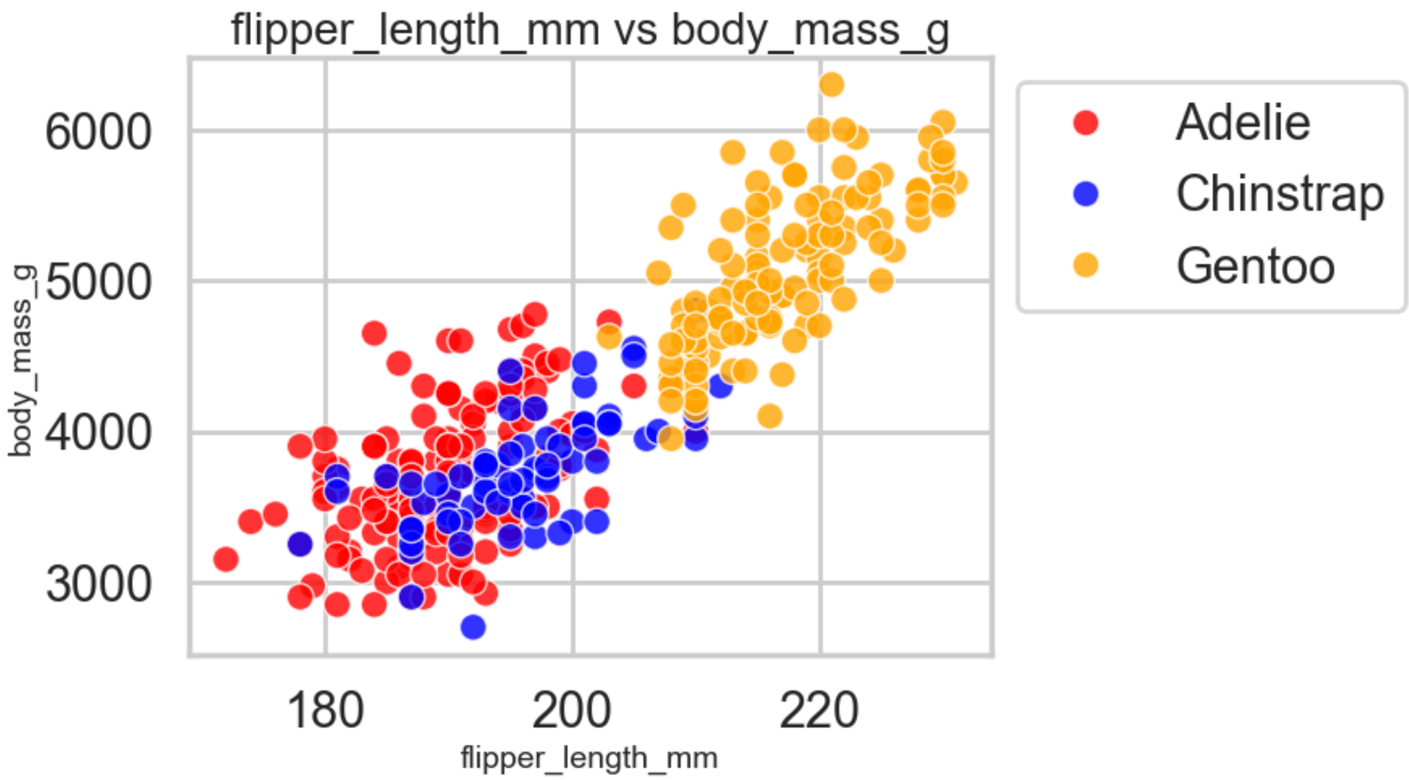
3a

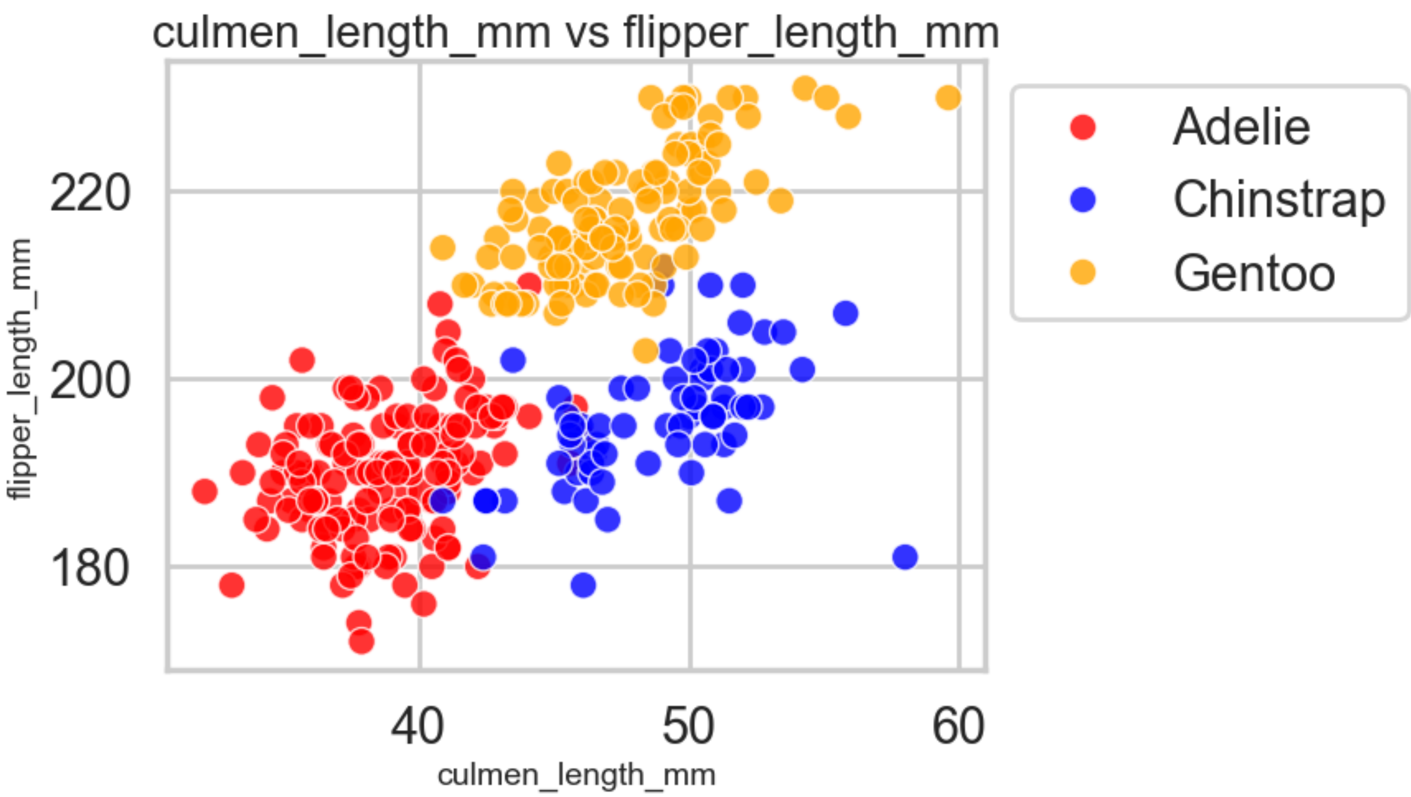
Spend some time to visualize different pairs of features and their relationships to the species. We've done one for you. Include in your report at least two other visualizations.

