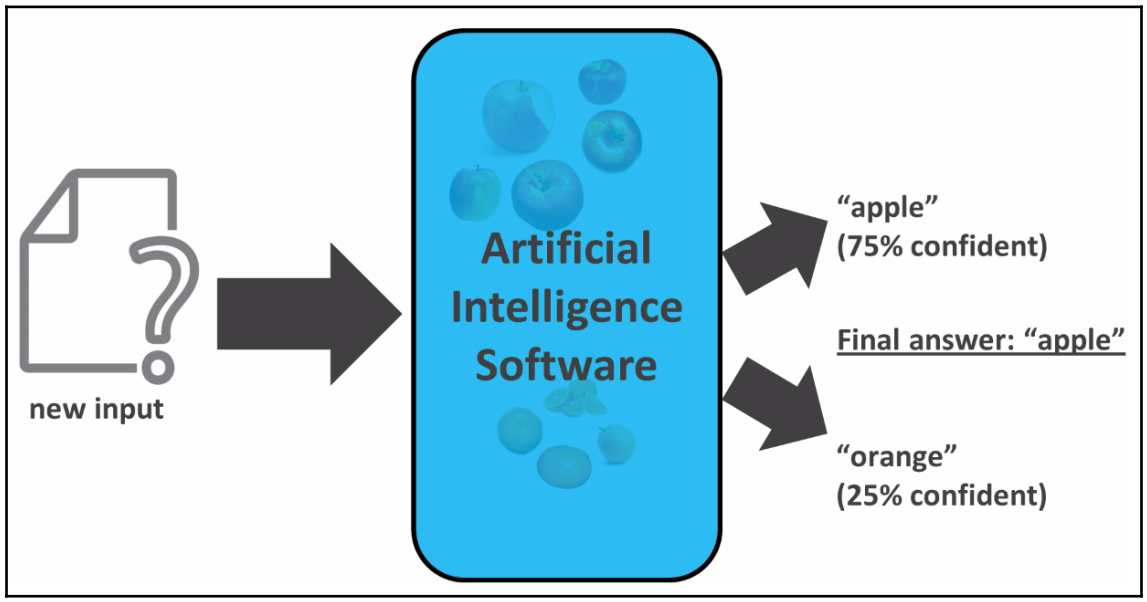
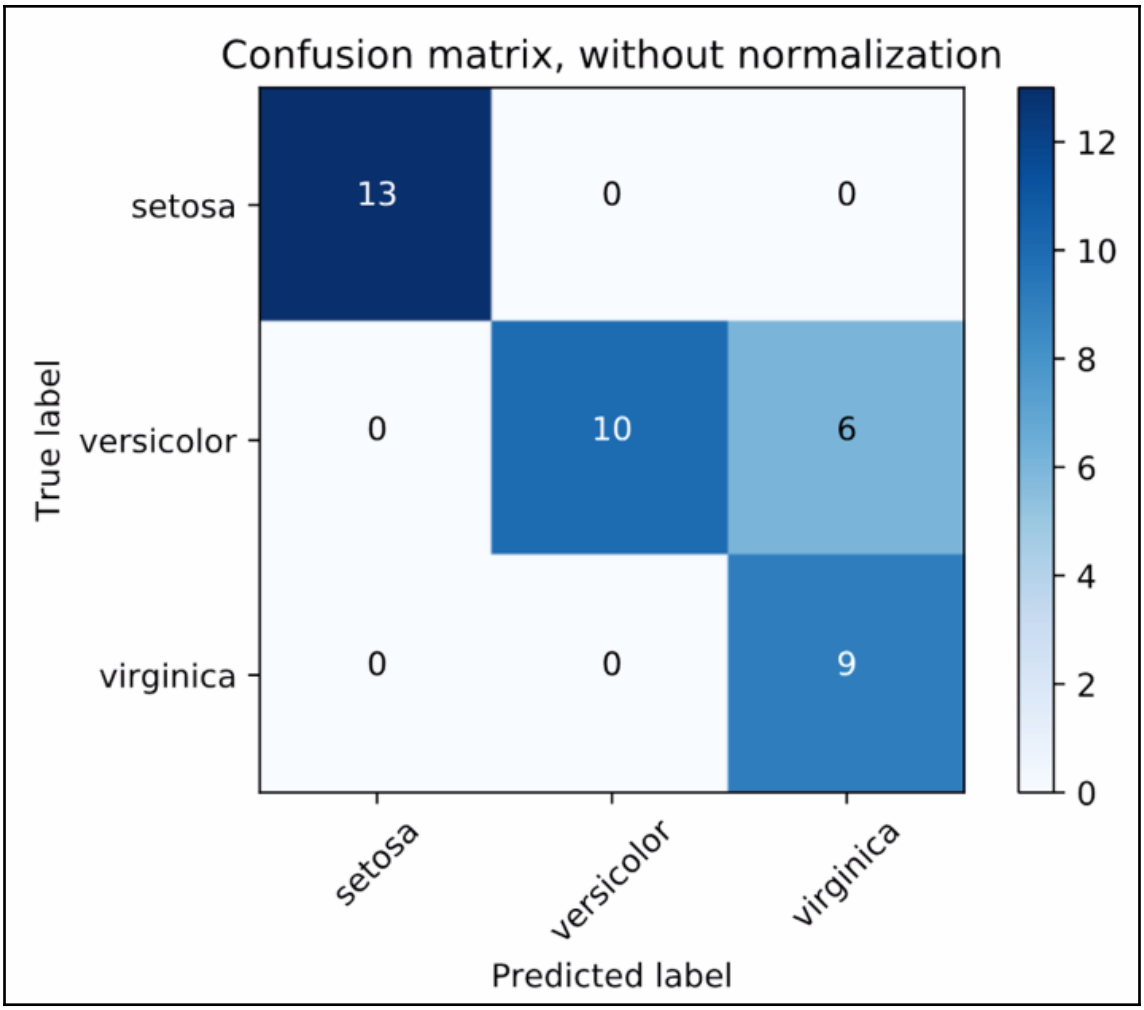
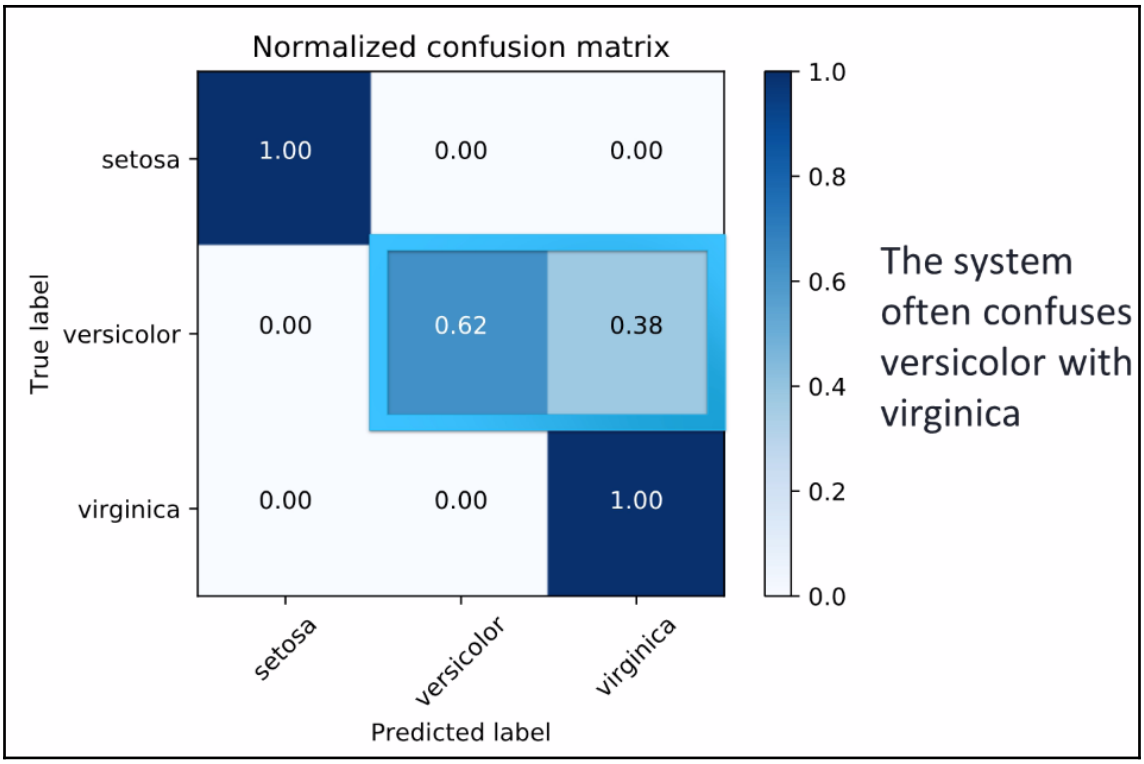
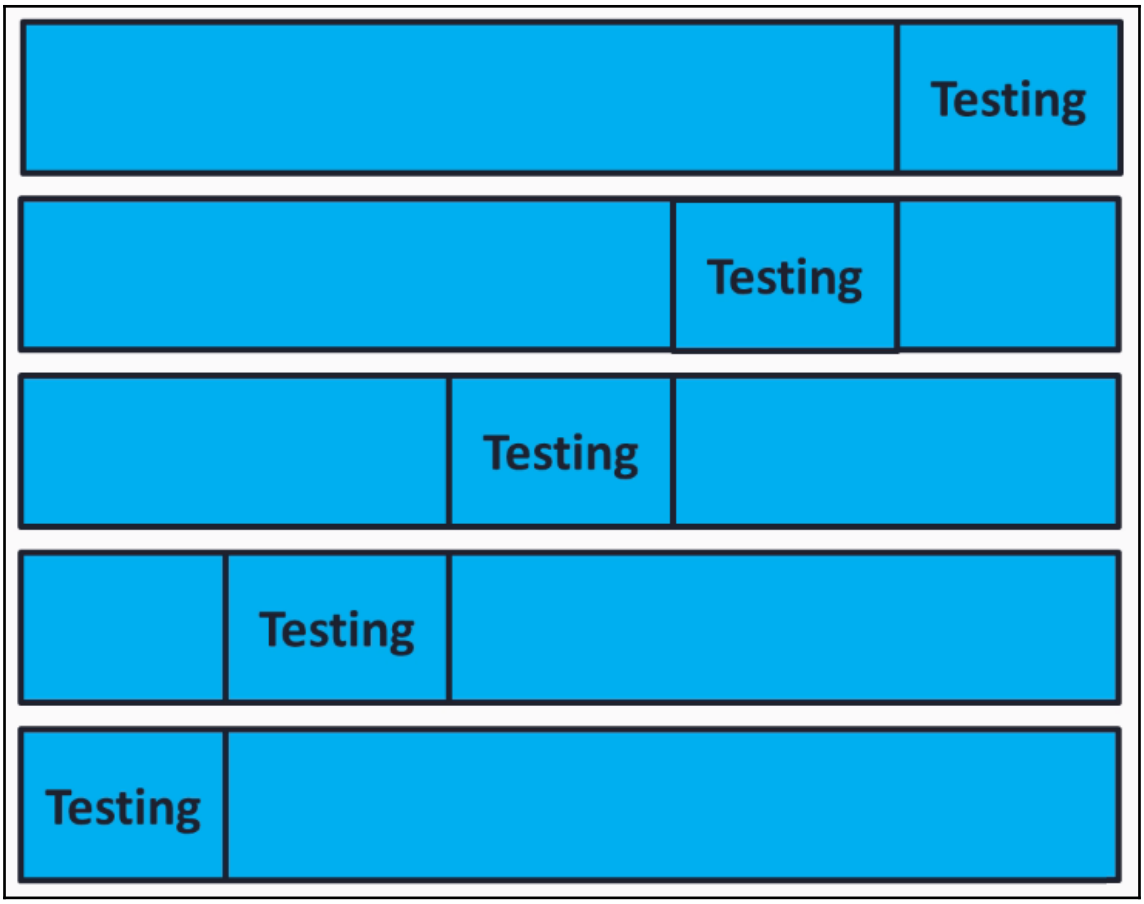
Chapter 1: Building Your Own Prediction Models

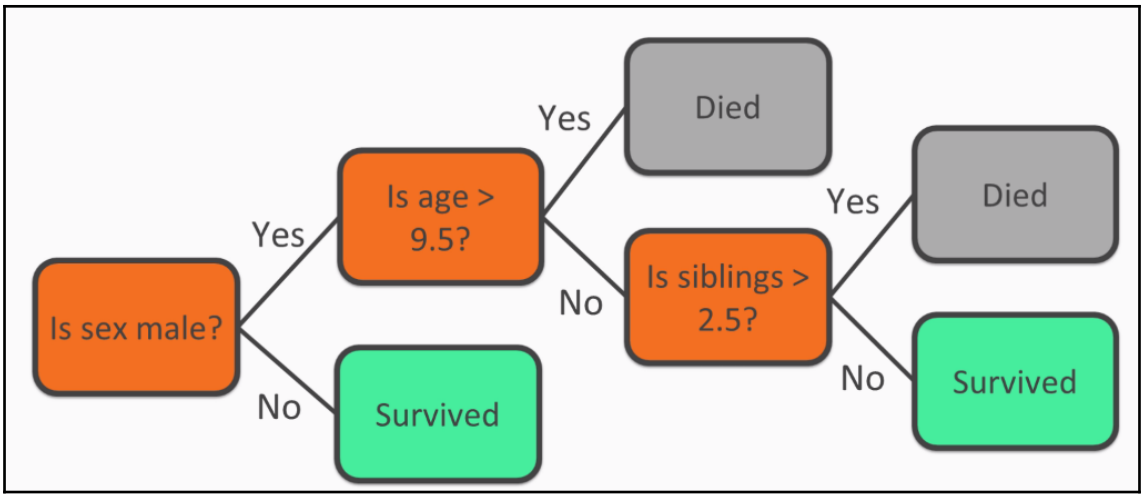


	Predicted "apple"	Predicted "orange"
True "apple"	20	5
True "orange"	3	22

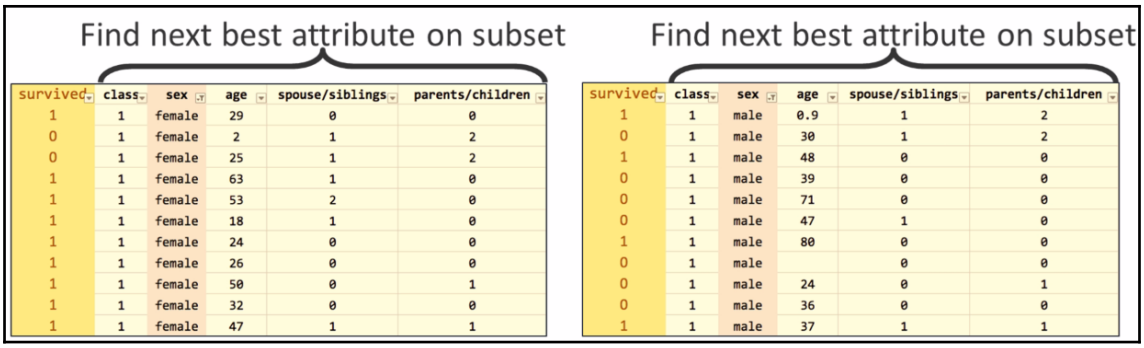

Confusion matrix, without normalization
setosa
versicolor
virginica
True label
13
0
0
0
10
6
0
0
9
setosa
versicolor
virginica
Predicted label
12
10
8
6
4
2
0


Normalized confusion matrix
setosa
versicolor
virginica
True label
1.00
0.00
0.00
0.00
0.62
0.38
0.00
0.00
1.00
setosa
versicolor
virginica
Predicted label
1.0
0.8
0.6
0.4
0.2
0.0
The system often confuses versicolor with virginica


Testing
Testing
Testing
Testing
Testing

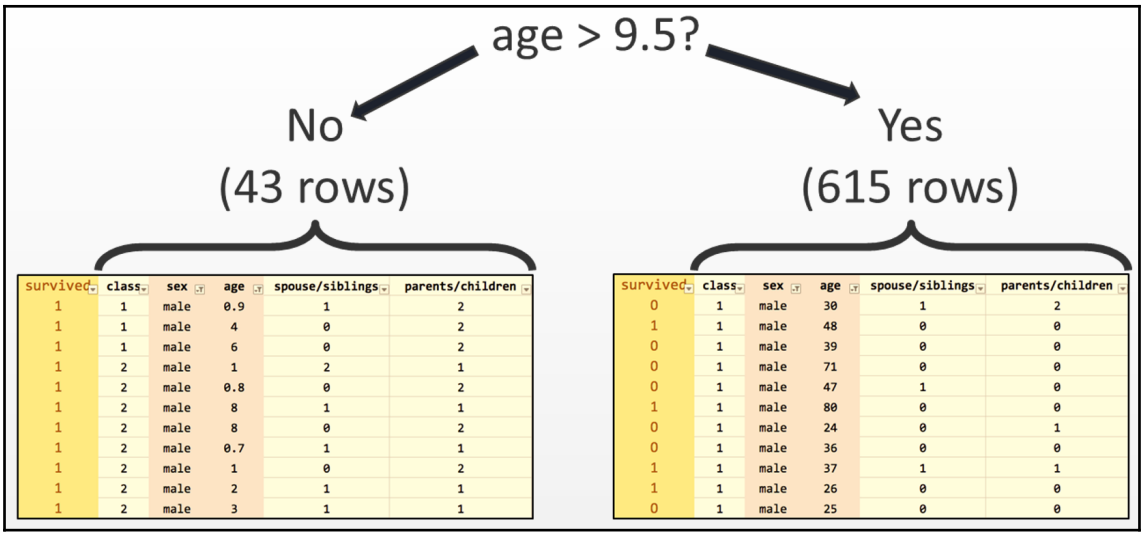


survived	class	sex	age	spouse/siblings	parents/children
1	1	female	29	0	0
1	1	male	0.9	1	2
0	1	female	2	1	2
0	1	male	30	1	2
0	1	female	25	1	2
1	1	male	48	0	0
1	1	female	63	1	0
0	1	male	39	0	0
1	1	female	53	2	0
0	1	male	71	0	0
0	1	male	47	1	0
1	1	female	18	1	0
1	1	female	24	0	0
1	1	female	26	0	0
1	1	male	80	0	0



survived	class	sex	age	spouse/siblings	parents/children
1	1	female	29	0	0
0	1	female	2	1	2
0	1	female	25	1	2
1	1	female	63	1	0
1	1	female	53	2	0
1	1	female	18	1	0
1	1	female	24	0	0
1	1	female	26	0	0
1	1	female	50	0	1
1	1	female	32	0	0
1	1	female	47	1	1

survived	class	sex	age	spouse/siblings	parents/children
1	1	male	0.9	1	2
0	1	male	30	1	2
1	1	male	48	0	0
0	1	male	39	0	0
0	1	male	71	0	0
0	1	male	47	1	0
1	1	male	80	0	0
0	1	male		0	0
0	1	male	24	0	1
0	1	male	36	0	0
1	1	male	37	1	1



survived	class	sex	age	spouse/siblings	parents/children
1	1	male	0.9	1	2
1	1	male	4	0	2
1	1	male	6	0	2
1	2	male	1	2	1
1	2	male	0.8	0	2
1	2	male	8	1	1
1	2	male	8	0	2
1	2	male	0.7	1	1
1	2	male	1	0	2
1	2	male	2	1	1
1	2	male	3	1	1

survived	class	sex	age	spouse/siblings	parents/children
0	1	male	30	1	2
1	1	male	48	0	0
0	1	male	39	0	0
0	1	male	71	0	0
0	1	male	47	1	0
1	1	male	80	0	0
0	1	male	24	0	1
0	1	male	36	0	0
1	1	male	37	1	1
1	1	male	26	0	0
0	1	male	25	0	0

```
from sklearn import tree
t = tree.DecisionTreeClassifier(criterion="entropy")

t = t.fit(train_attributes, train_labels)          Build the decision tree

t.score(test_attributes, test_labels)              Build the decision tree

t.predict(example_attributes)                      Predict a new example

cross_val_score(t, all_attributes, all_labels)     Average scores with
                                                   cross-validation
```

1 school - student's school (binary: 'GP' - Gabriel Pereira or 'MS' - Mousinho da Silveira)

2 sex - student's sex (binary: 'F' - female or 'M' - male)

3 age - student's age (numeric: from 15 to 22)

4 address - student's home address type (binary: 'U' - urban or 'R' - rural)

5 famsize - family size (binary: 'LE3' - less or equal to 3 or 'GT3' - greater than 3)

6 Pstatus - parent's cohabitation status (binary: 'T' - living together or 'A' - apart)

7 Medu - mother's education (numeric: 0 - none, 1 - primary education (4th grade), 2 - 5th to 9th grade, ...

8 Fedu - father's education (numeric: 0 - none, 1 - primary education (4th grade), 2 - 5th to 9th grade, ...

9 Mjob - mother's job (nominal: 'teacher', 'health' care related, civil 'services' (e.g. administrative or police), 'at_home' or 'other')

10 Fjob - father's job (nominal: 'teacher', 'health' care related, civil 'services' (e.g. administrative or police), 'at_home' or 'other')

11 reason - reason to choose this school (nominal: close to 'home', school 'reputation', 'course' preference or 'other')

12 guardian - student's guardian (nominal: 'mother', 'father' or 'other')

13 traveltime - home to school travel time (numeric: 1 - <15 min., 2 - 15 to 30 min., 3 - 30 min. to 1 hour, or 4 - >1 hour)

14 studytime - weekly study time (numeric: 1 - <2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 - >10 hours)

15 failures - number of past class failures (numeric: n if 1<=n<3, else 4)

16 schoolsup - extra educational support (binary: yes or no)

17 famsup - family educational support (binary: yes or no)

18 paid - extra paid classes within the course subject (Math or Portuguese) (binary: yes or no)

19 activities - extra-curricular activities (binary: yes or no)

20 nursery - attended nursery school (binary: yes or no)

21 higher - wants to take higher education (binary: yes or no)

22 internet - Internet access at home (binary: yes or no)

23 romantic - with a romantic relationship (binary: yes or no)

24 famrel - quality of family relationships (numeric: from 1 - very bad to 5 - excellent)

25 freetime - free time after school (numeric: from 1 - very low to 5 - very high)

26 goout - going out with friends (numeric: from 1 - very low to 5 - very high)

27 Dalc - workday alcohol consumption (numeric: from 1 - very low to 5 - very high)

28 Walc - weekend alcohol consumption (numeric: from 1 - very low to 5 - very high)

29 health - current health status (numeric: from 1 - very bad to 5 - very good)

30 absences - number of school absences (numeric: from 0 to 93)

```
In [1]: # load dataset (student Portuguese scores)
        import pandas as pd
        d = pd.read_csv('student-por.csv', sep=';')
        len(d)

Out[1]: 649
```

```
In [2]: # generate binary label (pass/fail) based on G1+G2+G3 (test grades, each 0-20 pts); threshold for passing is sum>=30
        d['pass'] = d.apply(lambda row: 1 if (row['G1']+row['G2']+row['G3']) >= 35 else 0, axis=1)
        d = d.drop(['G1', 'G2', 'G3'], axis=1)
        d.head()
```

Out[2]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	internet	romantic	famrel	freetime	goout	Dalc	Walc	health	absences
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	no	no	4	3	4	1	1	3	4
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	yes	no	5	3	3	1	1	3	2
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	yes	no	4	3	2	2	3	3	6
3	GP	F	15	U	GT3	T	4	2	health	services	...	yes	yes	3	2	2	1	1	5	0
4	GP	F	16	U	GT3	T	3	3	other	other	...	no	no	4	3	2	1	2	5	0

5 rows × 31 columns

In [3]:
```
# use one-hot encoding on categorical columns
d = pd.get_dummies(d, columns=['sex', 'school', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob',
                               'reason', 'guardian', 'schoolsup', 'famsup', 'paid', 'activities',
                               'nursery', 'higher', 'internet', 'romantic'])
d.head()
```

Out[3]:

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	...	activities_no	activities_yes	nursery_no
0	18	4	4	2	2	0	4	3	4	1	...	1	0	0
1	17	1	1	1	2	0	5	3	3	1	...	1	0	1
2	15	1	1	1	2	0	4	3	2	2	...	1	0	0
3	15	4	2	1	3	0	3	2	2	1	...	0	1	0
4	16	3	3	1	2	0	4	3	2	1	...	1	0	0

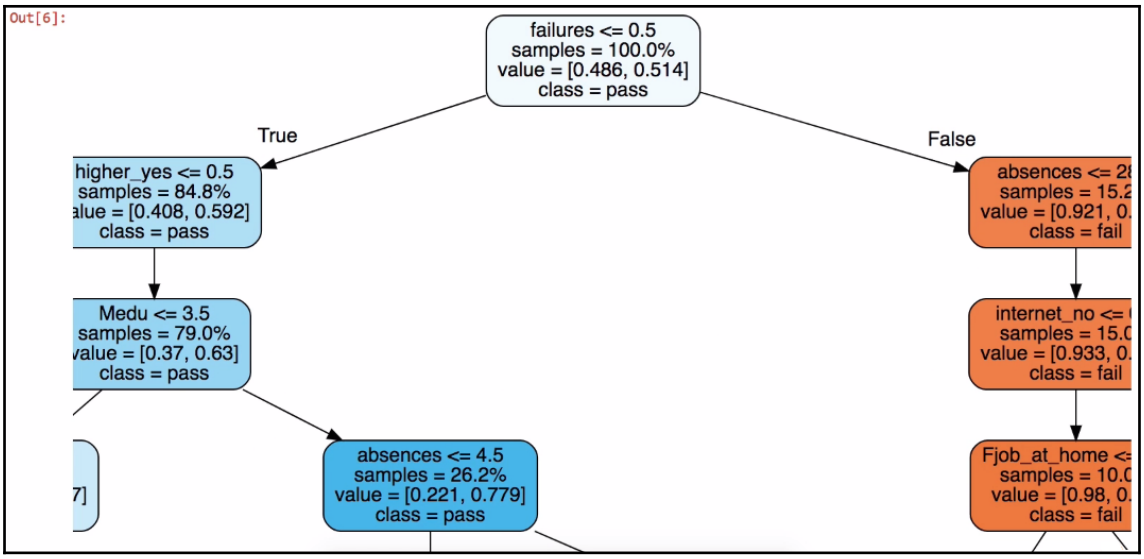
5 rows × 57 columns

In [4]:
```
# shuffle rows
d = d.sample(frac=1)
# split training and testing data
d_train = d[:500]
d_test = d[500:]

d_train_att = d_train.drop(['pass'], axis=1)
d_train_pass = d_train['pass']

d_test_att = d_test.drop(['pass'], axis=1)
d_test_pass = d_test['pass']

d_att = d.drop(['pass'], axis=1)
d_pass = d['pass']

# number of passing students in whole dataset:
import numpy as np
print("Passing: %d out of %d (%.2f%%)" % (np.sum(d_pass), len(d_pass), 100*float(np.sum(d_pass)) / len(d_pass)))
```

```
Passing: 328 out of 649 (50.54%)
```

In [5]:
```
# fit a decision tree
from sklearn import tree
t = tree.DecisionTreeClassifier(criterion="entropy", max_depth=5)
t = t.fit(d_train_att, d_train_pass)
```

Out[6]:



In [7]:
```
# save tree
tree.export_graphviz(t, out_file="student-performance.dot", label="all", impurity=False, proportion=True,
                     feature_names=list(d_train_att), class_names=["fail", "pass"],
                     filled=True, rounded=True)
```

