

CS541: Applied Machine Learning, Spring 2025, Problem Set 4

Problem set 4 is due in Gradescope on **April 10, Thursday at 11:59pm**. All the questions are in this jupyter notebook file. There are four questions in this assignment, each of which could have multiple parts and consists of a mix of coding and short answer questions. This assignment is worth a total of **100 points (80 pts coding, and 20 pts short answer)**. There is a bonus question at the end which is worth an extra 10 pts but your maximum final score will be capped at 100 if you scored beyond 100. Note that each individual pset contributes the same amount to the final grade regardless of the number of points it is worth.

After completing these questions you will need to convert this notebook into a .py file named **ps4.py** and a pdf file named **ps4.pdf** in order to submit it (details below).

Submission instructions: please upload your completed solution files to Gradescope by the due date. **Make sure you have run all code cells and rendered all markdown/Latex without any errors.**

**THE SUBMISSION IS DUE BY April 10, Thursday at 11:59pm. **

There will be three separate submission links for the assignment:

1. Submit **ps3.py** to PS4-Code
2. Submit **ONLY your typed code** to PS4-Typed Code .
 - The .py file should contain **ONLY your typed code** (Do not include any other code apart from what you coded for the assignment).
 - The .py should not contain any written answers. Only the code you wrote.
 - If your typed code falls under a function definition that's predefined by us, **ONLY include your typed code** and nothing else.
 - For each cell block within colab/jupyter that you typed your code in, Add 2 new lines before pasting your typed code in the .py file.
 - Please name the .py file your actual name.
3. Submit a single .pdf report that contains your work for all written questions to PS4 . You can type your responses in LaTeX, or any other word processing software. You can also hand write them on a tablet, or scan in hand-written answers. If you hand-write, please make sure they are neat and legible. If you are scanning, make sure that the scans are legible. Lastly, convert your work into a PDF . You can use Jupyter Notebook to convert the formats:
 - Convert to PDF file: Go to File->Download as->PDF
 - Convert py file: Go to File->Download as->pyYou can take a look at an example [here](#)

(https://raw.githubusercontent.com/chaudatascience/cs599_fall2022/master/ps1/convert_py.gif)

Your written responses in the PDF report should be self-contained. It should include all the output you want us to look at. **You will not receive credit for any results you have obtained, but failed to include directly in the PDF report file. Please tag the responses in your PDF with the Gradescope questions outline as described in [Submitting an Assignment](https://youtu.be/u-pK4GzpId0) (<https://youtu.be/u-pK4GzpId0>).** Failure to follow these instructions will result in a loss of points.

Assignment Setup

You are strongly encouraged to use [Google Colab](https://colab.research.google.com/) (<https://colab.research.google.com/>) for this assignment.

If you would prefer to setup your code locally on your own machine, you will need [Jupyter Notebook](https://jupyter.org/install#jupyter-notebook) (<https://jupyter.org/install#jupyter-notebook>) or [JupyterLab](https://jupyter.org/install#jupyterlab) (<https://jupyter.org/install#jupyterlab>) installation. One way to set it up is to install “Anaconda” distribution, which has Python (you should install python version ≥ 3.9 as this notebook is tested with python 3.9), several libraries including the Jupyter Notebook that we will use in class. It is available for Windows, Linux, and Mac OS X [here](https://docs.conda.io/en/latest/miniconda.html) (<https://docs.conda.io/en/latest/miniconda.html>).

If you are not familiar with Jupyter Notebook, you can follow [this blog](https://realpython.com/jupyter-notebook-introduction/) (<https://realpython.com/jupyter-notebook-introduction/>) for an introduction. After developing your code using Jupyter, you are encouraged to test it on Google Colab to ensure it works in both settings.

You cannot use packages other than the ones already imported in this assignment.

Jupyter Tip 1: To run a cell, press `Shift+Enter` or click on "play" button above. To edit any code or text cell [double] click on its content. To change cell type, choose "Markdown" or "Code" in the drop-down menu above.

Jupyter Tip 2: Use shortcut "Shift + Tab" to show the documentation of a function in Jupyter Notebook/ Jupyterlab. Press Shift then double Tab (i.e., press Tab twice) to show the full documentation.

For example, type `sum(` then Shift + Tab to show the documentation for the function, as shown in this the picture below.

```
In [1]: 1 ## import some libraries
2 import sklearn
3 from sklearn import datasets
4 import numpy as np
5 from typing import Tuple, List, Dict
6 import pandas as pd
7 import matplotlib.pyplot as plt
8 import seaborn as sns
9 from sklearn.preprocessing import StandardScaler
10 from sklearn.model_selection import cross_val_score, train_test_split
11 from sklearn.metrics import mean_squared_error
12 from scipy.spatial.distance import cdist
13 from sklearn import tree
14 from tqdm import tqdm
15 import torch
16 from torch import Tensor
17 import numpy as np
18 from typing import Tuple, List, Dict
19 import pandas as pd
20 import matplotlib
21 import matplotlib.pyplot as plt
22 from collections import namedtuple
23 from IPython.display import display, Image
24 import random
25
```

<frozen importlib._bootstrap>:228: RuntimeWarning: scipy._lib.messagestream.MessageStream size changed, may indicate binary incompatibility. Expected 56 from C header, got 64 from PyObject

Question 1. Markov Decision Process (5 total points)

The Markov property states that the future depends only on the present and not on the past. The Markov chain is the probabilistic model that solely depends on the current state to predict the next state. In this section, we will join a dice game to determine a MDP.

1.1 Short answer: Dice game (5 pts)

Assume you are in Las Vegas, sitting on a playing slot machines for dice game.

For each round $r = 1, 2, \dots$ you choose **stay** or **quit**.

- If **quit**, you get \$10 and we end the game.
- If **stay**, you get \$4 and then roll a 6-sided dice.
 - If the dice results in 1 or 2, we end the game.
 - Otherwise, continue to the next round.

Question: Define states, actions, and determine the transition probabilities $T(\text{state}, \text{action}, \text{new_state})$ for the game.

The MDP has two states, s_0 where the player is in the game and s_{end} where the game ends. The MDP also has two actions, *stay* and *quit*. When the player is in s_0 , they choose between these two actions. If they choose *quit*, the game transitions to s_{end} with probability 1. If they choose *stay* they receive \$4 and roll a dice. With probability $1/3$ (which corresponds to sides 1 and 2) the game ends (transitions to s_{end}), while with probability $2/3$ (which corresponds to sides 3, 4, 5 or 6) the player remains in s_0 for the next round.

Question 2.1. Hidden Markov Models (HMMs) (2.5 total points)

Here is a nice brief tutorial on HMMs: <https://web.stanford.edu/~jurafsky/slp3/A.pdf> (<https://web.stanford.edu/~jurafsky/slp3/A.pdf>) .

Given a 3x3 grid shown in the figure below



The agent ended with a reward of 7, which means it lost 3 points before reaching the goal. We need to find the probability $P(\text{wall} | \text{Reward} = 7)$. In order to do that, we list all the paths that give a total reward of 7, and then check which ones involve hitting a wall. We will take the total probability of the wall paths and divide it by the total probability of all reward-7 paths.

Paths that give reward 7 and hit a wall:

1. $U, R, U, R, R = 0.4 \times 0.8 \times 0.5 \times 0.8 \times 0.5 \times 0.8 \times 0.8 \times 0.8 \times 0.9 \times 0.8 = 0.024$
2. $U, U(\text{failed}), U, R, R = 0.4 \times 0.8 \times 0.5 \times 0.2 \times 0.5 \times 0.8 \times 0.8 \times 0.8 \times 0.9 \times 0.8 = 0.006$
3. $R(\text{failed}), R, U, R, R = 0.6 \times 0.2 \times 0.5 \times 0.8 \times 0.5 \times 0.8 \times 0.8 \times 0.8 \times 0.9 \times 0.8 = 0.009$
4. $R(\text{failed}), U(\text{failed}), U, R, R = 0.6 \times 0.2 \times 0.5 \times 0.2 \times 0.5 \times 0.8 \times 0.8 \times 0.8 \times 0.9 \times 0.8 = 0.0022$

Total probability of hitting wall and getting reward 7 = $0.024 + 0.006 + 0.009 + 0.0022 = 0.0412$

Now, we need to identify all the possible paths that will take the agent to the Goal state with a reward of 7. Ofcourse all the previous paths are valid, but there are additional paths:

5. $R, R, U, U = 0.6 \times 0.8 \times 0.7 \times 0.8 \times 0.9 \times 0.8 \times 0.9 \times 0.8 = 0.1399$

$$6. U(\text{failed}), R, U, U = 0.4 \times 0.2 \times 0.7 \times 0.8 \times 0.9 \times 0.8 \times 0.9 \times 0.8 = 0.0323$$

$$7. R, U(\text{failed}), U, U = 0.6 \times 0.8 \times 0.3 \times 0.2 \times 0.9 \times 0.8 \times 0.9 \times 0.8 = 0.0249$$

$$8. U(\text{failed}), U(\text{failed}), U, U = 0.4 \times 0.2 \times 0.3 \times 0.2 \times 0.9 \times 0.8 \times 0.9 \times 0.8 = 0.0035$$

