MNIST Classification – Dataset Augmentation by Affine Transformations on Training Images

Kiran Mohan September 10th 2016 Instructor: Prof. Jacob Whitehill

Network Architecture

The network architecture[1] was chosen to have two sets of convolutional - max pooling layers. The first convolutional layer was made up of 64 filters while the second was built with 128 filters. Each filter was 5x5 pixels in size. The max-pooling operation was done with a filter size of 2x2. Rectified Linear Units (ReLU) was used as the activation function after the convolution operation. After the two convolution-pooling layers, two fully connected layers were included to provide the final predictions. The first fully connected layer was built with 1024 nodes while the second was built with 10 nodes, one for each class. The output of each of these 10 nodes gives the probability of the current input belonging to each of the 10 classes.

Sample Input Images

The following 5 images are raw samples of input training images obtained without any processing. No rotation, translation or scaling has been applied to these images. There are 60,000 such images in the dataset, each having a resolution of 28 x 28 pixels.

Affine Transformations

Affine transformations are transformations that retain the collinearity property. In particular, points that were lying on a straight line in the input image would still lie on a straight line in the transformed image. Three types of affine transformations were applied to the input images and the transformed images were appended to the dataset. Thus, the amount of training data is increased and such a technique is called data augmentation. Data augmentation is a standard technique used by the Machine Learning community to achieve increase in accuracy. The number of augmented data was treated as a hyperparameter ∈ {0,10000,20000,30000,40000,50000,60000}. Testing accuracy, time taken to apply transformations and training time were recorded and plotted for each case.

Translated Images

The three images above are images translated from their center by ±3 pixels in their X and Y directions.

Rotated Images

The above three images are rotated versions of randomly sampled input images. Rotations were restricted to [-15,+15] in order to maintain sanity of image representations.

Scaled Images

The images displayed above are scaled images truncated to a 28 x 28 window. The truncation was done to maintain consistency in size of images input into the convolutional neural network.

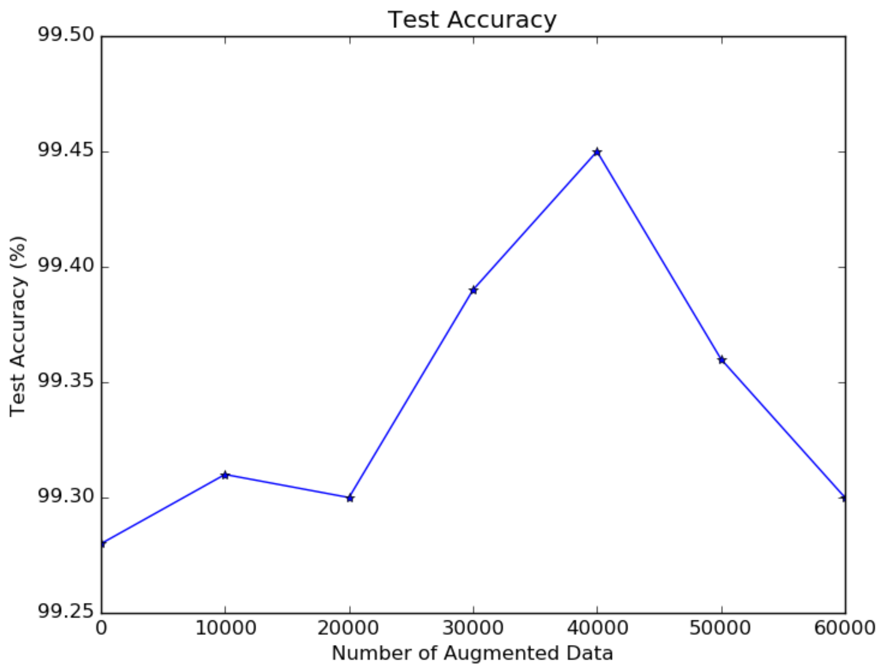
Results & Analysis

The following table illustrates the variation of test accuracy, transformation time and training time with increase in number of transformed, augmented data.

