18-819F: Introduction to Quantum Computing
47-779/47-785: Quantum Integer Programming & Quantum Machine Learning

Image Classification using MNIST Dataset (PyTorch : MLP & TF-Keras : CNN)
& CNN Visualizer

2022.09.19.

Electrical & Computer ENGINEERING
TEPPER
PruthviRaj G
USRA Universities Space Research Association

Agenda

- Image Classification using MNIST Dataset
  - MNIST Dataset Description
  - Multi-layer Perceptron example using PyTorch.
  - Convolution Neural Network example using TensorFlow (Keras).

Keras is a high level Neural Network library that runs on top of Tensorflow

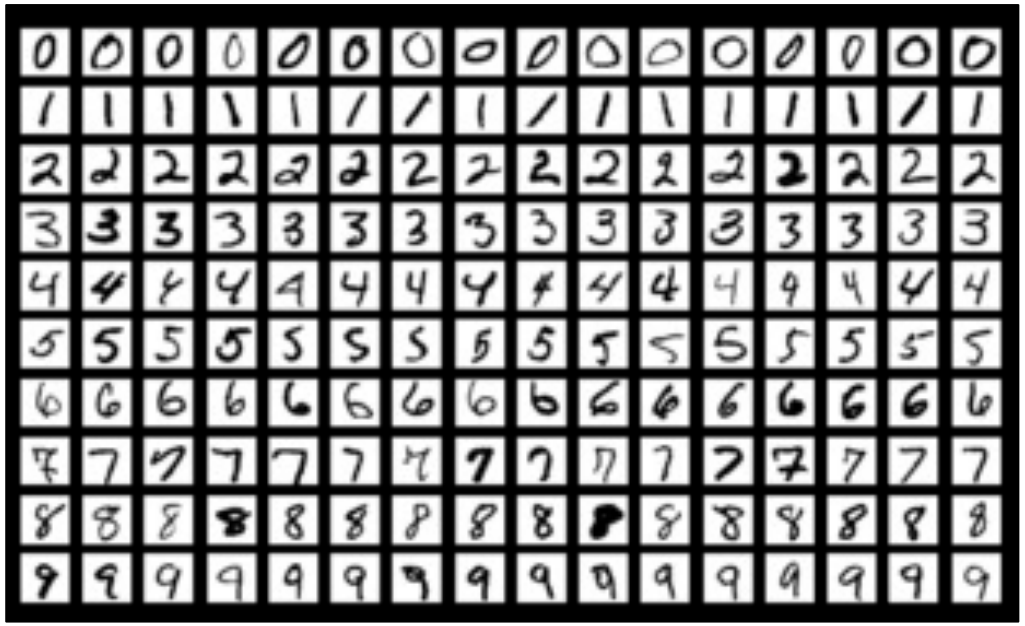
- CNN Visualizer
  - https://poloclub.github.io/cnn-explainer

MNIST Dataset Description

Modified National Institute of Standards and Technology database

- Training set – 60,000 images
- Test set – 10,000 images

* Images of digits are centered and scaled to the same size.



Input :

- Image: tensor containing the 28x28 image.
- Label: an integer between 0 and 9 representing the digit.

Multi-layer Perceptron example using PyTorch.

Step 0: Import libraries

```
import numpy as np
import torch
import sys
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

from torch.utils import data
from torchvision import transforms
from torchvision.datasets import MNIST

import matplotlib.pyplot as plt
import time
```

import the pytorch package

import the neural networks

import the functions from the torch library as F.

import optimization algorithms

access to various image transformation algorithms from pytorch

downloading MNIST dataset.

importing python plot from matplotlib library.

Time for calculating duration

Multi-layer Perceptron example using PyTorch.

Step 1: Download MNIST Dataset

download train dataset

```
train_set = MNIST('./data', train=True, download=True, transform=transforms.ToTensor())
```

download test dataset

```
test_set = MNIST('./data', train=False, download=True, transform=transforms.ToTensor())
train_data = train_set.train_data
train_data = train_set.transform(train_data.numpy())
```

Convert numpy data to torch.

```
print('[Train Data Information]')
print(' - Numpy Shape:', train_data.cpu().numpy().shape)
print(' - Tensor Shape:', train_data.size())

print('\n[Train Labels Information]')
print(' - Numpy Shape:', train.train_labels.cpu().numpy().shape)
print(' - Tensor Shape:', train.train_labels.size())
```

=> (28, 60000, 28)

=> torch.Size([28, 60000, 28])

=> (60000,)

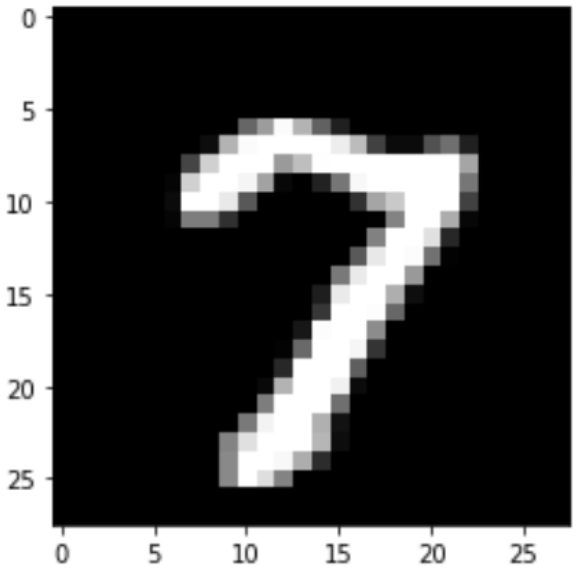
=> torch.Size([60000])

Multi-layer Perceptron example using PyTorch.

