3-Convolution Layers

```
# 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,BatchNormalization
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()

print(x_train.shape)

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=(2, 2),activation='relu',input_shape=input_shape,kernel_
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Conv2D(32, (5, 5), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.Adam()

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

```
Using TensorFlow backend.
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 2s 0us/step
(60000, 28, 28)
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 27, 27, 128)       640       
_________________________________________________________________
batch_normalization_1 (Batch (None, 27, 27, 128)       512       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 25, 25, 64)        73792     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 32)          51232     
_________________________________________________________________
batch_normalization_2 (Batch (None, 8, 8, 32)          128       
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 32)          0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 4, 4, 32)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 128)               65664     
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290      
=================================================================
Total params: 193,258
Trainable params: 192,938
Non-trainable params: 320
_________________________________________________________________
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:79

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 22s 374us/step - loss: 0.2054 - acc: 0
Epoch 2/12
60000/60000 [==============================] - 19s 310us/step - loss: 0.0615 - acc: 0
Epoch 3/12
60000/60000 [==============================] - 19s 310us/step - loss: 0.0451 - acc: 0
Epoch 4/12
60000/60000 [==============================] - 19s 311us/step - loss: 0.0395 - acc: 0
Epoch 5/12
60000/60000 [==============================] - 19s 311us/step - loss: 0.0314 - acc: 0
Epoch 6/12
60000/60000 [==============================] - 19s 312us/step - loss: 0.0287 - acc: 0
Epoch 7/12
60000/60000 [==============================] - 19s 311us/step - loss: 0.0268 - acc: 0
Epoch 8/12
60000/60000 [==============================] - 19s 310us/step - loss: 0.0243 - acc: 0
Epoch 9/12
60000/60000 [==============================] - 19s 310us/step - loss: 0.0218 - acc: 0
Epoch 10/12
60000/60000 [==============================] - 19s 310us/step - loss: 0.0207 - acc: 0
Epoch 11/12
60000/60000 [==============================] - 19s 311us/step - loss: 0.0199 - acc: 0
Epoch 12/12
60000/60000 [==============================] - 19s 311us/step - loss: 0.0176 - acc: 0
Test loss: 0.028177190259779036
Test accuracy: 0.9919
```

```
# 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,BatchNormalization
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()

print(x_train.shape)

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:
```

```
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=(2, 2),activation='relu',input_shape=input_shape,kernel_
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Conv2D(64, (3, 3), activation='sigmoid'))
model.add(Conv2D(32, (5, 5), activation='sigmoid'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Flatten())
model.add(Dense(128, activation='sigmoid'))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.Adagra

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

```
(60000, 28, 28)
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 27, 27, 128)       640
_________________________________________________________________
batch_normalization_3 (Batch (None, 27, 27, 128)       512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 13, 13, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 11, 11, 64)        73792
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 7, 7, 32)          51232
_________________________________________________________________
batch_normalization_4 (Batch (None, 7, 7, 32)          128
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 3, 3, 32)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 288)               0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               36992
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1290
=================================================================
Total params: 164,586
Trainable params: 164,266
Non-trainable params: 320
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 14s 233us/step - loss: 0.1210 - acc: 0
Epoch 2/12
60000/60000 [==============================] - 13s 220us/step - loss: 0.0365 - acc: 0
Epoch 3/12
60000/60000 [==============================] - 13s 220us/step - loss: 0.0247 - acc: 0
Epoch 4/12
60000/60000 [==============================] - 13s 220us/step - loss: 0.0185 - acc: 0
Epoch 5/12
60000/60000 [==============================] - 13s 220us/step - loss: 0.0141 - acc: 0
Epoch 6/12
60000/60000 [==============================] - 13s 220us/step - loss: 0.0107 - acc: 0
Epoch 7/12
60000/60000 [==============================] - 13s 220us/step - loss: 0.0086 - acc: 0
Epoch 8/12
60000/60000 [==============================] - 13s 220us/step - loss: 0.0070 - acc: 0
Epoch 9/12
60000/60000 [==============================] - 13s 220us/step - loss: 0.0058 - acc: 0
Epoch 10/12
60000/60000 [==============================] - 13s 220us/step - loss: 0.0049 - acc: 0
Epoch 11/12
60000/60000 [==============================] - 13s 220us/step - loss: 0.0041 - acc: 0
Epoch 12/12
60000/60000 [==============================] - 13s 220us/step - loss: 0.0036 - acc: 0
Test loss: 0.023953616692498327
Test accuracy: 0.993
```

```
# 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,BatchNormalization
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()

print(x_train.shape)

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=(2, 2),activation='tanh',input_shape=input_shape,kernel_
model.add(Conv2D(64, (3, 3), activation='tanh'))
model.add(Dropout(0.5))
model.add(Conv2D(32, (5, 5), activation='tanh'))
model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
```

```
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.Adadel

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

```
(60000, 28, 28)
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 27, 27, 128)       640
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 25, 25, 64)        73792
_________________________________________________________________
dropout_3 (Dropout)          (None, 25, 25, 64)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 21, 21, 32)        51232
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 10, 10, 32)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 3200)              0
_________________________________________________________________
dense_5 (Dense)              (None, 128)               409728
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1290
=================================================================
Total params: 536,682
Trainable params: 536,682
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 19s 316us/step - loss: 0.2839 - acc: 0
Epoch 2/12
60000/60000 [==============================] - 18s 303us/step - loss: 0.1130 - acc: 0
Epoch 3/12
60000/60000 [==============================] - 18s 302us/step - loss: 0.0850 - acc: 0
Epoch 4/12
60000/60000 [==============================] - 18s 303us/step - loss: 0.0677 - acc: 0
Epoch 5/12
60000/60000 [==============================] - 18s 303us/step - loss: 0.0577 - acc: 0
Epoch 6/12
60000/60000 [==============================] - 18s 303us/step - loss: 0.0522 - acc: 0
Epoch 7/12
60000/60000 [==============================] - 18s 302us/step - loss: 0.0453 - acc: 0
Epoch 8/12
60000/60000 [==============================] - 18s 303us/step - loss: 0.0417 - acc: 0
Epoch 9/12
60000/60000 [==============================] - 18s 305us/step - loss: 0.0383 - acc: 0
Epoch 10/12
60000/60000 [==============================] - 18s 303us/step - loss: 0.0331 - acc: 0
Epoch 11/12
60000/60000 [==============================] - 18s 302us/step - loss: 0.0297 - acc: 0
Epoch 12/12
60000/60000 [==============================] - 18s 302us/step - loss: 0.0289 - acc: 0
Test loss: 0.02994774283710726
Test accuracy: 0.9904
```

```
# 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,BatchNormalization
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()

print(x_train.shape)

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=(2, 2),activation='relu',input_shape=input_shape,kernel_
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(32, (5, 5), activation='relu'))



model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.summary()
```

```
model.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.sgd(),

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

```
(60000, 28, 28)
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 27, 27, 128)       640
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 25, 25, 64)        73792
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 21, 21, 32)        51232
_________________________________________________________________
flatten_4 (Flatten)          (None, 14112)             0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               1806464
_________________________________________________________________
dense_8 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,933,418
Trainable params: 1,933,418
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 17s 276us/step - loss: 0.6677 - acc: 0
Epoch 2/12
60000/60000 [==============================] - 16s 266us/step - loss: 0.2070 - acc: 0
Epoch 3/12
60000/60000 [==============================] - 16s 267us/step - loss: 0.1461 - acc: 0
Epoch 4/12
60000/60000 [==============================] - 16s 266us/step - loss: 0.1135 - acc: 0
Epoch 5/12
60000/60000 [==============================] - 16s 266us/step - loss: 0.0935 - acc: 0
Epoch 6/12
60000/60000 [==============================] - 16s 267us/step - loss: 0.0785 - acc: 0
Epoch 7/12
60000/60000 [==============================] - 16s 267us/step - loss: 0.0676 - acc: 0
Epoch 8/12
60000/60000 [==============================] - 16s 266us/step - loss: 0.0576 - acc: 0
Epoch 9/12
60000/60000 [==============================] - 16s 266us/step - loss: 0.0502 - acc: 0
Epoch 10/12
60000/60000 [==============================] - 16s 267us/step - loss: 0.0447 - acc: 0
Epoch 11/12
60000/60000 [==============================] - 16s 266us/step - loss: 0.0402 - acc: 0
Epoch 12/12
60000/60000 [==============================] - 16s 266us/step - loss: 0.0360 - acc: 0
Test loss: 0.06601626948665362
Test accuracy: 0.979
```

