Classify different data sets

Basic includes

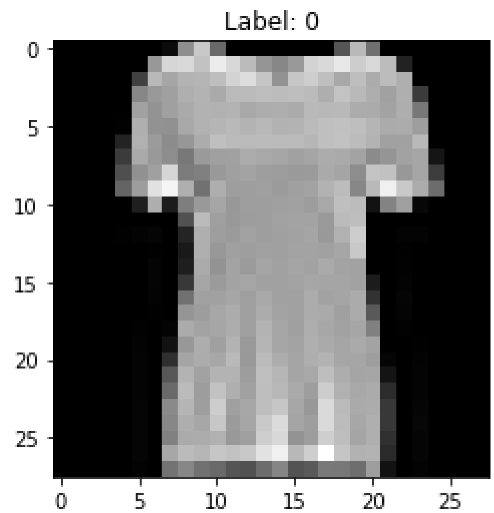
In [69]:
```
# Using pandas to load the csv file
import pandas as pd

import numpy as np
import matplotlib.pyplot as plt

from keras import models
from keras import layers
from keras import callbacks
from keras.utils import to_categorical

# reuters and fashin mnist data set from keras
from keras.datasets import reuters
from keras.datasets import fashion_mnist

# needed to preprocess text
from keras.preprocessing.text import Tokenizer
```

Classify the Fashion Mnist

In [70]:
```
(fashion_train_data, fashion_train_labels), (fashion_test_data, fashion_tes

print(fashion_train_data.shape)

test_index = 10

plt.title("Label: " + str(fashion_train_labels[test_index]))
plt.imshow(fashion_train_data[test_index], cmap="gray")
```

```
(60000, 28, 28)
```

Out[70]: <matplotlib.image.AxesImage at 0x12cbe73c8>



TO DO: Preprocess the data

1. Normalize the input data set
2. Perform one hot encoding
3. Create a train, test, and validation set

In [71]:
```
fashion_train_data = fashion_train_data.reshape((60000, 28 * 28))
fashion_train_data = fashion_train_data.astype('float32') / 255

# same starndadization for the test images
fashion_test_data = fashion_test_data.reshape((10000, 28 * 28))
fashion_test_data = fashion_test_data.astype('float32') / 255

# one hot encoding
fashion_train_labels = to_categorical(fashion_train_labels)
fashion_test_labels = to_categorical(fashion_test_labels)

validation_data = fashion_train_data[:10000]
validation_labels = fashion_train_labels[:10000]

x_data = fashion_train_data[50000:]
y_data = fashion_train_labels[50000:]

print(fashion_train_labels[0].shape)
```

```
(10,)
```

TO DO: Define and train a network, then plot the accuracy of the training, validation, and testing

1. Use a validation set
2. Propose and train a network
3. Print the history of the training
4. Evaluate with a test set

In [72]:
```
network = models.Sequential()

network.add(layers.Dense(64, activation='sigmoid', input_shape=(28 * 28,)))
network.add(layers.Dense(32, activation='sigmoid'))
network.add(layers.Dense(10, activation='softmax'))

network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metri

network.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_25 (Dense)             (None, 64)                50240
_________________________________________________________________
dense_26 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_27 (Dense)             (None, 10)                330
=================================================================
Total params: 52,650
Trainable params: 52,650
Non-trainable params: 0
_________________________________________________________________
```

In [73]:
```
early_stop = callbacks.EarlyStopping(monitor='val_loss', patience=5)
network.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

history = network.fit(fashion_train_data, fashion_train_labels,
          epochs=50,
          validation_split=0.2,
          callbacks=[early_stop],
          verbose=2)

test_loss, test_acc = network.evaluate(fashion_test_data, fashion_test_labe

print()
print("test loss: ", test_loss, "test accuracy: ", test_acc)
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/50
 - 6s - loss: 0.7771 - acc: 0.7482 - val_loss: 0.4881 - val_acc: 0.8299
Epoch 2/50
 - 4s - loss: 0.4412 - acc: 0.8434 - val_loss: 0.4091 - val_acc: 0.8540
Epoch 3/50
 - 4s - loss: 0.3907 - acc: 0.8612 - val_loss: 0.3968 - val_acc: 0.8578
Epoch 4/50
 - 4s - loss: 0.3652 - acc: 0.8693 - val_loss: 0.3709 - val_acc: 0.8663
Epoch 5/50
 - 4s - loss: 0.3468 - acc: 0.8749 - val_loss: 0.3575 - val_acc: 0.8705
Epoch 6/50
 - 4s - loss: 0.3319 - acc: 0.8812 - val_loss: 0.3481 - val_acc: 0.8732
Epoch 7/50
 - 5s - loss: 0.3219 - acc: 0.8844 - val_loss: 0.3364 - val_acc: 0.8785
Epoch 8/50
 - 4s - loss: 0.3114 - acc: 0.8881 - val_loss: 0.3350 - val_acc: 0.8781
Epoch 9/50
 - 4s - loss: 0.3036 - acc: 0.8909 - val_loss: 0.3392 - val_acc: 0.8780
Epoch 10/50
 - 4s - loss: 0.2956 - acc: 0.8923 - val_loss: 0.3368 - val_acc: 0.8792
Epoch 11/50
 - 5s - loss: 0.2892 - acc: 0.8947 - val_loss: 0.3396 - val_acc: 0.8787
Epoch 12/50
 - 4s - loss: 0.2822 - acc: 0.8973 - val_loss: 0.3328 - val_acc: 0.8770
Epoch 13/50
 - 4s - loss: 0.2763 - acc: 0.9002 - val_loss: 0.3199 - val_acc: 0.8869
Epoch 14/50
 - 4s - loss: 0.2713 - acc: 0.9015 - val_loss: 0.3316 - val_acc: 0.8845
Epoch 15/50
 - 4s - loss: 0.2644 - acc: 0.9037 - val_loss: 0.3217 - val_acc: 0.8855
Epoch 16/50
 - 4s - loss: 0.2601 - acc: 0.9055 - val_loss: 0.3312 - val_acc: 0.8865
Epoch 17/50
 - 4s - loss: 0.2560 - acc: 0.9069 - val_loss: 0.3208 - val_acc: 0.8880
Epoch 18/50
 - 4s - loss: 0.2517 - acc: 0.9081 - val_loss: 0.3234 - val_acc: 0.8853
10000/10000 [==============================] - 0s 34us/step

test loss:  0.3515301880478859 test accuracy:  0.8801
```

Classifying newswires

Build a network to classify Reuters newswires into 46 different mutually-exclusive topics.