Step 3: Visualize Dataset sample inputs

```
# Visualize a training instance with matplotlib
plt.imshow(train_set.train_data.cpu().numpy()[819], cmap='gray')
print(f"Label: ", train_set.train_labels.cpu().numpy()[819])
```

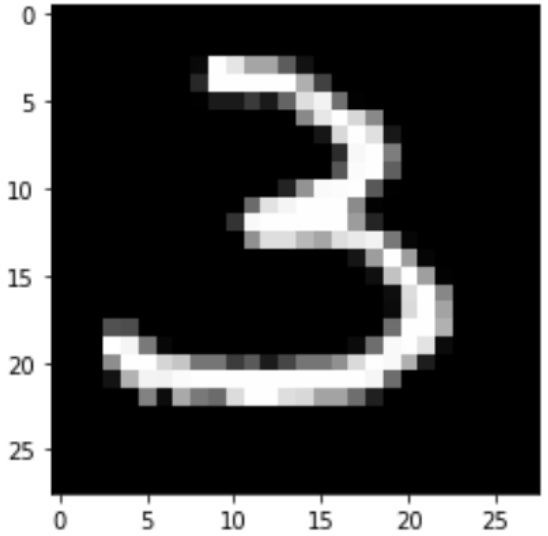
Label: 7



Train Sample label

```
# Visualize a Testing instance with matplotlib
plt.imshow(test_set.test_data.cpu().numpy()[819], cmap='gray')
print(f"Label: ",test_set.test_labels.cpu().numpy()[819])
```

Label: 3



Test Sample label.

Multi-layer Perceptron example using PyTorch.

Step 4: Create DataLoader

```
class MNIST_Dataset(data.Dataset):
    def __init__(self, X, Y):
        self.X = X
        self.Y = Y
    def __len__(self):
        return len(self.Y)

    def __getitem__(self,index):
        X = self.X[index].float().reshape(-1) #flatten the input
        Y = self.Y[index].long()
        return X,Y
```

initialization

convert to float and flatten the input

Multi-layer Perceptron example using PyTorch.

Step 4: Create DataLoader

```
cuda = torch.cuda.is_available()
num_workers = 8 if cuda else 0

# Training
train_dataset = MNIST_Dataset(train_set.train_data, train_set.train_labels)

train_loader_args = dict(shuffle=True, batch_size=256, num_workers=num_workers, pin_memory=True) if cuda\
                    else dict(shuffle=True, batch_size=64)
train_loader = data.DataLoader(train_dataset, **train_loader_args)

# Testing
test_dataset = MNIST_Dataset(test_set.test_data, test_set.test_labels)

test_loader_args = dict(shuffle=False, batch_size=256, num_workers=num_workers, pin_memory=True) if cuda\
                    else dict(shuffle=False, batch_size=1)
test_loader = data.DataLoader(test_dataset, **test_loader_args)
```

if cuda is available use 8 workers.

shuffle the dataset and set Batch to 256 if cuda available else 64

Don't shuffle the test set loader.

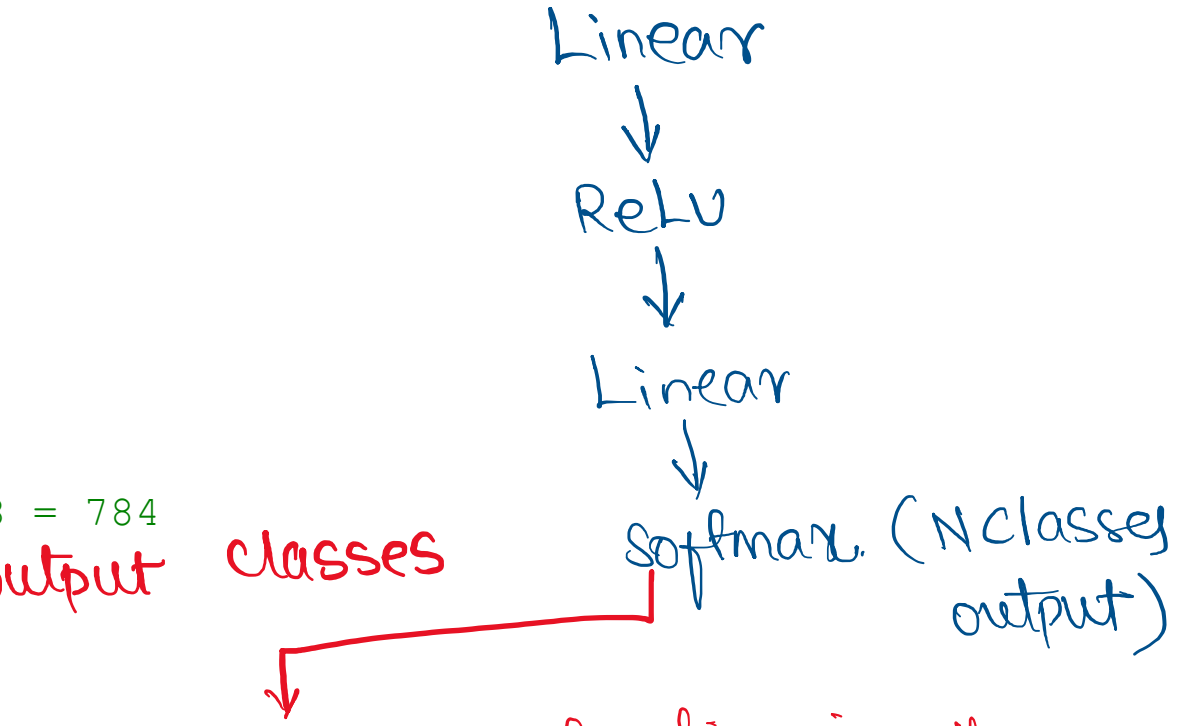
It can accelerate CPU to GPU memory copy operation.