In [ ]:

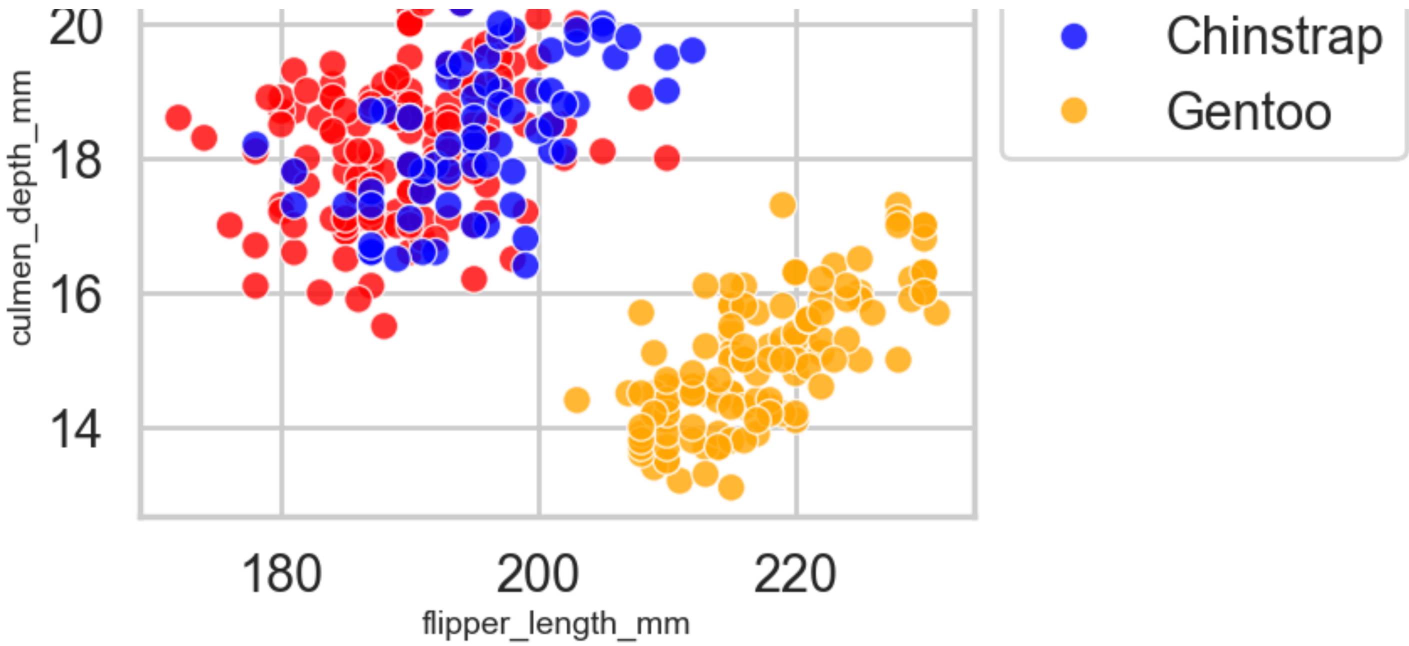
```
def plot_scatter(feature1, feature2):
  '''
  Provide names of two features to create a scatterplot of them
  E.g. plot_scatter('culmen_length_mm', 'culmen_depth_mm')
  Possible features: 'culmen_length_mm', 'culmen_depth_mm', 'flipper_length_mm', 'body_ma
ss_g'
  '''

  palette = ["red", "blue", "orange"]
```

```
    plt.xlabel(feature1, fontsize=14)
    plt.ylabel(feature2, fontsize=14)
    plt.title(feature1 + ' vs ' + feature2, fontsize=20)
    plt.legend(bbox_to_anchor=(1.0, 1.0), loc='upper left')
    plt.show()

# TO DO call plot_scatter with different feature pairs to create some visualizations

plot_scatter('flipper_length_mm', 'body_mass_g')
plot_scatter('culmen_length_mm', 'flipper_length_mm')
plot_scatter('flipper_length_mm', 'culmen_depth_mm')
```