5-Layers

```
# 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 = 512
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()

print(x_train.shape)

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=(2, 2),activation='relu',input_shape=input_shape,kernel_
model.add(Conv2D(96, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

```
model.add(Conv2D(84, (5, 5),padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3),padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))



model.add(Conv2D(32, (7, 7),padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

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.summary()

model.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.Adam()

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

```
(60000, 28, 28)
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 27, 27, 128)       640
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 25, 25, 96)        110688
_________________________________________________________________
batch_normalization_5 (Batch (None, 25, 25, 96)        384
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 12, 12, 96)        0
_________________________________________________________________
dropout_5 (Dropout)          (None, 12, 12, 96)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 12, 12, 84)        201684
_________________________________________________________________
batch_normalization_6 (Batch (None, 12, 12, 84)        336
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 12, 12, 64)        48448
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_6 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 6, 6, 32)          100384
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 3, 3, 32)          0
_________________________________________________________________
dropout_7 (Dropout)          (None, 3, 3, 32)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 288)               0
_________________________________________________________________
dense_9 (Dense)              (None, 128)               36992
_________________________________________________________________
dropout_8 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_10 (Dense)             (None, 10)                1290
=================================================================
Total params: 500,846
Trainable params: 500,486
Non-trainable params: 360
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 26s 430us/step - loss: 0.7600 - acc: 0
Epoch 2/10
60000/60000 [==============================] - 23s 377us/step - loss: 0.1271 - acc: 0
Epoch 3/10
60000/60000 [==============================] - 23s 376us/step - loss: 0.0804 - acc: 0
Epoch 4/10
60000/60000 [==============================] - 23s 376us/step - loss: 0.0699 - acc: 0
Epoch 5/10
60000/60000 [==============================] - 23s 376us/step - loss: 0.0552 - acc: 0
Epoch 6/10
60000/60000 [==============================] - 23s 377us/step - loss: 0.0461 - acc: 0
Epoch 7/10
60000/60000 [==============================] - 23s 375us/step - loss: 0.0419 - acc: 0
```

```
Epoch 8/10
60000/60000 [==============================] - 23s 376us/step - loss: 0.0383 - acc: 0
Epoch 9/10
60000/60000 [==============================] - 23s 376us/step - loss: 0.0346 - acc: 0
Epoch 10/10
60000/60000 [==============================] - 23s 376us/step - loss: 0.0296 - acc: 0
Test loss: 0.02720410974002425
Test accuracy: 0.9921
```

```
# 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 = 512
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()

print(x_train.shape)

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=(2, 2),activation='relu',input_shape=input_shape,kernel
```

```
model.add(Conv2D(128, kernel_size (2, 2),activation ='relu',input_shape=input_shape,kernel_
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(96, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())


model.add(Conv2D(84, (5, 5),padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3),padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (7, 7),padding='same', activation='relu'))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.Adam()

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

```
(60000, 28, 28)
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 27, 27, 128)       640
_________________________________________________________________
batch_normalization_7 (Batch (None, 27, 27, 128)       512
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 13, 13, 128)       0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 11, 11, 96)        110688
_________________________________________________________________
batch_normalization_8 (Batch (None, 11, 11, 96)        384
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 5, 5, 96)          0
_________________________________________________________________
batch_normalization_9 (Batch (None, 5, 5, 96)          384
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 5, 5, 84)          201684
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 2, 2, 84)          0
_________________________________________________________________
batch_normalization_10 (Batc (None, 2, 2, 84)          336
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 2, 2, 64)          48448
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 1, 1, 64)          0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 1, 1, 32)          100384
_________________________________________________________________
flatten_6 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_11 (Dense)             (None, 128)               4224
_________________________________________________________________
dense_12 (Dense)             (None, 10)                1290
=================================================================
Total params: 468,974
Trainable params: 468,166
Non-trainable params: 808
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 17s 278us/step - loss: 0.4008 - acc: 0
Epoch 2/10
60000/60000 [==============================] - 15s 244us/step - loss: 0.0344 - acc: 0
Epoch 3/10
60000/60000 [==============================] - 15s 244us/step - loss: 0.0218 - acc: 0
Epoch 4/10
60000/60000 [==============================] - 15s 244us/step - loss: 0.0127 - acc: 0
Epoch 5/10
60000/60000 [==============================] - 15s 244us/step - loss: 0.0077 - acc: 0
Epoch 6/10
60000/60000 [==============================] - 15s 243us/step - loss: 0.0070 - acc: 0
Epoch 7/10
60000/60000 [==============================] - 15s 243us/step - loss: 0.0045 - acc: 0
Epoch 8/10
60000/60000 [==============================] - 15s 244us/step - loss: 0.0050 - acc: 0
```

