CNN on MNIST Using Keras

Model1: 7x7 Kernel Matrix

In [1]:

```
# Credits: https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py


from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
from keras.layers.normalization import BatchNormalization

batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    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')
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 = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model1 = Sequential()

model1.add(Conv2D(16, kernel_size=(7, 7),activation='relu',input_shape=input_shape))
model1.add(BatchNormalization())
model1.add(Dropout(0.25))

model1.add(Conv2D(32, kernel_size=(7, 7), activation='relu'))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(BatchNormalization())
model1.add(Dropout(0.25))

model1.add(Conv2D(64, kernel_size=(7, 7), activation='relu'))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(BatchNormalization())
model1.add(Dropout(0.25))
```

```
model1.add(Flatten())
model1.add(Dense(128, activation='relu'))
model1.add(BatchNormalization())
model1.add(Dropout(0.7))
model1.add(Dense(num_classes, activation='softmax'))

model1.summary()



model1.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history = model1.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = model1.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Using TensorFlow backend.
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.
py:516: FutureWarning: Passing (type, 1) or '1type' as a synonym of type i
s deprecated; in a future version of numpy, it will be understood as (typ
e, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.
py:517: FutureWarning: Passing (type, 1) or '1type' as a synonym of type i
s deprecated; in a future version of numpy, it will be understood as (typ
e, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.
py:518: FutureWarning: Passing (type, 1) or '1type' as a synonym of type i
s deprecated; in a future version of numpy, it will be understood as (typ
e, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.
py:519: FutureWarning: Passing (type, 1) or '1type' as a synonym of type i
s deprecated; in a future version of numpy, it will be understood as (typ
e, (1,)) / '(1,)type'.
```

In [2]:

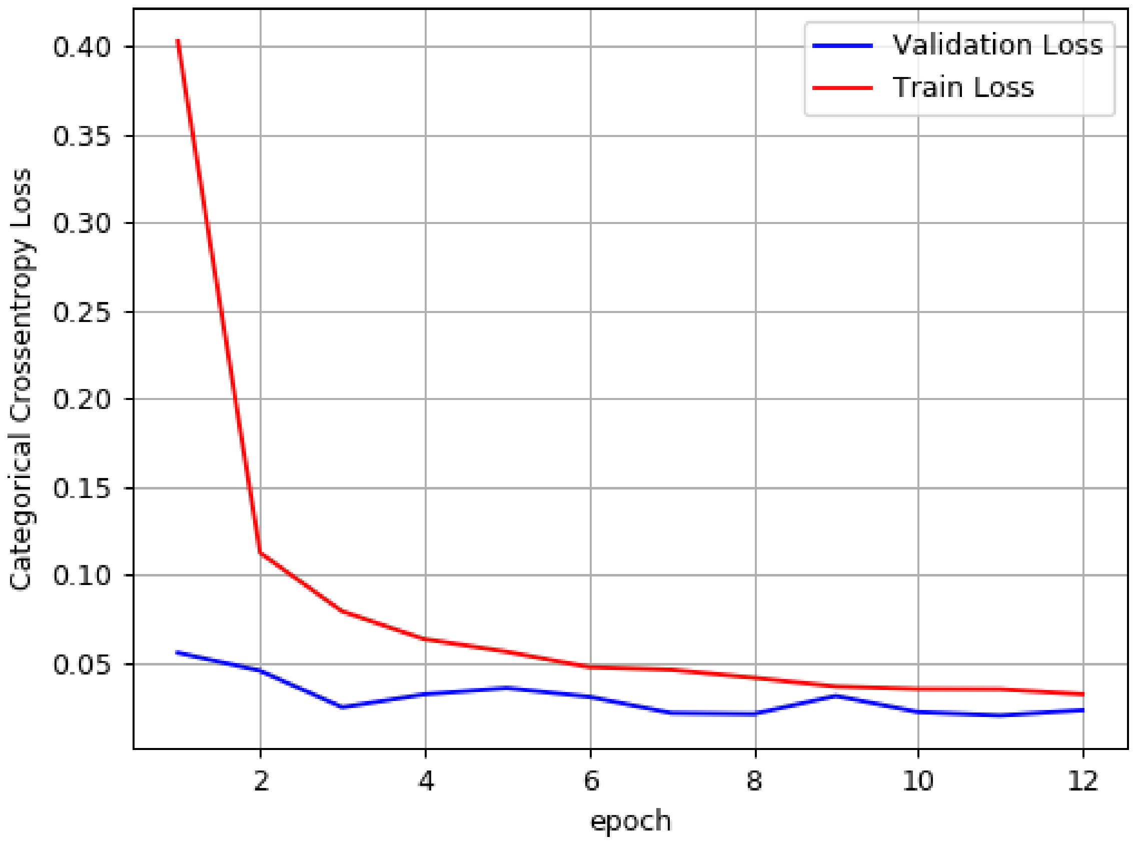
```
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
    plt.show()
```