In [8]:
```
t.score(d_test_att, d_test_pass)
```

Out[8]: 0.59731543624161076

In [9]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(t, d_att, d_pass, cv=5)
# show average score and +/- two standard deviations away (covering 95% of scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.67 (+/- 0.06)
```

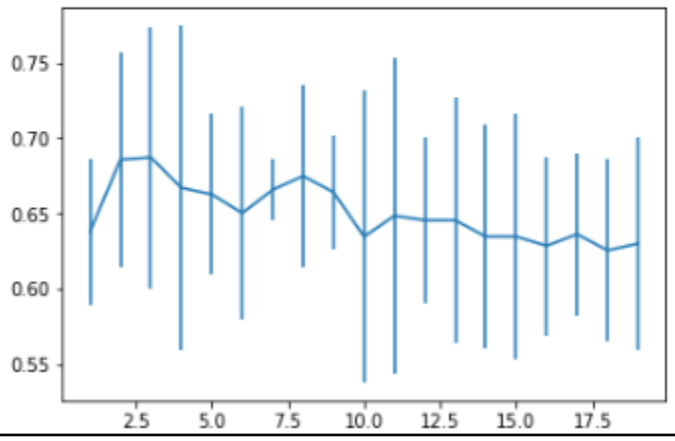
In [10]:
```
for max_depth in range(1, 20):
    t = tree.DecisionTreeClassifier(criterion="entropy", max_depth=max_depth)
    scores = cross_val_score(t, d_att, d_pass, cv=5)
    print("Max depth: %d, Accuracy: %0.2f (+/- %0.2f)" % (max_depth, scores.mean(), scores.std() * 2))
```

```
Max depth: 1, Accuracy: 0.64 (+/- 0.05)
Max depth: 2, Accuracy: 0.69 (+/- 0.08)
Max depth: 3, Accuracy: 0.69 (+/- 0.09)
Max depth: 4, Accuracy: 0.66 (+/- 0.10)
Max depth: 5, Accuracy: 0.67 (+/- 0.06)
Max depth: 6, Accuracy: 0.64 (+/- 0.08)
Max depth: 7, Accuracy: 0.67 (+/- 0.02)
Max depth: 8, Accuracy: 0.67 (+/- 0.07)
Max depth: 9, Accuracy: 0.67 (+/- 0.06)
Max depth: 10, Accuracy: 0.63 (+/- 0.12)
Max depth: 11, Accuracy: 0.65 (+/- 0.07)
Max depth: 12, Accuracy: 0.63 (+/- 0.07)
Max depth: 13, Accuracy: 0.63 (+/- 0.07)
Max depth: 14, Accuracy: 0.63 (+/- 0.08)
Max depth: 15, Accuracy: 0.64 (+/- 0.06)
Max depth: 16, Accuracy: 0.62 (+/- 0.05)
Max depth: 17, Accuracy: 0.64 (+/- 0.09)
Max depth: 18, Accuracy: 0.63 (+/- 0.08)
Max depth: 19, Accuracy: 0.63 (+/- 0.06)
```

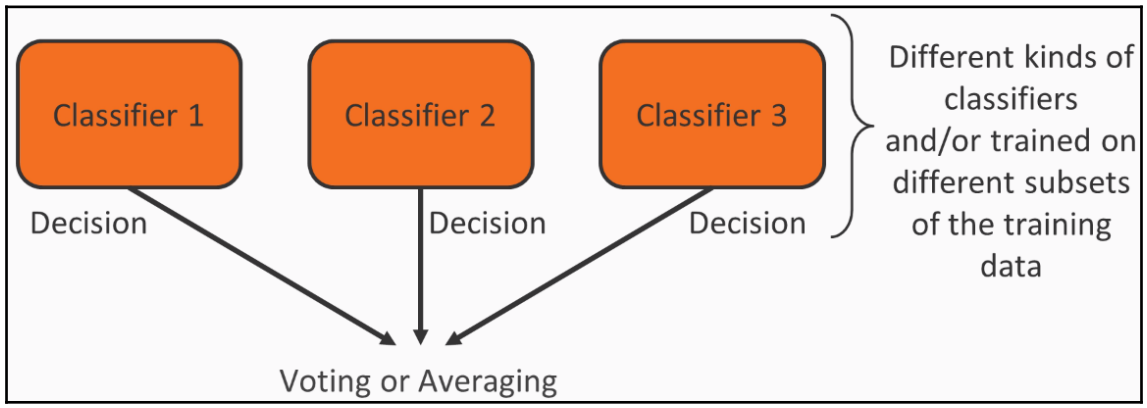
In [11]:
```
depth_acc = np.empty((19,3), float)
i = 0
for max_depth in range(1, 20):
    t = tree.DecisionTreeClassifier(criterion="entropy", max_depth=max_depth)
    scores = cross_val_score(t, d_att, d_pass, cv=5)
    depth_acc[i,0] = max_depth
    depth_acc[i,1] = scores.mean()
    depth_acc[i,2] = scores.std() * 2
    i += 1

depth_acc
```

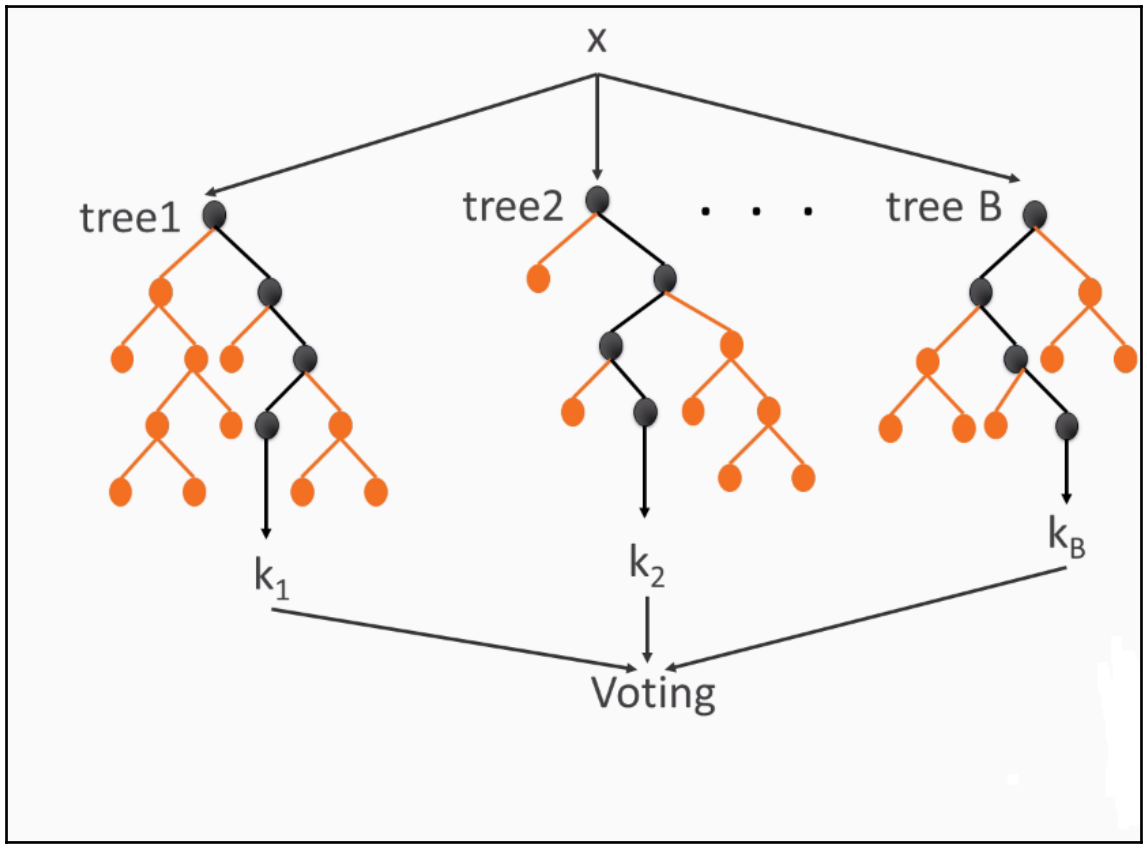
Out[11]:
```
array([[ 1.        ,   0.63790456,   0.04848398],
       [ 2.        ,   0.68559869,   0.07148267],
       [ 3.        ,   0.68710174,   0.0865951 ],
       [ 4.        ,   0.6669467 ,   0.10726248],
       [ 5.        ,   0.66261518,   0.05307124],
       [ 6.        ,   0.65018859,   0.07040891],
       [ 7.        ,   0.66564494,   0.02029519],
       [ 8.        ,   0.67474598,   0.05984916],
       [ 9.        ,   0.6640118 ,   0.03746891],
       [10.        ,   0.6346137 ,   0.09657669],
       [11.        ,   0.6484015 ,   0.10475147],
       [12.        ,   0.64545485,   0.05529647],
       [13.        ,   0.64544256,   0.08167465],
       [14.        ,   0.6346614 ,   0.07458128],
       [15.        ,   0.63463773,   0.08162646],
       [16.        ,   0.62853141,   0.05926906],
       [17.        ,   0.63622335,   0.05390067],
       [18.        ,   0.62548936,   0.06050112],
       [19.        ,   0.63004547,   0.07022296]])
```

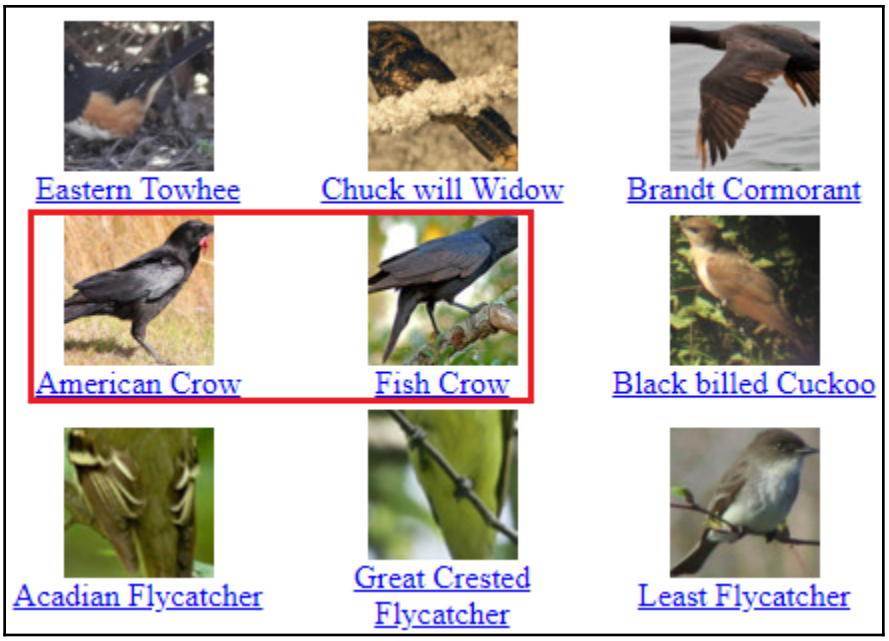
In [12]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.errorbar(depth_acc[:,0], depth_acc[:,1], yerr=depth_acc[:,2])
plt.show()
```

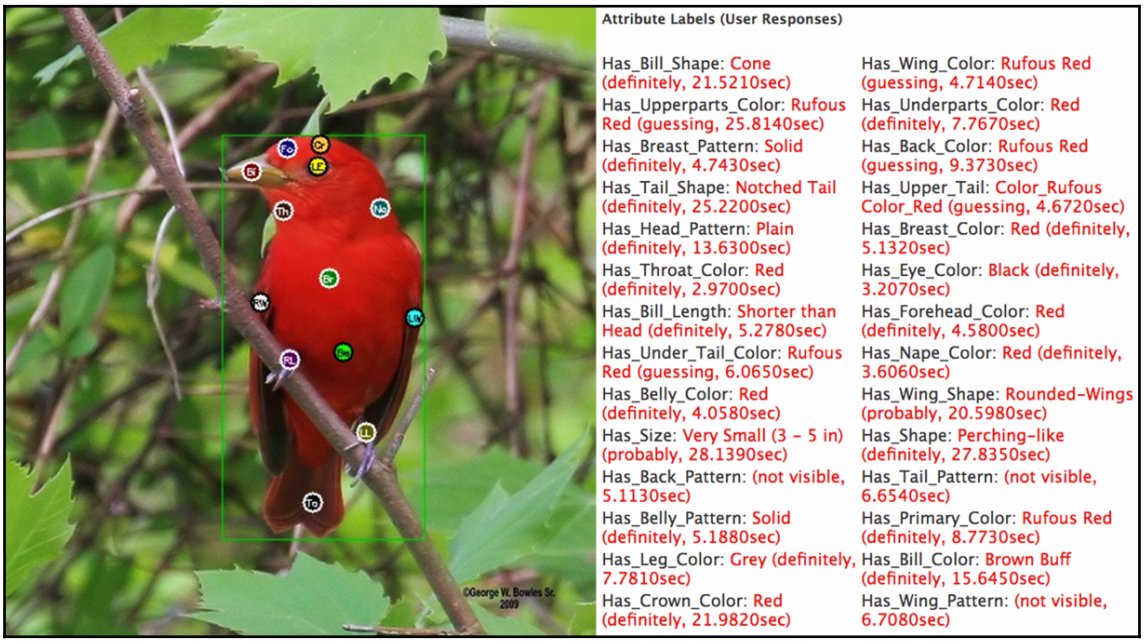


Chapter 2: Prediction with Random Forests






Eastern Towhee
Chuck will Widow
Brandt Cormorant
American Crow
Fish Crow
Black billed Cuckoo
Acadian Flycatcher
Great Crested Flycatcher
Least Flycatcher


Attribute Labels (User Responses)
Has_Bill_Shape: Cone (definitely, 21.5210sec)
Has_Wing_Color: Rufous Red (guessing, 4.7140sec)
Has_Upperparts_Color: Rufous Red (guessing, 25.8140sec)
Has_Underparts_Color: Red (definitely, 7.7670sec)
Has_Breast_Pattern: Solid (definitely, 4.7430sec)
Has_Back_Color: Rufous Red (guessing, 9.3730sec)
Has_Tail_Shape: Notched Tail (definitely, 25.2200sec)
Has_Upper_Tail: Color_Rufous Color_Red (guessing, 4.6720sec)
Has_Head_Pattern: Plain (definitely, 13.6300sec)
Has_Breast_Color: Red (definitely, 5.1320sec)
Has_Throat_Color: Red (definitely, 2.9700sec)
Has_Eye_Color: Black (definitely, 3.2070sec)
Has_Bill_Length: Shorter than Head (definitely, 5.2780sec)
Has_Forehead_Color: Red (definitely, 4.5800sec)
Has_Under_Tail_Color: Rufous Red (guessing, 6.0650sec)
Has_Nape_Color: Red (definitely, 3.6060sec)
Has_Belly_Color: Red (definitely, 4.0580sec)
Has_Wing_Shape: Rounded-Wings (probably, 20.5980sec)
Has_Size: Very Small (3 - 5 in) (probably, 28.1390sec)
Has_Shape: Perching-like (definitely, 27.8350sec)
Has_Back_Pattern: (not visible, 5.1130sec)
Has_Tail_Pattern: (not visible, 6.6540sec)
Has_Belly_Pattern: Solid (definitely, 5.1880sec)
Has_Primary_Color: Rufous Red (definitely, 8.7730sec)
Has_Leg_Color: Grey (definitely, 7.7810sec)
Has_Bill_Color: Brown Buff (definitely, 15.6450sec)
Has_Crown_Color: Red (definitely, 21.9820sec)
Has_Wing_Pattern: (not visible, 6.7080sec)
©George W. Bowles Sr. 2009

Class ids/names (classes.txt)	Image ids/file names (images.txt)	Image ids/class ids (image_class_labels.txt)
1 001.Black_footed_Albatross	1 001.Black_footed_Albatross/Black_Footed_Albatross_0046_18.jpg	1 1
2 002.Laysan_Albatross	2 001.Black_footed_Albatross/Black_Footed_Albatross_0009_34.jpg	2 1
3 003.Sooty_Albatross	3 001.Black_footed_Albatross/Black_Footed_Albatross_0002_55.jpg	3 1
4 004.Groove_billed_Ani	4 001.Black_footed_Albatross/Black_Footed_Albatross_0074_59.jpg	4 1
5 005.Crested_Auklet	5 001.Black_footed_Albatross/Black_Footed_Albatross_0014_89.jpg	5 1
6 006.Least_Auklet	6 001.Black_footed_Albatross/Black_Footed_Albatross_0085_92.jpg	6 1
7 007.Parakeet_Auklet	7 001.Black_footed_Albatross/Black_Footed_Albatross_0031_100.jpg	7 1
8 008.Rhinoceros_Auklet	8 001.Black_footed_Albatross/Black_Footed_Albatross_0051_796103.jpg	8 1
9 009.Brewer_Blackbird	9 001.Black_footed_Albatross/Black_Footed_Albatross_0010_796097.jpg	9 1
10 010.Red_winged_Blackbird	10 001.Black_footed_Albatross/Black_Footed_Albatross_0025_796057.jpg	10 1

Attribute ids/names (attributes.txt)

```
1 has_bill_shape::curved_(up_or_down)
2 has_bill_shape::dagger
3 has_bill_shape::hooked
4 has_bill_shape::needle
5 has_bill_shape::hooked_seabird
6 has_bill_shape::spatulate
7 has_bill_shape::all-purpose
8 has_bill_shape::cone
9 has_bill_shape::specialized
10 has_wing_color::blue
11 has_wing_color::brown
12 has_wing_color::iridescent
13 has_wing_color::purple
14 has_wing_color::rufous
```

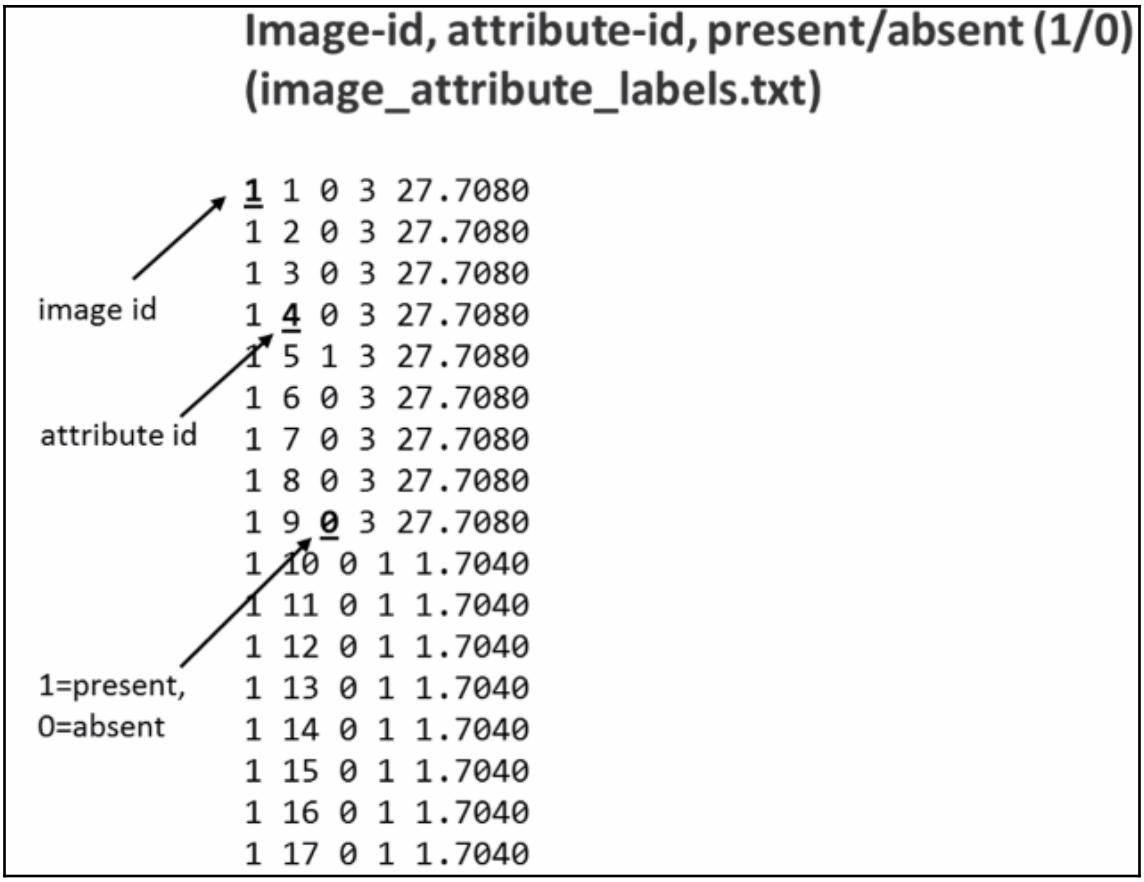


Image-id, attribute-id, present/absent (1/0)
(image_attribute_labels.txt)
1 1 0 3 27.7080
1 2 0 3 27.7080
1 3 0 3 27.7080
1 4 0 3 27.7080
1 5 1 3 27.7080
1 6 0 3 27.7080
1 7 0 3 27.7080
1 8 0 3 27.7080
1 9 0 3 27.7080
1 10 0 1 1.7040
1 11 0 1 1.7040
1 12 0 1 1.7040
1 13 0 1 1.7040
1 14 0 1 1.7040
1 15 0 1 1.7040
1 16 0 1 1.7040
1 17 0 1 1.7040
image id
attribute id
1=present,
0=absent

In [44]:
```
import pandas as pd

# some lines have too many fields (?), so skip bad lines
imgatt = pd.read_csv("data/CUB_200_2011/attributes/image_attribute_labels.txt",
                     sep='\s+', header=None, error_bad_lines=False, warn_bad_lines=False,
                     usecols=[0,1,2], names=['imgid', 'attid', 'present'])

# description from dataset README:
#
# The set of attribute labels as perceived by MTurkers for each image
# is contained in the file attributes/image_attribute_labels.txt, with
# each line corresponding to one image/attribute/worker triplet:
#
# <image_id> <attribute_id> <is_present> <certainty_id> <time>
#
# where <image_id>, <attribute_id>, <certainty_id> correspond to the IDs
# in images.txt, attributes/attributes.txt, and attributes/certainties.txt
# respectively.  <is_present> is 0 or 1 (1 denotes that the attribute is
# present).  <time> denotes the time spent by the MTurker in seconds.
```

In [45]:
```
imgatt.head()
```

Out[45]:

	imgid	attid	present
0	1	1	0
1	1	2	0
2	1	3	0
3	1	4	0
4	1	5	1

In [46]:
```
imgatt.shape
```

Out[46]: (3677856, 3)

In [47]:
```
# need to reorganize imgatt to have one row per imgid, and 312 columns (one column per attribute),
# with 1/0 in each cell representing if that imgid has that attribute or not

imgatt2 = imgatt.pivot(index='imgid', columns='attid', values='present')
```

In [48]:
```
imgatt2.head()
```

Out[48]:

attid	1	2	3	4	5	6	7	8	9	10	...	303	304	305	306	307	308	309	310	311	312
imgid																					
1	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	1	0	0	1	0
4	0	0	0	0	1	0	0	0	0	0	...	0	0	0	1	0	0	1	0	0	0
5	0	0	0	0	1	0	0	0	0	0	...	0	0	1	0	0	0	0	0	0	0

5 rows × 312 columns

In [49]:
```
imgatt2.shape
```

Out[49]: (11788, 312)

In [50]:
```
# now we need to load the image true classes

imglabels = pd.read_csv("data/CUB_200_2011/image_class_labels.txt",
                        sep=' ', header=None, names=['imgid', 'label'])

imglabels = imglabels.set_index('imgid')

# description from dataset README:
#
# The ground truth class labels (bird species labels) for each image are contained
# in the file image_class_labels.txt, with each line corresponding to one image:
#
# <image_id> <class_id>
#
# where <image_id> and <class_id> correspond to the IDs in images.txt and classes.txt,
# respectively.
```

```
In [51]: imglabels.head()
```

Out[51]:

imgid	label
1	1
2	1
3	1
4	1
5	1

```
In [52]: imglabels.shape
Out[52]: (11788, 1)
```

```
In [53]: # now we need to attach the labels to the attribute data set,
         # and shuffle; then we'll separate a test set from a training set

         df = imgatt2.join(imglabels)
         df = df.sample(frac=1)
```

```
In [54]: df_att = df.iloc[:, :312]
         df_label = df.iloc[:, 312:]
```

```
In [55]: df_att.head()
```

Out[55]:

imgid	1	2	3	4	5	6	7	8	9	10	...	303	304	305	306	307	308	309	310	311	312
527	0	0	0	0	0	0	1	0	0	0	...	0	0	1	0	0	0	0	0	0	1
1532	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0	0
9137	0	0	0	0	0	0	1	0	0	0	...	0	0	1	0	0	0	0	0	0	1
487	0	1	0	0	0	0	0	0	0	0	...	0	0	1	0	0	0	0	0	0	1
2444	0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	1

5 rows × 312 columns

```
In [57]: df_train_att = df_att[:8000]
         df_train_label = df_label[:8000]
         df_test_att = df_att[8000:]
         df_test_label = df_label[8000:]

         df_train_label = df_train_label['label']
         df_test_label = df_test_label['label']
```

```
In [58]: from sklearn.ensemble import RandomForestClassifier
         clf = RandomForestClassifier(max_features=50, random_state=0, n_estimators=100)
```

```
In [59]: clf.fit(df_train_att, df_train_label)

Out[59]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features=50, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
                     oob_score=False, random_state=0, verbose=0, warm_start=False)
```

```
In [60]: print(clf.predict(df_train_att.head()))
         [ 10  28 156  10  43]
```

```
In [61]: clf.score(df_test_att, df_test_label)

Out[61]: 0.44297782470960928
```

```
In [62]: from sklearn.metrics import confusion_matrix
         pred_labels = clf.predict(df_test_att)
         cm = confusion_matrix(df_test_label, pred_labels)
```

```
In [63]: cm

Out[63]: array([[ 5,  1,  6, ...,  0,  1,  0],
                [ 0, 12,  0, ...,  0,  0,  0],
                [ 0,  0,  8, ...,  0,  0,  0],
                ...,
                [ 0,  0,  0, ...,  6,  0,  0],
                [ 0,  0,  0, ...,  0, 11,  0],
                [ 0,  0,  0, ...,  0,  0, 13]])
```

