Computer Science 481/581

Final Project (50 points)
Project Check-in Due Week 9 (March 3-7)
Report Due Friday, March 14th at 10:00pm
Evaluation Submissions Due Monday, March 17th at 10:00pm

Read all of the instructions. Late work will not be accepted.

Overview

In the final project you will use your deep learning skills to address a set of real world problems. You will work individually or in an assigned group up to 5 students on the assignment. If your group has N members, you must select N distinct tasks to submit (you do not need to decide which tasks until you submit your report). The tasks to choose from are:

1. Speaker Verification
2. Dialect Identification
3. Grapheme to Phoneme Conversion
4. Fake Image Detection
5. Weather Localization in Time

Details of each of these tasks can be found at the end of this document, under "Task Details." See the "System Evaluation" section for some grade implications of your group size.

Collaboration Policy

Each group member will serve as the lead on one of the N tasks, where N is the number of group members. Your grade is based primarily on *your task* (the approach taken, the analyses conducted, performance on the task). Ultimately, you can work as closely or independently with your fellow group members as you like. To encourage collaboration within groups, and incentivize supporting your fellow group members, there is opportunity for extra credit based on the performance of others in your group. Within a group, you are welcome to share code, ideas, models, etc.

In contrast, you cannot collaborate across groups. You may disclose to members of other groups the performance of your models on the dev set, but **cannot** share any other information, including models, algorithms, outputs, strategies, etc.

Here are some of the pros of larger vs smaller group sizes:

- Benefits of larger groups:
  - More opportunities for collaboration and learning
  - More people who can help you
  - Increases the expected extra credit bonus (see "System Evaluation" below)
- Benefits of smaller groups:
  - Your have more control over which task you lead
  - You have more flexibility to change tasks if you find one is not working out

Language and Toolkits

Your solutions should use PyTorch v2. You may use any of the functionality of this version of PyTorch, including included architectures. You must not use or plagiarize any third party PyTorch code, although you may implement models described in the literature (i.e. academic papers) based only on descriptions in said papers: if you want to use any deep learning functionality outside of the `pytorch`, `torchvision`, or `torchaudio` libraries, you must implement it or get explicit permission from the instructor. Even if they are distributed with PyTorch, you may not use *pretrained models* (i.e., whose weights have been trained on a separate dataset), since that would violate the requirements described in the following section.

Data

Training and development data for the tasks will be provided to you in relatively raw/original format. While you may use data augmentation to create new datapoints from the provided training set, you **must not** train on any new data outside of the provided training set, including the dev set data, data you create or new data obtained from any source. **You must not distribute any of the provided data. For the datasets requiring it, WWU has purchased the appropriate licenses allowing use by WWU students. However, you risk legal liability if you distribute this data.**

Experiments and Tuning

You will want to extensively experiment with different models and hyperparameters. The best "easy" way to tune is with random search. With random search, you randomly draw your hyperparameters (e.g. uniformly among the valid values) each time. Beyond that, there are hyperparameter tuning tools (I recommend Weights & Biases) that may help you find even better hyperparameters faster, though there is a learning curve for those tools that might not pay off for this particular assignment.

Computing Resources

At any given time, each *group* may have

- x GPU compute job running on the cluster, where x is
  - 0 if your are working individually,
  - 1 if your group size is 2 or 3
  - 2 size is 4 or 5
- jobs using a total of up 32 CPU threads on the cluster

You must coordinate with your group members to ensure you do not exceed these limits. In addition, at any given time, each *individual* may have

- up to four GPU compute jobs running on lab machines, but they must be submitted through condor.

Information will be provided on how to run compute jobs via condor. While you can develop and debug code on other machines, **you must not perform sustained compute on any resources beyond these, including but not limited to resources that you have**

access to through a research group, your own personal computational resources, or cloud compute. This policy is to ensure that having the means to access additional computing resources does not offer an unfair advantage.

