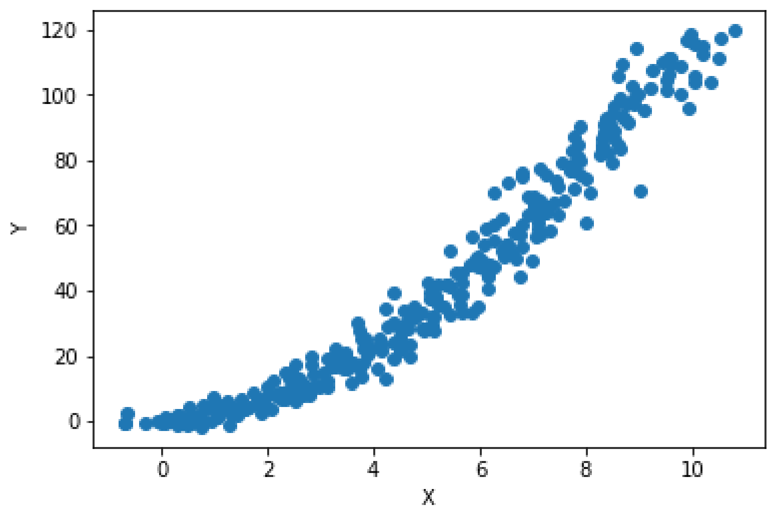
COGS 260 Assignment3

1 Least Square Estimation

In [134]:
```
# 1.1 Import packages and load data
import numpy as np
import matplotlib.pyplot as plt
X_and_Y = np.load('./q1-least-square.npy')
X = X_and_Y[:, 0] # Shape: (300,)
Y = X_and_Y[:, 1] # Shape: (300,)
```

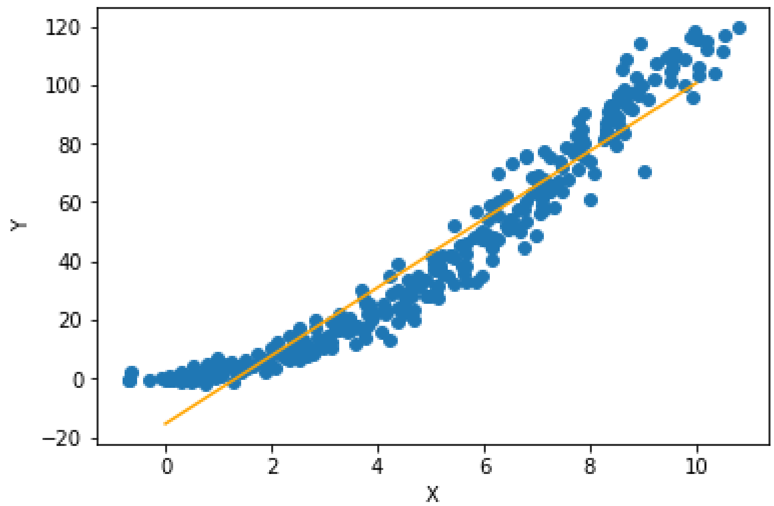
In [52]:
```
# 1.2 Plot the scatter graph of data
plt.scatter(X, Y)
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



In [40]:
```
# 1.3 Compute the least square line over the given data.
# Assume Y = w0 + w1*X = (w0, w1).(1, X) = W.X1
# X1 contains 1 and X.
X1 = np.matrix(np.hstack((np.ones((len(X),1)),X.reshape(-1,1))))
W = X1.T.dot(X1).I.dot(X1.T).dot(Y)
w0, w1 = np.array(W).reshape(-1)
print('Y = {:.2f} + {:.2f}*X'.format(w0, w1))
```

```
Y = -15.47 + 11.61*X
```

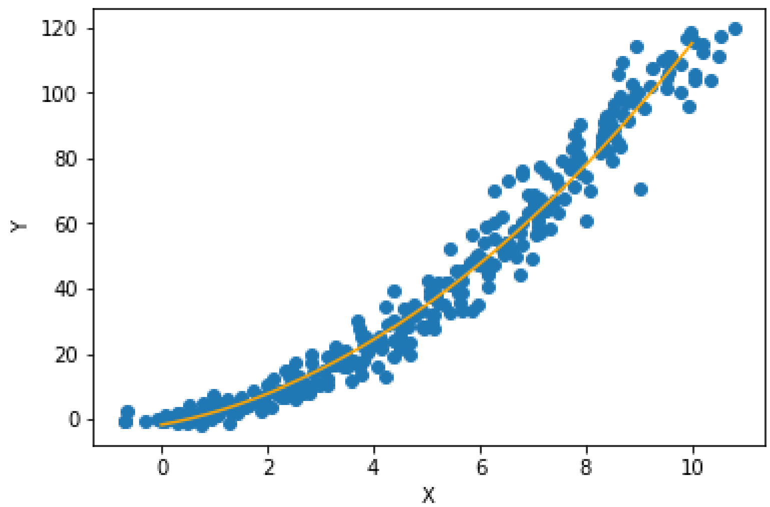
In [42]:
```
# 1.4 Plot the scatter graph of data and estimated line.
X_line = np.linspace(0,10,300)
Y_line = w0 + w1 * X_line
plt.scatter(X, Y)
plt.plot(X_line, Y_line, color='orange')
plt.xlabel('X')
plt.ylabel('Y')
plt.savefig('imgs/1_1.png')
plt.show()
```



In [33]:
```
# 1.5 Compute the least square parabola over the given data.
# Assume Y = w0 + w1*X + w2*X^2 = (w0, w1, w2).(1, X, X^2) = W.X2
# X2 contains 1, X and X^2.
X2 = np.matrix(np.hstack((np.hstack((np.ones((len(X),1)),X.reshape(-1,1
))),X.reshape(-1,1)**2)))
W = X2.T.dot(X2).I.dot(X2.T).dot(Y)
w0, w1, w2 = np.array(W).reshape(-1)
print('Y = {:.2f} + {:.2f}*X + {:.2f}*X^2'.format(w0, w1, w2))
```

```
Y = -1.71 + 3.02*X + 0.87*X^2
```

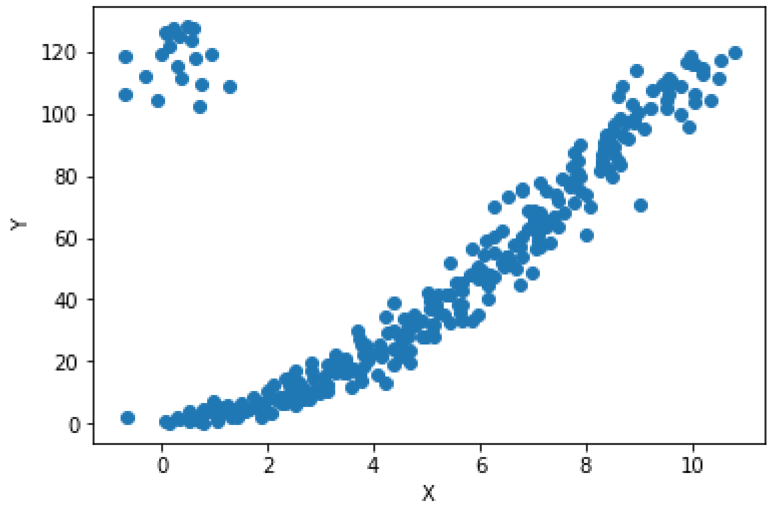
In [35]:
```
# 1.6 Plot the scatter graph of data and estimated parabola
X_line = np.linspace(0,10,300)
Y_line = w0 + w1 * X_line + w2 * (X_line**2)
plt.scatter(X, Y)
plt.plot(X_line, Y_line, color='orange')
plt.xlabel('X')
plt.ylabel('Y')
plt.savefig('imgs/1_2.png')
plt.show()
```



2 Parabola Estimation

In [109]:
```
# 2.1 Import packages and load data
import numpy as np
import matplotlib.pyplot as plt
X_and_Y = np.load('./q2-parabola.npy')
X = X_and_Y[:, 0] # Shape: (300,)
Y = X_and_Y[:, 1] # Shape: (300,)
```

In [110]:
```
# 2.2 Plot the scatter graph of data
plt.scatter(X, Y)
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

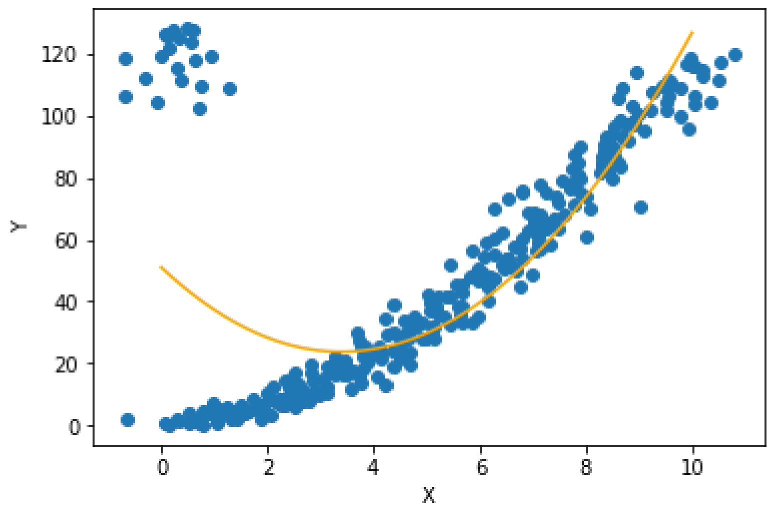


We are comparing the robustness of different loss definition so we can see a bunch of outliers at the upleft corner.

In [111]:
```
# 2.3 Compute the least square(L2 norm) parabola over the given data.
# Assume Y = w0 + w1*X + w2*X^2 = (w0, w1, w2).(1, X, X^2) = W.X2
# X2 contains 1, X and X^2.
X1 = np.matrix(np.hstack((np.hstack((np.ones((len(X),1)),X.reshape(-1,1
))),X.reshape(-1,1)**2)))
W = X1.T.dot(X1).I.dot(X1.T).dot(Y)
w0, w1, w2 = np.array(W).reshape(-1)
print('Y = {:.2f} + {:.2f}*X + {:.2f}*X^2'.format(w0, w1, w2))
```

```
Y = 51.07 + -16.06*X + 2.36*X^2
```

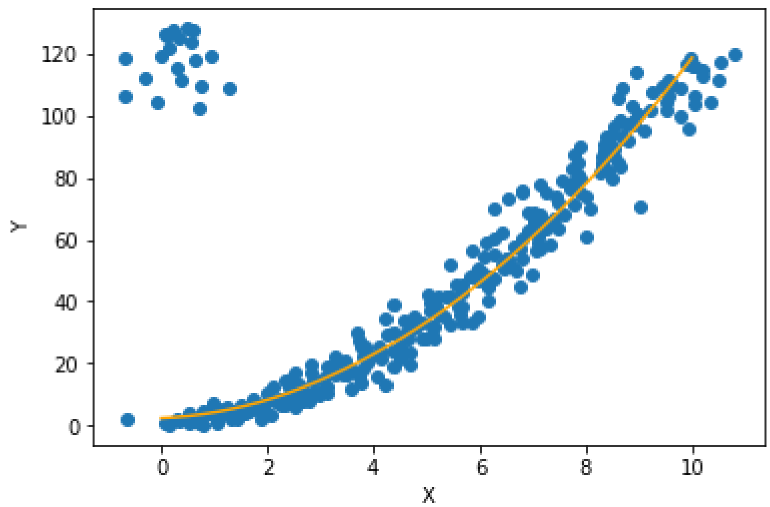
In [112]:
```
# 2.4 Plot the scatter graph of data and estimated parabola
X_line = np.linspace(0,10,300)
Y_line = w0 + w1 * X_line + w2 * (X_line**2)
plt.scatter(X, Y)
plt.plot(X_line, Y_line, color='orange')
plt.xlabel('X')
plt.ylabel('Y')
plt.savefig('imgs/2_1.png')
plt.show()
```



In [124]:
```
# 2.5 Compute the L1 norm loss parabola over the given data.
# Assume Y = w0 + w1*X + w2*X^2 = (w0, w1, w2).(1, X, X^2) = W.X2
# X2 contains 1, X and X^2.
X2 = np.matrix(np.hstack((np.hstack((np.ones((len(X),1)),X.reshape(-1,1
))),X.reshape(-1,1)**2)))
Y2 = np.expand_dims(Y,1)
# First randomly pick a W
W = np.random.normal(0,5,(3,1))
# Define the L1-norm loss
loss = np.sum(np.abs(X2.dot(W) - Y2))
# gradient descent parameters (refer to the hint)
num_point = len(Y)
max_iterations = 300000
learning_rate = 0.000001
threshold = 0.00001
iters = 0
while loss/num_point > threshold and iters < max_iterations:
    grad = np.sign(X2.dot(W) - Y2).T.dot(X2).T
    W = W - learning_rate*grad
    loss = np.sum(np.abs(X2.dot(W) - Y2))
    iters += 1
print("total iterations %d \t average loss %f"%(iters,loss/num_point))

w0, w1, w2 = np.array(W).reshape(-1)
print('Y = {:.2f} + {:.2f}*X + {:.2f}*X^2'.format(w0, w1, w2))
```

```
total iterations 300000          average loss 12.058130
Y = 2.48 + 0.77*X + 1.08*X^2
```

In [125]:
```
# 2.6 Plot the scatter graph of data and estimated parabola
X_line = np.linspace(0,10,300)
Y_line = w0 + w1 * X_line + w2 * (X_line**2)
plt.scatter(X, Y)
plt.plot(X_line, Y_line, color='orange')
plt.xlabel('X')
plt.ylabel('Y')
plt.savefig('imgs/2_2.png')
plt.show()
```