In [3]:

```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,epochs+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
plt.show()
```

Figure 1



x=3.89073 y=0.3

Model2: 3x3 Kernel Matrix

In [34]:

```
model2 = Sequential()
model2.add(Conv2D(16, kernel_size=(3, 3),activation='relu',input_shape=input_shape))
model2.add(BatchNormalization())

model2.add(Conv2D(16, kernel_size=(3, 3),strides= 2,activation='relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.25))


model2.add(Conv2D(32, kernel_size=(3, 3),strides= 2, activation='relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(BatchNormalization())
model2.add(Dropout(0.25))



model2.add(Flatten())
model2.add(Dense(128, activation='relu'))
model2.add(BatchNormalization())
model2.add(Dropout(0.7))
model2.add(Dense(num_classes, activation='softmax'))

model2.summary()



model2.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history = model2.fit(x_train, y_train,batch_size=batch_size, epochs=epochs, verbose=1, vali
score = model2.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
W1107 23:05:26.520308 139831160698624 nn_ops.py:4224] Large dropout rate: 0.
7 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate instead of keep_pro
b. Please ensure that this is intended.

_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_58 (Conv2D)           (None, 26, 26, 16)        160       
_________________________________________________________________
batch_normalization_32 (Batc (None, 26, 26, 16)        64        
_________________________________________________________________
conv2d_59 (Conv2D)           (None, 12, 12, 16)        2320      
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 6, 6, 16)          0         
_________________________________________________________________
dropout_41 (Dropout)         (None, 6, 6, 16)          0         
_________________________________________________________________
conv2d_60 (Conv2D)           (None, 2, 2, 32)          4640      
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 1, 1, 32)          0         
_________________________________________________________________
batch_normalization_33 (Batc (None, 1, 1, 32)          128       
_________________________________________________________________
```

```
dropout_42 (Dropout)         (None, 1, 1, 32)          0         
_________________________________________________________________
flatten_5 (Flatten)          (None, 32)                0         
_________________________________________________________________
dense_9 (Dense)              (None, 128)               4224      
_________________________________________________________________
batch_normalization_34 (Batc (None, 128)               512       
_________________________________________________________________
dropout_43 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_10 (Dense)             (None, 10)                1290      
=================================================================
Total params: 13,338
Trainable params: 12,986
Non-trainable params: 352
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 61s 1ms/step - loss: 1.3415 -
acc: 0.6000 - val_loss: 0.2235 - val_acc: 0.9360
Epoch 2/12
60000/60000 [==============================] - 33s 556us/step - loss: 0.4355
- acc: 0.8667 - val_loss: 0.1171 - val_acc: 0.9639
Epoch 3/12
60000/60000 [==============================] - 36s 594us/step - loss: 0.3061
- acc: 0.9077 - val_loss: 0.1012 - val_acc: 0.9691
Epoch 4/12
60000/60000 [==============================] - 64s 1ms/step - loss: 0.2563 -
acc: 0.9240 - val_loss: 0.0915 - val_acc: 0.9715
Epoch 5/12
60000/60000 [==============================] - 79s 1ms/step - loss: 0.2316 -
acc: 0.9318 - val_loss: 0.0756 - val_acc: 0.9752
Epoch 6/12
60000/60000 [==============================] - 27s 445us/step - loss: 0.2033
- acc: 0.9388 - val_loss: 0.0695 - val_acc: 0.9783
Epoch 7/12
60000/60000 [==============================] - 21s 346us/step - loss: 0.1940
- acc: 0.9434 - val_loss: 0.0647 - val_acc: 0.9788
Epoch 8/12
60000/60000 [==============================] - 51s 858us/step - loss: 0.1865
- acc: 0.9458 - val_loss: 0.0537 - val_acc: 0.9822
Epoch 9/12
60000/60000 [==============================] - 25s 414us/step - loss: 0.1754
- acc: 0.9481 - val_loss: 0.0559 - val_acc: 0.9831
Epoch 10/12
60000/60000 [==============================] - 23s 383us/step - loss: 0.1722
- acc: 0.9492 - val_loss: 0.0561 - val_acc: 0.9811
Epoch 11/12
60000/60000 [==============================] - 22s 364us/step - loss: 0.1684
- acc: 0.9512 - val_loss: 0.0548 - val_acc: 0.9832
Epoch 12/12
60000/60000 [==============================] - 61s 1ms/step - loss: 0.1644 -
acc: 0.9514 - val_loss: 0.0495 - val_acc: 0.9843
Test loss: 0.049488892110204324
Test accuracy: 0.9843
```