Project Check-in (5 pts)

During the project check-in window, you should drop in to one of my office hours for a brief check-in. You should be prepared to discuss the following:

- the methods you have tried,
- how and what you have been hyperparameter tuning,
- the best performance you have obtained on dev,
- the configuration that produced the best results so far, and
- what else you plan to try next for the task before the report deadline.

This is a chance for you to get feedback on your progress and plans, and to make sure you are on track to complete the project successfully.

Final Report (30 pts)

Each group will need to submit one final report per task. These reports must be typeset, using 1 inch margins, size 11 font, single spacing, and one of the following fonts: Computer Modern, Times New Roman, Calibri, or Arial. Each report can be *up to* two pages, including all prose, figures, tables and any references you may choose to include. Each report should contain the following information, clearly labeled:

- **Methods**: This section should address, potentially in subsubsections, things like the following
  - Data Preprocessing / Feature Extraction
  - Models Developed
  - Training and Tuning
  - Baselines

  This section should clearly and explicitly specify your submission model; i.e., the model you will use to make predictions on the test set. Please include which file(s) in your repo implement this model, but also describe it in prose and/or equations and figures.

- **Results**. This section should include (in tabular or figure form) the most important results of your experiments. How do different models compare to each other and to your baseline(s)? What effect do the hyperparameters have (if any)? Tables and figures should also be accompanying by prose highlighting key findings, offering plausible explanations for observed trends, etc.

- **Conclusions**. What did you learn through your experiments? Given the various models and hyperparameter configurations discussed in the previous section, here you should clearly describe the model configuration you choose as your "submission model," and justify that decision.

- **Contributions**. Who was the lead on this task? Briefly but clearly describe here what role different team members had in this section. Examples of how student might contribute include, but are not limited to: literature search, model development, running experiments, visualizing and analyzing results, and writing. Not all team members need to contribute to all tasks, but all contributions must be acknowledged.

Two pages per report is a maximum length - you are not obligated to reach the maximum length. Your priority should be making your report clear, organized and detailed. Tables and figures are encouraged, and your prose should read like prose, not like a bulleted list. Good reports with no problems will earn 90% of the report grade. Scores of 91%-100% are reserved for exceptional reports. Please submit your reports, one per task completed by your group, as pdfs titled `task1.pdf`, `task2.pdf`,, `task3.pdf`, `task4.pdf`, or `task5.pdf` for tasks 1, 2, 3, 4 or 5, respectively.

Report Grading

The report grades are broken down into

- 12 pts: Methods
- 12 pts: Analysis
- 6 pts: Report Quality

Your report grade will be based on the task you led on.

Methods

This category will be judged based on the description in your report of the methods you tried. Your grade here will be determined by factors such as:

- Do you present good justification for the models and training algorithms you used?
- Were you using the models appropriately?
- Did you tune your models appropriately?

Analysis

This category will be judged based on the description in your report of your experiments and results. Your grade here will be determined by factors such as:

- Does your experimental setup allow a fair comparison between models and baseline(s)?
- Do you present your results clearly?
- Does your prose regarding the figures demonstrate an understanding of the findings and offer plausible reasons for the observed trends?

Writeup Quality

This category refers specifically to the clarity and professionalism of your written report. Your grade here will be determined by factors such as:

- Is your writing clear and concise?
- Is the report free from typos and grammatical errors?

- Is notation consistent? Are all (non-standard) symbols or acronyms defined?
- Are the figures clear and easy to interpret (appropriated labeled, etc.)?
- Is the general appearance professional (properly typeset mathematics, etc.)?

System Evaluation (15 pts)

After the report is submitted, test data will released for all tasks (you will be given the input features but not the labels). You will have a fixed window of time to push this data through your system and submit the results in the specified format. Your system's predictions will be scored and compared to your peers' submissions (i.e., from the other groups). Your grade will be determined (in part) by your performance on this evaluation. The format of your prediction submissions and the particular performance metrics depend on the task – see the "Task Details" section below.