flipper_length_mm vs culmen_depth_mm

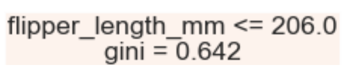


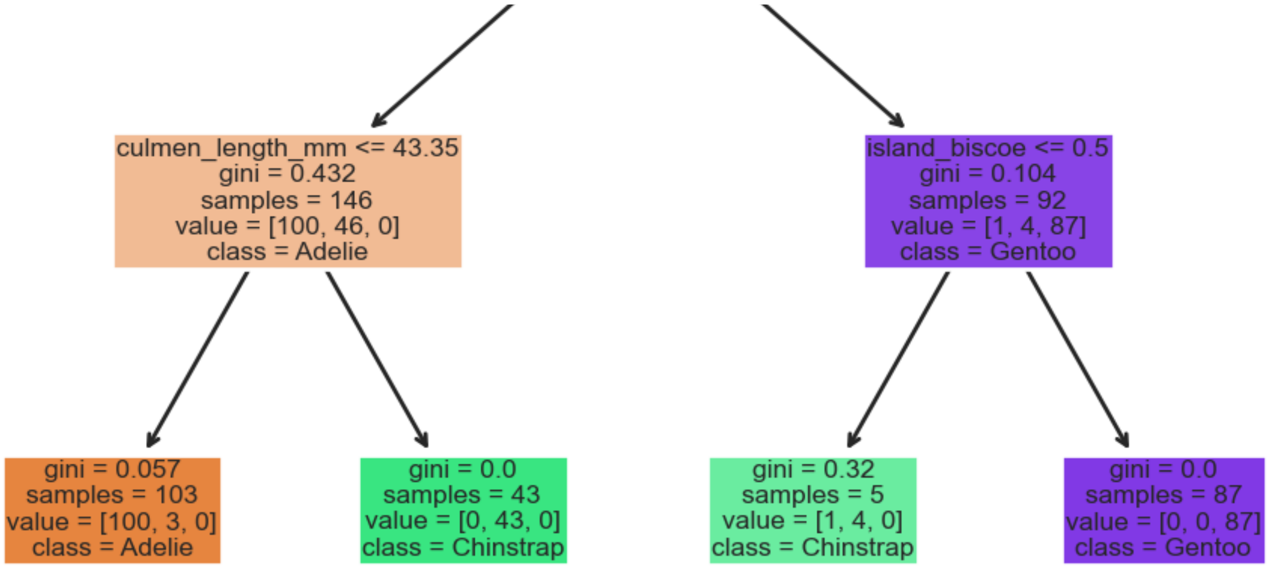
3b

Suppose you want to be able to identify the Gentoo species with a simple rule with very high accuracy. Use a decision tree classifier to figure out such a rule that has only two checks (e.g. “mass greater than 4000 g, and culmen length less than 40 mm is Gentoo; otherwise, not”). You can use the library DecisionTreeClassifier with either ‘gini’ or ‘entropy’ criterion. Use sklearn.tree.plot_tree with feature_names and class_names arguments to visualize the decision tree. Include the tree that you used to find the rule in your report and the rule.

In [ ]:

```
# TO DO (Train a short tree to identify a good rule, plot the tree, report the rule and i
ts precision/recall in your report)

from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import precision_score, recall_score
from sklearn.model_selection import train_test_split

# get_penguin_xy is defined and returns X, y, feature_names, class_names
X, y, feature_names, class_names = get_penguin_xy(df_penguins)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42
)

# Initialize and train the classifier
clf = DecisionTreeClassifier(criterion='gini', max_depth=2, random_state=42)
clf.fit(X_train, y_train)

# Visualize the tree
plt.figure(figsize=(12,8))
plot_tree(clf, filled=True, feature_names=feature_names, class_names=class_names)
plt.show()

y_pred = clf.predict(X_test)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')

print(f"Precision: {precision}")
print(f"Recall: {recall}")
```






```
Precision: 0.8949065119277885
Recall: 0.9201058201058201
```

3c

Use any method at your disposal to achieve maximum 5-fold cross-validation accuracy on this problem. To keep it simple, we will use sklearn.model_selection to perform the cross-validation for us. Report your model design and 5-fold accuracy. It is possible to get more than 99% accuracy.

In [ ]:

```
# design a classification model, import libraries as needed
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

X, y, feature_names, class_names = get_penguin_xy(df_penguins)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42
)

# TO DO -- choose some model and fit the data
model = RandomForestClassifier()
model.fit(X_train, y_train)

scores = cross_val_score(model, np.array(X), np.array(y), cv=5)
print('CV Accuracy: {}'.format(scores.mean()))
```

```
CV Accuracy: 0.9941176470588236
```

Part 4: Stretch Goals

Include any new code needed for Part 4 here

4a

In [ ]:

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
import tqdm
```

```python
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, hidden_size)  # Second hidden layer
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden_size, hidden_size)  # Third hidden layer
        self.relu3 = nn.ReLU()
        self.fc5 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = self.relu1(self.fc1(x))
        x = self.relu2(self.fc2(x))
        x = self.relu3(self.fc3(x))
        x = self.fc5(x)
        return x


def train_MLP_mnist(train_loader, val_loader, lr, num_epochs):
  '''
  Train a MLP
  Input: train_loader and val_loader are dataloaders for the training and
  val data, respectively. lr is the learning rate, and the network will
  be trained for num_epochs epochs.
  Output: return a trained MLP
  '''
  # TODO: fill in all code

  input_size = 28*28
  hidden_size = 128
  output_size = 10

  mlp = MLP(input_size, output_size).to(device)
  loss_func = nn.CrossEntropyLoss()
  optimizer = torch.optim.Adam(mlp.parameters(), lr=lr)

  training_losses = []
  validation_losses = []

  for epoch in range(num_epochs):
      mlp.train()
      running_loss = 0.0
      for inputs, labels in tqdm.tqdm(train_loader):
          inputs, labels = inputs.to(device), labels.to(device)
          optimizer.zero_grad()
          outputs = mlp(inputs)
          loss = loss_func(outputs, labels)
          loss.backward()
          optimizer.step()

          running_loss += loss.item()
      training_losses.append(running_loss / len(train_loader))

      mlp.eval()
      running_val_loss = 0.0
      with torch.no_grad():
          for inputs, labels in val_loader:
              inputs, labels = inputs.to(device), labels.to(device)
              outputs = mlp(inputs)
              val_loss = loss_func(outputs, labels)
              running_val_loss += val_loss.item()
      validation_losses.append(running_val_loss / len(val_loader))

  return mlp


device = torch.device("mps")

tensor_x_train = torch.FloatTensor(x_train)
```

```python
# Convert test data to DataLoader
tensor_x_test = torch.FloatTensor(x_test)
tensor_y_test = torch.LongTensor(y_test)
test_dataset = torch.utils.data.TensorDataset(tensor_x_test, tensor_y_test)
test_loader = DataLoader(dataset=test_dataset, batch_size=256, shuffle=False)

train_dataset = torch.utils.data.TensorDataset(tensor_x_train, tensor_y_train)
val_dataset = torch.utils.data.TensorDataset(tensor_x_val, tensor_y_val)

train_loader = DataLoader(dataset=train_dataset, batch_size=256, shuffle=True)
val_loader = DataLoader(dataset=val_dataset, batch_size=256, shuffle=False)