In [ ]:

```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,epochs+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
plt.show()
```

In [ ]:

```
# Model3: 2x2 Kernel Matrix
```

In [13]:

```
model3 = Sequential()

model3.add(Conv2D(4, kernel_size=(2, 2),activation='relu',input_shape=input_shape))
model3.add(Dropout(0.25))

model3.add(Conv2D(8, kernel_size=(2, 2),activation='relu'))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))

model3.add(Conv2D(16, kernel_size=(2, 2),activation='relu'))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))

model3.add(Conv2D(32, kernel_size=(2, 2), activation='relu'))
model3.add(BatchNormalization())
model3.add(Dropout(0.25))

model3.add(Conv2D(64, kernel_size=(2, 2), activation='relu'))
model3.add(BatchNormalization())
model3.add(Dropout(0.25))


model3.add(Flatten())
model3.add(Dense(128, activation='relu'))
model3.add(BatchNormalization())
model3.add(Dropout(0.7))
model3.add(Dense(num_classes, activation='softmax'))

model3.summary()



model3.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history = model3.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = model3.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
W1107 22:11:50.141709 139831160698624 nn_ops.py:4224] Large dropout rate: 0.
7 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate instead of keep_pro
b. Please ensure that this is intended.


_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_9 (Conv2D)            (None, 27, 27, 4)         20        
_________________________________________________________________
dropout_11 (Dropout)         (None, 27, 27, 4)         0         
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 26, 26, 8)         136       
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 13, 13, 8)         0         
_________________________________________________________________
dropout_12 (Dropout)         (None, 13, 13, 8)         0         
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 12, 12, 16)        528       
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 6, 6, 16)          0         
_________________________________________________________________
dropout_13 (Dropout)         (None, 6, 6, 16)          0         
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 5, 5, 32)          2080      
_________________________________________________________________
batch_normalization_8 (Batch (None, 5, 5, 32)          128       
_________________________________________________________________
dropout_14 (Dropout)         (None, 5, 5, 32)          0         
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 4, 4, 64)          8256      
_________________________________________________________________
batch_normalization_9 (Batch (None, 4, 4, 64)          256       
_________________________________________________________________
dropout_15 (Dropout)         (None, 4, 4, 64)          0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 1024)              0         
_________________________________________________________________
dense_5 (Dense)              (None, 128)               131200    
_________________________________________________________________
batch_normalization_10 (Batc (None, 128)               512       
_________________________________________________________________
dropout_16 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1290      
=================================================================
Total params: 144,406
Trainable params: 143,958
Non-trainable params: 448
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 96s 2ms/step - loss: 0.8181
- acc: 0.7528 - val_loss: 0.3360 - val_acc: 0.8837
Epoch 2/12
60000/60000 [==============================] - 94s 2ms/step - loss: 0.3727
- acc: 0.8851 - val_loss: 0.3064 - val_acc: 0.9023
Epoch 3/12
60000/60000 [==============================] - 73s 1ms/step - loss: 0.2748
- acc: 0.9157 - val_loss: 0.3712 - val_acc: 0.8905
Epoch 4/12
60000/60000 [==============================] - 74s 1ms/step - loss: 0.2280
- acc: 0.9296 - val_loss: 0.3964 - val_acc: 0.8850
Epoch 5/12
60000/60000 [==============================] - 69s 1ms/step - loss: 0.2043
- acc: 0.9380 - val_loss: 0.2821 - val_acc: 0.9178
Epoch 6/12
60000/60000 [==============================] - 86s 1ms/step - loss: 0.1839
- acc: 0.9436 - val_loss: 0.2029 - val_acc: 0.9378
Epoch 7/12
60000/60000 [==============================] - 83s 1ms/step - loss: 0.1707
- acc: 0.9492 - val_loss: 0.1511 - val_acc: 0.9520
Epoch 8/12
60000/60000 [==============================] - 81s 1ms/step - loss: 0.1612
- acc: 0.9505 - val_loss: 0.2230 - val_acc: 0.9325
Epoch 9/12
```