During the evaluation, for each task, every submission will be scored, producing a set of scores $s_1, s_2, \ldots, s_K$, where K is the number of submissions. First, I will compute the mean μ and standard deviation σ for each task. Then each score s_i is converted to an adjusted score $\hat{s}_i$ as follows. First, the scores are standardized

$$z_i = \frac{s_i - \mu}{\sigma}$$

Second, the standardized score z_i is linearly scaled and passed through a logistic sigmoid.

$$\hat{s}_i = \sigma(\ln(3) z_i + \ln(3))$$

Your grade for System Performance will be your $\hat{s}$ times the points possible, i.e.:

$$\lceil 15 \hat{s}_i \rceil$$

This assumes higher score is better; if the performance metrics is some sort of error, then I will flip the sign on the zs before computing $\hat{s}_i$. The constants in the equation were picked so that an average score $z_i = 0$ will earn 75% of the available points. One standard deviation below the average nets you 50% of the points, and one standard deviation above nets you 90%. To earn 15/15, you need to be 1.4 standard deviations above the mean. The above process of mapping a raw score to the points will happen independently for each of the tasks.

Extra Credit: the above calculation gives the "raw" system evaluation points for each student. To incentivize collaboration within groups, your final system evaluation points will be your raw points plus 10% of the raw system evaluation points of your fellow group member (excluding yourself) with the highest raw system evaluation points. To maximize expected extra credit, you will want to help your group members do as well as possible on their tasks.

Submitting Your Work

You will use an official final project group github repository (I will create this for you). The repository should have two directories: `code` and `deliverables`. By the report deadline, the `deliverables` directory should contain one report pdf per task, named by the task number

(`task1.pdf`, `task2.pdf`, etc., as described earlier in the document). Lastly, by the evaluation deadline, the `deliverables` subdirectory should also contain up to N additional files: the predictions for the submitted tasks (see Task Details below for information about the file formats and naming schemes). All of the code and configs used to train and evaluate your models should be included in the `code` subdirectory. You can organize your code and configs however you want within that `code` subdirectory. Please use this repository throughout the development process. **Do NOT add, commit or push data to your github repository.**

Academic Honesty

Below are some tips to help clarify what you can and cannot do for the final project. When in doubt, check with me. I am available for help during the dedicated project hours, during office hours, and on Piazza (do not wait until the last minute to reach out). If you participate in academic dishonesty, you will fail the course.

You can

- Discard some of the training data points
- Manufacture new data points *from the provided data points* (e.g. by adding random noise, doing transformations, etc.)
- Use code you have written for this class
- Use built-in models in PyTorch (as long as they are not pretrained)
- Implement new models in PyTorch (not plagiarizing from any third party source)
- Heavily tune your models to find the optimal hyperparameters
- Use deep learning algorithms that were not covered in this class
- Invent your own deep learning models and use them
- Share code, tips, models, with members of your own group
- Run compute as described in "Computing Resources" above.
- Tell other groups what scores you are getting on different tasks (without further details).

You cannot

- Obtain and use any new training data not directly derived from the **training** data files provided to you (e.g., by searching the web, making synthetic data, buying corpora, training on the dev set, etc.)
- Exchange any models with any another group/team
- Exchange any test set predictions with any other group/team
- Provide any other group/team details on your approach
- Distribute the models, data or predictions to anyone not enrolled in our class
- Obtain models, data or predictions from anyone not enrolled in our class (recall that from our Syllabus, ChatGPT, Copilot and other LLM/AI tools are considered "people" outside of our class for academic honesty purposes).
- Use computing resources not explicitly authorized in "Computing Resources" above.
- Run your compute jobs in a way that interferes with the normal operation of department machines (e.g., excessive I/O straining the file server).