3 Perceptron Learning

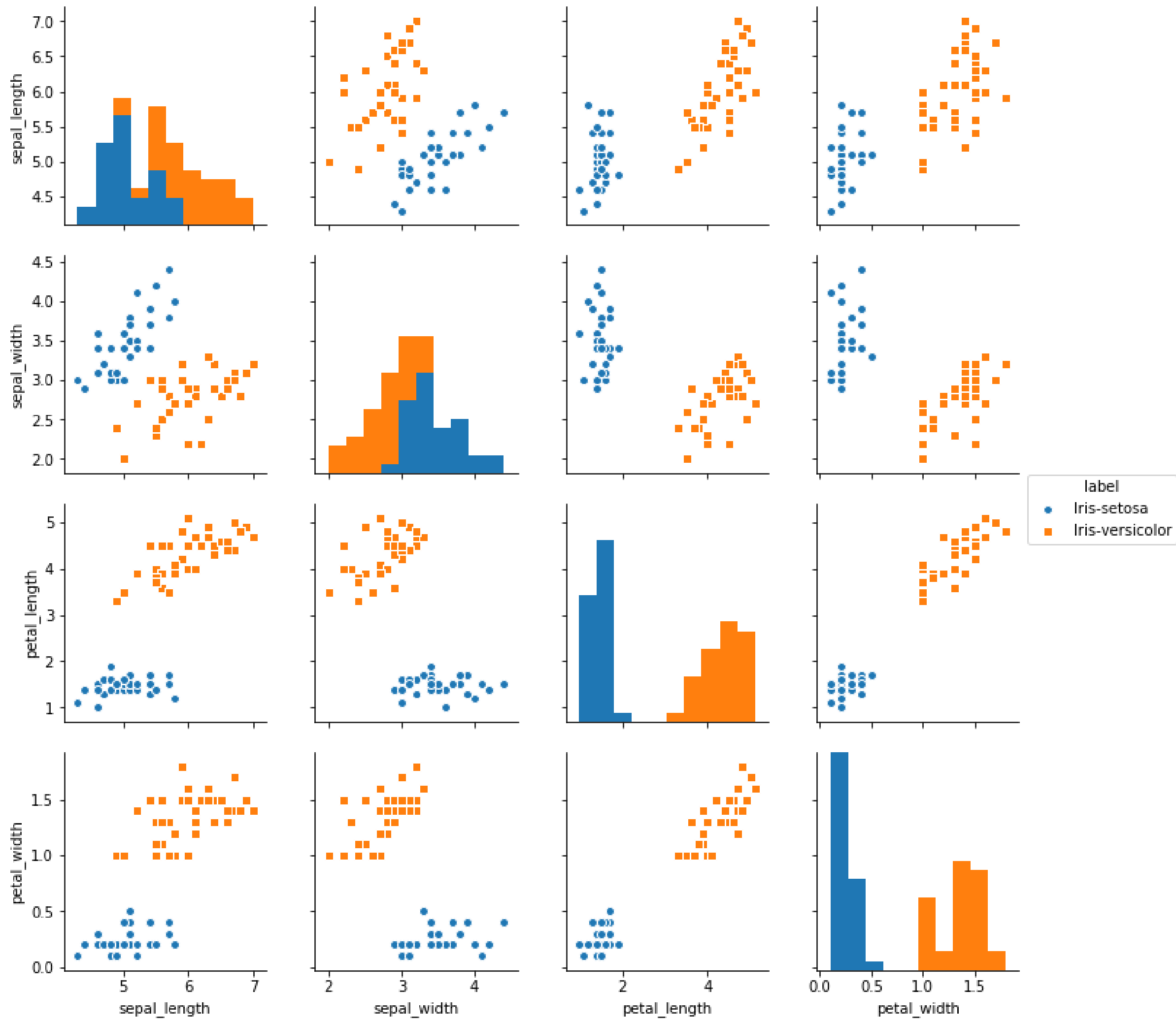
In [110]:
```
# 3.1 Dataset
from sklearn import preprocessing
import numpy as np
import pandas as pd

# load iris_train.data and iris_test.data
pd_train = pd.read_csv("iris/iris_train.data", names=["sepal_length", "s
epal_width", "petal_length", "petal_width", "label"])
pd_test = pd.read_csv("iris/iris_test.data", names=["sepal_length", "sep
al_width", "petal_length", "petal_width", "label"])

# parse the features and labels as numpy arrays.
X_train = pd_train.as_matrix(columns=["sepal_length", "sepal_width", "pe
tal_length", "petal_width"])
y_train = pd_train.as_matrix(columns=["label"]).ravel()
X_test = pd_test.as_matrix(columns=["sepal_length", "sepal_width", "peta
l_length", "petal_width"])
y_test = pd_test.as_matrix(columns=["label"]).ravel()

# Encode labels
le = preprocessing.LabelEncoder()
le.fit(y_train)
print(le.classes_)
y_train = le.transform(y_train)
y_test = le.transform(y_test)
```

```
['Iris-setosa' 'Iris-versicolor']
```

In [115]:
```
type(pd_train)
```

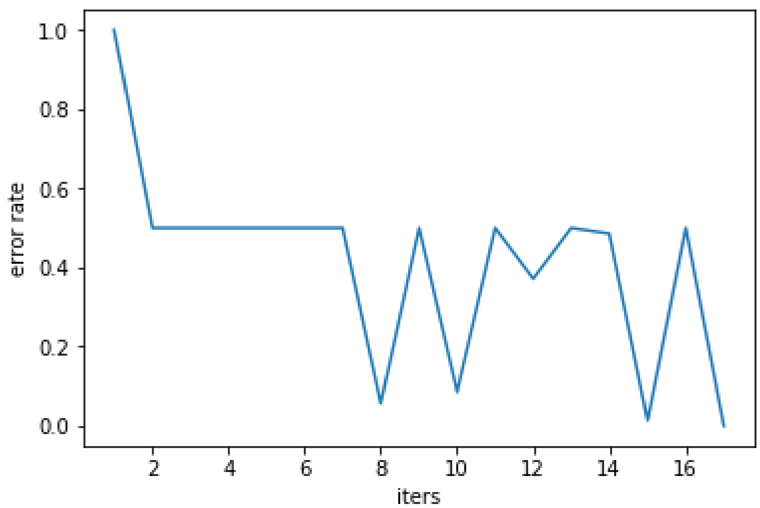
Out[115]:
```
pandas.core.frame.DataFrame
```

In [119]:
```
import seaborn as sns
g = sns.pairplot(pd_train,hue="label",markers=["o", "s"])
```



In [299]:
```
# 3.2 Programming
W = np.random.normal(0,5,(4,1))
b = np.random.normal(0,5,1)
y_predict = (X_train.dot(W)+b>0).reshape(-1)
loss = np.sum(y_predict!=y_train)

num_point = len(y_train)
max_iterations = 300000
Lambda = 1
iters = 0
loss_list = []
while loss > 0 and iters < max_iterations:
    rand_int = np.random.randint(num_point)
    y_predict = X_train[rand_int].dot(W)+b>0
    if y_train[rand_int] == y_predict:
        continue
    else:
        W = W + Lambda*(y_train[rand_int]-y_predict)*np.expand_dims(X_tr
ain[rand_int],1)
        b = b + Lambda*(y_train[rand_int]-y_predict)
    y_predict = (X_train.dot(W)+b>0).reshape(-1)
    loss = np.sum(y_predict!=y_train)/num_point
    iters += 1
    loss_list.extend([loss])
    print("iter %2d \t loss %f"%(iters,loss))

plt.plot(range(1,1+iters),loss_list)
plt.xlabel('iters')
plt.ylabel('error rate')
plt.show()
```

```
iter  1         loss 1.000000
iter  2         loss 0.500000
iter  3         loss 0.500000
iter  4         loss 0.500000
iter  5         loss 0.500000
iter  6         loss 0.500000
iter  7         loss 0.500000
iter  8         loss 0.057143
iter  9         loss 0.500000
iter 10         loss 0.085714
iter 11         loss 0.500000
iter 12         loss 0.371429
iter 13         loss 0.500000
iter 14         loss 0.485714
iter 15         loss 0.014286
iter 16         loss 0.500000
iter 17         loss 0.000000
```



In [300]:

```
# 3.3 Decision Boundary
w0, w1, w2, w3 = np.array(W).reshape(-1)
b = b[0]
print('Y = {:.2f}*sepal_length + {:.2f}*sepal_width + {:.2f}*petal_lengt
h + {:.2f}*petal_width + {:.2f}'.format(w0, w1, w2, w3, b))
```

```
Y = 2.14*sepal_length + -15.07*sepal_width + 7.66*petal_length + 6.91*p
etal_width + 2.32
```

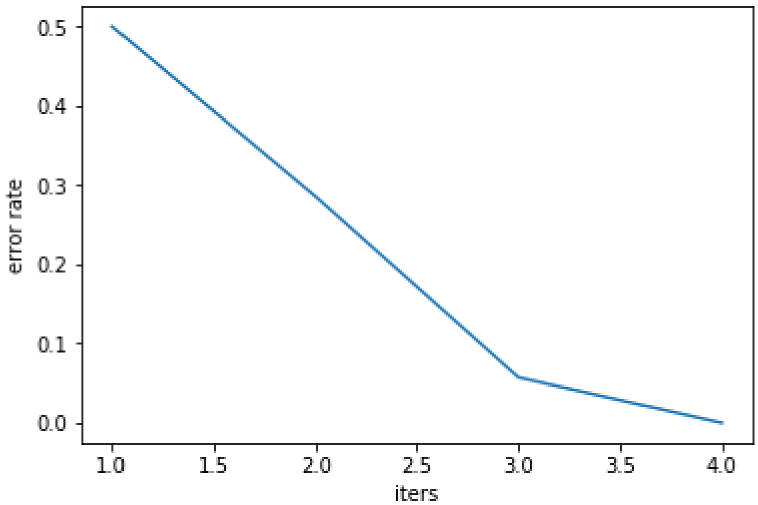
In [301]:
```
# 3.4 Test
# 1.accuracy
num_point = len(y_test)
y_predict = (X_test.dot(W)+b>0).reshape(-1)
accuracy = np.sum(y_predict == y_test) / num_point
print('accuracy:\t', accuracy)
# For the following, let y_test==1 be positive
true_positive = np.sum(np.logical_and(y_test==1,y_predict==1))
# 2.Precision
precision = true_positive/np.sum(y_predict)
print('precision:\t', precision)
# 3.Recall
recall = true_positive/np.sum(y_test)
print('recall:\t\t',recall)
# 4.F-value
F_value = 2*precision*recall / (precision + recall)
print('F-value:\t',F_value)
```

```
accuracy:        1.0
precision:       1.0
recall:          1.0
F-value:         1.0
```

In [287]:
```
# 3.5 [bonus]Z-score
X_mean = np.mean(X_train,axis=0)
X_var = np.var(X_train,axis=0)

X_train_normalized = (X_train-X_mean)/X_var

W = np.random.normal(0,5,(4,1))
b = np.random.normal(0,5,1)
y_predict = (X_train_normalized.dot(W)+b>0).reshape(-1)
loss = np.sum(y_predict!=y_train)

num_point = len(y_train)
max_iterations = 300000
Lambda = 1
iters = 0
loss_list = []
while loss > 0 and iters < max_iterations:
    rand_int = np.random.randint(num_point)
    y_predict = X_train_normalized[rand_int].dot(W)+b>0
    if y_train[rand_int] == y_predict:
        continue
    else:
        W = W + Lambda*(y_train[rand_int]-y_predict)*np.expand_dims(X_tr
ain_normalized[rand_int],1)
        b = b + Lambda*(y_train[rand_int]-y_predict)
    y_predict = (X_train_normalized.dot(W)+b>0).reshape(-1)
    loss = np.sum(y_predict!=y_train)/num_point
    iters += 1
    loss_list.extend([loss])
    print("iter %2d \t loss %f"%(iters,loss))

plt.plot(range(1,1+iters),loss_list)
plt.xlabel('iters')
plt.ylabel('error rate')
plt.show()

w0, w1, w2, w3 = np.array(W).reshape(-1)
b = b[0]
print('Y = {:.2f}*sepal_length + {:.2f}*sepal_width + {:.2f}*petal_lengt
h + {:.2f}*petal_width + {:.2f}'.format(w0, w1, w2, w3, b))

num_point = len(y_test)
X_test_normalized = (X_test-X_mean)/X_var
y_predict = (X_test.dot(W)+b>0).reshape(-1)
accuracy = np.sum(y_predict == y_test) / num_point
print('accuracy:\t', accuracy)
# For the following, let y_test==1 be positive
true_positive = np.sum(np.logical_and(y_test==1,y_predict==1))
# 2.Precision
precision = true_positive/np.sum(y_predict)
print('precision:\t', precision)
# 3.Recall
recall = true_positive/np.sum(y_test)
print('recall:\t\t',recall)
# 4.F-value
```

```
F_value = 2*precision*recall / (precision + recall)
print('F-value:\t', F_value)
```

```
iter  1          loss 0.500000
iter  2          loss 0.285714
iter  3          loss 0.057143
iter  4          loss 0.000000
```




```
Y = 5.46*sepal_length + -3.56*sepal_width + 1.39*petal_length + 0.23*pe
tal_width + 3.19
accuracy:        0.5
precision:       0.5
recall:          1.0
F-value:         0.6666666666666666
```

After normalizing the features, the weights on features have changed. Before sepal_width has the largest weight. Now sepal_length is the most important one.