#Augmented Data	#Total training data	% Test Accuracy	Transformation time (s)	Training time (s)
0	60000	99.28%	0	756
10000	70000	99.31%	744	886
20000	80000	99.30%	1599	998
30000	90000	99.39%	2569	1118
40000	100000	99.45%	3659	1240
50000	110000	99.36%	4849	1365
60000	120000	99.30%	6152	1482

Accuracy Plot

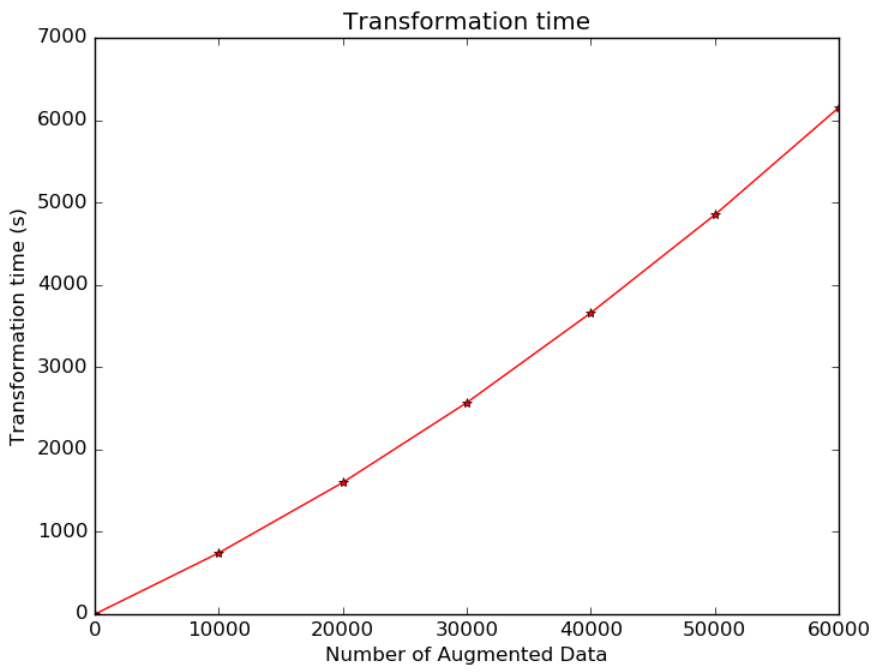
The plot below shows the change in accuracy with increase in number of data augmented after applying affine transformations.



From the plot, it is evident that the test accuracy increases with increase in augmented data up to a break-point and then starts decreasing. In this case, the break-point is 40000 augmented data. We get a maximum of 99.45% accuracy which is 0.17% higher than 99.28% which is the accuracy obtained without data augmentation.

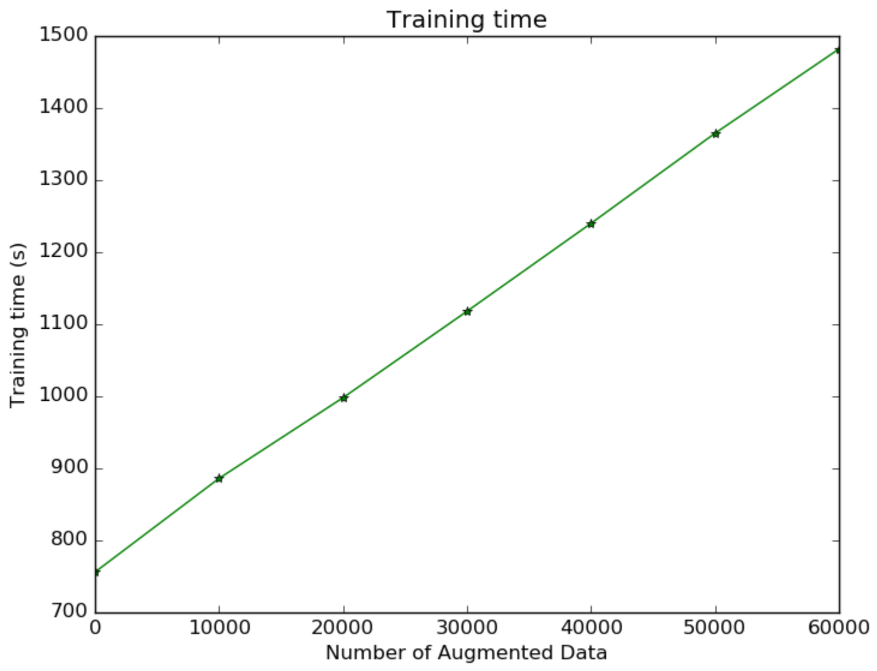
Transformation Time

The transformation time plot seems to vary almost linearly with increase in number of augmented data. This is logical because the time taken to augment a single image is almost a constant. Hence, an increase in the number of augmented data causes a linear increase in the time taken for transformations. We also see that for number of augmented data > 10000, the time taken for transformations is higher than the time taken for training the network.