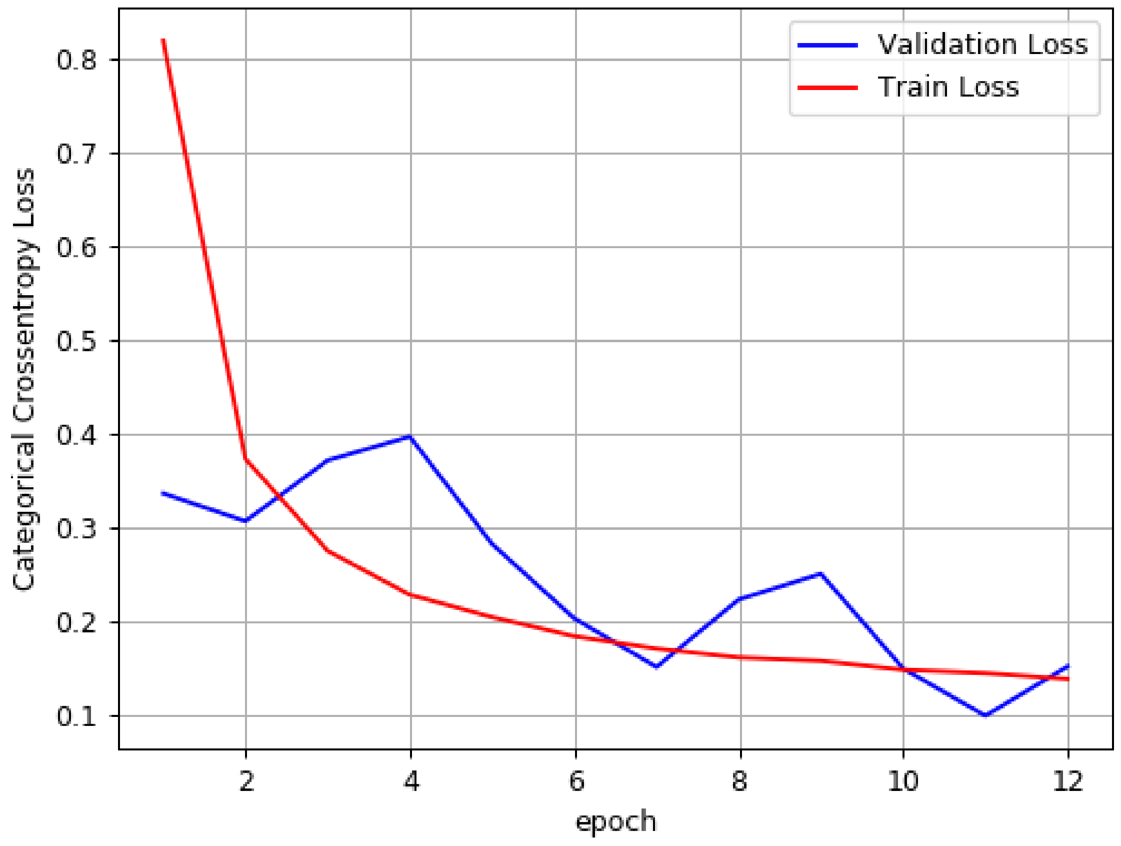
```
60000/60000 [==============================] - 81s 1ms/step - loss: 0.1576
- acc: 0.9522 - val_loss: 0.2503 - val_acc: 0.9284
Epoch 10/12
60000/60000 [==============================] - 74s 1ms/step - loss: 0.1482
- acc: 0.9556 - val_loss: 0.1494 - val_acc: 0.9544
Epoch 11/12
60000/60000 [==============================] - 70s 1ms/step - loss: 0.1446
- acc: 0.9563 - val_loss: 0.0992 - val_acc: 0.9680
Epoch 12/12
60000/60000 [==============================] - 72s 1ms/step - loss: 0.1384
- acc: 0.9593 - val_loss: 0.1517 - val_acc: 0.9535
Test loss: 0.1517157553706318
Test accuracy: 0.9535
```

In [15]:

```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,epochs+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
plt.show()
```

Figure 2



x=1.71956 y=0

In [35]:

```
from prettytable import PrettyTable

#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytab

x = PrettyTable()
x.field_names = ["Kernel Matrix","No.of ConVNetlayers","Test Accuracy", "Test Loss"]

x.add_row(["7x7", 3, 0.9935, 0.0231])
x.add_row(["3x3", 3, 0.984, 0.0494])
x.add_row(["2x2", 5, 0.95, 0.151])

print(x)
```

```
+---------------+---------------------+---------------+-----------+
| Kernel Matrix | No.of ConVNetlayers | Test Accuracy | Test Loss |
+---------------+---------------------+---------------+-----------+
|      7x7      |          3          |     0.9935    |   0.0231  |
|      3x3      |          3          |     0.984     |   0.0494  |
|      2x2      |          5          |      0.95     |   0.151   |
+---------------+---------------------+---------------+-----------+
```

In [ ]: