3 Different Architectures for MNIST using CNN

3X3 +Max pooling+BATCH NORMALISATION+ DROPOUT

In [0]:

```
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.layers import BatchNormalization
from keras import backend as K

batch_size = 128
num_classes = 10
epochs = 10

# 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)

model = Sequential()
model.add(Conv2D(128, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

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

history=model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
```

```
Using TensorFlow backend.
```

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 917s 15ms/step - loss:
0.2504 - acc: 0.9241 - val_loss: 0.0468 - val_acc: 0.9848
Epoch 2/10
60000/60000 [==============================] - 902s 15ms/step - loss:
0.0934 - acc: 0.9723 - val_loss: 0.0361 - val_acc: 0.9882
Epoch 3/10
60000/60000 [==============================] - 899s 15ms/step - loss:
0.0723 - acc: 0.9792 - val_loss: 0.0313 - val_acc: 0.9902
Epoch 4/10
60000/60000 [==============================] - 900s 15ms/step - loss:
0.0622 - acc: 0.9823 - val_loss: 0.0323 - val_acc: 0.9908
Epoch 5/10
60000/60000 [==============================] - 898s 15ms/step - loss:
0.0544 - acc: 0.9839 - val_loss: 0.0248 - val_acc: 0.9919
Epoch 6/10
60000/60000 [==============================] - 910s 15ms/step - loss:
0.0480 - acc: 0.9859 - val_loss: 0.0247 - val_acc: 0.9929
Epoch 7/10
60000/60000 [==============================] - 916s 15ms/step - loss:
0.0465 - acc: 0.9867 - val_loss: 0.0316 - val_acc: 0.9911
Epoch 8/10
60000/60000 [==============================] - 911s 15ms/step - loss:
0.0427 - acc: 0.9880 - val_loss: 0.0240 - val_acc: 0.9933
Epoch 9/10
60000/60000 [==============================] - 901s 15ms/step - loss:
0.0395 - acc: 0.9882 - val_loss: 0.0231 - val_acc: 0.9927
Epoch 10/10
60000/60000 [==============================] - 902s 15ms/step - loss:
0.0376 - acc: 0.9889 - val_loss: 0.0245 - val_acc: 0.9922
```

In [0]:

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

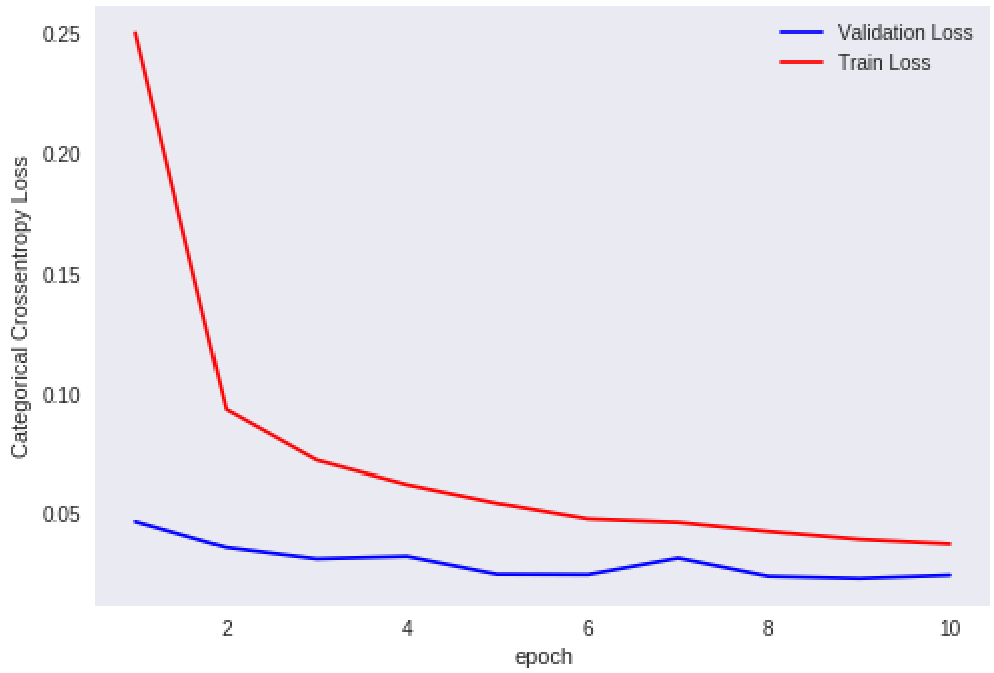
```
Test loss: 0.024462868741243166
Test accuracy: 0.9922
```

In [0]:

```
import matplotlib.pyplot as plt
import numpy as np
import time

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()
```

In [0]:

```
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)
```



5X5 +Max pooling+BATCH NORMALISATION+ DROPOUT

In [0]:

```
epochs = 10
model = Sequential()

model.add(Conv2D(32, (5,5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Conv2D(64, (5,5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history=model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 102s 2ms/step - loss: 0.
2489 - acc: 0.9243 - val_loss: 0.0493 - val_acc: 0.9843
Epoch 2/10
60000/60000 [==============================] - 100s 2ms/step - loss: 0.
0906 - acc: 0.9731 - val_loss: 0.0370 - val_acc: 0.9894
Epoch 3/10
60000/60000 [==============================] - 101s 2ms/step - loss: 0.
0662 - acc: 0.9799 - val_loss: 0.0352 - val_acc: 0.9893
Epoch 4/10
60000/60000 [==============================] - 100s 2ms/step - loss: 0.
0584 - acc: 0.9825 - val_loss: 0.0329 - val_acc: 0.9895
Epoch 5/10
60000/60000 [==============================] - 99s 2ms/step - loss: 0.0
483 - acc: 0.9854 - val_loss: 0.0305 - val_acc: 0.9908
Epoch 6/10
60000/60000 [==============================] - 100s 2ms/step - loss: 0.
0427 - acc: 0.9872 - val_loss: 0.0342 - val_acc: 0.9907
Epoch 7/10
60000/60000 [==============================] - 99s 2ms/step - loss: 0.0
396 - acc: 0.9879 - val_loss: 0.0225 - val_acc: 0.9923
Epoch 8/10
60000/60000 [==============================] - 100s 2ms/step - loss: 0.
0360 - acc: 0.9888 - val_loss: 0.0259 - val_acc: 0.9920
Epoch 9/10
60000/60000 [==============================] - 101s 2ms/step - loss: 0.
0350 - acc: 0.9897 - val_loss: 0.0259 - val_acc: 0.9924
Epoch 10/10
```

```
60000/60000 [==============================] - 100s 2ms/step - loss: 0.
0303 - acc: 0.9906 - val_loss: 0.0235 - val_acc: 0.9930
```

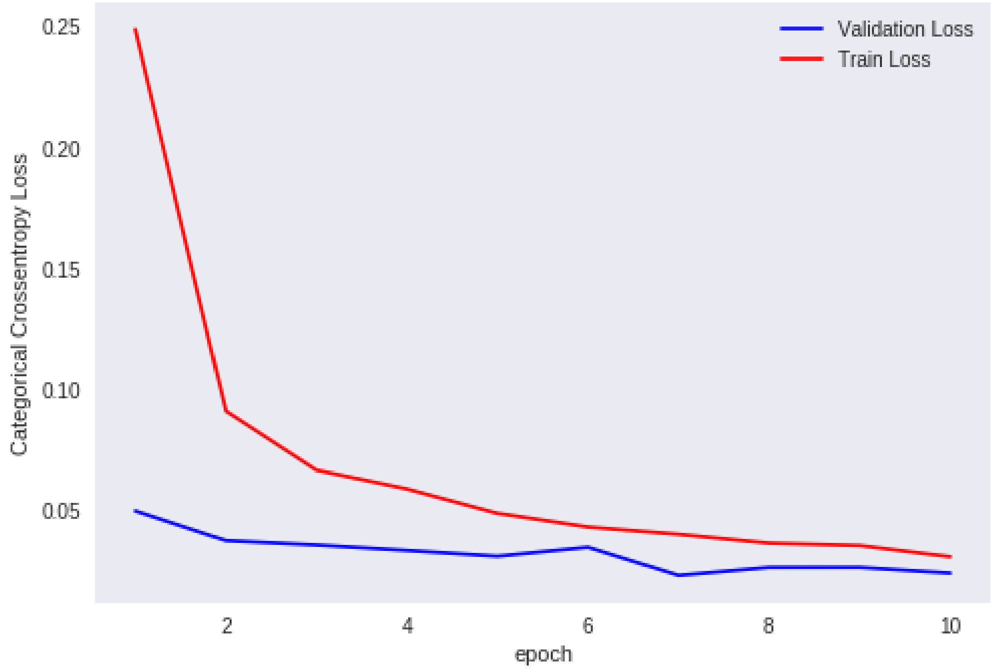
In [0]:

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.023485564049196637
Test accuracy: 0.993
```