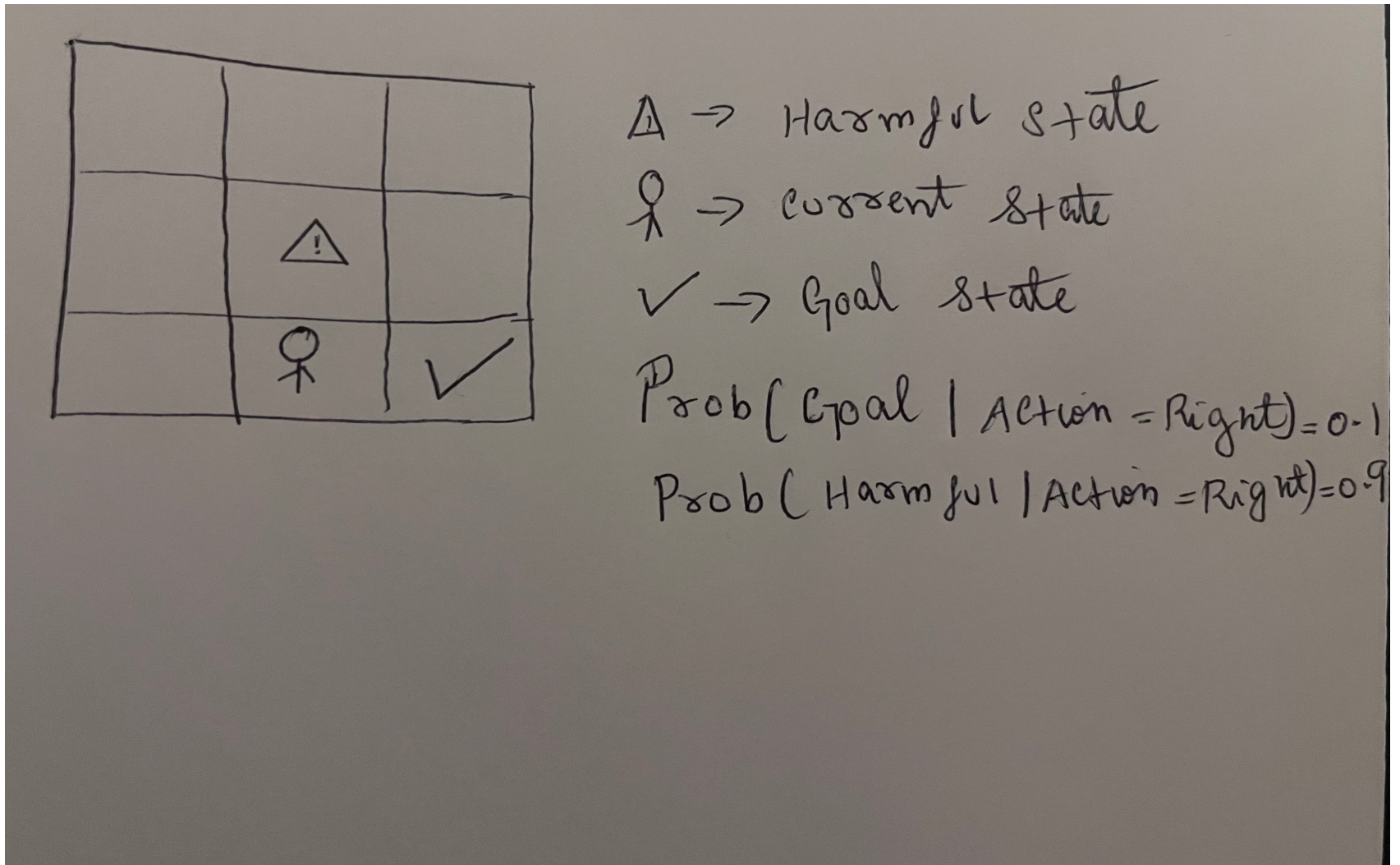
$$\text{Total probability of not hitting wall and getting reward 7} = 0.1399 + 0.0323 + 0.0249 + 0.0035 = 0.2006$$

To get the final answer, we divide the total probability of the wall paths by the total probability of all these paths
 $= 0.0412 / (0.0412 + 0.2006) = 0.1704$

Question 2.2. Hidden Markov Models (HMMs) (2.5 total points)

Based on the below diagram, Should the person move right or not ? Justify your answer elaborately.


```
In [ ]: 1 #Image("./Decision.jpg")
```



Write your answer in this block.

The person should not move right because the HMM indicates a 90% chance of ending up in a harmful state and only a 10% chance of reaching the goal. Even though the goal looks like it's to the right, the high risk makes it a bad choice.

Question 3. Text Denoising using HMM (45 total points)

In this question we will correct text errors using a hidden Markov model. To obtain text we can obtain copyright-free book in plain characters from Project Gutenberg. For this question we will be looking at Saddle room songs and hunting ballads by Frederick C. Palmer. You can download the txt version at <https://www.gutenberg.org/cache/epub/74589/pg74589.txt> (<https://www.gutenberg.org/cache/epub/74589/pg74589.txt>).

3.1 Code: Preprocess Text (5 pts)

Let us now preprocess the text by dropping all punctuation marks except for spaces, convert all capital letters to lower case, and mapping multiple spaces to a single space. The preprocessed text will have 27 symbols (26 lower case letters and the space character).

In [3]:

```
1 import requests
2 import re
3
4 # Function to download and save the file
5 def download_file(url, local_filename):
6     response = requests.get(url, verify=False)
7     response.raise_for_status() # Ensure the download was successful
8     with open(local_filename, 'wb') as f:
9         f.write(response.content)
10    return local_filename
11
12 def question_3_1(file_path: str) -> str:
13     '''
14     - file_path: path to the txt file downloaded from the URL
15     Returns:
16     - simplified_text: contains only lowercase letters and single spaces.
17     '''
18     with open(file_path, 'r', encoding='utf-8') as f:
19         text = f.read()
20
21     # Write your code in this block -----
22     text = text.lower()
23     text = re.sub(r'^a-z\s', '', text)
24     text = re.sub(r'\s+', ' ', text)
25     simplified_text = text
26
27
28
29     # End of your code -----
30
31    return simplified_text
32
33 url = "https://www.gutenberg.org/cache/epub/74589/pg74589.txt"
34 file_path = "pg74589.txt"
35
36 # Download the file and save it locally
37 download_file(url, file_path)
38
39 # Process the downloaded file
40 cleaned_text = question_3_1(file_path)
41 print(cleaned_text[:200])
```

42

the project gutenber ebook of saddle room songs and hunting ballads this ebook is for the use of an yone anywhere in the united states and most other parts of the world at no cost and with almost no r

```
/Users/themistoklis/opt/anaconda3/lib/python3.9/site-packages/urllib3/connectionpool.py:1043: Insecu  
reRequestWarning: Unverified HTTPS request is being made to host 'www.gutenberg.org'. Adding certifi  
cate verification is strongly advised. See: https://urllib3.readthedocs.io/en/1.26.x/advanced-usage.  
html#ssl-warnings (https://urllib3.readthedocs.io/en/1.26.x/advanced-usage.html#ssl-warnings)  
warnings.warn(
```

3.2 Code: Letter Frequencies (2 pts)

Based on the given text, count unigram, bigram, and trigram letter frequencies.

```
In [4]: 1 from collections import Counter
2
3 def question_3_2(text: str, n: int) -> Counter:
4     '''
5     - text: str, the input text to process
6     - n: int, the length of the n-grams to count
7
8     Returns:
9     - Counter: a Counter object where keys are n-grams and values are their respective counts
10    '''
11
12    # Write your code in this block -----
13    grams = []
14    for i in range(len(text) - n + 1):
15        ngram = text[i:i+n]
16        grams.append(ngram)
17    # End of your code -----
18    return Counter(grams)
19
20 unigram_counts = question_3_2(cleaned_text, 1)
21 unigrams = list(unigram_counts.items())[:10]
22 print(unigrams)
23
24
```

```
[('t', 2969), ('h', 1611), ('e', 3842), (' ', 7092), ('p', 652), ('r', 2197), ('o', 2617), ('j', 116), ('c', 912), ('g', 824)]
```

3.3 Code: Building n-gram Models (5 pts)

Use the counts from question 3.2 build models of unigram, bigram, and trigram letter probabilities.

```

In [5]: 1 def question_3_3(ngram_counts: dict, vocab_size: int) -> dict:
        2     '''
        3     - ngram_counts: dict, a dictionary containing n-gram counts
        4     - total_ngrams: int, the total number of n-grams in the text
        5     - vocab_size: int, the size of the vocabulary (unique n-grams)
        6
        7     Returns:
        8     - ngram_probs: dict, a dictionary with n-grams as keys and their smoothed probabilities as values
        9     '''
        10
        11     # Write your code in this block -----
        12     total_ngrams = sum(ngram_counts.values())
        13     ngram_probs = {}
        14     for ngram, count in ngram_counts.items():
        15         # Add-one smoothing
        16         ngram_probs[ngram] = (count + 1) / (total_ngrams + vocab_size)
        17
        18     # End of your code -----
        19     return ngram_probs
        20
        21 vocab_size=27
        22 unigram_probs = question_3_3(unigram_counts, vocab_size)
        23 print("Unigram probabilities:", list(unigram_probs.items())[:10])
        24

```

```

Unigram probabilities: [('t', 0.07690315898498187), ('h', 0.04174003107198343), ('e', 0.099508026929
0523), (' ', 0.1836613153806318), ('p', 0.016908337648886586), ('r', 0.0569135163127913), ('o', 0.06
778871051268773), ('j', 0.0030295183842568617), ('c', 0.023640600725012947), ('g', 0.021361988606939
41)]

