2 layer NN: (Input Size) → 300 → 10

- Don't follow the paper exactly - Just see if you can get the same test error

→ Preprocessing - I will use a validation set and create model complexity graph if necessary.

- Train on the whole set → Train + Validate
  - → Model Cpxity Graph → Early Stopping or how many iterations.
  - → Save Initialization → Random Initialization ALWAYS
  - → Learning Rate (Not argument) OR Initialization from file
- Evaluate → Model File, MNIST test set, Metrics: Precision, Recall, Accuracy
- Predict → Model File, JPG image

Project Structure

- Train on Training Data.
- Evaluate on Test Data.
- Predict for new image.

Modules to use

- PIL, os, numpy, matplotlib, pickle, random, logging, pydocs, pip

```
2-layer-nn-300HU-MSE-none-4p7/
    README.md          .gitignore
    train.py           requirements.txt
    evaluate.py        venv/
    predict.py         "training-results-1/"
    net.py             "evaluate-results-1/"
    preprocess.py      "predict-results-1/"
    write.py
    graph.py
```

OPEN-FILE → Check if binary file is of proper format
- Is the right file.

- First check size, magic number, num_example
- Extract data.
- Convert to numpy.

Use global variables - Treat the whole module as an object.

_MNIST_SIZES = {
"train-images": 47040016, "train-labels": 60008
"test-images": 7840016, "train-labels": 10008 }

_ MNIST_MAGIC_NUM = . . . -
- MNIST_NUM_EXAMPLES= ---
check-file(f, use = "train", type = "image"):

File Checker, File Extractor, ImageMaker
StaticMethod, ClassMethod.
→ __init__() → Set all these variables.

checkfile_size → size = mnist.sizes [f"{3}_{3}"]

Don't go too fast - check_magic_number →

FileChecker → Checks whether files are correct →
Main method is check_file → Returns the type of file
Checks size, magic_num + num_examples.

```
for use in ["train", "test"]:
    for type in ["image", "label"]:
        if is_mnist(f, use, type):
            return use, type
return None, None
```

extract_file(

FileExtractor → Converts to numpy array
InputOperations → Normalization + PicturizationFunctions.

```
is_mnist(f, use, type):
  return correctsize(f, use, type) and correct_magic_num(f, type)
                                   and correct_num_examples(f, use)

Mnist extract → extract_all
 FILE_PATHS = ["assets/    ]

    ~~for use in ["train", "test"]:~~
for ~~for type in ["images", "labels"]:~~   np_arrays = [ ]
        with open(FILE_PATH[f"  ]"rb").as f :
use, type = mnist check(f)
    if use is not NONE:
      np_arrays.append(extract ~~images~~(f, type))
```

Miscellaneous Python Knowledge

→ If I have a bytes object - an immutable collection of numbers (0-255), and if it happens to have 4 bytes - then

int.from_bytes(bytes, endianness="big" / "little") will decode the integer value of bytes

→ PIL.Image.fromarray(obj, mode) will convert a numpy array to a PIL image.