Daniel Wilczak
Jack Nguyen
Liam Kehoe
Nathan Foster

How Easy? Try it!

Try it:

```
pip install EasyNN
```

Requires: Python version >= 3.9.7

Example Code

https://bit.ly/3pGHuAI

Sprint Before and After

Refined Usability

- **Before:**

```
from EasyNN.dataset.mnist.number import trained_model, dataset, show

# Downloads dataset to computer
train_data,train_labels,test_data,test_labels = dataset

# Grab a training data image
image = test_data[0]

# Uses the EasyNN train model on an example test image.
print(trained_model(image))
```

- **After:**

```
from EasyNN.examples.mnist.number.trained import model
from EasyNN.examples.mnist.number.data import dataset

images, labels = dataset

# Classify what the second image is in the dataset.
print(model.classify(images[1]))
```

Sprint Before and After

High Modularity

Before:

- Book had no modules.
- Lots of code shoved into each module.

After:

```
from EasyNN.examples.mnist.number.untrained import model
from EasyNN.examples.mnist.number.data import dataset
from EasyNN.model import Model

images, labels = dataset

# Train the model.
model.train()
model.save("number")

# Load the saved model
model = Model.load("number")
```

Sprint Before and After

Customizable NN structures

Before:

- No inheritance, little code flexibility.

After:

- Abstraction/inheritance avoids repeated code.
- Clearer interfaces.

```
from EasyNN.model import Network, Normalize, Randomize, ReLU, LogSoftMax
from EasyNN.examples.mnist.number.data import dataset

# Create the mnist model.
model = Network(
    Normalize(1e-3), Randomize(0.01),
    1024, ReLU,
    256, ReLU,
    10, LogSoftMax,
)

model.training.data = dataset
```

Sprint
Before and After

Callbacks

Before:

- No ability to stop or run code during training.

After:

- Callback functions for easy flexibility.
- Multiple callbacks for different parts of training.

```
@model.on_optimization_start
def setup(model):
    model.validation.batch = MiniBatch(1024)

@model.on_validation_end
def terminate(model):
    if model.accuracy(*model.validation.sample) > model.validation.accuracy_limit:
        model.validation.successes += 1
    else:
        model.validation.successes = 0
    model.stop_training |= model.validation.successes >= model.validation.accuracy_
```

Sprint 2 Summary

Sprint 2 goals:

- Swap to Version 2.
- Create custom structured NN’s.
- Save progression data.
- Plot progression data.
- Add documentation.

Sprint 2 accomplishments:

- Swap to Version 2.
- Create custom structured NN’s.
- Added documentation.

Other accomplishments:

- Refined designs.
- High modularity.
- High inter-modular cohesion.
- Lower intra-class coupling.
- New features.

Modules

Before:

- Shoved lots of stuff into one module.

```
from EasyNN.dataset.mnist.number import trained_model, dataset, show
```

After:

- Separated stuff into separate modules/directories.
- GitHub stores large files.
- Large files are downloaded locally the first time.

```
from EasyNN.examples.mnist.number.trained import model
from EasyNN.examples.mnist.number.data import dataset, labels
```

Classes

Before:

- Hardly any abstraction/inheritance.
- No type-hints.
- No documentation for attributes.
- Hard to modify.

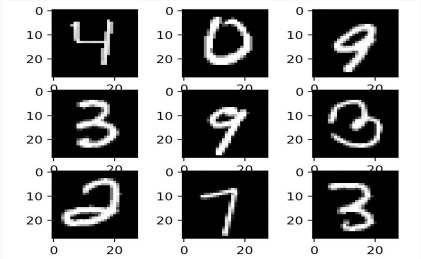
After:

- Implemented abstract classes and used inheritance.
- Attributes and methods are type-hinted.
- Attributes are documented e.g. `help(Model.loss)`.
- Easy to modify e.g. callback functions.