In [0]:

```
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)
```



7X7 +Max pooling+BATCH NORMALISATION+ DROPOUT

In [0]:

```
model = Sequential()

model.add(Conv2D(128, (7,7), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Conv2D(64, (7,7), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history=model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 297s 5ms/step - loss: 0.
2403 - acc: 0.9261 - val_loss: 0.0549 - val_acc: 0.9837
Epoch 2/10
60000/60000 [==============================] - 293s 5ms/step - loss: 0.
0870 - acc: 0.9735 - val_loss: 0.0380 - val_acc: 0.9891
Epoch 3/10
60000/60000 [==============================] - 293s 5ms/step - loss: 0.
0643 - acc: 0.9810 - val_loss: 0.0363 - val_acc: 0.9882
Epoch 4/10
60000/60000 [==============================] - 289s 5ms/step - loss: 0.
0518 - acc: 0.9849 - val_loss: 0.0289 - val_acc: 0.9906
Epoch 5/10
60000/60000 [==============================] - 293s 5ms/step - loss: 0.
0441 - acc: 0.9871 - val_loss: 0.0266 - val_acc: 0.9909
Epoch 6/10
60000/60000 [==============================] - 294s 5ms/step - loss: 0.
0380 - acc: 0.9890 - val_loss: 0.0270 - val_acc: 0.9913
Epoch 7/10
60000/60000 [==============================] - 294s 5ms/step - loss: 0.
0352 - acc: 0.9896 - val_loss: 0.0283 - val_acc: 0.9925
Epoch 8/10
60000/60000 [==============================] - 289s 5ms/step - loss: 0.
0313 - acc: 0.9905 - val_loss: 0.0266 - val_acc: 0.9929
Epoch 9/10
60000/60000 [==============================] - 293s 5ms/step - loss: 0.
0279 - acc: 0.9920 - val_loss: 0.0342 - val_acc: 0.9905
Epoch 10/10
```

```
60000/60000 [==============================] - 293s 5ms/step - loss: 0.
0232 - acc: 0.9926 - val_loss: 0.0267 - val_acc: 0.9926
```

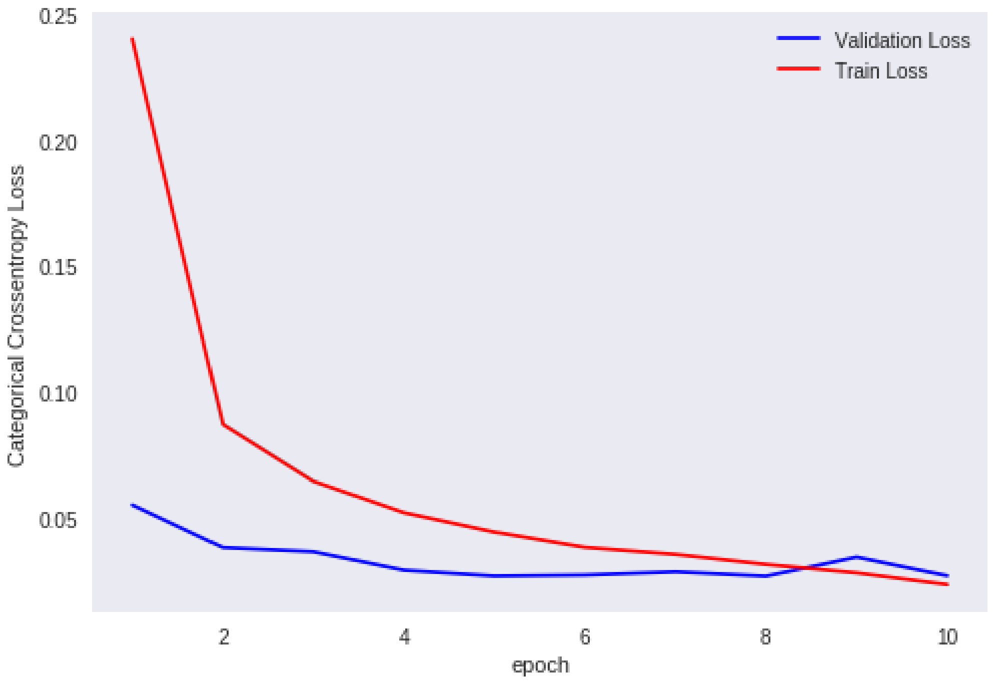
In [0]:

```python
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.026723121056987112
Test accuracy: 0.9926
```

In [0]:

```python
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)
```



In [0]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Kernal","Test score","Test accuracy","no_of_epoch"]

x.add_row(["3x3","2.4","99.2","10"])
x.add_row(["5x5","2.3","99.3","10"])
x.add_row(["7x7","2.6","99.2","10"])


print(x)
```

```
+--------+------------+---------------+-------------+
| Kernal | Test score | Test accuracy | no_of_epoch |
+--------+------------+---------------+-------------+
|  3x3   |    2.4     |      99.2     |      10     |
|  5x5   |    2.3     |      99.3     |      10     |
|  7x7   |    2.6     |     99.2      |     10      |
+--------+------------+---------------+-------------+
```