# Evaluation

mlp = train_MLP_mnist(train_loader, val_loader, lr=0.001, num_epochs=300)

train_loss, train_error = evaluate_MLP(mlp, train_loader)
val_loss, val_error = evaluate_MLP(mlp, val_loader)
test_loss, test_error = evaluate_MLP(mlp, test_loader)

print(f"Training Loss: {train_loss}, Training Error: {train_error}")
print(f"Validation Loss: {val_loss}, Validation Error: {val_error}")
print(f"Test Loss: {test_loss}, Test Error: {test_error}")
```

```
100%|██████████| 196/196 [00:01<00:00, 173.80it/s]
100%|██████████| 196/196 [00:01<00:00, 160.02it/s]
100%|██████████| 196/196 [00:01<00:00, 145.23it/s]
100%|██████████| 196/196 [00:01<00:00, 144.56it/s]
100%|██████████| 196/196 [00:01<00:00, 193.32it/s]
100%|██████████| 196/196 [00:01<00:00, 171.30it/s]
100%|██████████| 196/196 [00:01<00:00, 141.98it/s]
100%|██████████| 196/196 [00:01<00:00, 173.95it/s]
100%|██████████| 196/196 [00:01<00:00, 165.53it/s]
100%|██████████| 196/196 [00:01<00:00, 192.01it/s]
100%|██████████| 196/196 [00:01<00:00, 186.11it/s]
100%|██████████| 196/196 [00:01<00:00, 176.27it/s]
100%|██████████| 196/196 [00:01<00:00, 177.56it/s]
100%|██████████| 196/196 [00:01<00:00, 182.97it/s]
100%|██████████| 196/196 [00:01<00:00, 149.19it/s]
100%|██████████| 196/196 [00:01<00:00, 156.83it/s]
100%|██████████| 196/196 [00:01<00:00, 178.35it/s]
100%|██████████| 196/196 [00:00<00:00, 196.06it/s]
100%|██████████| 196/196 [00:01<00:00, 162.29it/s]
100%|██████████| 196/196 [00:01<00:00, 190.01it/s]
100%|██████████| 196/196 [00:01<00:00, 185.64it/s]
100%|██████████| 196/196 [00:01<00:00, 183.18it/s]
100%|██████████| 196/196 [00:01<00:00, 183.51it/s]
100%|██████████| 196/196 [00:00<00:00, 199.63it/s]
100%|██████████| 196/196 [00:00<00:00, 199.19it/s]
100%|██████████| 196/196 [00:01<00:00, 195.56it/s]
100%|██████████| 196/196 [00:01<00:00, 194.89it/s]
100%|██████████| 196/196 [00:00<00:00, 198.21it/s]
100%|██████████| 196/196 [00:00<00:00, 198.23it/s]
100%|██████████| 196/196 [00:01<00:00, 188.05it/s]
100%|██████████| 196/196 [00:01<00:00, 165.22it/s]
100%|██████████| 196/196 [00:01<00:00, 195.63it/s]
100%|██████████| 196/196 [00:01<00:00, 173.89it/s]
100%|██████████| 196/196 [00:01<00:00, 156.05it/s]
100%|██████████| 196/196 [00:01<00:00, 176.13it/s]
100%|██████████| 196/196 [00:00<00:00, 200.10it/s]
100%|██████████| 196/196 [00:01<00:00, 179.41it/s]
100%|██████████| 196/196 [00:01<00:00, 178.43it/s]
100%|██████████| 196/196 [00:01<00:00, 182.79it/s]
100%|██████████| 196/196 [00:01<00:00, 181.79it/s]
100%|██████████| 196/196 [00:00<00:00, 197.68it/s]
100%|██████████| 196/196 [00:01<00:00, 168.73it/s]
100%|██████████| 196/196 [00:01<00:00, 161.19it/s]
```