Notice that with z-score features, the training process tends to converge faster. However, it is likely to overfit the training set since this dataset is pretty small so the distribution difference between training set and test set is not too small to ignore. That's why the test performance is not as good as using origin features

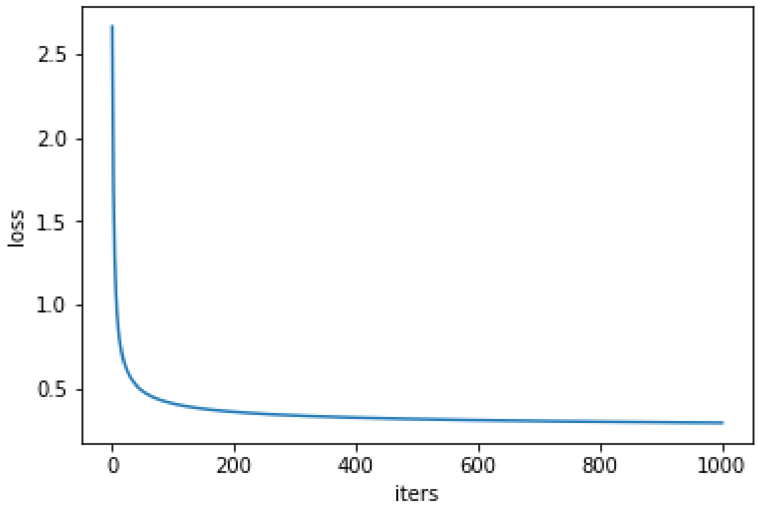
4 Feed Forward Neural Network

In [18]:
```
# preparing the data
import numpy as np
import matplotlib.pyplot as plt
from utils import load_mnist
(x_train, y_train), (x_test, y_test) = load_mnist()
x_train_vector = np.reshape(x_train,(-1,28*28))
x_test_vector = np.reshape(x_test,(-1,28*28))
x_valid_vector = x_train_vector[50000:]
y_valid = y_train[50000:]
x_train_vector = x_train_vector[:50000]
y_train = y_train[:50000]
print('train num:%d valid num:%d test num:%d'%(len(y_train),len(y_valid
),len(y_test)))

def softmax(x):
    e_x = np.exp(x - np.max(x))
    return e_x / np.expand_dims(e_x.sum(axis=1),1)
```

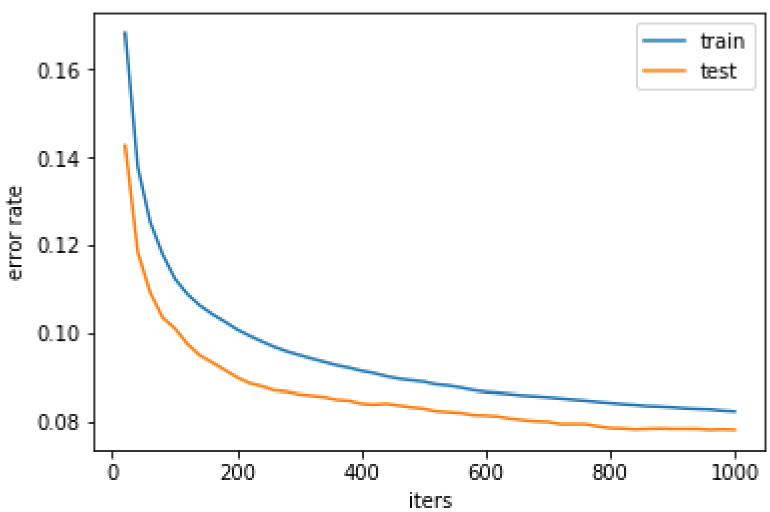
```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
train num:50000 valid num:10000 test num:10000
```

In [22]:
```
# 4.1 Train a feed forward network with 1 hidden layer
# Here we use Cross Entropy Loss with Softmax activation function
W = np.random.normal(0,0.1,(784,10))
num_point = len(y_train)
max_iterations = 1000
learning_rate = 1e-5
threshold = 0.0001
loss = num_point
iters = 0
loss_list = []
train_error_list = []
valid_error_list = []
while loss > threshold and iters < max_iterations:
    layer_output = x_train_vector.dot(W)
    y_predict = softmax(layer_output)
    loss = - np.sum(y_train * np.log(np.clip(y_predict,1e-15,np.inf)))/n
um_point
    grad = y_predict - y_train
    W_grad = x_train_vector.T.dot(grad)
    W -= learning_rate*W_grad
    iters += 1
    loss_list.extend([loss])
    if iters%20 == 0:
        y_max = np.argmax(y_predict,axis=1)
        train_error = 1-np.sum(y_train[range(len(y_train)),y_max])/len(y
_train)
        train_error_list.extend([train_error])
        layer_output = x_valid_vector.dot(W)
        y_predict = softmax(layer_output)
        y_max = np.argmax(y_predict,axis=1)
        valid_error = 1-np.sum(y_valid[range(len(y_valid)),y_max])/len(y
_valid)
        valid_error_list.extend([valid_error])
        print('iter %4d loss: %.3f train: %.3f valid: %.3f'%(iters,loss,
train_error,valid_error))
plt.plot(range(1,1+iters),loss_list)
plt.xlabel('iters')
plt.ylabel('loss')
plt.show()
```

```
iter   20 loss: 0.655 train: 0.168 valid: 0.143
iter   40 loss: 0.517 train: 0.138 valid: 0.119
iter   60 loss: 0.463 train: 0.125 valid: 0.109
iter   80 loss: 0.432 train: 0.118 valid: 0.104
iter  100 loss: 0.411 train: 0.112 valid: 0.101
iter  120 loss: 0.396 train: 0.109 valid: 0.098
iter  140 loss: 0.385 train: 0.106 valid: 0.095
iter  160 loss: 0.375 train: 0.104 valid: 0.093
iter  180 loss: 0.368 train: 0.103 valid: 0.092
iter  200 loss: 0.361 train: 0.101 valid: 0.090
iter  220 loss: 0.356 train: 0.099 valid: 0.089
iter  240 loss: 0.351 train: 0.098 valid: 0.088
iter  260 loss: 0.347 train: 0.097 valid: 0.087
iter  280 loss: 0.343 train: 0.096 valid: 0.087
iter  300 loss: 0.340 train: 0.095 valid: 0.086
iter  320 loss: 0.337 train: 0.094 valid: 0.086
iter  340 loss: 0.334 train: 0.094 valid: 0.086
iter  360 loss: 0.331 train: 0.093 valid: 0.085
iter  380 loss: 0.329 train: 0.092 valid: 0.085
iter  400 loss: 0.327 train: 0.092 valid: 0.084
iter  420 loss: 0.325 train: 0.091 valid: 0.084
iter  440 loss: 0.323 train: 0.090 valid: 0.084
iter  460 loss: 0.321 train: 0.090 valid: 0.084
iter  480 loss: 0.320 train: 0.089 valid: 0.083
iter  500 loss: 0.318 train: 0.089 valid: 0.083
iter  520 loss: 0.317 train: 0.088 valid: 0.082
iter  540 loss: 0.315 train: 0.088 valid: 0.082
iter  560 loss: 0.314 train: 0.088 valid: 0.082
iter  580 loss: 0.313 train: 0.087 valid: 0.082
iter  600 loss: 0.312 train: 0.087 valid: 0.081
iter  620 loss: 0.311 train: 0.087 valid: 0.081
iter  640 loss: 0.309 train: 0.086 valid: 0.081
iter  660 loss: 0.308 train: 0.086 valid: 0.080
iter  680 loss: 0.307 train: 0.086 valid: 0.080
iter  700 loss: 0.307 train: 0.086 valid: 0.080
iter  720 loss: 0.306 train: 0.085 valid: 0.080
iter  740 loss: 0.305 train: 0.085 valid: 0.080
iter  760 loss: 0.304 train: 0.085 valid: 0.080
iter  780 loss: 0.303 train: 0.085 valid: 0.079
iter  800 loss: 0.302 train: 0.084 valid: 0.079
iter  820 loss: 0.302 train: 0.084 valid: 0.079
iter  840 loss: 0.301 train: 0.084 valid: 0.078
iter  860 loss: 0.300 train: 0.084 valid: 0.078
iter  880 loss: 0.300 train: 0.083 valid: 0.079
iter  900 loss: 0.299 train: 0.083 valid: 0.078
iter  920 loss: 0.298 train: 0.083 valid: 0.078
iter  940 loss: 0.298 train: 0.083 valid: 0.078
iter  960 loss: 0.297 train: 0.083 valid: 0.078
iter  980 loss: 0.296 train: 0.083 valid: 0.078
iter 1000 loss: 0.296 train: 0.082 valid: 0.078
```



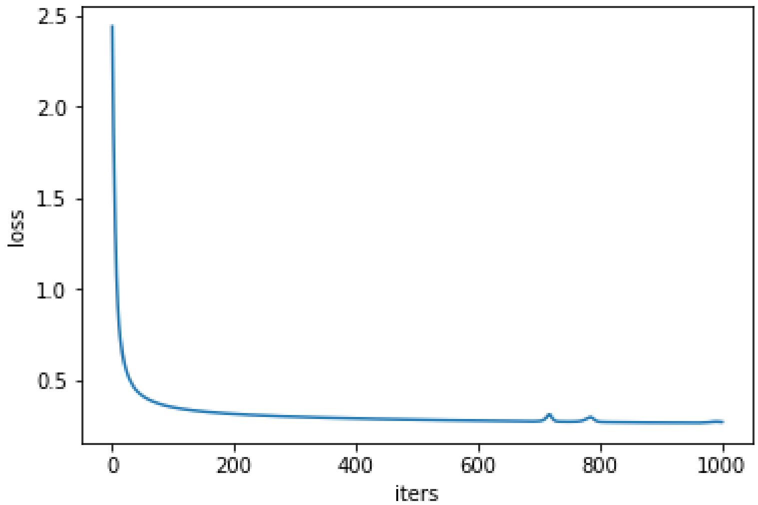
In [23]:
```
plt.plot(range(20,1+iters,20),train_error_list)
plt.plot(range(20,1+iters,20),valid_error_list)
plt.legend(['train','test'])
plt.xlabel('iters')
plt.ylabel('error rate')
plt.show()

print('Final result: training error rate %.3f testing error rate %.3f'%(
train_error_list[-1],valid_error_list[-1]))
```



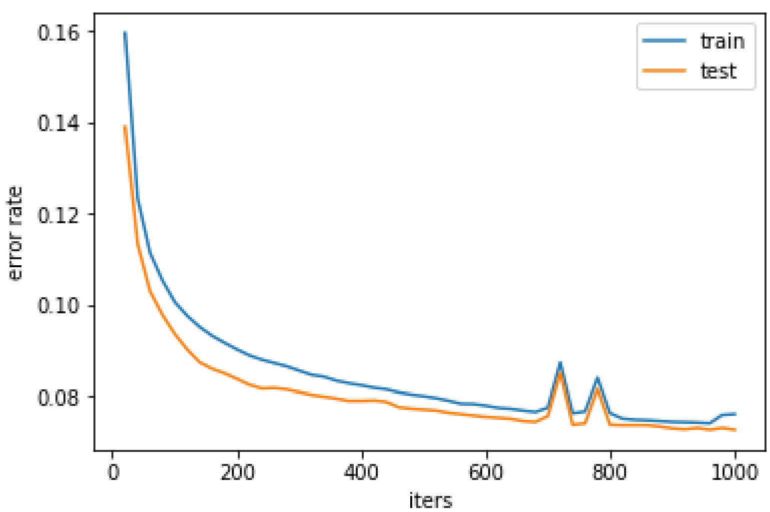

```
Final result: training error rate 0.082 testing error rate 0.078
```

In [24]:
```
# 4.2 feed forward network with 2 hidden layers
# 4.1 Train a feed forward network with 1 hidden layer
# Here we use Cross Entropy Loss with Softmax activation function
W1 = np.random.normal(0,0.1,(784,32))
W2 = np.random.normal(0,0.1,(32,10))
num_point = len(y_train)
max_iterations = 1000
learning_rate = 1e-5
threshold = 0.0001
loss = num_point
iters = 0
loss_list = []
train_error_list = []
valid_error_list = []
while loss > threshold and iters < max_iterations:
    layer_output_1 = x_train_vector.dot(W1)
    layer_output_2 = layer_output_1.dot(W2)
    y_predict = softmax(layer_output_2)
    loss = - np.sum(y_train * np.log(np.clip(y_predict,1e-15,np.inf)))/n
um_point
    grad_2 = y_predict - y_train
    W_grad_2 = layer_output_1.T.dot(grad_2)
    grad_1 = (y_predict - y_train).dot(W2.T)
    W_grad_1 = x_train_vector.T.dot(grad_1)

    W2 -= learning_rate*W_grad_2#/num_point
    W1 -= learning_rate*W_grad_1#/num_point

    iters += 1
    loss_list.extend([loss])

    if iters%20 == 0:
        y_max = np.argmax(y_predict,axis=1)
        train_error = 1-np.sum(y_train[range(len(y_train)),y_max])/len(y
_train)
        train_error_list.extend([train_error])
        layer_output_1 = x_valid_vector.dot(W1)
        layer_output_2 = layer_output_1.dot(W2)
        y_predict = softmax(layer_output_2)
        y_max = np.argmax(y_predict,axis=1)
        valid_error = 1-np.sum(y_valid[range(len(y_valid)),y_max])/len(y
_valid)
        valid_error_list.extend([valid_error])
        print('iter %4d loss: %.3f train: %.3f valid: %.3f'%(iters,loss,
train_error,valid_error))
plt.plot(range(1,1+iters),loss_list)
plt.xlabel('iters')
plt.ylabel('loss')
plt.show()
```

```
iter   20 loss: 0.598 train: 0.160 valid: 0.139
iter   40 loss: 0.447 train: 0.123 valid: 0.113
iter   60 loss: 0.398 train: 0.111 valid: 0.103
iter   80 loss: 0.371 train: 0.105 valid: 0.098
iter  100 loss: 0.355 train: 0.101 valid: 0.094
iter  120 loss: 0.343 train: 0.098 valid: 0.090
iter  140 loss: 0.335 train: 0.095 valid: 0.087
iter  160 loss: 0.328 train: 0.093 valid: 0.086
iter  180 loss: 0.323 train: 0.092 valid: 0.085
iter  200 loss: 0.318 train: 0.090 valid: 0.084
iter  220 loss: 0.314 train: 0.089 valid: 0.083
iter  240 loss: 0.311 train: 0.088 valid: 0.082
iter  260 loss: 0.308 train: 0.087 valid: 0.082
iter  280 loss: 0.305 train: 0.087 valid: 0.082
iter  300 loss: 0.303 train: 0.086 valid: 0.081
iter  320 loss: 0.301 train: 0.085 valid: 0.080
iter  340 loss: 0.299 train: 0.084 valid: 0.080
iter  360 loss: 0.297 train: 0.083 valid: 0.080
iter  380 loss: 0.295 train: 0.083 valid: 0.079
iter  400 loss: 0.293 train: 0.082 valid: 0.079
iter  420 loss: 0.292 train: 0.082 valid: 0.079
iter  440 loss: 0.291 train: 0.082 valid: 0.079
iter  460 loss: 0.289 train: 0.081 valid: 0.078
iter  480 loss: 0.288 train: 0.080 valid: 0.077
iter  500 loss: 0.287 train: 0.080 valid: 0.077
iter  520 loss: 0.286 train: 0.080 valid: 0.077
iter  540 loss: 0.284 train: 0.079 valid: 0.076
iter  560 loss: 0.283 train: 0.078 valid: 0.076
iter  580 loss: 0.282 train: 0.078 valid: 0.076
iter  600 loss: 0.281 train: 0.078 valid: 0.076
iter  620 loss: 0.281 train: 0.077 valid: 0.075
iter  640 loss: 0.280 train: 0.077 valid: 0.075
iter  660 loss: 0.279 train: 0.077 valid: 0.075
iter  680 loss: 0.278 train: 0.077 valid: 0.074
iter  700 loss: 0.279 train: 0.078 valid: 0.076
iter  720 loss: 0.304 train: 0.087 valid: 0.085
iter  740 loss: 0.276 train: 0.076 valid: 0.074
iter  760 loss: 0.276 train: 0.077 valid: 0.074
iter  780 loss: 0.295 train: 0.084 valid: 0.082
iter  800 loss: 0.275 train: 0.076 valid: 0.074
iter  820 loss: 0.273 train: 0.075 valid: 0.074
iter  840 loss: 0.273 train: 0.075 valid: 0.074
iter  860 loss: 0.272 train: 0.075 valid: 0.074
iter  880 loss: 0.271 train: 0.075 valid: 0.073
iter  900 loss: 0.271 train: 0.074 valid: 0.073
iter  920 loss: 0.270 train: 0.074 valid: 0.073
iter  940 loss: 0.270 train: 0.074 valid: 0.073
iter  960 loss: 0.270 train: 0.074 valid: 0.073
iter  980 loss: 0.274 train: 0.076 valid: 0.073
iter 1000 loss: 0.274 train: 0.076 valid: 0.073
```



