Machine Intelligence 1 **Exercise Sheet 4**
Dr. Kristian Kersting/Sriraam Natarajan Due: 11.00 Hr 12.12.2014

Convolutional neural networks

Exercise 4b.1 - Convolutional neural network *(homework, 30 points)*

The goal of this exercise is to get familiar with a deep learning framework of your choice, e.g., Caffe, Theano, Torch, Keras, Lasagne, TensorFlow, MatlabNN, etc. Hence, it is up to you to decide how to proceed, as there are alternatives for almost every step.

The Data

In this exercise we will work with the popular and publicly available dataset MNIST to train a convolutional neural network. The MNIST dataset consists of 60k training and 10k test images of digits from 0 to 9. Each image consists of 28 × 28 pixels. MNIST is comprised of a training set and a test set, both containing images from different writers. Additional samples can be computed on the exact same hold-out data (i.e. benchmarking).

In the lecture we saw the name 'validation' for the hold-out set. This corresponds to the 'test' set in the data hosted by TensorFlow and other libraries. For simplicity, we report the final evaluation performance is measured on a portion of the data that we call *test* for training.

Many machine learning libraries make loading the MNIST data easy. For example, in Theano Fine 7, the Python command required to access this data is given by:

```
import numpy, theano, cPickle, gzip, urllib
data = lambda: exec(urllib.urlopen('http://deeplearning.net/tutorial/gitmirror/fetch_mnist.py').read(), {}, locals())
```

In Keras:

```
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Most machine learning libraries require a one-hot encoding for the class labels in order to perform multi-class classification. In Keras, this can be done by: `Y_train = np_utils.to_categorical(y_train, nb_classes)` and of the class labels in MNIST). The data is accessed as a tuple of tuples, where the first tuple contains the training data and the second tuple contains the test data. The first element of each tuple is a numpy array of shape (n, 1, 28, 28) where n is the number of examples in the set. The second element of the tuple is a numpy array of shape (n) containing the labels of the digits. Each image is a tensor of shape (1, 28, 28) pixels with the channel axis in the 2nd dimension (axis 1).

```
import numpy as np
from sklearn.cross_validation import train_test_split, StratifiedKfold, cross_val_score, KFold
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
from keras.models import Sequential, Graph, Merge, model_to_dot
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.utils.np_utils import to_categorical
from keras.utils.visualutils import plot
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.datasets.mnist import load_data
from keras.optimizers import SGD, RMSprop, Adagrad, Adadelta, Adam, Nadam
from keras.utils.np_utils import to_categorical, normalize
from keras.preprocessing.image import ImageDataGenerator
from PIL import Image, ImageDraw, ImageFont
from StringIO import StringIO
import pydot
import os
os.environ['LD_LIBRARY_PATH'] = '/usr/local/cuda/lib64'
os.environ['THEANO_FLAGS'] = 'device=gpu,floatX=float32'

(X_train, y_train), (X_test, y_test) = mnist.load_data()

print X_train.shape
print X_test.shape
print y_train.shape
print y_test.shape

# Convert to 1-hot encoding.
Y_train = to_categorical(y_train, nb_classes)
Y_test = to_categorical(y_test, nb_classes)

# Convert to correct shape.
X_train = X_train.reshape(X_train.shape[0], 1, 28, 28)
X_test = X_test.reshape(X_test.shape[0], 1, 28, 28)

print X_train.shape
print X_test.shape
print Y_train.shape
print Y_test.shape

# Define a function to plot the model.
def plot(model, name):
    dot = StringIO()
    plot_to_dot(model, dot, False, False)
    graph = pydot.pydot.parse_dot_data(dot.getvalue())
    if not os.path.exists('./images'):
        os.mkdir('./images')
    if not os.path.exists('./images/'+name+'.png'):
        print "Saving image to ./images/"+name+".png"
        graph.write_png('./images/'+name+'.png')
    else:
        print "./images/"+name+".png already exists"