```
Epoch 9/10
60000/60000 [==============================] - 15s 245us/step - loss: 0.0061 - acc: 0
Epoch 10/10
60000/60000 [==============================] - 15s 243us/step - loss: 0.0056 - acc: 0
Test loss: 0.04050913316059923
Test accuracy: 0.9901
```

```
# 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 = 512
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()

print(x_train.shape)

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(512, kernel_size=(2, 2),activation='sigmoid',input_shape=input_shape,kern
model.add(Conv2D(256, (3, 3),padding='same', activation='sigmoid'))
model.add(Conv2D(128, (5, 5),padding='same', activation='sigmoid'))
```

```
model.add(Conv2D(128, (5, 5),padding='same', activation='sigmoid'))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))


model.add(Conv2D(64, (3, 3),padding='same', activation='sigmoid'))
model.add(Conv2D(32, (7, 7),padding='same', activation='sigmoid'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))



model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(128, activation='sigmoid'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.Adagra

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

```
(60000, 28, 28)
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_23 (Conv2D)           (None, 27, 27, 512)       2560
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 27, 27, 256)       1179904
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 27, 27, 128)       819328
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 13, 13, 128)       0
_________________________________________________________________
dropout_9 (Dropout)          (None, 13, 13, 128)       0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 13, 13, 64)        73792
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 13, 13, 32)        100384
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 6, 6, 32)          0
_________________________________________________________________
dropout_10 (Dropout)         (None, 6, 6, 32)          0
_________________________________________________________________
dropout_11 (Dropout)         (None, 6, 6, 32)          0
_________________________________________________________________
flatten_7 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_13 (Dense)             (None, 128)               147584
_________________________________________________________________
dropout_12 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_14 (Dense)             (None, 10)                1290
=================================================================
Total params: 2,324,842
Trainable params: 2,324,842
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 108s 2ms/step - loss: 2.3278 - acc: 0.
Epoch 2/10
60000/60000 [==============================] - 94s 2ms/step - loss: 2.3027 - acc: 0.1
Epoch 3/10
60000/60000 [==============================] - 94s 2ms/step - loss: 2.3020 - acc: 0.1
Epoch 4/10
60000/60000 [==============================] - 94s 2ms/step - loss: 2.3016 - acc: 0.1
Epoch 5/10
60000/60000 [==============================] - 94s 2ms/step - loss: 2.3015 - acc: 0.1
Epoch 6/10
60000/60000 [==============================] - 94s 2ms/step - loss: 2.3017 - acc: 0.1
Epoch 7/10
60000/60000 [==============================] - 94s 2ms/step - loss: 2.3014 - acc: 0.1
Epoch 8/10
60000/60000 [==============================] - 94s 2ms/step - loss: 2.3015 - acc: 0.1
Epoch 9/10
60000/60000 [==============================] - 94s 2ms/step - loss: 2.3012 - acc: 0.1
Epoch 10/10
60000/60000 [==============================] - 94s 2ms/step - loss: 2.3012 - acc: 0.1
```

```
Test loss: 2.301075766372681
Test accuracy: 0.1135
```

```
# 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 = 512
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()

print(x_train.shape)

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(256, kernel_size=(2, 2),padding='same',activation='tanh',input_shape=inpu
model.add(Conv2D(128, (3, 3),padding='same', activation='tanh'))
model.add(Conv2D(128, (3, 3),padding='same', activation='tanh'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Conv2D(64, (5, 5),padding='same', activation='tanh'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (7, 7),padding='same', activation='tanh'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(128, activation='tanh'))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.Adadel

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