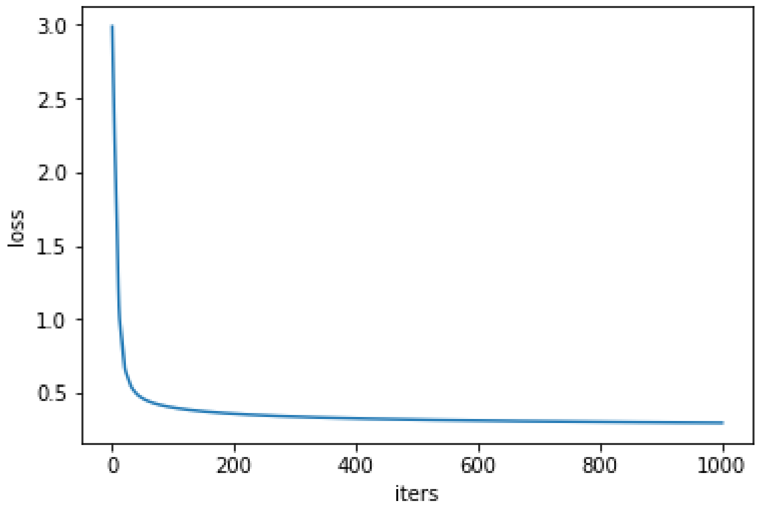
In [32]:
```
plt.plot(range(20,1+iters,20),train_error_list)
plt.plot(range(20,1+iters,20),valid_error_list)
plt.legend(['train','test'])
plt.xlabel('iters')
plt.ylabel('error rate')
plt.show()

print('Final result: training error rate %.3f testing error rate %.3f'%(
train_error_list[-1],valid_error_list[-1]))
```



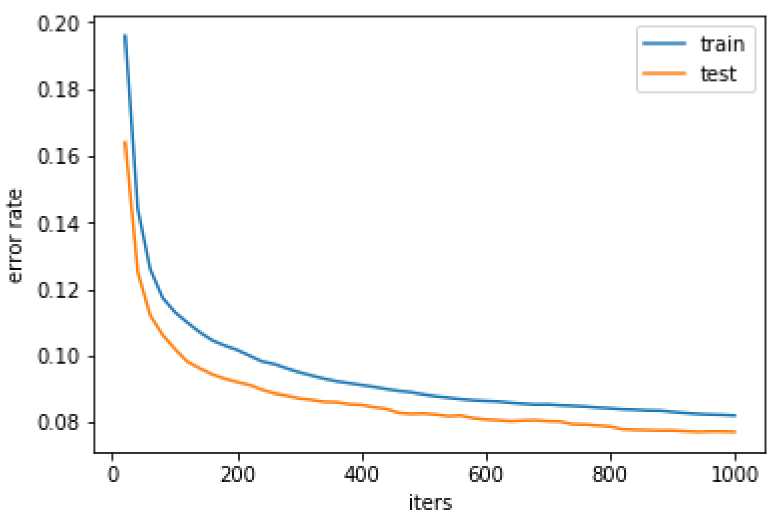

```
Final result: training error rate 0.076 testing error rate 0.073
```

In [33]:
```
# 4.3 Retrain the network in part 1 with regularization and momentum and
 report the results
W = np.random.normal(0,0.1,(784,10))
num_point = len(y_train)
max_iterations = 1000
learning_rate = 1e-5
threshold = 0.0001
sigma = 1e-5
beta = 0.9
loss = num_point
iters = 0
W_Momentum = np.zeros(W.shape)
loss_list = []
train_error_list = []
valid_error_list = []
while loss > threshold and iters < max_iterations:
    layer_output = x_train_vector.dot(W)
    y_predict = softmax(layer_output)
    loss = - np.sum(y_train * np.log(y_predict))/num_point + sigma*np.su
m(W**2)
    grad = y_predict - y_train
    W_grad = x_train_vector.T.dot(grad) + sigma*2*W
    W_Momentum = beta*W_Momentum + (1-beta)*W_grad
    W -= learning_rate*W_Momentum
    iters += 1
    loss_list.extend([loss])
    if iters%20 == 0:
        y_max = np.argmax(y_predict,axis=1)
        train_error = 1-np.sum(y_train[range(len(y_train)),y_max])/len(y
_train)
        train_error_list.extend([train_error])
        layer_output = x_valid_vector.dot(W)
        y_predict = softmax(layer_output)
        y_max = np.argmax(y_predict,axis=1)
        valid_error = 1-np.sum(y_valid[range(len(y_valid)),y_max])/len(y
_valid)
        valid_error_list.extend([valid_error])
        print('iter %4d loss: %.3f train: %.3f valid: %.3f'%(iters,loss,
train_error,valid_error))
plt.plot(range(1,1+iters),loss_list)
plt.xlabel('iters')
plt.ylabel('loss')
plt.show()
```

```
iter   20 loss: 0.690 train: 0.196 valid: 0.164
iter   40 loss: 0.496 train: 0.144 valid: 0.125
iter   60 loss: 0.444 train: 0.126 valid: 0.112
iter   80 loss: 0.418 train: 0.117 valid: 0.106
iter  100 loss: 0.401 train: 0.113 valid: 0.102
iter  120 loss: 0.389 train: 0.110 valid: 0.098
iter  140 loss: 0.379 train: 0.107 valid: 0.096
iter  160 loss: 0.371 train: 0.105 valid: 0.094
iter  180 loss: 0.365 train: 0.103 valid: 0.093
iter  200 loss: 0.359 train: 0.102 valid: 0.092
iter  220 loss: 0.354 train: 0.100 valid: 0.091
iter  240 loss: 0.350 train: 0.098 valid: 0.090
iter  260 loss: 0.346 train: 0.097 valid: 0.089
iter  280 loss: 0.342 train: 0.096 valid: 0.088
iter  300 loss: 0.339 train: 0.095 valid: 0.087
iter  320 loss: 0.336 train: 0.094 valid: 0.087
iter  340 loss: 0.334 train: 0.093 valid: 0.086
iter  360 loss: 0.331 train: 0.092 valid: 0.086
iter  380 loss: 0.329 train: 0.092 valid: 0.085
iter  400 loss: 0.327 train: 0.091 valid: 0.085
iter  420 loss: 0.325 train: 0.091 valid: 0.085
iter  440 loss: 0.323 train: 0.090 valid: 0.084
iter  460 loss: 0.322 train: 0.089 valid: 0.083
iter  480 loss: 0.320 train: 0.089 valid: 0.083
iter  500 loss: 0.319 train: 0.088 valid: 0.083
iter  520 loss: 0.317 train: 0.088 valid: 0.082
iter  540 loss: 0.316 train: 0.087 valid: 0.082
iter  560 loss: 0.315 train: 0.087 valid: 0.082
iter  580 loss: 0.314 train: 0.087 valid: 0.081
iter  600 loss: 0.312 train: 0.086 valid: 0.081
iter  620 loss: 0.311 train: 0.086 valid: 0.081
iter  640 loss: 0.310 train: 0.086 valid: 0.080
iter  660 loss: 0.309 train: 0.086 valid: 0.081
iter  680 loss: 0.308 train: 0.085 valid: 0.081
iter  700 loss: 0.307 train: 0.085 valid: 0.080
iter  720 loss: 0.307 train: 0.085 valid: 0.080
iter  740 loss: 0.306 train: 0.085 valid: 0.079
iter  760 loss: 0.305 train: 0.085 valid: 0.079
iter  780 loss: 0.304 train: 0.084 valid: 0.079
iter  800 loss: 0.303 train: 0.084 valid: 0.079
iter  820 loss: 0.303 train: 0.084 valid: 0.078
iter  840 loss: 0.302 train: 0.084 valid: 0.078
iter  860 loss: 0.301 train: 0.084 valid: 0.078
iter  880 loss: 0.301 train: 0.083 valid: 0.078
iter  900 loss: 0.300 train: 0.083 valid: 0.078
iter  920 loss: 0.299 train: 0.083 valid: 0.077
iter  940 loss: 0.299 train: 0.082 valid: 0.077
iter  960 loss: 0.298 train: 0.082 valid: 0.077
iter  980 loss: 0.298 train: 0.082 valid: 0.077
iter 1000 loss: 0.297 train: 0.082 valid: 0.077
```



In [34]:
```
plt.plot(range(20,1+iters,20),train_error_list)
plt.plot(range(20,1+iters,20),valid_error_list)
plt.legend(['train','test'])
plt.xlabel('iters')
plt.ylabel('error rate')
plt.show()

print('Final result: training error rate %.3f testing error rate %.3f'%(
train_error_list[-1],valid_error_list[-1]))
```




```
Final result: training error rate 0.082 testing error rate 0.077
```

5 Convolutional Neural Network

In [23]:
```
from keras.datasets import cifar10
import keras
import numpy as np
import cv2
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train_VGG = np.array(x_train,dtype=np.float32)
x_test_VGG = np.array(x_test,dtype=np.float32)

'''
x_train_VGG = np.zeros((50000,224,224,3),dtype = np.float32)
for index in range(50000):
    x_train_VGG[index,:,:,:] = cv2.resize(x_train[index,:,:,:],(224,22
4))
x_test_VGG = np.zeros((10000,224,224,3),dtype = np.float32)
for index in range(10000):
    x_test_VGG[index,:,:,:] = cv2.resize(x_test[index,:,:,:],(224,224))
'''
y_train_VGG = keras.utils.to_categorical(y_train, 10)
y_test_VGG = keras.utils.to_categorical(y_test, 10)
```

1. Stochastic Gradient Descent [4]
2. Batch Normalization
3. Replace the fully connected layer by average pooling layer
4. Adaptive Gradient [bonus + 2] [3]
5. Nesterov’s Accelerated Gradient [bonus + 2] [4]
6. RMSprop [bonus +2] [3]

In [151]:
```
import keras
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from keras.models import Model
from keras.layers import Input, Dense, GlobalAveragePooling2D, Conv2D, M
axPooling2D, Flatten
import numpy as np

img_input = Input((32,32,3))
# Block 1
x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_c
onv1')(img_input)
x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_c
onv2')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block1_pool')(x)

# Block 2
x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_
conv1')(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_
conv2')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block2_pool')(x)

# Block 3
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_
conv1')(x)
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_
conv2')(x)
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_
conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block3_pool')(x)
'''
# Block 4
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_
conv1')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_
conv2')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_
conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block4_pool')(x)

# Block 5
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_
conv1')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_
conv2')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_
conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block5_pool')(x)
'''
# Final Block
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
x = Dense(32, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)
```

```
model = Model(inputs=img_input, outputs=predictions)
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_51 (InputLayer)        (None, 32, 32, 3)         0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 32, 32, 64)        1792      
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 32, 32, 64)        36928     
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0         
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 16, 16, 128)       73856     
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 16, 16, 128)       147584    
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0         
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 8, 8, 256)         295168    
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080    
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 8, 8, 256)         590080    
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0         
_________________________________________________________________
flatten_11 (Flatten)         (None, 4096)              0         
_________________________________________________________________
dense_130 (Dense)            (None, 128)               524416    
_________________________________________________________________
dense_131 (Dense)            (None, 32)                4128      
_________________________________________________________________
dense_132 (Dense)            (None, 10)                330       
=================================================================
Total params: 2,264,362
Trainable params: 2,264,362
Non-trainable params: 0
_________________________________________________________________
```

