Nokia-vs-Hammer Classification

March 2, 2018

In [46]:
```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop
from skimage import io
import os
import numpy as np
```

0.0.1 hyperparameters

In [91]:
```
batch_size = 5
num_classes = 2
epochs = 5
```

0.0.2 train, test data

In [92]:
```
def read_images(path, label=1):
    data = []
    for img in os.listdir(path):
        data.append(io.imread(path + '/' + img, as_grey=True).reshape(1024))
    return data, [label] * len(data)
```

In [93]:
```
def compile_set(path, label):
    x_train, y_train = read_images(path + '/train', label)
    x_dev, y_dev = read_images(path + '/dev', label)
    x_test, y_test = read_images(path + '/test', label)
    return {'train': (x_train, y_train),
            'dev': (x_dev, y_dev),
            'test': (x_test, y_test)}
```

In [94]:
```
# the data, split between train and test sets
phone_set = compile_set('phone_dataset', label=0)
hammer_set = compile_set('hammer_dataset', label=1)

x_train = np.array(phone_set['train'][0] + hammer_set['train'][0])
y_train = np.array(phone_set['train'][1] + hammer_set['train'][1])
```

```
x_dev = np.array(phone_set['dev'][0] + hammer_set['dev'][0])
y_dev = np.array(phone_set['dev'][1] + hammer_set['dev'][1])

x_test = np.array(phone_set['test'][0] + hammer_set['test'][0])
y_test = np.array(phone_set['test'][1] + hammer_set['test'][1])

print(x_train.shape[0], 'train samples')
print(x_dev.shape[0], 'development samples')
print(x_test.shape[0], 'test samples')
```

```
20 train samples
4 development samples
4 test samples
```

0.0.3 neural network classifier

```
In [96]: model = Sequential()
         model.add(Dense(64, activation='relu', input_shape=(1024,)))
         model.add(Dropout(0.5))
         model.add(Dense(1, activation='sigmoid'))

         model.summary()

         model.compile(loss='binary_crossentropy',
                       optimizer=RMSprop(),
                       metrics=['accuracy'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_22 (Dense)             (None, 64)                65600
_________________________________________________________________
dropout_12 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_23 (Dense)             (None, 1)                 65
=================================================================
Total params: 65,665
Trainable params: 65,665
Non-trainable params: 0
_________________________________________________________________
```

0.0.4 training

```
In [97]: history = model.fit(x_train, y_train,
                             batch_size=batch_size,
                             epochs=epochs,
```

```
                          verbose=1,
                          validation_data=(x_dev, y_dev))
```

```
Train on 20 samples, validate on 4 samples
Epoch 1/5
20/20 [==============================] - 1s 44ms/step - loss: 7.4974 - acc: 0.5000 - val_loss: 7
Epoch 2/5
20/20 [==============================] - 0s 403us/step - loss: 9.6052 - acc: 0.4000 - val_loss:
Epoch 3/5
20/20 [==============================] - 0s 822us/step - loss: 7.9784 - acc: 0.5000 - val_loss:
Epoch 4/5
20/20 [==============================] - 0s 515us/step - loss: 7.9721 - acc: 0.5000 - val_loss:
Epoch 5/5
20/20 [==============================] - 0s 346us/step - loss: 7.9849 - acc: 0.5000 - val_loss:
```

```
In [98]: score = model.evaluate(x_dev, y_dev, verbose=0)
         print('Dev loss:', score[0])
         print('Dev accuracy:', score[1])
```

```
Dev loss: 7.971192359924316
Dev accuracy: 0.5
```

0.0.5 accuracy

```
In [99]: score = model.evaluate(x_test, y_test, verbose=0)
         print('Test loss:', score[0])
         print('Test accuracy:', score[1])
```

```
Test loss: 7.971192359924316
Test accuracy: 0.5
```