Load and review the data

In [74]:
```
uters_train_data, reuters_train_labels), (reuters_test_data, reuters_test_l

nt(reuters_train_data.shape)
nt(reuters_train_labels.shape)
nt(reuters_train_data[0])
nt(reuters_train_labels[0])

nt(set(reuters_train_labels))
```

```
(8982,)
(8982,)
[1, 2, 2, 8, 43, 10, 447, 5, 25, 207, 270, 5, 3095, 111, 16, 369, 186, 9
0, 67, 7, 89, 5, 19, 102, 6, 19, 124, 15, 90, 67, 84, 22, 482, 26, 7, 48,
4, 49, 8, 864, 39, 209, 154, 6, 151, 6, 83, 11, 15, 22, 155, 11, 15, 7, 4
8, 9, 4579, 1005, 504, 6, 258, 6, 272, 11, 15, 22, 134, 44, 11, 15, 16,
8, 197, 1245, 90, 67, 52, 29, 209, 30, 32, 132, 6, 109, 15, 17, 12]
3
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2
0, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 3
8, 39, 40, 41, 42, 43, 44, 45}
```

Load the word index to decode the train data.

In [75]:
```
word_index = reuters.get_word_index()

reverse_index = dict([(value+3, key) for (key, value) in word_index.items()

reverse_index[0] = "<PAD>"
reverse_index[1] = "<START>"
reverse_index[2] = "<UNKNOWN>"  # unknown
reverse_index[3] = "<UNUSED>"

decoded_review = ' '.join([reverse_index.get(i,'?') for i in reuters_train_

print(decoded_review)

word_index = reuters.get_word_index()# Turning the output into vector mode,
tokenizer = Tokenizer(num_words=10000)
train_data_token = tokenizer.sequences_to_matrix(reuters_train_data, mode='
test_data_token = tokenizer.sequences_to_matrix(reuters_test_data, mode='co
print(train_data_token.shape)
print(test_data_token.shape)

# One-hot encoding the output
num_classes = 46
one_hot_train_labels = to_categorical(reuters_train_labels,num_classes)
one_hot_test_labels = to_categorical(reuters_test_labels, num_classes)
print(one_hot_train_labels.shape)
print(one_hot_test_labels.shape)
```

```
<START> <UNKNOWN> <UNKNOWN> said as a result of its december acquisition
of space co it expects earnings per share in 1987 of 1 15 to 1 30 dlrs pe
r share up from 70 cts in 1986 the company said pretax net should rise to
nine to 10 mln dlrs from six mln dlrs in 1986 and rental operation revenu
es to 19 to 22 mln dlrs from 12 5 mln dlrs it said cash flow per share th
is year should be 2 50 to three dlrs reuter 3
(8982, 10000)
(2246, 10000)
(8982, 46)
(2246, 46)
```

TO DO: Preprocess the data

1. Normalize the input data set
2. Perform one hot encoding
3. Create a train, test, and validation set

In [76]:

```
# Turning the output into vector mode, each of length 10000
tokenizer = Tokenizer(num_words=10000)
train_data_token = tokenizer.sequences_to_matrix(reuters_train_data, mode='
test_data_token = tokenizer.sequences_to_matrix(reuters_test_data, mode='bi
print(train_data_token.shape)
print(test_data_token.shape)

# One-hot encoding the output
num_classes = 46
one_hot_train_labels = to_categorical(reuters_train_labels,num_classes)
one_hot_test_labels = to_categorical(reuters_test_labels, num_classes)
print(one_hot_train_labels.shape)
print(one_hot_test_labels.shape)

# Creating a validation set with the first 1000 reviews
validation_data = train_data_token[:1000]
validation_labels = one_hot_train_labels[:1000]
```

```
(8982, 10000)
(2246, 10000)
(8982, 46)
(2246, 46)
```

In [77]:

```
net = models.Sequential()
net.add(layers.Dense(128, activation='relu', input_dim=10000))
net.add(layers.Dropout(0.3))
net.add(layers.Dense(64, activation='relu'))
net.add(layers.Dropout(0.3))
net.add(layers.Dense(num_classes, activation='softmax'))
net.summary()

# included the early stopping which monitors the validation loss
early_stop = callbacks.EarlyStopping(monitor='val_loss', patience=5)
net.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_28 (Dense)             (None, 128)               1280128
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_29 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_8 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_30 (Dense)             (None, 46)                2990
=================================================================
Total params: 1,291,374
Trainable params: 1,291,374
Non-trainable params: 0
_________________________________________________________________
```

In [78]:
```
history = net.fit(train_data_token, one_hot_train_labels,
          batch_size=512,
          epochs=40,
          validation_split=0.2,
          callbacks=[early_stop],
          verbose=2)

test_loss2, test_acc2 = net.evaluate(test_data_token, one_hot_test_labels)

print("test loss: ", test_loss2, "test accuracy: ", test_acc2)
```

```
Train on 7185 samples, validate on 1797 samples
Epoch 1/40
 - 3s - loss: 2.5539 - acc: 0.4553 - val_loss: 1.6511 - val_acc: 0.6561
Epoch 2/40
 - 2s - loss: 1.5520 - acc: 0.6593 - val_loss: 1.3376 - val_acc: 0.6973
Epoch 3/40
 - 2s - loss: 1.2427 - acc: 0.7250 - val_loss: 1.1932 - val_acc: 0.7357
Epoch 4/40
 - 2s - loss: 1.0347 - acc: 0.7729 - val_loss: 1.1218 - val_acc: 0.7518
Epoch 5/40
 - 2s - loss: 0.8862 - acc: 0.8054 - val_loss: 1.0285 - val_acc: 0.7802
Epoch 6/40
 - 2s - loss: 0.7650 - acc: 0.8267 - val_loss: 0.9957 - val_acc: 0.7858
Epoch 7/40
 - 2s - loss: 0.6656 - acc: 0.8493 - val_loss: 0.9751 - val_acc: 0.7913
Epoch 8/40
 - 2s - loss: 0.5750 - acc: 0.8667 - val_loss: 0.9558 - val_acc: 0.7991
Epoch 9/40
 - 2s - loss: 0.5208 - acc: 0.8823 - val_loss: 0.9680 - val_acc: 0.7947
Epoch 10/40
 - 2s - loss: 0.4682 - acc: 0.8924 - val_loss: 0.9497 - val_acc: 0.8030
Epoch 11/40
 - 2s - loss: 0.3974 - acc: 0.9115 - val_loss: 0.9494 - val_acc: 0.8075
Epoch 12/40
 - 2s - loss: 0.3634 - acc: 0.9179 - val_loss: 0.9629 - val_acc: 0.8047
Epoch 13/40
 - 2s - loss: 0.3222 - acc: 0.9262 - val_loss: 1.0048 - val_acc: 0.7930
Epoch 14/40
 - 2s - loss: 0.2949 - acc: 0.9312 - val_loss: 0.9887 - val_acc: 0.7997
Epoch 15/40
 - 2s - loss: 0.2657 - acc: 0.9379 - val_loss: 1.0341 - val_acc: 0.7891
Epoch 16/40
 - 2s - loss: 0.2495 - acc: 0.9431 - val_loss: 1.0301 - val_acc: 0.7963
2246/2246 [==============================] - 1s 239us/step
test loss:  1.0610846908413079 test accuracy:  0.7960819234194123
```