New Features

Features:

- Callback functions.
- Utility functions.
- Automatic learning rate tuning.
- Loading/saving models.



```
# Download an example image.
download("four.jpg","https://bit.ly/3lAJrMe")

format_options = dict(
    grayscale=True,
    invert=True,
    process=True,
    contrast=30,
    resize=(28, 28),
    rotate=3,
)

# Converting your image into the correct format for the mnist number dataset.
image = image("four.jpg").format(**format_options)

compare(image,dataset=dataset)
```

Basic Pre-Trained Models

Models:

- MNIST numbers.
- MNIST fashion.
- CIFAR10.

Future:

- Drone turtle images.
- Rhino Net Images.
- CIFAR-100.

Project Timeline

Sprint 1:
- Test version 1.
- Datasets:
  - Number MNIST
  - Fashion MNIST
  - CIFAR10
- GitHub Wiki documentation.

Sprint 2:
- Swap to Version 2.
- Create custom structured NN’s.
- Graph training progression.
- Add documentation.
- Save/load custom models.

Sprint 3:
- Database support
- More models
- Web Scraping feature
- Dataset directory support.
- More features and customizability.
- More documentation.

Questions?

Documentation is key.

Home

EasyNN

- Getting Started Guide

Trained Models

- MNIST Numbers
- MNIST Fashion

Try it: pip3 install EasyNN
Requires: Python version >= 3.9.7

Documentation makes our software SO easy:

MNIST Numbers

Daniel Wilczak edited this page 2 days ago · 70 revisions

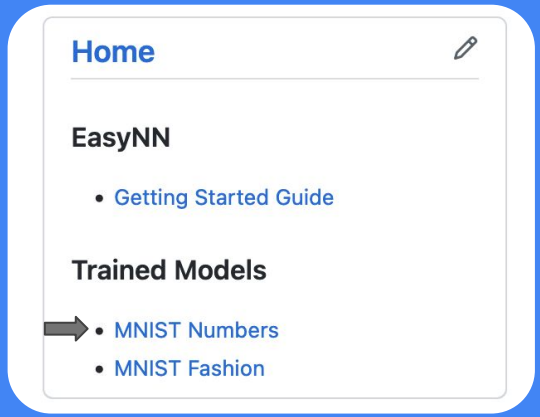
MNIST Number:

The MNIST dataset is an acronym that stands for the Modified National Institute of Standards and Technology dataset. It a dataset of 60,000 small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9.

EasyNN Functions:

model: Untrained model that has structure setup. Model -> Train -> Predict
trained_model: Use an EasyNN trained model to identify a test image.
show: Shows the image as either a matplotlib graph or numpy array printout with proper width.
dataset: Grab 60,000 training image/labels and 10,000 test images/labels from EasyNN's github.
preprocess: Preprocessing an image so it looks similar to the datasets images. **(STILL IN WORK)**
compare: Compare users image, dataset image and preprocessed image. **(STILL IN WORK)**

Documentation is key.



Try it: pip3 install EasyNN
Requires: Python version >= 3.9.7

Documentation makes your software easy:

Model Output Dictionary:

This seems straight forward in the MNSIST Number dataset but on other datasets the number output can relate to a string. See the MNIST Fashion dataset to see an example of this.

```
number_mnist_labels = {
    0: 0,
    1: 1,
    2: 2,
    3: 3,
    4: 4,
    5: 5,
    6: 6,
    7: 7,
    8: 8,
    9: 9
}
```

EasyNN Model Functions:

model(image):

Untrained model that has structure setup. Model -> Train -> Predict

```
from EasyNN.dataset.mnist.number import model, dataset

# Downloads dataset to computer
train_data,train_labels,test_data,test_labels = dataset

# Trains model and use an example test image to get a prediction on an image.
print(model(test_data[0]))
```

Documentation is key.