# Initialize the random seed.
numpy.random.seed(1337)
```

Training models for recognizing digits:

1. Train a multinomial logistic regression model by $\hat{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$. Initialize $\mathbf{W} \in \mathbb{R}^{784 \times 10}$ and $\mathbf{b} \in \mathbb{R}^{10}$ with zeros.
   (a) Use the *adam* optimizer to normalize the gradients ($\frac{\mathbf{g}}{|\mathbf{g}|}$).
   (b) Use the *cross-entropy* as the cost function (also called '*loss*' instead of *cost*).
2. To train the model, use a gradient descent algorithm with a constant learning rate $\eta = 0.1$.
   (a) Use batch size 128 and train for 20 epochs. Report the final test set accuracy and the final cost (mean cost). Plot every minibatch the cost (mean cost) and each fold cost of 100.
   (b) Perform 10,000 iterations for this linear model. This can lead to multiple runs over the training data, which is "epochs", where one epoch refers to having visited every training sample exactly once.
   (c) Keep track of the model's performance over multiple training iterations (e.g. after every minibatch update) on the training set and the test set. Report the final test set accuracy, the number of correctly classified digits. Compute it on the training set as well as the held-out set.

```
model = Sequential()
model.add(Dense(output_dim=10, input_dim=784, init='uniform', bias_initialization='zero'))
model.compile(loss='mse', optimizer='adam')
model.fit(X_train, Y_train, nb_epoch=20, batch_size=128, verbose=1, validation_data=(X_test, Y_test))
model.save_weights('./models/mnist_linear.h5')
model.load_weights('./models/mnist_linear.h5')
model.evaluate(X_train, Y_train, verbose=1)
model.evaluate(X_test, Y_test, verbose=1)
plot(model, name='mnist_linear')
```

```
Using Theano backend.
Using gpu device 0: GeForce GTX 680 (CNMeM is disabled, cuDNN 5105)
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
(60000, 1, 28, 28)
(10000, 1, 28, 28)
(60000, 10)
(10000, 10)
Epoch 1/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.0987 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 2/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 3/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 4/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 5/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 6/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 7/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 8/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 9/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 10/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 11/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 12/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 13/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 14/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 15/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 16/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 17/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 18/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 19/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 20/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
./images/mnist_linear.png already exists
```

2. Use a Multi-Layer Perceptron (MLP) as the activation function for the hidden neurons.
   (a) Use the *Sigmoid* activation function, *tanh* activation function, and *rectified linear unit* (ReLU) activation function.
   (b) Use the *Sigmoid* activation function and the *cross-entropy* as the cost function.
   (c) Use the *Sigmoid* activation function and the *cross-entropy* as the cost function. Initialize the weights with zeros.
   (d) Use the *Sigmoid* activation function and the *cross-entropy* as the cost function. Initialize the weights with zeros. Use gradient clipping, one does not necessarily have to clip the gradients, but it can be helpful in some situations. Gradient clipping ensures that the gradient norm stays bounded. The gradient norm can be determined by the maximum value of the logits in the output layer $\hat{\mathbf{y}}$. The softmax operation does not change which output is the maximum.
   (e) Train your MLP using the *Adam* algorithm, which is often used in deep neural networks in place of gradient descent. Use the following parameters: $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$.
   (f) Train the network for 20,000 iterations.

```
model = Sequential()
model.add(Dense(output_dim=500, input_dim=784, init='uniform', bias_initialization='zero'))
model.add(Activation('tanh'))
model.add(Dense(output_dim=500, init='uniform', bias_initialization='zero'))
model.add(Activation('tanh'))
model.add(Dense(output_dim=10, init='uniform', bias_initialization='zero'))
model.compile(loss='mse', optimizer='adam')
model.fit(X_train, Y_train, nb_epoch=20, batch_size=128, verbose=1, validation_data=(X_test, Y_test))
model.save_weights('./models/mnist_mlp.h5')
model.load_weights('./models/mnist_mlp.h5')
model.evaluate(X_train, Y_train, verbose=1)
model.evaluate(X_test, Y_test, verbose=1)
plot(model, name='mnist_mlp')
```