```

3.4 Code: Corrupt Input Text (5 pts)

Now let us corrupt the input text through the following process: with probability P_c we will replace a character with a randomly selected character, else we will keep the original character with probability $1 - P_c$.

```

In [6]: 1 import random
        2 import string
        3
        4 def question_3_4(text: str, pc: float, vocab: list, seed: int) -> str:
        5     '''
        6     Arguments:
        7     - text: str, the input text to be corrupted
        8     - pc: float, the probability of replacing each character
        9     - vocab: list, the vocabulary of characters to choose from when replacing text
       10
       11     Returns:
       12     - corrupted_text: the corrupted version of the input text
       13     '''
       14
       15     # Write your code in this block -----
       16
       17     # Set the random seed for reproducibility
       18     random.seed(seed)
       19     corrupted_text = ""
       20
       21     for char in text:
       22         if random.random() < pc:
       23             replacement = random.choice(vocab)
       24             corrupted_text += replacement
       25         else: corrupted_text += char
       26     return corrupted_text
       27     # End of your code -----
       28
       29
       30 vocab = list(string.ascii_lowercase + ' ')
       31 seed=42
       32
       33 pc=0.1
       34 original_text = "this is just an example"
       35 corrupted_text = question_3_4(original_text, pc, vocab, seed)
       36 print("Original text:", original_text)
       37 print("Corrupted text:", corrupted_text)

```

Original text: this is just an example
 Corrupted text: tiis is cug an example

3.5 Code: Building transition matrix for HMM (5 pts)

Now you will build a transition matrix for a Hidden Markov Model (HMM), which is used to model the probabilities of transitioning from one character to another in a sequence of text. This matrix is essential for recovering sequence of true characters given some corrupted or noisy observations.

In [7]:

```

1 import numpy as np
2 def question3_5(bigram_counts: dict, vocab: list) -> np.ndarray:
3     '''
4     Arguments:
5     - bigram_counts: dict, a dictionary where keys are bigrams and values are the counts of those
6     - vocab: a list of all possible characters
7
8     Returns:
9     - transition_matrix: shape (vocab_size, vocab_size) where each entry represents the
10                          probability of transitioning from one character to another
11     '''
12
13     # Write your code in this block -----
14     # Initialize transition matrix
15     vocab_size = len(vocab)
16     transition_matrix = np.zeros((vocab_size, vocab_size))
17     char_to_index = {char: i for i, char in enumerate(vocab)}
18
19     # Calculate total bigram counts for each character
20     char_totals = {char: 0 for char in vocab}
21     for bigram, count in bigram_counts.items():
22         first_char = bigram[0]
23         char_totals[first_char] += count
24     # Fill transition matrix with bigram probabilities
25     for bigram, count in bigram_counts.items():
26         if len(bigram) == 2:
27             first_char = bigram[0]
28             second_char = bigram[1]
29             i = char_to_index[first_char]
30             j = char_to_index[second_char]
31             total_count = char_totals.get(first_char, 0)
32             transition_matrix[i, j] = count / (total_count + vocab_size)
33
34     row_sums = transition_matrix.sum(axis=1, keepdims=True)
35     transition_matrix = transition_matrix / row_sums
36
37
38     # End of your code -----
39     return transition_matrix
40
41
42
43

```

```
44 vocab = list(string.ascii_lowercase + ' ')
45 bigram_counts = question_3_2(cleaned_text, 2)
46 transition_matrix = question3_5(bigram_counts, vocab)
47 print(transition_matrix[0])
48
```

```
[0.          0.01637331 0.04379861 0.0462546  0.00081867 0.001228
0.02947196 0.          0.03029063 0.          0.01309865 0.07081457
0.01801064 0.20016373 0.          0.01760131 0.          0.10601719
0.1011052  0.13180516 0.01023332 0.01882931 0.01637331 0.00327466
0.04216128 0.00163733 0.08063856]
```

3.6 Code: Building emission matrix for HMM (5 pts)

We now need to build an emission matrix for a Hidden Markov Model (HMM). The emission matrix represents the probability of observing a certain character given the true underlying (hidden) character in a sequence of text.

The diagonal of the matrix should represent the probability that a character remains unchanged. The off-diagonal entries will represent the probability that a character is corrupted into another character.

```

In [8]: 1 def question3_6(vocab_size: int, corruption_prob: float) -> np.ndarray:
        2     '''
        3     Arguments:
        4     - vocab_size: int, the number of characters in the vocabulary
        5     - corruption_prob: float, the probability of a character being corrupted
        6
        7     Returns:
        8     - emission_matrix: shape (vocab_size, vocab_size) where each entry represents the probability
        9     '''
        10
        11     # Write your code in this block -----
        12     emission_matrix = np.zeros((vocab_size, vocab_size))
        13     off_diagonal = corruption_prob / (vocab_size - 1)
        14     for i in range(vocab_size):
        15         for j in range(vocab_size):
        16             if i == j:
        17                 emission_matrix[i, j] = 1 - corruption_prob
        18             else:
        19                 emission_matrix[i, j] = off_diagonal
        20     # End of your code -----
        21     return emission_matrix
        22 pc=0.1
        23 emission_matrix = question3_6(len(vocab), pc)
        24 print(emission_matrix[0])

```

```

[0.9          0.00384615 0.00384615 0.00384615 0.00384615 0.00384615
 0.00384615 0.00384615 0.00384615 0.00384615 0.00384615 0.00384615
 0.00384615 0.00384615 0.00384615 0.00384615 0.00384615 0.00384615
 0.00384615 0.00384615 0.00384615]

```

3.7 Code: Converting Text to Indices Based on Vocabulary (2 pts)

Write a function that takes a string of text and a vocabulary list. The function should convert each character in the input text into its corresponding index in the vocabulary list.

```
In [9]: 1 def question3_7(text: str, vocab: list) -> list:
2     '''
3     Arguments:
4     - text: str, the input text to be converted into indices
5     - vocab: list, the list of characters in the vocabulary
6
7     Returns:
8     - indices: a list of integers where each integer is the index of the corresponding character
9     '''
10
11     # Write your code in this block -----
12     char_to_index = {char: i for i, char in enumerate(vocab)}
13     results = []
14     for char in text:
15         if char in char_to_index:
16             results.append(char_to_index[char])
17
18
19     # End of your code -----
20     return results
```

3.8 Code: Converting Indices to Text Based on Vocabulary (2 pts)

Write a function that takes a list of indices and converts them back into a string using the provided vocabulary. Each index in the list corresponds to a character in the vocabulary.

```
In [10]: 1 def question3_8(indices: list, vocab: list) -> str:
2         '''
3         Arguments:
4         - indices: list of integers, where each integer is an index in the vocabulary
5         - vocab: list of characters in the vocabulary
6
7         Returns:
8         - A string where each character corresponds to the index from the vocabulary
9         '''
10
11         # Write your code in this block -----
12         result = ""
13         for i in indices:
14             if i < len(vocab):
15                 result += vocab[i]
16
17         # End of your code -----
18
19         return result
```

3.9 Code: One-Hot Encoding Observations for HMM (2 pts)

write a function that takes a list of indices and the size of the vocabulary, and returns a one-hot encoded matrix.

```
In [11]: 1 import numpy as np
2
3 def question3_9(indices: list, vocab_size: int) -> np.ndarray:
4     '''
5
6     - indices: list of integers, where each integer is an index in the vocabulary
7     - vocab_size: the size of the vocabulary
8
9     Returns:
10    - A 2D numpy array representing one-hot encoded vectors with shape (len(indices), vocab_size)
11    '''
12
13    # Write your code in this block -----
14    one_hot_encoded = np.zeros((len(indices), vocab_size))
15    for i, index in enumerate(indices):
16        if index < vocab_size:
17            one_hot_encoded[i][index] = 1
18
19
20    # End of your code -----
21
22    return one_hot_encoded
23
```

3.10 Code: Training HMM (5 pts)

In this question we will train a Hidden Markov Model (HMM) using bigram counts from a given text to decode the corrupted text.

We will use hmmlearn library's **MultinomialHMM** class to create the model.

In [12]: 1 !pip install hmmlearn

```
Requirement already satisfied: hmmlearn in /Users/themistoklis/opt/anaconda3/lib/python3.9/site-packages (0.3.3)
Requirement already satisfied: numpy>=1.10 in /Users/themistoklis/opt/anaconda3/lib/python3.9/site-packages (from hmmlearn) (1.23.5)
Requirement already satisfied: scipy>=0.19 in /Users/themistoklis/opt/anaconda3/lib/python3.9/site-packages (from hmmlearn) (1.10.1)
Requirement already satisfied: scikit-learn!=0.22.0,>=0.16 in /Users/themistoklis/opt/anaconda3/lib/python3.9/site-packages (from hmmlearn) (1.4.1.post1)
Requirement already satisfied: joblib>=1.2.0 in /Users/themistoklis/opt/anaconda3/lib/python3.9/site-packages (from scikit-learn!=0.22.0,>=0.16->hmmlearn) (1.3.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/themistoklis/opt/anaconda3/lib/python3.9/site-packages (from scikit-learn!=0.22.0,>=0.16->hmmlearn) (3.2.0)
```


