Georgia Institute of Technology

ECE 4803: Fundamentamentals of Machine Learning (FunML)

Spring 2022

Homework Assignment # 5

Due: Friday, April 1, 2022 @8PM

Please read the following instructions carefully.

- The entire homework assignment is to be completed on this `ipython` notebook. It is designed to be used with `Google Colab`, but you may use other tools (e.g., Jupyter Lab) as well.
- Make sure that you execute all cells in a way so their output is printed beneath the corresponding cell. Thus, after successfully executing all cells properly, the resulting notebook has all the questions and your answers.
- Print a PDF copy of the notebook with all its outputs printed and submit the **PDF** on `Canvas` under Assignments.
- Make sure you delete any scratch cells before you export this document as a PDF. Do not change the order of the questions and do not remove any part of the questions. Edit at the indicated places only.
- Rename the PDF according to the format: ***LastName_FirstName_ECE_4803_sp22_assignment_#.pdf***
- It is encouraged for you to discuss homework problems amongst each other, but any copying is strictly prohibited and will be subject to Georgia Tech Honor Code.
- Late homework is not accepted unless arranged otherwise and in advance.
- Comment on your codes.
- Refer to the tutorial and the supplementary/reading materials that are posted on `Canvas` for lectures 14, 15, 16, 17 to help you with this

- Refer to the tutorial and the supplementary/reading materials that are posted on `Canvas` for lectures 14, 15, 16, 17 to help you with this assignment.
- **IMPORTANT:** Start your solution with a **BOLD RED** text that includes the words *solution* and the part of the problem you are working on. For example, start your solution for Part (c) of Problem 2 by having the first line as: **Solution to Problem 2 Part (c)**. Failing to do so may result in a *20% penalty* of the total grade.

+ Code + Text

Assignment Objectives:

- Introduction to the use of `PyTorch` basic functions and libraries
- Learn deep model trainining and evaluation
- Learn to use pre-trained deep models for classifying your own images
- Advance in your `PyTorch` knowledge and experience

+ Code + Text

Recommended Readings

This assignment uses the popular Python-based deep learning framework, `Pytorch`. You are highly encouraged to refer to the following resources as references. The key functions you will be using relate to neural network and defining a dataloader pipeline.

- Introduction to `Pytorch` tensors and autograd
- Defining a neural network architecture
- Setting up a dataloader

The following web pages offer a summarized overview that is worth reading before starting the work in this assignment.

- Overview of `Pytorch`
- How to use a dataloader in `Pytorch`?

Guide for Exporting Ipython Notebook to PDF:

Guide for Exporting Ipython Notebook to PDF:

Here is a video summarizes how to export Ipythin Notebook into PDF.

- **[Method1: Print to PDF]**
  After you run every cell and get their outputs, you can use **[File] -> [Print]** and then choose **[Save as PDF]** to export this Ipython Notebook to PDF for submission.
  Note: Sometimes figures or texts are splited into different pages. Try to tweak the layout by adding empty lines to avoid this effect as much as you can.
- **[Method2: colab-pdf script]**
  The author of that video provided an alternative method that can generate better layout PDF. However, it only works for Ipythin Notebook without embedded images.
  How to use: Put the script below into cells at the end of your Ipythin Notebook. After you run the fisrt cell, it will ask for google drive permission. Executing the second cell will generate the PDF file in your google drive home directory. Make sure you use the correct path and file name.

```
## this will link colab with your google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
%%capture
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('LastName_FirstName_ECE_4803_sp22_assignment_#.ipynb') ## change path and file name
```

- **[Method3: GoFullPage Chrome Extension] (most recommended)**
  Install the extension and generate PDF file of the Ipython Notebook in the browser.

Note: Georgia Tech provides a student discount for Adobe Acrobat subscription. Further information can be found here.

Problem 1: Introduction to PyTorch

`PyTorch` is a Python-based scientific computing library for deep learning. `PyTorch` executes deep learning computations over `Tensor` object, which is a specialized data structure that behaves similarly as `Numpy` array but can run on GPUs. It also provides a powerful automatic differentiation engine that computes the gradients during the back-propagation. To enable the GPU go to:

```
Edit >> Notebook settings >> Hardware accelerator >> GPU >> save
```

This problem introduces tensor initialization, basic operations and automatic differentiation that you can carry out with `PyTorch` and the standard library `torch`. You may find the Link useful.

Suppose that you are given the following:

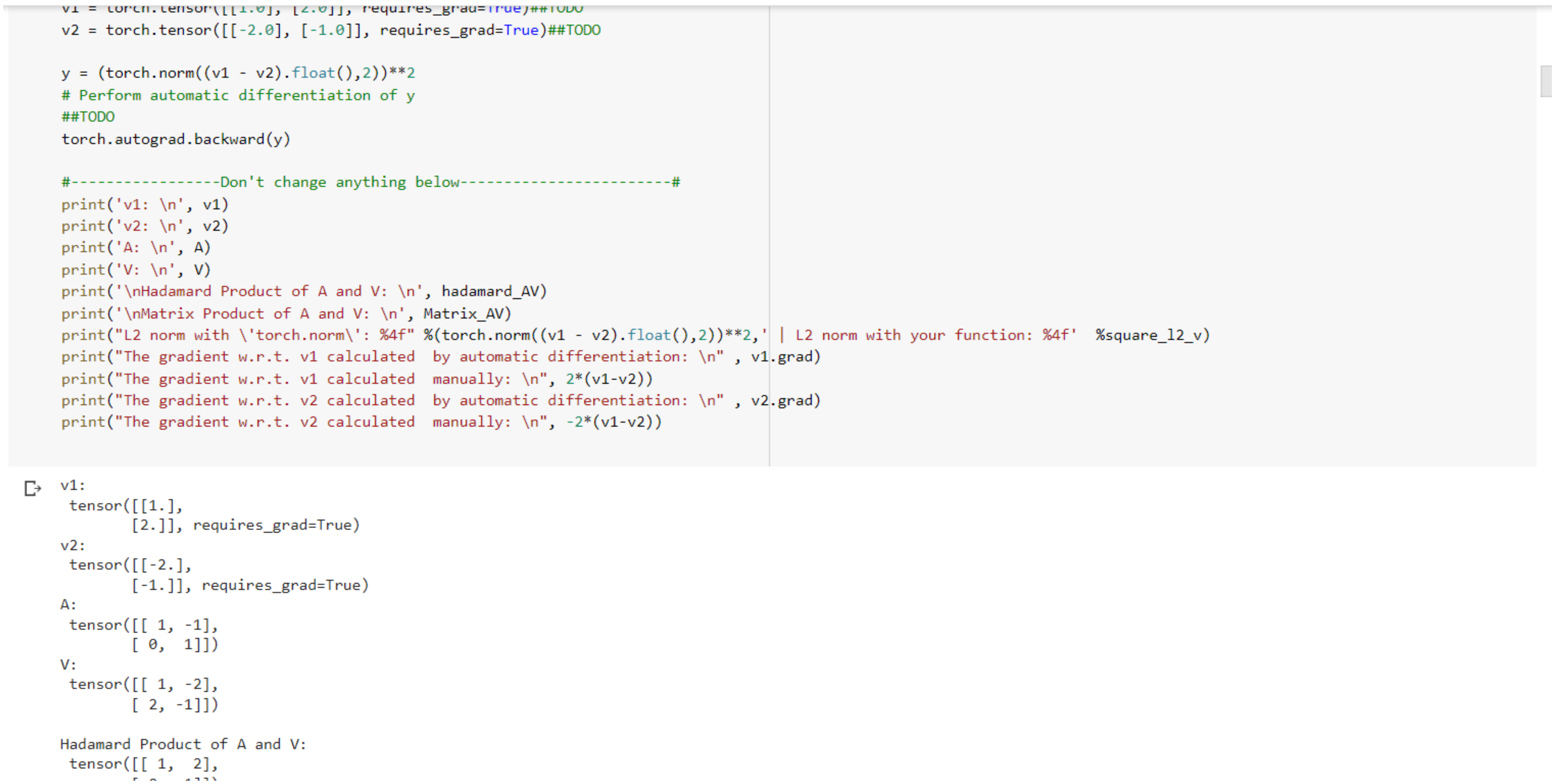
$$A = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}, \mathbf{v}_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \text{and } \mathbf{v}_2 = \begin{bmatrix} -2 \\ -1 \end{bmatrix}, V = \begin{bmatrix} | & | \\ \mathbf{v}_1 & \mathbf{v}_2 \\ | & | \end{bmatrix}.$$

You are provided with the template code below. Do not change the parts indicated. Your task below is to:
(a) Create the vectors $\mathbf{v}_1$, $\mathbf{v}_2$, matrices A and V above as `tensor` objects. $\mathbf{v}_1$ and $\mathbf{v}_2$ should have shape `torch.Size([2, 1])`, and A and V should have shape `torch.Size([2, 2])`. *Note: Data shape is important in `numpy` and `torch`. Although (2, 1) and (2,) both have 2 elements in the data, some operations are only valid in one of the forms.*
(b) Write code to create the Hadamard product of A and V
(c) Write code to create the Matrix product AV.
(d) Write your own code to compute the **square** of L2 norm $\|\mathbf{v}_1 - \mathbf{v}_2\|_2^2$. In this part, do not use the `torch.norm` function.
(e) Given $y = \|\mathbf{v}_1 - \mathbf{v}_2\|_2^2$, you will implement automatic differentiation to compute gradients of y with respect to $\mathbf{v}_1$ and $\mathbf{v}_2$. First, re-create two tensors $\mathbf{v}_1$ and $\mathbf{v}_2$ with requires_grad=True. This signals to autograd that every operation on the tensors should be tracked. Thus, the

two tensors $\mathbf{v}_1$ and $\mathbf{v}_2$ with requires_grad=True. This signals to autograd that every operation on the tensors should be tracked. Thus, the gradients with respect to the tensors can be automatically computed using chain rule. Then, write the code to perform automatic differentiation of y using function torch.autograd.backward. In the printing functions below, `v1.grad` and `v2.grad` automatically compute $\frac{\partial y}{\partial \mathbf{v}_1}$ and $\frac{\partial y}{\partial \mathbf{v}_2}$, respectively. Compare them with the printed manual gradients calculation $\frac{\partial y}{\partial \mathbf{v}_1} = 2(\mathbf{v}_1 - \mathbf{v}_2)$ and $\frac{\partial y}{\partial \mathbf{v}_2} = -2(\mathbf{v}_1 - \mathbf{v}_2)$, do they match?

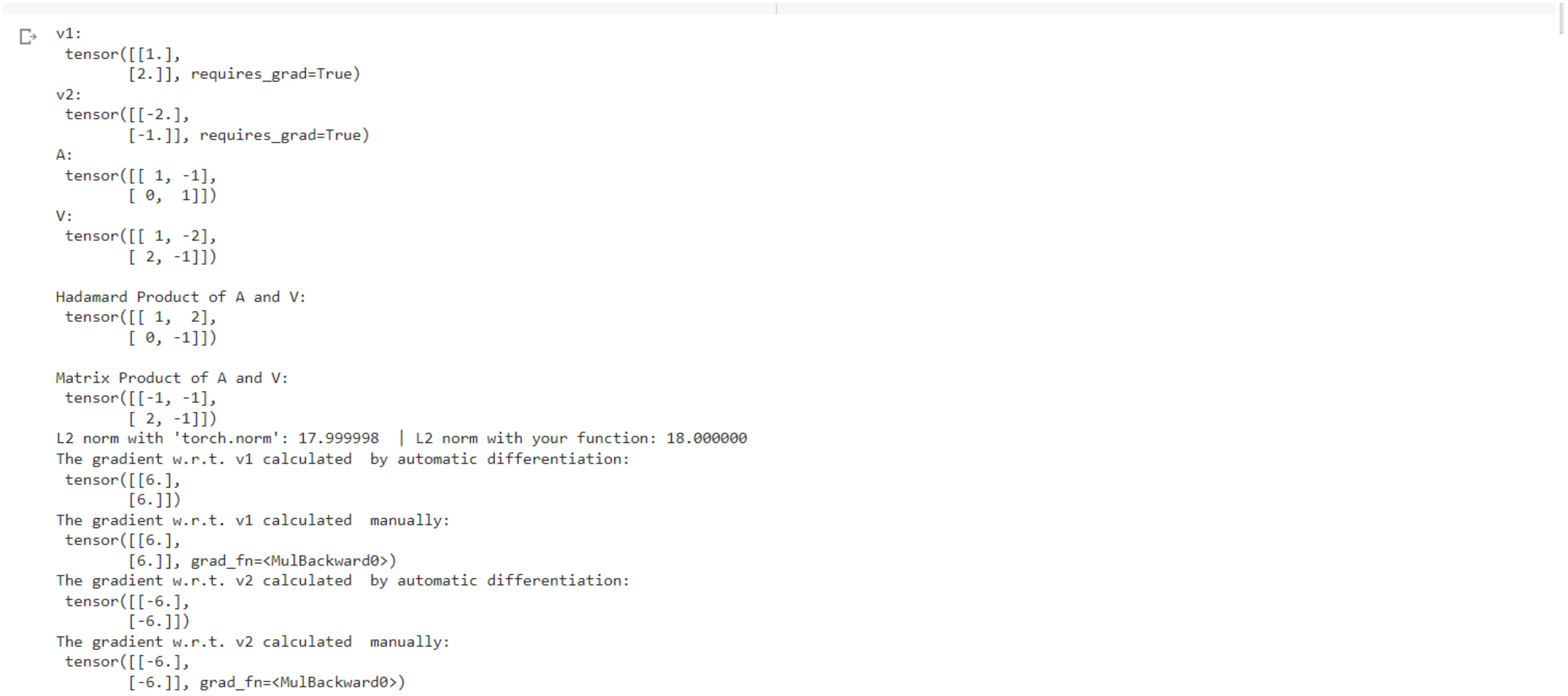
Problem 1 (a)-(e) Solution

```
[2]  ## Problem 1

     # Import Pytorch Libraries
     import torch

     ## part (a) Create matrices
     v1 = torch.tensor([[1], [2]])
     v2 = torch.tensor([[-2], [-1]])
     A = torch.tensor([[1, -1], [0, 1]])##TODO # define A
     V = torch.tensor([[1, -2], [2, -1]])##TODO # define V

     ## part (b) Compute the hadamard product
     hadamard_AV = torch.mul(A, V)##TODO   #  hadamard product of A and V

     ## part (c) Compute the matrix product
     Matrix_AV = torch.matmul(A, V)##TODO   #  matrix product of A and V

     ## part (d) Compute the L2 norm
     square_l2_v = torch.sum(torch.square(v1-v2))##TODO   #  the L2 norm of v1 - v2

     # ## part (e)
     # # Create tensors and keep track of operations on them
     v1 = torch.tensor([[1.0], [2.0]], requires_grad=True)##TODO
```



```
v1 = torch.tensor([[1.0], [2.0]], requires_grad=True)##TODO
v2 = torch.tensor([[-2.0], [-1.0]], requires_grad=True)##TODO

y = (torch.norm((v1 - v2).float(),2))**2
# Perform automatic differentiation of y
##TODO
torch.autograd.backward(y)

#----------------Don't change anything below-----------------------#
print('v1: \n', v1)
print('v2: \n', v2)
print('A: \n', A)
print('V: \n', V)
print('\nHadamard Product of A and V: \n', hadamard_AV)
print('\nMatrix Product of A and V: \n', Matrix_AV)
print("L2 norm with \'torch.norm\': %4f" %(torch.norm((v1 - v2).float(),2))**2,' | L2 norm with your function: %4f'  %square_l2_v)
print("The gradient w.r.t. v1 calculated  by automatic differentiation: \n" , v1.grad)
print("The gradient w.r.t. v1 calculated  manually: \n", 2*(v1-v2))
print("The gradient w.r.t. v2 calculated  by automatic differentiation: \n" , v2.grad)
print("The gradient w.r.t. v2 calculated  manually: \n", -2*(v1-v2))
```

```
v1: 
 tensor([[1.],
        [2.]], requires_grad=True)
v2: 
 tensor([[-2.],
        [-1.]], requires_grad=True)
A: 
 tensor([[ 1, -1],
        [ 0,  1]])
V: 
 tensor([[ 1, -2],
        [ 2, -1]])

Hadamard Product of A and V: 
 tensor([[ 1,  2],
```



```
v1:
 tensor([[1.],
        [2.]], requires_grad=True)
v2:
 tensor([[-2.],
        [-1.]], requires_grad=True)
A:
 tensor([[ 1, -1],
        [ 0,  1]])
V:
 tensor([[ 1, -2],
        [ 2, -1]])

Hadamard Product of A and V:
 tensor([[ 1,  2],
        [ 0, -1]])

Matrix Product of A and V:
 tensor([[-1, -1],
        [ 2, -1]])
L2 norm with 'torch.norm': 17.999998  | L2 norm with your function: 18.000000
The gradient w.r.t. v1 calculated  by automatic differentiation:
 tensor([[6.],
        [6.]])
The gradient w.r.t. v1 calculated  manually:
 tensor([[6.],
        [6.]], grad_fn=<MulBackward0>)
The gradient w.r.t. v2 calculated  by automatic differentiation:
 tensor([[-6.],
        [-6.]])
The gradient w.r.t. v2 calculated  manually:
 tensor([[-6.],
        [-6.]], grad_fn=<MulBackward0>)
```

Problem 1 (e) Solution

Yes they match

Problem 2 : Computational Graph and Backpropagation

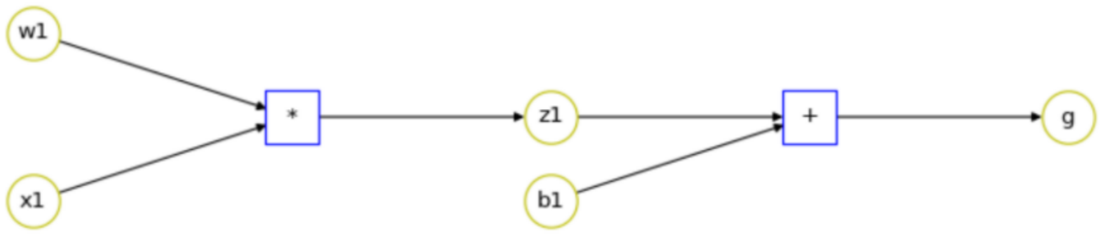
In Problem 1 (e), we have seen the autograd feature in `Pytorch` in a very simple function. Here, we will learn how to plot a computational graph and calculate backpropagation on a more complex function.