```
(60000, 28, 28)
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_28 (Conv2D)           (None, 28, 28, 256)       1280
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 28, 28, 128)       295040
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 28, 28, 128)       147584
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 14, 14, 64)        204864
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 7, 7, 32)          100384
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 3, 3, 32)          0
_________________________________________________________________
flatten_8 (Flatten)          (None, 288)               0
_________________________________________________________________
dense_15 (Dense)             (None, 128)               36992
_________________________________________________________________
dense_16 (Dense)             (None, 10)                1290
=================================================================
Total params: 787,434
Trainable params: 787,434
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 50s 830us/step - loss: 0.3525 - acc: 0
Epoch 2/10
60000/60000 [==============================] - 44s 728us/step - loss: 0.0633 - acc: 0
Epoch 3/10
60000/60000 [==============================] - 44s 728us/step - loss: 0.0406 - acc: 0
Epoch 4/10
60000/60000 [==============================] - 44s 729us/step - loss: 0.0224 - acc: 0
Epoch 5/10
60000/60000 [==============================] - 44s 729us/step - loss: 0.0141 - acc: 0
Epoch 6/10
60000/60000 [==============================] - 44s 729us/step - loss: 0.0083 - acc: 0
Epoch 7/10
60000/60000 [==============================] - 44s 729us/step - loss: 0.0044 - acc: 0
Epoch 8/10
60000/60000 [==============================] - 44s 729us/step - loss: 0.0023 - acc: 0
Epoch 9/10
60000/60000 [==============================] - 44s 729us/step - loss: 0.0015 - acc: 0
Epoch 10/10
60000/60000 [==============================] - 44s 729us/step - loss: 0.0010 - acc: 1
Test loss: 0.021898437683159137
Test accuracy: 0.9925
```

7-layers

```
# 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,BatchNormalization
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()

print(x_train.shape)

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=(2, 2),padding='same',activation='relu',input_shape=inpu
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3),padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))
```

```
model.add(Conv2D(64, (3, 3),padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Conv2D(64, (5, 5),padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (5, 5),padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Conv2D(32, (7, 7),padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(16, (7, 7),padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Flatten())
model.add(Dense(400, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.Adam()

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

```
(60000, 28, 28)
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_33 (Conv2D)           (None, 28, 28, 128)       640
_________________________________________________________________
batch_normalization_11 (Batc (None, 28, 28, 128)       512
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 28, 28, 128)       147584
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 14, 14, 128)       0
_________________________________________________________________
dropout_13 (Dropout)         (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 14, 14, 64)        73792
_________________________________________________________________
batch_normalization_12 (Batc (None, 14, 14, 64)        256
_________________________________________________________________
dropout_14 (Dropout)         (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_36 (Conv2D)           (None, 14, 14, 64)        102464
_________________________________________________________________
batch_normalization_13 (Batc (None, 14, 14, 64)        256
_________________________________________________________________
dropout_15 (Dropout)         (None, 14, 14, 64)        0
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_37 (Conv2D)           (None, 7, 7, 32)          51232
_________________________________________________________________
batch_normalization_14 (Batc (None, 7, 7, 32)          128
_________________________________________________________________
dropout_16 (Dropout)         (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 7, 7, 32)          50208
_________________________________________________________________
batch_normalization_15 (Batc (None, 7, 7, 32)          128
_________________________________________________________________
dropout_17 (Dropout)         (None, 7, 7, 32)          0
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 3, 3, 32)          0
_________________________________________________________________
conv2d_39 (Conv2D)           (None, 3, 3, 16)          25104
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 1, 1, 16)          0
_________________________________________________________________
flatten_9 (Flatten)          (None, 16)                0
_________________________________________________________________
dense_17 (Dense)             (None, 400)               6800
_________________________________________________________________
dense_18 (Dense)             (None, 200)               80200
_________________________________________________________________
dropout_18 (Dropout)         (None, 200)               0
_________________________________________________________________
dense_19 (Dense)             (None, 10)                2010
=================================================================
Total params: 541,314
```

```
Trainable params: 540,674
Non-trainable params: 640
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 44s 732us/step - loss: 0.5045 - acc: 0
Epoch 2/12
60000/60000 [==============================] - 41s 680us/step - loss: 0.1028 - acc: 0
Epoch 3/12
60000/60000 [==============================] - 41s 684us/step - loss: 0.0731 - acc: 0
Epoch 4/12
60000/60000 [==============================] - 41s 683us/step - loss: 0.0622 - acc: 0
Epoch 5/12
60000/60000 [==============================] - 41s 681us/step - loss: 0.0551 - acc: 0
Epoch 6/12
60000/60000 [==============================] - 41s 681us/step - loss: 0.0519 - acc: 0
Epoch 7/12
60000/60000 [==============================] - 41s 680us/step - loss: 0.0466 - acc: 0
Epoch 8/12
60000/60000 [==============================] - 41s 682us/step - loss: 0.0440 - acc: 0
Epoch 9/12
60000/60000 [==============================] - 41s 680us/step - loss: 0.0423 - acc: 0
Epoch 10/12
60000/60000 [==============================] - 41s 681us/step - loss: 0.0387 - acc: 0
Epoch 11/12
60000/60000 [==============================] - 41s 682us/step - loss: 0.0391 - acc: 0
Epoch 12/12
60000/60000 [==============================] - 41s 680us/step - loss: 0.0372 - acc: 0
Test loss: 0.038856790525408
Test accuracy: 0.99
```

```
# 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,BatchNormalization
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()

print(x_train.shape)

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(256, kernel_size=(2, 2),padding='same',activation='relu',input_shape=inpu
model.add(Conv2D(128, (3, 3),padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3),padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3),padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (5, 5),padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (5, 5),padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(26, (7, 7),padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Flatten())
model.add(Dense(400, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.Adam()

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