In [13]:

```

1 from hmmlearn import hmm
2 def question3_10(original_text: str, bigram_counts: dict, vocab: list, pc: float) -> hmm.MultinomialHMM:
3     '''
4     Arguments:
5     - original_text: str, the original text from which to train the HMM
6     - vocab: list, the list of characters in the vocabulary
7     - pc: float, the probability of character corruption (used for the emission matrix)
8
9     Returns:
10    - model: hmm.MultinomialHMM, the trained Hidden Markov Model
11    '''
12
13    # Write your code in this block -----
14
15    # build the transition matrix and emission matrix. You can use question3_5 and question3_6
16    transition_matrix = question3_5(bigram_counts, vocab)
17    emission_matrix = question3_6(len(vocab), pc)
18
19    # Initialize the HMM model using hmm.MultinomialHMM and set n_trials to 1
20    model = hmm.MultinomialHMM(n_components=len(vocab), n_trials=1)
21    # Initialize with uniform start probability
22    start_prob = np.ones(len(vocab)) / len(vocab)
23    model.startprob_ = start_prob
24    model.transmat_ = transition_matrix
25    model.emissionprob_ = emission_matrix
26
27
28    # End of your code -----
29    return model
30
31 original_text=cleaned_text
32 corrupted_text = question3_4(original_text, 0.1, vocab, 42)
33 bigram_counts = question3_2(original_text, 2)
34 model = question3_10(original_text, bigram_counts, vocab, 0.1)
35
36 # Convert corrupted text to indices and one-hot encode it
37 corrupted_indices = question3_7(corrupted_text, vocab)
38 one_hot_corrupted = question3_9(corrupted_indices, len(vocab))
39
40 # Decode the corrupted text
41 _, recovered_indices = model.decode(one_hot_corrupted)

```

```
42 recovered_text = question3_8(recovered_indices, vocab)
```

MultinomialHMM has undergone major changes. The previous version was implementing a CategoricalHMM (a special case of MultinomialHMM). This new implementation follows the standard definition for a Multinomial distribution (e.g. as in https://en.wikipedia.org/wiki/Multinomial_distribution). (https://en.wikipedia.org/wiki/Multinomial_distribution.) See these issues for details: <https://github.com/hmmlearn/hmmlearn/issues/335> (<https://github.com/hmmlearn/hmmlearn/issues/335>) <https://github.com/hmmlearn/hmmlearn/issues/340> (<https://github.com/hmmlearn/hmmlearn/issues/340>)

3.11 Code: Estimating Error Rate Using Hamming Distance (2 pts)

We will estimate the error rate between an original text and a recovered (corrupted or noisy) version of that text using the Hamming distance. The Hamming distance between two sequences is the total number of locations in which the values are different.

Hamming distance is expressed as follows:

$$d_h(x, x(u)) = \sum_k \mathbb{1}(x_k \neq x(u)_k)$$

```

In [14]: 1 def question3_11(original_text: str, recovered_text: str) -> float:
          2     '''
          3     - original_text: str, the original text
          4     - recovered_text: str, the recovered text
          5
          6     Returns:
          7     - error_rate: float, the error rate as the proportion of differing characters
          8     '''
          9     # Write your code in this block -----
         10     minimum_length = min(len(original_text), len(recovered_text))
         11     hamming_distance = 0
         12     for i in range(minimum_length):
         13         if original_text[i] != recovered_text[i]:
         14             hamming_distance += 1
         15     error_rate = hamming_distance / minimum_length
         16
         17
         18
         19     # End of your code -----
         20
         21     return error_rate
         22 error_rate = question3_11(original_text, recovered_text)
         23 print(f"Error rate: {error_rate * 100:.2f}%")

```

Error rate: 8.70%

3.12 Short answer: Estimate Error Rate (5 pts)

For $p_c = 0.01$ and $p_c = 0.1$, estimate the respective error rate for the corrected text. Does the result make sense? Why or why not?

Write your answer in this block

For $p_c = 0.01$, the estimated error rate is 0.86%, while for $p_c = 0.1$, it increases to 8.7%. This result makes sense because a higher corruption probability leads to more noise in the text, which also makes recovery more difficult. Although the HMM may be able to correct some errors, its accuracy decreases as the corruption becomes higher.

Question 4. Creating Dataloaders in Pytorch (5 total points)

This homework will introduce you to [PyTorch \(https://pytorch.org\)](https://pytorch.org), currently the fastest growing deep learning library, and the one we will use in this course.

Before starting the homework, please go over these introductory tutorials on the PyTorch webpage: [60-minute Blitz \(https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html\)](https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html).

4.1 Code: Data Loader (5 pts)

For the following sections, we will work on [CIFAR-10 \(https://www.cs.toronto.edu/~kriz/cifar.html\)](https://www.cs.toronto.edu/~kriz/cifar.html) dataset.

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

First, let's download and preprocess the dataset

```
In [15]: 1 import torchvision
2 import torchvision.transforms as transforms
3 from torchvision import transforms, datasets
4 from torch.utils.data.dataloader import DataLoader
5
6 transform = transforms.Compose(
7     [
8         transforms.ToTensor(),
9         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
10    ])
11
12
13 batch_size = 16
14
15 trainset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
16 trainloader = DataLoader(trainset, batch_size=batch_size, shuffle=True, num_workers=2, drop_last=True)
17
18 classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

Downloading <https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz> (<https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz>) to ./data/cifar-10-python.tar.gz

100%|██████████| 170498071/170498071 [00:38<00:00, 4446897.49it/s]

Extracting ./data/cifar-10-python.tar.gz to ./data

Let's show some of the pictures in the dataset.


```

In [16]: 1 def imshow(img):
          2     img = img / 2 + 0.5
          3     npimg = img.numpy()
          4     plt.imshow(np.transpose(npimg, (1, 2, 0)))
          5
          6
          7     # get some random training images
          8     sample = iter(trainloader)
          9     images, labels = next(sample)
         10
         11     # show some images
         12     imshow(torchvision.utils.make_grid(images))
         13
         14     # print labels for the first row
         15     print(' '.join('%4s' % classes[labels[j]] for j in range(8)))

```

ship frog ship frog cat dog ship frog



Similar to the example above, complete the function below to create a "testloader"

```

In [17]: 1 def question_4_2(transform: transforms.Compose, batch_size:int, shuffle:bool, drop_last:bool) ->
          2     """
          3     Similar to the example above, create then return a DataLoader for testset
          4     """
          5     # Write your code in this block -----
          6     ## Step 1: create `testset` using datasets.CIFAR10: similar to `trainset` above, with
          7     # download=True, transform=transform but set `train` to False
          8     testset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
          9     ## Step 2: create `testloader` using DataLoader and passing params `batch_size`, `shuffer`, `
         10     testloader = DataLoader(testset, batch_size=batch_size, shuffle=shuffle, num_workers=2, drop_
         11     ## Step 3: return `testloader`
         12     return testloader
         13     # End of your code -----
         14
         15
         16 shuffle = False
         17 drop_last = True
         18 testloader = question_4_2(transform, batch_size, shuffle, drop_last)
         19 testloader

```

Files already downloaded and verified

Out[17]: <torch.utils.data.data_loader.DataLoader at 0x7fa856e4afa0>

Question 5. Training Models in Pytorch (25 total points)

In this problem, we build a 2-layer MLP to predict the class for the images.

If you're using Colab, you can use its GPU to speed up the training. To opt to GPU, go to tab [Runtime -> Change Runtime type](#) (https://raw.githubusercontent.com/chaudatascience/cs599_fall2022/master/ps4/gpu.png) -> Choose GPU for Hardware accelerator.

5.1 Code: 2-layer MLP (5 pts)

Make a subclass of the `Module` class, called `MyMLP`. You will need to define the `__init__` and `forward` methods. Our neuron would have 2 linear layers and 1 ReLU layer.

You can refer to the [Neural Networks tutorial of Pytorch \(https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html\)](https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html).

```

In [18]: 1 import torch.nn as nn
          2
          3 class MyMLP(nn.Module):
          4     def __init__(self, in_dim: int, hid_dim: int, out_dim: int):
          5         """
          6         in_dim: input dimension, usually we flatten 3d images (num_channels, width, height) to 1
          7         so we have in_dim = num_channels * width * height
          8         hid_dim: hidden dimension
          9         out_dim: output dimension
          10        """
          11         super().__init__()
          12
          13         ## Complete the code below to initialize self.linear_1, self.linear_2, self.relu
          14         # where self.linear_1 and self.linear_2 are `nn.Linear` objects, shape of (in_dim, hid_dim)
          15         # and self.relu is a `nn.ReLU` object.
          16         # Write your code in this block -----
          17         self.linear_1 = nn.Linear(in_dim, hid_dim)
          18         self.relu = nn.ReLU()
          19         self.linear_2 = nn.Linear(hid_dim, out_dim)
          20         # End of your code -----
          21
          22     def forward(self, x: Tensor) -> Tensor:
          23         ## Assume we want to build a model as following: input `x` -> linear_1 -> relu -> linear_2
          24         ## Write your forward pass
          25         # Write your code in this block -----
          26         x = self.linear_1(x)
          27         x = self.relu(x)
          28         x = self.linear_2(x)
          29         return x
          30         # End of your code -----
          31

```

5.2 Code: Train 1 epoch (5 pts)

For each batch in the training set, we use 5 steps to train a Pytorch model You will need to fill in the steps below.

In [19]:

```

1
2 def question_5_2_train_one_epoch(model: nn.Module, trainloader: DataLoader, device: torch.device,
3                                   optimizer: torch.optim.SGD, criterion: torch.nn.CrossEntropyLoss)
4     """
5     Train 1 epoch on trainloader. You need to fill in after "##### [YOUR CODE]"
6     """
7
8     ## Set model to "train" model
9     model = model.train()
10
11     ## Keep track of loss and accuracy
12     train_loss = 0.0
13     train_acc = 0.0
14
15     ## Loop over all the batches
16     for i, (images, labels) in tqdm(enumerate(trainloader, 1), total=len(trainloader), desc=f"tra
17         # For each batch, we have:
18         #     + `images`: `batch_size` images in training set
19         #     + `labels`: labels of the images (`batch_size` labels)
20
21
22     ## Reshape the input dimension if we use MLP: instead of 3d (num_channels, width, height)
23     # we flatten it to 1d (num_channels * width * height)
24     if flatten:
25         images = images.reshape(batch_size, -1)
26
27     ## Move images and labels to `device` (CPU or GPU)
28     images = images.to(device)
29     labels = labels.to(device)
30
31     # Write your code in this block -----
32     ## We use 5 following steps to train a Pytorch model
33
34     ##### [YOUR CODE] Step 1. Forward pass: pass the data forward, the model try its best to
35     # You need to get the output from the model, store in a new variable named `logits`
36     logits = model(images)
37
38     ##### [YOUR CODE] Step 2. Compare the output that the model gives us with the real labels
39     ## You need to compute the loss, store in a new variable named `loss`
40     loss = criterion(logits, labels)
41
42
43     ##### [YOUR CODE] Step 3. Clear the gradient buffer

```

```
44 optimizer.zero_grad()
45
46 ##### [YOUR CODE] Step 4. Backward pass: calculate partial derivatives of the loss w.r.t
47 loss.backward()
48
49 ##### [YOUR CODE] Step 5. Update the parameters by stepping in the opposite direction from
50 optimizer.step()
51
52 # End of your code -----
53 ## Compute loss and accuracy for this batch
54 train_loss += loss.detach().item()
55 train_acc += question_5_3_compute_accuracy(logits, labels, batch_size)
56
57 return train_loss/i, train_acc/i ## avg loss and acc over all batches
58
```


5.3 Code: Compute accuracy (5 pts)

```
In [20]: 1  ## compute accuracy score in a batch
2  def question_5_3_compute_accuracy(logits: Tensor, labels: Tensor, batch_size: int) -> float:
3      '''
4          Obtain accuracy for a training batch
5          logits: float Tensor, shape (batch_size, num_classes), output from the model
6          labels: Long Tensor, shape (batch_size, ), contains labels for the predictions
7          batch_size: int, batch size
8
9          Return accuracy for this batch, which should be a float number in [0, 100], NOT a Tensor
10     '''
11
12     # Write your code in this block -----
13     predictions = torch.argmax(logits, dim=1)
14     correct_predictions = (predictions == labels).sum().item()
15     accuracy = (correct_predictions / batch_size) * 100
16     return accuracy
17     # End of your code -----
18
```

5.4 Code: evaluate (5 pts)

We will write a function to evaluate our model on test set after each epoch

In [21]:

```
1  ## Note that we use `torch.no_grad()` here to disable gradient calculation.
2  # It will reduce memory consumption as we don't need to compute gradients in inference.
3
4  @torch.no_grad()
5  def question_5_4_evaluate(model: nn.Module, testloader: DataLoader, criterion, batch_size, device):
6      """
7      You need to fill in after "#### [YOUR CODE]"
8      """
9
10     test_acc = 0.0
11     test_loss = 0.0
12
13     ## Turn on the evaluation mode
14     model.eval()
15
16     ## Loop through each batch on test set
17     for i, (images, labels) in enumerate(testloader, 1):
18
19         ## Flatten the image into 1d if using MLP
20         if flatten:
21             images = images.reshape(batch_size, -1)
22
23         # Write your code in this block -----
24
25         #### [YOUR CODE] Move data to `device`
26         images = images.to(device)
27         labels = labels.to(device)
28         #### [YOUR CODE] forward pass to get the output of the model
29         logits = model(images)
30
31         #### [YOUR CODE] Compute the loss
32         loss = criterion(logits, labels)
33
34         #### [YOUR CODE] Compute accuracy (re-use question 5.3)
35         acc = question_5_3_compute_accuracy(logits, labels, batch_size)
36         test_acc += acc
37
38         test_loss += loss.item()
39         # End of your code -----
40
41     return test_loss/i, test_acc/i ## avg loss and acc over all batches
```

42

5.5 Code: Train model (5 pts)

Let's put everything together. Now we're ready to train a neural network!

In [22]:

```

1 import torch.optim as optim
2 from time import time
3 from collections import defaultdict
4
5 def question_5_5_train_model(model, device, num_epochs, batch_size, trainloader, testloader, flatten):
6     """
7     model: Our neural net
8     device: CPU/GPU
9     num_epochs: How many epochs to train
10    batch_size: batch size
11    train/test loaders: training/test data
12    flatten: whether we want to flatten the input image from 3d to 1d
13
14    You need to fill in after "##### [YOUR CODE]"
15    """
16
17    # Write your code in this block -----
18
19    ##### [YOUR CODE] create your optimizer using `optim.SGD`, set learning rate to 0.001, and momentum=0.9
20    optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
21
22    ##### [YOUR CODE] create criterion using `nn.CrossEntropyLoss`
23    criterion = nn.CrossEntropyLoss()
24
25    ## Measure runtime
26    t_start = time()
27
28    ## Store training log
29    history = defaultdict(list)
30
31    # We will train the model `num_epochs` times
32    for i in range(1, num_epochs+1):
33        ##### [YOUR CODE] train 1 epoch: call the function in question 5.2
34        train_loss, train_acc = question_5_2_train_one_epoch(model, trainloader, device, optimizer, criterion, batch_size, dev
35
36
37        ##### [YOUR CODE] call function in question 5.4 to see how it performs on test set
38        test_loss, test_acc = question_5_4_evaluate(model, testloader, criterion, batch_size, device)
39
40    # End of your code -----
41
42
43    ## store train/test loss, accuracy

```

```

44     history["train_loss"].append(train_loss)
45     history["train_acc"].append(train_acc)
46     history["test_loss"].append(test_loss)
47     history["test_acc"].append(test_acc)
48
49     ## print out train/test loss, accuracy
50     print(f'Epoch: {i} | Runtime: {((time()-t_start)/60):.2f}[m] | train_loss: {train_loss:.3f} | test_loss: {test_loss:.3f} | train_acc: {train_acc:.3f} | test_acc: {test_acc:.3f}')
51     return history

```

Let's check again on the shape of each batch

In [23]: `1 images.shape ## (batch size, num channels, width, height)`

Out[23]: `torch.Size([16, 3, 32, 32])`

In [24]:

```

1 ## Let's create our MLP
2 in_dim, hid_dim, out_dim = 3*32*32, 256, 10
3
4 # set device: We will use GPU if it's available, otherwise CPU
5 # GPU is much faster to train neural nets
6 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
7 num_epochs = 10
8
9 my_mlp = MyMLP(in_dim, hid_dim, out_dim)
10
11 ## Move model to `device`: We need everthing (i.e., model, data) on the same device, either CPU,
12 my_mlp.to(device)

```

Out[24]: `MyMLP(
 (linear_1): Linear(in_features=3072, out_features=256, bias=True)
 (relu): ReLU()
 (linear_2): Linear(in_features=256, out_features=10, bias=True)
)`

Before training, let take a look at `torchsummary` [\(https://pypi.org/project/torch-summary/\)](https://pypi.org/project/torch-summary/). It provides a useful tool for a summary of the model.

```
In [25]: 1 from torchsummary import summary
        2 summary(my_mlp, (in_dim, ))
```

```
-----
              Layer (type)              Output Shape          Param #
=====
              Linear-1                  [-1, 256]              786,688
              ReLU-2                    [-1, 256]                0
              Linear-3                  [-1, 10]               2,570
=====
Total params: 789,258
Trainable params: 789,258
Non-trainable params: 0
-----
Input size (MB): 0.01
Forward/backward pass size (MB): 0.00
Params size (MB): 3.01
Estimated Total Size (MB): 3.03
-----
```

Our model has 786k trainable parameters.

Note that "-1" in *Output Shape* denotes batch size dimension, which is not a part of the model architecture.

In [26]:

```
1 ## Train MLP model
2 flatten = True
3 history_mlp = question_5_5_train_model(my_mlp, device, num_epochs, batch_size, trainloader, testloader)
```

training 1 epoch...: 100%|██████████| 3125/3125 [00:23<00:00, 133.54it/s]

Epoch: 1 | Runtime: 0.90[m] | train_loss: 1.703 | train_acc: 40.324 | test_loss: 1.555 | test_acc: 45.970

training 1 epoch...: 100%|██████████| 3125/3125 [00:19<00:00, 157.86it/s]

Epoch: 2 | Runtime: 1.67[m] | train_loss: 1.495 | train_acc: 47.694 | test_loss: 1.460 | test_acc: 49.110

training 1 epoch...: 100%|██████████| 3125/3125 [00:21<00:00, 147.65it/s]

Epoch: 3 | Runtime: 2.47[m] | train_loss: 1.399 | train_acc: 51.134 | test_loss: 1.418 | test_acc: 50.240

training 1 epoch...: 100%|██████████| 3125/3125 [00:24<00:00, 129.39it/s]

Epoch: 4 | Runtime: 3.40[m] | train_loss: 1.329 | train_acc: 53.732 | test_loss: 1.389 | test_acc: 51.360

training 1 epoch...: 100%|██████████| 3125/3125 [00:20<00:00, 150.60it/s]

Epoch: 5 | Runtime: 4.21[m] | train_loss: 1.273 | train_acc: 55.822 | test_loss: 1.377 | test_acc: 51.740

training 1 epoch...: 100%|██████████| 3125/3125 [00:23<00:00, 132.14it/s]

Epoch: 6 | Runtime: 5.10[m] | train_loss: 1.223 | train_acc: 57.310 | test_loss: 1.361 | test_acc: 52.360

training 1 epoch...: 100%|██████████| 3125/3125 [00:22<00:00, 138.82it/s]

Epoch: 7 | Runtime: 5.98[m] | train_loss: 1.178 | train_acc: 59.080 | test_loss: 1.374 | test_acc: 51.680

training 1 epoch...: 100%|██████████| 3125/3125 [00:20<00:00, 150.46it/s]

Epoch: 8 | Runtime: 6.79[m] | train_loss: 1.136 | train_acc: 60.524 | test_loss: 1.362 | test_acc: 52.360

training 1 epoch...: 100%|██████████| 3125/3125 [00:21<00:00, 147.67it/s]

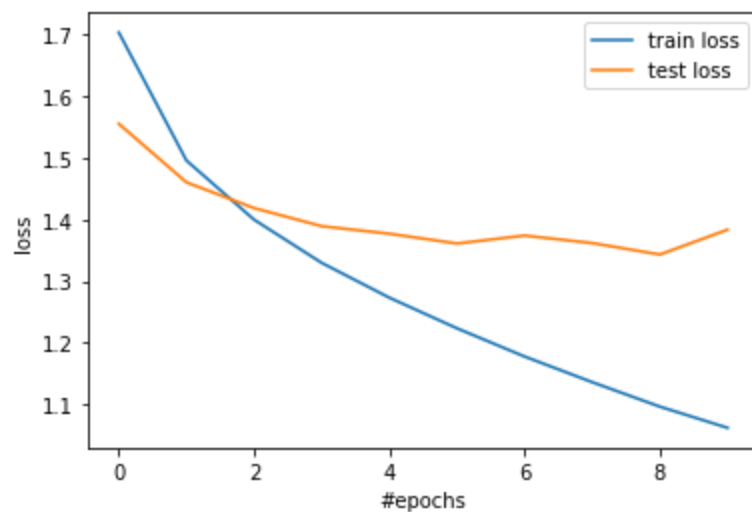
Epoch: 9 | Runtime: 7.58[m] | train_loss: 1.096 | train_acc: 61.870 | test_loss: 1.343 | test_acc: 53.700

training 1 epoch...: 100%|██████████| 3125/3125 [00:27<00:00, 115.21it/s]

Epoch: 10 | Runtime: 8.50[m] | train_loss: 1.062 | train_acc: 63.242 | test_loss: 1.383 | test_acc: 51.890

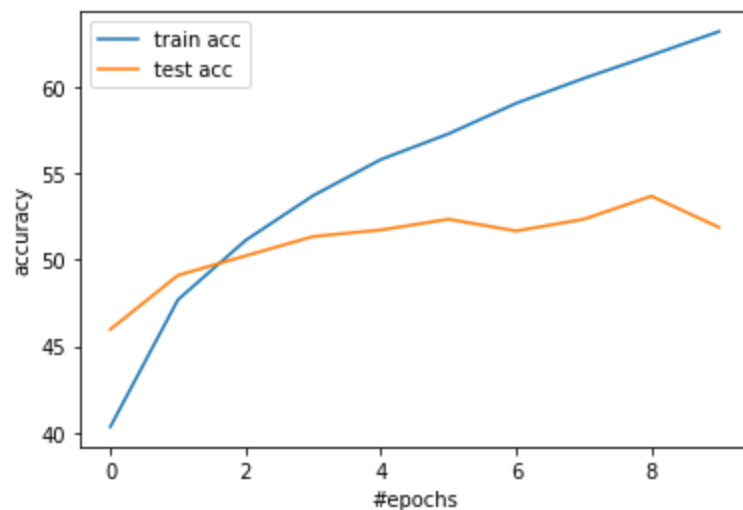
```
In [27]: 1 ## Plot train_loss, test_loss
2 plt.plot(np.arange(num_epochs), history_mlp["train_loss"], label='train loss')
3 plt.plot(np.arange(num_epochs), history_mlp["test_loss"], label='test loss')
4 plt.xlabel('#epochs')
5 plt.ylabel('loss')
6 plt.legend()
```

Out[27]: <matplotlib.legend.Legend at 0x7fa8207301c0>



```
In [28]: 1 ## Plot accuracy
2 plt.plot(np.arange(num_epochs), history_mlp["train_acc"], label='train acc')
3 plt.plot(np.arange(num_epochs), history_mlp["test_acc"], label='test acc')
4 plt.xlabel('#epochs')
5 plt.ylabel('accuracy')
6 plt.legend()
```

Out[28]: <matplotlib.legend.Legend at 0x7fa825bca1f0>



Question 6. CNN (15 total points)

CNNs work well with datasets that have locality (data near each other will be more similar) and compositionality (the object consists of small parts). It has been using as a go-to method for images.

In this section, we will make a simple CNN model and train it on CIFAR10 dataset.

6.1 Code: CNN model (10 pts)

Now Let's build the 1 CNN model.

- In our CNN model, we have a convolutional layer denoted by `nn.Conv2d(...)`. We are dealing with an image dataset that is in RGB, so we need 3 channel going in, hence `in_channels=3`. We hope to get a nice representation of this layer, so we use `out_channels=32`. Kernel size is 5, and for the rest of parameters we use the default values which you can find [here](https://pytorch.org/docs/stable/nn.html?highlight=conv2d#conv2d) (<https://pytorch.org/docs/stable/nn.html?highlight=conv2d#conv2d>).
- After each one of CONV or Linear Layers, we also apply an activation function such as `ReLU`.

Denote: **block_i = (conv_i -> relu -> pool)**

Let's say we want to build a CNN model as follows: **input x -> block_1 -> block_2 -> flatten -> fc1 -> relu -> fc2 -> relu -> fc3**
(where fc is linear layer, you can refer to the model summary below)

In [29]:

```

1 import torch.nn as nn
2 import torch.nn.functional as F
3
4 class MyCNN(nn.Module):
5     def __init__(self):
6         super().__init__()
7         ## Recall that input images are 3x32x32,
8         # i.e., 3 channels (red, green, blue), each of size 32x32 pixels.
9
10
11     ## An example of conv and pooling layers
12     self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=5)
13     # The first convolutional layer, 'conv1', expects 3 input channels,
14     # and will convolve 32 filters each of size 3x5x5.
15     # Since padding is set to 0 and stride is set to 1 as default,
16     # the output size is (32, 28, 28).
17     # This layer has ((3*5*5)+1)*32 = 2,432 parameters
18
19
20     self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
21     ## The first down-sampling layer uses max pooling with a (2,2) kernel
22     # and a stride of 2. This effectively drops half of spatial size.
23
24     self.conv2 = nn.Conv2d(in_channels=32, out_channels=16, kernel_size=5)
25     ## Similarly, we make another conv layer
26
27
28     # Write your code in this block -----
29
30     self.fc1 = nn.Linear(16 * 5 * 5, 64)
31     ## fc1 is a Linear layer. You'll need to look at the output of conv2 and the input of fc2
32     # determine the in_dim and out_dim for fc1. You can do this by printing out the shape of
33     # End of your code -----
34
35
36     self.fc2 = nn.Linear(64, 32)
37     self.fc3 = nn.Linear(32, 10)
38     self.relu = nn.ReLU()
39
40     def forward(self, x):
41
42     # Implement your forward pass
43     # Write your code in this block -----

```

```
44     x = self.conv1(x)
45     x = self.relu(x)
46     x = self.pool(x)
47
48     x = self.conv2(x)
49     x = self.relu(x)
50     x = self.pool(x)
51
52     x = x.view(x.size(0), -1)
53
54     x = self.fc1(x)
55     x = self.relu(x)
56     x = self.fc2(x)
57     x = self.relu(x)
58     x = self.fc3(x)
59     # End of your code -----
60
61     return x
```

```
In [30]: 1 my_cnn = MyCNN().to(device)
          2 summary(my_cnn, images.shape[1:])
```

```
-----
Layer (type)           Output Shape          Param #
=====
      Conv2d-1         [-1, 32, 28, 28]      2,432
      ReLU-2           [-1, 32, 28, 28]        0
    MaxPool2d-3        [-1, 32, 14, 14]        0
      Conv2d-4         [-1, 16, 10, 10]     12,816
      ReLU-5           [-1, 16, 10, 10]        0
    MaxPool2d-6        [-1, 16, 5, 5]         0
      Linear-7          [-1, 64]             25,664
      ReLU-8           [-1, 64]                0
      Linear-9          [-1, 32]              2,080
      ReLU-10          [-1, 32]                0
      Linear-11         [-1, 10]               330
=====
Total params: 43,322
Trainable params: 43,322
Non-trainable params: 0
-----
Input size (MB): 0.01
Forward/backward pass size (MB): 0.46
Params size (MB): 0.17
Estimated Total Size (MB): 0.64
-----
```