+ Code + Text

Problem 2 (a) Plot Computational Graph
Computationl Graph is a directional graph that defines all the arithmetic calculations in variables within the model. In `Pytorch`, the autograd feature uses dynamic computational graph mechanism to keep track of the gradient of every variable with `requires_grad=True` in the computational graph. Here is an overview about `Pyotrch` autograd feature. For example, we can plot the computational graph of

$$g(x_1) = w_1 x_1 + b_1$$

by:



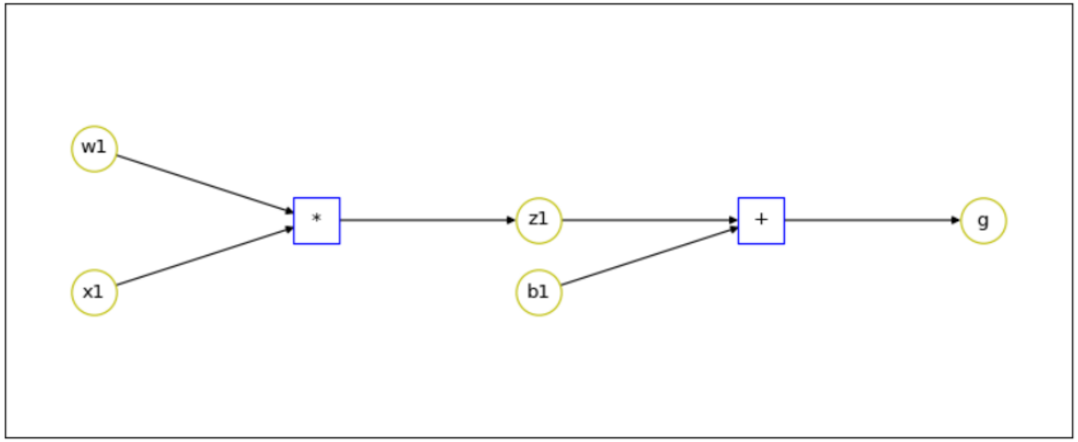
In this question, plot the computaional graph of

$$f(x_1, x_2, w_1, w_2) = e^{-(w_1 x_1 + w_2 x_2)}$$

You can plot it by hand or use `networkx` and `matplotlib` libraries. Use circles for variables and rectangles for arithmetic operations. Intermediate states should be labled as z_n.

```python
## Example Code of Computational Graph for g(x1)=w1x1+b1
import networkx as nx
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
figure(figsize=(12, 5), dpi=80)
class Vert:

    # default constructor
    def __init__(self, name, edges):
        self.name = name
        self.edges = edges

nodes = []
nodes.append(Vert('w1', ['mul1']))
nodes.append(Vert('x1', ['mul1']))
nodes.append(Vert('mul1', ['z1']))
nodes.append(Vert('z1', ['add1']))
nodes.append(Vert('b1', ['add1']))
nodes.append(Vert('add1', ['g']))
nodes.append(Vert('g', []))

G = nx.DiGraph()

for v  in nodes:
    G.add_node(v.name)
    for e in v.edges:
        G.add_edge(v.name, e)
```

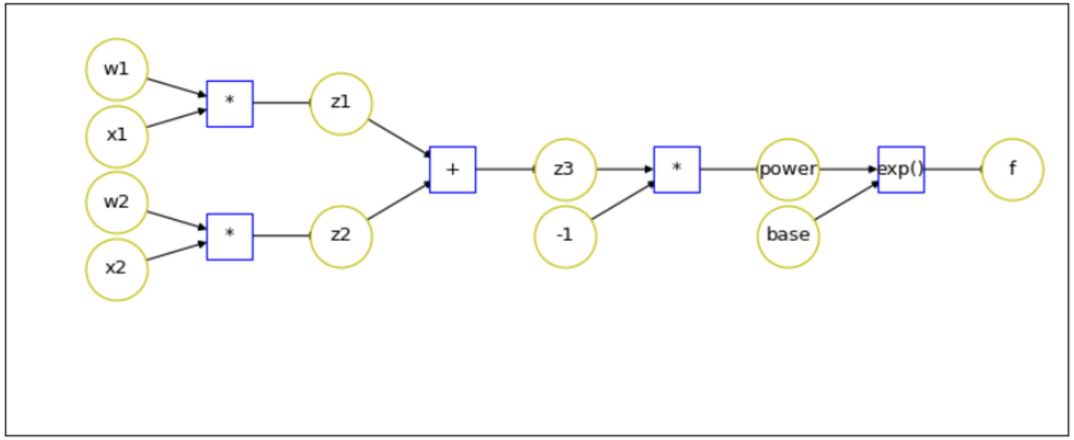
```
positions = {'w1':(-100, 10),
             'x1':(-100, -10),
             'mul1':(-50, 0),
             'z1': (0, 0),
             'b1': (0, -10),
             'add1':(50, 0),
             'g':(100, 0)}

labels = {'w1':'w1',
          'x1':'x1',
          'mul1':'*',
          'z1':'z1',
          'b1':'b1',
          'add1':'+',
          'g':'g'}

nx.draw_networkx_nodes(G, pos=positions, node_shape='o', nodelist=['w1', 'x1', 'z1', 'b1', 'g'], node_size=800, node_color='w', edgecolors='y')
nx.draw_networkx_nodes(G, pos=positions, node_shape='s', nodelist=['mul1', 'add1'], node_size=800, node_color='w', edgecolors='b')
nx.draw_networkx_labels(G, pos=positions, labels=labels)
nx.draw_networkx_edges(G, pos=positions, arrowsize=10, node_size=800)

plt.xlim(-120, 120)
plt.ylim(-30, 30)

plt.show()
plt.show()
```



Problem 2 (a) Solution

```
[ ] ## Problem 2 (a)

    # Plot computational graph by hand or using networkx and matplotlib libraries (or any library)
    ##TODO

    import networkx as nx
    import matplotlib.pyplot as plt
    from matplotlib.pyplot import figure
```

```python
figure(figsize=(12, 5), dpi=80)
class Vert:

    # default constructor
    def __init__(self, name, edges):
        self.name = name
        self.edges = edges

nodes = []
nodes.append(Vert('w1', ['mul1']))
nodes.append(Vert('x1', ['mul1']))
nodes.append(Vert('mul1', ['z1']))
nodes.append(Vert('z1', ['add1']))

nodes.append(Vert('w2', ['mul2']))
nodes.append(Vert('x2', ['mul2']))
nodes.append(Vert('mul2', ['z2']))
nodes.append(Vert('z2', ['add1']))

nodes.append(Vert('add1', ['z3']))
nodes.append(Vert('z3', ['inv1']))
nodes.append(Vert('inv1', ['z4']))
nodes.append(Vert('z4', ['exp1']))
nodes.append(Vert('exp1', ['f']))
nodes.append(Vert('f', []))

G = nx.DiGraph()

for v  in nodes:
    G.add_node(v.name)
    for e in v.edges:
        G.add_edge(v.name, e)

positions = {'w1':(-100, 10),
             'x1':(-100, -10),
                'mul1':(-50, 0),
                'z1': (0, 0),
                'w2':(-100, -30),
                'x2':(-100, -50),
                'mul2':(-50, -40),
                'z2': (0, -40),
                'add1':(50, -20),
                'z3':(100, -20),
                'inv1':(150, -20),
                'z4':(200, -20),
                'exp1':(250, -20),
                'f':(300, -20)}

labels = {'w1':'w1',
          'x1':'x1',
          'mul1':'*',
          'z1':'z1',
          'w2':'w2',
          'x2':'x2',
          'mul2':'*',
          'z2':'z2',
          'add1':'+',
          'z3':'z3',
          'inv1':'-',
          'z4':'z4',
          'exp1':'exp()',
          'f':'f'}

nx.draw_networkx_nodes(G, pos=positions, node_shape='o', nodelist=['w1', 'x1', 'z1','w2', 'x2', 'z2', 'z3','z4', 'f'], node_size=800, node_color='w', edgecolors='y')
nx.draw_networkx_nodes(G, pos=positions, node_shape='s', nodelist=['mul1','mul2', 'add1','inv1', 'exp1'], node_size=800, node_color='w', edgecolors='b')
nx.draw_networkx_labels(G, pos=positions, labels=labels)
nx.draw_networkx_edges(G, pos=positions, arrowsize=10, node_size=800)

plt.xlim(-120, 400)
plt.ylim(-100, 30)
plt.show()
```

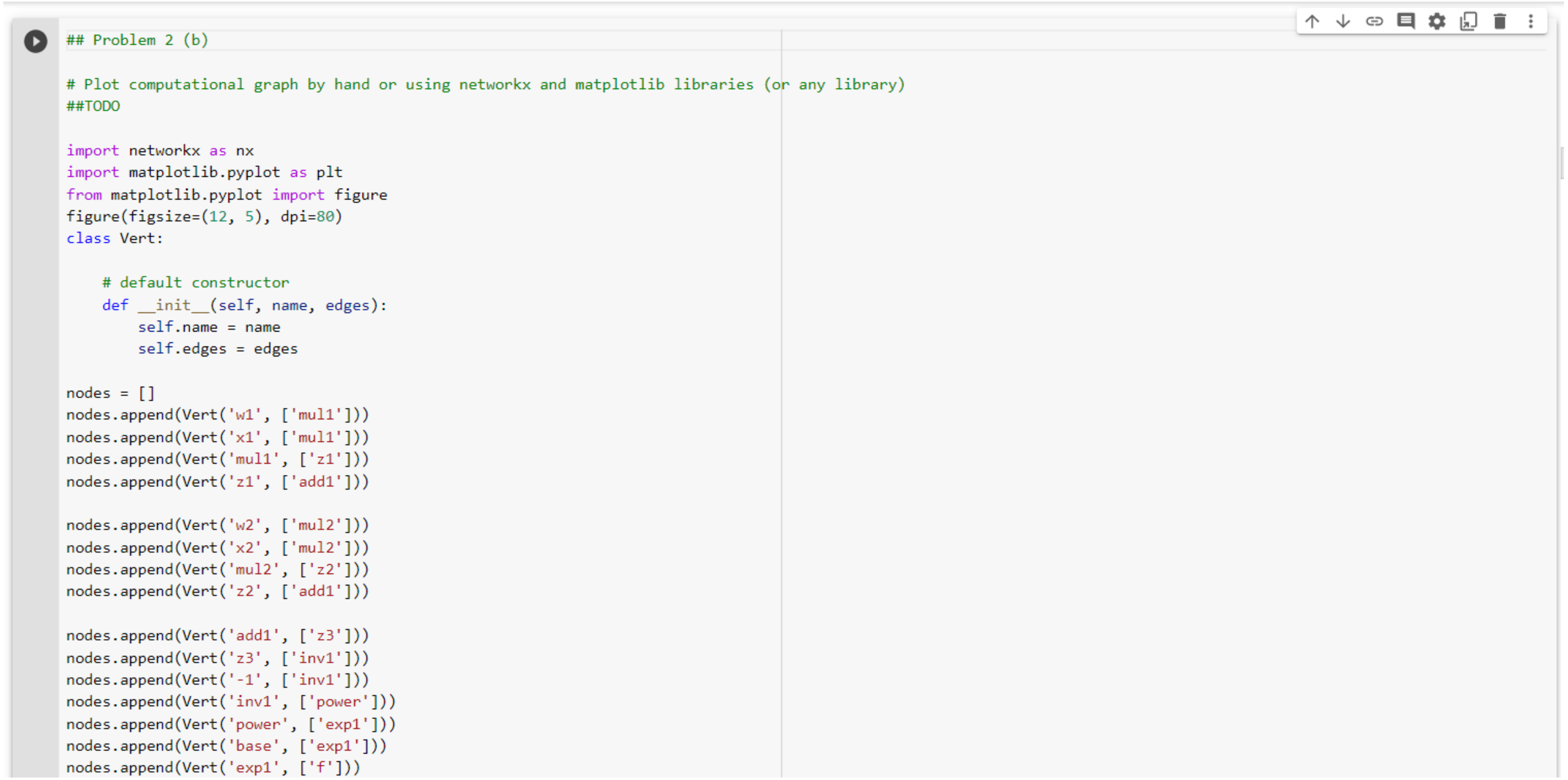


Problem 2 (b) Forward pass

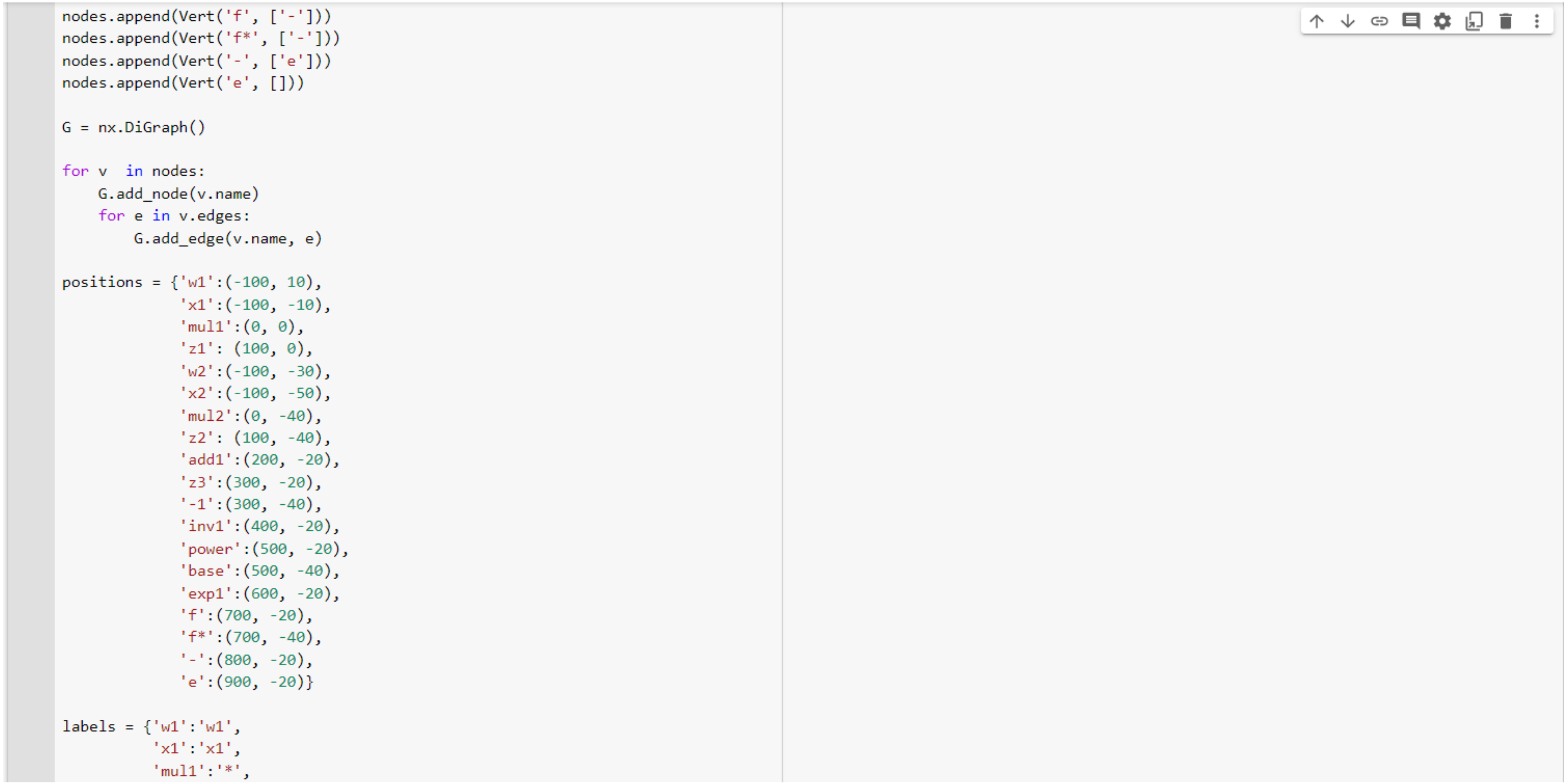
Assume we have $w_1 = 0.1$, $x_1 = 0.3$, $w_2 = -0.2$, $x_2 = 0.4$ and we have $f^* = 1$ which is the desired value of f. And we define error $e = f - f^*$. Calculate the value of f and e, and add f^* and e in the computationl graph.
(all numbers round to at least 3 decimals.)

Problem 2 (b) Solution

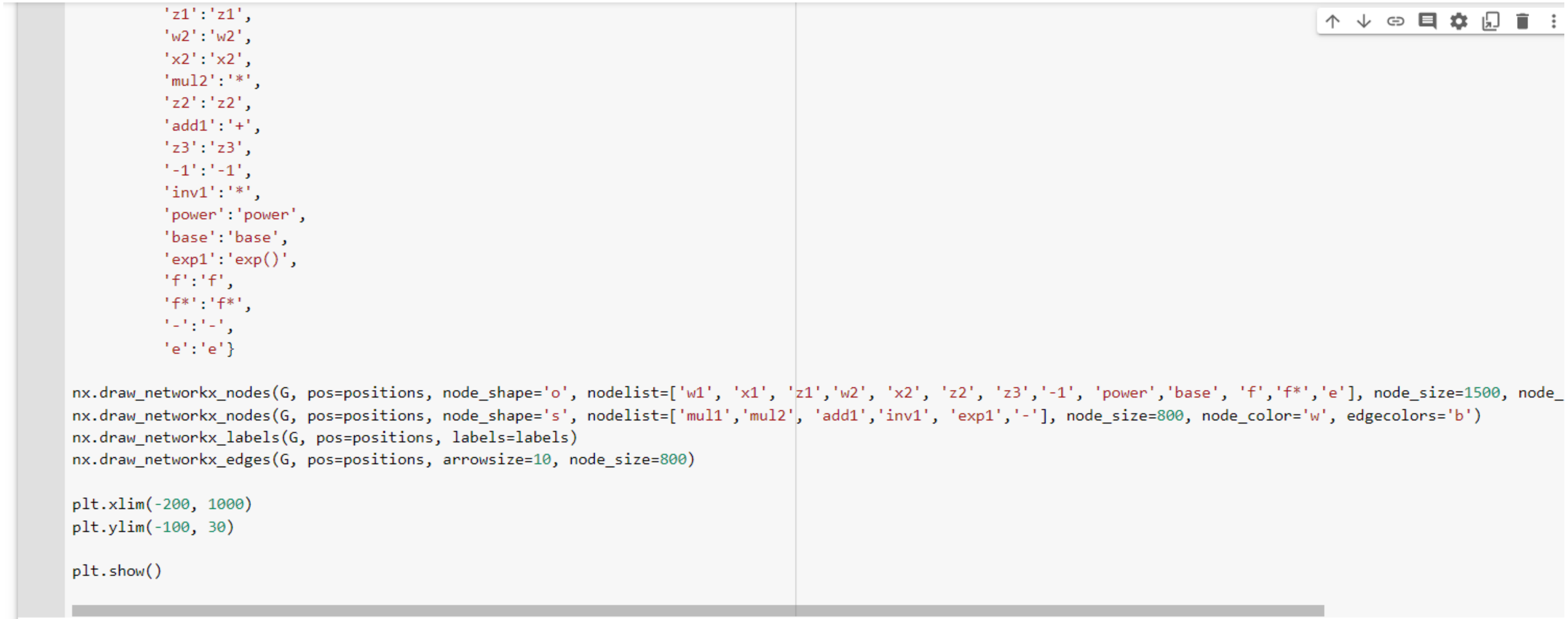
$$e = e^{-(0.1*0.3-0.2*0.4)} - f^* = 1.05127109638 - 1 = 0.05127109638$$



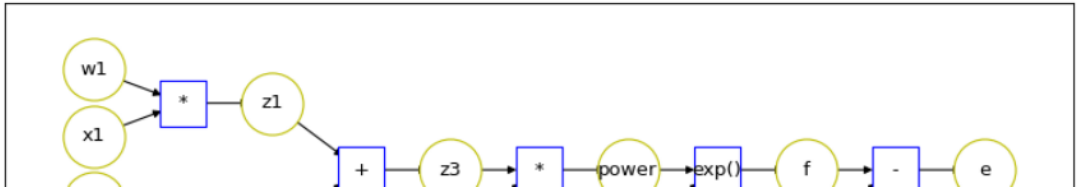
```
## Problem 2 (b)

# Plot computational graph by hand or using networkx and matplotlib libraries (or any library)
##TODO

import networkx as nx
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
figure(figsize=(12, 5), dpi=80)
class Vert:

    # default constructor
    def __init__(self, name, edges):
        self.name = name
        self.edges = edges

nodes = []
nodes.append(Vert('w1', ['mul1']))
nodes.append(Vert('x1', ['mul1']))
nodes.append(Vert('mul1', ['z1']))
nodes.append(Vert('z1', ['add1']))

nodes.append(Vert('w2', ['mul2']))
nodes.append(Vert('x2', ['mul2']))
nodes.append(Vert('mul2', ['z2']))
nodes.append(Vert('z2', ['add1']))

nodes.append(Vert('add1', ['z3']))
nodes.append(Vert('z3', ['inv1']))
nodes.append(Vert('-1', ['inv1']))
nodes.append(Vert('inv1', ['power']))
nodes.append(Vert('power', ['exp1']))
nodes.append(Vert('base', ['exp1']))
nodes.append(Vert('exp1', ['f']))
```