Home

EasyNN

- Getting Started Guide

Trained Models

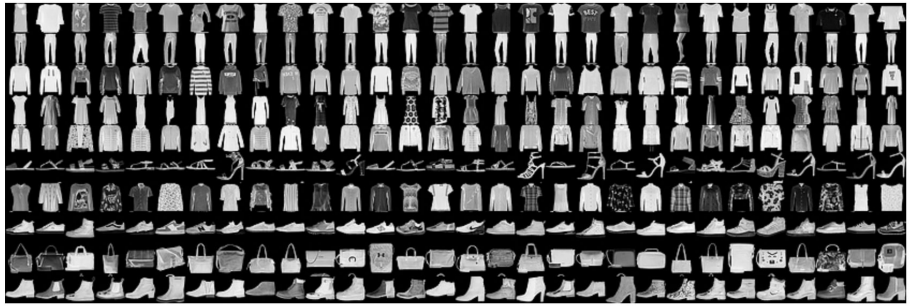
- MNIST Numbers
- MNIST Fashion

Try it: pip3 install EasyNN
Requires: Python version >= 3.9.7

Documentation makes your software easy:

MNIST Fashion

Daniel Wilczak edited this page 2 days ago · 2 revisions



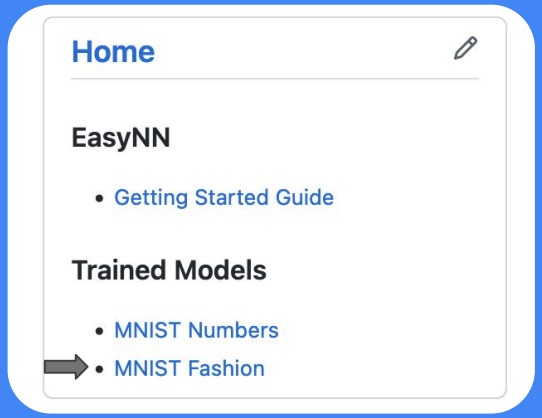
MNIST Fashion:

Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.

EasyNN Functions:

model: Untrained model that has structure setup. Model -> Train -> Predict
trained_model: Use an EasyNN trained model to identify a test image.
show: Shows the image as either a matplotlib graph or numpy array printout with proper width.
dataset: Grab 60,000 training image/labels and 10,000 test images/labels from EasyNN's github.
preprocess: Preprocessing an image so it looks similar to the datasets images. **(STILL IN WORK)**
compare: Compare users image, dataset image and preprocessed image. **(STILL IN WORK)**

Documentation is key.



Try it: pip3 install EasyNN
Requires: Python version >= 3.9.7

Documentation makes your software easy:

trained_model(image):

Use an EasyNN trained model to identify a test image.

```
from EasyNN.dataset.mnist.fashion import trained_model, dataset

# Downloads dataset to computer
train_data,train_labels,test_data,test_labels = dataset

# Uses the EasyNN train model on an example test image.
print(trained_model(test_data[0]))
```

show(image,output_type):

Shows the image as either a matplotlib graph or numpy array printout with proper width.

```
show(image, "image")  # Shows image as matplotlib graph
show(image, "array")  # Shows image as numpy array print out to proper width.
```

dataset():

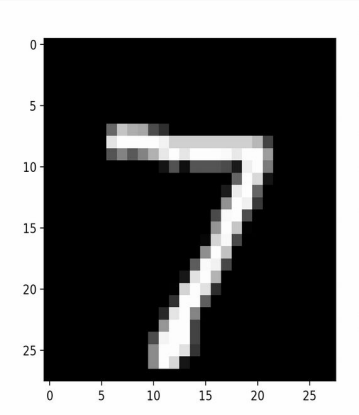
Grab 60,000 training image/labels and 10,000 test images/labels from EasyNN's github.