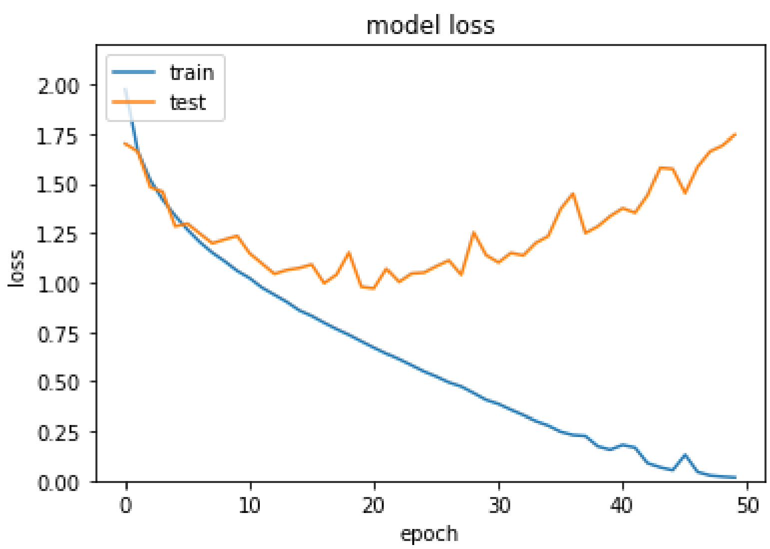
In [91]:
```
BATCH_SIZE = 128
MAX_EPOCH = 50
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.SGD(lr=1e-3),
              metrics=['accuracy'])
history_SGD_1 = model.fit(x_train_VGG, y_train_VGG,
          batch_size=BATCH_SIZE,
          epochs=MAX_EPOCH,
          verbose=1,
          validation_data=(x_test_VGG, y_test_VGG))
score = model.evaluate(x_test_VGG, y_test_VGG, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

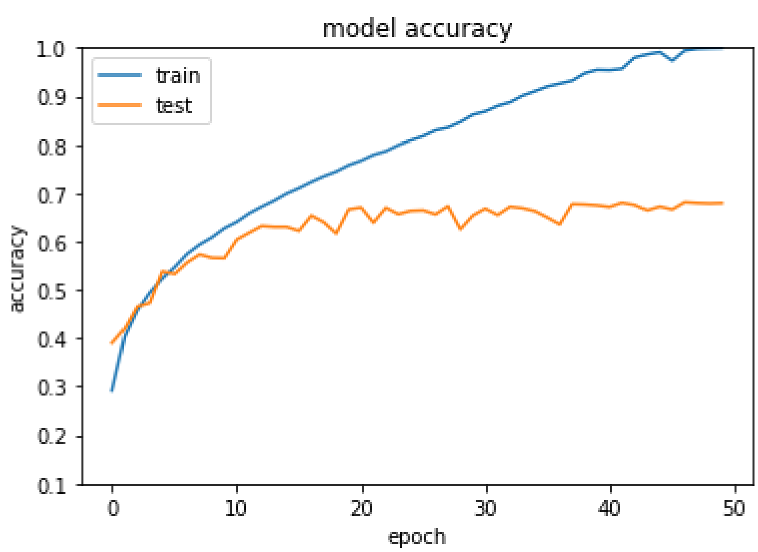
```
Train on 50000 samples, validate on 10000 samples
Epoch 1/50
50000/50000 [==============================] - 12s 248us/step - loss:
1.9752 - acc: 0.2917 - val_loss: 1.7017 - val_acc: 0.3904
Epoch 2/50
50000/50000 [==============================] - 11s 214us/step - loss:
1.6618 - acc: 0.4039 - val_loss: 1.6621 - val_acc: 0.4201
Epoch 3/50
50000/50000 [==============================] - 10s 209us/step - loss:
1.5181 - acc: 0.4576 - val_loss: 1.4824 - val_acc: 0.4653
Epoch 4/50
50000/50000 [==============================] - 10s 209us/step - loss:
1.4184 - acc: 0.4944 - val_loss: 1.4592 - val_acc: 0.4728
Epoch 5/50
50000/50000 [==============================] - 10s 204us/step - loss:
1.3378 - acc: 0.5233 - val_loss: 1.2845 - val_acc: 0.5382
Epoch 6/50
50000/50000 [==============================] - 10s 207us/step - loss:
1.2664 - acc: 0.5471 - val_loss: 1.2978 - val_acc: 0.5329
Epoch 7/50
50000/50000 [==============================] - 11s 214us/step - loss:
1.2033 - acc: 0.5744 - val_loss: 1.2471 - val_acc: 0.5566
Epoch 8/50
50000/50000 [==============================] - 11s 210us/step - loss:
1.1504 - acc: 0.5936 - val_loss: 1.1983 - val_acc: 0.5732
Epoch 9/50
50000/50000 [==============================] - 10s 205us/step - loss:
1.1069 - acc: 0.6085 - val_loss: 1.2177 - val_acc: 0.5662
Epoch 10/50
50000/50000 [==============================] - 10s 208us/step - loss:
1.0587 - acc: 0.6275 - val_loss: 1.2358 - val_acc: 0.5656
Epoch 11/50
50000/50000 [==============================] - 11s 213us/step - loss:
1.0218 - acc: 0.6406 - val_loss: 1.1479 - val_acc: 0.6039
Epoch 12/50
50000/50000 [==============================] - 10s 209us/step - loss:
0.9738 - acc: 0.6582 - val_loss: 1.0958 - val_acc: 0.6182
Epoch 13/50
50000/50000 [==============================] - 10s 208us/step - loss:
0.9376 - acc: 0.6718 - val_loss: 1.0436 - val_acc: 0.6321
Epoch 14/50
50000/50000 [==============================] - 11s 213us/step - loss:
0.9013 - acc: 0.6846 - val_loss: 1.0632 - val_acc: 0.6302
Epoch 15/50
50000/50000 [==============================] - 11s 211us/step - loss:
0.8587 - acc: 0.6992 - val_loss: 1.0733 - val_acc: 0.6302
Epoch 16/50
50000/50000 [==============================] - 10s 207us/step - loss:
0.8306 - acc: 0.7105 - val_loss: 1.0912 - val_acc: 0.6220
Epoch 17/50
50000/50000 [==============================] - 10s 207us/step - loss:
0.7961 - acc: 0.7233 - val_loss: 0.9963 - val_acc: 0.6534
Epoch 18/50
50000/50000 [==============================] - 10s 202us/step - loss:
0.7642 - acc: 0.7347 - val_loss: 1.0398 - val_acc: 0.6399
Epoch 19/50
50000/50000 [==============================] - 10s 210us/step - loss:
```

```
0.7348 - acc: 0.7444 - val_loss: 1.1518 - val_acc: 0.6165
Epoch 20/50
50000/50000 [==============================] - 11s 214us/step - loss:
0.7021 - acc: 0.7574 - val_loss: 0.9785 - val_acc: 0.6661
Epoch 21/50
50000/50000 [==============================] - 10s 208us/step - loss:
0.6703 - acc: 0.7667 - val_loss: 0.9700 - val_acc: 0.6702
Epoch 22/50
50000/50000 [==============================] - 10s 208us/step - loss:
0.6400 - acc: 0.7787 - val_loss: 1.0687 - val_acc: 0.6390
Epoch 23/50
50000/50000 [==============================] - 10s 205us/step - loss:
0.6118 - acc: 0.7863 - val_loss: 1.0030 - val_acc: 0.6695
Epoch 24/50
50000/50000 [==============================] - 10s 210us/step - loss:
0.5813 - acc: 0.7986 - val_loss: 1.0456 - val_acc: 0.6564
Epoch 25/50
50000/50000 [==============================] - 11s 214us/step - loss:
0.5497 - acc: 0.8090 - val_loss: 1.0500 - val_acc: 0.6632
Epoch 26/50
50000/50000 [==============================] - 10s 208us/step - loss:
0.5239 - acc: 0.8178 - val_loss: 1.0825 - val_acc: 0.6642
Epoch 27/50
50000/50000 [==============================] - 10s 210us/step - loss:
0.4946 - acc: 0.8297 - val_loss: 1.1117 - val_acc: 0.6559
Epoch 28/50
50000/50000 [==============================] - 10s 203us/step - loss:
0.4739 - acc: 0.8355 - val_loss: 1.0388 - val_acc: 0.6728
Epoch 29/50
50000/50000 [==============================] - 10s 204us/step - loss:
0.4409 - acc: 0.8470 - val_loss: 1.2532 - val_acc: 0.6253
Epoch 30/50
50000/50000 [==============================] - 10s 207us/step - loss:
0.4064 - acc: 0.8620 - val_loss: 1.1386 - val_acc: 0.6535
Epoch 31/50
50000/50000 [==============================] - 11s 216us/step - loss:
0.3856 - acc: 0.8686 - val_loss: 1.1013 - val_acc: 0.6678
Epoch 32/50
50000/50000 [==============================] - 11s 210us/step - loss:
0.3565 - acc: 0.8799 - val_loss: 1.1497 - val_acc: 0.6546
Epoch 33/50
50000/50000 [==============================] - 11s 212us/step - loss:
0.3298 - acc: 0.8874 - val_loss: 1.1369 - val_acc: 0.6718
Epoch 34/50
50000/50000 [==============================] - 11s 212us/step - loss:
0.3007 - acc: 0.9009 - val_loss: 1.2001 - val_acc: 0.6687
Epoch 35/50
50000/50000 [==============================] - 11s 212us/step - loss:
0.2783 - acc: 0.9100 - val_loss: 1.2343 - val_acc: 0.6626
Epoch 36/50
50000/50000 [==============================] - 10s 205us/step - loss:
0.2471 - acc: 0.9196 - val_loss: 1.3716 - val_acc: 0.6495
Epoch 37/50
50000/50000 [==============================] - 10s 204us/step - loss:
0.2304 - acc: 0.9258 - val_loss: 1.4500 - val_acc: 0.6355
Epoch 38/50
50000/50000 [==============================] - 10s 204us/step - loss:
```

```
0.2255 - acc: 0.9318 - val_loss: 1.2501 - val_acc: 0.6774
Epoch 39/50
50000/50000 [==============================] - 10s 204us/step - loss:
0.1734 - acc: 0.9472 - val_loss: 1.2839 - val_acc: 0.6765
Epoch 40/50
50000/50000 [==============================] - 11s 211us/step - loss:
0.1560 - acc: 0.9543 - val_loss: 1.3359 - val_acc: 0.6747
Epoch 41/50
50000/50000 [==============================] - 10s 205us/step - loss:
0.1806 - acc: 0.9535 - val_loss: 1.3756 - val_acc: 0.6714
Epoch 42/50
50000/50000 [==============================] - 10s 204us/step - loss:
0.1670 - acc: 0.9565 - val_loss: 1.3529 - val_acc: 0.6800
Epoch 43/50
50000/50000 [==============================] - 10s 205us/step - loss:
0.0889 - acc: 0.9795 - val_loss: 1.4415 - val_acc: 0.6755
Epoch 44/50
50000/50000 [==============================] - 11s 210us/step - loss:
0.0670 - acc: 0.9865 - val_loss: 1.5803 - val_acc: 0.6643
Epoch 45/50
50000/50000 [==============================] - 10s 202us/step - loss:
0.0526 - acc: 0.9910 - val_loss: 1.5749 - val_acc: 0.6721
Epoch 46/50
50000/50000 [==============================] - 11s 216us/step - loss:
0.1306 - acc: 0.9731 - val_loss: 1.4522 - val_acc: 0.6661
Epoch 47/50
50000/50000 [==============================] - 10s 205us/step - loss:
0.0436 - acc: 0.9944 - val_loss: 1.5846 - val_acc: 0.6812
Epoch 48/50
50000/50000 [==============================] - 10s 203us/step - loss:
0.0259 - acc: 0.9981 - val_loss: 1.6626 - val_acc: 0.6796
Epoch 49/50
50000/50000 [==============================] - 11s 215us/step - loss:
0.0198 - acc: 0.9990 - val_loss: 1.6924 - val_acc: 0.6787
Epoch 50/50
50000/50000 [==============================] - 11s 210us/step - loss:
0.0160 - acc: 0.9995 - val_loss: 1.7450 - val_acc: 0.6794
Test loss: 1.7449563205718994
Test accuracy: 0.6794
```

In [168]:
```
# summarize history for loss
plt.plot(history_SGD_1.history['loss'])
plt.plot(history_SGD_1.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.ylim(0,2.2)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for accuracy
plt.plot(history_SGD_1.history['acc'])
plt.plot(history_SGD_1.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.ylim(0.1,1)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





In [149]:
```
import keras
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from keras.models import Model
from keras.layers import Input, Dense, GlobalAveragePooling2D, Conv2D, M
axPooling2D, Flatten
import numpy as np

img_input = Input((32,32,3))
# Block 1
x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_c
onv1')(img_input)
x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_c
onv2')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block1_pool')(x)

# Block 2
x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_
conv1')(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_
conv2')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block2_pool')(x)

# Block 3
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_
conv1')(x)
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_
conv2')(x)
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_
conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block3_pool')(x)
'''
# Block 4
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_
conv1')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_
conv2')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_
conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block4_pool')(x)

# Block 5
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_
conv1')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_
conv2')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_
conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block5_pool')(x)
'''
# Final Block
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
x = Dense(32, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)
```

```
model = Model(inputs=img_input, outputs=predictions)
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_50 (InputLayer)        (None, 32, 32, 3)         0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 32, 32, 64)        1792      
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 32, 32, 64)        36928     
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0         
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 16, 16, 128)       73856     
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 16, 16, 128)       147584    
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0         
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 8, 8, 256)         295168    
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080    
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 8, 8, 256)         590080    
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0         
_________________________________________________________________
global_average_pooling2d_36  (None, 256)               0         
_________________________________________________________________
dense_127 (Dense)            (None, 128)               32896     
_________________________________________________________________
dense_128 (Dense)            (None, 32)                4128      
_________________________________________________________________
dense_129 (Dense)            (None, 10)                330       
=================================================================
Total params: 1,772,842
Trainable params: 1,772,842
Non-trainable params: 0
_________________________________________________________________
```

In [93]:
```
BATCH_SIZE = 128
MAX_EPOCH = 50
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.SGD(lr=1e-3),
              metrics=['accuracy'])
history_SGD_2 = model.fit(x_train_VGG, y_train_VGG,
          batch_size=BATCH_SIZE,
          epochs=MAX_EPOCH,
          verbose=1,
          validation_data=(x_test_VGG, y_test_VGG))
score = model.evaluate(x_test_VGG, y_test_VGG, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

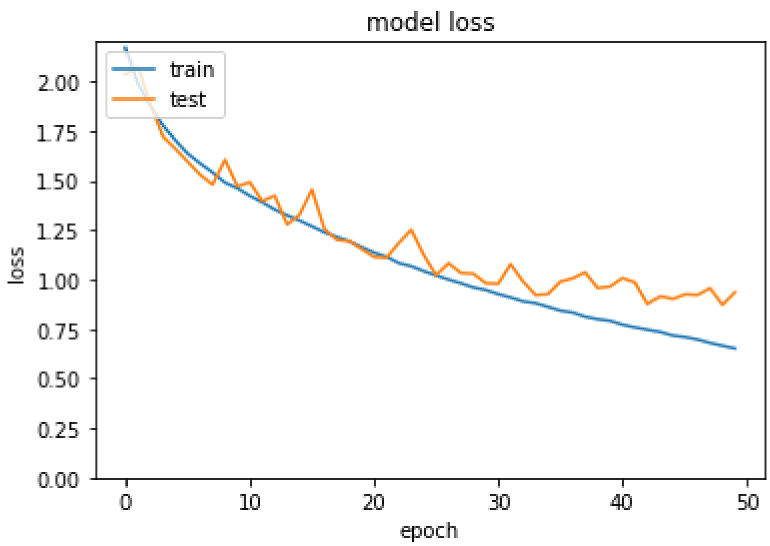
```
Train on 50000 samples, validate on 10000 samples
Epoch 1/50
50000/50000 [==============================] - 12s 236us/step - loss:
2.1698 - acc: 0.1813 - val_loss: 2.0338 - val_acc: 0.2514
Epoch 2/50
50000/50000 [==============================] - 10s 191us/step - loss:
1.9871 - acc: 0.2826 - val_loss: 2.0732 - val_acc: 0.2518
Epoch 3/50
50000/50000 [==============================] - 10s 194us/step - loss:
1.8747 - acc: 0.3281 - val_loss: 1.8847 - val_acc: 0.3191
Epoch 4/50
50000/50000 [==============================] - 9s 189us/step - loss: 1.
7769 - acc: 0.3587 - val_loss: 1.7226 - val_acc: 0.3710
Epoch 5/50
50000/50000 [==============================] - 10s 190us/step - loss:
1.7038 - acc: 0.3837 - val_loss: 1.6612 - val_acc: 0.3947
Epoch 6/50
50000/50000 [==============================] - 10s 198us/step - loss:
1.6365 - acc: 0.4097 - val_loss: 1.5955 - val_acc: 0.4206
Epoch 7/50
50000/50000 [==============================] - 10s 193us/step - loss:
1.5869 - acc: 0.4264 - val_loss: 1.5297 - val_acc: 0.4540
Epoch 8/50
50000/50000 [==============================] - 10s 192us/step - loss:
1.5411 - acc: 0.4420 - val_loss: 1.4800 - val_acc: 0.4619
Epoch 9/50
50000/50000 [==============================] - 10s 193us/step - loss:
1.4897 - acc: 0.4628 - val_loss: 1.6050 - val_acc: 0.4322
Epoch 10/50
50000/50000 [==============================] - 10s 197us/step - loss:
1.4610 - acc: 0.4739 - val_loss: 1.4712 - val_acc: 0.4629
Epoch 11/50
50000/50000 [==============================] - 10s 198us/step - loss:
1.4227 - acc: 0.4861 - val_loss: 1.4915 - val_acc: 0.4540
Epoch 12/50
50000/50000 [==============================] - 10s 193us/step - loss:
1.3902 - acc: 0.5000 - val_loss: 1.3962 - val_acc: 0.4965
Epoch 13/50
50000/50000 [==============================] - 9s 185us/step - loss: 1.
3535 - acc: 0.5115 - val_loss: 1.4236 - val_acc: 0.4827
Epoch 14/50
50000/50000 [==============================] - 10s 196us/step - loss:
1.3242 - acc: 0.5251 - val_loss: 1.2781 - val_acc: 0.5440
Epoch 15/50
50000/50000 [==============================] - 10s 193us/step - loss:
1.2976 - acc: 0.5351 - val_loss: 1.3272 - val_acc: 0.5198
Epoch 16/50
50000/50000 [==============================] - 10s 194us/step - loss:
1.2688 - acc: 0.5468 - val_loss: 1.4544 - val_acc: 0.4623
Epoch 17/50
50000/50000 [==============================] - 10s 191us/step - loss:
1.2383 - acc: 0.5593 - val_loss: 1.2560 - val_acc: 0.5571
Epoch 18/50
50000/50000 [==============================] - 10s 195us/step - loss:
1.2151 - acc: 0.5681 - val_loss: 1.2006 - val_acc: 0.5765
Epoch 19/50
50000/50000 [==============================] - 10s 193us/step - loss:
```

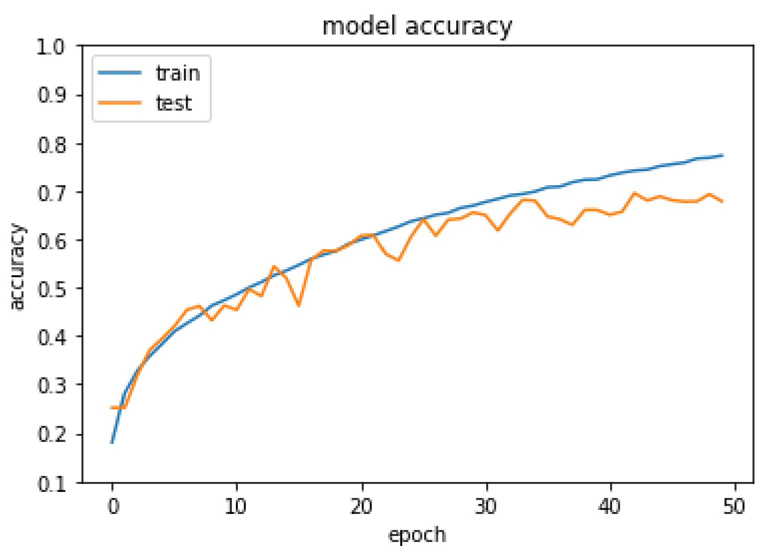
```
1.1941 - acc: 0.5755 - val_loss: 1.1932 - val_acc: 0.5758
Epoch 20/50
50000/50000 [==============================] - 10s 192us/step - loss:
1.1632 - acc: 0.5904 - val_loss: 1.1541 - val_acc: 0.5875
Epoch 21/50
50000/50000 [==============================] - 10s 192us/step - loss:
1.1356 - acc: 0.5994 - val_loss: 1.1136 - val_acc: 0.6071
Epoch 22/50
50000/50000 [==============================] - 9s 187us/step - loss: 1.
1146 - acc: 0.6075 - val_loss: 1.1093 - val_acc: 0.6089
Epoch 23/50
50000/50000 [==============================] - 9s 189us/step - loss: 1.
0826 - acc: 0.6171 - val_loss: 1.1850 - val_acc: 0.5690
Epoch 24/50
50000/50000 [==============================] - 10s 197us/step - loss:
1.0665 - acc: 0.6263 - val_loss: 1.2511 - val_acc: 0.5557
Epoch 25/50
50000/50000 [==============================] - 10s 194us/step - loss:
1.0422 - acc: 0.6370 - val_loss: 1.1241 - val_acc: 0.6054
Epoch 26/50
50000/50000 [==============================] - 10s 199us/step - loss:
1.0199 - acc: 0.6433 - val_loss: 1.0213 - val_acc: 0.6413
Epoch 27/50
50000/50000 [==============================] - 10s 194us/step - loss:
0.9993 - acc: 0.6505 - val_loss: 1.0818 - val_acc: 0.6072
Epoch 28/50
50000/50000 [==============================] - 10s 196us/step - loss:
0.9805 - acc: 0.6545 - val_loss: 1.0325 - val_acc: 0.6404
Epoch 29/50
50000/50000 [==============================] - 10s 196us/step - loss:
0.9595 - acc: 0.6648 - val_loss: 1.0289 - val_acc: 0.6426
Epoch 30/50
50000/50000 [==============================] - 10s 200us/step - loss:
0.9464 - acc: 0.6694 - val_loss: 0.9805 - val_acc: 0.6554
Epoch 31/50
50000/50000 [==============================] - 10s 196us/step - loss:
0.9257 - acc: 0.6770 - val_loss: 0.9781 - val_acc: 0.6498
Epoch 32/50
50000/50000 [==============================] - 10s 199us/step - loss:
0.9087 - acc: 0.6834 - val_loss: 1.0768 - val_acc: 0.6180
Epoch 33/50
50000/50000 [==============================] - 10s 195us/step - loss:
0.8891 - acc: 0.6904 - val_loss: 0.9895 - val_acc: 0.6535
Epoch 34/50
50000/50000 [==============================] - 10s 201us/step - loss:
0.8799 - acc: 0.6935 - val_loss: 0.9213 - val_acc: 0.6815
Epoch 35/50
50000/50000 [==============================] - 10s 192us/step - loss:
0.8611 - acc: 0.6986 - val_loss: 0.9255 - val_acc: 0.6796
Epoch 36/50
50000/50000 [==============================] - 10s 193us/step - loss:
0.8421 - acc: 0.7073 - val_loss: 0.9887 - val_acc: 0.6469
Epoch 37/50
50000/50000 [==============================] - 10s 195us/step - loss:
0.8322 - acc: 0.7087 - val_loss: 1.0056 - val_acc: 0.6412
Epoch 38/50
50000/50000 [==============================] - 10s 195us/step - loss:
```

```
0.8114 - acc: 0.7178 - val_loss: 1.0355 - val_acc: 0.6296
Epoch 39/50
50000/50000 [==============================] - 10s 197us/step - loss:
0.7989 - acc: 0.7230 - val_loss: 0.9568 - val_acc: 0.6606
Epoch 40/50
50000/50000 [==============================] - 10s 196us/step - loss:
0.7905 - acc: 0.7238 - val_loss: 0.9639 - val_acc: 0.6604
Epoch 41/50
50000/50000 [==============================] - 10s 194us/step - loss:
0.7710 - acc: 0.7318 - val_loss: 1.0069 - val_acc: 0.6506
Epoch 42/50
50000/50000 [==============================] - 10s 196us/step - loss:
0.7575 - acc: 0.7375 - val_loss: 0.9849 - val_acc: 0.6569
Epoch 43/50
50000/50000 [==============================] - 10s 206us/step - loss:
0.7457 - acc: 0.7419 - val_loss: 0.8761 - val_acc: 0.6954
Epoch 44/50
50000/50000 [==============================] - 10s 199us/step - loss:
0.7340 - acc: 0.7440 - val_loss: 0.9148 - val_acc: 0.6801
Epoch 45/50
50000/50000 [==============================] - 10s 197us/step - loss:
0.7163 - acc: 0.7511 - val_loss: 0.9015 - val_acc: 0.6886
Epoch 46/50
50000/50000 [==============================] - 10s 195us/step - loss:
0.7084 - acc: 0.7550 - val_loss: 0.9246 - val_acc: 0.6807
Epoch 47/50
50000/50000 [==============================] - 10s 196us/step - loss:
0.6960 - acc: 0.7585 - val_loss: 0.9215 - val_acc: 0.6777
Epoch 48/50
50000/50000 [==============================] - 10s 198us/step - loss:
0.6788 - acc: 0.7668 - val_loss: 0.9556 - val_acc: 0.6783
Epoch 49/50
50000/50000 [==============================] - 10s 202us/step - loss:
0.6645 - acc: 0.7685 - val_loss: 0.8713 - val_acc: 0.6931
Epoch 50/50
50000/50000 [==============================] - 10s 199us/step - loss:
0.6510 - acc: 0.7730 - val_loss: 0.9348 - val_acc: 0.6785
Test loss: 0.9347507353782654
Test accuracy: 0.6785
```

In [169]:

```
# summarize history for loss
plt.plot(history_SGD_2.history['loss'])
plt.plot(history_SGD_2.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.ylim(0,2.2)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for accuracy
plt.plot(history_SGD_2.history['acc'])
plt.plot(history_SGD_2.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.ylim(0.1,1)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





In [120]:
```
import keras
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from keras.models import Model
from keras.layers import Input, Dense, GlobalAveragePooling2D, Conv2D, M
axPooling2D, BatchNormalization
import numpy as np

img_input = Input((32,32,3))
# Block 1
x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_c
onv1')(img_input)
x = BatchNormalization()(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_c
onv2')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block1_pool')(x)

# Block 2
x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_
conv1')(x)
x = BatchNormalization()(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_
conv2')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block2_pool')(x)

# Block 3
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_
conv1')(x)
x = BatchNormalization()(x)
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_
conv2')(x)
x = BatchNormalization()(x)
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_
conv3')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block3_pool')(x)
'''
# Block 4
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_
conv1')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_
conv2')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_
conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block4_pool')(x)

# Block 5
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_
conv1')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_
conv2')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_
conv3')(x)
```

```
x = MaxPooling2D((2, 2), strides=(2, 2), name='block5_pool')(x)
'''
# Final Block
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
x = Dense(32, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

model = Model(inputs=img_input, outputs=predictions)
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_42 (InputLayer)        (None, 32, 32, 3)         0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 32, 32, 64)        1792      
_________________________________________________________________
batch_normalization_140 (Bat (None, 32, 32, 64)        256       
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 32, 32, 64)        36928     
_________________________________________________________________
batch_normalization_141 (Bat (None, 32, 32, 64)        256       
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0         
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 16, 16, 128)       73856     
_________________________________________________________________
batch_normalization_142 (Bat (None, 16, 16, 128)       512       
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 16, 16, 128)       147584    
_________________________________________________________________
batch_normalization_143 (Bat (None, 16, 16, 128)       512       
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0         
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 8, 8, 256)         295168    
_________________________________________________________________
batch_normalization_144 (Bat (None, 8, 8, 256)         1024      
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080    
_________________________________________________________________
batch_normalization_145 (Bat (None, 8, 8, 256)         1024      
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 8, 8, 256)         590080    
_________________________________________________________________
batch_normalization_146 (Bat (None, 8, 8, 256)         1024      
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0         
_________________________________________________________________
global_average_pooling2d_29  (None, 256)               0         
_________________________________________________________________
dense_103 (Dense)            (None, 128)               32896     
_________________________________________________________________
dense_104 (Dense)            (None, 32)                4128      
_________________________________________________________________
dense_105 (Dense)            (None, 10)                330       
=================================================================
Total params: 1,777,450
Trainable params: 1,775,146
Non-trainable params: 2,304
_________________________________________________________________
```

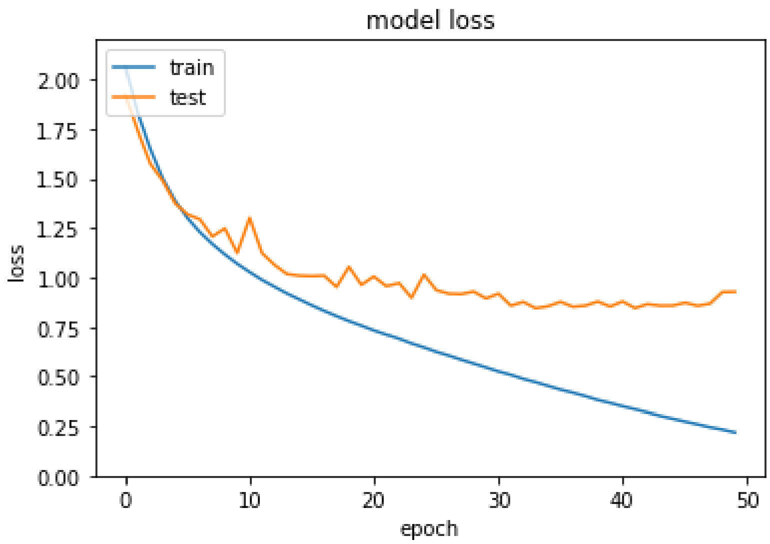
In [95]:
```
BATCH_SIZE = 128
MAX_EPOCH = 50
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.SGD(lr=1e-3),
              metrics=['accuracy'])
history_SGD_3 = model.fit(x_train_VGG, y_train_VGG,
          batch_size=BATCH_SIZE,
          epochs=MAX_EPOCH,
          verbose=1,
          validation_data=(x_test_VGG, y_test_VGG))
score = model.evaluate(x_test_VGG, y_test_VGG, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

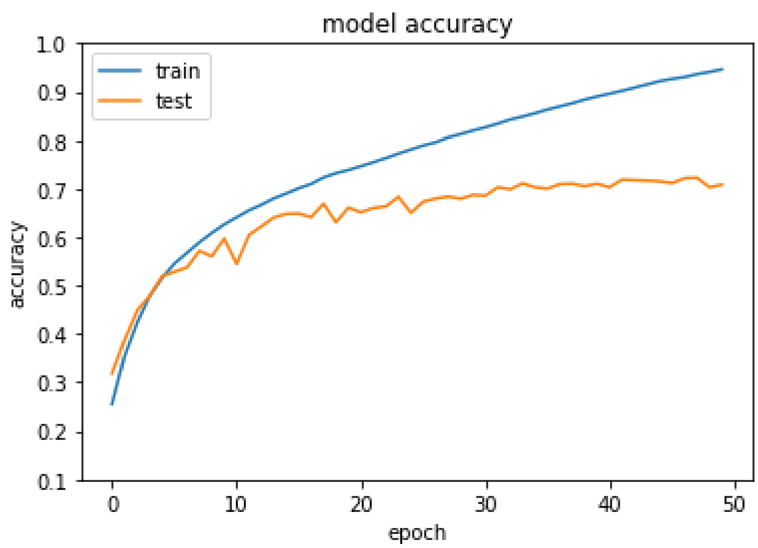
```
Train on 50000 samples, validate on 10000 samples
Epoch 1/50
50000/50000 [==============================] - 13s 267us/step - loss:
2.0653 - acc: 0.2551 - val_loss: 1.9189 - val_acc: 0.3186
Epoch 2/50
50000/50000 [==============================] - 11s 212us/step - loss:
1.8270 - acc: 0.3557 - val_loss: 1.7386 - val_acc: 0.3887
Epoch 3/50
50000/50000 [==============================] - 11s 219us/step - loss:
1.6504 - acc: 0.4240 - val_loss: 1.5751 - val_acc: 0.4490
Epoch 4/50
50000/50000 [==============================] - 11s 217us/step - loss:
1.5010 - acc: 0.4786 - val_loss: 1.4890 - val_acc: 0.4786
Epoch 5/50
50000/50000 [==============================] - 11s 216us/step - loss:
1.3882 - acc: 0.5161 - val_loss: 1.3747 - val_acc: 0.5193
Epoch 6/50
50000/50000 [==============================] - 11s 215us/step - loss:
1.2998 - acc: 0.5458 - val_loss: 1.3190 - val_acc: 0.5288
Epoch 7/50
50000/50000 [==============================] - 11s 214us/step - loss:
1.2288 - acc: 0.5677 - val_loss: 1.2938 - val_acc: 0.5375
Epoch 8/50
50000/50000 [==============================] - 11s 220us/step - loss:
1.1690 - acc: 0.5895 - val_loss: 1.2071 - val_acc: 0.5718
Epoch 9/50
50000/50000 [==============================] - 11s 217us/step - loss:
1.1165 - acc: 0.6088 - val_loss: 1.2479 - val_acc: 0.5607
Epoch 10/50
50000/50000 [==============================] - 11s 220us/step - loss:
1.0692 - acc: 0.6265 - val_loss: 1.1236 - val_acc: 0.5972
Epoch 11/50
50000/50000 [==============================] - 11s 216us/step - loss:
1.0271 - acc: 0.6412 - val_loss: 1.3015 - val_acc: 0.5448
Epoch 12/50
50000/50000 [==============================] - 11s 213us/step - loss:
0.9874 - acc: 0.6557 - val_loss: 1.1231 - val_acc: 0.6048
Epoch 13/50
50000/50000 [==============================] - 11s 213us/step - loss:
0.9529 - acc: 0.6671 - val_loss: 1.0636 - val_acc: 0.6225
Epoch 14/50
50000/50000 [==============================] - 11s 223us/step - loss:
0.9190 - acc: 0.6804 - val_loss: 1.0173 - val_acc: 0.6410
Epoch 15/50
50000/50000 [==============================] - 11s 218us/step - loss:
0.8892 - acc: 0.6905 - val_loss: 1.0088 - val_acc: 0.6483
Epoch 16/50
50000/50000 [==============================] - 11s 216us/step - loss:
0.8590 - acc: 0.7012 - val_loss: 1.0066 - val_acc: 0.6490
Epoch 17/50
50000/50000 [==============================] - 11s 216us/step - loss:
0.8309 - acc: 0.7102 - val_loss: 1.0090 - val_acc: 0.6415
Epoch 18/50
50000/50000 [==============================] - 11s 216us/step - loss:
0.8043 - acc: 0.7235 - val_loss: 0.9519 - val_acc: 0.6694
Epoch 19/50
50000/50000 [==============================] - 11s 218us/step - loss:
```

```
0.7792 - acc: 0.7324 - val_loss: 1.0536 - val_acc: 0.6307
Epoch 20/50
50000/50000 [==============================] - 11s 215us/step - loss:
0.7561 - acc: 0.7388 - val_loss: 0.9626 - val_acc: 0.6612
Epoch 21/50
50000/50000 [==============================] - 11s 212us/step - loss:
0.7320 - acc: 0.7467 - val_loss: 1.0048 - val_acc: 0.6516
Epoch 22/50
50000/50000 [==============================] - 11s 219us/step - loss:
0.7108 - acc: 0.7548 - val_loss: 0.9563 - val_acc: 0.6602
Epoch 23/50
50000/50000 [==============================] - 11s 215us/step - loss:
0.6899 - acc: 0.7635 - val_loss: 0.9713 - val_acc: 0.6639
Epoch 24/50
50000/50000 [==============================] - 11s 217us/step - loss:
0.6662 - acc: 0.7728 - val_loss: 0.8973 - val_acc: 0.6836
Epoch 25/50
50000/50000 [==============================] - 11s 216us/step - loss:
0.6460 - acc: 0.7812 - val_loss: 1.0141 - val_acc: 0.6505
Epoch 26/50
50000/50000 [==============================] - 11s 216us/step - loss:
0.6236 - acc: 0.7894 - val_loss: 0.9356 - val_acc: 0.6736
Epoch 27/50
50000/50000 [==============================] - 11s 220us/step - loss:
0.6047 - acc: 0.7960 - val_loss: 0.9178 - val_acc: 0.6801
Epoch 28/50
50000/50000 [==============================] - 11s 220us/step - loss:
0.5839 - acc: 0.8058 - val_loss: 0.9160 - val_acc: 0.6842
Epoch 29/50
50000/50000 [==============================] - 11s 217us/step - loss:
0.5644 - acc: 0.8126 - val_loss: 0.9283 - val_acc: 0.6798
Epoch 30/50
50000/50000 [==============================] - 11s 217us/step - loss:
0.5447 - acc: 0.8198 - val_loss: 0.8938 - val_acc: 0.6874
Epoch 31/50
50000/50000 [==============================] - 11s 215us/step - loss:
0.5244 - acc: 0.8265 - val_loss: 0.9177 - val_acc: 0.6859
Epoch 32/50
50000/50000 [==============================] - 11s 220us/step - loss:
0.5073 - acc: 0.8341 - val_loss: 0.8572 - val_acc: 0.7034
Epoch 33/50
50000/50000 [==============================] - 11s 217us/step - loss:
0.4869 - acc: 0.8424 - val_loss: 0.8759 - val_acc: 0.6992
Epoch 34/50
50000/50000 [==============================] - 11s 218us/step - loss:
0.4702 - acc: 0.8486 - val_loss: 0.8441 - val_acc: 0.7114
Epoch 35/50
50000/50000 [==============================] - 11s 216us/step - loss:
0.4515 - acc: 0.8554 - val_loss: 0.8544 - val_acc: 0.7033
Epoch 36/50
50000/50000 [==============================] - 11s 223us/step - loss:
0.4330 - acc: 0.8630 - val_loss: 0.8765 - val_acc: 0.7006
Epoch 37/50
50000/50000 [==============================] - 11s 220us/step - loss:
0.4173 - acc: 0.8695 - val_loss: 0.8517 - val_acc: 0.7099
Epoch 38/50
50000/50000 [==============================] - 11s 221us/step - loss:
```

```
0.3999 - acc: 0.8760 - val_loss: 0.8574 - val_acc: 0.7107
Epoch 39/50
50000/50000 [==============================] - 11s 222us/step - loss:
0.3808 - acc: 0.8835 - val_loss: 0.8778 - val_acc: 0.7056
Epoch 40/50
50000/50000 [==============================] - 11s 222us/step - loss:
0.3657 - acc: 0.8900 - val_loss: 0.8529 - val_acc: 0.7106
Epoch 41/50
50000/50000 [==============================] - 11s 217us/step - loss:
0.3491 - acc: 0.8957 - val_loss: 0.8783 - val_acc: 0.7033
Epoch 42/50
50000/50000 [==============================] - 11s 225us/step - loss:
0.3342 - acc: 0.9017 - val_loss: 0.8450 - val_acc: 0.7191
Epoch 43/50
50000/50000 [==============================] - 11s 214us/step - loss:
0.3185 - acc: 0.9080 - val_loss: 0.8649 - val_acc: 0.7181
Epoch 44/50
50000/50000 [==============================] - 11s 219us/step - loss:
0.3010 - acc: 0.9149 - val_loss: 0.8572 - val_acc: 0.7171
Epoch 45/50
50000/50000 [==============================] - 11s 218us/step - loss:
0.2869 - acc: 0.9216 - val_loss: 0.8575 - val_acc: 0.7155
Epoch 46/50
50000/50000 [==============================] - 11s 218us/step - loss:
0.2725 - acc: 0.9263 - val_loss: 0.8722 - val_acc: 0.7122
Epoch 47/50
50000/50000 [==============================] - 11s 215us/step - loss:
0.2588 - acc: 0.9301 - val_loss: 0.8565 - val_acc: 0.7220
Epoch 48/50
50000/50000 [==============================] - 11s 219us/step - loss:
0.2442 - acc: 0.9361 - val_loss: 0.8675 - val_acc: 0.7231
Epoch 49/50
50000/50000 [==============================] - 11s 214us/step - loss:
0.2325 - acc: 0.9407 - val_loss: 0.9260 - val_acc: 0.7034
Epoch 50/50
50000/50000 [==============================] - 11s 215us/step - loss:
0.2180 - acc: 0.9459 - val_loss: 0.9274 - val_acc: 0.7087
Test loss: 0.9273944653511047
Test accuracy: 0.7087
```

In [170]:
```
# summarize history for loss
plt.plot(history_SGD_3.history['loss'])
plt.plot(history_SGD_3.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.ylim(0,2.2)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for accuracy
plt.plot(history_SGD_3.history['acc'])
plt.plot(history_SGD_3.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.ylim(0.1,1)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