TO DO: Define and train a network, then plot the accuracy of the training, validation, and testing

1. Use a validation set
2. Propose and train a network
3. Print the history of the training
4. Evaluate with a test set

Predicting Student Admissions

Predict student admissions based on three pieces of data:

- GRE Scores
- GPA Scores
- Class rank

Load and visualize the data

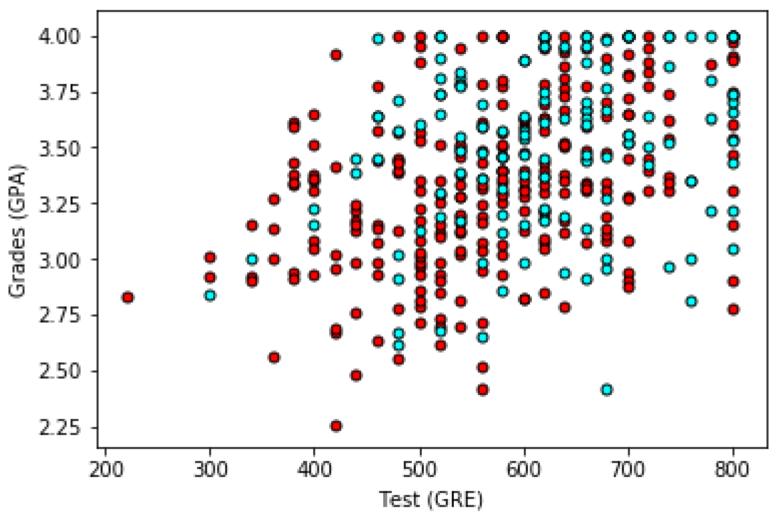
In [79]:
```python
student_data = pd.read_csv("student_data.csv")
print(student_data)
#plt.show(student_data)
```

```
     admit    gre   gpa  rank
0        0  380.0  3.61   3.0
1        1  660.0  3.67   3.0
2        1  800.0  4.00   1.0
3        1  640.0  3.19   4.0
4        0  520.0  2.93   4.0
5        1  760.0  3.00   2.0
6        1  560.0  2.98   1.0
7        0  400.0  3.08   2.0
8        1  540.0  3.39   3.0
9        0  700.0  3.92   2.0
10       0  800.0  4.00   4.0
11       0  440.0  3.22   1.0
12       1  760.0  4.00   1.0
13       0  700.0  3.08   2.0
14       1  700.0  4.00   1.0
15       0  480.0  3.44   3.0
16       0  780.0  3.87   4.0
17       0  360.0  2.56   3.0
18       0  800.0  3.75   2.0
19       1  540.0  3.81   1.0
20       0  500.0  3.17   3.0
21       1  660.0  3.63   2.0
22       0  600.0  2.82   4.0
23       0  680.0  3.19   4.0
24       1  760.0  3.35   2.0
25       1  800.0  3.66   1.0
26       1  620.0  3.61   1.0
27       1  520.0  3.74   4.0
28       1  780.0  3.22   2.0
29       0  520.0  3.29   1.0
..     ...    ...   ...   ...
370      1  540.0  3.77   2.0
371      1  680.0  3.76   3.0
372      1  680.0  2.42   1.0
373      1  620.0  3.37   1.0
374      0  560.0  3.78   2.0
375      0  560.0  3.49   4.0
376      0  620.0  3.63   2.0
377      1  800.0  4.00   2.0
378      0  640.0  3.12   3.0
379      0  540.0  2.70   2.0
380      0  700.0  3.65   2.0
381      1  540.0  3.49   2.0
382      0  540.0  3.51   2.0
383      0  660.0  4.00   1.0
384      1  480.0  2.62   2.0
385      0  420.0  3.02   1.0
386      1  740.0  3.86   2.0
387      0  580.0  3.36   2.0
```

```
388       0  640.0  3.17   2.0
389       0  640.0  3.51   2.0
390       1  800.0  3.05   2.0
391       1  660.0  3.88   2.0
392       1  600.0  3.38   3.0
393       1  620.0  3.75   2.0
394       1  460.0  3.99   3.0
395       0  620.0  4.00   2.0
396       0  560.0  3.04   3.0
397       0  460.0  2.63   2.0
398       0  700.0  3.65   2.0
399       0  600.0  3.89   3.0

[400 rows x 4 columns]
```

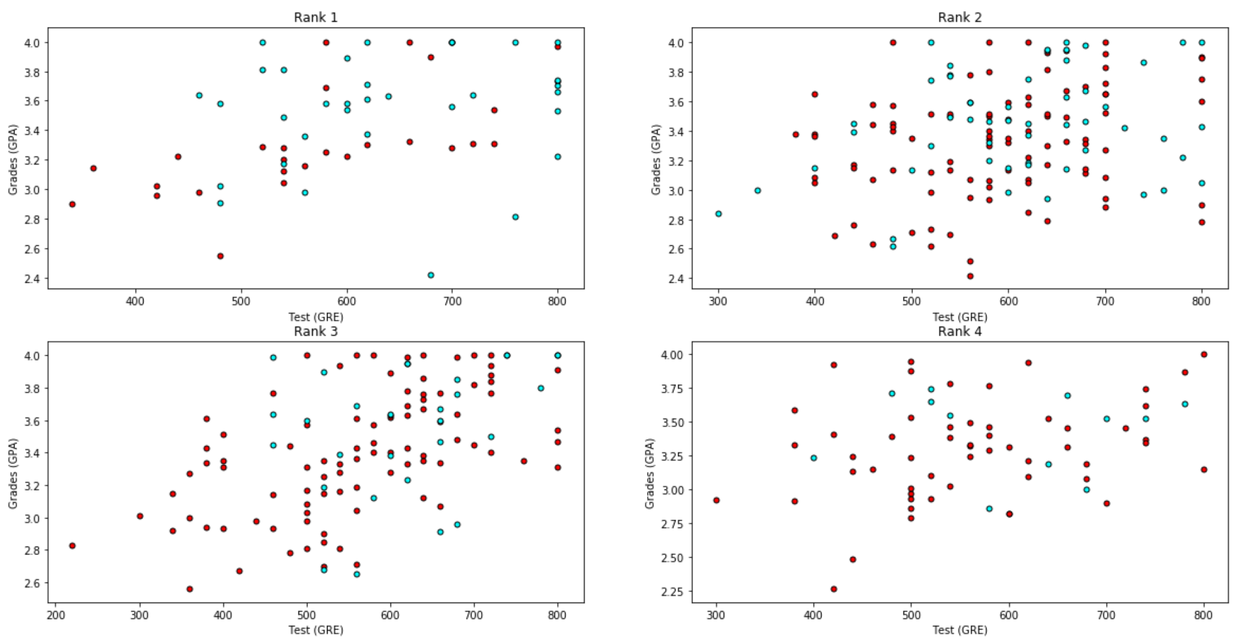
Plot of the GRE and the GPA from the data.