Multi-layer Perceptron example using PyTorch.

Step 5: MLP Model Definition

```
# Multi_layer Perceptron Model Creation
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        layers = []
        layers.append(nn.Linear(784,512)) # 28x28 = 784
        layers.append(nn.ReLU())
        layers.append(nn.Linear(512,10))
        self.net = nn.Sequential(*layers)

    def forward(self, x):
        return self.net(x)
```



Multi-layer Perceptron example using PyTorch.

Step 5.1: MLP Model Creation

$$L_{CE} = -\sum_{i=1}^{n} t_i \log(P_i), \text{ for } n \text{ classes.}$$

P_i: Softmax probability.

t_i: truth label

```
# Multi_layer Perceptron Model Creation
model = MLP()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters())
device = torch.device("cuda" if cuda else "cpu")
model.to(device)
print(model)
```

→ Adam combines the advantages of AdaGrad + RMSprop.

```
MLP(
  (net): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): ReLU() (2): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

Multi-layer Perceptron example using PyTorch.

Step 6: Training Function

```
def train_epoch(model, train_loader, criterion, optimizer):
    model.train()

    running_loss = 0.0

    start_time = time.time()
    for batch_idx, (data, target) in enumerate(train_loader):
        optimizer.zero_grad()
        data = data.to(device)
        target = target.to(device)
        outputs = model(data)

        loss = criterion(outputs, target)
        running_loss += loss.item()

        loss.backward()
        optimizer.step()
    end_time = time.time()

    running_loss /= len(train_loader)
    print('Training Loss: ', running_loss, 'Time: ',end_time - start_time, 's')
    return running_loss
```

forward propogation

loss calculation in BW

stepping with the optimizer.

Multi-layer Perceptron example using PyTorch.

Step 6.1: Testing Function

```
def test_model(model, test_loader, criterion):
    with torch.no_grad():
        model.eval()

        running_loss = 0.0
        total_predictions = 0.0
        correct_predictions = 0.0

        for batch_idx, (data, target) in enumerate(test_loader):
            data = data.to(device)
            target = target.to(device)
            outputs = model(data)

            _, predicted = torch.max(outputs.data, 1)
            total_predictions += target.size(0)
            correct_predictions += (predicted == target).sum().item()

            loss = criterion(outputs, target).detach()
            running_loss += loss.item()

        running_loss /= len(test_loader)
        accuracy = (correct_predictions/total_predictions)*100.0
        print('Testing Loss: ', running_loss)
        print('Testing Accuracy: ', accuracy, '%')
        return running_loss, accuracy
```

NO backpropogation in the Inference phase

Initializers.

computing the model output fwd pass

all predictions

if label is matching

$$\text{accuracy} = \frac{\text{Correct predictions}}{\text{Total predictions}} \times 100$$

Multi-layer Perceptron example using PyTorch.

Step 7: Training the Model for N Epochs

```
n_epochs = 7
Train_loss = []
Test_loss = []
Test_acc = []