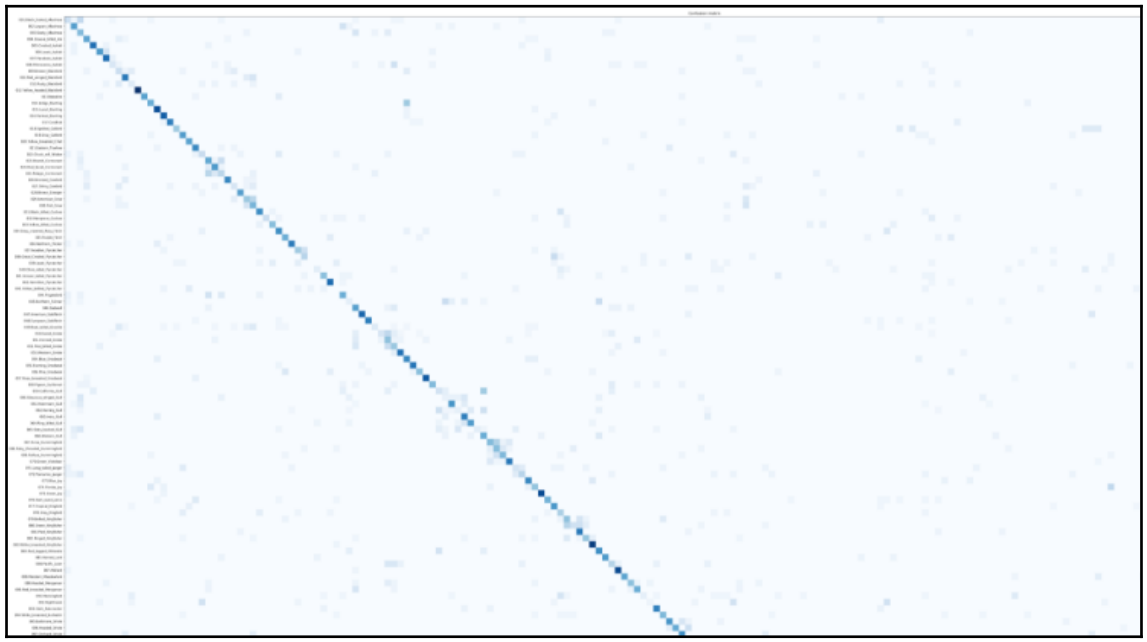
In [64]:
```
# from http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html
import matplotlib.pyplot as plt
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    #plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    #for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    #    plt.text(j, i, format(cm[i, j], fmt),
    #             horizontalalignment="center",
    #             color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

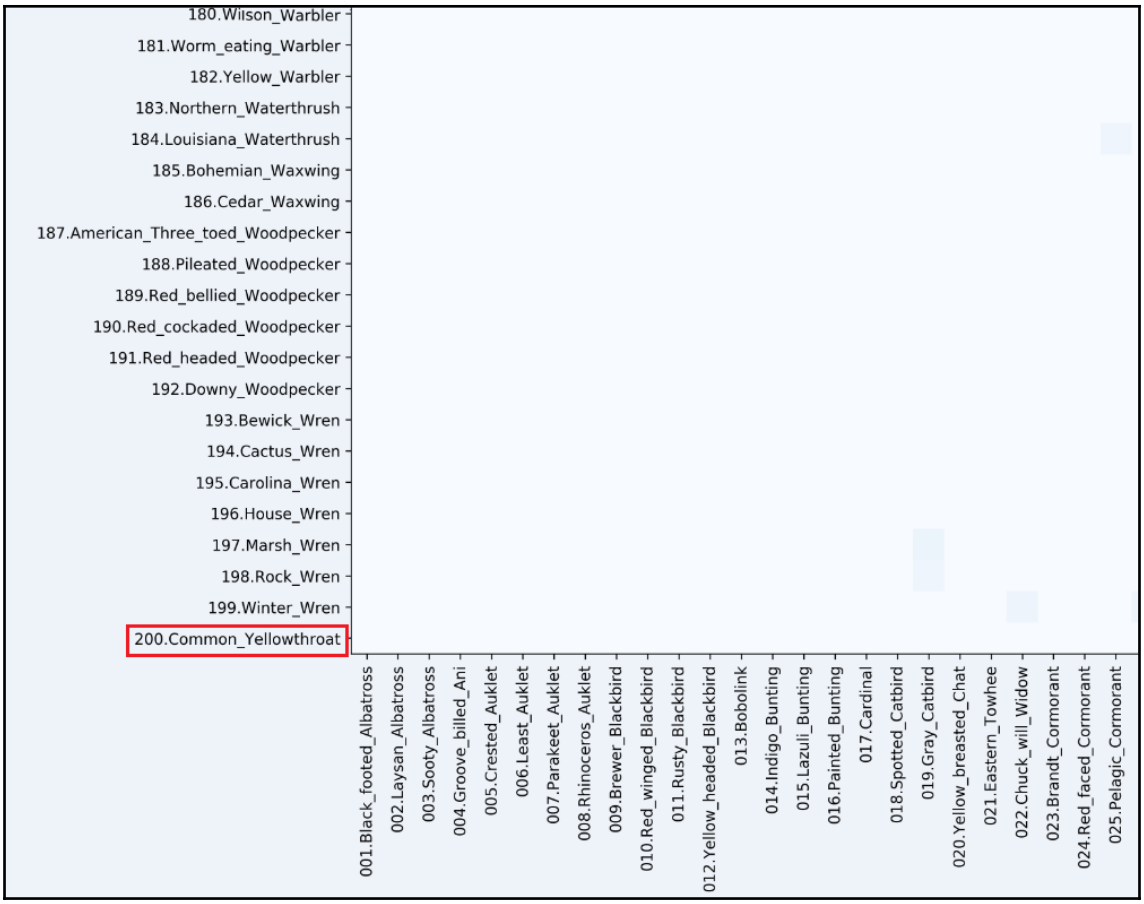
```
In [65]: birds = pd.read_csv("data/CUB_200_2011/classes.txt",
                             sep='\s+', header=None, usecols=[1], names=['birdname'])
         birds = birds['birdname']
         birds
```

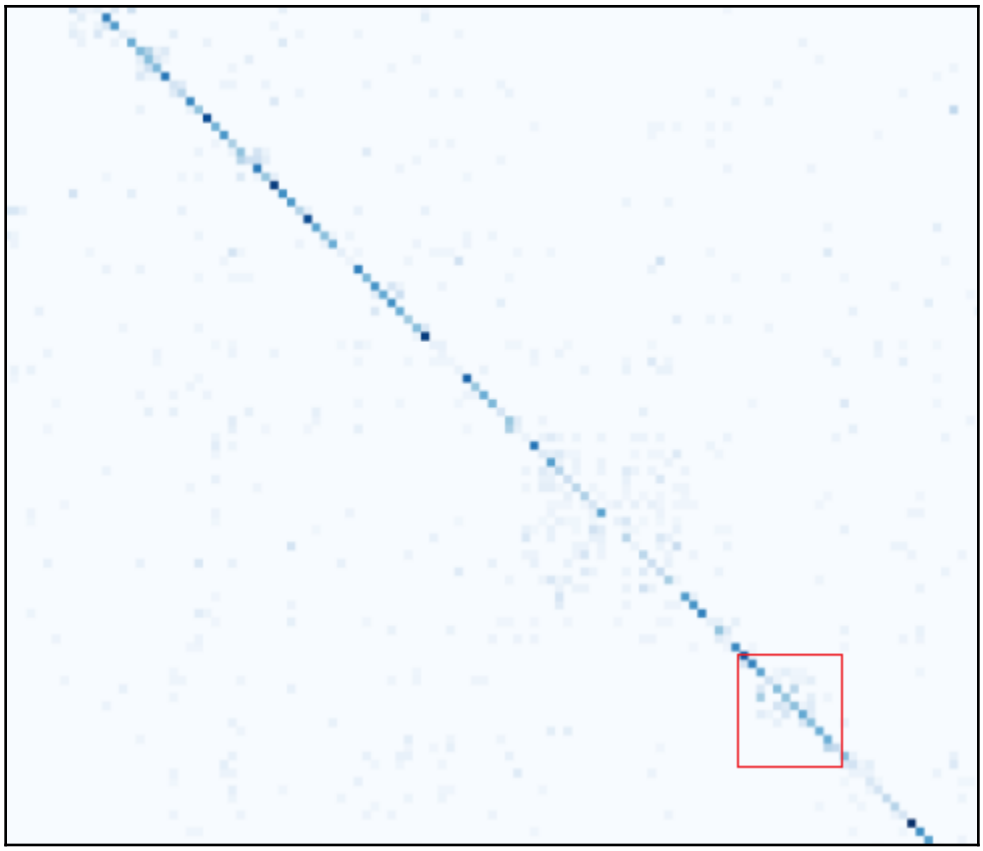
```
Out[65]: 0              001.Black_footed_Albatross
         1                   002.Laysan_Albatross
         2                    003.Sooty_Albatross
         3                  004.Groove_billed_Ani
         4                     005.Crested_Auklet
         5                       006.Least_Auklet
         6                    007.Parakeet_Auklet
         7                  008.Rhinoceros_Auklet
         8                   009.Brewer_Blackbird
         9                010.Red_winged_Blackbird
         10                    011.Rusty_Blackbird
         11             012.Yellow_headed_Blackbird
         12                          013.Bobolink
         13                   014.Indigo_Bunting
         14                   015.Lazuli_Bunting
         15                  016.Painted_Bunting
         16                         017.Cardinal
         17                  018.Spotted_Catbird
         18                     019.Gray_Catbird
         19               020.Yellow_breasted_Chat
         20                    021.Eastern_Towhee
         21                  022.Chuck_will_Widow
         22                  023.Brandt_Cormorant
         23               024.Red_faced_Cormorant
         24                 025.Pelagic_Cormorant
         25                   026.Bronzed_Cowbird
         26                     027.Shiny_Cowbird
         27                     028.Brown_Creeper
         28                     029.American_Crow
         29                         030.Fish_Crow
```

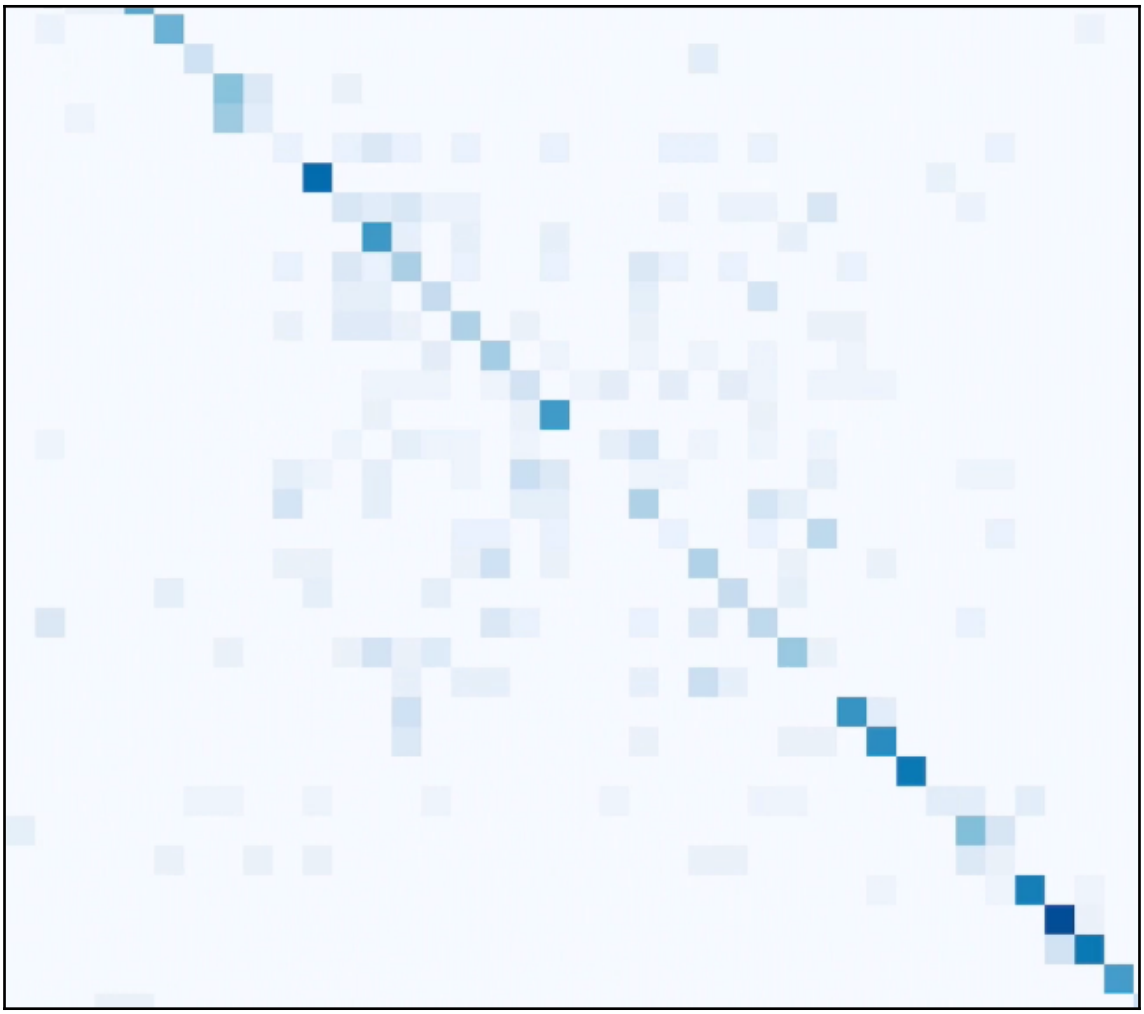
In [66]:
```
import numpy as np
np.set_printoptions(precision=2)
plt.figure(figsize=(60,60), dpi=300)
plot_confusion_matrix(cm, classes=birds, normalize=True)
plt.show()
```

```
Normalized confusion matrix
[[ 0.22  0.04  0.26 ...,  0.    0.04  0.  ]
 [ 0.    0.57  0.   ...,  0.    0.    0.  ]
 [ 0.    0.    0.44 ...,  0.    0.    0.  ]
 ...,
 [ 0.    0.    0.   ...,  0.25  0.    0.  ]
 [ 0.    0.    0.   ...,  0.    0.55  0.  ]
 [ 0.    0.    0.   ...,  0.    0.    0.76]]
```




180.Wilson_Warbler
181.Worm_eating_Warbler
182.Yellow_Warbler
183.Northern_Waterthrush
184.Louisiana_Waterthrush
185.Bohemian_Waxwing
186.Cedar_Waxwing
187.American_Three_toed_Woodpecker
188.Pileated_Woodpecker
189.Red_bellied_Woodpecker
190.Red_cockaded_Woodpecker
191.Red_headed_Woodpecker
192.Downy_Woodpecker
193.Bewick_Wren
194.Cactus_Wren
195.Carolina_Wren
196.House_Wren
197.Marsh_Wren
198.Rock_Wren
199.Winter_Wren
200.Common_Yellowthroat
001.Black_footed_Albatross
002.Laysan_Albatross
003.Sooty_Albatross
004.Groove_billed_Ani
005.Crested_Auklet
006.Least_Auklet
007.Parakeet_Auklet
008.Rhinoceros_Auklet
009.Brewer_Blackbird
010.Red_winged_Blackbird
011.Rusty_Blackbird
012.Yellow_headed_Blackbird
013.Bobolink
014.Indigo_Bunting
015.Lazuli_Bunting
016.Painted_Bunting
017.Cardinal
018.Spotted_Catbird
019.Gray_Catbird
020.Yellow_breasted_Chat
021.Eastern_Towhee
022.Chuck_will_Widow
023.Brandt_Cormorant
024.Red_faced_Cormorant
025.Pelagic_Cormorant





```
In [67]: from sklearn import tree
         clftree = tree.DecisionTreeClassifier()
         clftree.fit(df_train_att, df_train_label)
         clftree.score(df_test_att, df_test_label)

Out[67]: 0.26953537486800422
```

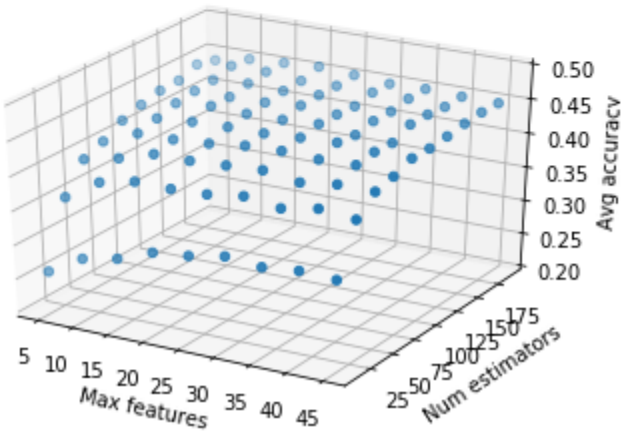
```
In [68]: from sklearn import svm
         clfsvm = svm.SVC()
         clfsvm.fit(df_train_att, df_train_label)
         clfsvm.score(df_test_att, df_test_label)

Out[68]: 0.28616684266103487
```

```
In [69]: from sklearn.model_selection import cross_val_score
         scores = cross_val_score(clf, df_train_att, df_train_label, cv=5)
         # show average score and +/- two standard deviations away (covering 95% of scores)
         print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

         Accuracy: 0.44 (+/- 0.02)

In [70]: scorestree = cross_val_score(clftree, df_train_att, df_train_label, cv=5)
         print("Accuracy: %0.2f (+/- %0.2f)" % (scorestree.mean(), scorestree.std() * 2))

         Accuracy: 0.25 (+/- 0.02)

In [71]: scoressvm = cross_val_score(clfsvm, df_train_att, df_train_label, cv=5)
         print("Accuracy: %0.2f (+/- %0.2f)" % (scoressvm.mean(), scoressvm.std() * 2))

         Accuracy: 0.27 (+/- 0.01)
```

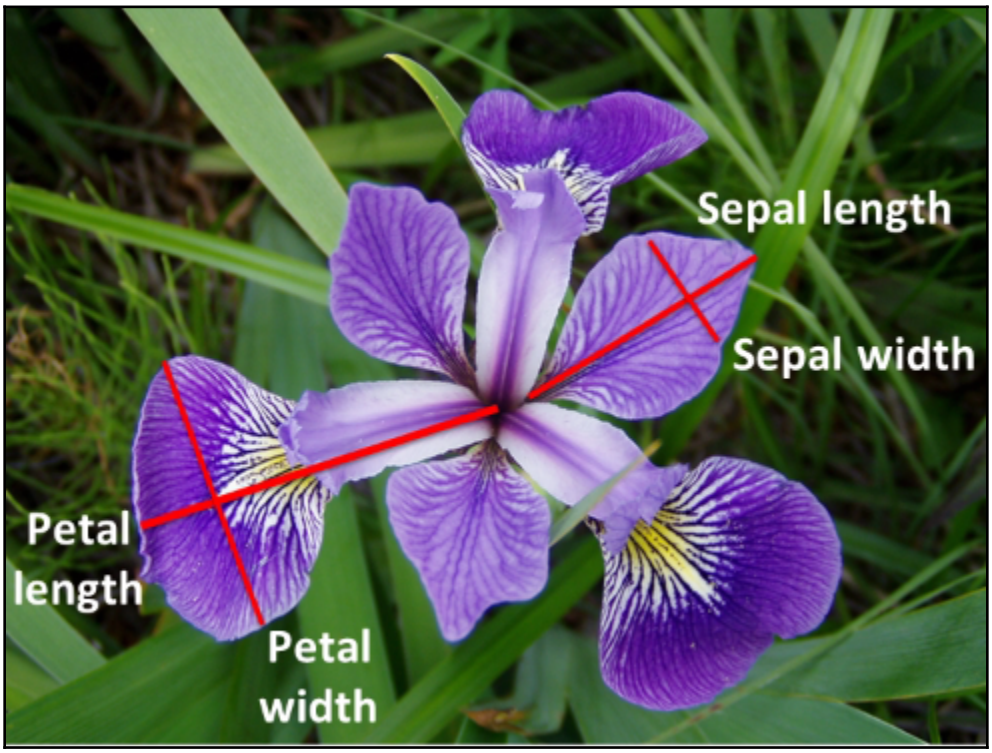
In [72]:
```
max_features_opts = range(5, 50, 5)
n_estimators_opts = range(10, 200, 20)
rf_params = np.empty((len(max_features_opts)*len(n_estimators_opts),4), float)
i = 0
for max_features in max_features_opts:
    for n_estimators in n_estimators_opts:
        clf = RandomForestClassifier(max_features=max_features, n_estimators=n_estimators)
        scores = cross_val_score(clf, df_train_att, df_train_label, cv=5)
        rf_params[i,0] = max_features
        rf_params[i,1] = n_estimators
        rf_params[i,2] = scores.mean()
        rf_params[i,3] = scores.std() * 2
        i += 1
        print("Max features: %d, num estimators: %d, accuracy: %0.2f (+/- %0.2f)" % \
              (max_features, n_estimators, scores.mean(), scores.std() * 2))
```

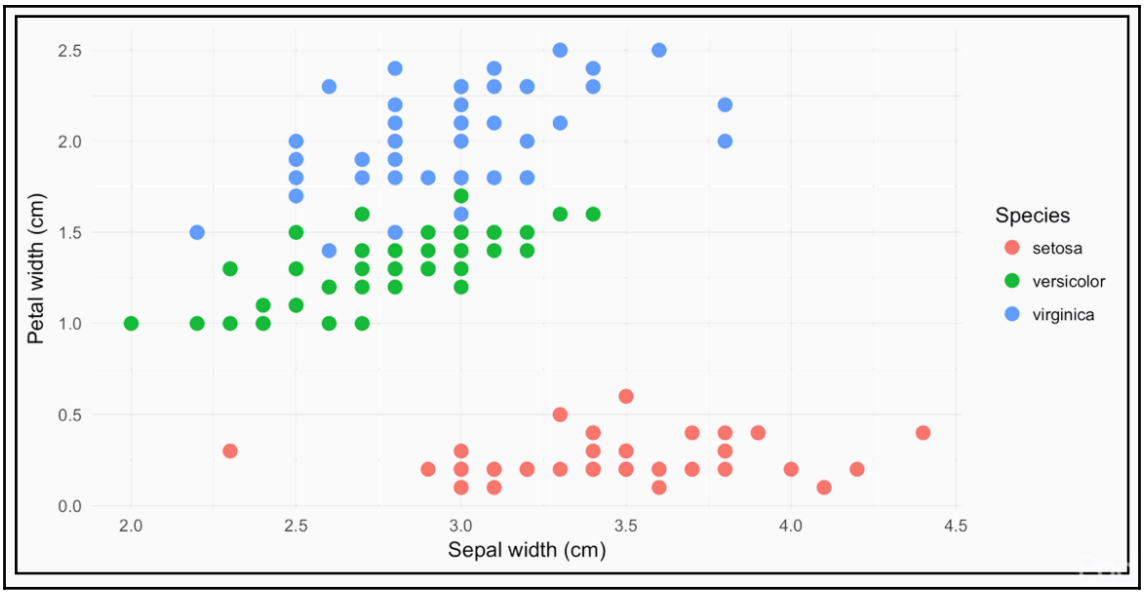
```
Max features: 5, num estimators: 10, accuracy: 0.26 (+/- 0.03)
Max features: 5, num estimators: 30, accuracy: 0.35 (+/- 0.02)
Max features: 5, num estimators: 50, accuracy: 0.39 (+/- 0.03)
Max features: 5, num estimators: 70, accuracy: 0.40 (+/- 0.04)
Max features: 5, num estimators: 90, accuracy: 0.42 (+/- 0.02)
Max features: 5, num estimators: 110, accuracy: 0.43 (+/- 0.02)
Max features: 5, num estimators: 130, accuracy: 0.44 (+/- 0.02)
Max features: 5, num estimators: 150, accuracy: 0.44 (+/- 0.03)
Max features: 5, num estimators: 170, accuracy: 0.45 (+/- 0.03)
Max features: 5, num estimators: 190, accuracy: 0.44 (+/- 0.02)
Max features: 10, num estimators: 10, accuracy: 0.29 (+/- 0.03)
Max features: 10, num estimators: 30, accuracy: 0.38 (+/- 0.03)
Max features: 10, num estimators: 50, accuracy: 0.40 (+/- 0.03)
Max features: 10, num estimators: 70, accuracy: 0.42 (+/- 0.02)
Max features: 10, num estimators: 90, accuracy: 0.43 (+/- 0.02)
Max features: 10, num estimators: 110, accuracy: 0.44 (+/- 0.03)
Max features: 10, num estimators: 130, accuracy: 0.44 (+/- 0.03)
Max features: 10, num estimators: 150, accuracy: 0.45 (+/- 0.02)
Max features: 10, num estimators: 170, accuracy: 0.45 (+/- 0.03)
Max features: 10, num estimators: 190, accuracy: 0.45 (+/- 0.03)
Max features: 15, num estimators: 10, accuracy: 0.30 (+/- 0.02)
Max features: 15, num estimators: 30, accuracy: 0.39 (+/- 0.02)
Max features: 15, num estimators: 50, accuracy: 0.42 (+/- 0.02)
Max features: 15, num estimators: 70, accuracy: 0.42 (+/- 0.02)
Max features: 15, num estimators: 90, accuracy: 0.44 (+/- 0.03)
Max features: 15, num estimators: 110, accuracy: 0.44 (+/- 0.03)
Max features: 15, num estimators: 130, accuracy: 0.44 (+/- 0.02)
Max features: 15, num estimators: 150, accuracy: 0.45 (+/- 0.03)
Max features: 15, num estimators: 170, accuracy: 0.45 (+/- 0.03)
```

In [90]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
fig = plt.figure()
fig.clf()
ax = fig.gca(projection='3d')
x = rf_params[:,0]
y = rf_params[:,1]
z = rf_params[:,2]
ax.scatter(x, y, z)
ax.set_zlim(0.2, 0.5)
ax.set_xlabel('Max features')
ax.set_ylabel('Num estimators')
ax.set_zlabel('Avg accuracy')
plt.show()
```



Chapter 3: Applications for Comment Classification





But what makes these phrases similar	And these dissimilar?
Please subscribe to my channel and I will subscribe back xx	The funny thing is that this song was made in 2009 but it took two years to get to America
Guys please subscribe me to help my channel grow please guys	Check my channel please! and listen to the best music ever :P

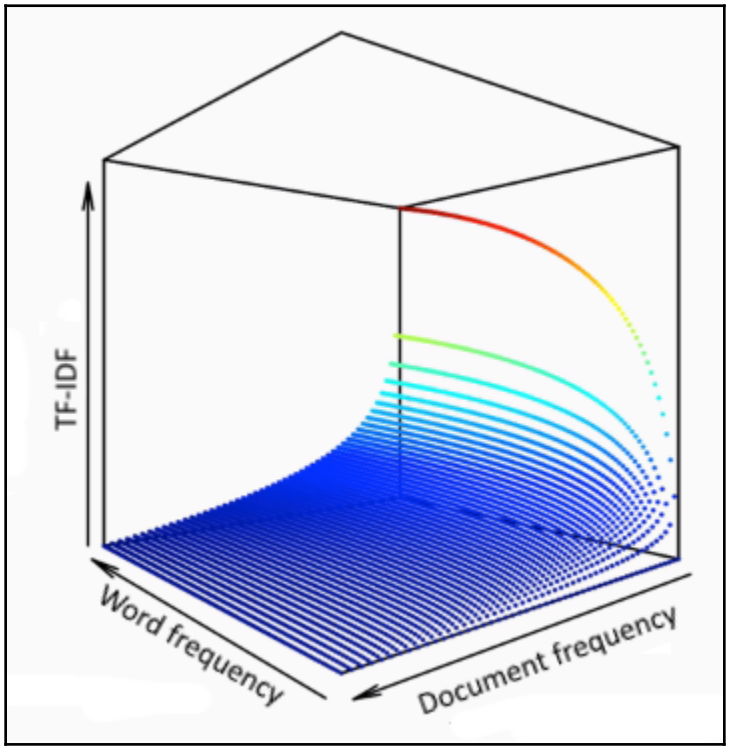
- Example one –

and	back	channel	i	my	plz	subscribe	to	xx
1	1	1	1	1	1	2	1	1

- Example two –

channel	grow	guys	help	me	my	please	subscribe	to
1	1	2	1	1	1	2	1	1

	and	back	channel	grow	guys	help	i	me	my	please	plz	subscribe	to	xx
Example one	1	1	1	0	0	0	1	0	1	0	1	2	1	1
Example two	0	0	1	1	2	1	0	1	1	1	0	1	1	0




```
In [1]: import pandas as pd
        d = pd.read_csv("D:/Chapter04/Youtube01-Psy.csv")
```

```
In [2]: d.tail()
```

Out[2]:

	COMMENT_ID	DATE	CONTENT	CLASS
345	z13th1q4yzihf1bll23qxzpjeujterydj	2014-11-14T13:27:52	How can this have 2 billion views when there's...	0
346	z13fcn1wfpb5e51xe04chdxakpzgchyaxzo0k	2014-11-14T13:28:08	I don't now why I'm watching this in 2014	0
347	z130zd5b3titudkoe04ccbeohojxuzppvbg	2015-05-23T13:04:32	subscribe to me for call of duty vids and give...	1
348	z12he50arvrkivl5u04cctawgxzkjfsjcc4	2015-06-05T14:14:48	hi guys please my android photo editor downloa...	1
349	z13vhvu54u3ewpp5h04ccb4zuoardrmjlyk0k	2015-06-05T18:05:16	The first billion viewed this because they tho...	0

```
In [3]: len(d.query('CLASS == 1'))
Out[3]: 175

In [4]: len(d.query('CLASS == 0'))
Out[4]: 175

In [5]: len(d)
Out[5]: 350
```

```
In [6]: from sklearn.feature_extraction.text import CountVectorizer
        vectorizer = CountVectorizer()

In [7]: dvec = vectorizer.fit_transform(d['CONTENT'])
```

```
In [7]: dvec = vectorizer.fit_transform(d['CONTENT'])

In [8]: dvec
Out[8]: <350x1418 sparse matrix of type '<class 'numpy.int64'>'
            with 4354 stored elements in Compressed Sparse Row format>
```

```
print(d['CONTENT'][349])
analyze(d['CONTENT'][349])
```

```
The first billion viewed this because they thought it was really cool, the  other billion and a half came to see how stupid t
he first billion were...

['the',
 'first',
 'billion',
 'viewed',
 'this',
 'because',
 'they',
 'thought',
 'it',
 'was',
 'really',
 'cool',
 'the',
 'other',
 'billion',
 'and',
 'half',
```

```
In [11]: vectorizer.get_feature_names()
```

```
Out[11]: ['00',
          '000',
          '02',
          '034',
          '05',
          '08',
          '10',
          '100',
          '100000415527985',
          '10200253113705769',
          '1030',
          '1073741828',
          '11',
          '1111',
          '112720997191206369631',
          '12',
          '123',
          '124',
          '124923004',
```

```
In [12]: dshuf = d.sample(frac=1)
```

```
In [13]: d_train = dshuf[:300]
         d_test = dshuf[300:]
         d_train_att = vectorizer.fit_transform(d_train['CONTENT']) # fit bag-of-words on training set
         d_test_att = vectorizer.transform(d_test['CONTENT']) # reuse on testing set
         d_train_label = d_train['CLASS']
         d_test_label = d_test['CLASS']
```

```
In [14]: d_train_att

Out[14]: <300x1253 sparse matrix of type '<class 'numpy.int64'>'
                with 3720 stored elements in Compressed Sparse Row format>

In [15]: d_test_att

Out[15]: <50x1253 sparse matrix of type '<class 'numpy.int64'>'
                with 467 stored elements in Compressed Sparse Row format>
```

```
In [16]: from sklearn.ensemble import RandomForestClassifier
         clf = RandomForestClassifier(n_estimators=80)

In [17]: clf.fit(d_train_att, d_train_label)

Out[17]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=80, n_jobs=1,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False)

In [18]: clf.score(d_test_att, d_test_label)

Out[18]: 0.97999999999999998
```

```
In [19]: from sklearn.metrics import confusion_matrix
         pred_labels = clf.predict(d_test_att)
         confusion_matrix(d_test_label, pred_labels)

Out[19]: array([[26,  0],
                [ 1, 23]], dtype=int64)

In [20]: from sklearn.model_selection import cross_val_score
         scores = cross_val_score(clf, d_train_att, d_train_label, cv=5)
         # show average score and +/- two standard deviations away (covering 95% of scores)
         print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

         Accuracy: 0.97 (+/- 0.03)
```

```
In [22]: # load all datasets and combine them
         d = pd.concat([pd.read_csv("D:/Chapter04/Youtube01-Psy.csv"),
                        pd.read_csv("D:/Chapter04/Youtube02-KatyPerry.csv"),
                        pd.read_csv("D:/Chapter04/Youtube03-LMFAO.csv"),
                        pd.read_csv("D:/Chapter04/Youtube04-Eminem.csv"),
                        pd.read_csv("D:/Chapter04/Youtube05-Shakira.csv")])

In [23]: len(d)

Out[23]: 1956
```

```
In [24]: len(d.query('CLASS == 1'))

Out[24]: 1005

In [25]: len(d.query('CLASS == 0'))

Out[25]: 951
```

```
In [26]: dshuf = d.sample(frac=1)
         d_content = dshuf['CONTENT']
         d_label = dshuf['CLASS']
```

```
In [27]: # set up a pipeline
         from sklearn.pipeline import Pipeline, make_pipeline
         pipeline = Pipeline([
             ('bag-of-words', CountVectorizer()),
             ('random forest', RandomForestClassifier()),
         ])
         pipeline

Out[27]: Pipeline(memory=None,
              steps=[('bag-of-words', CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                 dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                 lowercase=True, max_df=1.0, max_features=None, min_df=1,
                 ngram_range=(1, 1), preprocessor=None, stop_words=None,
                ...n_jobs=1,
                       oob_score=False, random_state=None, verbose=0,
                       warm_start=False))])
```

```
In [28]: # or: pipeline = make_pipeline(CountVectorizer(), RandomForestClassifier())
         make_pipeline(CountVectorizer(), RandomForestClassifier())

Out[28]: Pipeline(memory=None,
              steps=[('countvectorizer', CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                 dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                 lowercase=True, max_df=1.0, max_features=None, min_df=1,
                 ngram_range=(1, 1), preprocessor=None, stop_words=None,
                ...n_jobs=1,
                       oob_score=False, random_state=None, verbose=0,
                       warm_start=False))])
```

```
In [29]: pipeline.fit(d_content[:1500],d_label[:1500])

Out[29]: Pipeline(memory=None,
              steps=[('bag-of-words', CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                 dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                 lowercase=True, max_df=1.0, max_features=None, min_df=1,
                 ngram_range=(1, 1), preprocessor=None, stop_words=None,
                ...n_jobs=1,
                       oob_score=False, random_state=None, verbose=0,
                       warm_start=False))])
```

```
In [30]: pipeline.score(d_content[1500:], d_label[1500:])

Out[30]: 0.93859649122807021
```

```
In [31]: pipeline.predict(["what a neat video!"])

Out[31]: array([0], dtype=int64)
```

```
In [32]: pipeline.predict(["plz subscribe to my channel"])
Out[32]: array([1], dtype=int64)
```

```
In [33]: scores = cross_val_score(pipeline, d_content, d_label, cv=5)
         print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

         Accuracy: 0.94 (+/- 0.02)
```

```
In [34]: # add tfidf
         from sklearn.feature_extraction.text import TfidfTransformer
         pipeline2 = make_pipeline(CountVectorizer(),
                                   TfidfTransformer(norm=None),
                                   RandomForestClassifier())
```

```
In [35]: scores = cross_val_score(pipeline2, d_content, d_label, cv=5)
         print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

         Accuracy: 0.94 (+/- 0.03)
```

```
In [36]: pipeline2.steps
Out[36]: [('countvectorizer',
           CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                   dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                   lowercase=True, max_df=1.0, max_features=None, min_df=1,
                   ngram_range=(1, 1), preprocessor=None, stop_words=None,
                   strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                   tokenizer=None, vocabulary=None)),
          ('tfidftransformer',
           TfidfTransformer(norm=None, smooth_idf=True, sublinear_tf=False, use_idf=True)),
          ('randomforestclassifier',
           RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                       oob_score=False, random_state=None, verbose=0,
                       warm_start=False))]
```

In [37]:
```
# parameter search
parameters = {
    'countvectorizer__max_features': (None, 1000, 2000),
    'countvectorizer__ngram_range': ((1, 1), (1, 2)),  # unigrams or bigrams
    'countvectorizer__stop_words': ('english', None),
    'tfidftransformer__use_idf': (True, False), # effectively turn on/off tfidf
    'randomforestclassifier__n_estimators': (20, 50, 100)
}
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(pipeline2, parameters, n_jobs=-1, verbose=1)
```

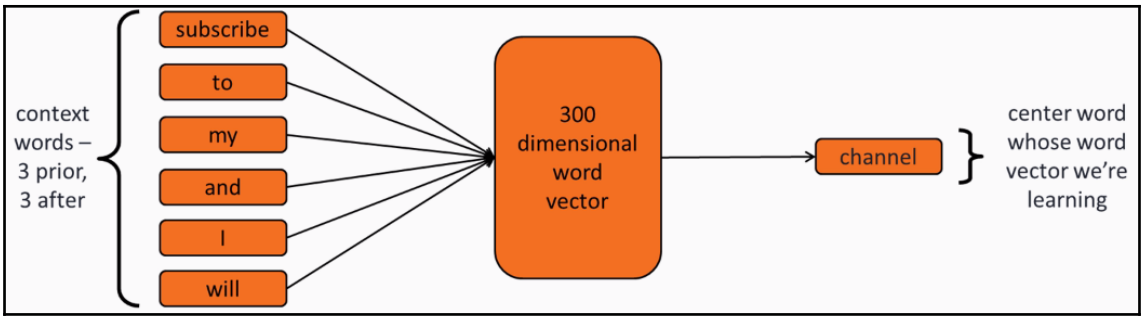
In [38]:
```
grid_search.fit(d_content, d_label)
```

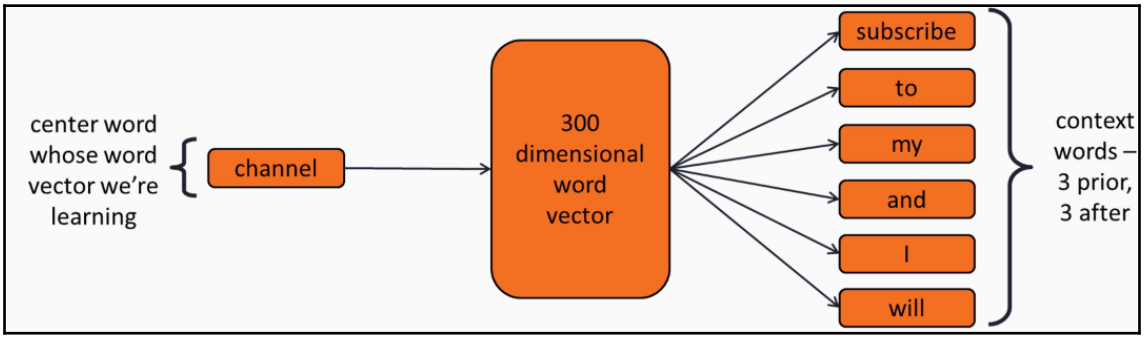
```
Fitting 3 folds for each of 72 candidates, totalling 216 fits

[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    6.6s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:   22.0s
[Parallel(n_jobs=-1)]: Done 216 out of 216 | elapsed:   25.0s finished
```

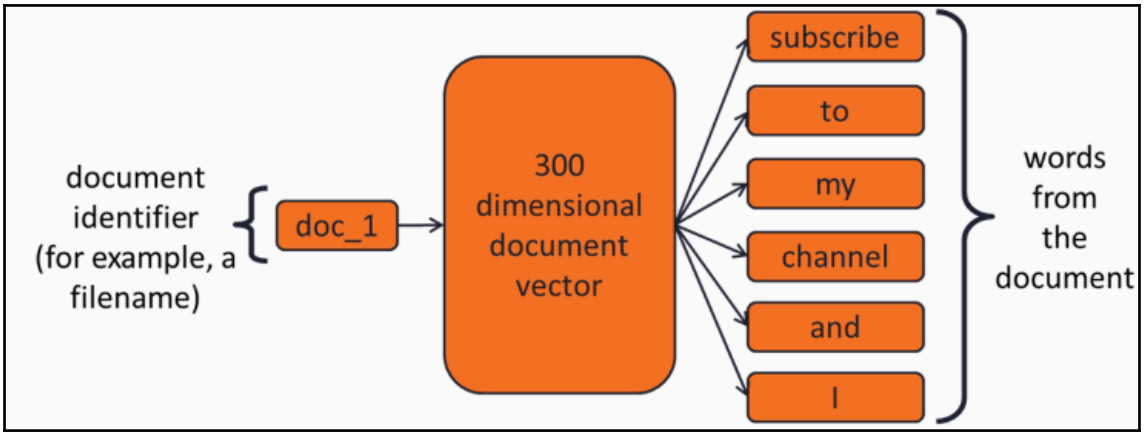
Out[38]:
```
GridSearchCV(cv=None, error_score='raise',
       estimator=Pipeline(memory=None,
     steps=[('countvectorizer', CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
  ...n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False))]),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'countvectorizer__max_features': (None, 1000, 2000), 'countvectorizer__ngram_range': ((1, 1), (1, 2)), 'count
vectorizer__stop_words': ('english', None), 'tfidftransformer__use_idf': (True, False), 'randomforestclassifier__n_estimators':
(20, 50, 100)},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=1)
```

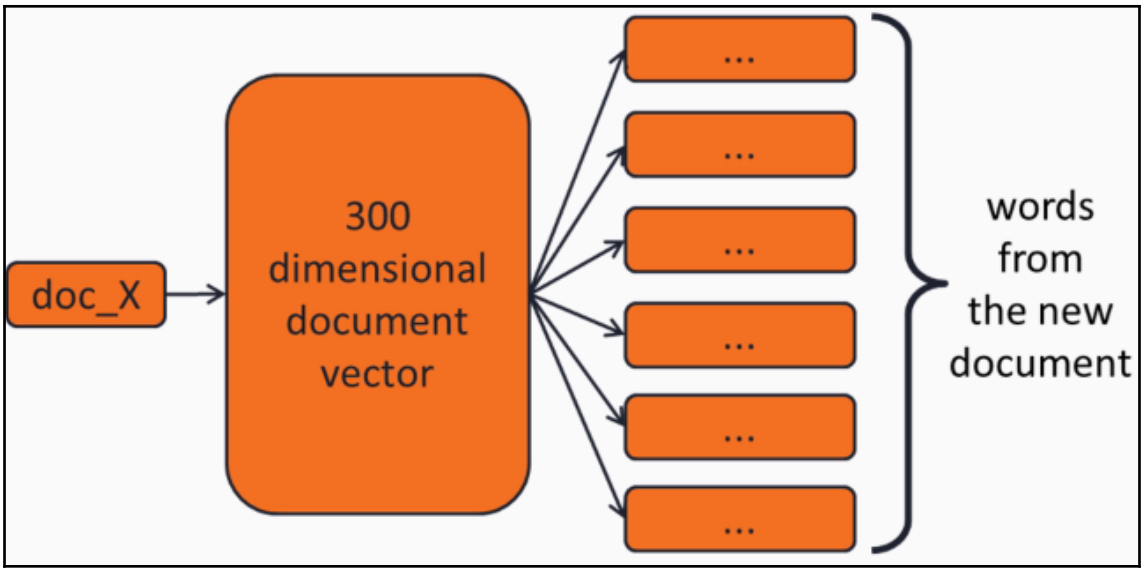
In [39]:
```
print("Best score: %0.3f" % grid_search.best_score_)
print("Best parameters set:")
best_parameters = grid_search.best_estimator_.get_params()
for param_name in sorted(parameters.keys()):
    print("\t%s: %r" % (param_name, best_parameters[param_name]))
```

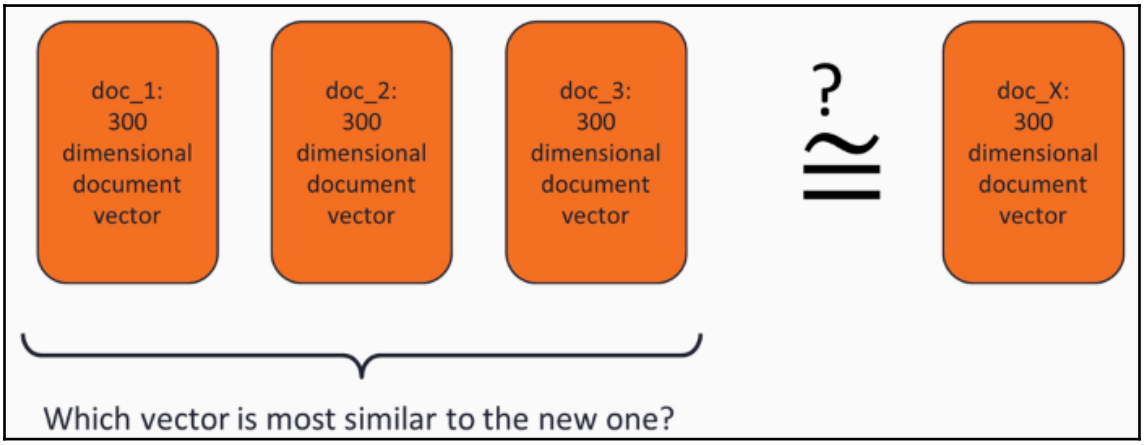
```
Best score: 0.963
Best parameters set:
        countvectorizer__max_features: 1000
        countvectorizer__ngram_range: (1, 1)
        countvectorizer__stop_words: 'english'
        randomforestclassifier__n_estimators: 50
        tfidftransformer__use_idf: True
```


context words – 3 prior, 3 after
subscribe
to
my
and
I
will
300 dimensional word vector
channel
center word whose word vector we're learning


center word whose word vector we're learning
channel
300 dimensional word vector
subscribe
to
my
and
I
will
context words – 3 prior, 3 after


document identifier (for example, a filename)
doc_1
300 dimensional document vector
subscribe
to
my
channel
and
I
words from the document






```
In [1]: import gensim, logging

In [2]: logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)

In [3]: gmodel = gensim.models.KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)

        2017-11-12 16:06:33,788 : INFO : loading projection weights from GoogleNews-vectors-negative300.bin
        2017-11-12 16:08:10,091 : INFO : loaded (3000000, 300) matrix from GoogleNews-vectors-negative300.bin
```

```
In [4]: gmodel['cat']

Out[4]: array([ 0.0123291 ,  0.20410156, -0.28515625,  0.21679688,  0.11816406,
                0.08300781,  0.04980469, -0.00952148,  0.22070312, -0.12597656,
                0.08056641, -0.5859375 , -0.00445557, -0.296875  , -0.01312256,
               -0.08349609,  0.05053711,  0.15136719, -0.44921875, -0.0135498 ,
                0.21484375, -0.14746094,  0.22460938, -0.125     , -0.09716797,
                0.24902344, -0.2890625 ,  0.36523438,  0.41210938, -0.0859375 ,
               -0.07861328, -0.19726562, -0.09082031, -0.14160156, -0.10253906,
                0.13085938, -0.00346375,  0.07226562,  0.04418945,  0.34570312,
                0.07470703, -0.11230469,  0.06738281,  0.11230469,  0.01977539,
               -0.12353516,  0.20996094, -0.07226562, -0.02783203,  0.05541992,
               -0.33398438,  0.08544922,  0.34375   ,  0.13964844,  0.04931641,
               -0.13476562,  0.16308594, -0.37304688,  0.39648438,  0.10693359,
                0.22167969,  0.21289062, -0.08984375,  0.20703125,  0.08935547,
               -0.08251953,  0.05957031,  0.10205078, -0.19238281, -0.09082031,
                0.4921875 ,  0.03955078, -0.07080078, -0.0019989 , -0.23046875,
                0.25585938,  0.08984375, -0.10644531,  0.00105286, -0.05883789,
                0.05102539, -0.0291748 ,  0.19335938, -0.14160156, -0.33398438,
                0.08154297, -0.27539062,  0.10058594, -0.10449219, -0.12353516,
               -0.140625  ,  0.03491211, -0.11767578, -0.1796875 , -0.21484375,
               -0.23828125,  0.08447266, -0.07519531, -0.25976562, -0.21289062,
               -0.22363281, -0.09716797,  0.11572266,  0.15429688,  0.07373047,
               -0.27539062,  0.14257812, -0.0201416 ,  0.10009766, -0.19042969,
               -0.09375   ,  0.14160156,  0.17089844,  0.3125    , -0.16699219,
```

```
In [5]: gmodel['dog']

Out[5]: array([  5.12695312e-02,  -2.23388672e-02,  -1.72851562e-01,
                 1.61132812e-01,  -8.44726562e-02,   5.73730469e-02,
                 5.85937500e-02,  -8.25195312e-02,  -1.53808594e-02,
                -6.34765625e-02,   1.79687500e-01,  -4.23828125e-01,
                -2.25830078e-02,  -1.66015625e-01,  -2.51464844e-02,
                 1.07421875e-01,  -1.99218750e-01,   1.59179688e-01,
                -1.87500000e-01,  -1.20117188e-01,   1.55273438e-01,
                -9.91210938e-02,   1.42578125e-01,  -1.64062500e-01,
                -8.93554688e-02,   2.00195312e-01,  -1.49414062e-01,
                 3.20312500e-01,   3.28125000e-01,   2.44140625e-02,
                -9.71679688e-02,  -8.20312500e-02,  -3.63769531e-02,
                -8.59375000e-02,  -9.86328125e-02,   7.78198242e-03,
                -1.34277344e-02,   5.27343750e-02,   1.48437500e-01,
                 3.33984375e-01,   1.66015625e-02,  -2.12890625e-01,
                -1.50756836e-02,   5.24902344e-02,  -1.07421875e-01,
                -8.88671875e-02,   2.49023438e-01,  -7.03125000e-02,
                -1.59912109e-02,   7.56835938e-02,  -7.03125000e-02,
                 1.19140625e-01,   2.29492188e-01,   1.41601562e-02,
                 1.15234375e-01,   7.50732422e-03,   2.75390625e-01,
                -2.44140625e-01,   2.96875000e-01,   3.49121094e-02,
                 2.42187500e-01,   1.35742188e-01,   1.42578125e-01,
```

```
In [6]: gmodel['spatula']

Out[6]: array([-0.19140625, -0.04296875,  0.27539062,  0.00488281, -0.3203125 ,
                0.08203125,  0.05566406, -0.03613281, -0.31445312,  0.10693359,
               -0.359375  ,  0.29882812,  0.02331543,  0.05517578, -0.140625  ,
                0.1953125 , -0.23632812, -0.22167969, -0.06542969, -0.3359375 ,
                0.25195312, -0.09326172,  0.54296875,  0.11328125, -0.28710938,
               -0.12011719, -0.11181641,  0.20996094, -0.33203125,  0.30273438,
               -0.3359375 , -0.12255859,  0.12890625, -0.28515625, -0.04223633,
                0.25585938,  0.3203125 ,  0.07177734,  0.19042969, -0.01379395,
                0.16992188, -0.22460938,  0.5078125 ,  0.08398438, -0.07519531,
               -0.06396484,  0.05371094,  0.34570312,  0.46289062, -0.16699219,
               -0.30664062,  0.15234375, -0.09765625, -0.26171875, -0.14160156,
                0.2265625 ,  0.49609375, -0.10791016, -0.08447266,  0.234375  ,
                0.04931641, -0.07128906,  0.05273438, -0.11914062,  0.09814453,
                0.11181641, -0.13574219, -0.46875   ,  0.26171875,  0.12158203,
                0.31445312,  0.05810547,  0.0703125 , -0.10107422, -0.27734375,
               -0.16796875, -0.07128906, -0.08007812,  0.07226562, -0.1484375 ,
```

```
In [7]: gmodel.similarity('cat', 'dog')
Out[7]: 0.76094570897822089
```

```
In [8]: gmodel.similarity('cat', 'spatula')
Out[8]: 0.12412612600429632
```

```
In [9]: from gensim.models.doc2vec import TaggedDocument
        from gensim.models import Doc2Vec
```

```
In [10]: def extract_words(sent):
             sent = sent.lower()
             sent = re.sub(r'<[^>]+>', ' ', sent) # strip html tags
             sent = re.sub(r'(\w)\'(\w)', '\1\2', sent) # remove apostrophes
             sent = re.sub(r'\W', ' ', sent) # remove punctuation
             sent = re.sub(r'\s+', ' ', sent) # remove repeated spaces
             sent = sent.strip()
             return sent.split()
```

```
In [11]: # unsupervised training data
         import re
         import os
         unsup_sentences = []

         # source: http://ai.stanford.edu/~amaas/data/sentiment/, data from IMDB
         for dirname in ["train/pos", "train/neg", "train/unsup", "test/pos", "test/neg"]:
             for fname in sorted(os.listdir("aclImdb/" + dirname)):
                 if fname[-4:] == '.txt':
                     with open("aclImdb/" + dirname + "/" + fname, encoding='UTF-8') as f:
                         sent = f.read()
                         words = extract_words(sent)
                         unsup_sentences.append(TaggedDocument(words, [dirname + "/" + fname]))

         # source: http://www.cs.cornell.edu/people/pabo/movie-review-data/
         for dirname in ["review_polarity/txt_sentoken/pos", "review_polarity/txt_sentoken/neg"]:
             for fname in sorted(os.listdir(dirname)):
                 if fname[-4:] == '.txt':
                     with open(dirname + "/" + fname, encoding='UTF-8') as f:
                         for i, sent in enumerate(f):
                             words = extract_words(sent)
                             unsup_sentences.append(TaggedDocument(words, ["%s/%s-%d" % (dirname, fname, i)]))

         # source: https://nlp.stanford.edu/sentiment/, data from Rotten Tomatoes
         with open("stanfordSentimentTreebank/original_rt_snippets.txt", encoding='UTF-8') as f:
             for i, line in enumerate(f):
                 words = extract_words(sent)
                 unsup_sentences.append(TaggedDocument(words, ["rt-%d" % i]))
```

```
In [12]: len(unsup_sentences)
Out[12]: 175325
```

```
In [13]: unsup_sentences[0:10]
```

```
Out[13]: [TaggedDocument(words=['bromwell', 'high', 'is', 'a', 'cartoon', 'comedy', 'it', 'ran', 'at', 'the', 'same', 'time', 'as', 's
ome', 'other', 'programs', 'about', 'school', 'life', 'such', 'as', 'teachers', 'my', '35', 'years', 'in', 'the', 'teaching',
'profession', 'lead', 'me', 'to', 'believe', 'that', 'bromwell', 'hig', 'satire', 'is', 'much', 'closer', 'to', 'reality', 't
han', 'is', 'teachers', 'the', 'scramble', 'to', 'survive', 'financially', 'the', 'insightful', 'students', 'who', 'can', 'se
e', 'right', 'through', 'their', 'pathetic', 'teachers', 'pomp', 'the', 'pettiness', 'of', 'the', 'whole', 'situation', 'al
l', 'remind', 'me', 'of', 'the', 'schools', 'i', 'knew', 'and', 'their', 'students', 'when', 'i', 'saw', 'the', 'episode', 'i
n', 'which', 'a', 'student', 'repeatedly', 'tried', 'to', 'burn', 'down', 'the', 'school', 'i', 'immediately', 'recalled', 'a
t', 'high', 'a', 'classic', 'line', 'inspector', 'here', 'to', 'sack', 'one', 'of', 'your', 'teachers', 'student', 'welcome',
'to', 'bromwell', 'high', 'i', 'expect', 'that', 'many', 'adults', 'of', 'my', 'age', 'think', 'that', 'bromwell', 'high', 'i
s', 'far', 'fetched', 'what', 'a', 'pity', 'that', 'it', 'is'], tags=['train/pos/0_9.txt']),
 TaggedDocument(words=['homelessness', 'or', 'houselessness', 'as', 'george', 'carlin', 'stated', 'has', 'been', 'an', 'issu
e', 'for', 'years', 'but', 'never', 'a', 'plan', 'to', 'help', 'those', 'on', 'the', 'street', 'that', 'were', 'once', 'consi
dered', 'human', 'who', 'did', 'everything', 'from', 'going', 'to', 'school', 'work', 'or', 'vote', 'for', 'the', 'matter',
'most', 'people', 'think', 'of', 'the', 'homeless', 'as', 'just', 'a', 'lost', 'cause', 'while', 'worrying', 'about', 'thing
s', 'such', 'as', 'racism', 'the', 'war', 'on', 'iraq', 'pressuring', 'kids', 'to', 'succeed', 'technology', 'the', 'election
s', 'inflation', 'or', 'worrying', 'if', 'the', 'l', 'be', 'next', 'to', 'end', 'up', 'on', 'the', 'streets', 'but', 'what',
'if', 'you', 'were', 'given', 'a', 'bet', 'to', 'live', 'on', 'the', 'streets', 'for', 'a', 'month', 'without', 'the', 'luxur
ies', 'you', 'once', 'had', 'from', 'a', 'home', 'the', 'entertainment', 'sets', 'a', 'bathroom', 'pictures', 'on', 'the', 'w
all', 'a', 'computer', 'and', 'everything', 'you', 'once', 'treasure', 'to', 'see', 'what', 'i', 'like', 'to', 'be', 'homeles
```

```
In [15]: permuter = PermuteSentences(unsup_sentences)
         model = Doc2Vec(permuter, dm=0, hs=1, size=50)

2017-11-12 16:09:07,700 : WARNING : consider setting layer size to a multiple of 4 for greater performance
2017-11-12 16:09:07,703 : INFO : collecting all words and their counts
2017-11-12 16:09:08,097 : INFO : PROGRESS: at example #0, processed 0 words (0/s), 0 word types, 0 tags
2017-11-12 16:09:08,837 : INFO : PROGRESS: at example #10000, processed 1389399 words (1883036/s), 44788 word types, 10000 ta
gs
2017-11-12 16:09:09,546 : INFO : PROGRESS: at example #20000, processed 2818609 words (2024079/s), 61270 word types, 20000 ta
gs
2017-11-12 16:09:10,234 : INFO : PROGRESS: at example #30000, processed 4225040 words (2051374/s), 72774 word types, 30000 ta
gs
2017-11-12 16:09:10,921 : INFO : PROGRESS: at example #40000, processed 5632885 words (2057947/s), 82225 word types, 40000 ta
gs
2017-11-12 16:09:11,602 : INFO : PROGRESS: at example #50000, processed 7040997 words (2074544/s), 90454 word types, 50000 ta
gs
2017-11-12 16:09:12,284 : INFO : PROGRESS: at example #60000, processed 8474509 words (2106160/s), 97853 word types, 60000 ta
gs
2017-11-12 16:09:12,983 : INFO : PROGRESS: at example #70000, processed 9855488 words (1983678/s), 104567 word types, 70000 t
ags
2017-11-12 16:09:13,669 : INFO : PROGRESS: at example #80000, processed 11260351 words (2053644/s), 110705 word types, 80000
tags
```

```
In [16]: # done with training, free up some memory
         model.delete_temporary_training_data(keep_inference=True)
```

```
In [17]: model.save('reviews.d2v')
         # in other program, we could write: model = Doc2Vec.load('reviews.d2v')

2017-11-12 16:14:35,953 : INFO : saving Doc2Vec object under reviews.d2v, separately None
2017-11-12 16:14:35,956 : INFO : not storing attribute syn0norm
2017-11-12 16:14:35,958 : INFO : not storing attribute cum_table
2017-11-12 16:14:40,700 : INFO : saved reviews.d2v
```

```
In [18]: model.infer_vector(extract_words("This place is not worth your time, let alone Vegas."))

Out[18]: array([ 0.16600764,  0.29806057, -0.37614173,  0.58661956,  0.31548923,
                -0.15109532, -0.19294184, -0.80975324, -0.13256417, -0.26431978,
                 0.15649557, -0.36540538, -0.33639464, -0.55479848, -0.02375498,
                 0.12179437, -0.06088163, -0.17349492, -0.19584687,  0.09399831,
                 0.01947556, -0.17546433, -0.07536539, -0.05634249,  0.2418247 ,
                -0.11649339, -0.18398936, -0.37568066, -0.04755535, -0.23786636,
                 0.35202903, -0.25357839,  0.05126057, -0.22089498,  0.09130105,
                -0.46730992,  0.34186646,  0.17174301,  0.51055247,  0.21438542,
                -0.41699263, -0.5968706 , -0.00541743,  0.39446551,  0.07960459,
                -0.20494871,  0.11499975,  0.22761559,  0.24039924, -0.06279976], dtype=float32)
```

```
In [19]: from sklearn.metrics.pairwise import cosine_similarity
         cosine_similarity(
             [model.infer_vector(extract_words("This place is not worth your time, let alone Vegas."))],
             [model.infer_vector(extract_words("Service sucks."))])

Out[19]: array([[ 0.48211202]], dtype=float32)
```

```
In [20]: cosine_similarity(
             [model.infer_vector(extract_words("Highly recommended."))],
             [model.infer_vector(extract_words("Service sucks."))])

Out[20]: array([[ 0.28899333]], dtype=float32)
```

```
In [21]: sentences = []
         sentvecs = []
         sentiments = []
         for fname in ["yelp", "amazon_cells", "imdb"]:
             with open("sentiment labelled sentences/%s_labelled.txt" % fname, encoding='UTF-8') as f:
                 for i, line in enumerate(f):
                     line_split = line.strip().split('\t')
                     sentences.append(line_split[0])
                     words = extract_words(line_split[0])
                     sentvecs.append(model.infer_vector(words, steps=10)) # create a vector for this document
                     sentiments.append(int(line_split[1]))

         # shuffle sentences, sentvecs, sentiments together
         combined = list(zip(sentences, sentvecs, sentiments))
         random.shuffle(combined)
         sentences, sentvecs, sentiments = zip(*combined)
```