In [80]:
```
X = np.array(student_data[["gre","gpa"]])
y = np.array(student_data["admit"])
admitted = X[np.argwhere(y==1)]
rejected = X[np.argwhere(y==0)]
plt.scatter([s[0][0] for s in rejected], [s[0][1] for s in rejected], s = 2
plt.scatter([s[0][0] for s in admitted], [s[0][1] for s in admitted], s = 2
plt.xlabel('Test (GRE)')
plt.ylabel('Grades (GPA)')

plt.show()
```



Plot of the data by class rank.

In [81]:
```
f, plots = plt.subplots(2, 2, figsize=(20,10))
plots = [plot for sublist in plots for plot in sublist]

for idx, plot in enumerate(plots):
    data_rank = student_data[student_data["rank"]==idx+1]
    plot.set_title("Rank " + str(idx+1))
    X = np.array(data_rank[["gre","gpa"]])
    y = np.array(data_rank["admit"])
    admitted = X[np.argwhere(y==1)]
    rejected = X[np.argwhere(y==0)]
    plot.scatter([s[0][0] for s in rejected], [s[0][1] for s in rejected],
    plot.scatter([s[0][0] for s in admitted], [s[0][1] for s in admitted],
    plot.set_xlabel('Test (GRE)')
    plot.set_ylabel('Grades (GPA)')
```



TO DO: Preprocess the data

1. Normalize the input data set
2. Perform one hot encoding
3. Create a train, test, and validation set

In [82]:
```
# admit    gre   gpa  rank
student_data.fillna(value= 0, inplace= True)
gpa = np.array(student_data["gpa"])
gpa = (gpa - np.nanmean(gpa))/np.nanstd(gpa)
gre = np.array(student_data["gre"])
gre = (gre - np.nanmean(gre))/np.nanstd(gre)
rank = np.array(student_data["rank"])
rank = to_categorical(rank)
standarizedStudentData = np.zeros((len(gre), 2))
standarizedStudentData[:,0] = gre
standarizedStudentData[:,1] = gpa
studentDataFinal = np.zeros((len(gre), 7))
for i, (studentData, cat) in enumerate(zip(standarizedStudentData, rank)):
    studentDataFinal[i] = np.concatenate((studentData, cat))


admit = np.array(student_data["admit"])
studentOneHotLabels = to_categorical(admit)

print(studentDataFinal.shape)
print(studentOneHotLabels.shape)

studentTrainData = studentDataFinal[:300]
studentTrainLabels = studentOneHotLabels[:300]
studentTestData = studentDataFinal[300:]
studentTestLabels = studentOneHotLabels[300:]
```

```
(400, 7)
(400, 2)
```

In [83]:
```
#Keras Model
model2 = models.Sequential()
model2.add(layers.Dense(32, activation='relu', input_dim=7))
model2.add(layers.Dense(32, activation='relu'))
model2.add(layers.Dense(2, activation='sigmoid'))
model2.summary()

# included the early stopping which monitors the validation loss
early_stop = callbacks.EarlyStopping(monitor='val_loss', patience=8)
model2.compile(loss='kullback_leibler_divergence',
              optimizer='nadam',
              metrics=['accuracy'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_31 (Dense)             (None, 32)                256       
_________________________________________________________________
dense_32 (Dense)             (None, 32)                1056      
_________________________________________________________________
dense_33 (Dense)             (None, 2)                 66        
=================================================================
Total params: 1,378
Trainable params: 1,378
Non-trainable params: 0
_________________________________________________________________
```