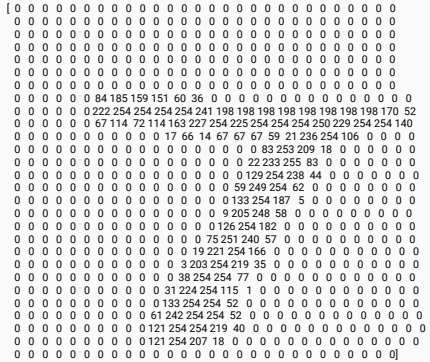
```
from EasyNN.dataset.mnist.fashion import trained_model, dataset

# Downloads dataset to computer
train_data,train_labels,test_data,test_labels = dataset

# Grab a test image from the dataset.
user_image = test_data[0]

# Use the show function to show a ploted image and array data.
show(user_image, "image")
show(user_image, "array")
```

User input Considerations

Images: We assume very little of the users knowledge of how images in the dataset work or even their own images.

Showing: Images can be shown as typical image or by seeing the data in the background as a numpy array.





User Success:

User success will be critical on whether or not a user can easily access and display the data they are using.

Design Considerations

Assumptions: We assume very little. The user should know basic Python syntax.

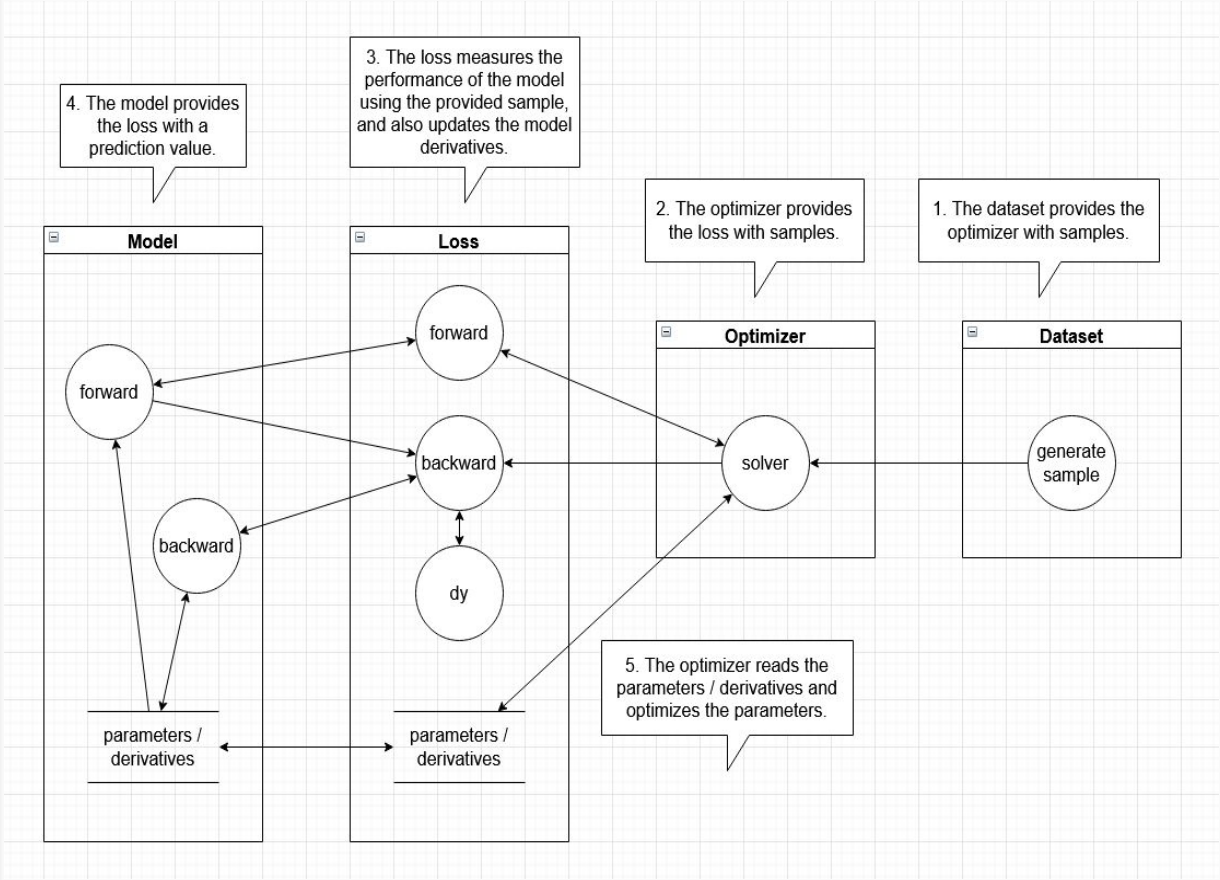
Dependencies: Numpy, Matplotlib, PIL

- **Handled by pip install**

General Constraints/ Core Values:
All users will understand.
All users can use.
(No fancy GPU acceleration).
(No big dependencies i.e Keras, Open CV etc)