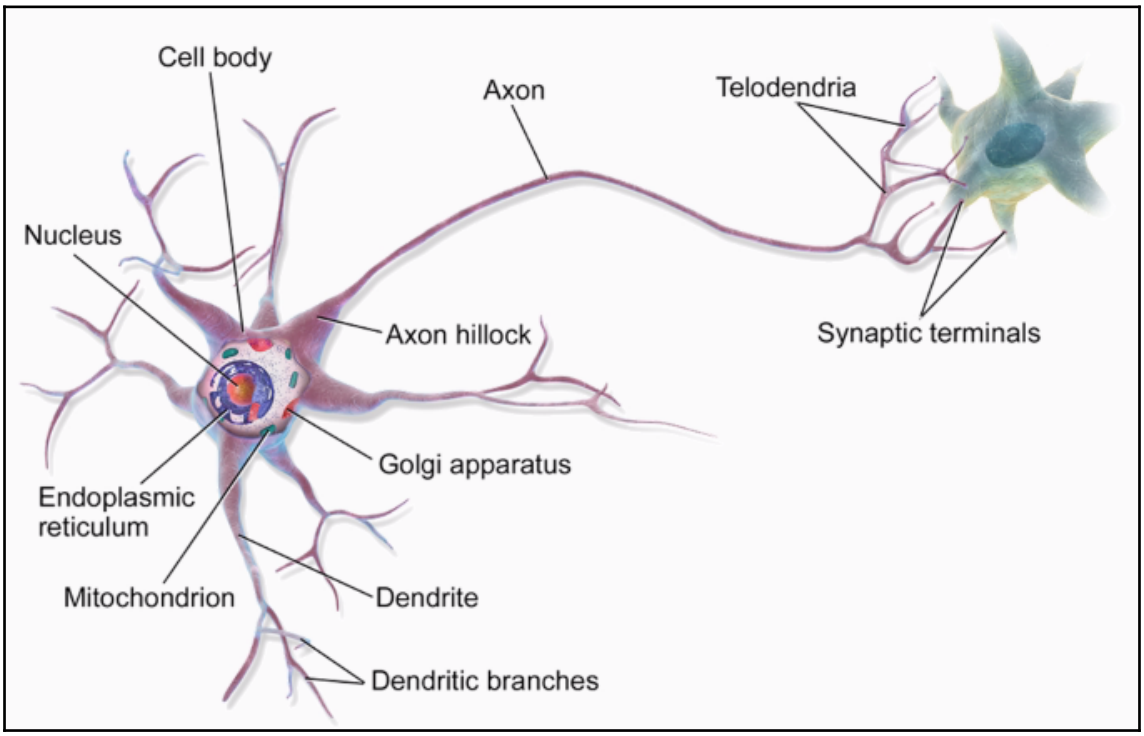
```
In [22]: from sklearn.neighbors import KNeighborsClassifier
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.model_selection import cross_val_score
         import numpy as np

         clf = KNeighborsClassifier(n_neighbors=9)
         clfrf = RandomForestClassifier()
```

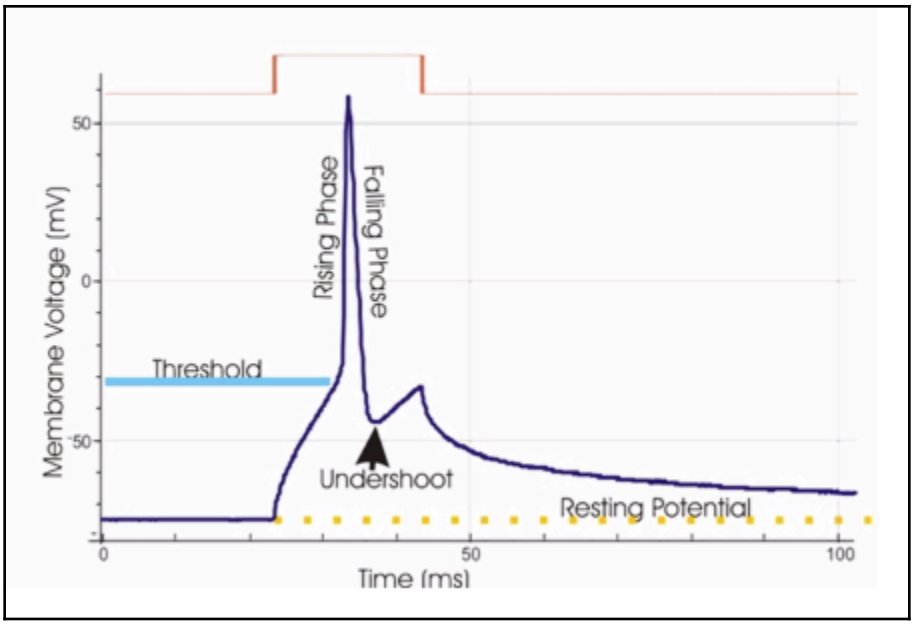
```
In [23]: scores = cross_val_score(clf, sentvecs, sentiments, cv=5)
         np.mean(scores), np.std(scores)

Out[23]: (0.7590000000000012, 0.016950909513454807)
```

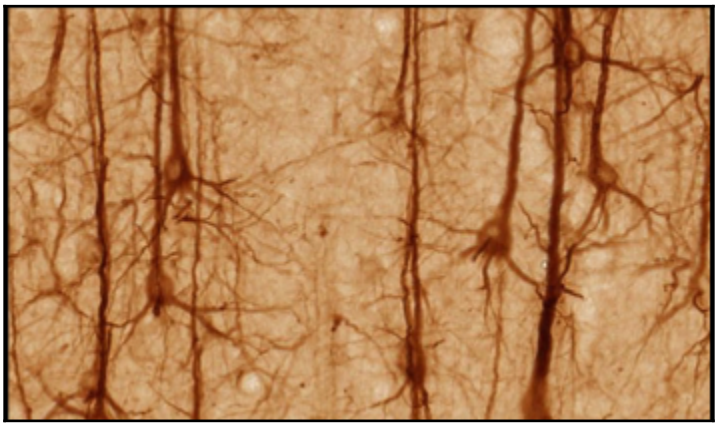
```
In [24]: scores = cross_val_score(clfrf, sentvecs, sentiments, cv=5)
         np.mean(scores), np.std(scores)

Out[24]: (0.69766666666666655, 0.019988885800753264)
```

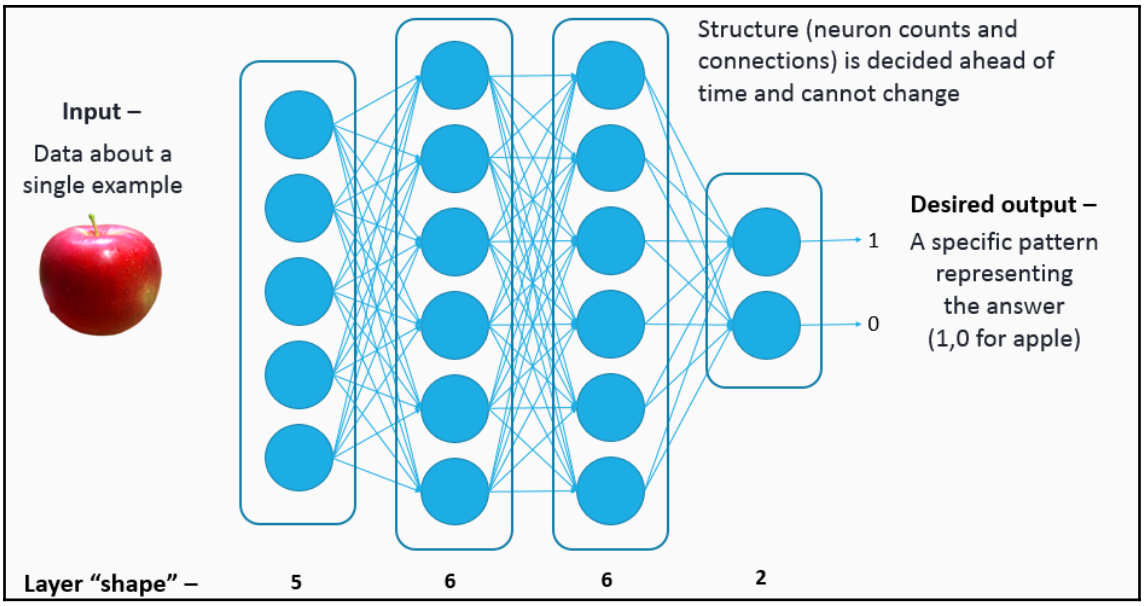
```
In [25]: # bag-of-words comparison
         from sklearn.pipeline import make_pipeline
         from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
         pipeline = make_pipeline(CountVectorizer(), TfidfTransformer(), RandomForestClassifier())

In [26]: scores = cross_val_score(pipeline, sentences, sentiments, cv=5)
         np.mean(scores), np.std(scores)

Out[26]: (0.73733333333333329, 0.015937377450509209)
```

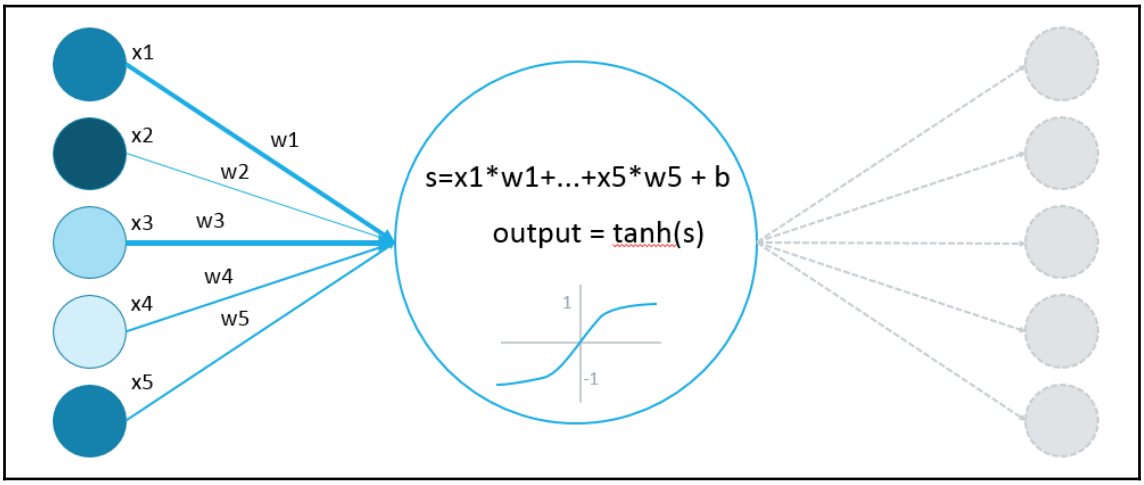
Chapter 4: Neural Networks

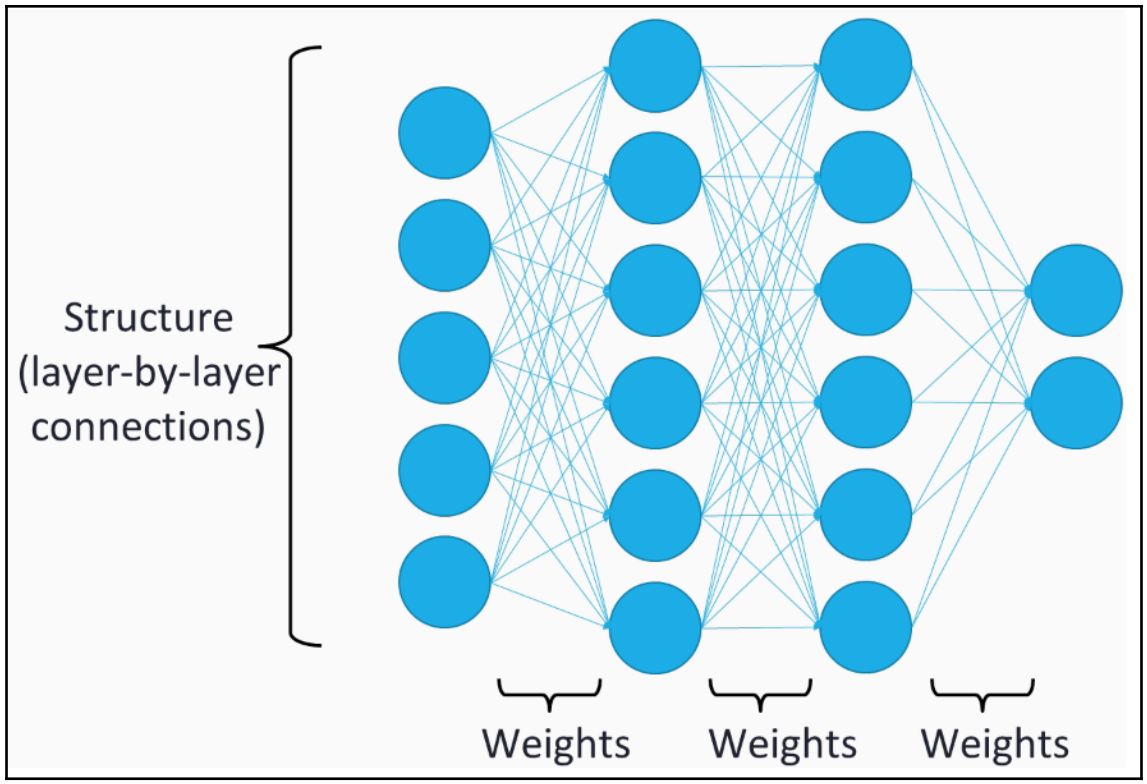


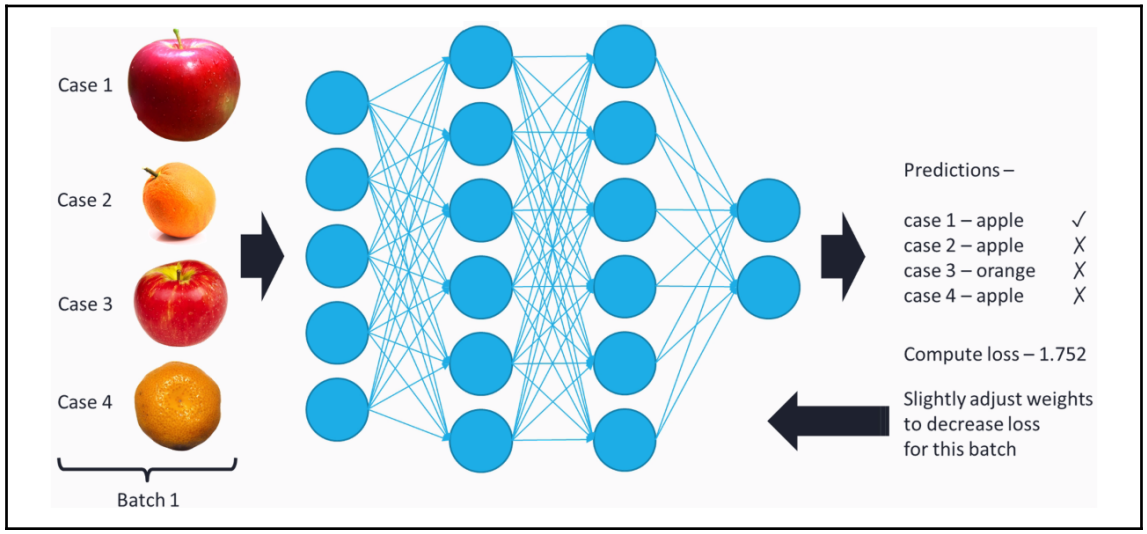

Membrane Voltage (mV)
50
0
-50
Rising Phase
Falling Phase
Threshold
Undershoot
Resting Potential
0
50
100
Time (ms)

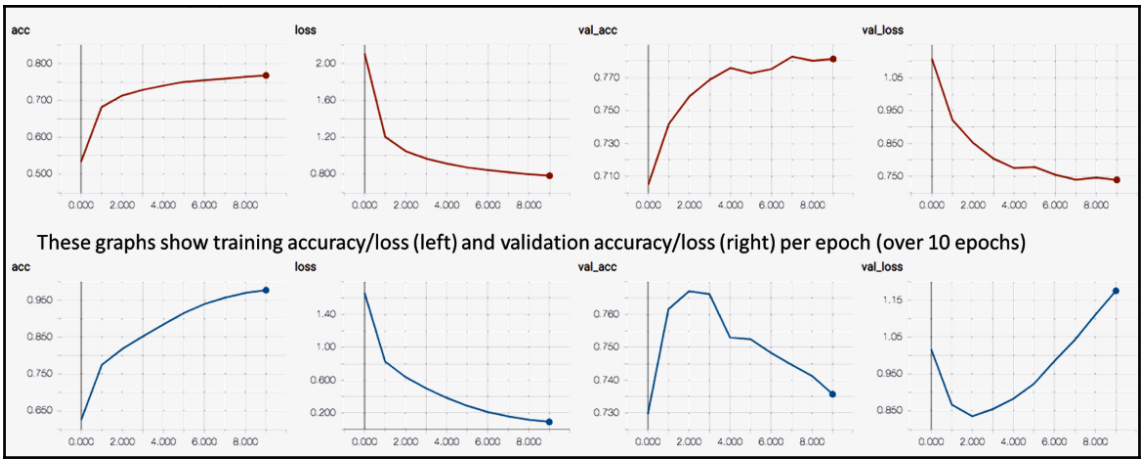



Input –
Data about a single example
Structure (neuron counts and connections) is decided ahead of time and cannot change
Desired output –
A specific pattern representing the answer (1,0 for apple)
1
0
Layer "shape" –
5
6
6
2


x1
x2
x3
x4
x5
w1
w2
w3
w4
w5
s=x1*w1+...+x5*w5 + b
output = tanh(s)
1
-1


Structure
(layer-by-layer
connections)
Weights
Weights
Weights


Case 1
Case 2
Case 3
Case 4
Batch 1
Predictions –
case 1 – apple ✓
case 2 – apple ✗
case 3 – orange ✗
case 4 – apple ✗
Compute loss – 1.752
Slightly adjust weights
to decrease loss
for this batch

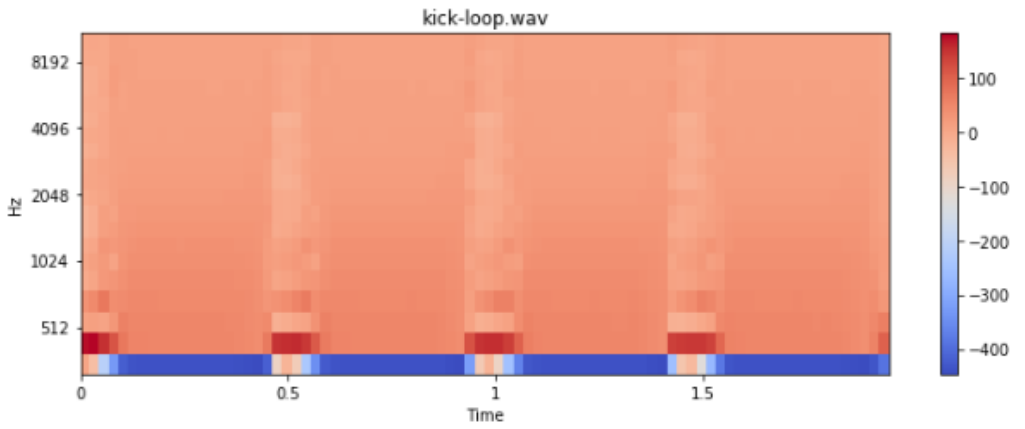


These graphs show training accuracy/loss (left) and validation accuracy/loss (right) per epoch (over 10 epochs)

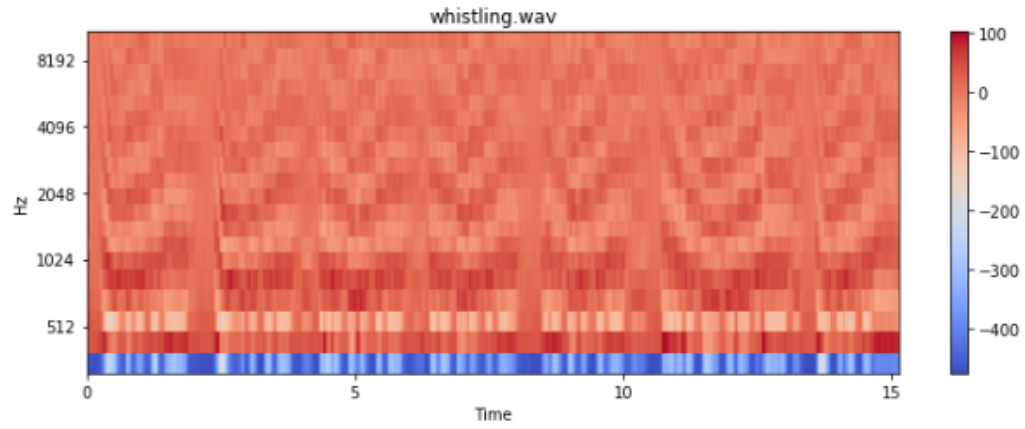
```
In [1]: import librosa
        import librosa.feature
        import librosa.display
        import glob
        import numpy as np
        import matplotlib.pyplot as plt
        from keras.models import Sequential
        from keras.layers import Dense, Activation
        from keras.utils.np_utils import to_categorical

        Using TensorFlow backend.
```

```
In [2]: def display_mfcc(song):
            y, _ = librosa.load(song)
            mfcc = librosa.feature.mfcc(y)

            plt.figure(figsize=(10, 4))
            librosa.display.specshow(mfcc, x_axis='time', y_axis='mel')
            plt.colorbar()
            plt.title(song)
            plt.tight_layout()
            plt.show()
```

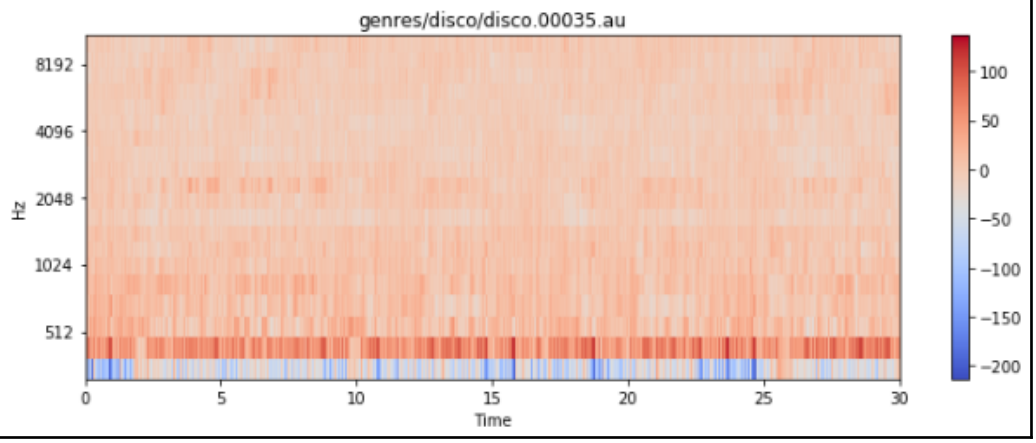
```
In [3]: display_mfcc('kick-loop.wav')
```