for i in range(n_epochs):
    train_loss = train_epoch(model, train_loader, criterion, optimizer)
    test_loss, test_acc = test_model(model, test_loader, criterion)
    Train_loss.append(train_loss)
    Test_loss.append(test_loss)
    Test_acc.append(test_acc)
    print('='*40)
```

train the model

Test the model

Test accuracy.

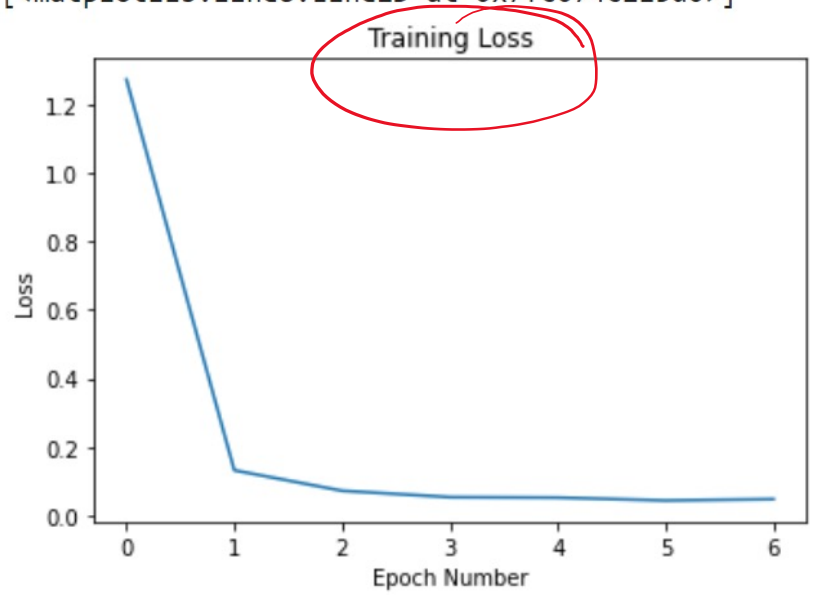
```
Training Loss:  1.2734107589468042 Time:  1.793694019317627 s
Testing Loss:  0.19492523474618792
Testing Accuracy:  94.82000000000001 %
========================================
Training Loss:  0.13140256043444287 Time:  1.7859649658203125 s
Testing Loss:  0.1799794932710938
Testing Accuracy:  95.35 %
========================================
Training Loss:  0.07155194909490169 Time:  1.8141794204711914 s
Testing Loss:  0.14994464091723786
Testing Accuracy:  96.41 %
========================================
Training Loss:  0.052934684491458724 Time:  1.7892048358917236 s
Testing Loss:  0.14333404847420753
Testing Accuracy:  96.63000000000001 %
========================================
Training Loss:  0.05158523537139309 Time:  1.8411898612976074 s
Testing Loss:  0.1396400388856364
Testing Accuracy:  96.8 %
========================================
Training Loss:  0.04323105293702572 Time:  1.8349244594573975 s
Testing Loss:  0.14093846913419839
Testing Accuracy:  96.89 %
========================================
Training Loss:  0.047089373836036216 Time:  1.7806172370910645 s
Testing Loss:  0.201253568019456
Testing Accuracy:  96.27 %
========================================
```

Multi-layer Perceptron example using PyTorch.

Step 8: Visualization

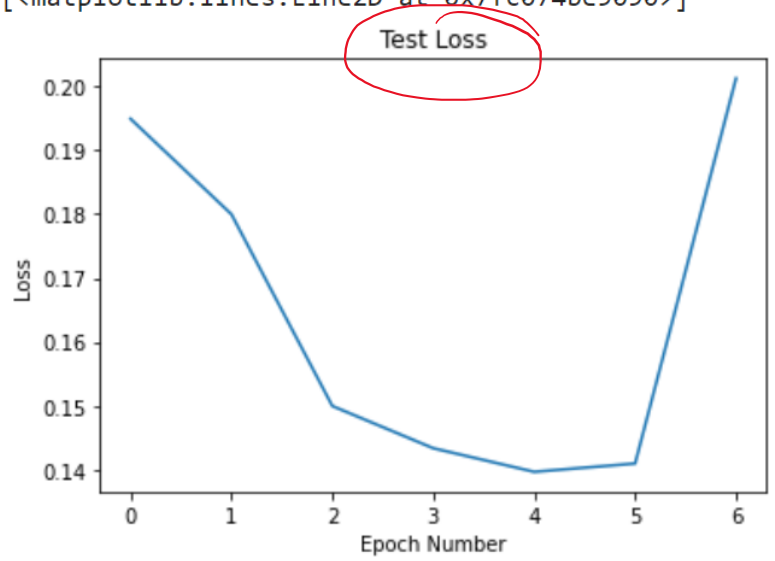
```
plt.title('Training Loss')
plt.xlabel('Epoch Number')
plt.ylabel('Loss')
plt.plot(Train_loss)
```

```
[<matplotlib.lines.Line2D at 0x7fc074c215d0>]
```



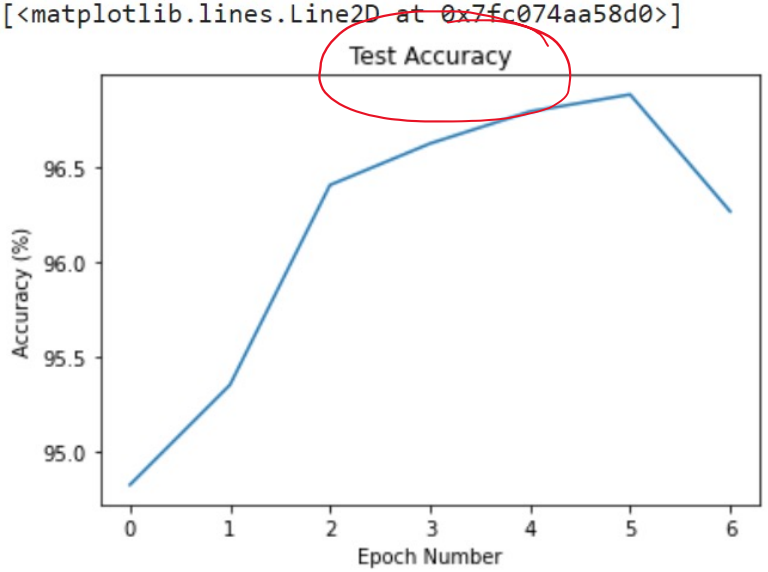

```
plt.title('Test Loss')
plt.xlabel('Epoch Number')
plt.ylabel('Loss')
plt.plot(Test_loss)
```

```
[<matplotlib.lines.Line2D at 0x7fc074be9090>]
```




```
plt.title('Test Accuracy')
plt.xlabel('Epoch Number')
plt.ylabel('Accuracy (%)')
plt.plot(Test_acc)
```

```
[<matplotlib.lines.Line2D at 0x7fc074aa58d0>]
```



Convolution Neural Network using TensorFlow (Keras)

Step 1: Import libraries and Load Dataset

```
# Import Tensorflow
import tensorflow as tf

# Import Matplotlib python plot as plt
import matplotlib.pyplot as plt

# Importing the required Keras modules -- model and layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D

# Load the internally available MNIST dataset.
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

→ Download the MNIST dataset from Keras.

```
# Print the shape of Train & Test Dataset
print('[Train Data Information]')
print(' - Train Data:', x_train.shape)
print(' - Train Labels:', y_train.shape)

print('\n[Test Data Information]')
print(' - Test Data:', x_test.shape)
print(' - Test Labels:', y_test.shape)
```

```
[Train Data Information]
 - Train Data: (60000, 28, 28)
 - Train Labels: (60000,)

[Test Data Information]
 - Test Data: (10000, 28, 28)
 - Test Labels: (10000,)
```

Convolution Neural Network using TensorFlow (Keras)