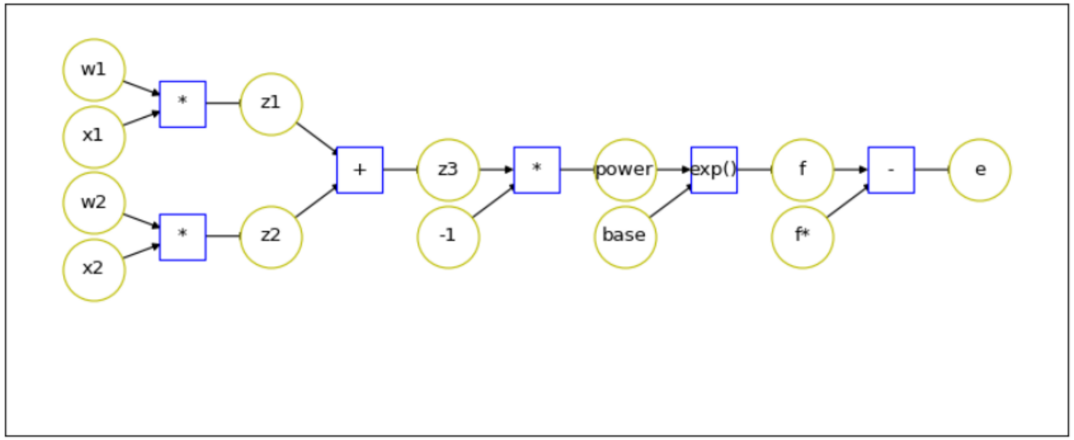
```python
nodes.append(Vert('f', ['-']))
nodes.append(Vert('f*', ['-']))
nodes.append(Vert('-', ['e']))
nodes.append(Vert('e', []))

G = nx.DiGraph()

for v  in nodes:
    G.add_node(v.name)
    for e in v.edges:
        G.add_edge(v.name, e)

positions = {'w1':(-100, 10),
             'x1':(-100, -10),
             'mul1':(0, 0),
             'z1': (100, 0),
             'w2':(-100, -30),
             'x2':(-100, -50),
             'mul2':(0, -40),
             'z2': (100, -40),
             'add1':(200, -20),
             'z3':(300, -20),
             '-1':(300, -40),
             'inv1':(400, -20),
             'power':(500, -20),
             'base':(500, -40),
             'exp1':(600, -20),
             'f':(700, -20),
             'f*':(700, -40),
             '-':(800, -20),
             'e':(900, -20)}

labels = {'w1':'w1',
          'x1':'x1',
          'mul1':'*',
```



```
          'z1':'z1',
          'w2':'w2',
          'x2':'x2',
          'mul2':'*',
          'z2':'z2',
          'add1':'+',
          'z3':'z3',
          '-1':'-1',
          'inv1':'*',
          'power':'power',
          'base':'base',
          'exp1':'exp()',
          'f':'f',
          'f*':'f*',
          '-':'-',
          'e':'e'}

nx.draw_networkx_nodes(G, pos=positions, node_shape='o', nodelist=['w1', 'x1', 'z1','w2', 'x2', 'z2', 'z3','-1', 'power','base', 'f','f*','e'], node_size=1500, node_
nx.draw_networkx_nodes(G, pos=positions, node_shape='s', nodelist=['mul1','mul2', 'add1','inv1', 'exp1','-'], node_size=800, node_color='w', edgecolors='b')
nx.draw_networkx_labels(G, pos=positions, labels=labels)
nx.draw_networkx_edges(G, pos=positions, arrowsize=10, node_size=800)

plt.xlim(-200, 1000)
plt.ylim(-100, 30)

plt.show()
```



```
plt.show()
```



Problem 2 (c) Backpropagation

In Lecture 15, we learned the idea of backpropagation. Follow the concepts in Lecture 15 page 16, we can calculate every of the gradient of the variable in the computational graph in Problem 2 (b) with chain rule. According to your computational graph, calculate the value of

(i) $\frac{\partial e}{\partial f}$

(ii) $\frac{\partial e}{\partial w_1}$

(iii) $\frac{\partial e}{\partial w_2}$

Problem 2 (c) Solution

(i) $\frac{\partial e}{\partial f}$ = 1

(ii) $\frac{\partial e}{\partial w_1} = \frac{\partial e}{\partial f} \times \frac{\partial f}{\partial w_1}$ = -0.3154

(iii) $\frac{\partial e}{\partial w_2} = \frac{\partial e}{\partial f} \times \frac{\partial f}{\partial w_2}$ = = -0.4205

Problem 2 (d) Verify with Pytorch

Now, verify $\frac{\partial e}{\partial w_1}$ and $\frac{\partial e}{\partial w_2}$ you calculate in part (c) using `Pytorch` in the cell below.

[68]
```
## Problem 2 (d)

import torch

w1 = torch.tensor([0.1], requires_grad=True)
x1 = torch.tensor([0.3])

w2 = torch.tensor([-0.2], requires_grad=True)
x2 = torch.tensor([0.4])

f_star = torch.tensor([1.0])
#----------------Don't change anything above----------------------#
##TODO
f = torch.exp(-(w1*x1+w2*x2))
e = f - f_star
e.backward()
dedw1 = w1.grad
dedw2 = w2.grad
#----------------Don't change anything below----------------------#
print('de/dw1:')
```

```
    print(dedw1)
    print('de/dw2:')
    print(dedw2)
```

```
de/dw1:
tensor([-0.3154])
de/dw2:
tensor([-0.4205])
```

Problem 3 : Jumpstart to CNN training with `Digits`

You may remember from assignment 4 that you were asked to perform image segmentation on the `Digits` dataset in `sklearn`. The `Digits` dataset contains images of handwritten digits $0, 1, \dots, 9$, each of size 8×8 pixels. We are now going to perform image classification with CNNs. In the process, you are going to learn how to set up a standard training and testing pipeline in `Pytorch`.

Problem 3 (a) Load Data from `sklearn`

Execute the code cell below to load the digits dataset into the workspace and divide it into training and testing sets. Use the `load_digits()` and `train_test_split` functions provided in `sklearn` for this purpose.

Problem 3 (a) Solution

```
[69] ## Problem 3 (a)

     # imports
     from sklearn.datasets import load_digits
     from sklearn.model_selection import train_test_split
     import numpy as np
     import matplotlib.pyplot as plt
     from torch.utils.data import Dataset, DataLoader
```

```
import torch
import torch.nn as nn

# load data
digit = load_digits()
X, y = digit.data, digit.target##TODO


#----------------Don't change anything below-----------------------#

# create train test splits
num_train = 500
num_test = X.shape[0] - num_train
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=num_test, train_size=num_train, random_state=4803)

# print(torch.FloatTensor(X[5].reshape((1,8,8))).shape)
```

Problem 3 (b) Implement Custom `Pytorch` Dataset

Training neural networks in `Pytorch` requires one to instantiate objects called *Dataloaders*, the purpose of which is to extract raw data and present it in the appropriate form to the neural networks. This requires one to set up a custom *Dataset* class and afterwards define its `__getitem__()` and `__len__()` functions. The dataset class is then input to the `torch.utils.data.DataLoader` object to create a data pipeline. This is where one may define various attributes like the batch size, if one wants the batches to be presented in a shuffled order, and so on. The `Pytorch` documentation page here provides an excellent overview of this process. You are provided the template for the *Digits* dataset class and its various functions below. Use the function documentations and helps to guide you to complete the dataset class definition.

Problem 3 (b) Solution

```
[70] ## Problem 3 (b)
```

```python
# set up custom dataset class
class Digits(Dataset):
  def __init__(self, X, y):
    """Function stores the data and label arrays returned by load_digits function.

    Parameters
    ----------
    X : array_like, shape(Num_samples, Num_of_features)
      numpy array containing the data matrix containing digits training examples
      and features.

    y : array_like, shape(num_samples)
      numpy array containing labels from 0,1,...,9 for each training sample in X
    """

    self.X = X
    self.y = y

  def __getitem__(self, index):
    """function extracts a single example from X and the label given its index.

    Parameters
    ----------
    index : int
      index of a single example to be extracted from X

    Returns
    -------
    input : torch.tensor, shape(1, 8, 8), type torch.float
      indexed example from X reshaped into a single channel grayscale image of
      size 8 x 8 and float datatype.

    target : int, dtype torch.longtensor
      label for input returned as an integer of torch.longtensor datatype
     """
    ##TODO
    # input = torch.FloatTensor(self.X[index].reshape((1,8,8)))
    # # print(input.dtype)
    # target = torch.LongTensor(self.y[index])
    # print(target.dtype)
    input = torch.tensor(self.X[index])
    input = input.type(torch.float)
    input = torch.reshape(input, (1,8,8))
    target = torch.tensor(self.y[index])
    target = target.type(torch.LongTensor)


    return input, target

  def __len__(self):
    return self.X.shape[0]


#----------------Don't change anything below-----------------------#

train_batch_size = 100
test_batch_size = 1

train_dataset = Digits(X_train, y_train)
trainloader = DataLoader(train_dataset, batch_size=train_batch_size, shuffle=True)

test_dataset = Digits(X_test, y_test)
testloader = DataLoader(test_dataset, batch_size=test_batch_size, shuffle=True)
```

Problem 3 (c) Define a CNN

The next stage in the process involves defining a neural network class inheriting from the `torch.nn.Module` parent class. The user is required to initialize the network and then define its `forward()` function. The `Pytorch` documentation page here gives an excellent example of this. For our purposes, we are going to use a simple 3-layer network to classify the images presented. The details of the layers are given below:

- 2D convolution layer, kernel size = 3, input channels = 1 , output channels = 16
- ReLU
- 2D convolution layer, kernel size = 3, input channels = 16 , output channels = 32
- ReLU
- Maxpool, reduces input dimensions by half i.e., from 8×8 to 4×4
- Linear layer, maps the 16 pixels in each 4×4 image to a 10-element vector, corresponding to the number of possible output labels.

Complete the class definition for the neural network given below as per the instructions above.

Problem 3 (c) Solution

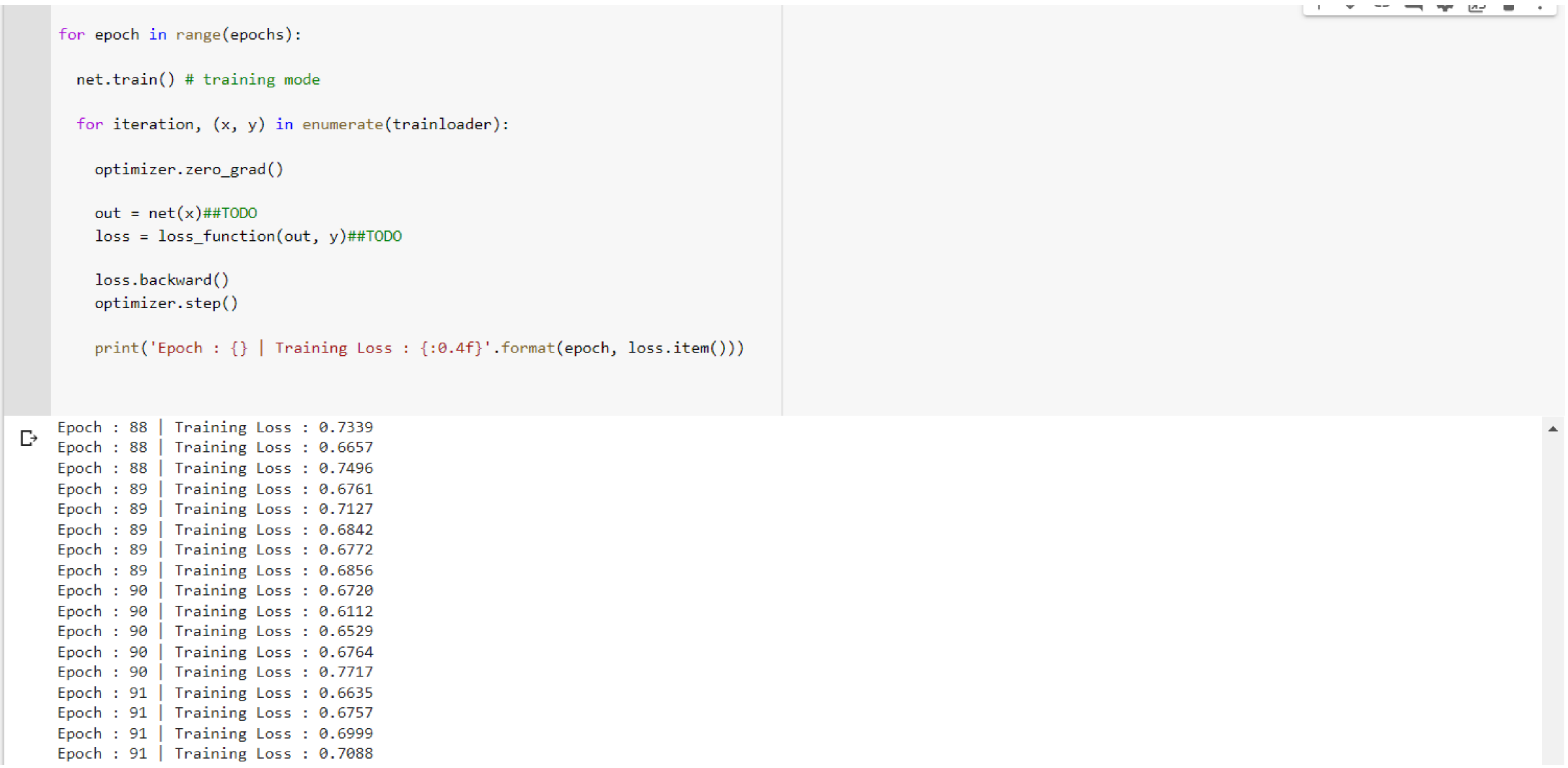
```
## Problem 3 (c)

# define network
class MySimpleCNN(torch.nn.Module):
  def __init__(self):
    """Inititalizes the various layers in the network"""
    super(MySimpleCNN, self).__init__()
    ##TODO
    # self.conv1 = nn.Conv2d(in_channels = 1, out_channels = 16, kernel_size = 3, stride = 1, padding = 1)
    # self.conv2 = nn.Conv2d(in_channels = 16, out_channels = 32, kernel_size = 3, stride = 1, padding = 1)
    # self.linear1 = nn.Linear(128, 10)
    # self.relu = nn.ReLU()
    # self.maxpool = nn.MaxPool2d(kernel_size = 2, stride = 2)
    self.conv1 = nn.Conv2d(1, 16, 3, 1, 1)
    self.conv2 = nn.Conv2d(16, 32, 3, 1, 1)
    self.linear1 = nn.Linear(128, 10)
    self.relu = nn.ReLU()
    self.maxpool = nn.MaxPool2d(2,2)

  def forward(self, x):
    """Processes the input from the dataloaders to return predicted output
    probability vectors for each example in the batch.

    Parameters
    ----------
    x : torch.tensor, shape(batch_size, 1, 8, 8), dtype torch.float
      output from dataloade containing batch_size number of 8 x 8 images as
      torch tensors

    Returns
    -------
    out : torch.tensor, shape (batch_size,10), dtype= torch.float.
      batch_size number of 10-element vectors for each image in the input batch.
    """
    ##TODO
    # x = self.relu(self.conv1(x))
    # x = self.maxpool(x)
    # x = self.relu(self.conv2(x))
    # x = self.maxpool(x)
    # x = torch.flatten(x, 1)
    # x = self.linear1(x)
    # x = self.relu(self.conv1(x)))
    x = self.maxpool(self.relu(self.conv1(x)))
    x = self.maxpool(self.relu(self.conv2(x)))

    x = torch.flatten(x, 1)
    x = self.linear1(x)


    out = torch.nn.functional.log_softmax(x, dim=1)

    return out
```

Problem 3 (d) Train the CNN

We now move to training the CNN. This requires defining objects for the loss function (Cross entropy in this case), the optimizer (any one of those you learnt in class), passing the network parameters to the optimizer object, and defining the learning rate. In the training loop, one is required to iterate through the training loader, predict the output vector (unnormalized) probabilities, compute the loss, backpropagate, and perform the gradient descent step. Complete the code below to execute the training step. A successfully training process would show the loss decreasing from a high starting value.

