Opened: Monday, 25 March 2024, 12:00 AM
Due: Friday, 12 April 2024, 11:59 PM

This assignment is worth 10% of your overall grade.

Your task in this assignment is to do classification on a small dataset using a Convolutional Neural Network (CNN). You are provided with code to load the dataset and perform training and validation. You only need to define the CNN architecture and optionally tune hyperparameters to train it.

In addition to classification accuracy, your classifier will also be evaluated on the object detection task using a sliding window detector that will run your classifier on a large number of patches from an image containing two objects and then select the two highest confidence patches as the object locations.

Datasets

You will be working with two datasets in this assignment – one for classification and one for sliding window detection.

NotMNIST-RGB:

This is a customized RGB version of the NotMNIST dataset. The original dataset contains 28x28 grayscale images showing ten letters of the alphabet from A to J in various fonts. This has been modified to add a random background and foreground colour to each image. There are **12000** images in the train set and **6700** in the test set.

The **test set is not released** to simulate real world conditions where trained models need to be deployed to work on previously unseen data and therefore must be trained to avoid overfitting to the training data.

There is no separate validation set so the provided code allows you to split the train set into training and validation subsets with a customizable ratio.

Example images from the dataset (both original and RGB) with corresponding labels:

To reiterate, you will only be working on the RGB version of the dataset.
Visualizations of all the images in the training dataset can be found here.
Images from the original NotMNIST dataset that were used for generating the RGB images are also provided.

NotMNIST-DL:

This is a simple object detection dataset constructed by placing two random images from the NotMNIST RGB dataset at random locations on a black background within a 64 x 64 image.

In order to make the detection task even easier, following two constraints are also observed:

1. the two images are always of **different letters** so that the same letter is never repeated twice in the same image
2. the two images **never have an overlap > 0.1** in terms of intersection over union (IOU)

Sliding window detection tends to be slow so the test set has only **500** images constructed using 1000 random images from the NotMNIST-RGB test set.

We also provide a training set with 6000 images constructed using the entire NotMNIST-RGB training set though you cannot directly use it for training.

In order to do that, you need to extract 28x28 patches from these images as explained in the next section.

As with NotMNIST RGB, there is no separate validation set and the test is **not released**.
Example images from NotMNIST-DL:

Task:

You are provided with three python files: **A6_main.py**, **A6_utils.py** and **A6_submission.py**.

You only need to add code / change parameters in A6_submission.py.

You are free to make any changes to A6_main.py you want to train your model (or even use your own training code) but make sure that the submitted A6_submission.py **can work with the original A6_main.py** with parameters **ckpt.load=2** and **test.enable=1** since that is how it will be evaluated.
We will **not** be training your model so it does not matter how you train it. The provided training code is only for your convenience and there is no requirement to use it.
To reiterate, you will only be submitting A6_submission.py and any changes you make to the other two files for your own use will not be available during evaluation so please make sure that your A6_submission.py works with the original A6_main.py and A6_utils.py.

You are also provided with a notebook **A6_run.ipynb** that demonstrates how to setup python environment and run A6_main.py in several different configurations, how to use tensorboard for monitoring training progress and how your submission will be evaluated.
If you would like to work on the assignment in a notebook. you can create one and copy all the code from the three python files into cells but **make sure that you only submit A6_submission.py**.
To reiterate, you need to submit the python file A6_submission.py rather than a notebook for this assignment.

Classification

You need to complete three methods of the **Classifier** class in A6_submission.py:

1. **__init__** : define the CNN architecture including number and configurations of layers and their connectivity

2. **init_weights**: optionally initialize the weights of your CNN using a custom distribution (or load pretrained weights)

3. **forward**: propagate the input x of size batch_size x 3 x 28 x 28 through your network to produce an output of size batch_size x 11 containing the probabilities of the images to contain the 10 letters and the background.
    - Each row in this array contains the probabilities for one image and each of the first 10 columns corresponds to one of the letters from A to J while the last column is for the background.
    - For example, the entry at index [3, 7] is the probability that the fourth image is of letter H.
    - All entries in each row should sum to 1
    - The letter with the maximum probability for each image is considered to be the class for that image

In addition to the above functions, you can change any of the parameters defined in the **Params** class to modify the behaviour of classifier training and sliding window detection.

You can also add any other functions or classes to this file or modify it in any way you want as long as it still works with the original A6_main.py.

Sliding Window Detection

This video shows how the sliding window detector works.

NotMNIST-DL image is on the left, with the two ground truth boxes in green and the sliding window in red. The corresponding patch image (that gets passed through the classifier) is on the right and output probabilities (in percentage) for all letters and background are at the top (only letters with probs > 0.1% are shown). All possible sliding window patches using a stride of 1 are shown. The classifier was trained only on NotMNIST-RGB so it is not very good at classifying letters that are either occluded or not centred on the patch.

There is a fourth function in **Classifier** named **get_best_patch_ids** which is used by the sliding window detector. It takes the output of the **forward** function over all patches in a NotMNIST-DL image and returns the IDs of the two patches that are most likely to contain the two letters in that image. By default, it selects the two patches with highest classification confidence that are classified as different letters while ignoring patches classified as background.

This technique is known to work quite well in our experiments but you can also optionally modify this if you would like to use some other method to select the two best patches.

Note that the patch order will be randomized during evaluation (**swd.shuffle=1**) to prevent you from combining the patches to reconstruct the original image and run another object detector on it as a means to bypass the sliding window detector.

In order for your classifier to work well at the sliding window detection task, it must be able to distinguish between foreground and background patches as well as between foreground patches with different degrees of object "centeredness". Training it only on the NotMNIST-RGB dataset will probably not work very well since your model never sees images of the background or images with partially visible letters during training so it will end up misclassifying them during the detection task.

The provided code allows you to extract sliding window paches from NotMNIST-DL images, each with probabilistic labels, by setting **swd.extract_patches=1**. The number of random patches extracted per image is specified by **swd.patches_per_img**. The probabilistic labels for each patch are generated by computing the area of intersection of that patch with the two letters in the NotMNIST-DL image.

This video shows the probability generation process:

NotMNIST-DL image is on the left with the two GT boxes in green and the patch box in red. The corresponding patch image is on the right and probabilities (in percentage) for the two digits and the background are at the top. Higher the intersection between the patch and a letter, higher will be the probability of that letter.

The patch images and probabilities are saved in an npz file which can be loaded alongside the NotMNIST-RGB dataset to train on the combined set of images by setting its path in **train.probs_data**
If you would like to train only on these patches and ignore the original NotMNIST-RGB training set, you can set **train.data=''**
There are also several parameters in **Params.SlidingWindowDetector** that you can optionally use to customize the performance of the sliding window detector:
Details of these parameters and how they affect the detection performance are in the docstring.

Evaluation:

Your marks will depend on both classification and detection performance of your model.

Classification performance is measured by the percentage of images in the NotMNIST-RGB test set that are classified correctly.

Detection performance is measured by two metrics – accuracy and IOU.Detection accuracy is the percentage of the 1000 letters in the NotMNIST-DL test set that are classified correctly and detection IOU is the average IOU between the corresponding bounding boxes and the ground truth.

You will get separate marks for classification and detection performance and your overall marks will be a weighted average of the two, with 70% weightage given to classification and 30% to detection.

Therefore, overall marks = 0.7*classification_marks + 0.3*detection_marks

Classification

Marks will scale linearly from **50% to 100%** for classification accuracies ranging from **80% - 95%**.

A submission with accuracy < 80% will get no marks and one with accuracy >= 95% will get full marks.
All intermedate accuracies will be marked by linear scaling with a base mark of 50%.
For example,

- **85%** accuracy gets 50 + (100 - 50) * (85 - 80) / (95 - 80) = **66.67%** marks.
- **90%** accuracy gets 50 + (100 - 50) * (90 - 80) / (95 - 80) = **83.33%** marks

Detection

You will get separate marks for detection accuracy and IOU and your overall detection marks will be the average of the two. Both marks will scale linearly from **50% to 100%** for detection accuracy / IOU ranging from **50% - 80%**.