Step 2: Transformations / Reshaping inputs for TF

```
# Reshaping the array to 4-dims so that it can work with the Keras API
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
input_shape = (28, 28, 1)
# Making sure that the values are float so that we can get decimal points after division
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
# Normalizing the RGB codes by dividing it to the max RGB value.
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print('Number of images in x_train', x_train.shape[0])
print('Number of images in x_test', x_test.shape[0])
```

Single channel as it is a grayscale image.

convert to float 32

```
x_train shape: (60000, 28, 28, 1)
Number of images in x_train 60000
Number of images in x_test 10000
```

Convolution Neural Network using TensorFlow (Keras)

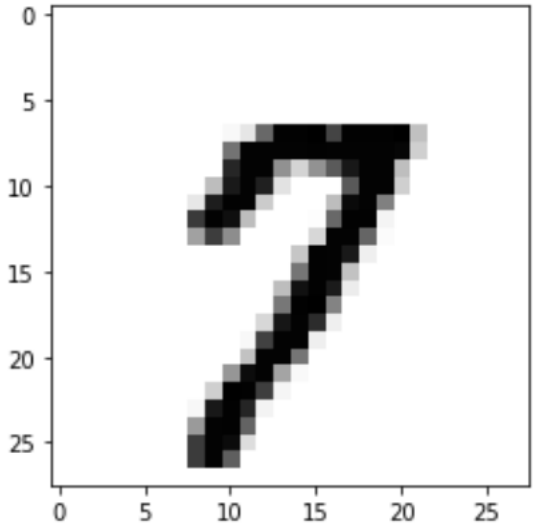
Step 3: Visualize Dataset sample inputs

```
# Plotting a random sample from the input and output label - Train set.
#%matplotlib inline # Only use this if using iPython

image_index = 18819 # You may select anything up to 60,000

print(f"Train Label for index {image_index}:", y_train[image_index])
plt.imshow(x_train[image_index], cmap='Greys')
```

```
Train Label for index 18819: 7
<matplotlib.image.AxesImage at 0x7f75c69eec90>
```



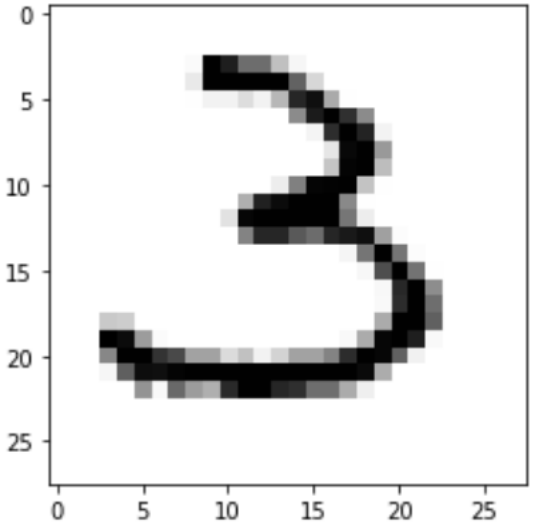
Training sample

```
# Plotting a random sample from the input and output label - Train set.
#%matplotlib inline # Only use this if using iPython

image_index = 819 # You may select anything up to 10,000

print(f"Test Label for index {image_index}:", y_test[image_index])
plt.imshow(x_test[image_index], cmap='Greys')
```

```
Test Label for index 819: 3
<matplotlib.image.AxesImage at 0x7f75c6a11110>
```



Testing sample

Convolution Neural Network using TensorFlow (Keras)

Step 4: Model Creation

Constructor for running certain layers sequentially.

```
# Creating a Sequential Model and adding the layers
model = Sequential()
model.add(Conv2D(28, kernel_size=(3,3), input_shape=input_shape))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten()) # Flattening the 2D arrays for fully connected layers

model.add(Dense(128, activation=tf.nn.relu))

model.add(Dropout(0.2))

model.add(Dense(10,activation=tf.nn.softmax))
```

① →

$$\text{Output shape} = \left[\frac{W-K+2P}{S}\right]+1$$

$$\text{output shape}^{①} = \left[\frac{28-3+2\times0}{1}\right]+1 = \underline{26}$$

Convolution Neural Network using TensorFlow (Keras)

Step 4.1: Model Summary

```
# Print the model architecture
  model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 28)        280

 max_pooling2d (MaxPooling2D  (None, 13, 13, 28)       0
 )

 flatten (Flatten)           (None, 4732)              0

 dense (Dense)               (None, 128)               605824

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 607,394
Trainable params: 607,394
Non-trainable params: 0
_________________________________________________________________
```

Convolution Neural Network using TensorFlow (Keras)

Step 5: Training the Model with validation split

```
#Training the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(x=x_train,y=y_train,validation_split = 0.1, epochs=7)
```

