

# Project 5: Recognition using Deep Networks

- Due Mar 27 by 11:59pm
- Points 30

## Overview

This project is about learning how to build, train, analyze, and modify a deep network for a recognition task. We will be using the MNIST digit recognition data set, primarily because it is simple enough to build and train a network without a GPU, but also because it is challenging enough to provide a good example of what deep networks can do.


As this is the last defined project. The last task of this project is to propose and design your final project. I would suggest thinking about this earlier than later and discussing it with me or a TA in office hours.


## Tasks

### 1. Build and train a network to recognize digits

The first task is to build and train a network to do digit recognition using the MNIST data base, then save the network to a file so it can be re-used for the later tasks. My strong recommendation is to use the python package pyTorch (torch) and the associated torchvision package. You can install both torch and torchvision using pip.

[PyTorch Home Page \(pytorch.org\)](https://pytorch.org)

There are a number of different tutorials on building and training a convolutional network to solve the MNIST digit recognition task. You can follow the [pyTorch Tutorial](#) 

[\(https://pytorch.org/tutorials/beginner/basics/intro.html\)](https://pytorch.org/tutorials/beginner/basics/intro.html) for the Fashion MNIST data set, but use the MNIST digit data set instead. A different tutorial I like that uses the MNIST digits is [here](#).   
 [\(https://nextjournal.com/gkoehler/pytorch-mnist\)](https://nextjournal.com/gkoehler/pytorch-mnist)

As you write your python code, write well-structured python code. **All** of your code should be in functions except for the top level function call which should be conditioned on whether the file is being executed or imported. Please use the following overall structure. This will be part of the grading. Please do not use a Jupyter notebook for this assignment (there will be a penalty).

```
# Your name here and a short header

# import statements
import sys
```

```

# class definitions
class MyNetwork(nn.Module):
    def __init__(self):
        pass

    # computes a forward pass for the network
    # methods need a summary comment
    def forward(self, x):
        return x

# useful functions with a comment for each function
def train_network( arguments ):
    return

# main function (yes, it needs a comment too)
def main(argv):
    # handle any command line arguments in argv

    # main function code
    return

if __name__ == "__main__":
    main(sys.argv)

```

## A. Get the MNIST digit data set

The MNIST digit data consists of a training set of 60k 28x28 labeled digits and a test set of 10k 28x28 labeled digits. The data set can be imported within your code directly from the torchvision package as torchvision.datasets.MNIST. Use the matplotlib pyplot package or OpenCV to look at the first six example digits of the test set (hint, don't shuffle the test set so you always get the same six digits). Look up examples that make use of the pyplot subplot method to create a grid of plots.

*Include a plot of the first six example digits from the test set in your report.*

## B. Build a network model

Similar to the example in the tutorial, create a network with the following layers.

- A convolution layer with 10 5x5 filters
- A max pooling layer with a 2x2 window and a ReLU function applied.
- A convolution layer with 20 5x5 filters
- A dropout layer with a dropout rate rate of your choice (anything from 5-50% is fine)
- A max pooling layer with a 2x2 window and a ReLU function applied
- A flattening operation followed by a fully connected Linear layer with 50 nodes and a ReLU function on the output
- A final fully connected Linear layer with 10 nodes and the log\_softmax function applied to the output.

The design of your network class will be in the constructor and forward method of your

Network class, derived from the torch nn.Module class.

*Put a diagram of your network in your report. A photo of a hand-drawn network is fine.*

### C. Train the model

Train the model for at least five epochs, one epoch at a time, evaluating the model on both the training and test sets after each epoch. An epoch is one complete pass through the training data. Pick a batch size of your choice. Make a plot, such as the one below, of the training and testing error.

*Collect the accuracy scores and plot the training and testing accuracy in a graph, such as the one above. Try to use different colors. Include this plot in your report along with a discussion of the training behavior.*

### D. Save the network to a file

When the network is trained, save it to a file.

