```
url = https://github.com/f00-/mnist-lenet-keras.git



# mnist-lenet-keras

from http://www.pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python/

## Install
```

  git clone https://github.com/f00-/mnist-lenet-keras.git
  cd mnist-lenet-keras
  pip install -r requirements.txt
```

## Usage
```

  python mnist.py
```

## Example Output
![Example Output](http://i.imgur.com/IqJeJKY.png)
h5py==2.6.0
Keras==1.2.0
numpy==1.11.3
PyYAML==3.12
scikit-learn==0.18.1
scipy==0.18.1
six==1.10.0
sklearn==0.0
Theano==0.8.2
from keras.models import Sequential
from keras.layers.convolutional import Convolution2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dense

class LeNet:
    @staticmethod
    def build(width, height, depth, classes,
weightsPath=None):
        # initialize the model
        model = Sequential()

        # first set of CONV => RELU => POOL
        model.add(Convolution2D(20, 5, 5,
border_mode="same",
            input_shape=(depth, height, width)))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2),
strides=(2, 2)))

        # second set of CONV => RELU => POOL
        model.add(Convolution2D(50, 5, 5,
border_mode="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2),
strides=(2, 2)))

        # set of FC => RELU layers
        model.add(Flatten())
        model.add(Dense(500))
        model.add(Activation("relu"))

        # softmax classifier
        model.add(Dense(classes))
        model.add(Activation("softmax"))

        # if weightsPath is specified load the weights
        if weightsPath is not None:
            model.load_weights(weightsPath)

        return model
from lenet import LeNet
from sklearn.cross_validation import train_test_split
from sklearn import datasets
from keras.optimizers import SGD
from keras.utils import np_utils
import numpy as np
import cv2

weightsPath = "weights/lenet_weights.hdf5"

print("downloading MNIST...")
dataset = datasets.fetch_mldata("MNIST Original")

# reshape the MNIST dataset from a flat list of 784-
dim vectors, to
# 28 x 28 pixel images, then scale the data to the
range [0, 1.0]
# and construct the training and testing splits
data = dataset.data.reshape((dataset.data.shape[0],
28, 28))
data = data[:, np.newaxis, :, :]
(trainData, testData, trainLabels, testLabels) =
train_test_split(
        data / 255.0, dataset.target.astype("int"),
test_size=0.33)

# transform the training and testing labels into
vectors in the
# range [0, classes] -- this generates a vector for
each label,
# where the index of the label is set to `1` and all
other entries
# to `0`; in the case of MNIST, there are 10 class
labels
trainLabels = np_utils.to_categorical(trainLabels, 10)
testLabels = np_utils.to_categorical(testLabels, 10)

# initialize the optimizer and model
print("[INFO] compiling model...")
opt = SGD(lr=0.01)
```

```
model = LeNet.build(width=28, height=28, depth=1,
classes=10,
        weightsPath=weightsPath)
model.compile(loss="categorical_crossentropy",
optimizer=opt,
        metrics=["accuracy"])

# if no weights specified train the model
if weightsPath is None:
        print("[INFO] training...")
        model.fit(trainData, trainLabels,
batch_size=128, nb_epoch=20,
                verbose=1)

        # show the accuracy on the testing set
        print("[INFO] evaluating...")
        (loss, accuracy) = model.evaluate(testData,
testLabels,
                batch_size=128, verbose=1)
        print("[INFO] accuracy: {:.2f}
%".format(accuracy * 100))

        print("[INFO] dumping weights to file...")
        model.save_weights(weightsPath, overwrite=True)

# randomly select a few testing digits
for i in np.random.choice(np.arange(0,
len(testLabels)), size=(10,)):
        # classify the digit
        probs = model.predict(testData[np.newaxis, i])
        prediction = probs.argmax(axis=1)

        # resize the image from a 28 x 28 to 96 x 96
        image = (testData[i][0] * 255).astype("uint8")
        image = cv2.merge([image] * 3)
        image = cv2.resize(image, (96, 96),
interpolation=cv2.INTER_LINEAR)
        cv2.putText(image, str(prediction[0]), (5, 20),
                cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0,
255, 0), 2)

        # show the image and prediction
        print("[INFO] Predicted: {}, Actual:
{}".format(prediction[0],
                np.argmax(testLabels[i])))
        cv2.imshow("Digit", image)
        cv2.waitKey(0)
```