In [84]:
```
history = model2.fit(studentTrainData, studentTrainLabels,
          batch_size=512,
          epochs=200,
          validation_split=0.2,
          verbose=2)
```

```
Train on 240 samples, validate on 60 samples
Epoch 1/200
 - 1s - loss: 0.6960 - acc: 0.3833 - val_loss: 0.6483 - val_acc: 0.3833
Epoch 2/200
 - 0s - loss: 0.6558 - acc: 0.4167 - val_loss: 0.6279 - val_acc: 0.4000
Epoch 3/200
 - 0s - loss: 0.6309 - acc: 0.4500 - val_loss: 0.6101 - val_acc: 0.4167
Epoch 4/200
 - 0s - loss: 0.6094 - acc: 0.4583 - val_loss: 0.5930 - val_acc: 0.4833
Epoch 5/200
 - 0s - loss: 0.5888 - acc: 0.4667 - val_loss: 0.5760 - val_acc: 0.4833
Epoch 6/200
 - 0s - loss: 0.5686 - acc: 0.5083 - val_loss: 0.5603 - val_acc: 0.5000
Epoch 7/200
 - 0s - loss: 0.5500 - acc: 0.5292 - val_loss: 0.5428 - val_acc: 0.5167
Epoch 8/200
 - 0s - loss: 0.5299 - acc: 0.5417 - val_loss: 0.5263 - val_acc: 0.5167
Epoch 9/200
 - 0s - loss: 0.5109 - acc: 0.5667 - val_loss: 0.5085 - val_acc: 0.5333
Epoch 10/200
 - 0s - loss: 0.4906 - acc: 0.5875 - val_loss: 0.4912 - val_acc: 0.5333
Epoch 11/200
 - 0s - loss: 0.4714 - acc: 0.6125 - val_loss: 0.4737 - val_acc: 0.5167
Epoch 12/200
 - 0s - loss: 0.4522 - acc: 0.6292 - val_loss: 0.4562 - val_acc: 0.5167
Epoch 13/200
 - 0s - loss: 0.4332 - acc: 0.6542 - val_loss: 0.4386 - val_acc: 0.5000
Epoch 14/200
 - 0s - loss: 0.4145 - acc: 0.6542 - val_loss: 0.4205 - val_acc: 0.5667
Epoch 15/200
 - 0s - loss: 0.3954 - acc: 0.6833 - val_loss: 0.4025 - val_acc: 0.5667
Epoch 16/200
 - 0s - loss: 0.3766 - acc: 0.7000 - val_loss: 0.3844 - val_acc: 0.6167
Epoch 17/200
 - 0s - loss: 0.3580 - acc: 0.7083 - val_loss: 0.3665 - val_acc: 0.6333
Epoch 18/200
 - 0s - loss: 0.3397 - acc: 0.7083 - val_loss: 0.3488 - val_acc: 0.6333
Epoch 19/200
 - 0s - loss: 0.3219 - acc: 0.7083 - val_loss: 0.3317 - val_acc: 0.6333
Epoch 20/200
 - 0s - loss: 0.3048 - acc: 0.7083 - val_loss: 0.3150 - val_acc: 0.6333
Epoch 21/200
 - 0s - loss: 0.2882 - acc: 0.7083 - val_loss: 0.2985 - val_acc: 0.6333
Epoch 22/200
 - 0s - loss: 0.2718 - acc: 0.7083 - val_loss: 0.2819 - val_acc: 0.6333
Epoch 23/200
 - 0s - loss: 0.2557 - acc: 0.7083 - val_loss: 0.2658 - val_acc: 0.6333
Epoch 24/200
 - 0s - loss: 0.2401 - acc: 0.7083 - val_loss: 0.2500 - val_acc: 0.6333
Epoch 25/200
```

```
 - 0s - loss: 0.2248 - acc: 0.7083 - val_loss: 0.2350 - val_acc: 0.6333
Epoch 26/200
 - 0s - loss: 0.2104 - acc: 0.7083 - val_loss: 0.2205 - val_acc: 0.6333
Epoch 27/200
 - 0s - loss: 0.1965 - acc: 0.7083 - val_loss: 0.2070 - val_acc: 0.6333
Epoch 28/200
 - 0s - loss: 0.1836 - acc: 0.7083 - val_loss: 0.1939 - val_acc: 0.6333
Epoch 29/200
 - 0s - loss: 0.1713 - acc: 0.7083 - val_loss: 0.1814 - val_acc: 0.6333
Epoch 30/200
 - 0s - loss: 0.1596 - acc: 0.7083 - val_loss: 0.1695 - val_acc: 0.6333
Epoch 31/200
 - 0s - loss: 0.1486 - acc: 0.7083 - val_loss: 0.1583 - val_acc: 0.6333
Epoch 32/200
 - 0s - loss: 0.1383 - acc: 0.7083 - val_loss: 0.1478 - val_acc: 0.6333
Epoch 33/200
 - 0s - loss: 0.1287 - acc: 0.7083 - val_loss: 0.1379 - val_acc: 0.6333
Epoch 34/200
 - 0s - loss: 0.1196 - acc: 0.7083 - val_loss: 0.1287 - val_acc: 0.6333
Epoch 35/200
 - 0s - loss: 0.1112 - acc: 0.7083 - val_loss: 0.1200 - val_acc: 0.6333
Epoch 36/200
 - 0s - loss: 0.1033 - acc: 0.7083 - val_loss: 0.1119 - val_acc: 0.6333
Epoch 37/200
 - 0s - loss: 0.0960 - acc: 0.7083 - val_loss: 0.1043 - val_acc: 0.6333
Epoch 38/200
 - 0s - loss: 0.0892 - acc: 0.7083 - val_loss: 0.0974 - val_acc: 0.6333
Epoch 39/200
 - 0s - loss: 0.0830 - acc: 0.7083 - val_loss: 0.0908 - val_acc: 0.6333
Epoch 40/200
 - 0s - loss: 0.0772 - acc: 0.7083 - val_loss: 0.0847 - val_acc: 0.6333
Epoch 41/200
 - 0s - loss: 0.0718 - acc: 0.7083 - val_loss: 0.0790 - val_acc: 0.6333
Epoch 42/200
 - 0s - loss: 0.0668 - acc: 0.7083 - val_loss: 0.0738 - val_acc: 0.6333
Epoch 43/200
 - 0s - loss: 0.0621 - acc: 0.7083 - val_loss: 0.0690 - val_acc: 0.6333
Epoch 44/200
 - 0s - loss: 0.0579 - acc: 0.7083 - val_loss: 0.0645 - val_acc: 0.6333
Epoch 45/200
 - 0s - loss: 0.0540 - acc: 0.7083 - val_loss: 0.0603 - val_acc: 0.6333
Epoch 46/200
 - 0s - loss: 0.0503 - acc: 0.7083 - val_loss: 0.0564 - val_acc: 0.6333
Epoch 47/200
 - 0s - loss: 0.0470 - acc: 0.7083 - val_loss: 0.0528 - val_acc: 0.6333
Epoch 48/200
 - 0s - loss: 0.0439 - acc: 0.7083 - val_loss: 0.0495 - val_acc: 0.6333
Epoch 49/200
 - 0s - loss: 0.0410 - acc: 0.7083 - val_loss: 0.0464 - val_acc: 0.6333
Epoch 50/200
 - 0s - loss: 0.0384 - acc: 0.7083 - val_loss: 0.0436 - val_acc: 0.6333
Epoch 51/200
 - 0s - loss: 0.0360 - acc: 0.7083 - val_loss: 0.0409 - val_acc: 0.6333
Epoch 52/200
 - 0s - loss: 0.0338 - acc: 0.7083 - val_loss: 0.0385 - val_acc: 0.6333
Epoch 53/200
 - 0s - loss: 0.0317 - acc: 0.7083 - val_loss: 0.0362 - val_acc: 0.6333