```
100%|██████████| 196/196 [00:00<00:00, 200.13it/s]
100%|██████████| 196/196 [00:01<00:00, 192.03it/s]
100%|██████████| 196/196 [00:00<00:00, 198.73it/s]
100%|██████████| 196/196 [00:00<00:00, 200.44it/s]
100%|██████████| 196/196 [00:01<00:00, 175.80it/s]
100%|██████████| 196/196 [00:00<00:00, 198.95it/s]
100%|██████████| 196/196 [00:00<00:00, 198.69it/s]
100%|██████████| 196/196 [00:00<00:00, 199.70it/s]
100%|██████████| 196/196 [00:00<00:00, 200.04it/s]
100%|██████████| 196/196 [00:01<00:00, 171.12it/s]
100%|██████████| 196/196 [00:01<00:00, 162.87it/s]
100%|██████████| 196/196 [00:01<00:00, 104.98it/s]
100%|██████████| 196/196 [00:01<00:00, 151.48it/s]
100%|██████████| 196/196 [00:01<00:00, 147.57it/s]
100%|██████████| 196/196 [00:01<00:00, 157.66it/s]
100%|██████████| 196/196 [00:01<00:00, 158.48it/s]
100%|██████████| 196/196 [00:01<00:00, 177.40it/s]
100%|██████████| 196/196 [00:01<00:00, 179.00it/s]
100%|██████████| 196/196 [00:01<00:00, 181.81it/s]
100%|██████████| 196/196 [00:01<00:00, 153.52it/s]
100%|██████████| 196/196 [00:01<00:00, 195.11it/s]
100%|██████████| 196/196 [00:01<00:00, 156.02it/s]
100%|██████████| 196/196 [00:01<00:00, 128.05it/s]
100%|██████████| 196/196 [00:01<00:00, 127.85it/s]
100%|██████████| 196/196 [00:01<00:00, 125.56it/s]
100%|██████████| 196/196 [00:01<00:00, 135.41it/s]
100%|██████████| 196/196 [00:01<00:00, 161.43it/s]
100%|██████████| 196/196 [00:01<00:00, 153.60it/s]
100%|██████████| 196/196 [00:01<00:00, 176.57it/s]
100%|██████████| 196/196 [00:01<00:00, 183.36it/s]
100%|██████████| 196/196 [00:01<00:00, 148.61it/s]
100%|██████████| 196/196 [00:01<00:00, 149.12it/s]
100%|██████████| 196/196 [00:01<00:00, 145.25it/s]
100%|██████████| 196/196 [00:01<00:00, 187.23it/s]
100%|██████████| 196/196 [00:01<00:00, 153.05it/s]
100%|██████████| 196/196 [00:01<00:00, 161.20it/s]
100%|██████████| 196/196 [00:00<00:00, 199.75it/s]
```

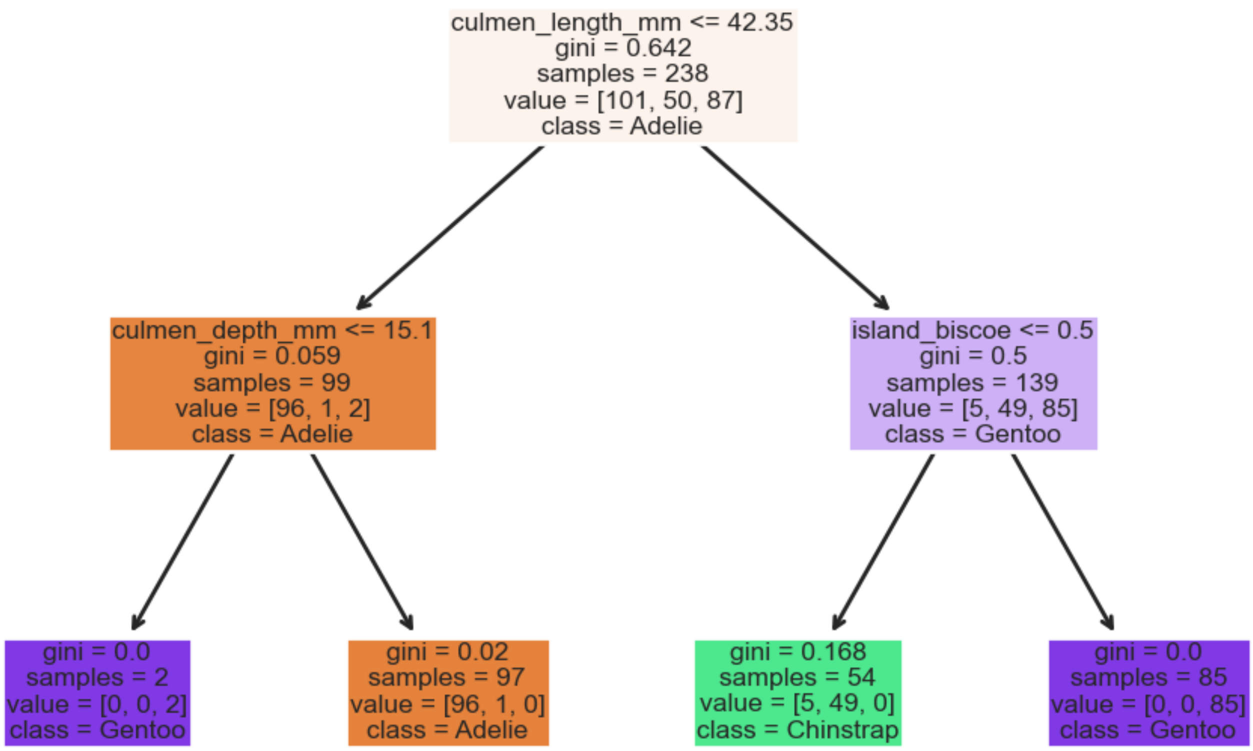
```
Training Loss: 5.245208427595571e-11, Training Error: 0.0
Validation Loss: 0.2797041184850037, Validation Error: 0.019399999999999973
Test Loss: 0.26092285749805977, Test Error: 0.018599999999999995
```

4b

In [ ]:

```python
# TO DO (Train a short tree to identify a good rule, plot the tree, report the rule and i
ts precision/recall in your report)

from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import precision_score, recall_score
from sklearn.model_selection import train_test_split

def get_penguin_xy_modified(df_penguins):
    # Select features excluding 'flipper_length_mm' and 'island_biscoe'
    selected_features = ['island', 'culmen_length_mm', 'culmen_depth_mm', 'body_mass_g',
'sex']
    data = np.array(df_penguins[selected_features])
    y = df_penguins['species']


    ui = np.unique(data[:,0])
    us = np.unique(data[:,-1])
    X = []
    feature_names = []

    for feature in selected_features:
```

```
        elif feature == 'sex':
            for sex in us:
                X.append((data[:, -1] == sex).astype(float))
                feature_names.append(f'sex_{sex.lower()}')
        else:
            X.append(data[:, selected_features.index(feature)].astype(float))
            feature_names.append(feature)

    X = np.array(X).T  # Transpose to get correct shape
    return pd.DataFrame(X, columns=feature_names), y, feature_names, np.unique(y)


# get_penguin_xy is defined and returns X, y, feature_names, class_names
X, y, feature_names, class_names = get_penguin_xy_modified(df_penguins)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42
)