```
(60000, 28, 28)
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_40 (Conv2D)           (None, 28, 28, 256)       1280
_________________________________________________________________
conv2d_41 (Conv2D)           (None, 28, 28, 128)       295040
_________________________________________________________________
batch_normalization_16 (Batc (None, 28, 28, 128)       512
_________________________________________________________________
conv2d_42 (Conv2D)           (None, 28, 28, 128)       147584
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_43 (Conv2D)           (None, 14, 14, 64)        73792
_________________________________________________________________
batch_normalization_17 (Batc (None, 14, 14, 64)        256
_________________________________________________________________
conv2d_44 (Conv2D)           (None, 14, 14, 32)        51232
_________________________________________________________________
batch_normalization_18 (Batc (None, 14, 14, 32)        128
_________________________________________________________________
conv2d_45 (Conv2D)           (None, 14, 14, 32)        25632
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_46 (Conv2D)           (None, 7, 7, 26)          40794
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 3, 3, 26)          0
_________________________________________________________________
flatten_10 (Flatten)         (None, 234)               0
_________________________________________________________________
dense_20 (Dense)             (None, 400)               94000
_________________________________________________________________
dense_21 (Dense)             (None, 200)               80200
_________________________________________________________________
dropout_19 (Dropout)         (None, 200)               0
_________________________________________________________________
dense_22 (Dense)             (None, 10)                2010
=================================================================
Total params: 812,460
Trainable params: 812,012
Non-trainable params: 448
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 63s 1ms/step - loss: 0.1832 - acc: 0.9
Epoch 2/12
60000/60000 [==============================] - 60s 997us/step - loss: 0.0513 - acc: 0
Epoch 3/12
60000/60000 [==============================] - 60s 996us/step - loss: 0.0415 - acc: 0
Epoch 4/12
60000/60000 [==============================] - 60s 997us/step - loss: 0.0297 - acc: 0
Epoch 5/12
60000/60000 [==============================] - 60s 998us/step - loss: 0.0284 - acc: 0
Epoch 6/12
60000/60000 [==============================] - 60s 996us/step - loss: 0.0219 - acc: 0
```

```
Epoch 7/12
60000/60000 [==============================] - 60s 997us/step - loss: 0.0241 - acc: 0
Epoch 8/12
60000/60000 [==============================] - 60s 995us/step - loss: 0.0188 - acc: 0
Epoch 9/12
60000/60000 [==============================] - 60s 995us/step - loss: 0.0188 - acc: 0
Epoch 10/12
60000/60000 [==============================] - 60s 996us/step - loss: 0.0168 - acc: 0
Epoch 11/12
60000/60000 [==============================] - 60s 997us/step - loss: 0.0152 - acc: 0
Epoch 12/12
60000/60000 [==============================] - 60s 994us/step - loss: 0.0144 - acc: 0
Test loss: 0.03264233518390265
Test accuracy: 0.9923
```