A submission with accuracy / IOU < 50% will get no marks and one with accuracy / IOU >= 80% will get full marks.
All intermedate accuracies will be marked by linear scaling with a base mark of 50%.
For example,

- **75%** detection accuracy gets 50 + (100 - 50) * (75- 50) / (80- 50) = **91.67%** marks
- **60%** detection IOU gets 50 + (100 - 50) * (60- 50) / (80- 50) = **66.67%** marks

Your detection marks will then be (91.67+66.67)/2 = **79.17%**

With classification marks = **66.67%** and detection marks = **79.17%**, your overall marks would be **0.7*66.67 + 0.3*79.17** = **70.42%.**

Note that detection evaluation will be **order-agnostic** so that the specific order in which you detect the two boxes in each image does not matter. Both detected boxes will be compared with each of the two ground truth boxes and the matching that gives you higher score will be chosen.

Constraints

You can use any (new or existing) CNN architecture you want. One of the objectives of this assignment is for you to learn how to adapt existing models on the Internet to solve a new problem so there is no restriction or guideline as to which CNN you should use.

The only limitation is that it must be able to run on Colab GPU at a speed of >= 200 images/sec and >= 2 images/sec on the NotMNIST-RGB and NotMNIST-DL test sets respectively.

Another practical constraint on the size of your model is the upload limit of **400 MB** on eclass so your network **must be small enough for its checkpoint to fit in this limit**. Requests for supplying checkpoints through other means like Google Drive will **not** be entertained. This datasets are far too simple for such a large network to be needed anyway. Even networks with checkpoints < 10 MB should be enough.

Any submissions failing these criteria will get **no marks** for this assignment.

There are no marks for your training procedure either. You marks will depend entirely on how your submitted model performs on the test sets.

What to Submit:

The provided code saves a checkpoint whenever some criterion of model improvement is satisfied (e.g. decrease in training loss or increase in validation accuracy) which can be set in **ckpt.save_criteria**. Each checkpoint is suffixed by its epoch so there might be multiple checkpoints in the checkpoints folder after a training session. You need to submit **only the best checkpoint** you find along with A6_submission.py.

Including multiple checkpoints will not be penalized in itself but please be aware that A6_main.py automatically loads the **latest** checkpoint in the **checkpoints** folder regardless of whether it is the best one or the one you intended. Any requests to test on a different checkpoint where multiple checkpoints have been submitted will be heavily penalized.

Create a folder named <first_name>_<last_name>_<student_id> (e.g. John_Doe_1234567) and place A6_submission.py along with the **checkpoints** directory inside this folder and zip it. **Submit only this zip file.**

To reiterate, the zip file should have the John_Doe_1234567 folder in its root rather than A6_submission.py or the checkpoints folder.

A sample submission with dummy files is available here to demonstrate the correct process.

Please do not include any other files or folders and do not rename the included python file or the checkpoints folder in any way (including changing their case). Also, make sure to submit a zip file rather than any other archive format (like rar, tar or 7z).

Your submission will be used in an automated evaluation system which will only work if these instructions are followed strictly. Any submission that fails to do so will be severely penalized and might even get no marks.

Resources

A simple search of "image classification with pytorch" will let you find many online tutorials on how you can get started with this assignment. it should likewise be easy to find tutorials on the related MNIST dataset and maybe even on notMNIST. so you should even be able to use the network architectures from there virtually unchanged while only adapting for the extra channels in the RGB version

This is the official pytorch tutorial and following are some others that show up first on Google:

General:

- Basics of Image Classification with PyTorch
- How to Train an Image Classifier in PyTorch and use it to Perform Basic Inference on Single Images
- Image Classification with PyTorch
  Collection of related tutorials.

MNIST:

- MNIST Digit Classification In Pytorch
- Handwritten Digit Recognition Using PyTorch -- Intro To Neural Networks
- MNIST Handwritten Digit Recognition in PyTorch
- Deep Learning with PyTorch: Image Classification using Neural Networks

Add submission

Submission status

Attempt number	This is attempt 1 (1 attempts allowed).
Submission status	No submissions have been made yet
Grading status	Not graded
Time remaining	Assignment is overdue by: 22 hours 1 min
Last modified	-

Submission comments	▸ Comments (0)