```

```
Epoch 54/200
 - 0s - loss: 0.0298 - acc: 0.7083 - val_loss: 0.0341 - val_acc: 0.6333
Epoch 55/200
 - 0s - loss: 0.0281 - acc: 0.7083 - val_loss: 0.0322 - val_acc: 0.6333
Epoch 56/200
 - 0s - loss: 0.0264 - acc: 0.7083 - val_loss: 0.0304 - val_acc: 0.6333
Epoch 57/200
 - 0s - loss: 0.0249 - acc: 0.7083 - val_loss: 0.0287 - val_acc: 0.6333
Epoch 58/200
 - 0s - loss: 0.0236 - acc: 0.7083 - val_loss: 0.0272 - val_acc: 0.6333
Epoch 59/200
 - 0s - loss: 0.0223 - acc: 0.7083 - val_loss: 0.0258 - val_acc: 0.6333
Epoch 60/200
 - 0s - loss: 0.0211 - acc: 0.7125 - val_loss: 0.0244 - val_acc: 0.6333
Epoch 61/200
 - 0s - loss: 0.0200 - acc: 0.7125 - val_loss: 0.0232 - val_acc: 0.6333
Epoch 62/200
 - 0s - loss: 0.0190 - acc: 0.7125 - val_loss: 0.0221 - val_acc: 0.6333
Epoch 63/200
 - 0s - loss: 0.0180 - acc: 0.7125 - val_loss: 0.0210 - val_acc: 0.6333
Epoch 64/200
 - 0s - loss: 0.0171 - acc: 0.7125 - val_loss: 0.0200 - val_acc: 0.6333
Epoch 65/200
 - 0s - loss: 0.0163 - acc: 0.7125 - val_loss: 0.0191 - val_acc: 0.6333
Epoch 66/200
 - 0s - loss: 0.0155 - acc: 0.7125 - val_loss: 0.0182 - val_acc: 0.6333
Epoch 67/200
 - 0s - loss: 0.0148 - acc: 0.7125 - val_loss: 0.0174 - val_acc: 0.6333
Epoch 68/200
 - 0s - loss: 0.0141 - acc: 0.7125 - val_loss: 0.0166 - val_acc: 0.6333
Epoch 69/200
 - 0s - loss: 0.0135 - acc: 0.7125 - val_loss: 0.0159 - val_acc: 0.6333
Epoch 70/200
 - 0s - loss: 0.0129 - acc: 0.7125 - val_loss: 0.0152 - val_acc: 0.6333
Epoch 71/200
 - 0s - loss: 0.0124 - acc: 0.7125 - val_loss: 0.0146 - val_acc: 0.6333
Epoch 72/200
 - 0s - loss: 0.0119 - acc: 0.7125 - val_loss: 0.0140 - val_acc: 0.6333
Epoch 73/200
 - 0s - loss: 0.0114 - acc: 0.7125 - val_loss: 0.0134 - val_acc: 0.6333
Epoch 74/200
 - 0s - loss: 0.0109 - acc: 0.7125 - val_loss: 0.0129 - val_acc: 0.6333
Epoch 75/200
 - 0s - loss: 0.0105 - acc: 0.7125 - val_loss: 0.0124 - val_acc: 0.6333
Epoch 76/200
 - 0s - loss: 0.0101 - acc: 0.7125 - val_loss: 0.0119 - val_acc: 0.6333
Epoch 77/200
 - 0s - loss: 0.0097 - acc: 0.7125 - val_loss: 0.0115 - val_acc: 0.6333
Epoch 78/200
 - 0s - loss: 0.0093 - acc: 0.7125 - val_loss: 0.0111 - val_acc: 0.6333
Epoch 79/200
 - 0s - loss: 0.0090 - acc: 0.7125 - val_loss: 0.0107 - val_acc: 0.6333
Epoch 80/200
 - 0s - loss: 0.0087 - acc: 0.7125 - val_loss: 0.0103 - val_acc: 0.6333
Epoch 81/200
 - 0s - loss: 0.0084 - acc: 0.7125 - val_loss: 0.0099 - val_acc: 0.6333
Epoch 82/200
```

```
 - 0s - loss: 0.0081 - acc: 0.7125 - val_loss: 0.0096 - val_acc: 0.6333
Epoch 83/200
 - 0s - loss: 0.0078 - acc: 0.7125 - val_loss: 0.0093 - val_acc: 0.6333
Epoch 84/200
 - 0s - loss: 0.0075 - acc: 0.7125 - val_loss: 0.0090 - val_acc: 0.6333
Epoch 85/200
 - 0s - loss: 0.0073 - acc: 0.7125 - val_loss: 0.0087 - val_acc: 0.6333
Epoch 86/200
 - 0s - loss: 0.0071 - acc: 0.7125 - val_loss: 0.0084 - val_acc: 0.6333
Epoch 87/200
 - 0s - loss: 0.0068 - acc: 0.7125 - val_loss: 0.0082 - val_acc: 0.6333
Epoch 88/200
 - 0s - loss: 0.0066 - acc: 0.7125 - val_loss: 0.0079 - val_acc: 0.6333
Epoch 89/200
 - 0s - loss: 0.0064 - acc: 0.7125 - val_loss: 0.0077 - val_acc: 0.6333
Epoch 90/200
 - 0s - loss: 0.0062 - acc: 0.7125 - val_loss: 0.0074 - val_acc: 0.6333
Epoch 91/200
 - 0s - loss: 0.0060 - acc: 0.7125 - val_loss: 0.0072 - val_acc: 0.6333
Epoch 92/200
 - 0s - loss: 0.0059 - acc: 0.7125 - val_loss: 0.0070 - val_acc: 0.6333
Epoch 93/200
 - 0s - loss: 0.0057 - acc: 0.7125 - val_loss: 0.0068 - val_acc: 0.6333
Epoch 94/200
 - 0s - loss: 0.0055 - acc: 0.7125 - val_loss: 0.0066 - val_acc: 0.6333
Epoch 95/200
 - 0s - loss: 0.0054 - acc: 0.7125 - val_loss: 0.0064 - val_acc: 0.6333
Epoch 96/200
 - 0s - loss: 0.0052 - acc: 0.7125 - val_loss: 0.0063 - val_acc: 0.6333

Epoch 97/200
 - 0s - loss: 0.0051 - acc: 0.7125 - val_loss: 0.0061 - val_acc: 0.6333
Epoch 98/200
 - 0s - loss: 0.0050 - acc: 0.7125 - val_loss: 0.0059 - val_acc: 0.6333
Epoch 99/200
 - 0s - loss: 0.0048 - acc: 0.7125 - val_loss: 0.0058 - val_acc: 0.6333
Epoch 100/200
 - 0s - loss: 0.0047 - acc: 0.7125 - val_loss: 0.0056 - val_acc: 0.6333
Epoch 101/200
 - 0s - loss: 0.0046 - acc: 0.7125 - val_loss: 0.0055 - val_acc: 0.6333
Epoch 102/200
 - 0s - loss: 0.0045 - acc: 0.7125 - val_loss: 0.0054 - val_acc: 0.6333
Epoch 103/200
 - 0s - loss: 0.0044 - acc: 0.7167 - val_loss: 0.0052 - val_acc: 0.6333
Epoch 104/200
 - 0s - loss: 0.0042 - acc: 0.7167 - val_loss: 0.0051 - val_acc: 0.6333
Epoch 105/200
 - 0s - loss: 0.0041 - acc: 0.7167 - val_loss: 0.0050 - val_acc: 0.6333
Epoch 106/200
 - 0s - loss: 0.0040 - acc: 0.7167 - val_loss: 0.0049 - val_acc: 0.6333
Epoch 107/200
 - 0s - loss: 0.0039 - acc: 0.7167 - val_loss: 0.0047 - val_acc: 0.6333
Epoch 108/200
 - 0s - loss: 0.0039 - acc: 0.7167 - val_loss: 0.0046 - val_acc: 0.6333