Task Details

Task 1: Speaker Verification

For this task you will train a model to take as input two audio (wav) files, each containing speech from a single speaker, and output the probability that they are the same speaker. For each of train and dev, I am providing a set of wav file and their corresponding speaker: each `I` present in the data, `fileI.wav` is the input and `fileI.spkid.txt` is a plaintext file containing the corresponding speaker id. We do not care about the actual ID values directly: they are only provided so that you can create your own positive examples (where both files are from the same speaker) and negative examples (where the files are from different speakers). How you craft your training batches is up to you. You can ignore the `fileI.dialect.txt` files, which are actually for Task 2.

Input File Format

The audio files are in `wavs/train` and `wavs/dev`, in RIFF wave format. I strongly recommend using the functionality in the `torchaudio` library for processing the wav files. Two common strategies for representing audio are:

1. Extracting log mel filterbank features for every frame, where frames are overlapping 25ms slices of audio, each starting 10ms after the previous one. The 25ms/10ms magic numbers are common, but you could explore other choices. This yields data that has dimension # frames by # mel filterbank bins (e.g., 40). This considers the file to be a time series, with $D = 40$ features per timestep.

2. Explicitly converting the audio file into a spectrogram "image." In many ways, a spectrogram is similar to the first option above, but instead treating the # frames by # mel filterbank bins as a 2d image. This makes it easy to apply image processing techniques to the wav. Note that each file will be a different length, leading to different sized images. There are pooling strategies for turning a variable length image into a fixed length one; padding or truncating the images to a fixed length is another strategy to consider.

Evaluation Protocol

At the start of the evaluation period, you will be given

1. a `test` directory containing a set of `wav` files (no labels)
2. a plaintext "script" file, where each line contains a pair of test file names separated by a space; e.g.,

   ```
   test/file2535.wav test/file5277.wav
   test/file992.wav test/file2513.wav
   ```

   etc.

If there are N lines in the script file, your predictions should be a vector in $\mathbb{R}^N$, where the ith element is the probability that the pair of files on line i contain the same speaker. This should be saved as numpy file with type `np.float32` named `task1_predictions.npy` in the `deliverables` directory of your github repository.

The overall performance metric is negative cross entropy computed using your model's probabilities and the ground truth labels.

Task 2: Dialect Identification

This task is a multiclass classification problem in which a single audio (wav) file containing a single English speaker is mapped to one of eight English dialects. The labels for this task are actually distributed with the Task 1 data (see the `fileI.dialect.txt` files).

Input File Format

See Task 1's "Input File Format" section.

Evaluation Protocol

At the start of the evaluation period, you will be given

1. a `test` directory containing a set of `wav` files (no labels)
2. a plaintext "script" file, where each line contains a single test file name; e.g.,

   ```
   test/file2535.wav
   test/file992.wav
   test/file2513.wav
   ```

   etc.

If there are N lines in the script file, your predictions should be a matrix in $\mathbb{R}^{N\times 8}$, where the ith row gives the probabilities your model assigns dialects, $0, 1, 2, \ldots, 7$. This should be saved as numpy file with type `np.float32` named `task2_predictions.npy` in the `deliverables` directory of your github repository.

For each of the N files in the script, I will compute pick the class that your models assigns the highest probability as your prediction. The overall performance metric is the cross-entropy loss on the test set.

Task 3: Grapheme to Phoneme (g2p) Conversion

For this task you will train a model that takes as input a sequence of graphemes (essentially, characters) and outputs its pronunciation as a sequence of phonemes (speech sounds).

Input File Format

The data format is fairly simple for this one: (a) train and dev are each single files, (b) each line in the file is an input-output pair (i.e. a single datapoint); for example

```
FLOURISH  F L ER1 IH0 SH
STAUDINGER  S T AW1 D IH0 NG ER0
LIBRARIES  L AY1 B R EH0 R IY2 Z
HOLDA  HH OW1 L D AH0
KETTERMAN  K EH1 T ER0 M AH0 N
```

The first line in this example contains an eight grapheme sequence (the characters of the word "flourish") then two spaces, and then five space-delimited phonemes. Our data uses the "ARPABET" phonetic dictionary. Each vowel is also marked with a lexical stress level, where 0 denotes no stress, 1 denotes primary stress and 2 denotes secondary stress. (Lexical stress indicates the relative emphasis of the syllable(s) of the word.) Including the different stress levels, this yields 69 distinct phoneme symbols.

