## Try various CNN networks on MNIST dataset

In [8]:

```
# 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

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)
```

```
Using TensorFlow backend.
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info.

```
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
```

In [0]:

```
%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_plot(x, vy, ty, ax, colors=['b']):
    fig , ax = plt.subplots(1,1)
```

```
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

**2,2 kernel with 3 Conv2D layers**

In [9]:

```
model1 = Sequential()
model1.add(Conv2D(32, kernel_size=(2, 2),activation='relu',input_shape=input_shape))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(Conv2D(64, (2, 2), activation='relu'))
model1.add(Dropout(0.25))
model1.add(Conv2D(64, (2, 2), activation='relu'))
model1.add(Dropout(0.30))
model1.add(Flatten())
model1.add(Dense(128, activation='relu'))
model1.add(Dropout(0.5))
model1.add(Dense(num_classes, activation='softmax'))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:66: The name tf.get_default_graph is deprecated. Plea
se use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:541: The name tf.placeholder is deprecated. Please us
e tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:4432: The name tf.random_uniform is deprecated. Pleas
e use tf.random.uniform instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:4267: The name tf.nn.max_pool is deprecated. Please u
se tf.nn.max_pool2d instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:148: The name tf.placeholder_with_default is
deprecated. Please use tf.compat.v1.placeholder_with_default instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:3733: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future
version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
```

In [17]:

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

history1 = model1.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score1 = model1.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score1[0])
print('Test accuracy:', score1[1])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 81s 1ms/step - loss: 0.0362 - acc: 0.9892 - val_los
s: 0.0301 - val_acc: 0.9901
Epoch 2/12
60000/60000 [==============================] - 82s 1ms/step - loss: 0.0327 - acc: 0.9902 - val_los
s: 0.0251 - val_acc: 0.9907
Epoch 3/12
60000/60000 [==============================] - 81s 1ms/step - loss: 0.0316 - acc: 0.9904 - val los
```

```
s: 0.0232 - val_acc: 0.9917
Epoch 4/12
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0286 - acc: 0.9914 - val_los
s: 0.0246 - val_acc: 0.9919
Epoch 5/12
60000/60000 [==============================] - 81s 1ms/step - loss: 0.0280 - acc: 0.9913 - val_los
s: 0.0253 - val_acc: 0.9916
Epoch 6/12
60000/60000 [==============================] - 79s 1ms/step - loss: 0.0293 - acc: 0.9912 - val_los
s: 0.0246 - val_acc: 0.9909
Epoch 7/12
60000/60000 [==============================] - 79s 1ms/step - loss: 0.0281 - acc: 0.9916 - val_los
s: 0.0283 - val_acc: 0.9909
Epoch 8/12
60000/60000 [==============================] - 79s 1ms/step - loss: 0.0279 - acc: 0.9921 - val_los
s: 0.0246 - val_acc: 0.9920
Epoch 9/12
60000/60000 [==============================] - 78s 1ms/step - loss: 0.0266 - acc: 0.9916 - val_los
s: 0.0271 - val_acc: 0.9912
Epoch 10/12
60000/60000 [==============================] - 77s 1ms/step - loss: 0.0257 - acc: 0.9917 - val_los
s: 0.0268 - val_acc: 0.9914
Epoch 11/12
60000/60000 [==============================] - 78s 1ms/step - loss: 0.0232 - acc: 0.9928 - val_los
s: 0.0320 - val_acc: 0.9910
Epoch 12/12
60000/60000 [==============================] - 76s 1ms/step - loss: 0.0248 - acc: 0.9924 - val_los
s: 0.0253 - val_acc: 0.9915
Test loss: 0.025287141826237668
Test accuracy: 0.9915
```

**Train vs Validation Loss plot**

In [19]:

```
import mpld3
from mpld3 import plugins

score_task1 = model1.evaluate(x_test, y_test, verbose=0)

print('Test score:', score_task1[0])
print('Test accuracy:', score_task1[1])

ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x_task1 = list(range(1,epochs+1))

vy_task1 = history1.history['val_loss']
ty_task1 = history1.history['loss']

print(vy_task1)
print(ty_task1)

plt_dynamic_plot(x_task1, vy_task1, ty_task1, ax)
mpld3.display()
```

```
Test score: 0.025287141826237668
Test accuracy: 0.9915
[0.030144057344489557, 0.025085610742116113, 0.023178132422553607, 0.02459734082815121,
0.025346980234188958, 0.024620658679318147, 0.028283333141754338, 0.024588756948978698,
0.027148247838640237, 0.026806357018148992, 0.03204613807176647, 0.025287141948716816]
[0.0361943228016297, 0.03266321079296371, 0.031573505687961974, 0.028629884781564276,
0.028032565595582128, 0.029291857266798615, 0.02810095411228637, 0.027880893707772095,
0.02660320466607809, 0.025652228073899944, 0.023163021868964035, 0.024849909565473595]
```

Out[19]:

**3,3 kernel with 5 Conv2D layers**

In [0]:

```
from keras.layers.normalization import BatchNormalization

model2 = Sequential()
model2.add(Conv2D(32, (3, 3), activation='relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Conv2D(32, (3, 3), activation='relu'))
model2.add(Dropout(0.25))
model2.add(BatchNormalization())
model2.add(Conv2D(32, (3, 3), activation='relu'))
model2.add(MaxPooling2D(pool_size=(1, 1)))
model2.add(Conv2D(32, (3, 3), activation='relu'))
model2.add(Dropout(0.30))
model2.add(BatchNormalization())
model2.add(Conv2D(64, (3, 3), activation='relu'))
model2.add(Flatten())
model2.add(Dense(128, activation='relu'))
model2.add(Dropout(0.5))
model2.add(Dense(num_classes, activation='softmax'))
```

In [0]:

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

model2.fit(x_train, y_train,
           batch_size=batch_size,
           epochs=epochs,
           verbose=1,
           validation_data=(x_test, y_test))
score2 = model2.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score2[0])
print('Test accuracy:', score2[1])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 72s 1ms/step - loss: 0.2677 - acc: 0.9172 - val_los
s: 0.0647 - val_acc: 0.9810
Epoch 2/12
60000/60000 [==============================] - 72s 1ms/step - loss: 0.0831 - acc: 0.9759 - val_los
s: 0.0490 - val_acc: 0.9842
Epoch 3/12
60000/60000 [==============================] - 71s 1ms/step - loss: 0.0619 - acc: 0.9808 - val_los
s: 0.0691 - val_acc: 0.9780
Epoch 4/12
60000/60000 [==============================] - 71s 1ms/step - loss: 0.0503 - acc: 0.9850 - val_los
s: 0.0368 - val_acc: 0.9905
Epoch 5/12
60000/60000 [==============================] - 71s 1ms/step - loss: 0.0428 - acc: 0.9876 - val_los
s: 0.0398 - val_acc: 0.9886
Epoch 6/12
60000/60000 [==============================] - 72s 1ms/step - loss: 0.0410 - acc: 0.9877 - val_los
s: 0.0307 - val_acc: 0.9916
Epoch 7/12
60000/60000 [==============================] - 72s 1ms/step - loss: 0.0350 - acc: 0.9899 - val_los
s: 0.0276 - val_acc: 0.9927
Epoch 8/12
60000/60000 [==============================] - 72s 1ms/step - loss: 0.0335 - acc: 0.9905 - val_los
s: 0.0268 - val_acc: 0.9926
Epoch 9/12
60000/60000 [==============================] - 72s 1ms/step - loss: 0.0307 - acc: 0.9913 - val_los
s: 0.0322 - val_acc: 0.9901
Epoch 10/12
60000/60000 [==============================] - 72s 1ms/step - loss: 0.0278 - acc: 0.9916 - val_los
s: 0.0237 - val_acc: 0.9927
Epoch 11/12
60000/60000 [==============================] - 72s 1ms/step - loss: 0.0272 - acc: 0.9921 - val_los
s: 0.0228 - val_acc: 0.9938
Epoch 12/12
60000/60000 [==============================] - 72s 1ms/step - loss: 0.0263 - acc: 0.9925 - val_los
s: 0.0271 - val_acc: 0.9927
Test loss: 0.027133449450492026
Test accuracy: 0.9927
```

**Train vs Validation Loss plot**

In [21]:

```
x_task2 = list(range(1,epochs+1))

print(vy_task2)
print(ty_task2)

plt_dynamic_plot(x_task2, vy_task2, ty_task2, ax)
mpld3.display()
```

```
[0.0647, 0.049, 0.0691, 0.0368, 0.0398, 0.0307, 0.0276, 0.0268, 0.0322, 0.0237, 0.0228, 0.0271]
[0.2677, 0.0831, 0.0619, 0.0503, 0.0428, 0.041, 0.035, 0.0335, 0.0307, 0.0278, 0.0272, 0.0263]
```

Out[21]:

**5,5 kernel with 7 Conv2D layers**

In [0]:

```
from keras.layers.normalization import BatchNormalization

model3 = Sequential()
model3.add(Conv2D(32, (5, 5), padding = 'same', activation='relu'))
model3.add(Dropout(0.25))
model3.add(Conv2D(32, (5, 5), padding = 'same', activation='relu'))
model3.add(Dropout(0.25))
model3.add(BatchNormalization())
model3.add(Conv2D(32, (5, 5), padding = 'same', activation='relu'))
model3.add(Dropout(0.30))
model3.add(Conv2D(32, (5, 5), padding = 'same', activation='relu'))
model3.add(Dropout(0.30))
model3.add(BatchNormalization())
model3.add(Conv2D(64, (5, 5), padding = 'same', activation='relu'))
model3.add(Dropout(0.35))
model3.add(BatchNormalization())
model3.add(Conv2D(64, (5, 5), padding = 'same',activation='relu'))
model3.add(Dropout(0.40))
model3.add(Conv2D(64, (5, 5), activation='relu'))
model3.add(Flatten())
model3.add(Dense(128, activation='relu'))
model3.add(Dropout(0.5))
model3.add(Dense(num_classes, activation='softmax'))
```

In [0]:

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

model3.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score3 = model3.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score3[0])
print('Test accuracy:', score3[1])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 2776s 46ms/step - loss: 0.4159 - acc: 0.8754 - val_
loss: 0.0560 - val_acc: 0.9836
Epoch 2/12
60000/60000 [==============================] - 2784s 46ms/step - loss: 0.0938 - acc: 0.9743 - val_
loss: 0.0738 - val_acc: 0.9790
Epoch 3/12
60000/60000 [==============================] - 2785s 46ms/step - loss: 0.0638 - acc: 0.9826 - val_
loss: 0.0284 - val_acc: 0.9919
Epoch 4/12
```

```
Epoch 4/12
60000/60000 [==============================] - 2765s 46ms/step - loss: 0.0549 - acc: 0.9848 - val_
loss: 0.0309 - val_acc: 0.9905
Epoch 5/12
60000/60000 [==============================] - 2779s 46ms/step - loss: 0.0459 - acc: 0.9871 - val_
loss: 0.0263 - val_acc: 0.9923
Epoch 6/12
60000/60000 [==============================] - 2783s 46ms/step - loss: 0.0388 - acc: 0.9895 - val_
loss: 0.0340 - val_acc: 0.9912
Epoch 7/12
60000/60000 [==============================] - 2772s 46ms/step - loss: 0.0365 - acc: 0.9895 - val_
loss: 0.0326 - val_acc: 0.9908
Epoch 8/12
60000/60000 [==============================] - 2790s 46ms/step - loss: 0.0327 - acc: 0.9911 - val_
loss: 0.0317 - val_acc: 0.9921
Epoch 9/12
60000/60000 [==============================] - 2808s 47ms/step - loss: 0.0291 - acc: 0.9919 - val_
loss: 0.0277 - val_acc: 0.9922
Epoch 10/12
60000/60000 [==============================] - 2786s 46ms/step - loss: 0.0284 - acc: 0.9925 - val_
loss: 0.0216 - val_acc: 0.9929
Epoch 11/12
60000/60000 [==============================] - 2821s 47ms/step - loss: 0.0249 - acc: 0.9931 - val_
loss: 0.0244 - val_acc: 0.9938
Epoch 12/12
60000/60000 [==============================] - 2830s 47ms/step - loss: 0.0257 - acc: 0.9927 - val_
loss: 0.0271 - val_acc: 0.9930
Test loss: 0.027074409721957453
Test accuracy: 0.993
```

#### Train vs Validation Loss plot

In [22]:

```
x_task3 = list(range(1,epochs+1))

print(vy_task3)
print(ty_task3)

plt_dynamic_plot(x_task3, vy_task3, ty_task3, ax)
mpld3.display()
```

```
[0.056, 0.0738, 0.0284, 0.0309, 0.0263, 0.034, 0.0326, 0.0317, 0.0277, 0.0216, 0.0244, 0.0271]
[0.4159, 0.0938, 0.0638, 0.0549, 0.0459, 0.0388, 0.0365, 0.0327, 0.0291, 0.0284, 0.0249, 0.0257]
```

Out[22]:

#### Conclusion

In [0]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Architecture", "Activation", "Kernel" , "Test Loss", "Test Accuracy"]
x.add_row(["3 Layer NN", "ReLU","2 * 2", 0.021 , 0.9914 ])
x.add_row(["5 Layer NN", "ReLU","3 * 3", 0.0271, 0.9927])
x.add_row(["7 Layer NN", "ReLU","5 * 5", 0.0270 , 0.9930 ])
print(x)
```

```
+--------------+------------+--------+-----------+---------------+
| Architecture | Activation | Kernel | Test Loss | Test Accuracy |
+--------------+------------+--------+-----------+---------------+
|  3 Layer NN  |    ReLU    | 2 * 2  |   0.021   |     0.9914    |
|  5 Layer NN  |    ReLU    | 3 * 3  |   0.0271  |     0.9927    |
|  7 Layer NN  |    ReLU    | 5 * 5  |   0.027   |     0.993     |
+--------------+------------+--------+-----------+---------------+
```