### E. Read the network and run it on the test set

*In a separate python file, read the network and run the model on the first 10 examples in the test set. Before you run the samples through the network, make sure you set the network to evaluation mode rather than training mode (`network.eval()`). In training mode the dropout layer randomly sets node values to zero. In evaluation mode, it multiplies each value by 1-dropout rate so the same pattern will generate the same output each time. For each example image, have your program print out (on the command line) the 10 network output values (use only 2 decimal places), the index of the max output value, and the correct label of the digit. The network should correctly classify all 10 of the examples (but it might make one mistake). Have your program also plot the first 9 digits as a 3x3 grid with the prediction for each example above it, as below.*

*Include a table (or screen shot) of your printed values and the plot of the first 9 digits of the test set in your report. Discuss the results.*

## F. Test the network on new inputs

Write out the ten digits [0-9] in your own handwriting on a piece of white paper (not too close together). You will want to use thick (really thick) lines when writing the digits. I suggest using a marker or sharpie. Writing them on a white board or digitally on your computer will also work. Take a picture of the digits, then crop each digit to its own square image. You may want to scale them to 28x28 and look at them to make sure the digits are visible. If you don't already have it installed, the ImageMagick package can be very useful for cropping and resizing images using the command line.

In a separate file, or as the last step in your code from the prior task, read the images you created, convert them to greyscale, resize them to 28x28 (if necessary) and run them through the network. During the resize process, double-check that the digits are still clearly visible. Make sure to match the intensities of the new images to the intensities of the test data. Are the MNIST data set digits white on black or are the digits black on white? **You may need to invert the intensities of your photos for them to match.**

*Display how well the network performed on this new input in your report by showing the digits and their classified result. Discuss the results.*

## 2. Examine your network

The second main task is to examine your network and analyze how it processes the data. Make this a separate code file from task one. Read in the trained network as the first step and print the model. This should show you the structure of the network and the name of each layer.

### A. Analyze the first layer

Get the weights of the first layer. When accessing a network, use the name of the layer as specified in the model printout. In this case, the name should be conv1. If your network variable is model, then you would access the weights using model.conv1.weight. The result is

a tensor that should have the shape [10, 1, 5, 5]. That means there are ten filters, 1 input channel, and each filter is 5x5 in size. To access the *i*th 5x5 filter, use `weights[i, 0]`. Print the filter weights and their shape.

*Visualize the ten filters using pyplot. You can use the pyplot functions `figure`, `subplot`, and `imshow` to make a 3x4 grid of figures such as the image below. If you set `xticks` and `yticks` to the empty list `[]`, it will give you a cleaner plot. Describe how you would interpret the filters at a high level. Do they look (roughly) like any of the standard filters?*


## B. Show the effect of the filters

Use OpenCV's `filter2D` function to apply the 10 filters to the first training example image. Generate a plot of the 10 filtered images such as the one below. When working with the weights, you will need to tell pyTorch it does not need to calculate gradients. You can do this using the following structure.

```
with torch.no_grad():  
    # put your code here
```

*In your report, include the plot and discuss whether the results make sense given the filters. How do the results match your interpretation of the filters?*

### 3. Transfer Learning on Greek Letters

The goal of this step is to re-use the the MNIST digit recognition network you built in step 1 to recognize three different greek letters: alpha, beta, and gamma. Here are **some examples of the greek letters** (<https://northeastern.instructure.com/courses/206145/files/31639249?wrap=1>)  ([https://northeastern.instructure.com/courses/206145/files/31639249/download?download\\_frd=1](https://northeastern.instructure.com/courses/206145/files/31639249/download?download_frd=1)) .