Epoch 109/200
 - 0s - loss: 0.0038 - acc: 0.7167 - val_loss: 0.0045 - val_acc: 0.6333
Epoch 110/200
```

```
 - 0s - loss: 0.0037 - acc: 0.7167 - val_loss: 0.0044 - val_acc: 0.6333
Epoch 111/200
 - 0s - loss: 0.0036 - acc: 0.7167 - val_loss: 0.0043 - val_acc: 0.6333
Epoch 112/200
 - 0s - loss: 0.0035 - acc: 0.7167 - val_loss: 0.0042 - val_acc: 0.6333
Epoch 113/200
 - 0s - loss: 0.0034 - acc: 0.7167 - val_loss: 0.0041 - val_acc: 0.6333
Epoch 114/200
 - 0s - loss: 0.0034 - acc: 0.7167 - val_loss: 0.0041 - val_acc: 0.6333
Epoch 115/200
 - 0s - loss: 0.0033 - acc: 0.7125 - val_loss: 0.0040 - val_acc: 0.6333
Epoch 116/200
 - 0s - loss: 0.0032 - acc: 0.7125 - val_loss: 0.0039 - val_acc: 0.6333
Epoch 117/200
 - 0s - loss: 0.0032 - acc: 0.7125 - val_loss: 0.0038 - val_acc: 0.6333
Epoch 118/200
 - 0s - loss: 0.0031 - acc: 0.7125 - val_loss: 0.0037 - val_acc: 0.6333
Epoch 119/200
 - 0s - loss: 0.0030 - acc: 0.7125 - val_loss: 0.0036 - val_acc: 0.6333
Epoch 120/200
 - 0s - loss: 0.0030 - acc: 0.7125 - val_loss: 0.0036 - val_acc: 0.6333
Epoch 121/200
 - 0s - loss: 0.0029 - acc: 0.7125 - val_loss: 0.0035 - val_acc: 0.6333
Epoch 122/200
 - 0s - loss: 0.0028 - acc: 0.7125 - val_loss: 0.0034 - val_acc: 0.6500
Epoch 123/200
 - 0s - loss: 0.0028 - acc: 0.7125 - val_loss: 0.0034 - val_acc: 0.6500
Epoch 124/200
 - 0s - loss: 0.0027 - acc: 0.7125 - val_loss: 0.0033 - val_acc: 0.6500
Epoch 125/200
 - 0s - loss: 0.0027 - acc: 0.7125 - val_loss: 0.0032 - val_acc: 0.6500
Epoch 126/200
 - 0s - loss: 0.0026 - acc: 0.7125 - val_loss: 0.0032 - val_acc: 0.6500
Epoch 127/200
 - 0s - loss: 0.0026 - acc: 0.7125 - val_loss: 0.0031 - val_acc: 0.6500
Epoch 128/200
 - 0s - loss: 0.0025 - acc: 0.7125 - val_loss: 0.0031 - val_acc: 0.6500
Epoch 129/200
 - 0s - loss: 0.0025 - acc: 0.7125 - val_loss: 0.0030 - val_acc: 0.6500
Epoch 130/200
 - 0s - loss: 0.0024 - acc: 0.7125 - val_loss: 0.0029 - val_acc: 0.6500
Epoch 131/200
 - 0s - loss: 0.0024 - acc: 0.7125 - val_loss: 0.0029 - val_acc: 0.6500
Epoch 132/200
 - 0s - loss: 0.0024 - acc: 0.7125 - val_loss: 0.0028 - val_acc: 0.6500
Epoch 133/200
 - 0s - loss: 0.0023 - acc: 0.7125 - val_loss: 0.0028 - val_acc: 0.6500
Epoch 134/200
 - 0s - loss: 0.0023 - acc: 0.7125 - val_loss: 0.0027 - val_acc: 0.6500
Epoch 135/200
 - 0s - loss: 0.0022 - acc: 0.7125 - val_loss: 0.0027 - val_acc: 0.6500
Epoch 136/200
 - 0s - loss: 0.0022 - acc: 0.7125 - val_loss: 0.0026 - val_acc: 0.6500
Epoch 137/200
 - 0s - loss: 0.0022 - acc: 0.7125 - val_loss: 0.0026 - val_acc: 0.6500
Epoch 138/200
 - 0s - loss: 0.0021 - acc: 0.7125 - val_loss: 0.0026 - val_acc: 0.6500
```

```
Epoch 139/200
 - 0s - loss: 0.0021 - acc: 0.7125 - val_loss: 0.0025 - val_acc: 0.6500
Epoch 140/200
 - 0s - loss: 0.0020 - acc: 0.7125 - val_loss: 0.0025 - val_acc: 0.6500
Epoch 141/200
 - 0s - loss: 0.0020 - acc: 0.7125 - val_loss: 0.0024 - val_acc: 0.6500
Epoch 142/200
 - 0s - loss: 0.0020 - acc: 0.7125 - val_loss: 0.0024 - val_acc: 0.6500
Epoch 143/200
 - 0s - loss: 0.0019 - acc: 0.7125 - val_loss: 0.0024 - val_acc: 0.6500
Epoch 144/200
 - 0s - loss: 0.0019 - acc: 0.7125 - val_loss: 0.0023 - val_acc: 0.6500
Epoch 145/200
 - 0s - loss: 0.0019 - acc: 0.7125 - val_loss: 0.0023 - val_acc: 0.6500
Epoch 146/200
 - 0s - loss: 0.0019 - acc: 0.7125 - val_loss: 0.0022 - val_acc: 0.6500
Epoch 147/200
 - 0s - loss: 0.0018 - acc: 0.7125 - val_loss: 0.0022 - val_acc: 0.6667
Epoch 148/200
 - 0s - loss: 0.0018 - acc: 0.7125 - val_loss: 0.0022 - val_acc: 0.6667
Epoch 149/200
 - 0s - loss: 0.0018 - acc: 0.7125 - val_loss: 0.0021 - val_acc: 0.6667
Epoch 150/200
 - 0s - loss: 0.0017 - acc: 0.7125 - val_loss: 0.0021 - val_acc: 0.6667
Epoch 151/200
 - 0s - loss: 0.0017 - acc: 0.7167 - val_loss: 0.0021 - val_acc: 0.6667
Epoch 152/200
 - 0s - loss: 0.0017 - acc: 0.7125 - val_loss: 0.0020 - val_acc: 0.6667
Epoch 153/200
 - 0s - loss: 0.0017 - acc: 0.7167 - val_loss: 0.0020 - val_acc: 0.6667
Epoch 154/200
 - 0s - loss: 0.0016 - acc: 0.7167 - val_loss: 0.0020 - val_acc: 0.6667
Epoch 155/200
 - 0s - loss: 0.0016 - acc: 0.7125 - val_loss: 0.0020 - val_acc: 0.6667
Epoch 156/200
 - 0s - loss: 0.0016 - acc: 0.7125 - val_loss: 0.0019 - val_acc: 0.6667
Epoch 157/200
 - 0s - loss: 0.0016 - acc: 0.7125 - val_loss: 0.0019 - val_acc: 0.6667
Epoch 158/200
 - 0s - loss: 0.0015 - acc: 0.7125 - val_loss: 0.0019 - val_acc: 0.6667
Epoch 159/200
 - 0s - loss: 0.0015 - acc: 0.7125 - val_loss: 0.0018 - val_acc: 0.6667
Epoch 160/200
 - 0s - loss: 0.0015 - acc: 0.7125 - val_loss: 0.0018 - val_acc: 0.6667
Epoch 161/200
 - 0s - loss: 0.0015 - acc: 0.7125 - val_loss: 0.0018 - val_acc: 0.6667
Epoch 162/200
 - 0s - loss: 0.0015 - acc: 0.7125 - val_loss: 0.0018 - val_acc: 0.6667
Epoch 163/200
 - 0s - loss: 0.0014 - acc: 0.7125 - val_loss: 0.0017 - val_acc: 0.6667