Industrial Standards Followed:
Python PEP 8 formatting standards.
Following Google/Numpy in code documentation standards.

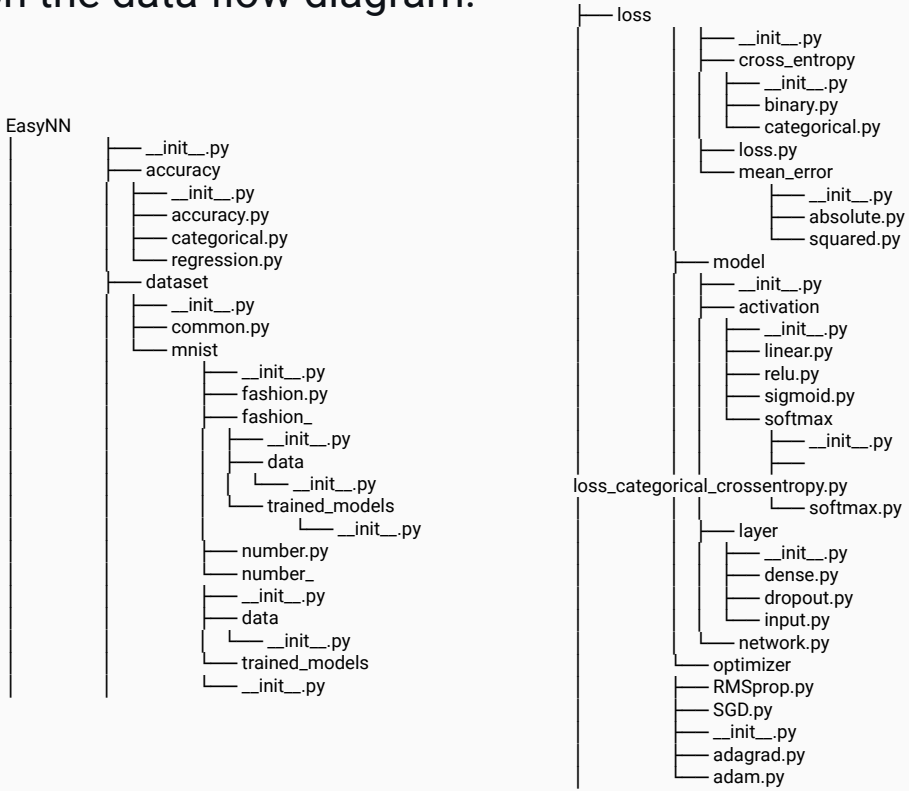
System Architecture



Sub-System Design

The class diagram is currently a work in progress, but it is partially laid out by the file structure.

The main interactions between classes are shown on the data flow diagram.



Lessons Learned

Scope reduction from last semester.

Wiki documentation is the key to success for a good user base.

Work together to follow standards.

Neural Networks are hard and take time for people to fully understand how they work.

Finding a middle ground between:

TensorFlow's: general complexity for all problems.
Books/Beginner: tutorials lack usability for new problems.

EasyNN bridging the middle ground.

Project Timeline

Sprint 1:
- Test version 1.
- Datasets:
  - Number MNIST
  - Fashion MNIST
  - CIFAR10
- GitHub Wiki documentation.

Sprint 2:
- Swap to Version 2.
- Create custom structured NN’s.
- Save training progression.
- Graph training progression.
- Add documentation.

Sprint 3:
- Save/load custom models.
- More features and customizability.