→ computes crossentropy loss b/w the labels and predictions

→ if validation split is not added the val-loss & val-accuracy are not computed

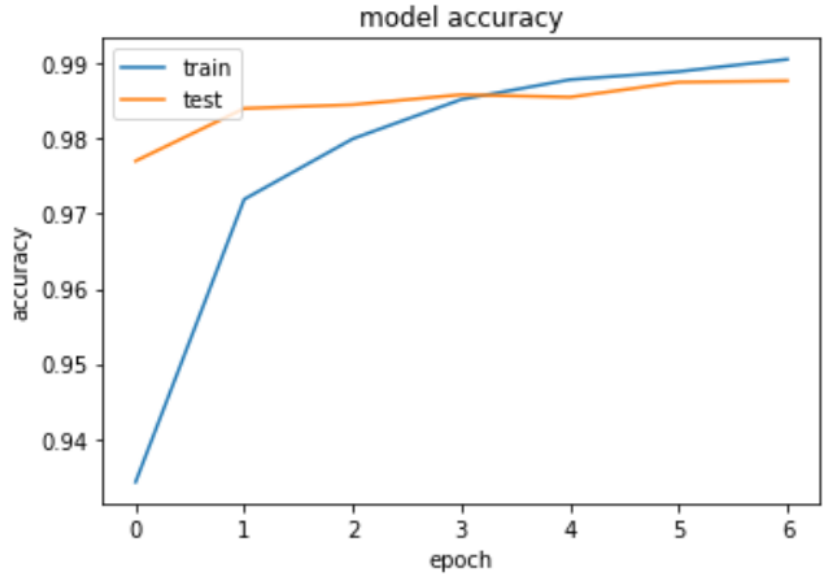
```
Epoch 1/7
1688/1688 [==============================] - 6s 3ms/step - loss: 0.2195 - accuracy: 0.9343 - val_loss: 0.0802 - val_accuracy: 0.9770
Epoch 2/7
1688/1688 [==============================] - 5s 3ms/step - loss: 0.0887 - accuracy: 0.9719 - val_loss: 0.0563 - val_accuracy: 0.9840
Epoch 3/7
1688/1688 [==============================] - 5s 3ms/step - loss: 0.0627 - accuracy: 0.9800 - val_loss: 0.0529 - val_accuracy: 0.9845
Epoch 4/7
1688/1688 [==============================] - 5s 3ms/step - loss: 0.0453 - accuracy: 0.9852 - val_loss: 0.0557 - val_accuracy: 0.9858
Epoch 5/7
1688/1688 [==============================] - 5s 3ms/step - loss: 0.0376 - accuracy: 0.9878 - val_loss: 0.0565 - val_accuracy: 0.9855
Epoch 6/7
1688/1688 [==============================] - 6s 3ms/step - loss: 0.0322 - accuracy: 0.9889 - val_loss: 0.0554 - val_accuracy: 0.9875
Epoch 7/7
1688/1688 [==============================] - 5s 3ms/step - loss: 0.0278 - accuracy: 0.9905 - val_loss: 0.0589 - val_accuracy: 0.9877
```

history stores → ① loss, ② accuracy, ③ val-loss, ④ val-accuracy.

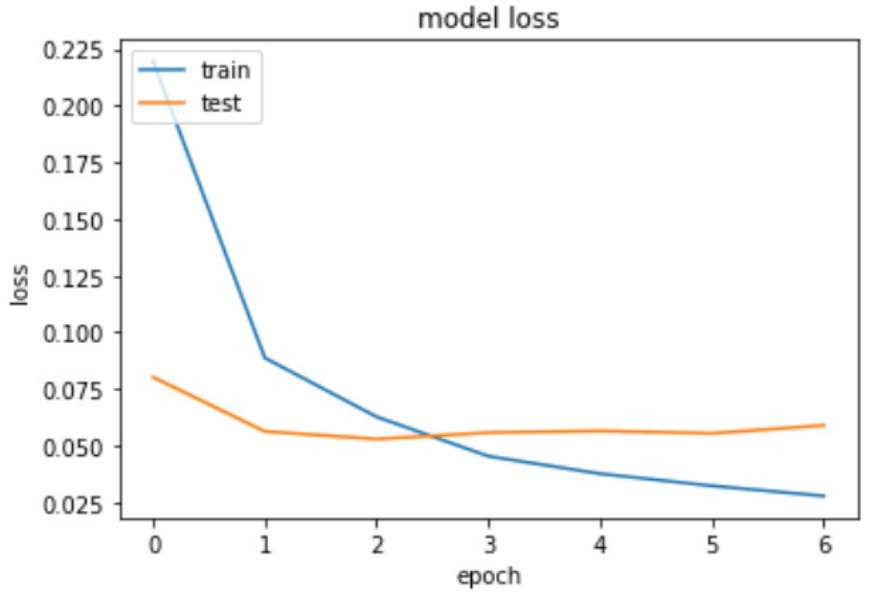
Convolution Neural Network using TensorFlow (Keras)

Step 6: Visualization

```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```




```
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Convolution Neural Network using TensorFlow (Keras)

Step 7: Inference & Manual Validation

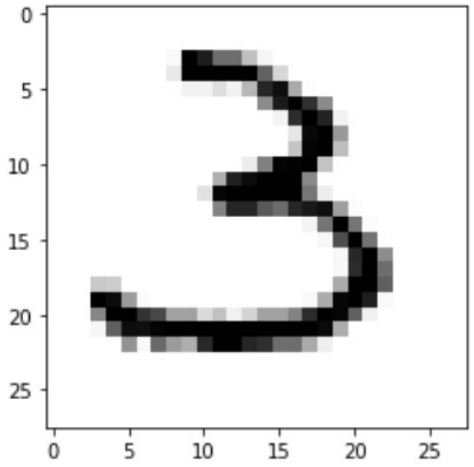
```
# Evaluate Model on the Test dataset
model.evaluate(x_test, y_test)
```

→ Inference on Test set (unseen data)

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0543 - accuracy: 0.9843
[0.05426904186606407, 0.9843000173568726]
```

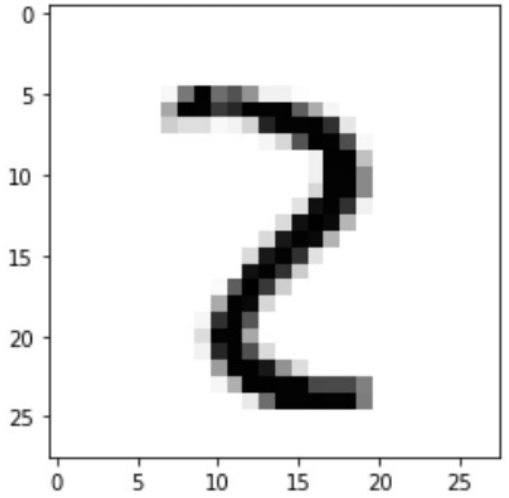
```
# Displaying output for after running inference - Inputs shuffled
image_index = 819
plt.imshow(x_test[image_index].reshape(28, 28),cmap='Greys')
pred = model.predict(x_test[image_index].reshape(1, 28, 28, 1))
print(f"Test Label for index {image_index}:", pred.argmax())
```