# Initialize and train the classifier
clf = DecisionTreeClassifier(criterion='gini', max_depth=2, random_state=42)
clf.fit(X_train, y_train)

# Visualize the tree
plt.figure(figsize=(12,8))
plot_tree(clf, filled=True, feature_names=feature_names, class_names=class_names)
plt.show()

y_pred = clf.predict(X_test)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')

print(f"Precision: {precision}")
print(f"Recall: {recall}")
```




```
Precision: 0.9130781499202553
Recall: 0.9533333333333333
```

In [62]:

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
import numpy as np
import cv2
import matplotlib.pyplot as plt


class MLP(nn.Module):
    def __init__(self, input_size=2, output_size=3, hidden_size=128):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, hidden_size)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden_size, hidden_size)
        self.relu3 = nn.ReLU()
        self.fc4 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = self.relu1(self.fc1(x))
        x = self.relu2(self.fc2(x))
        x = self.relu3(self.fc3(x))
        x = self.fc4(x)
        return x


def train_MLP_RGB(train_loader, lr, num_epochs):
  '''
  Train a MLP
  Input: train_loader and val_loader are dataloaders for the training and
  val data, respectively. lr is the learning rate, and the network will
  be trained for num_epochs epochs.
  Output: return a trained MLP
  '''
  # TODO: fill in all code

  input_size = 2     # xy coord
  hidden_size = 128
  output_size = 3    # RGB vals

  mlp = MLP(input_size, output_size).to(device)
  loss_func = nn.MSELoss()
  optimizer = torch.optim.Adam(mlp.parameters(), lr=lr)

  training_losses = []
  validation_losses = []

  for epoch in range(num_epochs):
      mlp.train()
      running_loss = 0.0
      for inputs, labels in tqdm.tqdm(train_loader):
          inputs, labels = inputs.to(device), labels.to(device)
          optimizer.zero_grad()
          outputs = mlp(inputs)
          loss = loss_func(outputs, labels)
          loss.backward()
          optimizer.step()

          running_loss += loss.item()
      print(f"Epoch {epoch+1}, Loss: {running_loss / len(train_loader)}")

  return mlp

device = torch.device("mps")
```

```python
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
im = im / 255.0  # Normalize to [0, 1]

# Prepare dataset
h, w, _ = im.shape
X = np.array([(x, y) for x in range(w) for y in range(h)])
y = im.reshape(-1, 3)
tensor_x = torch.FloatTensor(X)
tensor_y = torch.FloatTensor(y)
dataset = TensorDataset(tensor_x, tensor_y)
train_loader = DataLoader(dataset, batch_size=256, shuffle=True)


# Train the model
mlp = train_MLP_RGB(train_loader, lr=0.01, num_epochs=100)


# Predict and reconstruct image
mlp.eval()
with torch.no_grad():
    pred = mlp(tensor_x.to(device)).cpu().numpy()
pred_img = pred.reshape(h, w, 3)

# Display the reconstructed image
plt.imshow(pred_img)
plt.title("Reconstructed Image")
plt.show()
```

```
Corrupt JPEG data: premature end of data segment
100%|██████████| 16/16 [00:00<00:00, 188.32it/s]
```

```
Epoch 1, Loss: 6.361784530803561
```

```
100%|██████████| 16/16 [00:00<00:00, 220.27it/s]
```

```
Epoch 2, Loss: 0.097307488322258
```

```
100%|██████████| 16/16 [00:00<00:00, 199.98it/s]
```

```
Epoch 3, Loss: 0.0809036330319941
```

```
100%|██████████| 16/16 [00:00<00:00, 212.88it/s]
```

```
Epoch 4, Loss: 0.06743407342582941
```

```
100%|██████████| 16/16 [00:00<00:00, 232.40it/s]
```

```
Epoch 5, Loss: 0.045191442826762795
```

```
100%|██████████| 16/16 [00:00<00:00, 219.21it/s]
```

```
Epoch 6, Loss: 0.02667774073779583
```

```
100%|██████████| 16/16 [00:00<00:00, 228.22it/s]
```

```
Epoch 7, Loss: 0.01827424461953342
```

```
100%|██████████| 16/16 [00:00<00:00, 199.41it/s]
```

```
Epoch 8, Loss: 0.015216938685625792
```

```
100%|██████████| 16/16 [00:00<00:00, 192.17it/s]
```

```
Epoch 9, Loss: 0.013051668822299689
```