In order to make use of the pre-trained network, your code should (1) generate the MNIST network (you should import your code from task 1), (2) read an existing model from a file and load the pre-trained weights, (3) freeze the network weights, and (4) replace the last layer with a new Linear layer with three nodes. Your printout of the MNIST model in task 1 should show the name of the last layer. The following code snippet shows how to freeze all of the network weights. When you replace the last layer it will, by default, not be frozen.

```
# freezes the parameters for the whole network
for param in network.parameters():
    param.requires_grad = False
```

The greek letter images are 133 x 133 and in color. Here is some code that will transform the RGB images to grayscale, scale and crop them to the correct size, and invert the intensities to match the MNIST digits.

```
# greek data set transform
class GreekTransform:
    def __init__(self):
        pass

    def __call__(self, x):
        x = torchvision.transforms.functional.rgb_to_grayscale( x )
        x = torchvision.transforms.functional.affine( x, 0, (0,0), 36/128, 0 )
        x = torchvision.transforms.functional.center_crop( x, (28, 28) )
        return torchvision.transforms.functional.invert( x )
```

When you create the `DataLoader` for this data set, use the `ImageFolder` class to create the `DataSet` and use the `GreekTransform` as an argument to the `ImageFolder` constructor. Note that the variable `training_set_path` should be the path to the directory containing the three folders `alpha`, `beta`, and `gamma`.

```
# DataLoader for the Greek data set
greek_train = torch.utils.data.DataLoader(
    torchvision.datasets.ImageFolder(training_set_path,
                                     transform = torchvision.transforms.Compose([torchvision.transforms.ToTensor(),
                                                                                       torchvision.transforms.Normalize([0.4914, 0.4516, 0.4215], [0.8, 0.8, 0.8])])
                                     ),
    batch_size=128,
    shuffle=True,
    num_workers=4)

# DataLoader for the Greek test set
greek_test = torch.utils.data.DataLoader(
    torchvision.datasets.ImageFolder(test_set_path,
                                     transform = torchvision.transforms.Compose([torchvision.transforms.ToTensor(),
                                                                                       torchvision.transforms.Normalize([0.4914, 0.4516, 0.4215], [0.8, 0.8, 0.8])])
                                     ),
    batch_size=128,
    shuffle=True,
    num_workers=4)
```

```

ision.transforms.Normalize(
0.1307,), (0.3081,) ) ] ) ),
    batch_size = 5,
    shuffle = True )

```

How many epochs does it take using the 27 examples in order to reasonably identify them?

Take a picture of several of your own alpha, beta, and gamma symbols. Crop and rescale them to about 128x128 images to match the existing data and see if they are correctly classified by the trained network. As with the prior images you collected, use thick lines.

Here are some [additional greek letters](https://northeastern.instructure.com/courses/206145/files/33809021?wrap=1)

(<https://northeastern.instructure.com/courses/206145/files/33809021?wrap=1>)\_ ↓

([https://northeastern.instructure.com/courses/206145/files/33809021/download?download\\_frd=1](https://northeastern.instructure.com/courses/206145/files/33809021/download?download_frd=1)) captured by prior students.

**Include in your report** *a plot of the training error, a printout of your modified network, and the results on the additional data. In your readme, please include a link to a zip file or drive with your additional greek letter examples in your submission.*

#### 4. Design your own experiment

The final main task is to undertake some experimentation with the deep network for the MNIST task. Your goal is to evaluate the effect of changing different aspects of the network. Pick at least three dimensions along which you can change the network architecture and see if you can optimize the network performance and/or training time along those three dimensions. Do your best to automate the process.

You may want to try using the MNIST Fashion data set instead of the digits data set for this step. The fashion data set is a little more challenging, but similar in size, and it provides more room to see the effect of changes in the network.

Potential dimensions to evaluate include:

- The number of convolution layers
- The size of the convolution filters
- The number of convolution filters in a layer
- The number of hidden nodes in the Dense layer
- The dropout rates of the Dropout layer
- Whether to add another dropout layer after the fully connected layer
- The size of the pooling layer filters
- The number or location of pooling layers

- The activation function for each layer
- The number of epochs of training
- The batch size while training

### A. Develop a plan

Come up with a plan for what you want to explore and the metrics you will use. Determine the range of options in each dimension to explore (e.g. L options in dimension 1, M options in dimension 2, and N options in dimension 3). You don't have to evaluate all  $L * M * N$  options unless you want to. Instead, think about using a linear search strategy where you hold two parameters constant and optimize the third, then switch things up, optimizing one parameter at a time in a round-robin or randomized fashion. Overall, plan to evaluate 50-100 network variations (again, automate this process).