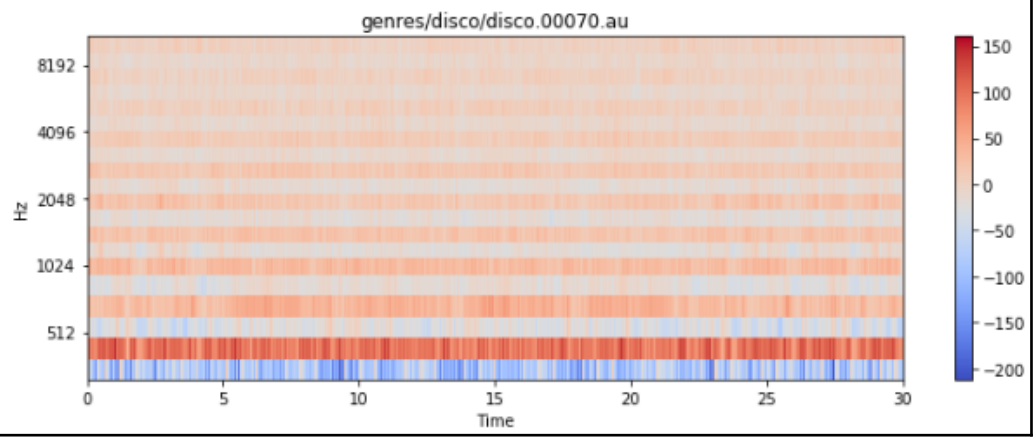

```
In [4]: display_mfcc('whistling.wav')
```



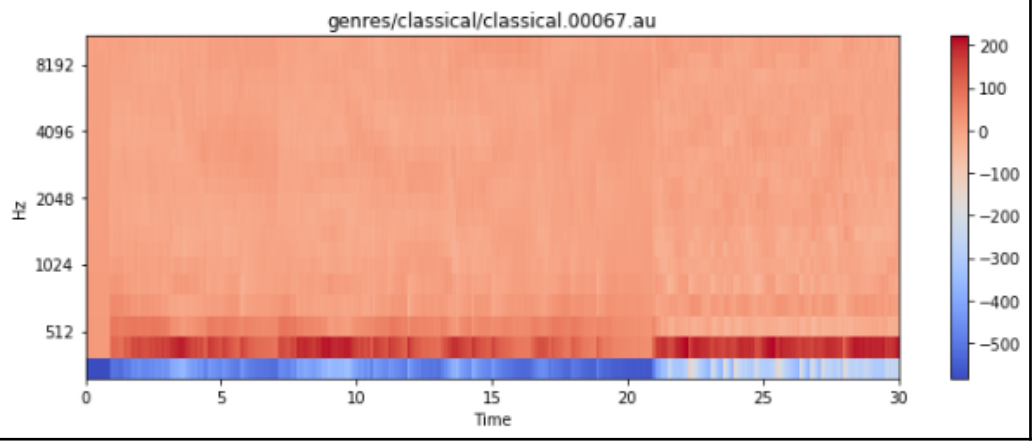
```
In [5]: display_mfcc('genres/disco/disco.00035.au')
```



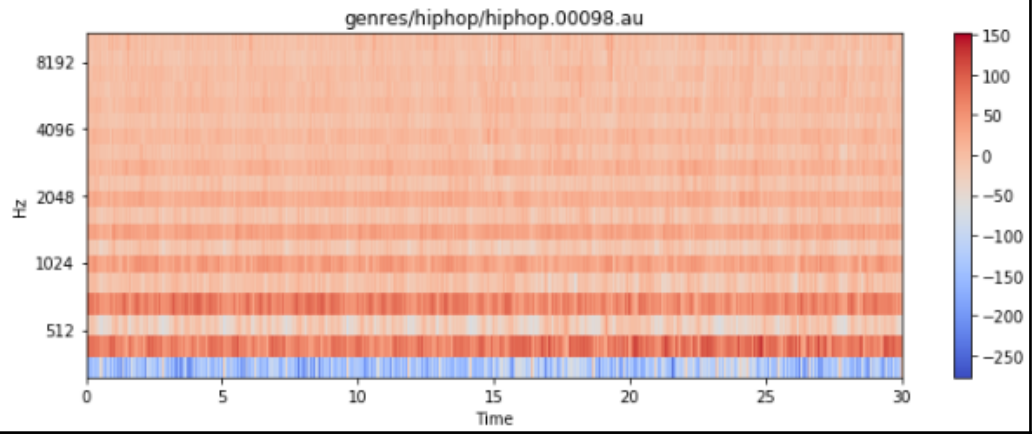

```
In [6]: display_mfcc('genres/disco/disco.00070.au')
```



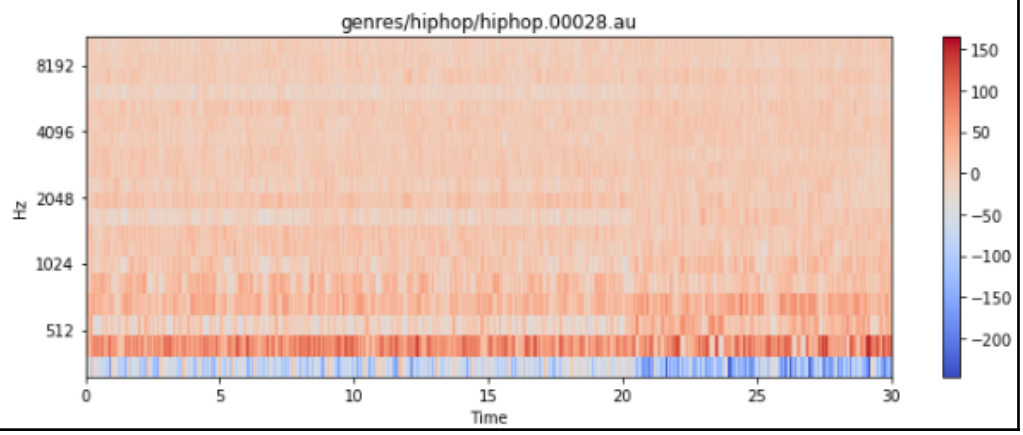
```
In [7]: display_mfcc('genres/classical/classical.00067.au')
```




```
In [9]: display_mfcc('genres/hiphop/hiphop.00098.au')
```



```
In [10]: display_mfcc('genres/hiphop/hiphop.00028.au')
```




```
In [11]: def extract_features_song(f):
             y, _ = librosa.load(f)

             # get Mel-frequency cepstral coefficients
             mfcc = librosa.feature.mfcc(y)
             # normalize values between -1,1 (divide by max)
             mfcc /= np.amax(np.absolute(mfcc))

             return np.ndarray.flatten(mfcc)[:25000]
```

```
In [12]: def generate_features_and_labels():
             all_features = []
             all_labels = []

             genres = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', 'pop', 'reggae', 'rock']
             for genre in genres:
                 sound_files = glob.glob('genres/'+genre+'/*.au')
                 print('Processing %d songs in %s genre...' % (len(sound_files), genre))
                 for f in sound_files:
                     features = extract_features_song(f)
                     all_features.append(features)
                     all_labels.append(genre)

             # convert labels to one-hot encoding
             label_uniq_ids, label_row_ids = np.unique(all_labels, return_inverse=True)
             label_row_ids = label_row_ids.astype(np.int32, copy=False)
             onehot_labels = to_categorical(label_row_ids, len(label_uniq_ids))
             return np.stack(all_features), onehot_labels
```

```
In [13]: features, labels = generate_features_and_labels()

         Processing 100 songs in blues genre...
         Processing 100 songs in classical genre...
         Processing 100 songs in country genre...
         Processing 100 songs in disco genre...
         Processing 100 songs in hiphop genre...
         Processing 100 songs in jazz genre...
         Processing 100 songs in metal genre...
         Processing 100 songs in pop genre...
         Processing 100 songs in reggae genre...
         Processing 100 songs in rock genre...
```

```
In [14]: print(np.shape(features))
         print(np.shape(labels))

         training_split = 0.8

         # last column has genre, turn it into unique ids
         alldata = np.column_stack((features, labels))

         np.random.shuffle(alldata)
         splitidx = int(len(alldata) * training_split)
         train, test = alldata[:splitidx,:], alldata[splitidx:,:]

         print(np.shape(train))
         print(np.shape(test))

         train_input = train[:,:-10]
         train_labels = train[:,-10:]

         test_input = test[:,:-10]
         test_labels = test[:,-10:]

         print(np.shape(train_input))
         print(np.shape(train_labels))

         (1000, 25000)
         (1000, 10)
         (800, 25010)
         (200, 25010)
         (800, 25000)
         (800, 10)
```

```
In [15]: model = Sequential([
             Dense(100, input_dim=np.shape(train_input)[1]),
             Activation('relu'),
             Dense(10),
             Activation('softmax'),
             ])

         model.compile(optimizer='adam',
                       loss='categorical_crossentropy',
                       metrics=['accuracy'])
         print(model.summary())

         model.fit(train_input, train_labels, epochs=10, batch_size=32,
                   validation_split=0.2)

         loss, acc = model.evaluate(test_input, test_labels, batch_size=32)

         print("Done!")
         print("Loss: %.4f, accuracy: %.4f" % (loss, acc))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 100)               2500100
_________________________________________________________________
activation_1 (Activation)    (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
=================================================================
Total params: 2,501,110
Trainable params: 2,501,110
Non-trainable params: 0
_________________________________________________________________
None
```

```
Train on 640 samples, validate on 160 samples
Epoch 1/10
640/640 [==============================] - 2s 3ms/step - loss: 2.0585 - acc: 0.2906 - val_loss: 1.7780 - val_acc: 0.3187
Epoch 2/10
640/640 [==============================] - 0s 720us/step - loss: 1.4080 - acc: 0.5031 - val_loss: 1.5680 - val_acc: 0.4562
Epoch 3/10
640/640 [==============================] - 0s 723us/step - loss: 1.1128 - acc: 0.6281 - val_loss: 1.5202 - val_acc: 0.4625
Epoch 4/10
640/640 [==============================] - 0s 697us/step - loss: 0.8968 - acc: 0.7422 - val_loss: 1.4163 - val_acc: 0.5062
Epoch 5/10
640/640 [==============================] - 0s 707us/step - loss: 0.7990 - acc: 0.7734 - val_loss: 1.9091 - val_acc: 0.4625
Epoch 6/10
640/640 [==============================] - 0s 712us/step - loss: 0.6336 - acc: 0.8266 - val_loss: 1.4158 - val_acc: 0.5375
Epoch 7/10
640/640 [==============================] - 0s 690us/step - loss: 0.4935 - acc: 0.9031 - val_loss: 1.4425 - val_acc: 0.5188
Epoch 8/10
640/640 [==============================] - 0s 717us/step - loss: 0.3547 - acc: 0.9500 - val_loss: 1.4821 - val_acc: 0.4813
Epoch 9/10
640/640 [==============================] - 0s 710us/step - loss: 0.2855 - acc: 0.9672 - val_loss: 1.4005 - val_acc: 0.5312
Epoch 10/10
640/640 [==============================] - 0s 706us/step - loss: 0.2247 - acc: 0.9891 - val_loss: 1.4223 - val_acc: 0.4938
200/200 [==============================] - 0s 452us/step
Done!
Loss: 1.5206, accuracy: 0.5250
```

```
In [28]: import pandas as pd
         from keras.preprocessing.text import Tokenizer
         import numpy as np
         from keras.models import Sequential
         from keras.layers import Dense, Dropout, Activation
         from keras.utils import np_utils
         from sklearn.model_selection import StratifiedKFold
```

```
In [29]: d = pd.concat([pd.read_csv("Youtube01-Psy.csv"),
                        pd.read_csv("Youtube02-KatyPerry.csv"),
                        pd.read_csv("Youtube03-LMFAO.csv"),
                        pd.read_csv("Youtube04-Eminem.csv"),
                        pd.read_csv("Youtube05-Shakira.csv")])
         d = d.sample(frac=1)
```

```
In [30]: kfold = StratifiedKFold(n_splits=5)
         splits = kfold.split(d, d['CLASS'])
```

```
In [31]: for train, test in splits:
             print("Split")
             print(test)
```

```
Split
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251
 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269
 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287
 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305
 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323
```

```
Split
[385 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409
 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427
 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445
 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463
 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481
 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499
 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517
 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535
 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553
 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571
 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589
 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607
 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625
 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643
 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661
 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679
 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697
 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715
```

```
Split
[ 757  759  761  763  765  771  772  773  776  777  781  783  784  787  789
  791  793  794  796  798  799  801  802  803  804  807  808  810  811  812
  813  814  815  816  817  818  819  820  821  822  823  824  825  826  827
  828  829  830  831  832  833  834  835  836  837  838  839  840  841  842
  843  844  845  846  847  848  849  850  851  852  853  854  855  856  857
  858  859  860  861  862  863  864  865  866  867  868  869  870  871  872
  873  874  875  876  877  878  879  880  881  882  883  884  885  886  887
  888  889  890  891  892  893  894  895  896  897  898  899  900  901  902
  903  904  905  906  907  908  909  910  911  912  913  914  915  916  917
  918  919  920  921  922  923  924  925  926  927  928  929  930  931  932
  933  934  935  936  937  938  939  940  941  942  943  944  945  946  947
  948  949  950  951  952  953  954  955  956  957  958  959  960  961  962
  963  964  965  966  967  968  969  970  971  972  973  974  975  976  977
  978  979  980  981  982  983  984  985  986  987  988  989  990  991  992
  993  994  995  996  997  998  999 1000 1001 1002 1003 1004 1005 1006 1007
 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022
 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037
```

```
Split
[1136 1139 1140 1143 1144 1147 1148 1150 1151 1153 1158 1160 1166 1167 1169
 1171 1172 1176 1180 1181 1184 1187 1190 1192 1194 1198 1200 1201 1202 1203
 1204 1205 1206 1207 1208 1209 1210 1211 1212 1213 1214 1215 1216 1217 1218
 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1229 1230 1231 1232 1233
 1234 1235 1236 1237 1238 1239 1240 1241 1242 1243 1244 1245 1246 1247 1248
 1249 1250 1251 1252 1253 1254 1255 1256 1257 1258 1259 1260 1261 1262 1263
 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278
 1279 1280 1281 1282 1283 1284 1285 1286 1287 1288 1289 1290 1291 1292 1293
 1294 1295 1296 1297 1298 1299 1300 1301 1302 1303 1304 1305 1306 1307 1308
 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323
 1324 1325 1326 1327 1328 1329 1330 1331 1332 1333 1334 1335 1336 1337 1338
 1339 1340 1341 1342 1343 1344 1345 1346 1347 1348 1349 1350 1351 1352 1353
 1354 1355 1356 1357 1358 1359 1360 1361 1362 1363 1364 1365 1366 1367 1368
 1369 1370 1371 1372 1373 1374 1375 1376 1377 1378 1379 1380 1381 1382 1383
 1384 1385 1386 1387 1388 1389 1390 1391 1392 1393 1394 1395 1396 1397 1398
 1399 1400 1401 1402 1403 1404 1405 1406 1407 1408 1409 1410 1411 1412 1413
```

```
Split
[1548 1551 1552 1555 1561 1566 1569 1572 1573 1574 1575 1576 1577 1578 1579
 1580 1581 1582 1583 1584 1585 1586 1587 1588 1589 1590 1591 1592 1593 1594
 1595 1596 1597 1598 1599 1600 1601 1602 1603 1604 1605 1606 1607 1608 1609
 1610 1611 1612 1613 1614 1615 1616 1617 1618 1619 1620 1621 1622 1623 1624
 1625 1626 1627 1628 1629 1630 1631 1632 1633 1634 1635 1636 1637 1638 1639
 1640 1641 1642 1643 1644 1645 1646 1647 1648 1649 1650 1651 1652 1653 1654
 1655 1656 1657 1658 1659 1660 1661 1662 1663 1664 1665 1666 1667 1668 1669
 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684
 1685 1686 1687 1688 1689 1690 1691 1692 1693 1694 1695 1696 1697 1698 1699
 1700 1701 1702 1703 1704 1705 1706 1707 1708 1709 1710 1711 1712 1713 1714
 1715 1716 1717 1718 1719 1720 1721 1722 1723 1724 1725 1726 1727 1728 1729
 1730 1731 1732 1733 1734 1735 1736 1737 1738 1739 1740 1741 1742 1743 1744
 1745 1746 1747 1748 1749 1750 1751 1752 1753 1754 1755 1756 1757 1758 1759
 1760 1761 1762 1763 1764 1765 1766 1767 1768 1769 1770 1771 1772 1773 1774
 1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1789
```

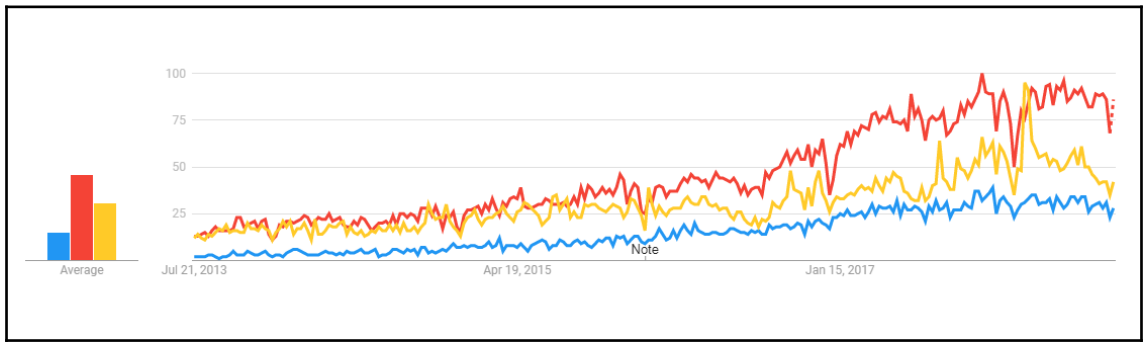
In [41]:
```python
def train_and_test(train_idx, test_idx):

    train_content = d['CONTENT'].iloc[train_idx]
    test_content = d['CONTENT'].iloc[test_idx]

    tokenizer = Tokenizer(num_words=2000)

    # learn the training words (not the testing words!)
    tokenizer.fit_on_texts(train_content)

    # options for mode: binary, freq, tfidf
    d_train_inputs = tokenizer.texts_to_matrix(train_content, mode='tfidf')
    d_test_inputs = tokenizer.texts_to_matrix(test_content, mode='tfidf')

    # divide tfidf by max
    d_train_inputs = d_train_inputs/np.amax(np.absolute(d_train_inputs))
    d_test_inputs = d_test_inputs/np.amax(np.absolute(d_test_inputs))

    # subtract mean, to get values between -1 and 1
    d_train_inputs = d_train_inputs - np.mean(d_train_inputs)
    d_test_inputs = d_test_inputs - np.mean(d_test_inputs)
```

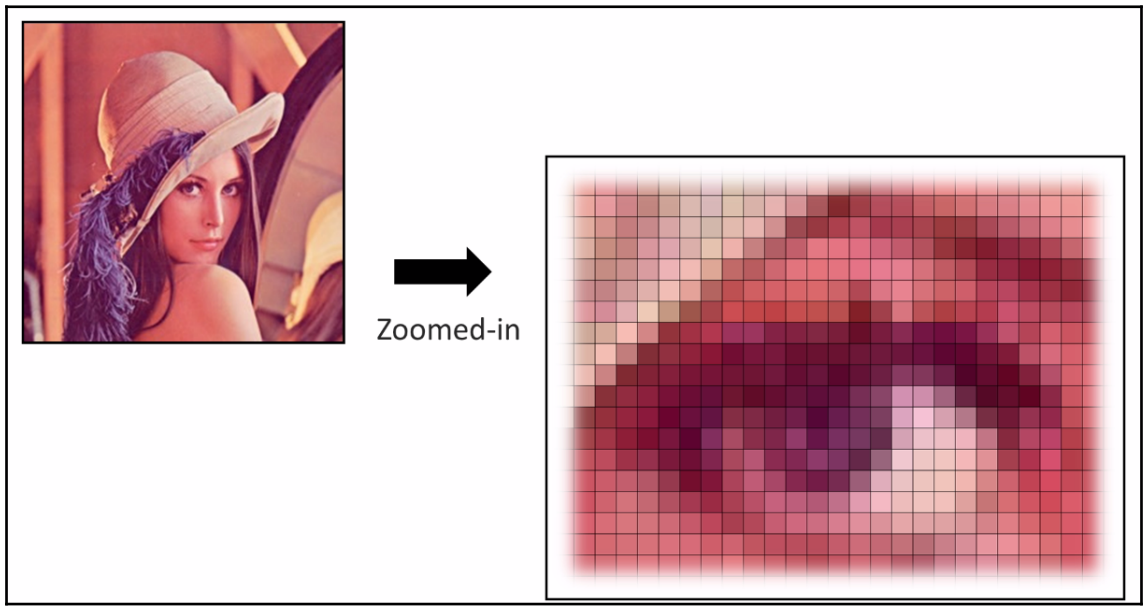
```
In [42]: kfold = StratifiedKFold(n_splits=5)
         splits = kfold.split(d, d['CLASS'])
         cvscores = []
         for train_idx, test_idx, in splits:
             scores = train_and_test(train_idx, test_idx)
             cvscores.append(scores[1] * 100)
```

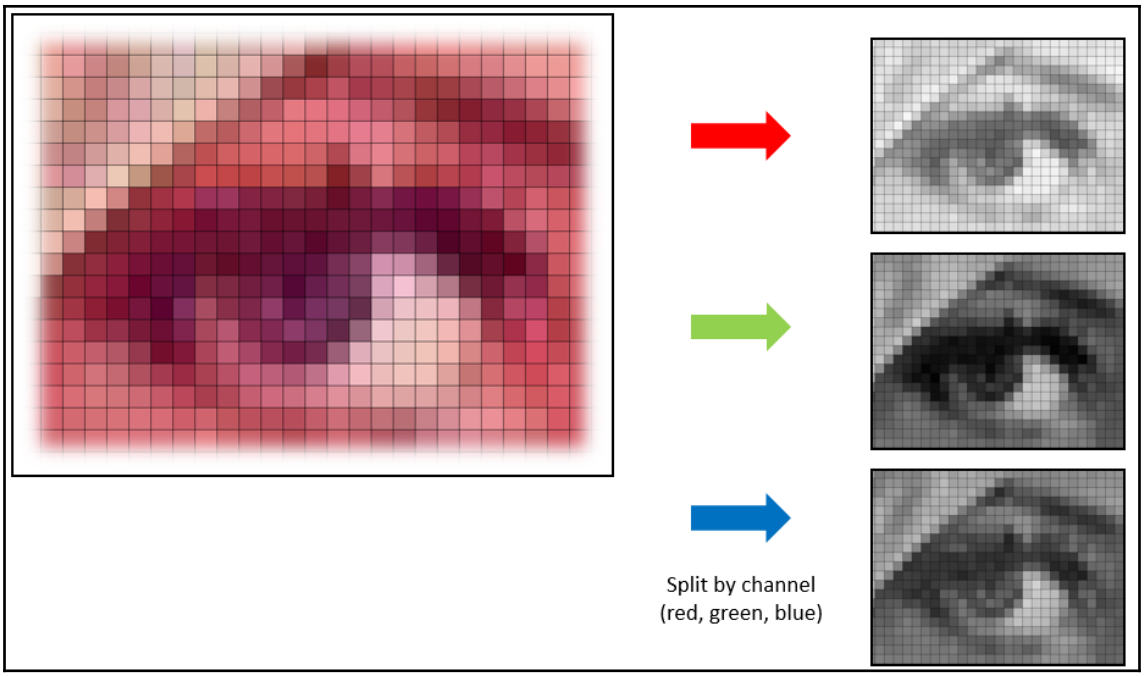
```
Epoch 1/10
1564/1564 [==============================] - 3s 2ms/step - loss: 0.5992 - acc: 0.7986
Epoch 2/10
1564/1564 [==============================] - 1s 582us/step - loss: 0.3709 - acc: 0.9137
Epoch 3/10
1564/1564 [==============================] - 1s 582us/step - loss: 0.2382 - acc: 0.9425
Epoch 4/10
1564/1564 [==============================] - 1s 581us/step - loss: 0.1761 - acc: 0.9520
Epoch 5/10
1564/1564 [==============================] - 1s 575us/step - loss: 0.1457 - acc: 0.9584
```

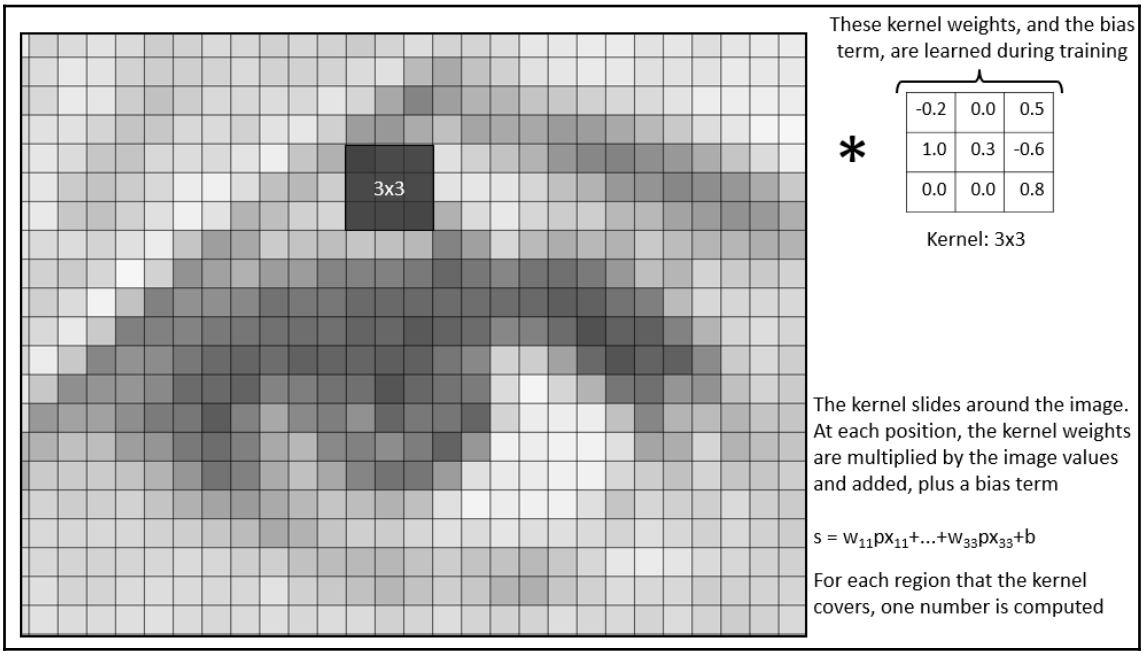
```
In [43]: print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
         95.09% (+/- 1.72%)
```