```
In [31]: 1 ## Train CNN
        2 history_cnn = question_5_5_train_model(my_cnn, device, num_epochs, batch_size, trainloader, testloader)

training 1 epoch...: 100%|██████████| 3125/3125 [00:33<00:00, 93.46it/s]
Epoch: 1 | Runtime: 1.03[m] | train_loss: 2.009 | train_acc: 26.040 | test_loss: 1.631 | test_acc: 40.190

training 1 epoch...: 100%|██████████| 3125/3125 [00:40<00:00, 77.35it/s]
Epoch: 2 | Runtime: 2.18[m] | train_loss: 1.515 | train_acc: 44.604 | test_loss: 1.434 | test_acc: 48.620

training 1 epoch...: 100%|██████████| 3125/3125 [00:39<00:00, 78.26it/s]
Epoch: 3 | Runtime: 3.33[m] | train_loss: 1.329 | train_acc: 51.656 | test_loss: 1.259 | test_acc: 54.850

training 1 epoch...: 100%|██████████| 3125/3125 [00:31<00:00, 98.88it/s]
Epoch: 4 | Runtime: 4.28[m] | train_loss: 1.200 | train_acc: 56.836 | test_loss: 1.159 | test_acc: 58.110

training 1 epoch...: 100%|██████████| 3125/3125 [00:31<00:00, 100.32it/s]
Epoch: 5 | Runtime: 5.22[m] | train_loss: 1.107 | train_acc: 60.574 | test_loss: 1.077 | test_acc: 61.730

training 1 epoch...: 100%|██████████| 3125/3125 [00:31<00:00, 100.03it/s]
Epoch: 6 | Runtime: 6.17[m] | train_loss: 1.030 | train_acc: 63.652 | test_loss: 1.029 | test_acc: 64.260

training 1 epoch...: 100%|██████████| 3125/3125 [00:33<00:00, 94.16it/s]
Epoch: 7 | Runtime: 7.15[m] | train_loss: 0.971 | train_acc: 65.764 | test_loss: 1.039 | test_acc: 63.900

training 1 epoch...: 100%|██████████| 3125/3125 [00:32<00:00, 95.31it/s]
Epoch: 8 | Runtime: 8.12[m] | train_loss: 0.926 | train_acc: 67.140 | test_loss: 0.981 | test_acc: 65.130

training 1 epoch...: 100%|██████████| 3125/3125 [00:33<00:00, 94.66it/s]
```

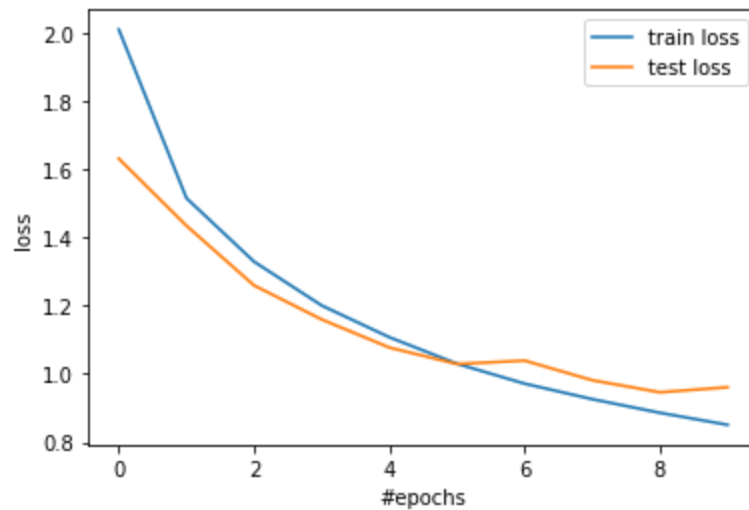
Epoch: 9 | Runtime: 9.09[m] | train_loss: 0.886 | train_acc: 68.690 | test_loss: 0.946 | test_acc: 66.640

training 1 epoch...: 100%|██████████| 3125/3125 [00:32<00:00, 95.83it/s]

Epoch: 10 | Runtime: 10.07[m] | train_loss: 0.851 | train_acc: 69.754 | test_loss: 0.961 | test_acc: 65.930

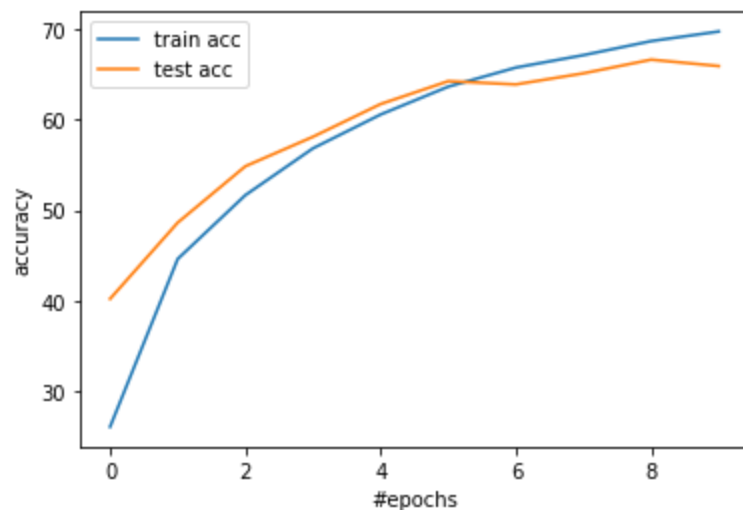
```
In [32]: 1 ## Plot train_loss, test_loss of CNN model
2 plt.plot(np.arange(num_epochs), history_cnn["train_loss"], label='train loss')
3 plt.plot(np.arange(num_epochs), history_cnn["test_loss"], label='test loss')
4 plt.xlabel('#epochs')
5 plt.ylabel('loss')
6 plt.legend()
```

Out[32]: <matplotlib.legend.Legend at 0x7fa820cb2e20>



```
In [33]: 1 ## Plot accuracy
2
3 plt.plot(np.arange(num_epochs), history_cnn["train_acc"], label='train acc')
4 plt.plot(np.arange(num_epochs), history_cnn["test_acc"], label='test acc')
5 plt.xlabel('#epochs')
6 plt.ylabel('accuracy')
7 plt.legend()
```

Out[33]: <matplotlib.legend.Legend at 0x7fa820c698e0>



6.2.1 Short answer: MLP vs. CNN (2.5 pts)

Compare to the results of MLP and CNN, which model learns better on image dataset - MLP or CNN? Explain your answer.

Hint: You can discuss on accuracy, runtime, number of parameters, etc.

Write your answer in this block

The CNN model learns better on the image dataset compared to the MLP. It achieves a higher test accuracy ~ 67 than the MLP ~ 53 and a lower test loss ~ 0.94 vs ~ 1.35 , showing better generalization. CNNs took slightly longer to train per epoch, but it's worth noting that I ran both models locally without using any GPUs, so slower overall training time was expected.

6.2.2 Short answer: You are given a MLP model. Briefly describe in 5 steps as to how you can convert a MLP into a CNN model. (2.5 pts)

Write your answer in this block

1. We should not flatten the input at the start but keep the image as a 3D tensor.
2. We should replace the first fully connected layer with a convolutional layer.
3. We should add convolutional blocks (like we did earlier Conv2d - Relu - MaxPool2d) to extract features
4. After the final convolutional or pooling layer, we should flatten the tensor before feeding it into fully connected layers.
5. Last but not least, we should adjust the first linear layer's input size to match the flattened size, and keep the later fully connected layers for classification.

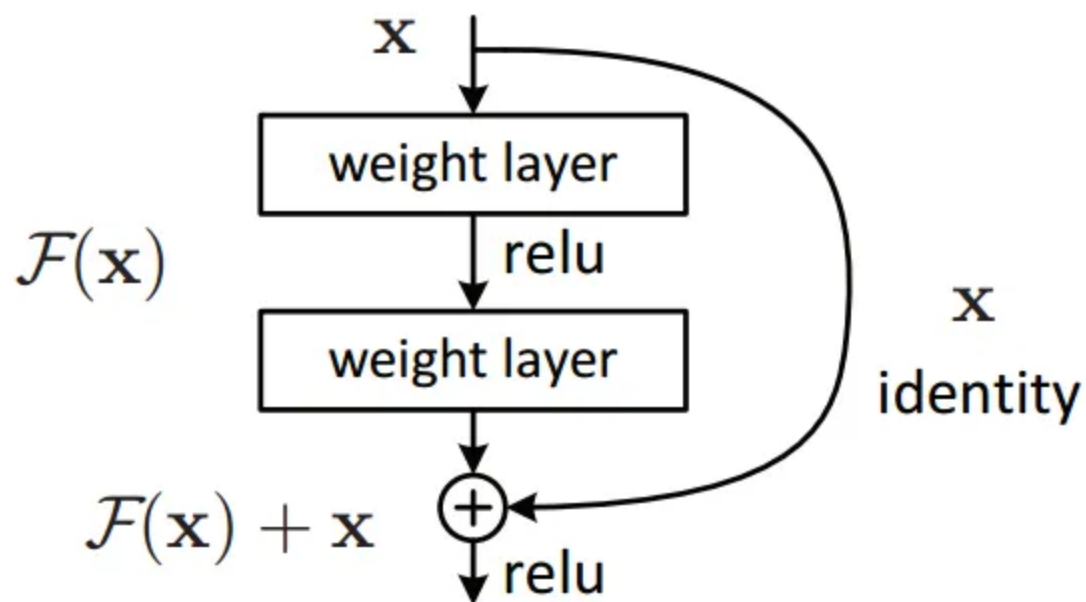
6.3 (Bonus) CNN Architecture. (10 points - Manual grading)

We have built a simple CNN model, and it works quite well on the dataset. Assume we fix the number of epochs (10), optimizer and the loss function, can we improve the CNN architecture to gain more accuracy?

One very impactful and simple concept is using residuals: <https://arxiv.org/pdf/1512.03385.pdf>
(<https://arxiv.org/pdf/1512.03385.pdf>)

The idea is simple: When stacking multiple layers, we sometimes add in the outputs from a few layers before to the output of a certain layer.