Evaluation Protocol

At the start of the evaluation period, you will be given a test file, which contains only the grapheme sequences; e.g.:

```
HUTCHINSON
WESTERN
```

You will then use your model to predict pronunciations for each word, and create a predictions file with the same format as the training and dev files; namely, your predictions file should have the provided grapheme sequence, then two spaces, and then space-delimited phonemes out of the phoneme set. For example:

```
HUTCHINSON  HH AH1 CH IH0 N S AH0 N
WESTERN  W EH1 S T ER0 N
```

Your predictions file should be saved as `task3_predictions.txt` and pushed to your `deliverables` directory. For each predicted phoneme sequence, I will compute phoneme error rate (PER; i.e. word error rate applied to phoneme sequences). You may notice that the same grapheme sequence can yields multiple, distinct pronunciations (think "to record" vs "a record"). For any such test set words, I will compute the PER against each of the valid pronunciations, and take the minimum. The overall performance metric is the average PER over all test set datapoints.

Task 4: Fake Image Detector

As we saw in M7, generative models have become capable of generating highly realistic images, including those of human faces. These generated photos can be used in nefarious ways, including as profile pictures for fake social media accounts used to spread disinformation. Your task here is to train a classifier that takes as input an image of a face and output the probability of it being a fake image.

Input File Format

The images are `256x256` color images, saved in `png` format, and split between the following four directories: `train/real`, `train/fake`, `dev/real`, and `dev/fake`. I encourage you to use routines found within the `torchvision` library to load and perform any needed image processing.

Evaluation Protocol

At the start of the evaluation period, you will be given

1. a `test` directory containing a set of `png` files (obviously not divided into real and fake)
2. a plaintext "script" file, where each line contains a single test file name; e.g.

   ```
   test/828YC5ZWE2.png
   test/G4CAT4CAVV.png
   test/N3HOZX6AXW.png
   test/TIAJXH112H.png
   ```

   etc.

If there are N lines in the script file, your predictions should be a vector in $\mathbb{R}^N$, where the ith element gives the probabilities your model assigns to the image being *fake*. This should be saved as numpy file with type `np.float32` named `task4_predictions.npy` in the `deliverables` directory of your github repository.

Task 5: Weather Localization in Time

The goal of this task is to train a model capable of taking an 8 day "week" and predicting what day of the year the first of those days is (in the set $\{0, 1, 2, \ldots, 364\}$[1]). You will be provided with a number of training and dev set files. Each file will be a `npy` file containing numpy ndarray of dimensions `(365,128,256)`, where the three dimensions correspond day of year, latitude/height and longitude/width, respectively. Each element of this ndarray is the daily mean temperature for the given day and spatial location. (The spatial grid maps to the entire globe.)

Input File Format

The `train` and `dev` directories each contain a set of single year `npy` files, as described above.

Evaluation Protocol

At the start of the evaluation period, you will be given

1. a `test` directory containing a set of 8-day long `npy` files (containing ndarrays of shape `(8,128,256)`).
2. a plaintext "script" file, where each line contains a single test file name.

If there are N lines in the script file, your predictions should be a vector in $\mathbb{N}^N$, where the ith element is an integer in $\{0, 1, 2, \ldots, 364\}$ listing your predicted start day of the 8-day sequence. This should be saved as numpy file with type `np.int64` named `task5_predictions.npy` in the `deliverables` directory of your github repository.

For each prediction, I will compute the absolute distance (in days) between your predicted day and the true start day, taking into account wrap-around[2]. The overall performance metric is the average of these absolute differences over all test datapoints.

[1] For simplicity, we ignore leap year.

[2] That is, if the truth is day 353 and you predict day 0, you are only 12 days off, not 353 days off