Chapter 5: Deep Learning





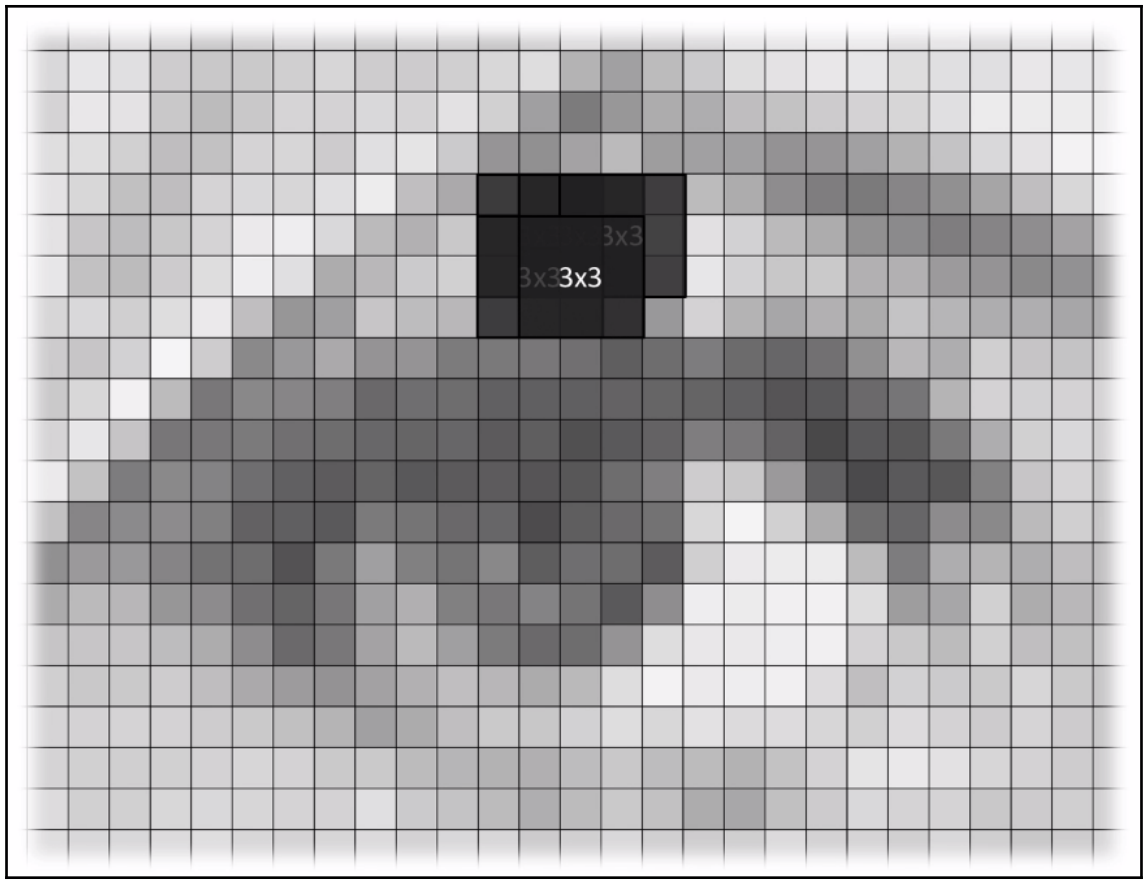

Split by channel
(red, green, blue)

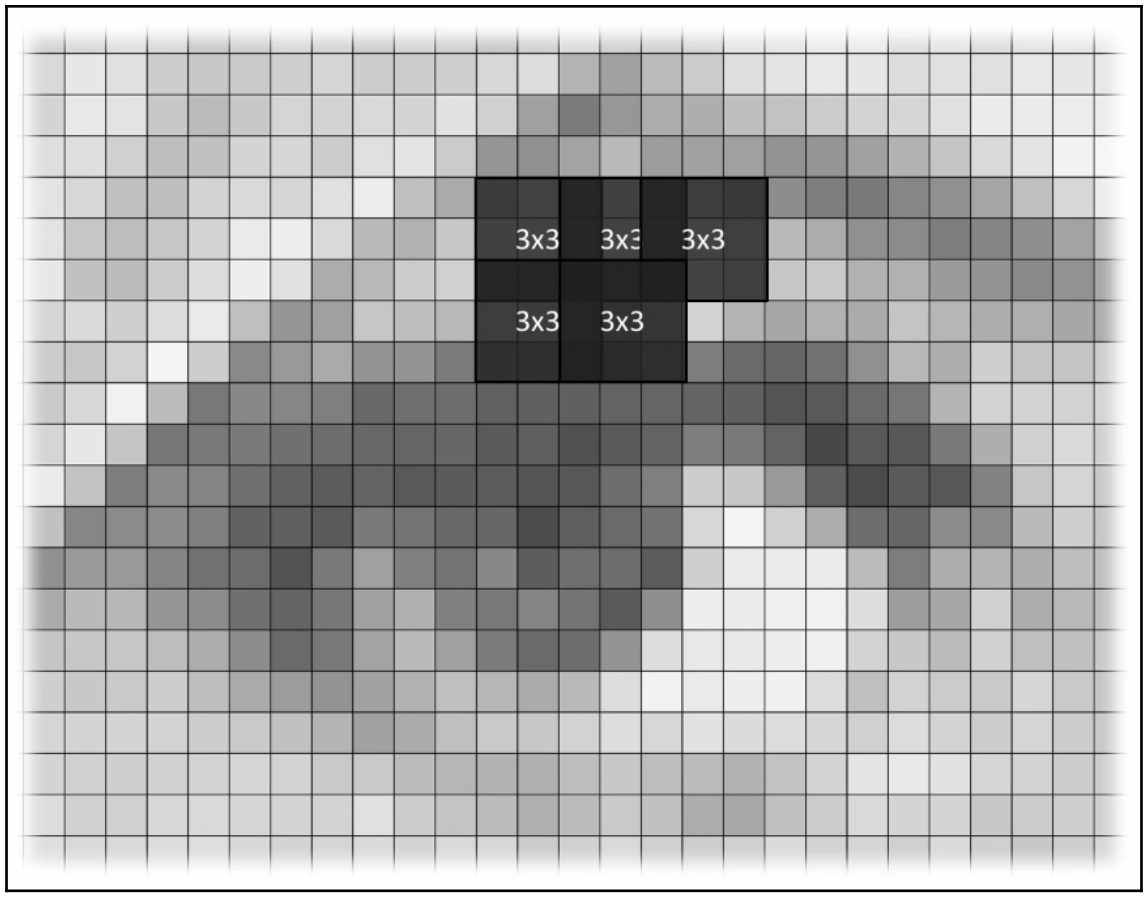

3x3
These kernel weights, and the bias term, are learned during training
-0.2 0.0 0.5
1.0 0.3 -0.6
0.0 0.0 0.8
Kernel: 3x3
The kernel slides around the image. At each position, the kernel weights are multiplied by the image values and added, plus a bias term
$s = w_{11}px_{11}+...+w_{33}px_{33}+b$
For each region that the kernel covers, one number is computed

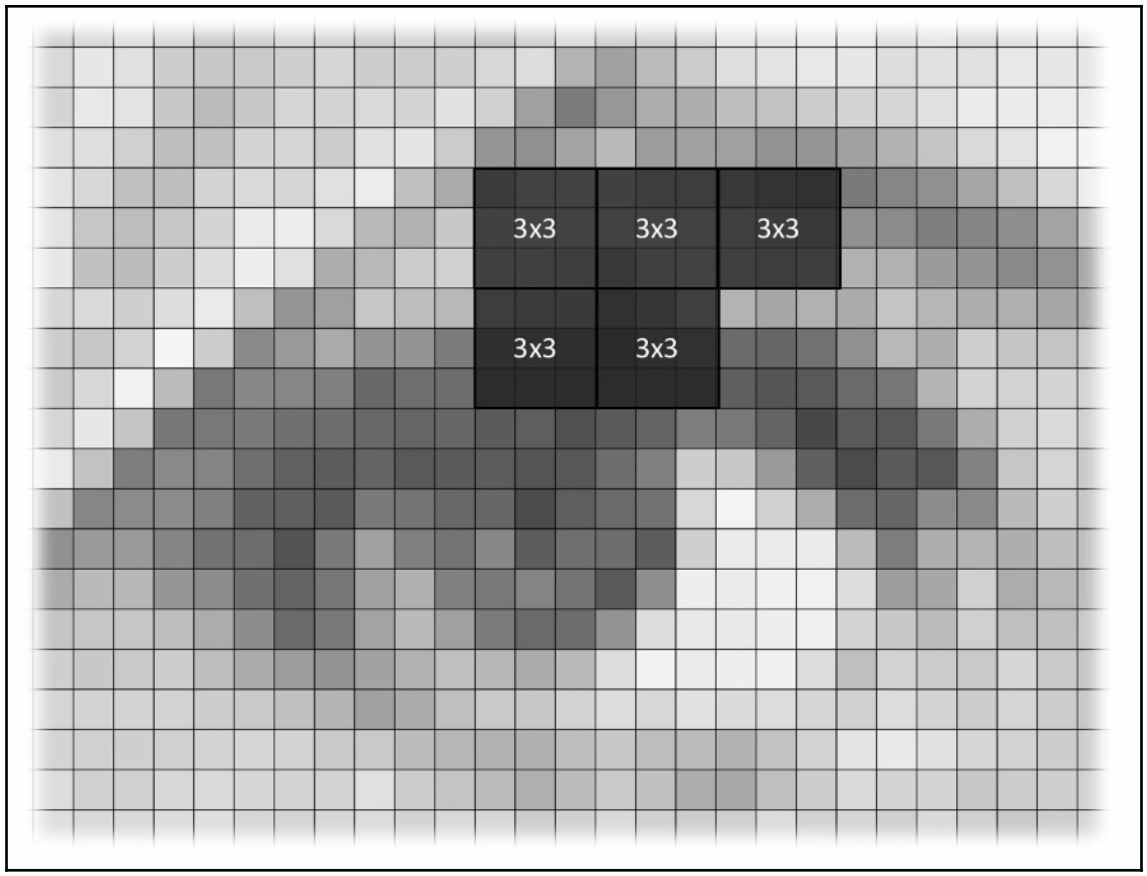
-0.2	0.0	0.5
1.0	0.3	-0.6
0.0	0.0	0.8

0.0	0.0	0.0
0.8	-0.5	0.8
0.0	-0.2	0.0

0.4	0.2	-0.2
-0.8	0.0	0.8
0.0	-0.5	0.2


3x3
3x3
3x3

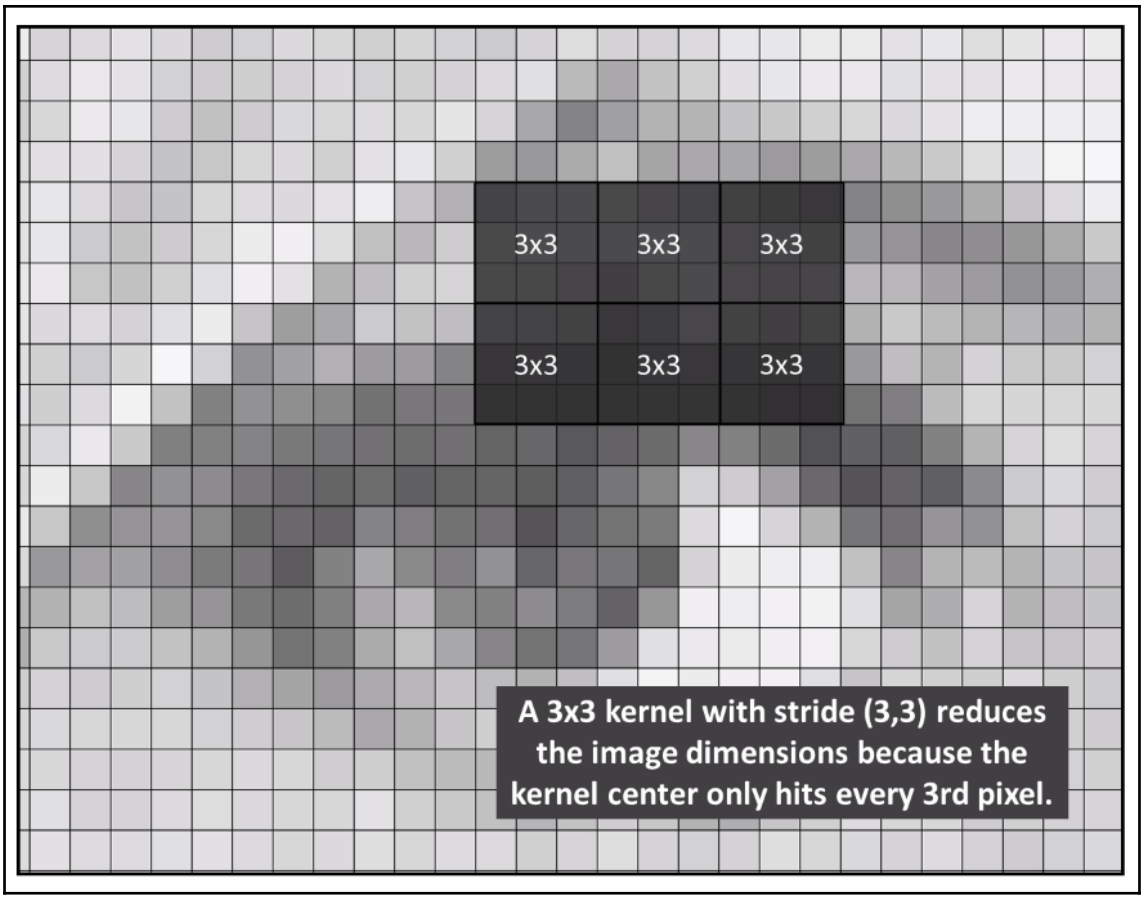

3x3
3x3
3x3
3x3
3x3

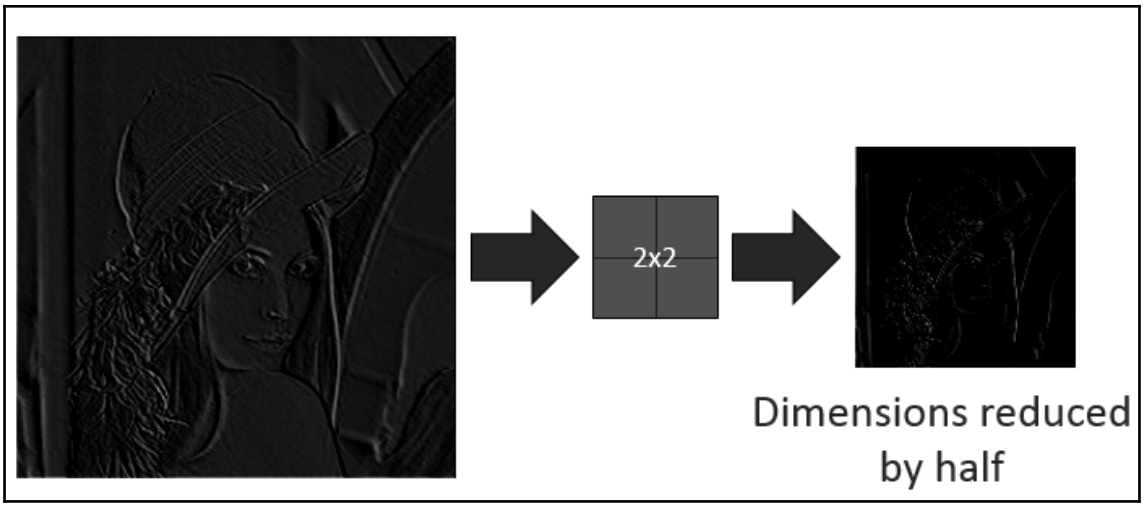

3x3
3x3
3x3
3x3
3x3

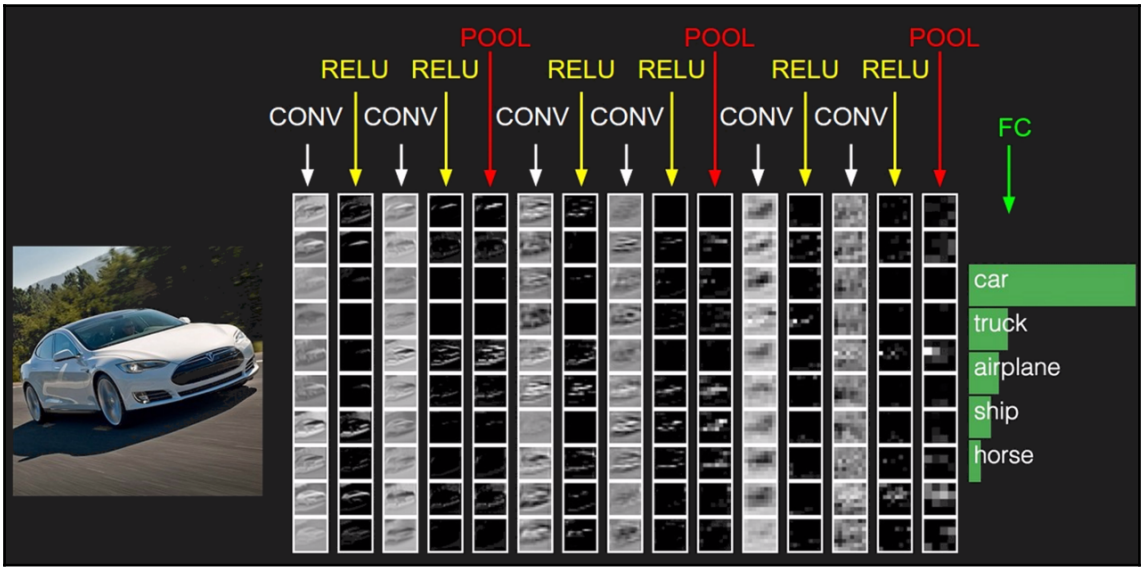
$D = 1 + (W - K + 2*P)/S,$

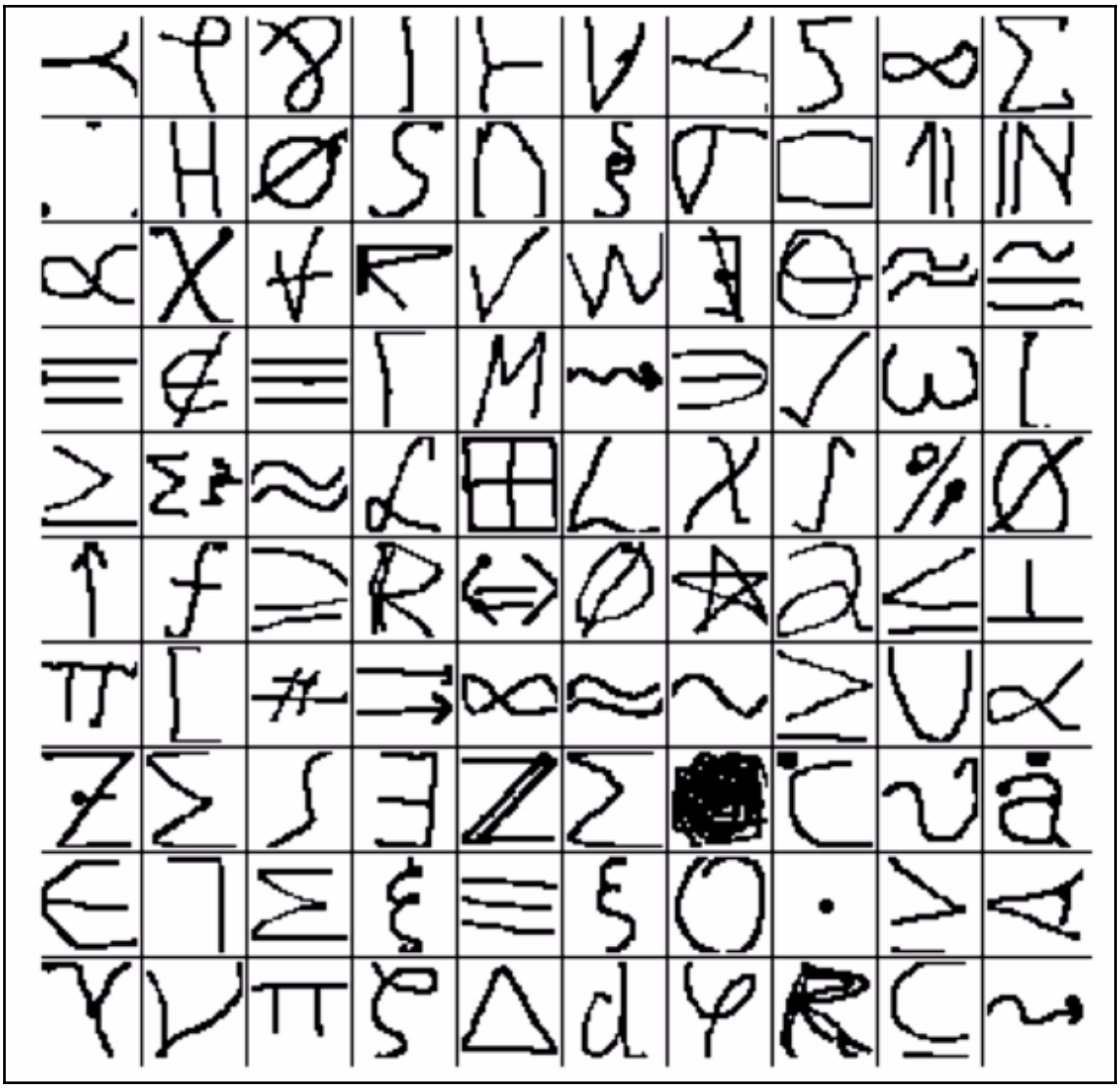
W = input width, K = kernel size,
P = padding size, S = stride size

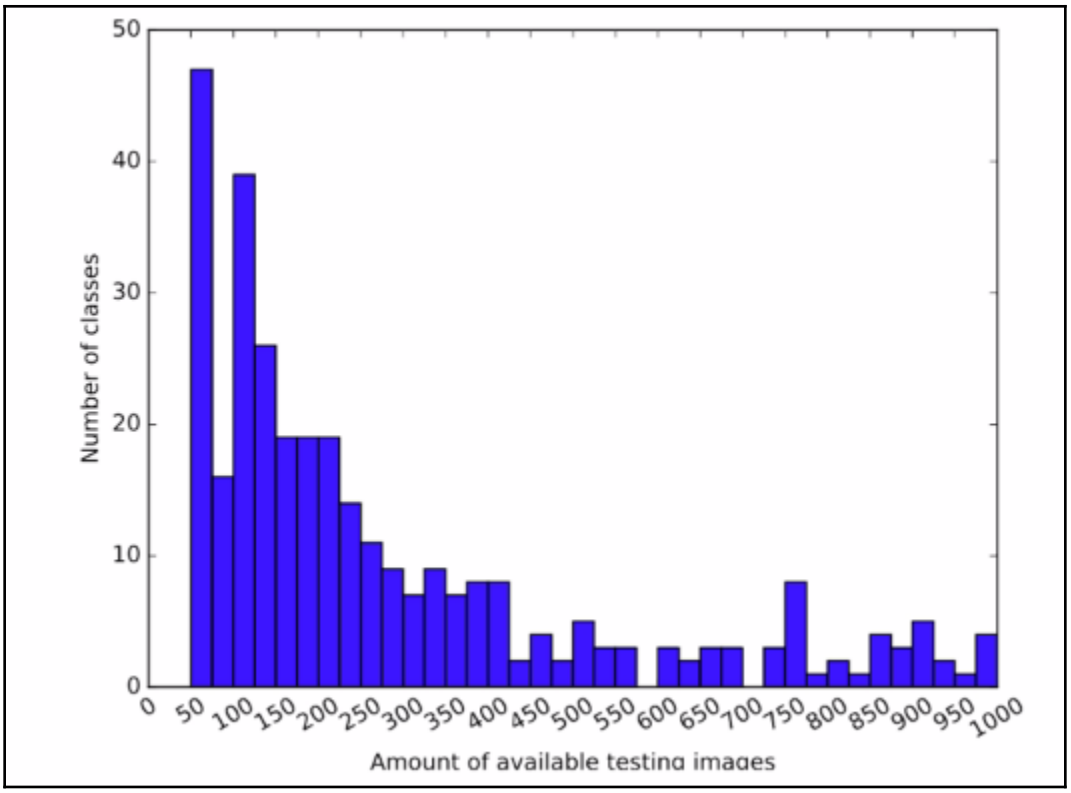
For example, if W = 256, K = 3,
P = 1, and S = 3, then the output
dimension is 85x85


3x3
3x3
3x3
3x3
3x3
3x3
A 3x3 kernel with stride (3,3) reduces the image dimensions because the kernel center only hits every 3rd pixel.