Problem 3 (d) Solution

[74]
```
## Problem 3 (d)

# perform training
lr = 1e-3
epochs = 100

# initliaze the network
net = MySimpleCNN()

loss_function = nn.CrossEntropyLoss() ##TODO
# loss_function = nn.NLLLoss()
# optimizer = torch.optim.Adam(net.parameters(), lr) ##TODO
optimizer = torch.optim.SGD(net.parameters(), lr = lr) ##TODO

for epoch in range(epochs):

  net.train() # training mode
```



```
for epoch in range(epochs):

  net.train() # training mode

  for iteration, (x, y) in enumerate(trainloader):

    optimizer.zero_grad()

    out = net(x)##TODO
    loss = loss_function(out, y)##TODO

    loss.backward()
    optimizer.step()

    print('Epoch : {} | Training Loss : {:0.4f}'.format(epoch, loss.item()))
```

```
Epoch : 88 | Training Loss : 0.7339
Epoch : 88 | Training Loss : 0.6657
Epoch : 88 | Training Loss : 0.7496
Epoch : 89 | Training Loss : 0.6761
Epoch : 89 | Training Loss : 0.7127
Epoch : 89 | Training Loss : 0.6842
Epoch : 89 | Training Loss : 0.6772
Epoch : 89 | Training Loss : 0.6856
Epoch : 90 | Training Loss : 0.6720
Epoch : 90 | Training Loss : 0.6112
Epoch : 90 | Training Loss : 0.6529
Epoch : 90 | Training Loss : 0.6764
Epoch : 90 | Training Loss : 0.7717
Epoch : 91 | Training Loss : 0.6635
Epoch : 91 | Training Loss : 0.6757
Epoch : 91 | Training Loss : 0.6999
Epoch : 91 | Training Loss : 0.7088
```

```
Epoch : 91 | Training Loss : 0.6615
Epoch : 92 | Training Loss : 0.6584
Epoch : 92 | Training Loss : 0.6536
Epoch : 92 | Training Loss : 0.6755
Epoch : 92 | Training Loss : 0.6895
Epoch : 92 | Training Loss : 0.6397
Epoch : 93 | Training Loss : 0.6824
Epoch : 93 | Training Loss : 0.5726
Epoch : 93 | Training Loss : 0.6316
Epoch : 93 | Training Loss : 0.7124
Epoch : 93 | Training Loss : 0.6818
Epoch : 94 | Training Loss : 0.6276
Epoch : 94 | Training Loss : 0.6369
Epoch : 94 | Training Loss : 0.7182
Epoch : 94 | Training Loss : 0.5831
Epoch : 94 | Training Loss : 0.6649
Epoch : 95 | Training Loss : 0.6086
Epoch : 95 | Training Loss : 0.6465
Epoch : 95 | Training Loss : 0.6375
Epoch : 95 | Training Loss : 0.6527
Epoch : 95 | Training Loss : 0.6590
Epoch : 96 | Training Loss : 0.5445
Epoch : 96 | Training Loss : 0.6404
Epoch : 96 | Training Loss : 0.5827
Epoch : 96 | Training Loss : 0.6782
Epoch : 96 | Training Loss : 0.7155
Epoch : 97 | Training Loss : 0.6153
Epoch : 97 | Training Loss : 0.6227
Epoch : 97 | Training Loss : 0.6393
Epoch : 97 | Training Loss : 0.6331
Epoch : 97 | Training Loss : 0.6173
Epoch : 98 | Training Loss : 0.5952
Epoch : 98 | Training Loss : 0.6150
Epoch : 98 | Training Loss : 0.5947
Epoch : 98 | Training Loss : 0.6148
Epoch : 98 | Training Loss : 0.6801
Epoch : 99 | Training Loss : 0.6309
Epoch : 99 | Training Loss : 0.5915
Epoch : 99 | Training Loss : 0.5829
```

Problem 3 (e) Evaluation Test Result

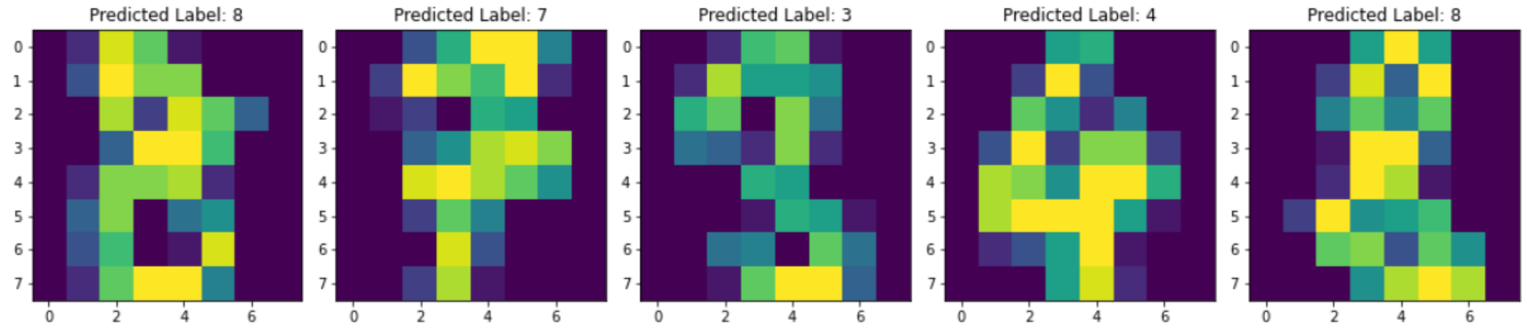
Finally, we move onto test evaluation using the trained CNN. After training the CNN, the trained parametes will be saved in the CNN object which is our `net` variable here. You simply execute the cell below to visualize randomly sampled images in the test set and the network predictions on those images. Execute the cell multiple times. Are you getting consistently good predictions for all images? Explain some of the reasons why the network is able to learn to classify so well on the Digits dataset with reference to the particular characteristics of the images in the dataset.

Problem 3 (e) Solution

- Yes, I am getting consistently good predictions for all images
- It predicts so well because these are limited range of numbers from 0~ 10 and are farily simple to read.

```
[73] ## Problem 3 (e)

     # perform inference and visualize predictions
     net.eval() # testing mode

     num_images = 5
     fig, axes = plt.subplots(1, num_images, figsize=(15,8))

     for i, ax in zip(range(num_images), axes.flatten()):
       x, _ = next(iter(testloader))
       out = net(x)
       ax.imshow(x.detach().cpu().numpy().squeeze())
       ax.set_title('Predicted Label: {}'.format(out.argmax().item()))

     plt.tight_layout()
```

```
    plt.show()
```

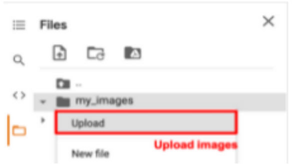


Problem 4: Image Classification Using Pre-trained Models

You have learned the pipeline of how to train a CNN on a specific dataset. In practice, we do not need to train an entire model from scratch, because it is relatively rare to have a dataset of sufficient size. Instead, it is common to use a pre-trained deep model that is trained on a very large dataset such as `ImageNet` to address your task of interest. In this exercise, you will learn how to directly use pre-trained deep architectures to classify your own images. You can choose photos that you already took or find photos from the web.

Problem 4 (a) Upload Images

Create a folder named `my_images` using the code below (`!mkdir my_images`) to store all your uploaded images. Then, upload your own images (at least 12 images) to the created `my_images` folder as the step shown below.



Problem 4 (a) Solution

```
[75] ## Problem 4 (a)

     # create a folder named `my_images`
     !mkdir my_images
```

```
mkdir: cannot create directory ‘my_images’: File exists
```

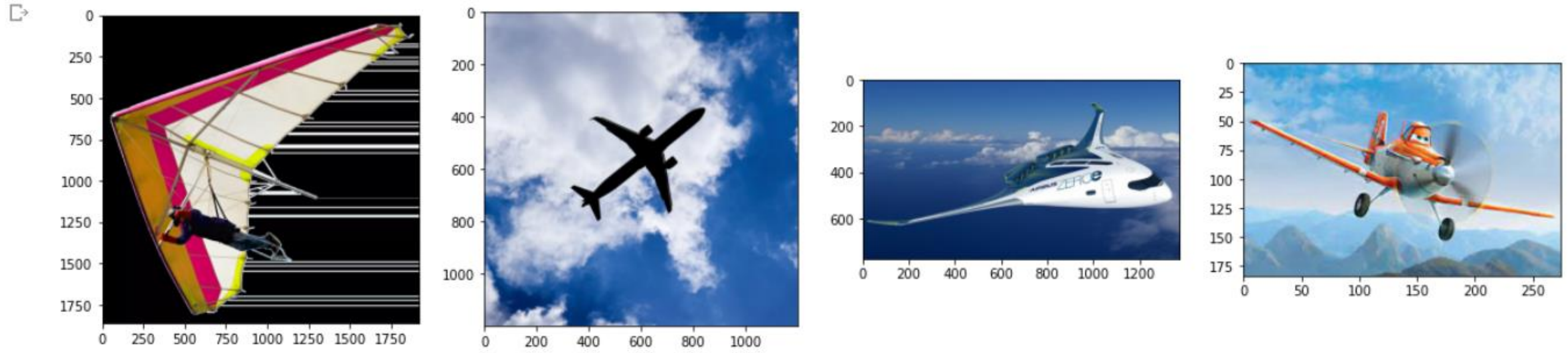
Problem 4 (b) Visualize Images
Visualize your uploaded images by simply executing the code below.

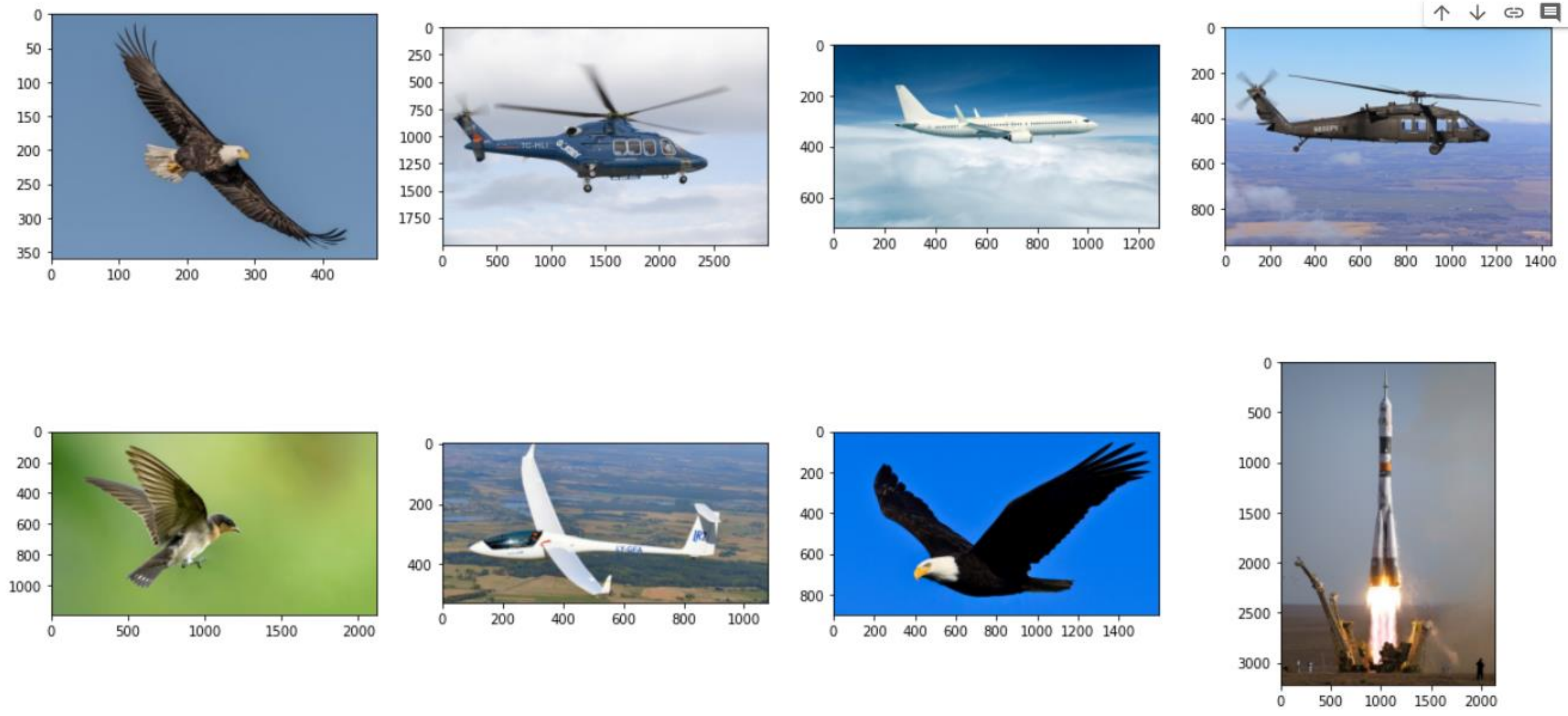
Problem 4 (b) Solution

```
[ ] ## Problem 4 (b)

     from PIL import Image
     import matplotlib.pyplot as plt
     from glob import glob
     import os
```

Problem 4 (b) Solution

```
## Problem 4 (b)

from PIL import Image
import matplotlib.pyplot as plt
from glob import glob
import os
import numpy as np

def imread(img_dir):
  # read the images into a list of `PIL.Image` objects
  images = []
  for f in glob(os.path.join(img_dir, "*")):
    images.append(Image.open(f).convert('RGB'))

  return images

def vis_img_label(image_list, label_list=None):
  # visualize the images w/ labels
  Tot = len(image_list)
  Cols = 4
  Rows = Tot // Cols
  Rows += (Tot % Cols)>0
  if label_list is None:
    label_list = [""]*Tot
  # Create a Position index
  Position = range(1,Tot + 1)
  fig = plt.figure(figsize=(Cols*5, Rows*5))
  for i in range(Tot):
    image = image_list[i]
    # add every single subplot to the figure
    ax = fig.add_subplot(Rows,Cols,Position[i])
    ax.imshow(np.asarray(image))
    ax.set_title(label_list[i])


## Load your uploaded images
img_dir = "/content/my_images"
image_list = imread(img_dir)
## visualize your uploaded images
vis_img_label(image_list)
```





Problem 4 (c) Define Custom Dataset

In this part, you will learn how to prepare your image data for classification by constructing a customized PyTorch dataloader. The customized dataset is provided below as a template class `myDataset`. The class `myDataset` overrides the following methods:

- `len` that returns the size of `myDataset` (the length of the `img list`)

- `__len__` that returns the size of `myDataset` (the length of the `img_list`)
- `__getitem__` to support the indexing such that `myDataset[i]` can be used to get the i-th sample of `img_list`.

Your tasks are:

- Write your code to return the size of `myDataset` in the method `__len__`.
- Write your code to index the i-th sample of `img_list` in the method `__getitem__`.

You may find more details about torch.utils.data.Dataset for constructing a customized PyTorch dataset.

Problem 4 (c) Solution

```
## Problem 4 (c)

from torch.utils.data import Dataset, DataLoader
from torchvision import transforms

# customized pytorch dataset
class myDataset(Dataset):
  def __init__(self, img_list, data_transform=None):
    self.img_list = img_list   # the list of all uploaded Images
    self.length = len(img_list)
    self.data_transform = data_transform

  def __getitem__(self, index):
    """function extracts a single example from img_list given its index.

    Parameters
    ----------
    index : int
      index of a single example to be extracted from img_list
    Returns
    -------
    img : torch.tensor, shape(3, 224, 224), type torch.float
      indexed example from img_list reshaped into a RGB channel image of
      size 224 x 224 and float datatype.
    """
    # convert_tensor = transforms.ToTensor()
    img = self.img_list[index]
    # img = img.type(torch.float)
    # img = torch.reshape(input, (3, 224, 224))


    # img = ##TODO

    if self.data_transform is not None:
        img =self.data_transform(img)  # apply data transformations
    assert img.shape == (3, 224, 224)

    return img

  def __len__(self):
    """
    Returns
    -------
    length : int
      length of img_list.
    """
    length = len(self.img_list) ##TODO

    return length

#----------------Don't change anything below-----------------------#

data_transform = transforms.Compose([transforms.Resize((224, 224)),
                                     transforms.ToTensor()
```

```
data_transform = transforms.Compose([transforms.Resize((224, 224)),
                                     transforms.ToTensor(),
                                     transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                                          std=[0.229, 0.224, 0.225])])

my_dataset = myDataset(image_list, data_transform)
my_dataloader = DataLoader(my_dataset, batch_size=64, shuffle=False, num_workers=2)

#----verify the customized dataset `myDataset`----#
print('The size of the dataset: ', len(my_dataset))
print('The dimension of the first image sample after transformations: ', my_dataset[0].shape)
```

```
The size of the dataset:  12
The dimension of the first image sample after transformations:  torch.Size([3, 224, 224])
```

+ Code + Text

Problem 4 (d) Load AlexNet

You have completed the data preparation by constructing a customized dataloader. Now you will start to use a pre-trained model for classifying your uploaded images. The provided code for this part loads the pre-trained AlexNet using the `torchvision.models` module. The pre-trained model is constructed by passing `pretrained=True`. Execute the code below as is and observe the printed AlexNet architecture. Does it match the AlexNet architecture illustrated in the slide 23 of lecture 17?

Problem 4 (d) Solution

[5]
```
## Problem 4 (d)

import torchvision

#Load the pre-trained AlexNet
```

```
alexnet = torchvision.models.alexnet(pretrained=True)
print(alexnet)   # print the model architecture
```

```
Downloading: "https://download.pytorch.org/models/alexnet-owt-7be5be79.pth" to /root/.cache/torch/hub/checkpoints/alexnet-owt-7be5be79.pth
100% 233M/233M [00:04<00:00, 53.7MB/s]
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

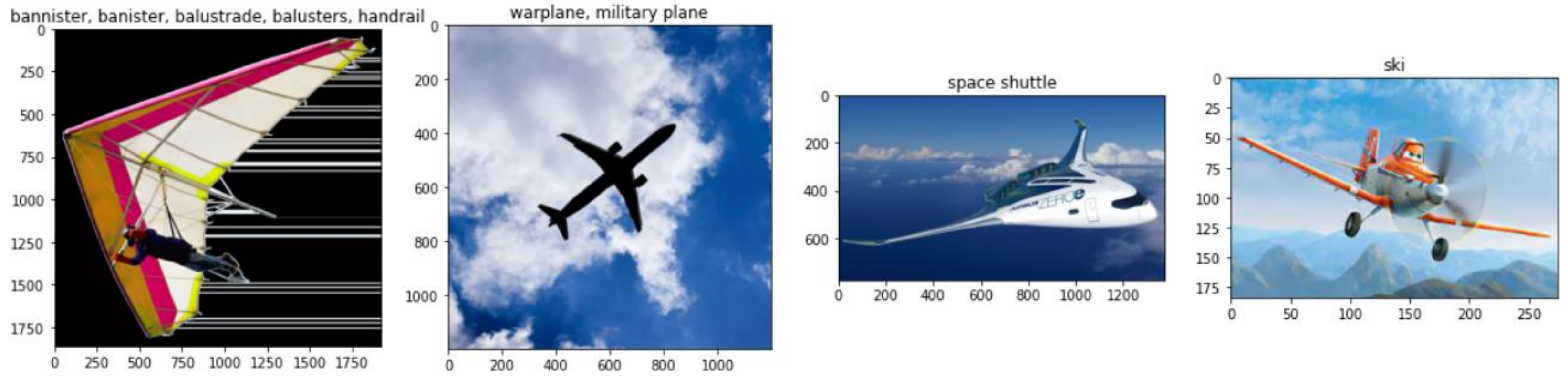
Problem 4 (e) Inference with AlexNet

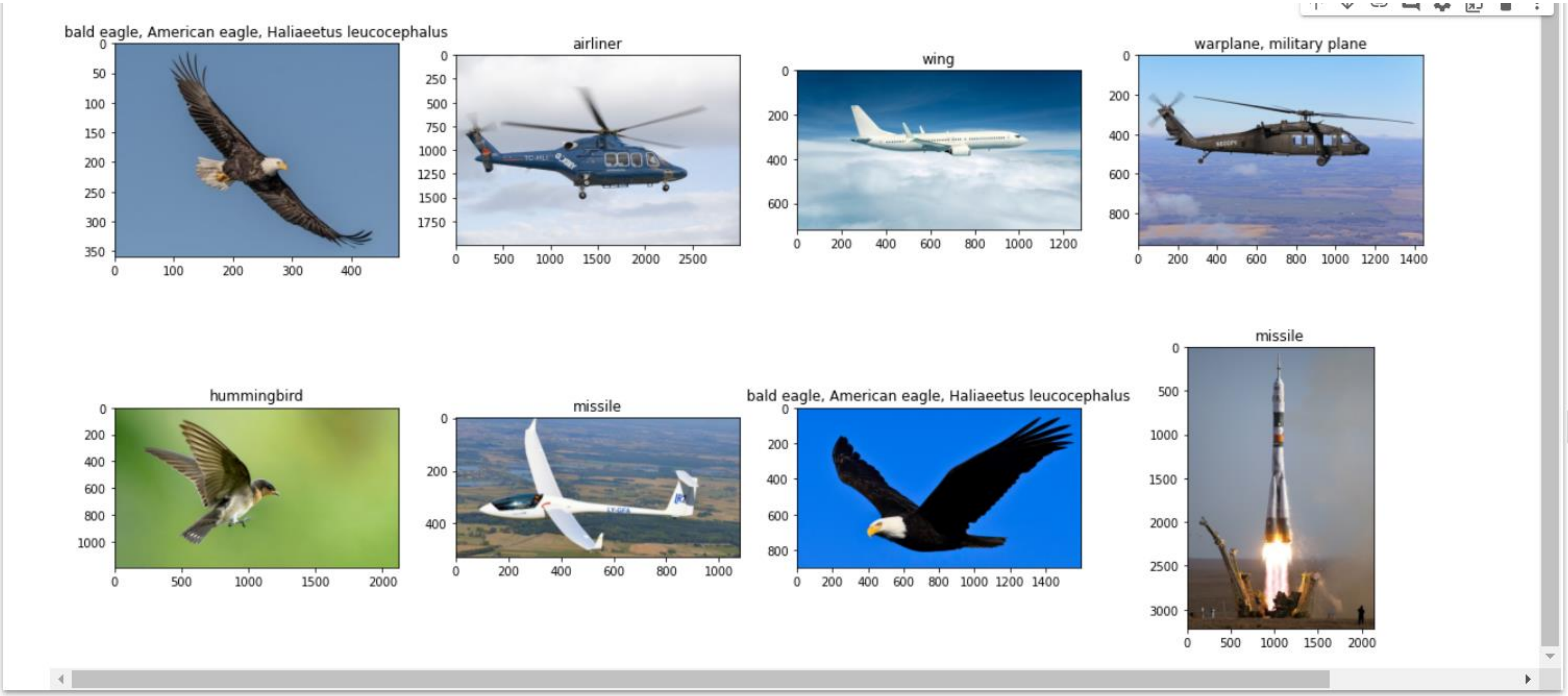
In this part, you will perform model inference to classify your uploaded images by the pre-trained AlexNet. Write your code to compute the predicted softmax probabilities in the function `predict(model, dataloader)`. You may consider to use the function torch.nn.functional.softmax. Run the code cell to visualize the images with predicted labels. Do these predictions make sense?

Problem 4 (e) Solution

```
[6] ## Problem 4 (e)

    # importing Pytorch Libraries
    import torch
    import numpy as np
    import matplotlib.pyplot as plt
    import torch.nn.functional as F

    !gdown --id 1bDrtvgX-ztIh7A46FQNvROS7bEVuYogn
    label_map = torch.load("/content/label_dict.pth")

    # ------------ Do NOT change anything above ------------ #

    def predict(model, dataloader):
      pred_total = []

      device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
      model.to(device)

      with torch.no_grad():
        model.eval()  # switch to inference mode
        for batch_id, img_tensor in enumerate(dataloader):

          img_tensor = img_tensor.to(device)
          logits = model(img_tensor) ##TODO
      output = torch.nn.functional.log_softmax(logits)##TODO
      pred_class_idx = output.argmax(dim=1)

      pred_total.append(pred_class_idx.data)

    pred_total = torch.cat(pred_total).cpu().numpy()
  return pred_total


predictions_alexnet = predict(alexnet, my_dataloader) # numpy array of predicted class labels
label_list_alexnet = [label_map[pred] for pred in predictions_alexnet]  # list of class names
print("------------- visualize the images with predicted class names by AlexNet  -------------")
vis_img_label(image_list, label_list_alexnet)
```

```
Downloading...
From: https://drive.google.com/uc?id=1bDrtvgX-ztIh7A46FQNvROS7bEVuYogn
To: /content/label_dict.pth
100% 33.1k/33.1k [00:00<00:00, 49.6MB/s]
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:26: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the call to incl
------------- visualize the images with predicted class names by AlexNet  -------------
```




bald eagle, American eagle, Haliaeetus leucocephalus
airliner
wing
warplane, military plane
hummingbird
missile
bald eagle, American eagle, Haliaeetus leucocephalus
missile

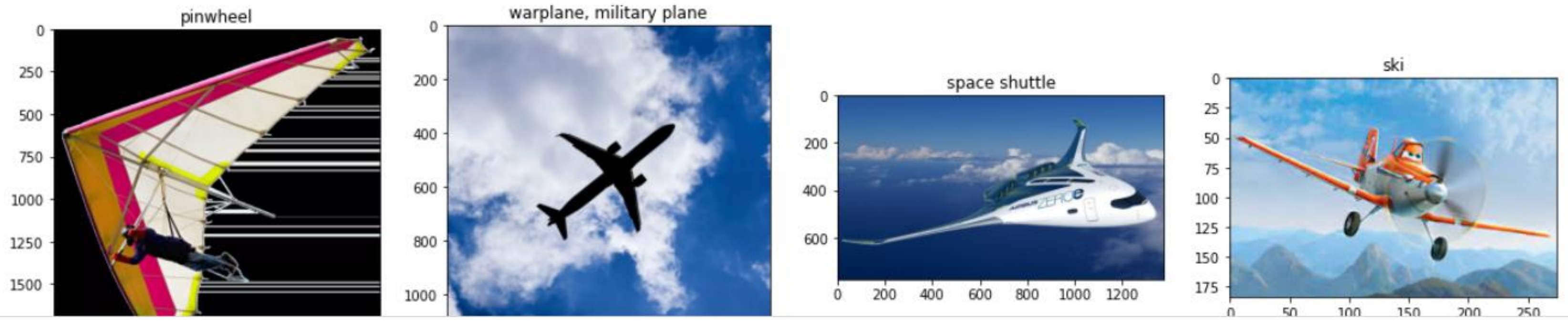
Problem 4 (f) Vgg16 and Resnet18

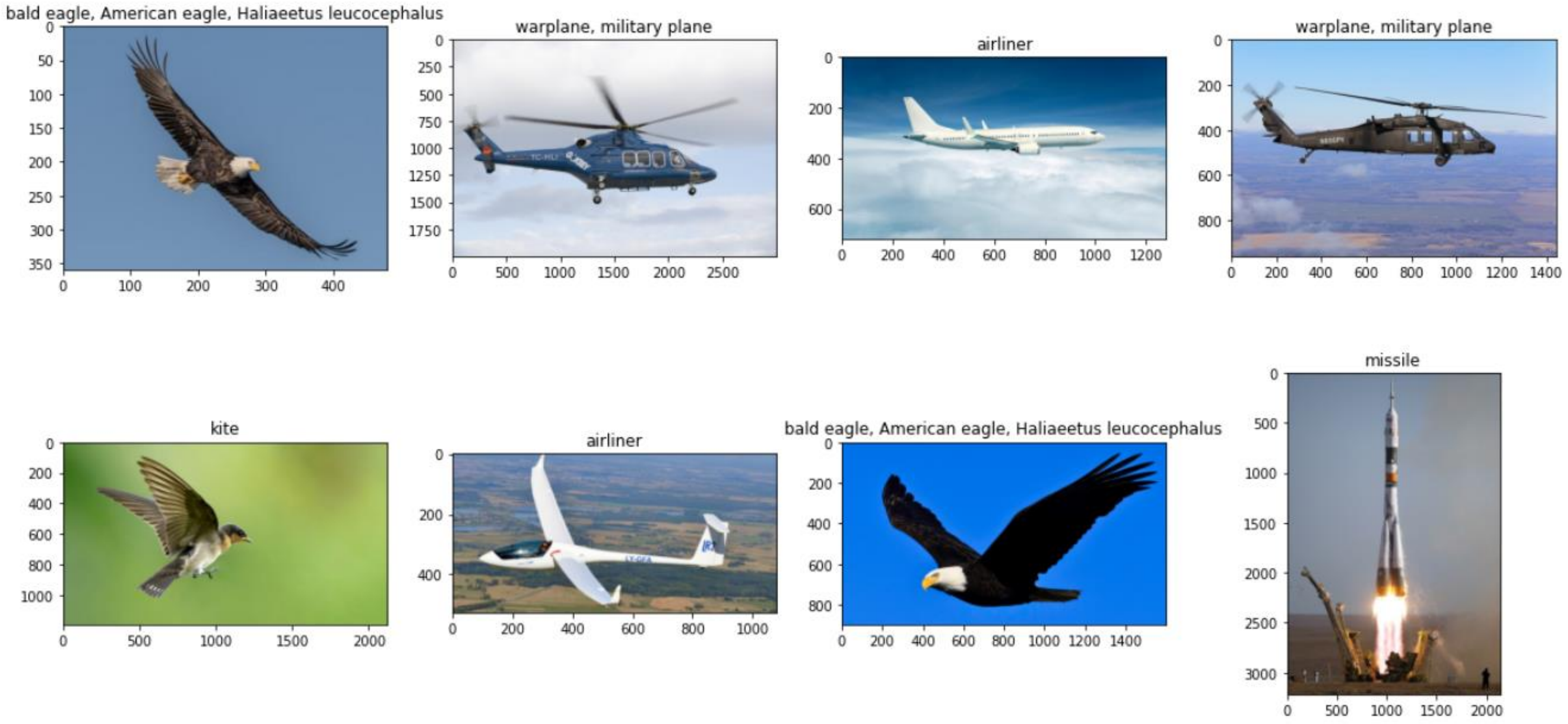
Now you will use other pre-trained model architectures including `vgg16` and `resnet18` for classifying your images. Simply execute the two code cells below for visualizing the images with predicted labels. Compare the predictions of different architectures. Do the predictions by a more complex model always make more sense?

Problem 4 (f) Solution

```
[7] ## Problem 4 (f) Vgg16

    vgg16 = torchvision.models.vgg16(pretrained=True)

    predictions_vgg16 = predict(vgg16, my_dataloader) # numpy array of predicted class labels
    label_list_vgg16 = [label_map[pred] for pred in predictions_vgg16]  # list of class names
    print("------------ visualize the images with predicted class names by VGG16  ------------")
    vis_img_label(image_list, label_list_vgg16)
```

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.cache/torch/hub/checkpoints/vgg16-397923af.pth
100% 528M/528M [00:02<00:00, 246MB/s]
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:26: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the call to include
------------ visualize the images with predicted class names by VGG16  ------------
```

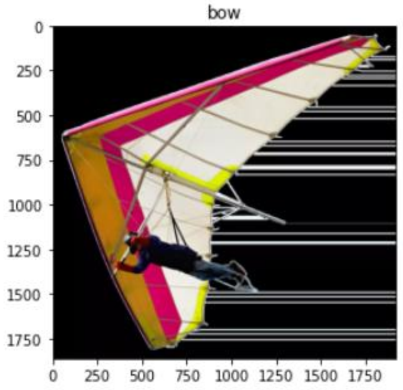


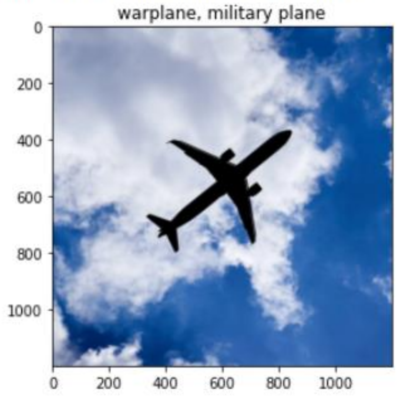

bald eagle, American eagle, Haliaeetus leucocephalus
warplane, military plane
airliner
warplane, military plane
kite
airliner
bald eagle, American eagle, Haliaeetus leucocephalus
missile

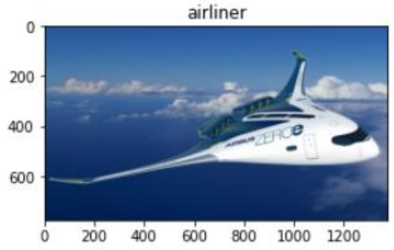
```
[8]  ## Problem 4 (f) Resnet18
```

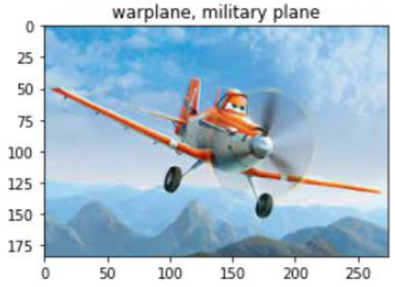
```
resnet18 = torchvision.models.resnet18(pretrained=True)

predictions_resnet18 = predict(resnet18, my_dataloader) # numpy array of predicted class labels
label_list_resnet18 = [label_map[pred] for pred in predictions_resnet18]  # list of class names
print("------------- visualize the images with predicted class names by ResNet18  -------------")
vis_img_label(image_list, label_list_resnet18)
```

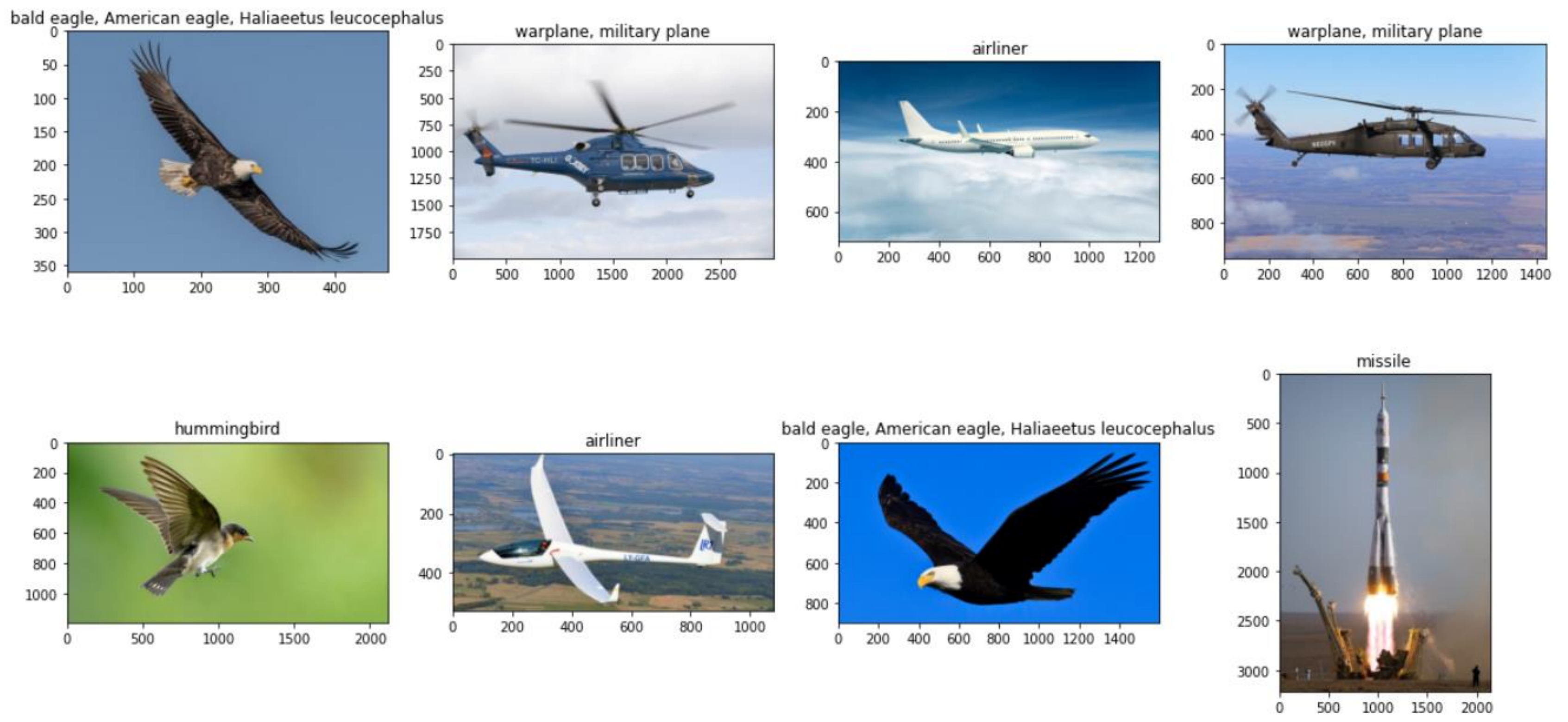
```
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
100% 44.7M/44.7M [00:01<00:00, 161MB/s]
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:26: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the call to include
------------- visualize the images with predicted class names by ResNet18  -------------
```










bald eagle, American eagle, Haliaeetus leucocephalus
warplane, military plane
airliner
warplane, military plane
hummingbird
airliner
bald eagle, American eagle, Haliaeetus leucocephalus
missile

Problem 4 (g) Interpret Result

Observe the predictions of different architectures and provide your answers for the following questions:

Observe the predictions of different architectures and provide your answers for the following questions:

- List some predictions that are closely related to your labels or what are presented in the corresponding images. For example, AlexNet predicts an image of a tiger as 'tiger' , or VGG16 predicts an image of an giraffe as 'gazelle'.
- Some of the predictions are not related but have reasonable explanations. For example, Resnet18 predicts a giraffe as 'honeycomb'. Such prediction is made because the fur pattern of a giraffe somehow looks alike the structure of honeycomb.
- The three architectures might have different predictions on the same images. For those cases, does a more complex model always generate predictions that are more related?

Problem 4 (g) Solution

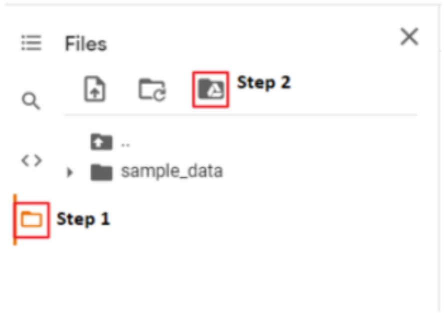
1. Predictions that are closely related to your labels: AlexNet and ResNet18 gets military plans, humming birds, missles, and Eagles correct; VGG16 gets airliners, planes, missles, and eagles correct.
2. ResNet18 label a glider as bow since the glider in the picture has a "<" shape; AlexNet label a comic plane as ski since it is surrounded by snowy mountains; VGG16 labels humming birds as kite since it also got similar shape.
3. No, more complex model does NOT generate more accurate predictions.

Problem 5-8 Setup Instructions

This is the first assignment you will encounter in the course that requires to work with a real world, image-based dataset. Since any variables and data created inside a colab session gets deleted from memory when you step out of that session, we are going to work with personal drive storage instead, especially since the dataset for this assignment is much larger in size than anything you will have seen thus far, and it can potentially use a significant portion of your internet data and time to download this from scratch every time you start a new session. Please pay careful attention to the instructions that follow, since they will greatly increase your convenience for this assignment.