```
# 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,BatchNormalization
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()

print(x_train.shape)

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(256, kernel_size=(2, 2),padding='same',activation='sigmoid',input_shape=i
model.add(Conv2D(128, (3, 3),padding='same', activation='sigmoid'))
model.add(Dropout(0.5))
model.add(Conv2D(128, (3, 3),padding='same', activation='sigmoid'))
model.add(Conv2D(128, (5, 5),padding='same', activation='sigmoid'))
model.add(Dropout(0.5))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (5, 5),padding='same', activation='sigmoid'))
model.add(Conv2D(128, (7, 7),padding='same', activation='sigmoid'))
model.add(Dropout(0.5))
model.add(Conv2D(128, (7, 7),padding='same', activation='sigmoid'))
model.add(Dropout(0.5))
model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Flatten())
model.add(Dense(400, activation='sigmoid'))
model.add(Dense(200, activation='sigmoid'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.Adagra

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

```
(60000, 28, 28)
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_47 (Conv2D)           (None, 28, 28, 256)       1280
_________________________________________________________________
conv2d_48 (Conv2D)           (None, 28, 28, 128)       295040
_________________________________________________________________
dropout_20 (Dropout)         (None, 28, 28, 128)       0
_________________________________________________________________
conv2d_49 (Conv2D)           (None, 28, 28, 128)       147584
_________________________________________________________________
conv2d_50 (Conv2D)           (None, 28, 28, 128)       409728
_________________________________________________________________
dropout_21 (Dropout)         (None, 28, 28, 128)       0
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_51 (Conv2D)           (None, 14, 14, 128)       409728
_________________________________________________________________
conv2d_52 (Conv2D)           (None, 14, 14, 128)       802944
_________________________________________________________________
dropout_22 (Dropout)         (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_53 (Conv2D)           (None, 14, 14, 128)       802944
_________________________________________________________________
dropout_23 (Dropout)         (None, 14, 14, 128)       0
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 7, 7, 128)         0
_________________________________________________________________
flatten_11 (Flatten)         (None, 6272)              0
_________________________________________________________________
dense_23 (Dense)             (None, 400)               2509200
_________________________________________________________________
dense_24 (Dense)             (None, 200)               80200
_________________________________________________________________
dropout_24 (Dropout)         (None, 200)               0
_________________________________________________________________
dense_25 (Dense)             (None, 10)                2010
=================================================================
Total params: 5,460,658
Trainable params: 5,460,658
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 103s 2ms/step - loss: 2.3165 - acc: 0.
Epoch 2/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3020 - acc: 0.1
Epoch 3/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3017 - acc: 0.1
Epoch 4/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3015 - acc: 0.1
Epoch 5/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3017 - acc: 0.1
Epoch 6/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3014 - acc: 0.1
```

```
Epoch 7/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3015 - acc: 0.1
Epoch 8/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3015 - acc: 0.1
Epoch 9/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3013 - acc: 0.1
Epoch 10/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3014 - acc: 0.1
Epoch 11/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3014 - acc: 0.1
Epoch 12/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3014 - acc: 0.1
Test loss: 2.3010334442138674
Test accuracy: 0.1135
```

```
# 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,BatchNormalization
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()

print(x_train.shape)

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(512, kernel_size=(2, 2),padding='same',activation='tanh',input_shape=inpu
model.add(Conv2D(256, (3, 3),padding='same', activation='tanh'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3),padding='same', activation='tanh'))
model.add(Conv2D(128, (5, 5),padding='same', activation='tanh'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (5, 5),padding='same', activation='tanh'))
model.add(Conv2D(64, (5, 5),padding='same', activation='tanh'))
model.add(Conv2D(32, (7, 7),padding='same', activation='tanh'))
model.add(Conv2D(16, (7, 7),padding='same', activation='tanh'))
model.add(MaxPooling2D(pool_size=(2, 2)))




model.add(Flatten())
model.add(Dense(400, activation='tanh'))
model.add(Dense(200, activation='tanh'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.Adadel

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