```
Using Theano backend.
Using gpu device 0: GeForce GTX 680 (CNMeM is disabled, cuDNN 5105)
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
(60000, 1, 28, 28)
(10000, 1, 28, 28)
(60000, 10)
(10000, 10)
Epoch 1/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.0987 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 2/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 3/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 4/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 5/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 6/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 7/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 8/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 9/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 10/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 11/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 12/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 13/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 14/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 15/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 16/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 17/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 18/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 19/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 20/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
./images/mnist_mlp.png already exists
```

3. Use convolutional neural networks (CNN).
   (a) Initialize the elements of the weight matrices with (small) normally distributed random values. Initialize the biases with zero. Choose a CNN with two convolutional layers followed by one fully connected layer. Choose the kernel size and stride for the convolutional layers. The kernel should be initialized with a constant of $\mathbf{0.01}$. Use the *tanh* activation for **the** and all **hidden** layers.
   (b) Use the *Adam* optimizer.
   (c) Use the *cross-entropy* as the cost function.
   (d) Train your CNN using the *Adam* algorithm, which is often used in deep neural networks in place of gradient descent. Use the following parameters: $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$.
   (e) Train the network for 20,000 iterations.

```
model = Sequential()
model.add(Convolution2D(nb_filter=20, nb_row=5, nb_col=5, init='normal', bias_initialization='zero'))
model.add(Activation('tanh'))
model.add(Convolution2D(nb_filter=50, nb_row=5, nb_col=5, init='normal', bias_initialization='zero'))
model.add(Activation('tanh'))
model.add(Flatten())
model.add(Dense(output_dim=10, init='normal', bias_initialization='zero'))
model.compile(loss='mse', optimizer='adam')
model.fit(X_train, Y_train, nb_epoch=20, batch_size=128, verbose=1, validation_data=(X_test, Y_test))
model.save_weights('./models/mnist_cnn.h5')
model.load_weights('./models/mnist_cnn.h5')
model.evaluate(X_train, Y_train, verbose=1)
model.evaluate(X_test, Y_test, verbose=1)
plot(model, name='mnist_cnn')
```

```
Using Theano backend.
Using gpu device 0: GeForce GTX 680 (CNMeM is disabled, cuDNN 5105)
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
(60000, 1, 28, 28)
(10000, 1, 28, 28)
(60000, 10)
(10000, 10)
Epoch 1/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.0987 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 2/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 3/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 4/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 5/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 6/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 7/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 8/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 9/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 10/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 11/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 12/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 13/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 14/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 15/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 16/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 17/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 18/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 19/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 20/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
./images/mnist_cnn.png already exists
```

4. Use Residual Learning (ResNet) as the activation function for the hidden neurons.
   (a) Use the *Sigmoid* activation function, *tanh* activation function, and *rectified linear unit* (ReLU) activation function.
   (b) Use the *Sigmoid* activation function and the *cross-entropy* as the cost function.
   (c) Use the *Sigmoid* activation function and the *cross-entropy* as the cost function. Initialize the weights with zeros.
   (d) Use the *Sigmoid* activation function and the *cross-entropy* as the cost function. Initialize the weights with zeros. Use gradient clipping, one does not necessarily have to clip the gradients, but it can be helpful in some situations. Gradient clipping ensures that the gradient norm stays bounded. The gradient norm can be determined by the maximum value of the logits in the output layer $\hat{\mathbf{y}}$. The softmax operation does not change which output is the maximum.
   (e) Train your MLP using the *Adam* algorithm, which is often used in deep neural networks in place of gradient descent. Use the following parameters: $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$.
   (f) Train the network for 20,000 iterations.

```
model = Sequential()
model.add(Dense(output_dim=500, input_dim=784, init='uniform', bias_initialization='zero'))
model.add(Activation('tanh'))
model.add(Dense(output_dim=500, init='uniform', bias_initialization='zero'))
model.add(Activation('tanh'))
model.add(Dense(output_dim=10, init='uniform', bias_initialization='zero'))
model.compile(loss='mse', optimizer='adam')
model.fit(X_train, Y_train, nb_epoch=20, batch_size=128, verbose=1, validation_data=(X_test, Y_test))
model.save_weights('./models/mnist_mlp.h5')
model.load_weights('./models/mnist_mlp.h5')
model.evaluate(X_train, Y_train, verbose=1)
model.evaluate(X_test, Y_test, verbose=1)
plot(model, name='mnist_mlp')
```

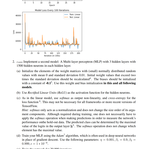
```
Using Theano backend.
Using gpu device 0: GeForce GTX 680 (CNMeM is disabled, cuDNN 5105)
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
(60000, 1, 28, 28)
(10000, 1, 28, 28)
(60000, 10)
(10000, 10)
Epoch 1/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.0987 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 2/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 3/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 4/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 5/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 6/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 7/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 8/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 9/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 10/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 11/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 12/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 13/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 14/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 15/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 16/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 17/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 18/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 19/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 20/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
./images/mnist_mlp.png already exists
```