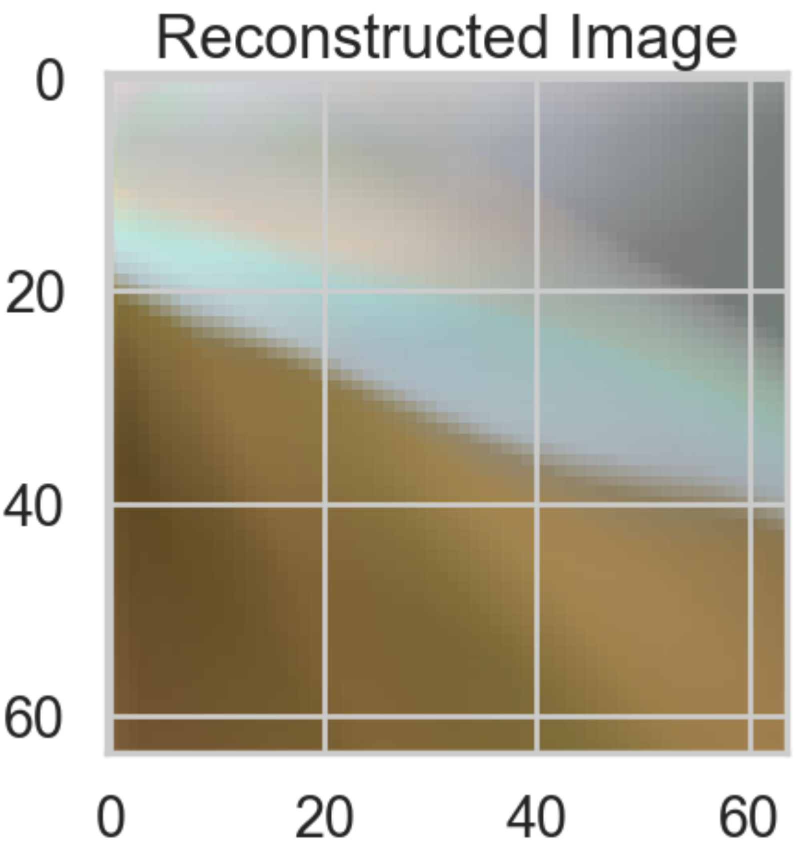
```
100%|██████████| 16/16 [00:00<00:00, 191.99it/s]
```

```
Epoch 10, Loss: 0.012649160344153643
```

```
100%|██████████| 16/16 [00:00<00:00, 207.00it/s]
```

```
Epoch 11, Loss: 0.011387166276108474
```

```
100%|██████████| 16/16 [00:00<00:00, 208.11it/s]
Epoch 93, Loss: 0.0077420717279892415
100%|██████████| 16/16 [00:00<00:00, 199.68it/s]
Epoch 94, Loss: 0.0086943804344628
100%|██████████| 16/16 [00:00<00:00, 198.53it/s]
Epoch 95, Loss: 0.009057783696334809
100%|██████████| 16/16 [00:00<00:00, 213.43it/s]
Epoch 96, Loss: 0.008273055602330714
100%|██████████| 16/16 [00:00<00:00, 226.82it/s]
Epoch 97, Loss: 0.007772155193379149
100%|██████████| 16/16 [00:00<00:00, 225.66it/s]
Epoch 98, Loss: 0.0076892363431397825
100%|██████████| 16/16 [00:00<00:00, 227.89it/s]
Epoch 99, Loss: 0.007563946273876354
100%|██████████| 16/16 [00:00<00:00, 229.02it/s]
Epoch 100, Loss: 0.007868390064686537
```



In [ ]:

```
# 4c.2 and 4c.3

import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
import numpy as np
import cv2
import matplotlib.pyplot as plt
```

```
datadir = "/Users/darian/Desktop/UIUC/Applied ML/HW4/Code/"
im = cv2.imread(datadir + 'scotland_photo_resized.jpg')
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
im = im / 255.0  # Normalize to [0, 1]


class MLP(nn.Module):
    def __init__(self, input_size=2, output_size=3, hidden_size=128):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, hidden_size)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden_size, hidden_size)
        self.relu3 = nn.ReLU()
        self.fc4 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = self.relu1(self.fc1(x))
        x = self.relu2(self.fc2(x))
        x = self.relu3(self.fc3(x))
        x = self.fc4(x)
        return x


def train_MLP_RGB(train_loader, lr, num_epochs):
  '''
  Train a MLP
  Input: train_loader and val_loader are dataloaders for the training and
  val data, respectively. lr is the learning rate, and the network will
  be trained for num_epochs epochs.
  Output: return a trained MLP
  '''
  # TODO: fill in all code

  input_size = 2     # xy coord
  hidden_size = 128
  output_size = 3    # RGB vals

  mlp = MLP(input_size, output_size).to(device)
  loss_func = nn.MSELoss()
  optimizer = torch.optim.Adam(mlp.parameters(), lr=lr)

  training_losses = []
  validation_losses = []

  for epoch in range(num_epochs):
      mlp.train()
      running_loss = 0.0
      for inputs, labels in tqdm.tqdm(train_loader):
          inputs, labels = inputs.to(device), labels.to(device)
          optimizer.zero_grad()
          outputs = mlp(inputs)
          loss = loss_func(outputs, labels)
          loss.backward()
          optimizer.step()

          running_loss += loss.item()
      print(f"Epoch {epoch+1}, Loss: {running_loss / len(train_loader)}")

  return mlp

device = torch.device("mps")

def positional_encoding(x, y, d_model):

    # x: x-coordinate of the pixel
    # y: y-coordinate of the pixel
```

```
    pe = np.zeros(d_model)
    for pos in range(d_model // 2):
        div_term = np.exp(pos * -np.log(10000.0) / (d_model // 2))
        pe[2 * pos] = np.sin(position[0] * div_term) + np.sin(position[1] * div_term)
        pe[2 * pos + 1] = np.cos(position[0] * div_term) + np.cos(position[1] * div_term
)
    return pe

# prepare dataset with pos enc
d_model = 128
X_encoded = np.array([positional_encoding(x, y, d_model) for x, y in X])
tensor_x_enc = torch.FloatTensor(X_encoded)

# Prepare dataset
h, w, _ = im.shape
X_encoded = np.array([(x, y) for x in range(w) for y in range(h)])
y = im.reshape(-1, 3)
tensor_x_enc = torch.FloatTensor(X)
tensor_y = torch.FloatTensor(y)
dataset = TensorDataset(tensor_x_enc, tensor_y)
train_loader = DataLoader(dataset, batch_size=256, shuffle=True)

# Load image

datadir = "/Users/darian/Desktop/UIUC/Applied ML/HW4/Code/"
im = cv2.imread(datadir + 'scotland_photo_resized.jpg')
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
im = im / 255.0  # Normalize to [0, 1]

# Prepare dataset
h, w, _ = im.shape
X = np.array([(x, y) for x in range(w) for y in range(h)])
y = im.reshape(-1, 3)
tensor_x = torch.FloatTensor(X)
tensor_y = torch.FloatTensor(y)
dataset = TensorDataset(tensor_x, tensor_y)
train_loader = DataLoader(dataset, batch_size=256, shuffle=True)

mlp = train_MLP_RGB(train_loader, lr=0.01, num_epochs=100)

# Predict and reconstruct image
mlp.eval()
with torch.no_grad():
    pred = mlp(tensor_x.to(device)).cpu().numpy()
pred_img = pred.reshape(h, w, 3)

# Display the reconstructed image
plt.imshow(pred_img)
plt.title("Reconstructed Image")
plt.show()
```

```
Corrupt JPEG data: premature end of data segment
Corrupt JPEG data: premature end of data segment
100%|██████████| 16/16 [00:00<00:00, 142.35it/s]
```

```
Epoch 1, Loss: 14.423560287803411
```

```
100%|██████████| 16/16 [00:00<00:00, 114.97it/s]
```

```
Epoch 2, Loss: 0.10160766169428825
```

```
100%|██████████| 16/16 [00:00<00:00, 160.48it/s]
```