```
(60000, 28, 28)
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_54 (Conv2D)           (None, 28, 28, 512)       2560
_________________________________________________________________
conv2d_55 (Conv2D)           (None, 28, 28, 256)       1179904
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 14, 14, 256)       0
_________________________________________________________________
conv2d_56 (Conv2D)           (None, 14, 14, 128)       295040
_________________________________________________________________
conv2d_57 (Conv2D)           (None, 14, 14, 128)       409728
_________________________________________________________________
max_pooling2d_28 (MaxPooling (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_58 (Conv2D)           (None, 7, 7, 128)         409728
_________________________________________________________________
conv2d_59 (Conv2D)           (None, 7, 7, 64)          204864
_________________________________________________________________
conv2d_60 (Conv2D)           (None, 7, 7, 32)          100384
_________________________________________________________________
conv2d_61 (Conv2D)           (None, 7, 7, 16)          25104
_________________________________________________________________
max_pooling2d_29 (MaxPooling (None, 3, 3, 16)          0
_________________________________________________________________
flatten_12 (Flatten)         (None, 144)               0
_________________________________________________________________
dense_26 (Dense)             (None, 400)               58000
_________________________________________________________________
dense_27 (Dense)             (None, 200)               80200
_________________________________________________________________
dropout_25 (Dropout)         (None, 200)               0
_________________________________________________________________
dense_28 (Dense)             (None, 10)                2010
=================================================================
Total params: 2,767,522
Trainable params: 2,767,522
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 105s 2ms/step - loss: 2.4997 - acc: 0.
Epoch 2/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3530 - acc: 0.0
Epoch 3/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3278 - acc: 0.1
Epoch 4/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3145 - acc: 0.1
Epoch 5/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3095 - acc: 0.1
Epoch 6/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3083 - acc: 0.1
Epoch 7/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3080 - acc: 0.1
Epoch 8/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3074 - acc: 0.1
```

```
Epoch 9/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3076 - acc: 0.1
Epoch 10/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3069 - acc: 0.1
Epoch 11/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3072 - acc: 0.1
Epoch 12/12
60000/60000 [==============================] - 99s 2ms/step - loss: 2.3066 - acc: 0.1
Test loss: 2.305227696228027
Test accuracy: 0.1135
```

Summary

```
#pretty table
#c-Convoluted layer M-Maxpool layer
from prettytable import PrettyTable
x = PrettyTable()

x.field_names = ["No of concoluted layers","Batch_Normalization", "Activation_function", "

x.add_row([3,"Y","Relu","Adam","Glorot Normal","CCMCM","Y",0.9919])
x.add_row([3,"Y","sigmoid","Adagrad","He_Normal","CMCCM","N",0.9930])
x.add_row([3,"N","tanh","Adadelta","Random Normal","CCCM","Y",0.9904])
x.add_row([3,"N","Relu","sgd","Glorot Normal","CCC","N",0.9790])

x.add_row([5,"Y","Relu","Adam","Random Normal","CCMCCMCM","Y",0.9921])
x.add_row([5,"Y","Relu","Adam","Random Normal","CMCMCMCMC","N",0.9901])
x.add_row([5,"N","sigmoid","Adagrad","He Normal","CCCMCCM","Y",0.1135])
x.add_row([5,"N","tanh","adadelta","Glorot Normal","CCCMCMCM","N",0.9925])



x.add_row([7,"Y","Relu","Adam","Random Normal","CCMCCMCCMCM","Y",0.9900])
x.add_row([7,"Y","Relu","Adam","Random Normal","CCCMCCCMCM","N",0.9923])
x.add_row([7,"N","sigmoid","Adagrad","He Normal","CCCCMCCCM","Y",0.1135])
x.add_row([7,"N","tanh","adadelta","Glorot Normal","CMCCMCCCCM","N",0.1135])
```

```
print(x)
```

No of concoluted layers	Batch_Normalization	Activation_function	optimizers
3	Y	Relu	Adam
3	Y	sigmoid	Adagrad
3	N	tanh	Adadelta
3	N	Relu	sgd
5	Y	Relu	Adam
5	Y	Relu	Adam
5	N	sigmoid	Adagrad
5	N	tanh	adadelta
7	Y	Relu	Adam
7	Y	Relu	Adam
7	N	sigmoid	Adagrad
7	N	tanh	adadelta

1.3 convoluted layers with sigmoid activation unit gives best accuarcy score followed by 7 convoluted layers wit

2.5 convoluted layers with sigmoid activation function and adagrad optimizer and 7 conoluted layers with sigmo