```
Test Label for index 819: 3
```



```
# Displaying output for after running inference - Inputs shuffled
image_index = 741
plt.imshow(x_test[image_index].reshape(28, 28),cmap='Greys')
pred = model.predict(x_test[image_index].reshape(1, 28, 28, 1))
print(f"Test Label for index {image_index}:", pred.argmax())
```

```
Test Label for index 741: 2
```



CNN – 0.9843
MLP – 0.9689

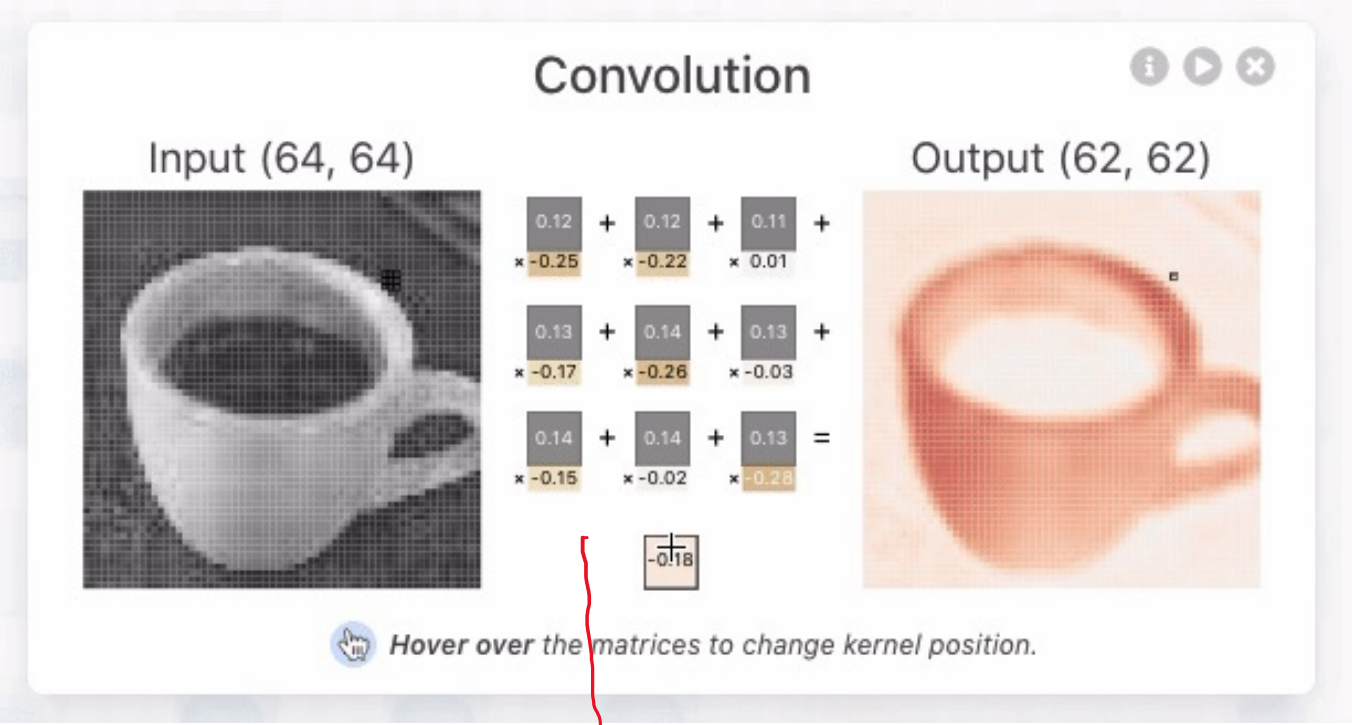
Convolution Neural Network (CNN) Interactive Visualizer

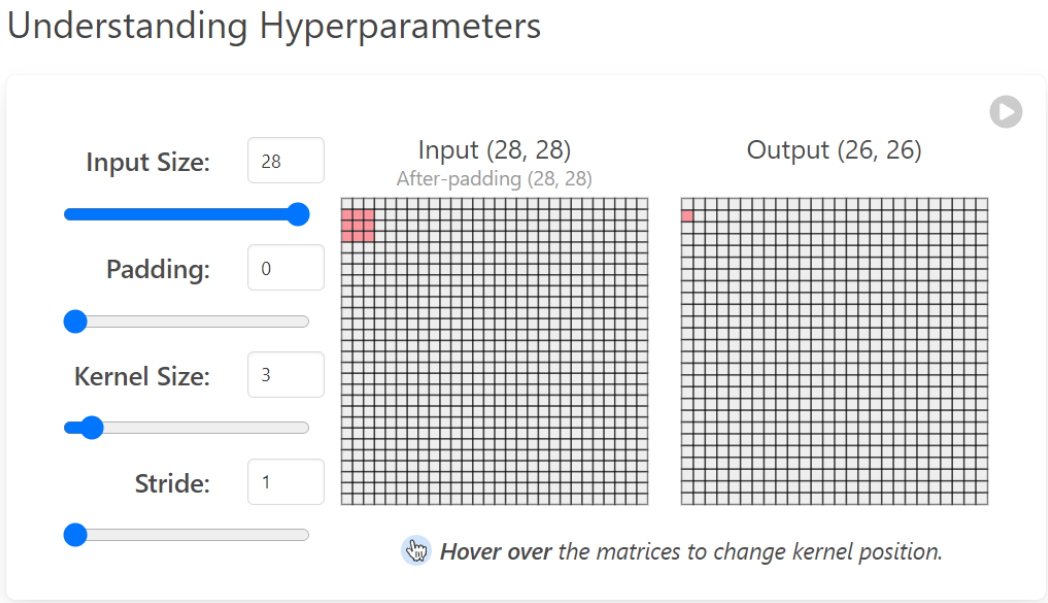
https://poloclub.github.io/cnn-explainer/

Paper: CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization

YouTube: Demo Video: CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization

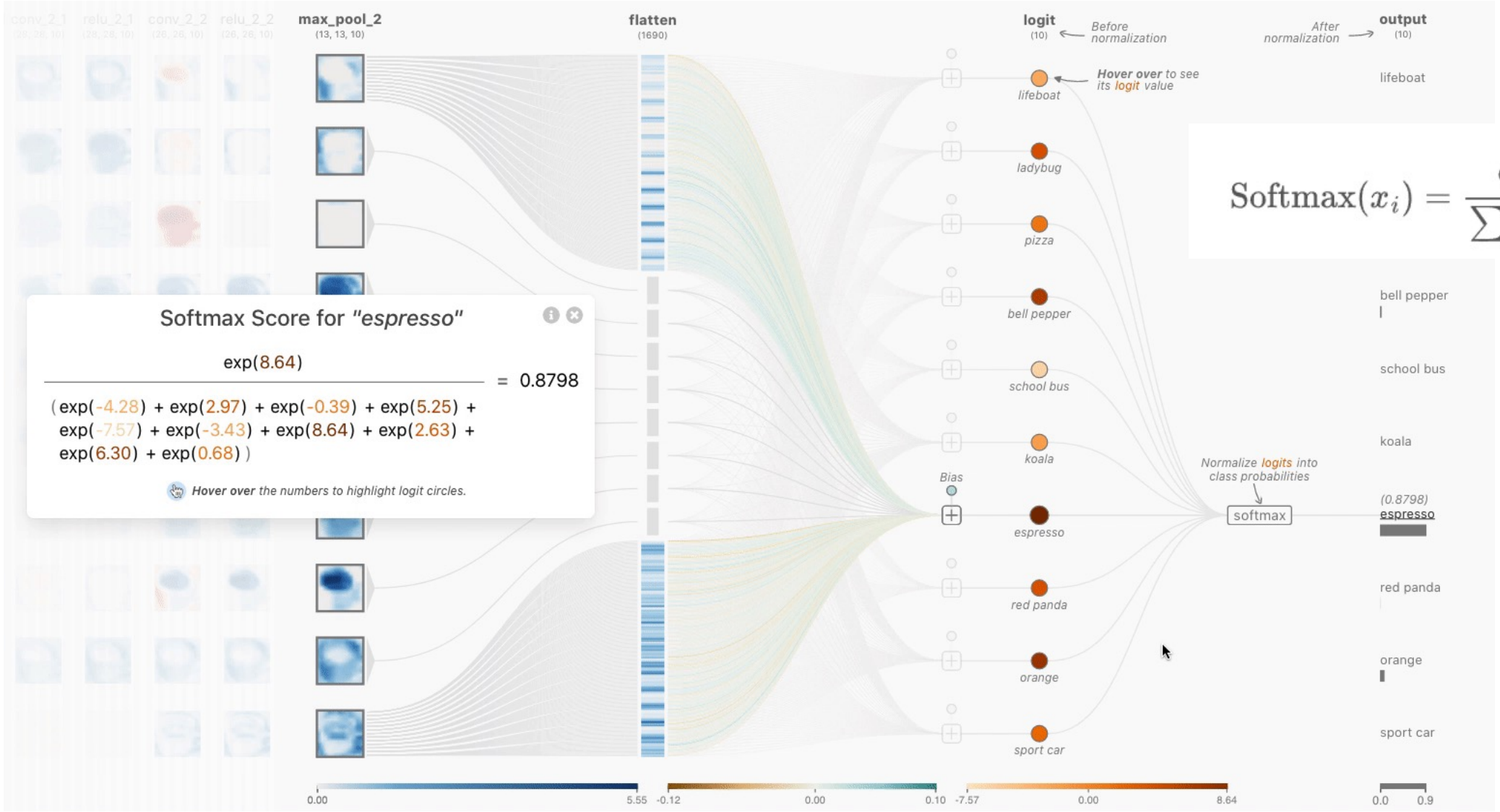
Convolution Neural Network (CNN) Interactive Visualizer





filter.

Convolution Neural Network (CNN) Interactive Visualizer



CNN EXPLAINER

Learning Convolutional Neural Networks with Interactive Visualization

Zijie J. Wang[1], Robert Turko[1], Omar Shaikh[1], Haekyu Park[1], Nilaksh Das[1], Fred Hohman[1], Minsuk Kahng[2], and Polo Chau[1]

[1] Georgia Tech

[2] Oregon State University

Summary

- Reviewed Image Classification using MNIST Dataset
  - MNIST Dataset Description
  - Multi-layer Perceptron example using PyTorch with code.
  - Convolution Neural Network example using TensorFlow (Keras) with code.

- Reviewed CNN Visualizer
  - https://poloclub.github.io/cnn-explainer