*Include your plan in your report.*

### B. Predict the results

Before starting your evaluation, come up with a hypothesis for how you expect the network to behave along each dimension. *Include these hypotheses in your report and then discuss whether the evaluation supported the hypothesis.*

### C. Execute your plan

*Run the evaluation, report on the results, and discuss how they matched your hypotheses.*

---

## Final Project Design

Think about what you are going to do for a final project. Form your groups, if you wish, up to five people per group. The project should be a little larger than one of the defined projects, scaled up a little if you have more than two people. Possible projects include:

- A significant continuation of one of the defined projects.
- Machine learning applied to a new data set.
- An application of your choice (e.g. tracking billiard balls and showing the path of each ball)
- Look at some of the computer vision competitions (e.g. Kaggle) and see what you can do.

Your project description should be 1 page, and it needs to include the following components.

- The problem statement (1 paragraph at most). Be very clear about the inputs and desired outputs.
- How you will obtain data for your project. (You do not have time to collect and annotate more than



a small data set.)

- If you are using an existing deep network architecture, where are you getting the model/code for it?
- Give a brief description of the design of your solution (1 paragraph at most).
- If you need to train a network, where are you going to find the computing power necessary to train and evaluate it?
- What would be a good outcome for the project?

*Submit a one-page final project proposal as a separate pdf along with your code on GradeScope. Include the names of everyone in your group, which may be different than the person you might be partnering with for project 5.*

---

## Extensions


- Evaluate more dimensions on task 3.
  - Try more greek letters than alpha, beta, and gamma.
  - Explore a different computer vision task using deep learning with some available data.
  - There are many pre-trained networks available in the PyTorch package. Try loading one and evaluate its first couple of convolutional layers as in task 2.
  - Replace the first layer of the MNIST network with a filter bank of your choosing (e.g. Gabor filters) and retrain the rest of the network, holding the first layer constant. How does it do?
  - Build a live video digit recognition application using the trained network.
- 

## Report

When you are done with your project, write a short report that demonstrates the functionality of each task. You can write your report in the application of your choice, but you need to submit it **as a pdf** along with your code. Your report should have the following structure. Please **do not include code** in your report.

1. A short description of the overall project in your own words. (200 words or less)
  2. Any required images along with a short description of the meaning of the image.
  3. A description and example images of any extensions.
  4. A short reflection of what you learned.
  5. Acknowledgement of any materials or people you consulted for the assignment.
-

# Submission

Submit your code and report to **Gradescope**  (<https://www.gradescope.com>). When you are ready to submit, upload your code, report, and a readme file. The readme file should contain the following information.

- Your name and any other group members, if any.
- Links/URLs to any videos you created and want to submit as part of your report.
- Links/URLs to any data you need to submit with the project.
- Whether you are using any time travel days and how many.

For project 5, submit your .py files, pdf report, and readme.txt (readme.md). Note, if you find any errors or need to update your code, you can resubmit as many times as you wish up until the deadline.

As noted in the syllabus, projects submitted by the deadline can receive full credit for the base project and extensions. (max 30/30). Projects submitted up to a week after the deadline can receive full credit for the base project, but not extensions (max 26/30). You also have eight time travel days you can use during the semester to adjust any deadline, using up to three days on any one assignment (no fractional days). If you want to use your time travel days, note that in your readme file. If you need to make use of the "stuff happens" clause of the syllabus, contact the instructor as soon as possible to make alternative arrangements.

## **Receiving grades and feedback**

After your project has been graded, you can find your grade and feedback on Gradescope.