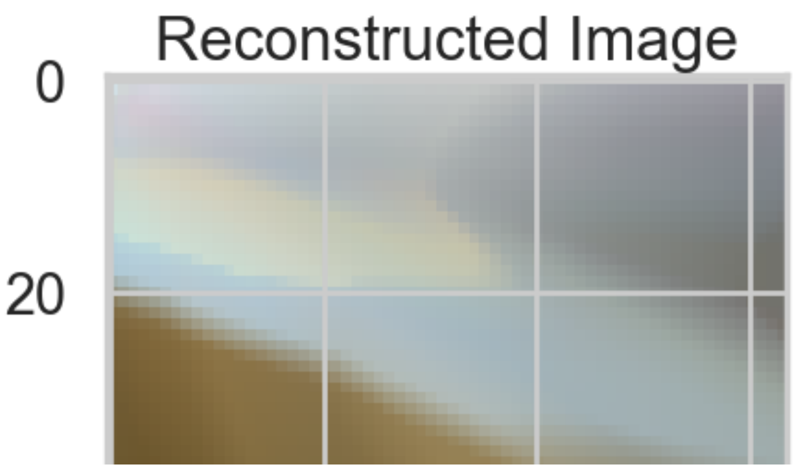
```
Epoch 3, Loss: 0.08556164475157857
```

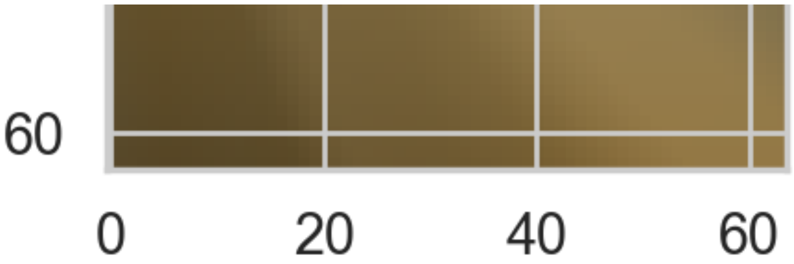
```
100%|██████████| 16/16 [00:00<00:00, 181.80it/s]
```

```
Epoch 4, Loss: 0.07205577474087477
```

```
100%|██████████| 16/16 [00:00<00:00, 184.31it/s]
```

```
Epoch 86, Loss: 0.00839236006140709
100%|██████████| 16/16 [00:00<00:00, 201.79it/s]
Epoch 87, Loss: 0.007733979582553729
100%|██████████| 16/16 [00:00<00:00, 194.00it/s]
Epoch 88, Loss: 0.007846991997212172
100%|██████████| 16/16 [00:00<00:00, 199.92it/s]
Epoch 89, Loss: 0.008405249653151259
100%|██████████| 16/16 [00:00<00:00, 194.44it/s]
Epoch 90, Loss: 0.008406280219787732
100%|██████████| 16/16 [00:00<00:00, 197.93it/s]
Epoch 91, Loss: 0.008626660273876041
100%|██████████| 16/16 [00:00<00:00, 193.43it/s]
Epoch 92, Loss: 0.008575594547437504
100%|██████████| 16/16 [00:00<00:00, 201.35it/s]
Epoch 93, Loss: 0.007901576376752928
100%|██████████| 16/16 [00:00<00:00, 190.07it/s]
Epoch 94, Loss: 0.008009086886886507
100%|██████████| 16/16 [00:00<00:00, 201.10it/s]
Epoch 95, Loss: 0.007925231097033247
100%|██████████| 16/16 [00:00<00:00, 204.87it/s]
Epoch 96, Loss: 0.008063661225605756
100%|██████████| 16/16 [00:00<00:00, 196.32it/s]
Epoch 97, Loss: 0.00797535470337607
100%|██████████| 16/16 [00:00<00:00, 207.72it/s]
Epoch 98, Loss: 0.008041169989155605
100%|██████████| 16/16 [00:00<00:00, 189.03it/s]
Epoch 99, Loss: 0.008494080655509606
100%|██████████| 16/16 [00:00<00:00, 194.87it/s]
Epoch 100, Loss: 0.008162668498698622
```





In [ ]:

```
# from https://gist.github.com/jonathanagustin/b67b97ef12c53a8dec27b343dca4abba
# install can take a minute

import os
# @title Convert Notebook to PDF. Save Notebook to given directory
NOTEBOOKS_DIR = "/content/drive/My Drive/CS441/24SP/hw2" # @param {type:"string"}
NOTEBOOK_NAME = "CS441_SP24_HW2_Solution.ipynb" # @param {type:"string"}
#------------------------------------------------------------------------------#
from google.colab import drive
drive.mount("/content/drive/", force_remount=True)
NOTEBOOK_PATH = f"{NOTEBOOKS_DIR}/{NOTEBOOK_NAME}"
assert os.path.exists(NOTEBOOK_PATH), f"NOTEBOOK NOT FOUND: {NOTEBOOK_PATH}"
!apt install -y texlive-xetex texlive-fonts-recommended texlive-plain-generic > /dev/null 2>&1
!jupyter nbconvert "$NOTEBOOK_PATH" --to pdf > /dev/null 2>&1
NOTEBOOK_PDF = NOTEBOOK_PATH.rsplit('.', 1)[0] + '.pdf'
assert os.path.exists(NOTEBOOK_PDF), f"ERROR MAKING PDF: {NOTEBOOK_PDF}"
print(f"PDF CREATED: {NOTEBOOK_PDF}")
```

In [ ]:

```

```