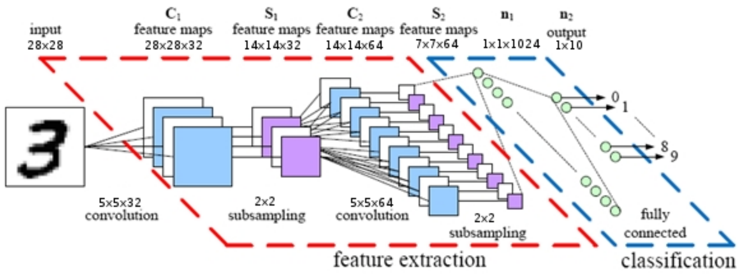
Basic CNNs with Keras

Now that we are on a roll, lets take a look into the more exciting realm of CNNs. (These definately are the in-thing ;)

Wondering what are CNNs?? (Wiki to the rescue)https://en.wikipedia.org/wiki/Convolutional_neural_network (https://en.wikipedia.org/wiki/Convolutional_neural_network)

Here we would create convolutional neural network (CNN) with 2 convolutional layers, one fully connected layer. Max pooling layers are followed after each convolutional layer and the activation function used are ReLU. For simplicity we would use the cross entropy loss function and Adam Optmization function for parameter update using the backpropagation algorithm. Note:: that this would be simple walk of a CNN architecture which is similar to the popular LeNet-5 architecture. More details and insights will be given in Lab2.

The below figure shows the CNN used in this example along with filter sizes used for each layer.



In this experiment we shall use CNN to classify MNIST dataset. Step 1) Load the MNIST data Step 2) Let the provided solver and net do their jobs. Step 3) Well we are all grown ups and would like things our own way. Feel free to modify the net and solver to get most juice.

You can try writing a net for yourself, but for the lazy ones:

Reference:https://github.com/fchollet/keras/blob/master/examples/mnist_cnn.py (https://github.com/fchollet/keras/blob/master/examples/mnist_cnn.py)

Doing the requisite imports.

In [1]:

```
from __future__ import print_function
# To run on gpu
#import os
#os.environ['THEANO_FLAGS'] = "device=gpu0"
import numpy as np
np.random.seed(1337)  # for reproducibility

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
```

```
Using Theano backend.
```

Setting the parameters for the net

In [5]:

```
batch_size = 128  # amount of samples
nb_classes = 10   # possible Output class
nb_epoch = 2     # 12 Parts of Training set

# input image dimensions
img_rows, img_cols = 28, 28  # image Size
# number of convolutional filters to use
nb_filters = 32
# size of pooling area for max pooling
nb_pool = 2
# convolution kernel size
nb_conv = 3
```

Loading the data and preprocessing it

In [8]:

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

X_train = X_train.reshape(X_train.shape[0], 1, img_rows, img_cols) # Dividing Da
ta
X_test = X_test.reshape(X_test.shape[0], 1, img_rows, img_cols)
X_train = X_train.astype('float32')    # Converting to float
X_test = X_test.astype('float32')
X_train /= 255      # Normalizing to 0-1
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)

''' above code will do following
ip      0123456789
1       0100000000
2       0010000000
'''
```

```
X_train shape: (60000, 1, 28, 28)
60000 train samples
10000 test samples
```

Out[8]:

```
' above code will do following\nip      0123456789\n1       0100000
000\n2       0010000000\n'
```

Writing the net using the convolutional layers,pooling and activation layers.

In [9]:

```
model = Sequential()

model.add(Convolution2D(nb_filters, nb_conv, nb_conv,
                        border_mode='valid',
                        input_shape=(1, img_rows, img_cols)))
model.add(Activation('relu'))
model.add(Convolution2D(nb_filters, nb_conv, nb_conv))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(nb_pool, nb_pool)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

Compile and then train the network

In [ ]:

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

model.fit(X_train, Y_train, batch_size=batch_size, nb_epoch=nb_epoch,
          verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/2
 5376/60000 [=>............................] - ETA: 1405s - loss:
 1.3271 - acc: 0.5733
```

Print the scores

In [ ]:

```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

Exercise

Now to get our hands ditry...

Q1: Check out what happens when we do the following:

- vary the learning rate (0.0001 - 100)
- change filter sizes of convolutional , max pooling and fully connected layers.
- try out different activation functions (sigmoid,tanh)

In [ ]:

```
# Your *hardwork* here
```

Q2: Train a similar CNN architecture for CIFAR-10 dataset and report the classification accuracy on test dataset.

In [ ]:

```
# Your *hardwork* here
```