1. Connecting to Google Drive

Ensure that you have atleast 5-6 GB of free storage space in your personal Gdrive. If not, it might be a good idea to set up a new google account and work on the assignment there. Once you have this sorted out, connect your notebook session to your Gdrive space by clicking on the files tab on the left of the screen, followed by the "Mount Drive" button, as shown in the image below:



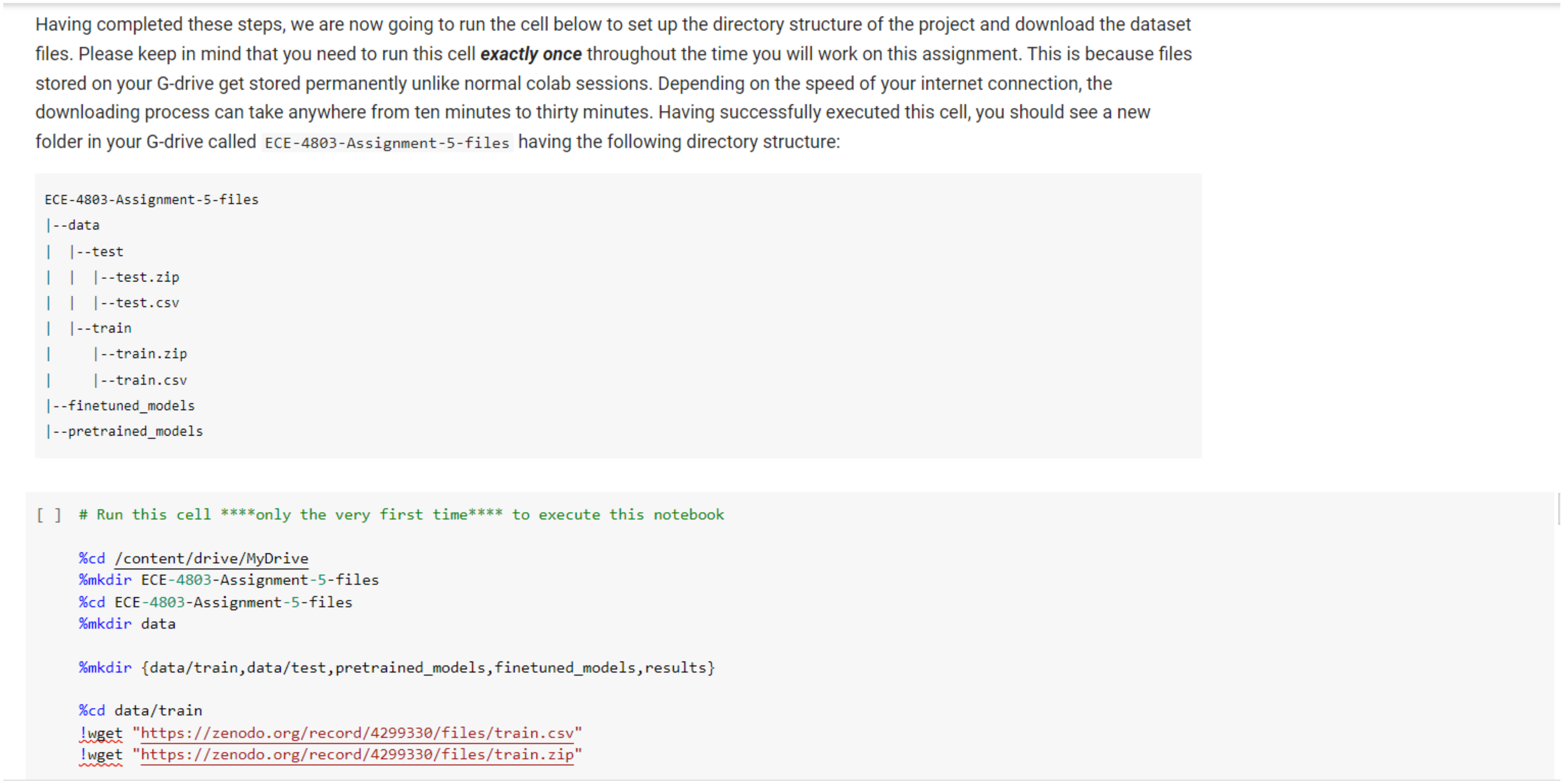
```
[2] from google.colab import drive
    drive.mount('/content/drive')
```

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

2. Changing Runtime session to "GPU type"

Since we are going to be training and running deep neural networks for this assignment, using GPUs would greatly speed up that process. On the tab above, click on the Runtime button followed by "Change Runtime type". In the dialog box that opens up, change the hardware accelerator type to GPU.

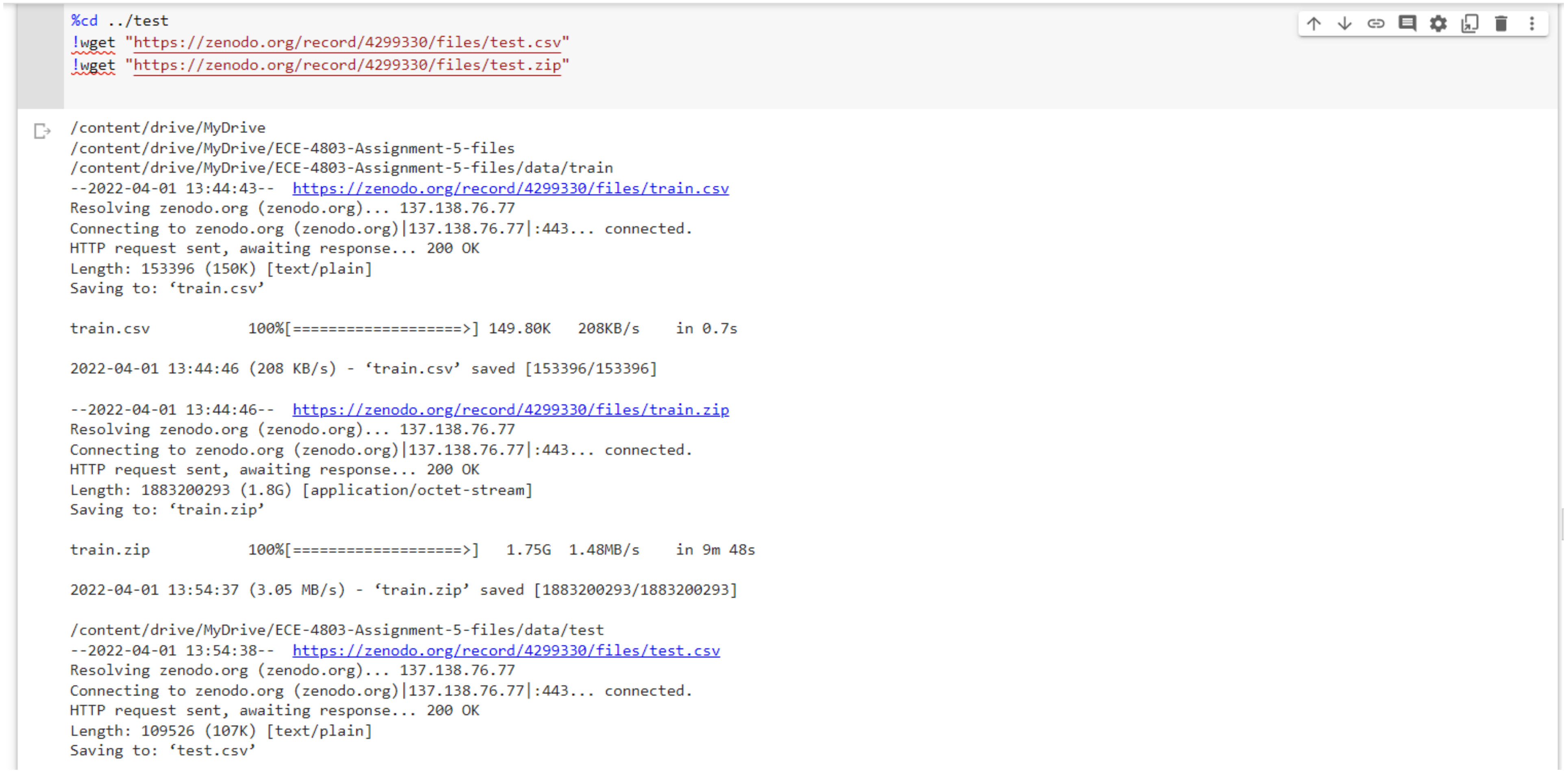
3. Setting up the directory structure and downloading the data



Having completed these steps, we are now going to run the cell below to set up the directory structure of the project and download the dataset files. Please keep in mind that you need to run this cell ***exactly once*** throughout the time you will work on this assignment. This is because files stored on your G-drive get stored permanently unlike normal colab sessions. Depending on the speed of your internet connection, the downloading process can take anywhere from ten minutes to thirty minutes. Having successfully executed this cell, you should see a new folder in your G-drive called `ECE-4803-Assignment-5-files` having the following directory structure:

```
ECE-4803-Assignment-5-files
|--data
|  |--test
|  |  |--test.zip
|  |  |--test.csv
|  |--train
|     |--train.zip
|     |--train.csv
|--finetuned_models
|--pretrained_models
```

```
[ ]  # Run this cell ****only the very first time**** to execute this notebook

     %cd /content/drive/MyDrive
     %mkdir ECE-4803-Assignment-5-files
     %cd ECE-4803-Assignment-5-files
     %mkdir data

     %mkdir {data/train,data/test,pretrained_models,finetuned_models,results}

     %cd data/train
     !wget "https://zenodo.org/record/4299330/files/train.csv"
     !wget "https://zenodo.org/record/4299330/files/train.zip"
```



```
%cd ../test
!wget "https://zenodo.org/record/4299330/files/test.csv"
!wget "https://zenodo.org/record/4299330/files/test.zip"
```

```
/content/drive/MyDrive
/content/drive/MyDrive/ECE-4803-Assignment-5-files
/content/drive/MyDrive/ECE-4803-Assignment-5-files/data/train
--2022-04-01 13:44:43--  https://zenodo.org/record/4299330/files/train.csv
Resolving zenodo.org (zenodo.org)... 137.138.76.77
Connecting to zenodo.org (zenodo.org)|137.138.76.77|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 153396 (150K) [text/plain]
Saving to: ‘train.csv’

train.csv           100%[===================>] 149.80K   208KB/s    in 0.7s

2022-04-01 13:44:46 (208 KB/s) - ‘train.csv’ saved [153396/153396]

--2022-04-01 13:44:46--  https://zenodo.org/record/4299330/files/train.zip
Resolving zenodo.org (zenodo.org)... 137.138.76.77
Connecting to zenodo.org (zenodo.org)|137.138.76.77|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1883200293 (1.8G) [application/octet-stream]
Saving to: ‘train.zip’

train.zip           100%[===================>]   1.75G  1.48MB/s    in 9m 48s

2022-04-01 13:54:37 (3.05 MB/s) - ‘train.zip’ saved [1883200293/1883200293]

/content/drive/MyDrive/ECE-4803-Assignment-5-files/data/test
--2022-04-01 13:54:38--  https://zenodo.org/record/4299330/files/test.csv
Resolving zenodo.org (zenodo.org)... 137.138.76.77
Connecting to zenodo.org (zenodo.org)|137.138.76.77|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 109526 (107K) [text/plain]
Saving to: ‘test.csv’
```

```
test.csv            100%[===================>] 106.96K   232KB/s    in 0.5s

2022-04-01 13:54:41 (232 KB/s) - ‘test.csv’ saved [109526/109526]

--2022-04-01 13:54:41--  https://zenodo.org/record/4299330/files/test.zip
Resolving zenodo.org (zenodo.org)... 137.138.76.77
Connecting to zenodo.org (zenodo.org)|137.138.76.77|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1526714682 (1.4G) [application/octet-stream]
Saving to: ‘test.zip’

test.zip            100%[===================>]   1.42G  4.71MB/s    in 11m 14s

2022-04-01 14:05:57 (2.16 MB/s) - ‘test.zip’ saved [1526714682/1526714682]
```

Problem 5: Inspecting the CURE-OR Dataset

The dataset we are working with for the purposes of the next set of questions in this assignment is called "CURE-OR", standing for Challenging Unreal and Real Environments for Object Recognition. The dataset was originally created to test the robustness of various off-the-shelf object recognition algorithms to various types and levels of distortion found in real world scenarios. The complete details on this may be found in the paper [link], the citation for which is as follows:

```
D. Temel, J. Lee, and G. AlRegib, "CURE-OR: Challenging unreal and real environments for object recognition," 2018 17th IEEE International Confere
```

What we are actually using here is a smaller subset of the orginal dataset. For any machine learning task, familiarizing oneself with the data is as important as setting up and training the machine learning model itself. The code cells below give a glimpse into the `train.csv` file you downloaded earlier, as well as sample randomly and display five images for a given class from the `train.zip` file. As you may see from the output of the `train.csv` file, each image in the `train.zip` has an image ID, a class, a background, a perspective, challenge type, and challenge level. Execute the fourth code cell displaying randomly sampled images from `train.zip` a number of times, and based off your observations

(as well as from what you can gather from reading the paper cited above and analyzing the csv file), answer the following questions:

(a) How many classes in total are contained within the `train.zip` file? What are the names of each of those classes?

(b) How many different backgrounds do you observe for the images in the `train.zip` file? What are those backgrounds?

(c) How many different camera perspectives (angles) have the pictures of the various objects in `train.zip` been taken?

(d) Describe some of the different kinds of noise types you can observe for the objects in `train.zip`. Please be concise and precise in your answers.

✓ [3]
```
# run this cell every time you execute the notebook afresh after the first time
# setting up and define the data directories

import os

root = '/content/drive/MyDrive/ECE-4803-Assignment-5-files'
train_directory = '/content/drive/My Drive/ECE-4803-Assignment-5-files/data/train'
test_directory = '/content/drive/My Drive/ECE-4803-Assignment-5-files/data/test'
os.chdir(root)
```

✓ 0s [4]
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

✓ 0s [14]
```
# inspect train.csv

os.chdir(root+'/data/train')

import pandas as pd
```

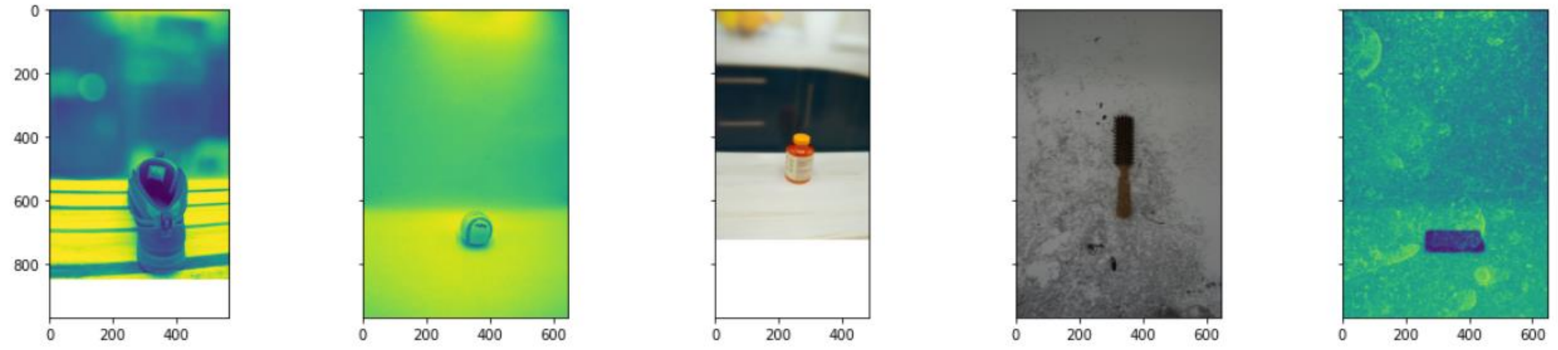
```
df = pd.read_csv('train.csv')
df.head()
```

	imageID	class	background	perspective	challengeType	challengeLevel
0	0	1	1	1	1	0
1	1	2	1	1	1	0
2	2	3	1	1	1	0
3	3	4	1	1	1	0
4	4	5	1	1	1	0

✓ [15]
6s

```
# inspect training images in train.zip

import random
from zipfile import ZipFile
import shutil

import matplotlib.pyplot as plt
import matplotlib.image as mpimg


zipobj = ZipFile('train.zip')
file_IDs = zipobj.namelist()[1:]  # all fileIds in the folder

random.shuffle(file_IDs)
filenames = file_IDs[:5]

with ZipFile('train.zip','r') as zipObject:
  for filename in filenames:
     zipObject.extract(filename)
```

```
random.shuffle(file_IDs)
filenames = file_IDs[:5]

with ZipFile('train.zip','r') as zipObject:
  for filename in filenames:
     zipObject.extract(filename)


fig, ax = plt.subplots(1,5, sharey=True, figsize=(20,4))

for filename,axis in zip(filenames, ax.flatten()):
  img=mpimg.imread(filename)
  axis.imshow(img)

plt.show()

shutil.rmtree('train')
```



Problem 5 (a) Solution

10 Class: Canon camera, Training marker cone, Baseball, Pan, Toy, LG Cell phone, Hair brush, DYMO Label maker, Calcium bottle, Shoes

Problem 5 (b) Solution

3 backgrounds: White, Texture 1 - living room, Texture 2 - kitchen

Problem 5 (c) Solution

5 perspectives: Front (0°), Left side (90°), Back (180°), Right side (270°), Top

Problem 5 (d) Solution

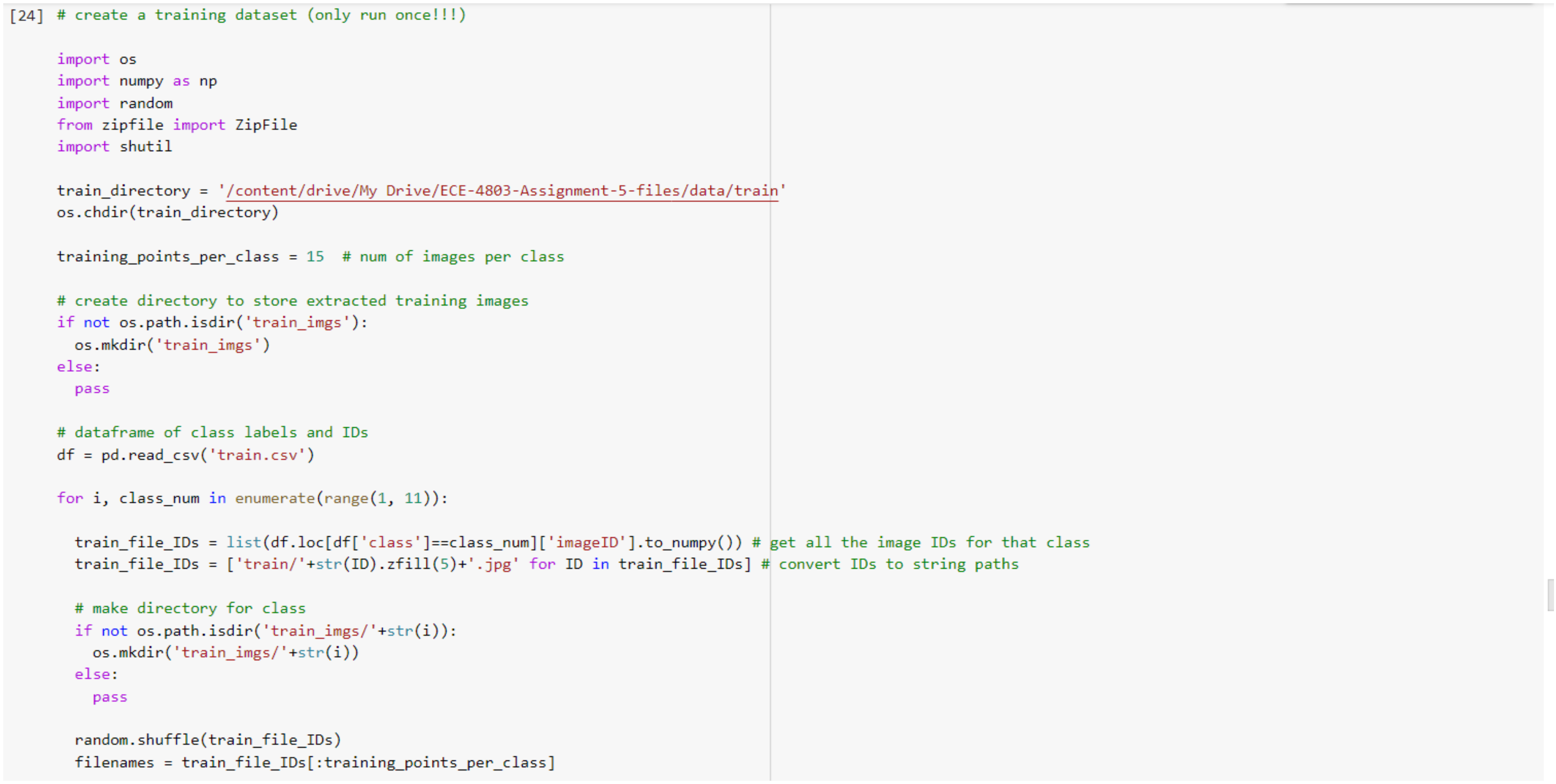
Noises such as: Grayscale, resizing images, blurring images, under/over-exposure, and the combinations of them

Creating the Train and Test Splits

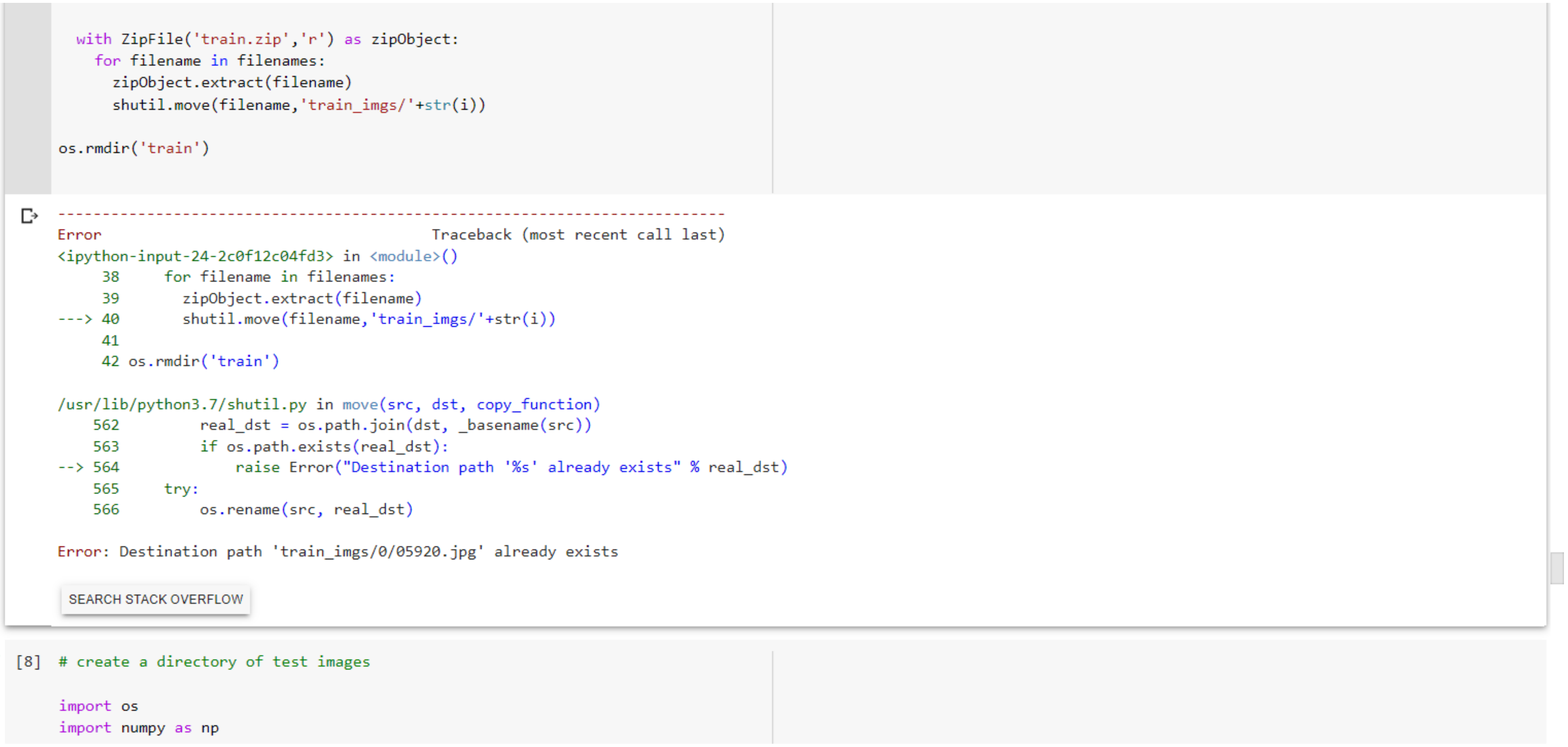
The two code cells below extract randomly a given number of images for each object class from the `train.zip` and `test.zip` files to create two additional file directories in the folder structure we set up earlier. After executing both of them, you should now observe the folder tree to have a structure similar to that shown below:

```
ECE-4803-Assignment-5-files
|--data
|  |--test
|  |  |--test_imgs
|  |  |  |--0
|  |  |  |--1
|  |  |  :
|  |  |  |--9
|  |  |--test.zip
|  |  |--test.csv
|  |--train
|  |  |--train_imgs
|  |  |  |--0
|  |  |  |--1
|  |  |  :
|  |  |  |--9
|     |--train.zip
|     |--train.csv
|--finetuned_models
|--pretrained_models
```

Each of the numbers 0,1,... denote a folder containing images extracted for that given class from the `train.zip` and `test.zip` files, respectively. After executing the cells, manually browse through your G-drive to see for yourself the changes that happened to the original file tree. These are the training and testing splits we are going to use for training and evaluating our models for this assignment.

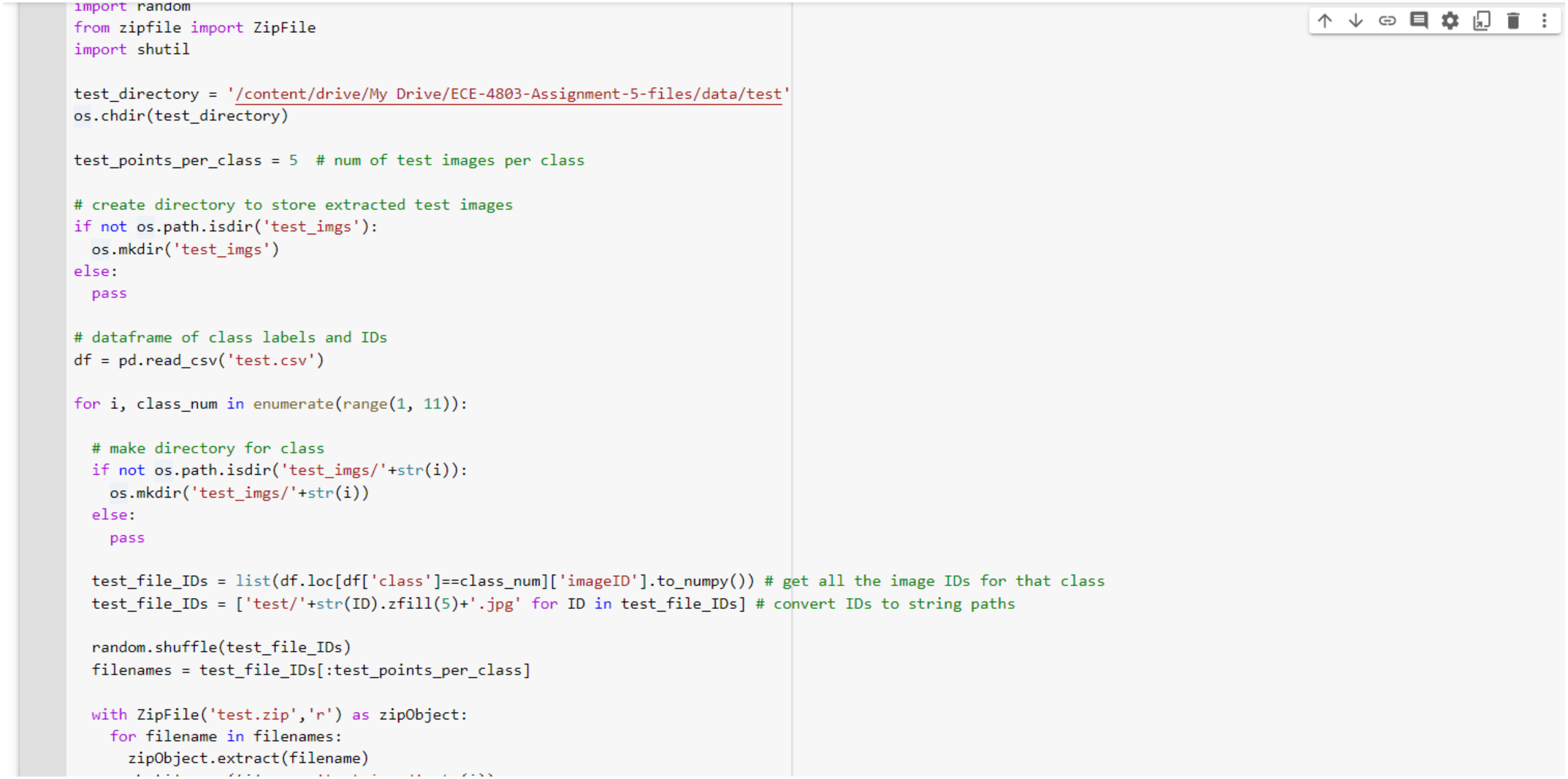


```
[24] # create a training dataset (only run once!!!)

     import os
     import numpy as np
     import random
     from zipfile import ZipFile
     import shutil

     train_directory = '/content/drive/My Drive/ECE-4803-Assignment-5-files/data/train'
     os.chdir(train_directory)

     training_points_per_class = 15  # num of images per class

     # create directory to store extracted training images
     if not os.path.isdir('train_imgs'):
       os.mkdir('train_imgs')
     else:
       pass

     # dataframe of class labels and IDs
     df = pd.read_csv('train.csv')

     for i, class_num in enumerate(range(1, 11)):

       train_file_IDs = list(df.loc[df['class']==class_num]['imageID'].to_numpy()) # get all the image IDs for that class
       train_file_IDs = ['train/'+str(ID).zfill(5)+'.jpg' for ID in train_file_IDs] # convert IDs to string paths

       # make directory for class
       if not os.path.isdir('train_imgs/'+str(i)):
         os.mkdir('train_imgs/'+str(i))
       else:
         pass

       random.shuffle(train_file_IDs)
       filenames = train_file_IDs[:training_points_per_class]
```


```
  with ZipFile('train.zip','r') as zipObject:
    for filename in filenames:
      zipObject.extract(filename)
      shutil.move(filename,'train_imgs/'+str(i))

os.rmdir('train')
```

```
---------------------------------------------------------------------------
Error                                     Traceback (most recent call last)
<ipython-input-24-2c0f12c04fd3> in <module>()
     38     for filename in filenames:
     39       zipObject.extract(filename)
---> 40       shutil.move(filename,'train_imgs/'+str(i))
     41 
     42 os.rmdir('train')

/usr/lib/python3.7/shutil.py in move(src, dst, copy_function)
    562         real_dst = os.path.join(dst, _basename(src))
    563         if os.path.exists(real_dst):
--> 564             raise Error("Destination path '%s' already exists" % real_dst)
    565     try:
    566         os.rename(src, real_dst)

Error: Destination path 'train_imgs/0/05920.jpg' already exists
```

```
[8] # create a directory of test images

    import os
    import numpy as np
```



```python
import random
from zipfile import ZipFile
import shutil

test_directory = '/content/drive/My Drive/ECE-4803-Assignment-5-files/data/test'
os.chdir(test_directory)

test_points_per_class = 5  # num of test images per class

# create directory to store extracted test images
if not os.path.isdir('test_imgs'):
  os.mkdir('test_imgs')
else:
  pass

# dataframe of class labels and IDs
df = pd.read_csv('test.csv')

for i, class_num in enumerate(range(1, 11)):

  # make directory for class
  if not os.path.isdir('test_imgs/'+str(i)):
    os.mkdir('test_imgs/'+str(i))
  else:
    pass

  test_file_IDs = list(df.loc[df['class']==class_num]['imageID'].to_numpy()) # get all the image IDs for that class
  test_file_IDs = ['test/'+str(ID).zfill(5)+'.jpg' for ID in test_file_IDs] # convert IDs to string paths

  random.shuffle(test_file_IDs)
  filenames = test_file_IDs[:test_points_per_class]

  with ZipFile('test.zip','r') as zipObject:
    for filename in filenames:
      zipObject.extract(filename)
```

```
        shutil.move(filename,'test_imgs/'+str(i))

os.rmdir('test')
```

Problem 6: Setting up the Dataloaders for CURE-OR

Here, we set up the dataloaders to load and present to the neural network images in the train and test splits respectively. Fill into the code cell below at the places indicated to instantiate the dataloader objects.

Remember that our images are `.jpg` files, and they have to be preprocessed and conditioned appropriately before they can be processed by a neural network. To be more specific, the following four operations have to be carried out (in the same order) before the image files can be presented to the neural network model.

1. Resizing the images to 256×256 pixels.

2. Center cropping the images to to 224×224 pixels.

3. Converting the image to a torch tensor.

4. Normalizing the image channels (Red, Green, and blue) to have 0 means and 1 standard deviation, respectively. For the ImageNet dataset, the means for the three channels are $0.485, 0.456$, and 0.406, respectively while the standard deviations are $0.229, 0.224$, and 0.225, respectively. We use this statistics to normalize our CURE-OR dataset as well.

Note: `Pytorch` provides handy tools for all of the above conversions/processes in the form of `Resize`, `CenterCrop`, `ToTensor`, and `Normalize` classes, respectively within the `torchvision.transforms` module. Moreover, one may construct a single pipeline involving the serial application of several processes using the very useful `transforms.Compose` class. Use these classes to construct a single pipeline called `preprocess` in the code indicated below applying the above transformations in the order mentioned.

Afterwards, use the `ImageFolder` class in the `torchvision.datasets` module to set up a dataset object for both training and testing. The two options you need to specify while instantiating these obejcts is the path of the folder containing the class image folders and the transform pipeline object (that you constructed above).

Problem 6 Solution

```
[25] # create a dataset and dataloader

     import torch
     from torchvision.datasets import ImageFolder
     from torchvision import transforms

     preprocess = transforms.Compose([transforms.Resize((256, 256)),
                                      transforms.CenterCrop(224),
                                      transforms.ToTensor(),
                                      transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                                           std=[0.229, 0.224, 0.225])])
     ##TODO

     # set up train loader
     train_dataset = ImageFolder(root="/content/drive/My Drive/ECE-4803-Assignment-5-files/data/train" , transform=preprocess) ##TODO
     trainloader = torch.utils.data.DataLoader(train_dataset, batch_size=1, shuffle=True)

     # set up test loader
     test_dataset = ImageFolder(root='/content/drive/My Drive/ECE-4803-Assignment-5-files/data/test' , transform=preprocess) ##TODO
     testloader = torch.utils.data.DataLoader(test_dataset, batch_size=1, shuffle=True)
```

Problem 7: Setting up and Testing a Pretrained Neural Network Model on CURE-OR

Having constructed the dataloaders, we now move onto the neural network model itself. For this question, we are going to use the AlexNet architecture, which you may remember as being the winner of the ImageNet 2012 competition. The first code cell below downloads the pretrained weights of the AlexNet on the ImageNet dataset into the `\content\drive\MyDrive\ECE-4803-Assignment-5-files\pretrained_models` directory. This code cell is to run ***only once*** throughout the time you work on this assignment.

We are going to test and see how well the pretrained Alexnet would perform on a dataset that it has not been trained on before. Remember that a neural network only predicts numbers corresponding to classes. To extract the actual classnames, we need a dictionary mapping each of the output integers to a class name. For the ImageNet dataset, we provide a python dictionary containing a mapping from the integers $0 - 999$ to one of the thousand classes the network was trained on. The dictionary may be downloaded by running the following snippet of code:

```
import os


os.chdir(root)
!gdown --id 1bDrtvgX-ztIh7A46FQNvROS7bEVuYogn
```

This downloads a file called `label_dict.pth` to the `ECE-4803-Assignment-5-files` folder in your Gdrive. To answer this question, fill in the second and third code cells below as instructed.

(a) Import the `label_dict` dictionary into the variable called `label_map` using the `torch.load` variable.

(b) Instantiate an object of `alexnet` class from the `torchvision.models` module.

(c) Load the pretrained weights you downloaded above into the model object you just instantiated using the model object's class function, `load_state_dict`, and the `torch.load` function.

(d) Using the trainloader object you defined in Problem 5, load training images and present to the pretrained alexnet architecture. The output of the final layer is a 1000 element vector. Use the index position for the highest activation value to extract the corresponding class label from the `label_map` variable. This is then set as the title of the image. The places in the code you need to fill out have been indicated in the third code cell below.

(e) Execute the third code cell multiple times. Based off your observations, do the network predictions make sense on all or some of the objects? Name some of these objects? For the objects the network fails at some or most of the times, do the incorrect network predictions on these have any significance in relation to the ground-truth class of those images?

(f) Explain some of the reasons why the network may be predicting very different classes to the ground-truths for some images.

```
[ ] # Only download models once!

    os.environ['TORCH_HOME'] = "/content/drive/MyDrive/ECE-4803-Assignment-5-files/pretrained_models/"

    from torchvision import models

    models.resnet18(pretrained=True)
    models.alexnet(pretrained=True)
    models.vgg16(pretrained=True)
```

```
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /content/drive/MyDrive/ECE-4803-Assignment-5-files/pretrained_models/hub/checkpoints/resı
100% ████████████████████ 44.7M/44.7M [00:00<00:00, 94.4MB/s]
Downloading: "https://download.pytorch.org/models/alexnet-owt-7be5be79.pth" to /content/drive/MyDrive/ECE-4803-Assignment-5-files/pretrained_models/hub/checkpoints/ɩ
100% ████████████████████ 233M/233M [00:05<00:00, 41.3MB/s]
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /content/drive/MyDrive/ECE-4803-Assignment-5-files/pretrained_models/hub/checkpoints/vgg16-
100% ████████████████████ 528M/528M [00:23<00:00, 49.3MB/s]
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

```
[ ] # Only download dictionary once!
    import os
```

```
os.chdir(root)
!gdown --id 1bDrtvgX-ztIh7A46FQNvROS7bEVuYogn
```

```
Downloading...
From: https://drive.google.com/uc?id=1bDrtvgX-ztIh7A46FQNvROS7bEVuYogn
To: /content/drive/MyDrive/ECE-4803-Assignment-5-files/label_dict.pth
100% 33.1k/33.1k [00:00<00:00, 45.8MB/s]
```

Problem 7 (a)-(c) Solution

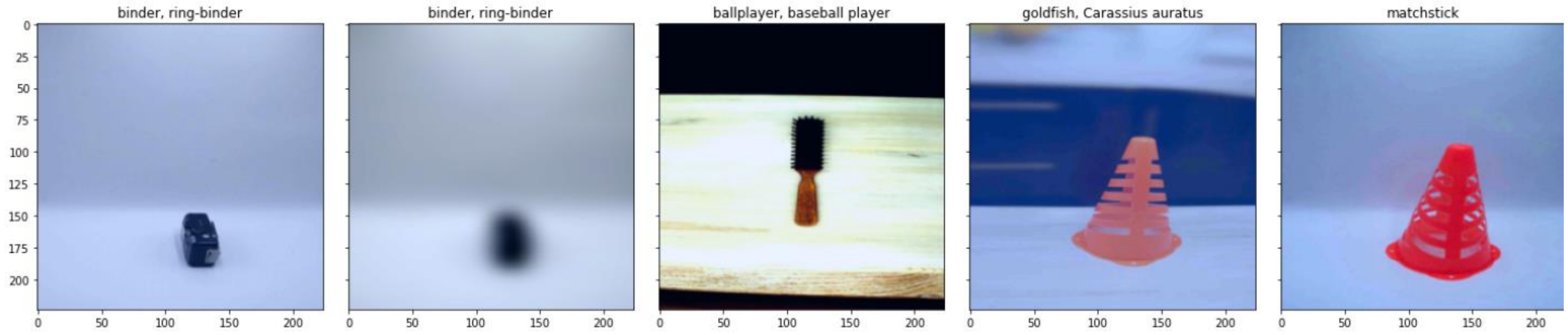
```
[ ] ## Problem 7 (a)-(c)
    # load models and label dictionary

    import torch
    from torchvision.models import alexnet, resnet18, vgg16

    # load dictionary for alexnet labels

    label_map = torch.load("/content/drive/MyDrive/ECE-4803-Assignment-5-files/label_dict.pth")##TODO