```
model = Sequential()
model.add(Convolution2D(nb_filter=20, nb_row=5, nb_col=5, init='normal', bias_initialization='zero'))
model.add(Activation('tanh'))
model.add(Convolution2D(nb_filter=50, nb_row=5, nb_col=5, init='normal', bias_initialization='zero'))
model.add(Activation('tanh'))
model.add(Flatten())
model.add(Dense(output_dim=10, init='normal', bias_initialization='zero'))
model.compile(loss='mse', optimizer='adam')
model.fit(X_train, Y_train, nb_epoch=20, batch_size=128, verbose=1, validation_data=(X_test, Y_test))
model.save_weights('./models/mnist_cnn.h5')
model.load_weights('./models/mnist_cnn.h5')
model.evaluate(X_train, Y_train, verbose=1)
model.evaluate(X_test, Y_test, verbose=1)
plot(model, name='mnist_cnn')
```

```
Using Theano backend.
Using gpu device 0: GeForce GTX 680 (CNMeM is disabled, cuDNN 5105)
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
(60000, 1, 28, 28)
(10000, 1, 28, 28)
(60000, 10)
(10000, 10)
Epoch 1/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.0987 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 2/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 3/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 4/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 5/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 6/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 7/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 8/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 9/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 10/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 11/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 12/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 13/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 14/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 15/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 16/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 17/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 18/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 19/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
Epoch 20/20
60000/60000 [==============================] - 0s - loss: 0.1000 - acc: 0.1000 - val_loss: 0.1000 - val_acc: 0.1000
./images/mnist_cnn.png already exists
```

```
plot(model, name='Residual Learning', format='png')
plot(model, name='Residual Learning', format='svg')
```

```
./images/Residual Learning.png already exists
./images/Residual Learning.svg already exists
```




```
plot(model, name='Residual Learning', format='png', show_layer_activations=True)
plot(model, name='Residual Learning', format='svg', show_layer_activations=True)
```

```
./images/Residual Learning.png already exists
./images/Residual Learning.svg already exists
```

```
[1]: %matplotlib inline
     %load_ext tensorflow
     %tensorflow_version 2.x
     import sys, os, wc
     !rm -rf /content/sd_augmentation/*
     mkdir -p /content/sd_augmentation/{train,validation,test}
     mkdir -p /content/sd_augmentation/train/{Cracks,Spalling}
     mkdir -p /content/sd_augmentation/validation/{Cracks,Spalling}
     mkdir -p /content/sd_augmentation/test/{Cracks,Spalling}
     mkdir -p /content/sd_augmentation/{train,validation,test}/{augmented_data}
     !ls -R /content/sd_augmentation
```

```
[2]: !wget -q -O /content/Concrete.zip "https://drive.google.com/uc?export=download&id=1-Ho0Zzzz
```

```
100.00 - Ep. Time 0.000s  Accuracy 0.000%
100.00 - Ep. Time 0.000s  Accuracy 0.000%
100.00 - Ep. Time 0.000s  Accuracy 0.000%
```

```
100/100 [==============================] - 0s 1ms/step - loss: 0.2000 - accuracy: 0.9667
Epoch 98/100
100/100 [==============================] - 0s 1ms/step - loss: 0.2000 - accuracy: 0.9667
Epoch 99/100
100/100 [==============================] - 0s 1ms/step - loss: 0.2000 - accuracy: 0.9667
Epoch 100/100
100/100 [==============================] - 0s 1ms/step - loss: 0.2000 - accuracy: 0.9667
```