2x2
Dimensions reduced by half


POOL
RELU
RELU
CONV
CONV
FC
car
truck
airplane
ship
horse






```
In [1]: from IPython.display import Image
```

```
In [2]: # example image from the dataset

        Image(url="HASYv2/hasy-data/v2-00010.png")
```

Out[2]: 

```
In [3]: import csv
        from PIL import Image as pil_image
        import keras.preprocessing.image

        Using TensorFlow backend.
```

```
In [4]: # load all images (as numpy arrays) and save their classes

        imgs = []
        classes = []
        with open('HASYv2/hasy-data-labels.csv') as csvfile:
            csvreader = csv.reader(csvfile)
            i = 0
            for row in csvreader:
                if i > 0:
                    img = keras.preprocessing.image.img_to_array(pil_image.open("HASYv2/" + row[0]))
                    # neuron activation functions behave best when input values are between 0.0 and 1.0 (or -1.0 and 1.0),
                    # so we rescale each pixel value to be in the range 0.0 to 1.0 instead of 0-255
                    img /= 255.0
                    imgs.append((row[0], row[2], img))
                    classes.append(row[2])
                i += 1
```

```
In [5]: imgs[0]

Out[5]: ('hasy-data/v2-00000.png', 'A', array([[[ 1.,  1.,  1.],
                [ 1.,  1.,  1.],
                [ 1.,  1.,  1.],
                ...,
                [ 1.,  1.,  1.],
                [ 1.,  1.,  1.],
                [ 1.,  1.,  1.]],

               [[ 1.,  1.,  1.],
                [ 1.,  1.,  1.],
                [ 1.,  1.,  1.],
                ...,
                [ 1.,  1.,  1.],
                [ 1.,  1.,  1.],
                [ 1.,  1.,  1.]],

               [[ 1.,  1.,  1.],
                [ 1.,  1.,  1.],
                [ 1.,  1.,  1.],
                ...,
                [ 1.,  1.,  1.],
                [ 1.,  1.,  1.],
                [ 1.,  1.,  1.]],
```

```
In [6]: len(imgs)

Out[6]: 168233
```

In [7]:
```
# shuffle the data, split into 80% train, 20% test

import random
random.shuffle(imgs)
split_idx = int(0.8*len(imgs))
train = imgs[:split_idx]
test = imgs[split_idx:]
```

In [8]:
```
import numpy as np

train_input = np.asarray(list(map(lambda row: row[2], train)))
test_input = np.asarray(list(map(lambda row: row[2], test)))

train_output = np.asarray(list(map(lambda row: row[1], train)))
test_output = np.asarray(list(map(lambda row: row[1], test)))
```

In [9]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
```

In [10]:
```
# convert class names into one-hot encoding

# first, convert class names into integers
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(classes)

# then convert integers into one-hot encoding
onehot_encoder = OneHotEncoder(sparse=False)
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
onehot_encoder.fit(integer_encoded)

# convert train and test output to one-hot
train_output_int = label_encoder.transform(train_output)
train_output = onehot_encoder.transform(train_output_int.reshape(len(train_output_int), 1))
test_output_int = label_encoder.transform(test_output)
test_output = onehot_encoder.transform(test_output_int.reshape(len(test_output_int), 1))

num_classes = len(label_encoder.classes_)
print("Number of classes: %d" % num_classes)
```

```
Number of classes: 369
```

In [11]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
```

In [12]:
```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
                 input_shape=np.shape(train_input[0])))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(1024, activation='tanh'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

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

print(model.summary())
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_1 (MaxPooling2	(None, 15, 15, 32)	0
conv2d_2 (Conv2D)	(None, 13, 13, 32)	9248
max_pooling2d_2 (MaxPooling2	(None, 6, 6, 32)	0
flatten_1 (Flatten)	(None, 1152)	0
dense_1 (Dense)	(None, 1024)	1180672
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 369)	378225

Total params: 1,569,041
Trainable params: 1,569,041
Non-trainable params: 0

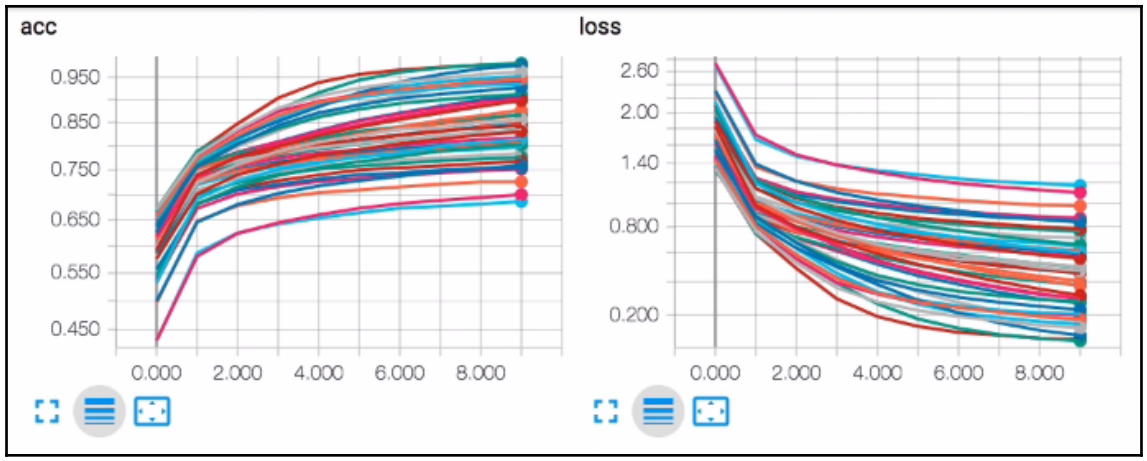
In [13]:
```
import keras.callbacks
tensorboard = keras.callbacks.TensorBoard(log_dir='./logs/mnist-style')
```

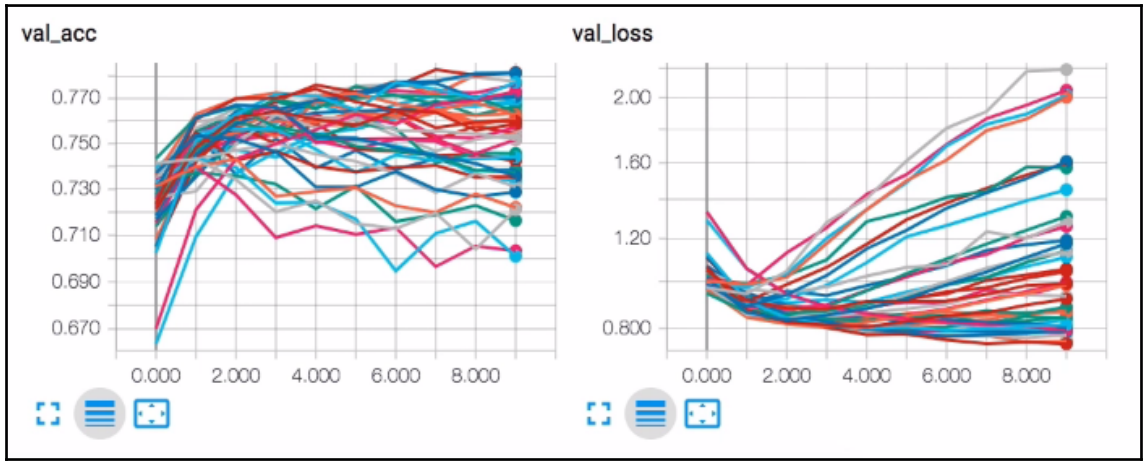
In [14]:
```
model.fit(train_input, train_output,
          batch_size=32,
          epochs=10,
          verbose=2,
          validation_split=0.2,
          callbacks=[tensorboard])
```

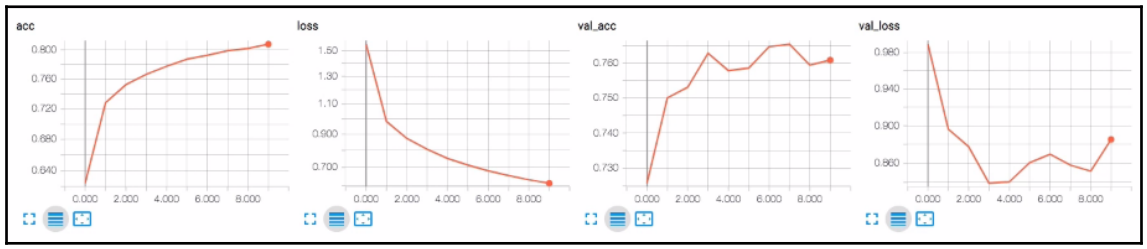
```
Train on 107668 samples, validate on 26918 samples
Epoch 1/10
 - 54s - loss: 1.5568 - acc: 0.6243 - val_loss: 0.9898 - val_acc: 0.7257
Epoch 2/10
 - 52s - loss: 0.9820 - acc: 0.7281 - val_loss: 0.8964 - val_acc: 0.7501
Epoch 3/10
 - 52s - loss: 0.8730 - acc: 0.7523 - val_loss: 0.8776 - val_acc: 0.7531
Epoch 4/10
 - 52s - loss: 0.8067 - acc: 0.7662 - val_loss: 0.8391 - val_acc: 0.7629
Epoch 5/10
 - 52s - loss: 0.7520 - acc: 0.7771 - val_loss: 0.8406 - val_acc: 0.7579
Epoch 6/10
 - 52s - loss: 0.7137 - acc: 0.7868 - val_loss: 0.8607 - val_acc: 0.7586
Epoch 7/10
 - 52s - loss: 0.6812 - acc: 0.7922 - val_loss: 0.8696 - val_acc: 0.7648
Epoch 8/10
 - 52s - loss: 0.6544 - acc: 0.7984 - val_loss: 0.8581 - val_acc: 0.7655
Epoch 9/10
 - 52s - loss: 0.6312 - acc: 0.8015 - val_loss: 0.8518 - val_acc: 0.7595
Epoch 10/10
 - 52s - loss: 0.6125 - acc: 0.8076 - val_loss: 0.8854 - val_acc: 0.7609
```

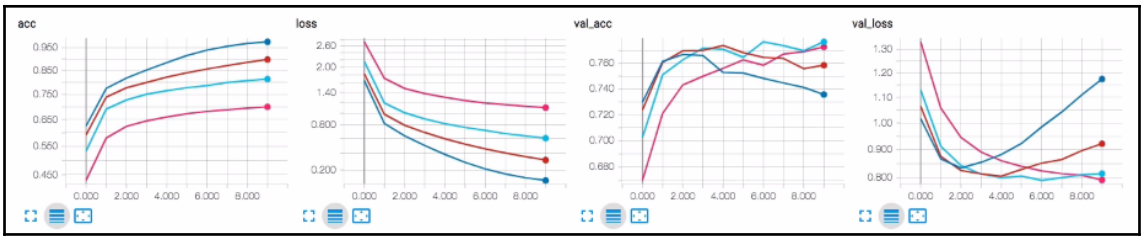
```
In [17]: # try various model configurations and parameters to find the best

         import time

         results = []
         for conv2d_count in [1, 2]:
             for dense_size in [128, 256, 512, 1024, 2048]:
                 for dropout in [0.0, 0.25, 0.50, 0.75]:
                     model = Sequential()
                     for i in range(conv2d_count):
                         if i == 0:
                             model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=np.shape(train_input[0])))
                         else:
                             model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
                         model.add(MaxPooling2D(pool_size=(2, 2)))
                     model.add(Flatten())
                     model.add(Dense(dense_size, activation='tanh'))
                     if dropout > 0.0:
                         model.add(Dropout(dropout))
                     model.add(Dense(num_classes, activation='softmax'))

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

                     log_dir = './logs/conv2d_%d-dense_%d-dropout_%.2f' % (conv2d_count, dense_size, dropout)
                     tensorboard = keras.callbacks.TensorBoard(log_dir=log_dir)

                     start = time.time()
                     model.fit(train_input, train_output, batch_size=32, epochs=10,
                               verbose=0, validation_split=0.2, callbacks=[tensorboard])
                     score = model.evaluate(test_input, test_output, verbose=2)
                     end = time.time()
```

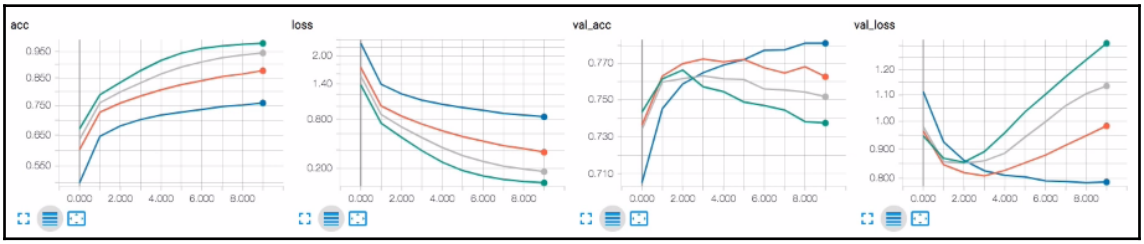
```
Conv2D count: 1, Dense size: 128, Dropout: 0.00 - Loss: 1.16, Accuracy: 0.74, Time: 419 sec
Conv2D count: 1, Dense size: 128, Dropout: 0.25 - Loss: 0.92, Accuracy: 0.76, Time: 447 sec
Conv2D count: 1, Dense size: 128, Dropout: 0.50 - Loss: 0.82, Accuracy: 0.77, Time: 452 sec
Conv2D count: 1, Dense size: 128, Dropout: 0.75 - Loss: 0.79, Accuracy: 0.77, Time: 458 sec
Conv2D count: 1, Dense size: 256, Dropout: 0.00 - Loss: 1.30, Accuracy: 0.74, Time: 430 sec
Conv2D count: 1, Dense size: 256, Dropout: 0.25 - Loss: 1.12, Accuracy: 0.76, Time: 459 sec
Conv2D count: 1, Dense size: 256, Dropout: 0.50 - Loss: 0.96, Accuracy: 0.77, Time: 461 sec
Conv2D count: 1, Dense size: 256, Dropout: 0.75 - Loss: 0.78, Accuracy: 0.78, Time: 461 sec
Conv2D count: 1, Dense size: 512, Dropout: 0.00 - Loss: 1.60, Accuracy: 0.74, Time: 440 sec
Conv2D count: 1, Dense size: 512, Dropout: 0.25 - Loss: 1.43, Accuracy: 0.75, Time: 466 sec
Conv2D count: 1, Dense size: 512, Dropout: 0.50 - Loss: 1.24, Accuracy: 0.75, Time: 471 sec
Conv2D count: 1, Dense size: 512, Dropout: 0.75 - Loss: 0.87, Accuracy: 0.77, Time: 475 sec
Conv2D count: 1, Dense size: 1024, Dropout: 0.00 - Loss: 2.13, Accuracy: 0.72, Time: 480 sec
Conv2D count: 1, Dense size: 1024, Dropout: 0.25 - Loss: 1.94, Accuracy: 0.73, Time: 517 sec
Conv2D count: 1, Dense size: 1024, Dropout: 0.50 - Loss: 1.59, Accuracy: 0.73, Time: 526 sec
Conv2D count: 1, Dense size: 1024, Dropout: 0.75 - Loss: 0.98, Accuracy: 0.76, Time: 527 sec
Conv2D count: 1, Dense size: 2048, Dropout: 0.00 - Loss: 2.00, Accuracy: 0.70, Time: 587 sec
Conv2D count: 1, Dense size: 2048, Dropout: 0.25 - Loss: 2.02, Accuracy: 0.70, Time: 629 sec
Conv2D count: 1, Dense size: 2048, Dropout: 0.50 - Loss: 1.55, Accuracy: 0.72, Time: 631 sec
Conv2D count: 1, Dense size: 2048, Dropout: 0.75 - Loss: 1.29, Accuracy: 0.73, Time: 636 sec
Conv2D count: 2, Dense size: 128, Dropout: 0.00 - Loss: 0.87, Accuracy: 0.76, Time: 531 sec
Conv2D count: 2, Dense size: 128, Dropout: 0.25 - Loss: 0.79, Accuracy: 0.77, Time: 570 sec
Conv2D count: 2, Dense size: 128, Dropout: 0.50 - Loss: 0.74, Accuracy: 0.78, Time: 568 sec
Conv2D count: 2, Dense size: 128, Dropout: 0.75 - Loss: 0.79, Accuracy: 0.77, Time: 573 sec
Conv2D count: 2, Dense size: 256, Dropout: 0.00 - Loss: 0.99, Accuracy: 0.76, Time: 550 sec
Conv2D count: 2, Dense size: 256, Dropout: 0.25 - Loss: 0.85, Accuracy: 0.77, Time: 583 sec
Conv2D count: 2, Dense size: 256, Dropout: 0.50 - Loss: 0.77, Accuracy: 0.78, Time: 579 sec
```

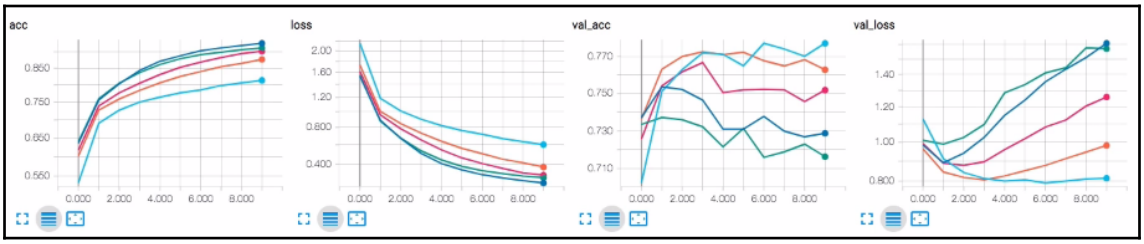

acc
0.950
0.850
0.750
0.650
0.550
0.450
0.000
2.000
4.000
6.000
8.000
loss
2.60
2.00
1.40
0.800
0.200
0.000
2.000
4.000
6.000
8.000

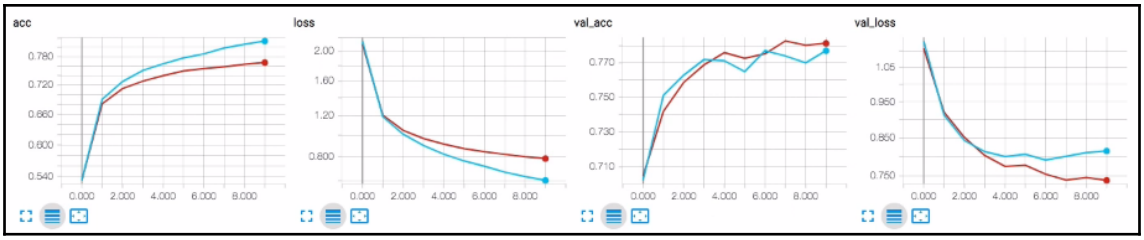

val_acc
0.770
0.750
0.730
0.710
0.690
0.670
0.000
2.000
4.000
6.000
8.000
val_loss
2.00
1.60
1.20
0.800
0.000
2.000
4.000
6.000
8.000


acc
loss
val_acc
val_loss


acc
loss
val_acc
val_loss


acc
loss
val_acc
val_loss


acc
loss
val_acc
val_loss


acc
loss
val_acc
val_loss

In [22]:
```
# rebuild/retrain a model with the best parameters (from the search) and use all data
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=np.shape(train_input[0])))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='tanh'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
# join train and test data so we train the network on all data we have available to us
model.fit(np.concatenate((train_input, test_input)),
          np.concatenate((train_output, test_output)),
          batch_size=32, epochs=10, verbose=2)

# save the trained model
model.save("mathsymbols.model")

# save label encoder (to reverse one-hot encoding)
np.save('classes.npy', label_encoder.classes_)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_68 (Conv2D)           (None, 30, 30, 32)        896
```

In [32]:
```
# load the pre-trained model and predict the math symbol for an arbitrary image;
# the code below could be placed in a separate file

import keras.models
model2 = keras.models.load_model("mathsymbols.model")
print(model2.summary())

# restore the class name to integer encoder
label_encoder2 = LabelEncoder()
label_encoder2.classes_ = np.load('classes.npy')

def predict(img_path):
    newimg = keras.preprocessing.image.img_to_array(pil_image.open(img_path))
    newimg /= 255.0

    # do the prediction
    prediction = model2.predict(newimg.reshape(1, 32, 32, 3))

    # figure out which output neuron had the highest score, and reverse the one-hot encoding
    inverted = label_encoder2.inverse_transform([np.argmax(prediction)]) # argmax finds highest-scoring output
    print("Prediction: %s, confidence: %.2f" % (inverted[0], np.max(prediction)))
```

Layer (type)	Output Shape	Param #
conv2d_68 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_68 (MaxPooling	(None, 15, 15, 32)	0
conv2d_69 (Conv2D)	(None, 13, 13, 32)	9248
max_pooling2d_69 (MaxPooling	(None, 6, 6, 32)	0
flatten_45 (Flatten)	(None, 1152)	0
dense_89 (Dense)	(None, 128)	147584
dropout_34 (Dropout)	(None, 128)	0
dense_90 (Dense)	(None, 369)	47601

Total params: 205,329

```
In [33]: # grab an image (we'll just use a random training image for demonstration purposes)
         predict("HASYv2/hasy-data/v2-00010.png")

         Prediction: A, confidence: 0.87

In [34]: predict("HASYv2/hasy-data/v2-00500.png")

         Prediction: \pi, confidence: 0.58

In [35]: predict("HASYv2/hasy-data/v2-00700.png")

         Prediction: \alpha, confidence: 0.88
```

```
In [1]: import numpy as np
        from keras.models import Sequential, load_model
        from keras.layers import Dropout, Flatten, Conv2D, MaxPooling2D, Dense, Activation
        from keras.utils import np_utils
        from keras.preprocessing.image import ImageDataGenerator
        from keras.callbacks import TensorBoard
        import itertools

        Using TensorFlow backend.
```

In [2]:
```
# all images will be converted to this size
ROWS = 256
COLS = 256
CHANNELS = 3
```

In [3]:
```
train_image_generator = ImageDataGenerator(horizontal_flip=True, rescale=1./255, rotation_range=45)
test_image_generator = ImageDataGenerator(horizontal_flip=False, rescale=1./255, rotation_range=0)

train_generator = train_image_generator.flow_from_directory('train', target_size=(ROWS, COLS), class_mode='categorical')
test_generator = test_image_generator.flow_from_directory('test', target_size=(ROWS, COLS), class_mode='categorical')
```

```
Found 5994 images belonging to 200 classes.
Found 5794 images belonging to 200 classes.
```

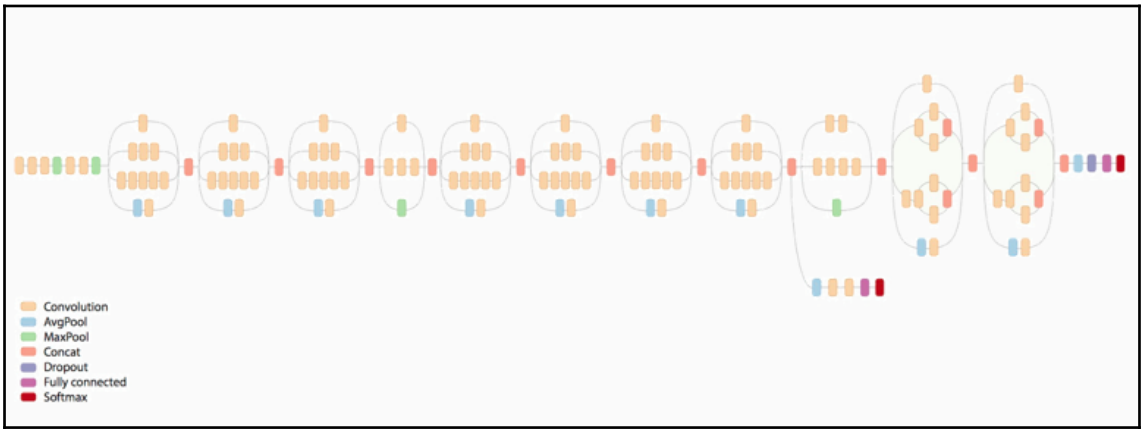
In [12]:
```
train_generator.reset()
test_generator.reset()

model = Sequential()
model.add(Conv2D(64, (3,3), input_shape=(ROWS, COLS, CHANNELS)))
model.add(Activation('relu'))
model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(4,4)))
model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(4,4)))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(400))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(200))
model.add(Activation('softmax'))

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

model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_19 (Conv2D)	(None, 254, 254, 64)	1792
activation_28 (Activation)	(None, 254, 254, 64)	0
conv2d_20 (Conv2D)	(None, 252, 252, 64)	36928
activation_29 (Activation)	(None, 252, 252, 64)	0
max_pooling2d_10 (MaxPooling	(None, 63, 63, 64)	0
conv2d_21 (Conv2D)	(None, 61, 61, 64)	36928
activation_30 (Activation)	(None, 61, 61, 64)	0
conv2d_22 (Conv2D)	(None, 59, 59, 64)	36928
activation_31 (Activation)	(None, 59, 59, 64)	0
max_pooling2d_11 (MaxPooling	(None, 14, 14, 64)	0
flatten_5 (Flatten)	(None, 12544)	0
dropout_8 (Dropout)	(None, 12544)	0
dense_9 (Dense)	(None, 400)	5018000
activation_32 (Activation)	(None, 400)	0
dropout_9 (Dropout)	(None, 400)	0
dense_10 (Dense)	(None, 200)	80200
activation_33 (Activation)	(None, 200)	0

Total params: 5,210,776
Trainable params: 5,210,776
Non-trainable params: 0

```
In [15]: tensorboard = TensorBoard(log_dir='./logs/custom')

         model.fit_generator(train_generator, steps_per_epoch=512, epochs=10, callbacks=[tensorboard], verbose=2)
```

```
Epoch 1/10
 - 434s - loss: 4.4682 - acc: 0.0687
Epoch 2/10
 - 440s - loss: 4.1851 - acc: 0.0919
Epoch 3/10
 - 443s - loss: 3.9278 - acc: 0.1270
Epoch 4/10
 - 428s - loss: 3.6948 - acc: 0.1615
Epoch 5/10
 - 437s - loss: 3.4944 - acc: 0.1935
Epoch 6/10
 - 439s - loss: 3.3103 - acc: 0.2196
Epoch 7/10
 - 438s - loss: 3.1253 - acc: 0.2492
Epoch 8/10
 - 443s - loss: 2.9927 - acc: 0.2757
Epoch 9/10
 - 431s - loss: 2.8474 - acc: 0.2998
Epoch 10/10
 - 430s - loss: 2.7354 - acc: 0.3271

Out[15]: <keras.callbacks.History at 0x7fe46c531be0>
```



In [1]:
```
import numpy as np
from keras.applications.inception_v3 import InceptionV3
from keras.models import Sequential, load_model, Model
from keras.layers import Input, Dropout, Flatten, Conv2D, MaxPooling2D, Dense, Activation, GlobalAveragePooling2D
from keras.optimizers import SGD
from keras.utils import np_utils
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import TensorBoard
import itertools
```

```
Using TensorFlow backend.
```

In [4]:
```
# create the base pre-trained model
base_model = InceptionV3(weights='imagenet', include_top=False)

# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
# add a fully-connected layer
x = Dense(1024, activation='relu')(x)
out_layer = Dense(200, activation='softmax')(x)

# this is the model we will train
model = Model(inputs=base_model.input, outputs=out_layer)
```

In [5]:
```
# first: train only the top layers (which were randomly initialized)
# i.e. freeze all convolutional InceptionV3 layers
for layer in base_model.layers:
    layer.trainable = False

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

model.summary()
```

```
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_1 (InputLayer)         (None, None, None, 3 0
_________________________________________________________________________________________________
conv2d_1 (Conv2D)            (None, None, None, 3 864         input_1[0][0]
_________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, None, None, 3 96          conv2d_1[0][0]
_________________________________________________________________________________________________
activation_1 (Activation)    (None, None, None, 3 0           batch_normalization_1[0][0]
```

```
In [6]: tensorboard = TensorBoard(log_dir='./logs')

        model.fit_generator(train_generator, steps_per_epoch=32, epochs=100, callbacks=[tensorboard], verbose=2)
Epoch 90/100
 - 26s - loss: 1.5382 - acc: 0.5576
Epoch 91/100
 - 26s - loss: 1.3803 - acc: 0.6133
Epoch 92/100
 - 25s - loss: 1.5539 - acc: 0.5556
Epoch 93/100
 - 26s - loss: 1.5569 - acc: 0.5703
Epoch 94/100
 - 25s - loss: 1.4826 - acc: 0.5791
Epoch 95/100
 - 26s - loss: 1.5378 - acc: 0.5586
Epoch 96/100
 - 26s - loss: 1.5142 - acc: 0.5947
Epoch 97/100
 - 26s - loss: 1.4756 - acc: 0.5762
Epoch 98/100
 - 26s - loss: 1.4465 - acc: 0.6025
Epoch 99/100
 - 26s - loss: 1.5138 - acc: 0.5820
```

```
In [7]: print(model.evaluate_generator(test_generator, steps=5000))

        [2.3260338047121207, 0.44336327658772534]
```

```
In [8]: # unfreeze all layers for more training
        for layer in model.layers:
            layer.trainable = True

        # we need to recompile the model for these modifications to take effect
        # we use SGD with a low learning rate
        model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy', metrics=['accuracy'])

        model.fit_generator(train_generator, steps_per_epoch=32, epochs=100)
Epoch 90/100
32/32 [==============================] - 27s 859ms/step - loss: 0.3098 - acc: 0.9180
Epoch 91/100
32/32 [==============================] - 27s 846ms/step - loss: 0.2865 - acc: 0.9268
Epoch 92/100
32/32 [==============================] - 27s 852ms/step - loss: 0.2902 - acc: 0.9141
Epoch 93/100
32/32 [==============================] - 27s 832ms/step - loss: 0.2934 - acc: 0.9227
Epoch 94/100
32/32 [==============================] - 28s 860ms/step - loss: 0.2731 - acc: 0.9307
Epoch 95/100
32/32 [==============================] - 27s 845ms/step - loss: 0.2777 - acc: 0.9195
```

```
In [9]: test_generator.reset()
        print(model.evaluate_generator(test_generator, steps=5000))

        [1.4155961280392264, 0.63971983164771662]
```

```
In [10]: model.save("birds-inceptionv3.model")
```

```
In [1]: from keras.models import load_model
        from keras.preprocessing import image
        from os import listdir
        import numpy as np

        Using TensorFlow backend.

In [2]: ROWS = 256
        COLS = 256

In [3]: CLASS_NAMES = sorted(listdir('images'))

        model = load_model('birds-inceptionv3.model')
```

```
In [4]: def predict(fname):
            img = image.load_img(fname, target_size=(ROWS, COLS))
            img_tensor = image.img_to_array(img) # (height, width, channels)
            # (1, height, width, channels), add a dimension because the model expects this shape:
            # (batch_size, height, width, channels)
            img_tensor = np.expand_dims(img_tensor, axis=0)
            img_tensor /= 255. # model expects values in the range [0, 1]
            prediction = model.predict(img_tensor)[0]
            best_score_index = np.argmax(prediction)
            bird = CLASS_NAMES[best_score_index] # retrieve original class name
            print("Prediction: %s (%.2f%%)" % (bird, 100*prediction[best_score_index]))
```

```
In [5]: predict('test-birds/annas_hummingbird_sim_1.jpg')
        predict('test-birds/house_wren.jpg')
        predict('test-birds/canada_goose_1.jpg')

        Prediction: 067.Anna_Hummingbird (98.76%)
        Prediction: 196.House_Wren (47.01%)
        Prediction: 071.Long_tailed_Jaeger (37.12%)
```