    # instantiate model object and load pretrained model weights
    model =  alexnet(pretrained=True) ##TODO
    model.load_state_dict(torch.load("/content/drive/MyDrive/ECE-4803-Assignment-5-files/pretrained_models/hub/checkpoints/alexnet-owt-7be5be79.pth"))##TODO
```

```
<All keys matched successfully>
```

Problem 7 (d) (e) Solution

Problem 7 (d) (e) Solution

```
## Problem 7 (d)(e)
# test pretrained model predictions on the train set
# import torch.nn.funtional as F


num_images = 5
model = model.to('cuda')

fig, ax = plt.subplots(1, num_images, sharey=True, figsize=(20,4))

with torch.no_grad():
  for i in range(num_images):

    img, _ = next(iter(trainloader))
    img = img.to('cuda')
    softmax = torch.nn.functional.log_softmax(img, dim = 1)
    # print(img.size)
    out = model(softmax) ##TODO
    pred_label = label_map.get(out.argmax(dim = 1).item())##TODO
    img = img.detach().cpu().numpy().squeeze().transpose(1,2,0)
    img = (img - img.min()) / (img.max() - img.min())

    ax[i].imshow(img)
    ax[i].set_title(pred_label)

plt.tight_layout()
plt.show()
plt.show()
```



Problem 7 (e) Solution

The network predictions make sense on some of the objects like brushes, and shoes. Yes, some of them have the same shape or color as the label.

Problem 7 (f) Solution

Since the number of images that we are feeding the model is not enough. Also, some of the picture has a very low resolution and even I could hardly identify them.

Problem 8: Training the Network from scratch on CURE-OR

As you may already have seen, the network predictions while making some sense on some images, appear to be completely off on others. We are going to attemp to fix this problem by retraining the alexnet network from scratch on our training split. Fill in the code below at the places indicated to train a randomly initialized alexnet network architecture on the training split we created earlier using the trainloader object instantiated in Problem 6. Answer the question by following the instructions below.

(a) Randomly intialize an alexnet architecture

(b) Since the original alexnet has been trained on 1000 classes, but CURE-OR only contains 10 classes, change the last layer to have 10 neurons instead of 1000.

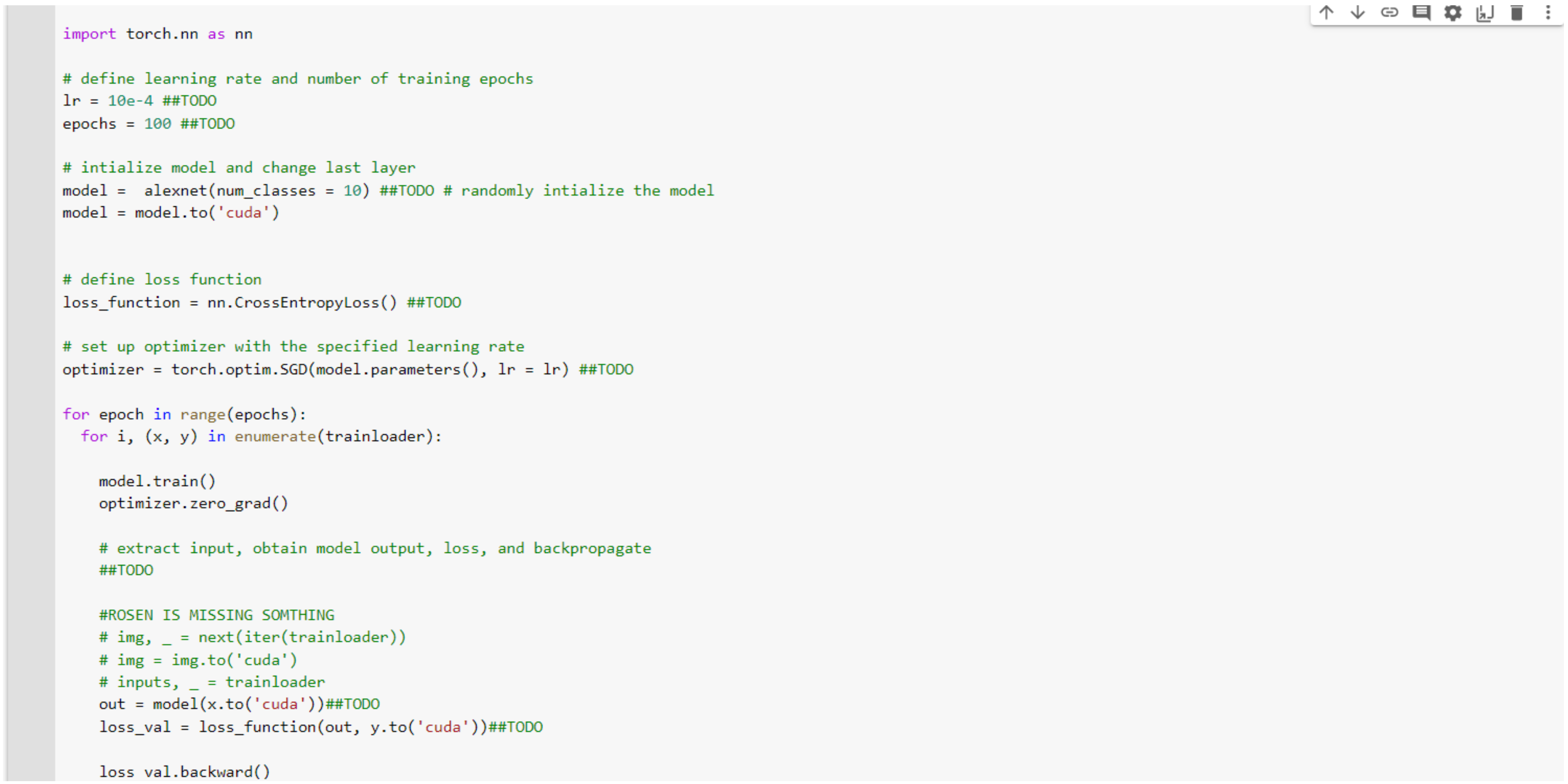
(c) Using the Adam optimizer, with a learning rate of 0.001, train the modified alexnet architecture for 100 epochs on the training split we created earlier (using its `trainloader` object).

(d) It is now time to test the trained model. Before we do that, however, we need to construct a python dictionary to map the integer predictions of the network (from 0 to 9) to each of the 10 classes present in CURE-OR. Using your answer to Problem 4 (a), define a dictionary called `label_map` creating this relationship.

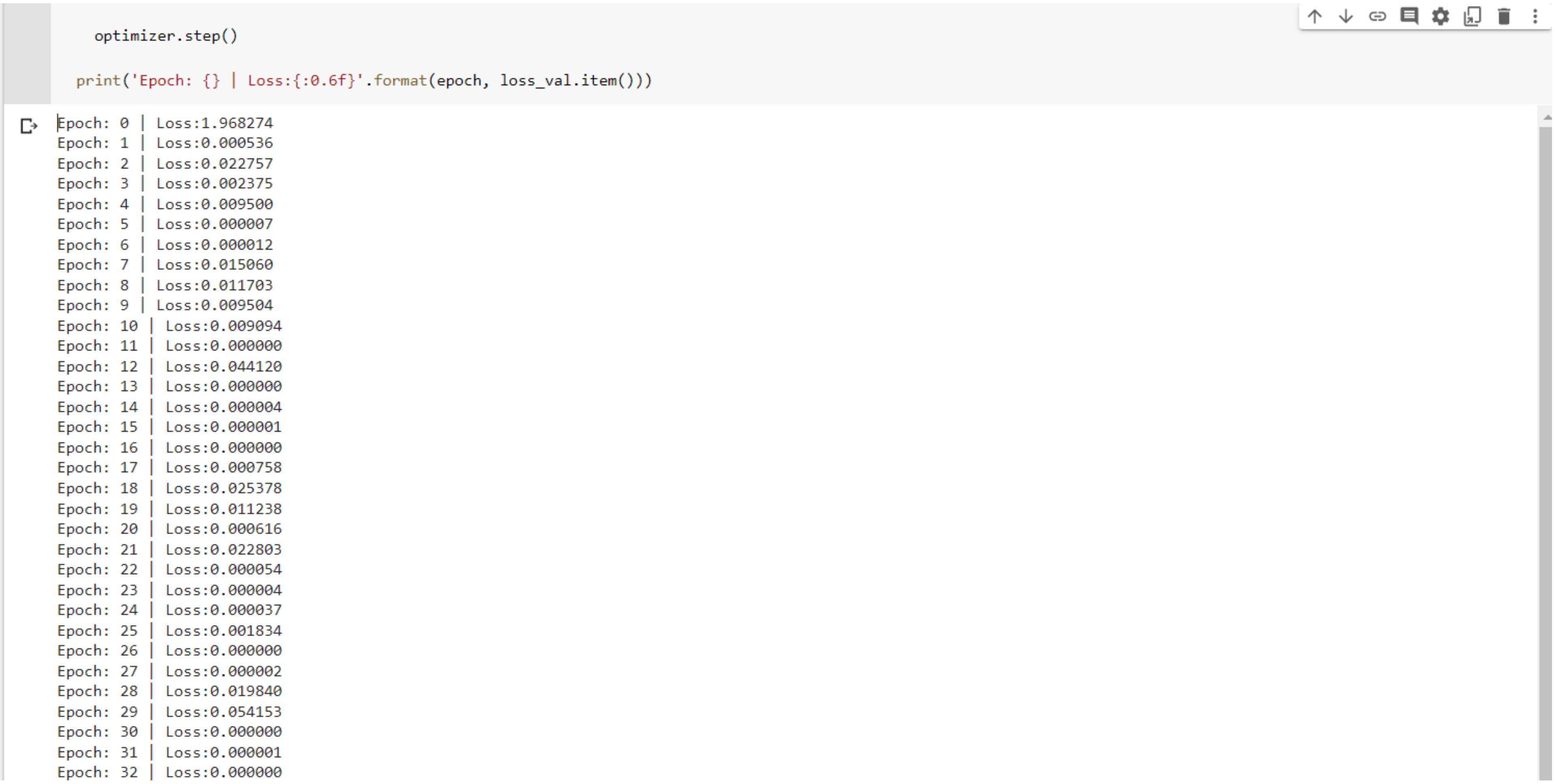
(e) Complete the inference code to randomly sample 5 images from the testing split and print the network predictions for them as their titles, as in Problem 6 (d) above. Execute the second code cell multiple times. What do you observe in regards to the network predictions? Which objects does the network consistently get right. Which ones does it fail more often at? If you observed a well performing network, why is that the case? If not, why do you think the network failed to perform better?

Problem 8 (a)-(c) Solution

```
[ ]  ## Problem 8 (a)-(c)
     # train model
```



```python
import torch.nn as nn

# define learning rate and number of training epochs
lr = 10e-4 ##TODO
epochs = 100 ##TODO

# intialize model and change last layer
model =  alexnet(num_classes = 10) ##TODO # randomly intialize the model
model = model.to('cuda')


# define loss function
loss_function = nn.CrossEntropyLoss() ##TODO

# set up optimizer with the specified learning rate
optimizer = torch.optim.SGD(model.parameters(), lr = lr) ##TODO

for epoch in range(epochs):
  for i, (x, y) in enumerate(trainloader):

    model.train()
    optimizer.zero_grad()

    # extract input, obtain model output, loss, and backpropagate
    ##TODO

    #ROSEN IS MISSING SOMTHING
    # img, _ = next(iter(trainloader))
    # img = img.to('cuda')
    # inputs, _ = trainloader
    out = model(x.to('cuda'))##TODO
    loss_val = loss_function(out, y.to('cuda'))##TODO

    loss val.backward()
```



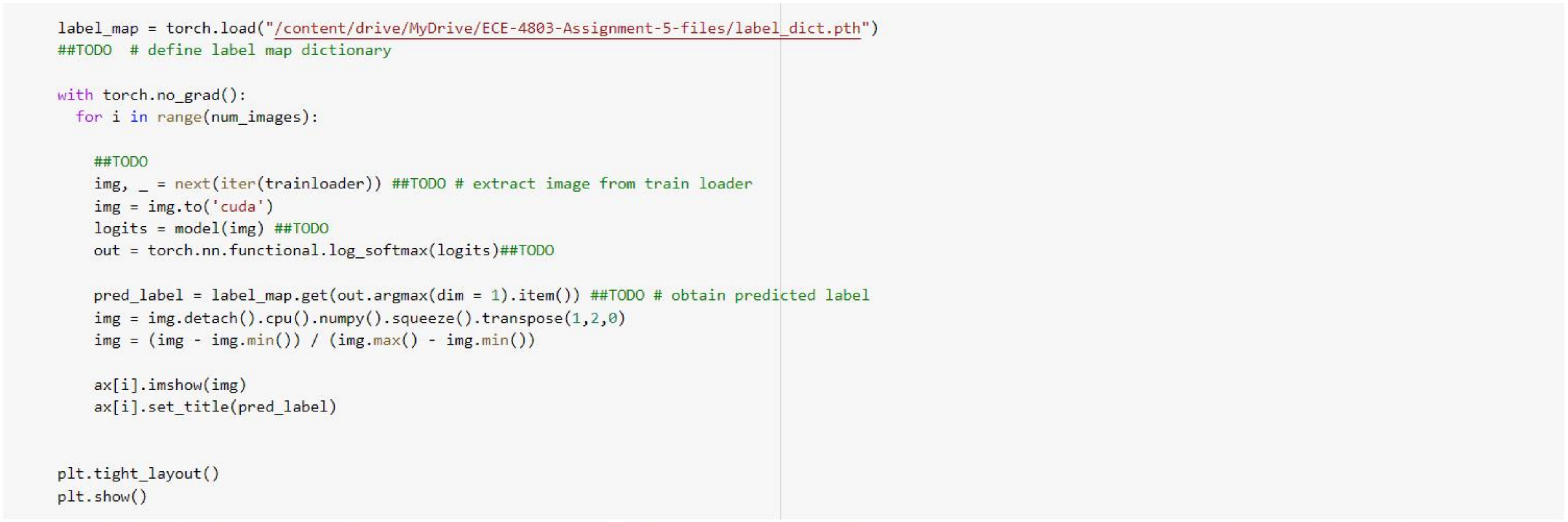
```
  optimizer.step()

print('Epoch: {} | Loss:{:0.6f}'.format(epoch, loss_val.item()))
```

```
Epoch: 0 | Loss:1.968274
Epoch: 1 | Loss:0.000536
Epoch: 2 | Loss:0.022757
Epoch: 3 | Loss:0.002375
Epoch: 4 | Loss:0.009500
Epoch: 5 | Loss:0.000007
Epoch: 6 | Loss:0.000012
Epoch: 7 | Loss:0.015060
Epoch: 8 | Loss:0.011703
Epoch: 9 | Loss:0.009504
Epoch: 10 | Loss:0.009094
Epoch: 11 | Loss:0.000000
Epoch: 12 | Loss:0.044120
Epoch: 13 | Loss:0.000000
Epoch: 14 | Loss:0.000004
Epoch: 15 | Loss:0.000001
Epoch: 16 | Loss:0.000000
Epoch: 17 | Loss:0.000758
Epoch: 18 | Loss:0.025378
Epoch: 19 | Loss:0.011238
Epoch: 20 | Loss:0.000616
Epoch: 21 | Loss:0.022803
Epoch: 22 | Loss:0.000054
Epoch: 23 | Loss:0.000004
Epoch: 24 | Loss:0.000037
Epoch: 25 | Loss:0.001834
Epoch: 26 | Loss:0.000000
Epoch: 27 | Loss:0.000002
Epoch: 28 | Loss:0.019840
Epoch: 29 | Loss:0.054153
Epoch: 30 | Loss:0.000000
Epoch: 31 | Loss:0.000001
Epoch: 32 | Loss:0.000000
```

```
Epoch: 33 | Loss:0.000025
Epoch: 34 | Loss:0.000000
Epoch: 35 | Loss:0.032437
Epoch: 36 | Loss:0.000000
Epoch: 37 | Loss:4.704785
Epoch: 38 | Loss:0.000001
Epoch: 39 | Loss:0.030162
Epoch: 40 | Loss:0.000000
Epoch: 41 | Loss:0.000051
Epoch: 42 | Loss:0.000098
Epoch: 43 | Loss:0.000000
Epoch: 44 | Loss:0.041639
Epoch: 45 | Loss:0.000000
Epoch: 46 | Loss:0.000000
Epoch: 47 | Loss:4.585609
Epoch: 48 | Loss:0.000000
Epoch: 49 | Loss:0.001583
Epoch: 50 | Loss:0.000000
Epoch: 51 | Loss:0.035281
Epoch: 52 | Loss:0.000017
Epoch: 53 | Loss:0.000012
Epoch: 54 | Loss:0.002553
Epoch: 55 | Loss:0.000025
Epoch: 56 | Loss:0.000066
Epoch: 57 | Loss:0.000000
```

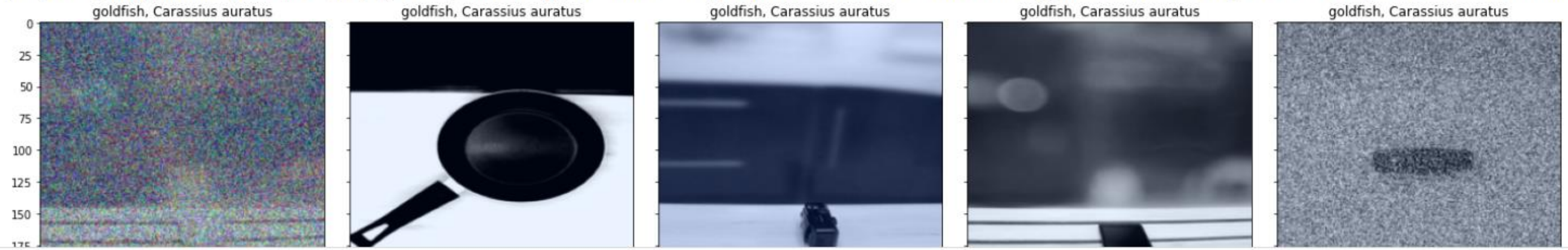
Problem 8 (d)(e) Solution

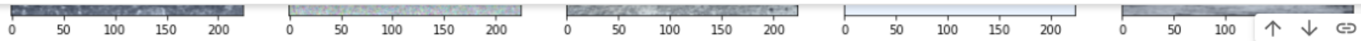
```
[ ] ## Problem 8 (d)(e)
    # test trained model predictions on the training set

    num_images = 5

    fig, ax = plt.subplots(1,num_images, sharey=True, figsize=(20,4))
```



```
label_map = torch.load("/content/drive/MyDrive/ECE-4803-Assignment-5-files/label_dict.pth")
##TODO  # define label map dictionary

with torch.no_grad():
  for i in range(num_images):

    ##TODO
    img, _ = next(iter(trainloader)) ##TODO # extract image from train loader
    img = img.to('cuda')
    logits = model(img) ##TODO
    out = torch.nn.functional.log_softmax(logits)##TODO

    pred_label = label_map.get(out.argmax(dim = 1).item()) ##TODO # obtain predicted label
    img = img.detach().cpu().numpy().squeeze().transpose(1,2,0)
    img = (img - img.min()) / (img.max() - img.min())

    ax[i].imshow(img)
    ax[i].set_title(pred_label)


plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the call to include
```





Problem 8 (e) Solution

The type of object with no noise or greyscale noise are constantly correct while the blurry ones constantly fail. With a well-performed network, I think it is because blurry object is has a ambiguous shape and shape is a critical factor for predicting stuff. If we can improve the network, maybe we can improve its ability on recognizing shapes or colors or some even more detailed features.

✓ 1s completed at 5:48 PM