```
In [34]: 1 import requests
2 from IPython.display import Image, display
3
4 url = "https://miro.medium.com/v2/resize:fit:640/format:webp/1*D0F3UitQ2l5Q0Ak-tjEdJg.png"
5 response = requests.get(url)
6
7 display(Image(response.content))
8
```



5 points: Let's implement a 3-layer CNN with a residual connection.

The CNN should have 3 conv layers : conv1, conv2, conv3 and be implemented similar to MyCNN in 5.1. Now, add a residual connection from the output of the first conv1 layer to the output of the conv3 layer.

Hint 1: The output of the Conv1 has 32 channels - it might be beneficial to change that to 16 in this case so that it can be easily added to the output of the conv3 - which also has 16 channels. Even after this, the output shape of conv1

Hint 2: Note that the convolution layers reduce the height and width of the input. Hence, the output of the conv3 layer will be smaller

In [36]:

```
1 class ResidualCNN(nn.Module):
2     """
3     Feel free to experiment on CNN model.
4     You only need to report the model that has the best performance on the dataset.
5     """
6
7     def __init__(self):
8         super().__init__()
9         self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=5, padding="same")
10        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
11
12        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5, padding="same")
13        self.conv3 = nn.Conv2d(in_channels=32, out_channels=16, kernel_size=5, padding="same")
14
15        self.fc1 = nn.Linear(1024, 64)
16        self.fc2 = nn.Linear(64, 32)
17        self.fc3 = nn.Linear(32, 10)
18        self.relu = nn.ReLU()
19
20    def forward(self, x):
21        conv1_out = self.conv1(x)
22        x = self.relu(conv1_out)
23        x = self.pool(x)
24
25        x = self.conv2(x)
26        x = self.relu(x)
27        x = self.pool(x)
28
29        residual = self.pool(self.pool(conv1_out))
30
31        x = self.conv3(x)
32        x = x + residual
33        x = self.relu(x)
34
35        x = x.view(x.size(0), -1)
36        x = self.fc1(x)
37        x = self.relu(x)
38        x = self.fc2(x)
39        x = self.relu(x)
40        x = self.fc3(x)
41        return x
42
43
```



```

44 ## You need to:
45 # + Fill in the `MyBestCNN` class, use the best model you have
46 # + Show the training log (train/test loss and accuracy) of the best model (you should use `que
47 # + Discuss about your experiment
48
49 ### Initialize your model
50 cnn_ver2 = ResidualCNN()
51
52
53 #### Show the training log.
54 history_best_cnn = question_5_5_train_model(cnn_ver2, device, num_epochs, batch_size, trainloader

```

training 1 epoch...: 100%|██████████| 3125/3125 [00:42<00:00, 74.25it/s]

Epoch: 1 | Runtime: 1.17[m] | train_loss: 1.771 | train_acc: 35.338 | test_loss: 1.432 | test_acc: 48.820

training 1 epoch...: 100%|██████████| 3125/3125 [00:47<00:00, 65.62it/s]

Epoch: 2 | Runtime: 2.43[m] | train_loss: 1.299 | train_acc: 53.104 | test_loss: 1.198 | test_acc: 57.210

training 1 epoch...: 100%|██████████| 3125/3125 [00:48<00:00, 63.96it/s]

Epoch: 3 | Runtime: 3.70[m] | train_loss: 1.098 | train_acc: 60.944 | test_loss: 1.044 | test_acc: 63.400

training 1 epoch...: 100%|██████████| 3125/3125 [00:42<00:00, 73.48it/s]

Epoch: 4 | Runtime: 4.85[m] | train_loss: 0.970 | train_acc: 65.704 | test_loss: 0.984 | test_acc: 65.680

training 1 epoch...: 100%|██████████| 3125/3125 [00:44<00:00, 69.60it/s]

Epoch: 5 | Runtime: 6.07[m] | train_loss: 0.882 | train_acc: 68.822 | test_loss: 0.934 | test_acc: 67.320

training 1 epoch...: 100%|██████████| 3125/3125 [00:43<00:00, 72.24it/s]

Epoch: 6 | Runtime: 7.24[m] | train_loss: 0.810 | train_acc: 71.370 | test_loss: 0.917 | test_acc: 67.880

training 1 epoch...: 100%|██████████| 3125/3125 [00:42<00:00, 73.63it/s]

Epoch: 7 | Runtime: 8.39[m] | train_loss: 0.753 | train_acc: 73.696 | test_loss: 0.882 | test_acc: 69.690

training 1 epoch...: 100%|██████████| 3125/3125 [00:43<00:00, 71.89it/s]

Epoch: 8 | Runtime: 9.55[m] | train_loss: 0.705 | train_acc: 75.378 | test_loss: 0.837 | test_acc: 70.860

training 1 epoch...: 100%|██████████| 3125/3125 [00:49<00:00, 62.73it/s]

Epoch: 9 | Runtime: 10.88[m] | train_loss: 0.661 | train_acc: 76.684 | test_loss: 0.840 | test_acc: 70.670

training 1 epoch...: 100%|██████████| 3125/3125 [01:31<00:00, 34.32it/s]

Epoch: 10 | Runtime: 12.95[m] | train_loss: 0.620 | train_acc: 78.160 | test_loss: 0.842 | test_acc: 71.050

5 points: Short Answer: Compare to network without the residual connections

Make the same network below and remove the residual connection. Compare the performance.

Feel free to change the number of layers of the CNN. Does the residual connection help more with more layers? What about changing the number of residual connections?

This is an open-ended exploratory answer.

In [37]:

```

1 class NonResidualCNN(nn.Module):
2     def __init__(self):
3         super().__init__()
4         self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=5, padding="same")
5         self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
6
7         self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5, padding="same")
8         self.conv3 = nn.Conv2d(in_channels=32, out_channels=16, kernel_size=5, padding="same")
9
10        self.fc1 = nn.Linear(1024, 64)
11        self.fc2 = nn.Linear(64, 32)
12        self.fc3 = nn.Linear(32, 10)
13        self.relu = nn.ReLU()
14
15        def forward(self, x):
16            x = self.conv1(x)
17            x = self.relu(x)
18            x = self.pool(x)
19
20            x = self.conv2(x)
21            x = self.relu(x)
22            x = self.pool(x)
23
24            x = self.conv3(x)
25            x = self.relu(x)
26
27            x = x.view(x.size(0), -1)
28            x = self.fc1(x)
29            x = self.relu(x)
30            x = self.fc2(x)
31            x = self.relu(x)
32            x = self.fc3(x)
33            return x
34
35 cnn_ver3 = NonResidualCNN()
36
37
38 ##### Show the training log.
39 history_best_cnn = question_5_5_train_model(cnn_ver3, device, num_epochs, batch_size, trainloader

```

training 1 epoch...: 100%|██████████| 3125/3125 [00:42<00:00, 73.44it/s]

```
Epoch: 1 | Runtime: 1.20[m] | train_loss: 1.801 | train_acc: 33.556 | test_loss: 1.432 | test_acc: 48.360  
training 1 epoch...: 100%|██████████| 3125/3125 [00:42<00:00, 73.93it/s]  
Epoch: 2 | Runtime: 2.36[m] | train_loss: 1.322 | train_acc: 52.048 | test_loss: 1.175 | test_acc: 58.120  
training 1 epoch...: 100%|██████████| 3125/3125 [00:42<00:00, 73.18it/s]  
Epoch: 3 | Runtime: 3.53[m] | train_loss: 1.117 | train_acc: 60.136 | test_loss: 1.071 | test_acc: 62.340  
training 1 epoch...: 100%|██████████| 3125/3125 [00:54<00:00, 57.44it/s]  
Epoch: 4 | Runtime: 4.91[m] | train_loss: 0.988 | train_acc: 64.942 | test_loss: 0.992 | test_acc: 65.200  
training 1 epoch...: 100%|██████████| 3125/3125 [00:45<00:00, 67.99it/s]  
Epoch: 5 | Runtime: 6.14[m] | train_loss: 0.893 | train_acc: 68.186 | test_loss: 0.967 | test_acc: 66.170  
training 1 epoch...: 100%|██████████| 3125/3125 [00:52<00:00, 60.00it/s]  
Epoch: 6 | Runtime: 7.47[m] | train_loss: 0.818 | train_acc: 70.858 | test_loss: 0.860 | test_acc: 70.010  
training 1 epoch...: 100%|██████████| 3125/3125 [02:11<00:00, 23.80it/s]  
Epoch: 7 | Runtime: 10.19[m] | train_loss: 0.757 | train_acc: 73.344 | test_loss: 0.837 | test_acc: 71.230  
training 1 epoch...: 100%|██████████| 3125/3125 [00:40<00:00, 76.46it/s]  
Epoch: 8 | Runtime: 11.31[m] | train_loss: 0.709 | train_acc: 74.858 | test_loss: 0.807 | test_acc: 72.050  
training 1 epoch...: 100%|██████████| 3125/3125 [00:41<00:00, 75.95it/s]  
Epoch: 9 | Runtime: 12.44[m] | train_loss: 0.666 | train_acc: 76.458 | test_loss: 0.802 | test_acc: 72.570  
training 1 epoch...: 100%|██████████| 3125/3125 [00:44<00:00, 70.62it/s]
```

Epoch: 10 | Runtime: 13.62[m] | train_loss: 0.622 | train_acc: 77.930 | test_loss: 0.851 | test_acc: 71.570

Your discussion

Comparing the CNN models with and without residual connections does not reveal any interesting performance differences. More specifically, the regular CNN without residual connections achieves a slightly higher test accuracy 72.57% vs 71.05%. This may be due to the fact that we did not modify the network (we maintained 3-layers), but we only removed the residual stream. Something to comment on is that while the residual network shows slightly better performance in the early epochs, the standard CNN catches up around epoch 6 and eventually surpasses it. However, testing with more layers would likely show the benefits from residual connections. Like before, it's worth noting that I ran both models locally without using any GPUs, so slower overall training time was expected.

Congrats! You have reached to the end of Pset4