Training Time

The training time also varies linearly with increase in number of augmented data as seen from the plot below.



References

1. K. Mohan, "MNIST Classification using ConvNets on CPU", Report sent by email on 08/26/2016.
2. Partial code credit: https://github.com/fchollet/keras/blob/master/examples/mnist_cnn.py

Appendix – Code[2]

```
from __future__ import print_function
import numpy as np
np.random.seed(1337)  # for reproducibility
import cv2
import random
from time import time

from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras.utils import np_utils
from keras import backend as K

time_beginning = time()

batch_size = 128
nb_classes = 10
nb_epoch = 12

# input image dimensions
```

```
img_rows, img_cols = 28, 28
# number of convolutional filters to use
nb_filters1 = 64
nb_filters2 = 128
# size of pooling area for max pooling
pool_size = (2, 2)
# convolution kernel size
kernel_size = (5, 5)

time_start_load_data = time()

# the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()

if K.image_dim_ordering() == 'th':
    X_train = X_train.reshape(X_train.shape[0], 1, img_rows, img_cols)
    X_test = X_test.reshape(X_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
    X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

time_end_load_data = time()
print("Time taken to load data: ",time_start_load_data -
time_end_load_data)

time_start_transformations = time()
# Randomly rotate, translate and scale half of the training images to
achieve invariance
k = random.sample(range(60000),40000)
for j in range(len(k)):
  print(j)
  warpType = random.randint(1,3) # 1=Translation; 2=Rotation;
3=Scaling
  if warpType == 1: # Translation
    print("Translation")
    randTransX, randTransY = random.randint(0,6)-3,
random.randint(0,6)-3
    print("randTransX = {}".format(randTransX))
    print("randTransY = {}".format(randTransY))
    M = np.float32([[1,0,randTransX],[0,1,randTransY]])
    img = cv2.warpAffine(X_train[k[j]][0],M,(28,28))
    cv2.imwrite('X_trainTR.png',img[:28,:28])
    X_train = 
np.append(X_train,img[:28,:28]).reshape(X_train.shape[0]+1,1,28,28)

  elif warpType == 2: # Rotation
    print("Rotation")
    randRot = random.uniform(-15.0,15.0)
    print("randRot = {}".format(randRot))
    M = cv2.getRotationMatrix2D((14,14),randRot,1)
    img = cv2.warpAffine(X_train[k[j]][0],M,(28,28))
    cv2.imwrite('X_trainRot.png',img[:28,:28])
    X_train = 
np.append(X_train,img[:28,:28]).reshape(X_train.shape[0]+1,1,28,28)

  else: # Scaling
    print("Scaling")
    randScale = random.uniform(1,1.75)
    print("randScale = {}".format(randScale))
    img = cv2.resize(X_train[k[j]][0],None,fx=randScale, fy=randScale, 
interpolation = cv2.INTER_CUBIC)
    cv2.imwrite('X_trainSC.png',img[:28,:28])
    X_train = 
np.append(X_train,img[:28,:28]).reshape(X_train.shape[0]+1,1,28,28)

  y_train = np.append(y_train,y_train[k[j]])

time_end_transformations = time()
print("Time taken for transformations: ",time_end_transformations - 
time_start_transformations)

X_train /= 255
X_test /= 255
print('X_train shape:', X_train.shape)
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
Y_train = np_utils.to_categorical(y_train, nb_classes)
Y_test = np_utils.to_categorical(y_test, nb_classes)

time_start_model_build = time()

model = Sequential()

model.add(Convolution2D(nb_filters1, kernel_size[0], kernel_size[1],
                        border_mode='valid',
                        input_shape=input_shape))
model.add(Activation('relu'))
model.add(Convolution2D(nb_filters2, kernel_size[0], kernel_size[1]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=pool_size))

model.add(Flatten())
model.add(Dense(1024))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

time_end_model_build = time()
print("Time taken to build model: ",time_end_model_build -
time_start_model_build)

time_start_training = time()

model.fit(X_train, Y_train, batch_size=batch_size, nb_epoch=nb_epoch,
          verbose=1, validation_data=(X_test, Y_test))
time_end_training = time()
print("Time taken for training: ",time_end_training -
time_start_training)

time_start_testing = time()
score = model.evaluate(X_test, Y_test, verbose=0)
time_end_testing = time()
print("Time for testing: ",time_end_testing-time_start_testing)

print('Test score:', score[0])
print('Test accuracy:', score[1])

time_end = time()
print("Time taken for entire program to execute: ",time_end -
time_beginning)
```