Epoch 164/200
 - 0s - loss: 0.0014 - acc: 0.7125 - val_loss: 0.0017 - val_acc: 0.6667
Epoch 165/200
 - 0s - loss: 0.0014 - acc: 0.7125 - val_loss: 0.0017 - val_acc: 0.6667
Epoch 166/200
 - 0s - loss: 0.0014 - acc: 0.7125 - val_loss: 0.0017 - val_acc: 0.6667
Epoch 167/200
```

```
 - 0s - loss: 0.0014 - acc: 0.7125 - val_loss: 0.0017 - val_acc: 0.6667
Epoch 168/200
 - 0s - loss: 0.0013 - acc: 0.7167 - val_loss: 0.0016 - val_acc: 0.6667
Epoch 169/200
 - 0s - loss: 0.0013 - acc: 0.7167 - val_loss: 0.0016 - val_acc: 0.6667
Epoch 170/200
 - 0s - loss: 0.0013 - acc: 0.7167 - val_loss: 0.0016 - val_acc: 0.6667
Epoch 171/200
 - 0s - loss: 0.0013 - acc: 0.7167 - val_loss: 0.0016 - val_acc: 0.6667
Epoch 172/200
 - 0s - loss: 0.0013 - acc: 0.7208 - val_loss: 0.0015 - val_acc: 0.6667
Epoch 173/200
 - 0s - loss: 0.0013 - acc: 0.7208 - val_loss: 0.0015 - val_acc: 0.6667
Epoch 174/200
 - 0s - loss: 0.0012 - acc: 0.7208 - val_loss: 0.0015 - val_acc: 0.6667
Epoch 175/200
 - 0s - loss: 0.0012 - acc: 0.7208 - val_loss: 0.0015 - val_acc: 0.6667
Epoch 176/200
 - 0s - loss: 0.0012 - acc: 0.7208 - val_loss: 0.0015 - val_acc: 0.6667
Epoch 177/200
 - 0s - loss: 0.0012 - acc: 0.7208 - val_loss: 0.0014 - val_acc: 0.6667
Epoch 178/200
 - 0s - loss: 0.0012 - acc: 0.7208 - val_loss: 0.0014 - val_acc: 0.6667
Epoch 179/200
 - 0s - loss: 0.0012 - acc: 0.7208 - val_loss: 0.0014 - val_acc: 0.6667
Epoch 180/200
 - 0s - loss: 0.0011 - acc: 0.7208 - val_loss: 0.0014 - val_acc: 0.6667
Epoch 181/200
 - 0s - loss: 0.0011 - acc: 0.7208 - val_loss: 0.0014 - val_acc: 0.6667
Epoch 182/200
 - 0s - loss: 0.0011 - acc: 0.7208 - val_loss: 0.0014 - val_acc: 0.6667
Epoch 183/200
 - 0s - loss: 0.0011 - acc: 0.7250 - val_loss: 0.0013 - val_acc: 0.6667
Epoch 184/200
 - 0s - loss: 0.0011 - acc: 0.7250 - val_loss: 0.0013 - val_acc: 0.6667
Epoch 185/200
 - 0s - loss: 0.0011 - acc: 0.7250 - val_loss: 0.0013 - val_acc: 0.6667
Epoch 186/200
 - 0s - loss: 0.0011 - acc: 0.7250 - val_loss: 0.0013 - val_acc: 0.6667
Epoch 187/200
 - 0s - loss: 0.0011 - acc: 0.7292 - val_loss: 0.0013 - val_acc: 0.6667
Epoch 188/200
 - 0s - loss: 0.0010 - acc: 0.7292 - val_loss: 0.0013 - val_acc: 0.6667
Epoch 189/200
 - 0s - loss: 0.0010 - acc: 0.7292 - val_loss: 0.0013 - val_acc: 0.6667
Epoch 190/200
 - 0s - loss: 0.0010 - acc: 0.7292 - val_loss: 0.0012 - val_acc: 0.6667
Epoch 191/200
 - 0s - loss: 0.0010 - acc: 0.7292 - val_loss: 0.0012 - val_acc: 0.6667
Epoch 192/200
 - 0s - loss: 9.9607e-04 - acc: 0.7292 - val_loss: 0.0012 - val_acc: 0.66
67

Epoch 193/200
 - 0s - loss: 9.8470e-04 - acc: 0.7292 - val_loss: 0.0012 - val_acc: 0.
6667
Epoch 194/200
```

```
 - 0s - loss: 9.7352e-04 - acc: 0.7292 - val_loss: 0.0012 - val_acc: 0.
6667
Epoch 195/200
 - 0s - loss: 9.6251e-04 - acc: 0.7292 - val_loss: 0.0012 - val_acc: 0.
6667
Epoch 196/200
 - 0s - loss: 9.5171e-04 - acc: 0.7292 - val_loss: 0.0012 - val_acc: 0.
6667
Epoch 197/200
 - 0s - loss: 9.4091e-04 - acc: 0.7333 - val_loss: 0.0011 - val_acc: 0.
6667
Epoch 198/200
 - 0s - loss: 9.3025e-04 - acc: 0.7333 - val_loss: 0.0011 - val_acc: 0.
6667
Epoch 199/200
 - 0s - loss: 9.1919e-04 - acc: 0.7333 - val_loss: 0.0011 - val_acc: 0.
6667
Epoch 200/200
 - 0s - loss: 9.0763e-04 - acc: 0.7333 - val_loss: 0.0011 - val_acc: 0.
6667
```

In [85]:

```
network3 = models.Sequential()

network3.add(layers.Dense(128, activation='relu', input_dim=6))
network3.add(layers.Dropout(0.3))
network3.add(layers.Dense(16, activation='relu'))
network3.add(layers.Dropout(0.1))
network3.add(layers.Dense(2, activation='softmax'))
network3.summary()

# included the early stopping which monitors the validation loss
early_stop = callbacks.EarlyStopping(monitor='val_loss', patience=5)
network3.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_34 (Dense)             (None, 128)               896       
_________________________________________________________________
dropout_9 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_35 (Dense)             (None, 16)                2064      
_________________________________________________________________
dropout_10 (Dropout)         (None, 16)                0         
_________________________________________________________________
dense_36 (Dense)             (None, 2)                 34        
=================================================================
Total params: 2,994
Trainable params: 2,994
Non-trainable params: 0
_________________________________________________________________
```

TO DO: Define and train a network, then plot the accuracy of the training, validation, and

testing

1. Use a validation set
2. Propose and train a network
3. Print the history of the training
4. Evaluate with a test set