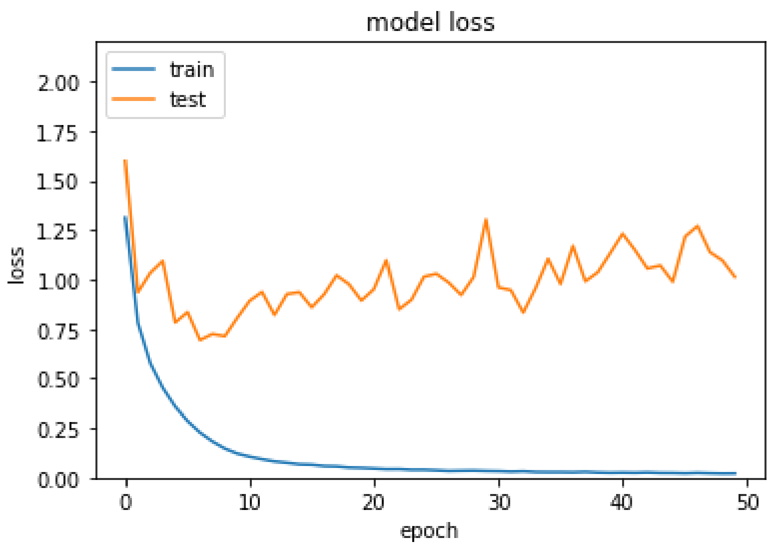
In [121]:
```
BATCH_SIZE = 128
MAX_EPOCH = 50
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.RMSprop(),
              metrics=['accuracy'])
history_RMSprop_nodrop = model.fit(x_train_VGG, y_train_VGG,
          batch_size=BATCH_SIZE,
          epochs=MAX_EPOCH,
          verbose=1,
          validation_data=(x_test_VGG, y_test_VGG))
score = model.evaluate(x_test_VGG, y_test_VGG, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

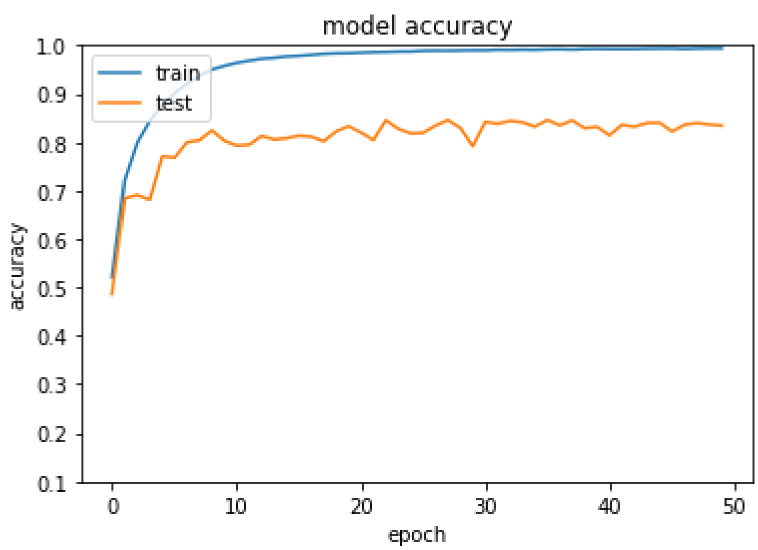
```
Train on 50000 samples, validate on 10000 samples
Epoch 1/50
50000/50000 [==============================] - 15s 295us/step - loss:
1.3123 - acc: 0.5213 - val_loss: 1.5995 - val_acc: 0.4860
Epoch 2/50
50000/50000 [==============================] - 12s 230us/step - loss:
0.7803 - acc: 0.7228 - val_loss: 0.9352 - val_acc: 0.6845
Epoch 3/50
50000/50000 [==============================] - 11s 229us/step - loss:
0.5757 - acc: 0.7998 - val_loss: 1.0337 - val_acc: 0.6909
Epoch 4/50
50000/50000 [==============================] - 12s 231us/step - loss:
0.4524 - acc: 0.8434 - val_loss: 1.0933 - val_acc: 0.6813
Epoch 5/50
50000/50000 [==============================] - 11s 223us/step - loss:
0.3579 - acc: 0.8771 - val_loss: 0.7822 - val_acc: 0.7710
Epoch 6/50
50000/50000 [==============================] - 11s 227us/step - loss:
0.2856 - acc: 0.9009 - val_loss: 0.8344 - val_acc: 0.7686
Epoch 7/50
50000/50000 [==============================] - 11s 225us/step - loss:
0.2277 - acc: 0.9211 - val_loss: 0.6931 - val_acc: 0.8003
Epoch 8/50
50000/50000 [==============================] - 11s 228us/step - loss:
0.1827 - acc: 0.9369 - val_loss: 0.7239 - val_acc: 0.8042
Epoch 9/50
50000/50000 [==============================] - 11s 222us/step - loss:
0.1470 - acc: 0.9500 - val_loss: 0.7129 - val_acc: 0.8241
Epoch 10/50
50000/50000 [==============================] - 11s 224us/step - loss:
0.1216 - acc: 0.9575 - val_loss: 0.8054 - val_acc: 0.8032
Epoch 11/50
50000/50000 [==============================] - 11s 228us/step - loss:
0.1060 - acc: 0.9637 - val_loss: 0.8918 - val_acc: 0.7932
Epoch 12/50
50000/50000 [==============================] - 11s 229us/step - loss:
0.0931 - acc: 0.9682 - val_loss: 0.9365 - val_acc: 0.7951
Epoch 13/50
50000/50000 [==============================] - 11s 225us/step - loss:
0.0820 - acc: 0.9722 - val_loss: 0.8201 - val_acc: 0.8127
Epoch 14/50
50000/50000 [==============================] - 11s 224us/step - loss:
0.0753 - acc: 0.9741 - val_loss: 0.9258 - val_acc: 0.8059
Epoch 15/50
50000/50000 [==============================] - 11s 226us/step - loss:
0.0675 - acc: 0.9769 - val_loss: 0.9355 - val_acc: 0.8083
Epoch 16/50
50000/50000 [==============================] - 11s 227us/step - loss:
0.0655 - acc: 0.9783 - val_loss: 0.8589 - val_acc: 0.8133
Epoch 17/50
50000/50000 [==============================] - 11s 225us/step - loss:
0.0584 - acc: 0.9801 - val_loss: 0.9247 - val_acc: 0.8113
Epoch 18/50
50000/50000 [==============================] - 11s 227us/step - loss:
0.0569 - acc: 0.9824 - val_loss: 1.0206 - val_acc: 0.8022
Epoch 19/50
50000/50000 [==============================] - 11s 225us/step - loss:
```

```
0.0508 - acc: 0.9834 - val_loss: 0.9758 - val_acc: 0.8221
Epoch 20/50
50000/50000 [==============================] - 11s 227us/step - loss:
0.0490 - acc: 0.9837 - val_loss: 0.8932 - val_acc: 0.8328
Epoch 21/50
50000/50000 [==============================] - 12s 230us/step - loss:
0.0462 - acc: 0.9847 - val_loss: 0.9501 - val_acc: 0.8197
Epoch 22/50
50000/50000 [==============================] - 11s 226us/step - loss:
0.0427 - acc: 0.9857 - val_loss: 1.0969 - val_acc: 0.8045
Epoch 23/50
50000/50000 [==============================] - 11s 221us/step - loss:
0.0431 - acc: 0.9861 - val_loss: 0.8489 - val_acc: 0.8451
Epoch 24/50
50000/50000 [==============================] - 11s 223us/step - loss:
0.0394 - acc: 0.9870 - val_loss: 0.8974 - val_acc: 0.8274
Epoch 25/50
50000/50000 [==============================] - 11s 224us/step - loss:
0.0395 - acc: 0.9872 - val_loss: 1.0136 - val_acc: 0.8182
Epoch 26/50
50000/50000 [==============================] - 11s 221us/step - loss:
0.0371 - acc: 0.9885 - val_loss: 1.0285 - val_acc: 0.8191
Epoch 27/50
50000/50000 [==============================] - 11s 227us/step - loss:
0.0338 - acc: 0.9890 - val_loss: 0.9837 - val_acc: 0.8343
Epoch 28/50
50000/50000 [==============================] - 11s 227us/step - loss:
0.0350 - acc: 0.9888 - val_loss: 0.9214 - val_acc: 0.8454
Epoch 29/50
50000/50000 [==============================] - 11s 225us/step - loss:
0.0357 - acc: 0.9892 - val_loss: 1.0120 - val_acc: 0.8291
Epoch 30/50
50000/50000 [==============================] - 11s 228us/step - loss:
0.0338 - acc: 0.9897 - val_loss: 1.3033 - val_acc: 0.7920
Epoch 31/50
50000/50000 [==============================] - 11s 223us/step - loss:
0.0328 - acc: 0.9895 - val_loss: 0.9596 - val_acc: 0.8416
Epoch 32/50
50000/50000 [==============================] - 11s 225us/step - loss:
0.0303 - acc: 0.9907 - val_loss: 0.9460 - val_acc: 0.8380
Epoch 33/50
50000/50000 [==============================] - 11s 230us/step - loss:
0.0321 - acc: 0.9904 - val_loss: 0.8325 - val_acc: 0.8437
Epoch 34/50
50000/50000 [==============================] - 11s 230us/step - loss:
0.0284 - acc: 0.9910 - val_loss: 0.9570 - val_acc: 0.8408
Epoch 35/50
50000/50000 [==============================] - 11s 229us/step - loss:
0.0279 - acc: 0.9906 - val_loss: 1.1044 - val_acc: 0.8321
Epoch 36/50
50000/50000 [==============================] - 12s 233us/step - loss:
0.0279 - acc: 0.9916 - val_loss: 0.9772 - val_acc: 0.8456
Epoch 37/50
50000/50000 [==============================] - 11s 227us/step - loss:
0.0271 - acc: 0.9919 - val_loss: 1.1696 - val_acc: 0.8348
Epoch 38/50
50000/50000 [==============================] - 11s 226us/step - loss:
```

```
0.0290 - acc: 0.9911 - val_loss: 0.9907 - val_acc: 0.8447
Epoch 39/50
50000/50000 [==============================] - 11s 229us/step - loss:
0.0260 - acc: 0.9923 - val_loss: 1.0360 - val_acc: 0.8289
Epoch 40/50
50000/50000 [==============================] - 12s 235us/step - loss:
0.0245 - acc: 0.9927 - val_loss: 1.1326 - val_acc: 0.8315
Epoch 41/50
50000/50000 [==============================] - 11s 225us/step - loss:
0.0254 - acc: 0.9923 - val_loss: 1.2312 - val_acc: 0.8142
Epoch 42/50
50000/50000 [==============================] - 11s 224us/step - loss:
0.0246 - acc: 0.9923 - val_loss: 1.1483 - val_acc: 0.8357
Epoch 43/50
50000/50000 [==============================] - 12s 231us/step - loss:
0.0263 - acc: 0.9923 - val_loss: 1.0552 - val_acc: 0.8317
Epoch 44/50
50000/50000 [==============================] - 11s 229us/step - loss:
0.0238 - acc: 0.9929 - val_loss: 1.0707 - val_acc: 0.8396
Epoch 45/50
50000/50000 [==============================] - 11s 227us/step - loss:
0.0236 - acc: 0.9932 - val_loss: 0.9883 - val_acc: 0.8398
Epoch 46/50
50000/50000 [==============================] - 11s 228us/step - loss:
0.0220 - acc: 0.9934 - val_loss: 1.2171 - val_acc: 0.8219
Epoch 47/50
50000/50000 [==============================] - 12s 230us/step - loss:
0.0241 - acc: 0.9926 - val_loss: 1.2704 - val_acc: 0.8359
Epoch 48/50
50000/50000 [==============================] - 11s 230us/step - loss:
0.0221 - acc: 0.9933 - val_loss: 1.1380 - val_acc: 0.8394
Epoch 49/50
50000/50000 [==============================] - 11s 224us/step - loss:
0.0209 - acc: 0.9938 - val_loss: 1.0960 - val_acc: 0.8363
Epoch 50/50
50000/50000 [==============================] - 11s 225us/step - loss:
0.0213 - acc: 0.9936 - val_loss: 1.0131 - val_acc: 0.8341
Test loss: 1.0131436536431313
Test accuracy: 0.8341
```

In [171]:
```
# summarize history for loss
plt.plot(history_RMSprop_nodrop.history['loss'])
plt.plot(history_RMSprop_nodrop.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.ylim(0,2.2)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for accuracy
plt.plot(history_RMSprop_nodrop.history['acc'])
plt.plot(history_RMSprop_nodrop.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.ylim(0.1,1)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





In [106]:
```
import keras
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from keras.models import Model
from keras.layers import Input, Dense, GlobalAveragePooling2D, Conv2D, M
axPooling2D, BatchNormalization, Dropout
import numpy as np

img_input = Input((32,32,3))
# Block 1
x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_c
onv1')(img_input)
x = BatchNormalization()(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_c
onv2')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block1_pool')(x)
x = Dropout(0.2)(x)

# Block 2
x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_
conv1')(x)
x = BatchNormalization()(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_
conv2')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block2_pool')(x)
x = Dropout(0.3)(x)

# Block 3
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_
conv1')(x)
x = BatchNormalization()(x)
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_
conv2')(x)
x = BatchNormalization()(x)
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_
conv3')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block3_pool')(x)
x = Dropout(0.4)(x)
'''
# Block 4
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_
conv1')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_
conv2')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_
conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block4_pool')(x)

# Block 5
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_
conv1')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_
```

```
conv2')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_
conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block5_pool')(x)
'''
# Final Block
x = GlobalAveragePooling2D()(x)
x = Dense(32, activation='relu')(x)
x = Dropout(0.4)(x)
predictions = Dense(10, activation='softmax')(x)

model = Model(inputs=img_input, outputs=predictions)
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_41 (InputLayer)        (None, 32, 32, 3)         0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 32, 32, 64)        1792      
_________________________________________________________________
batch_normalization_133 (Bat (None, 32, 32, 64)        256       
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 32, 32, 64)        36928     
_________________________________________________________________
batch_normalization_134 (Bat (None, 32, 32, 64)        256       
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0         
_________________________________________________________________
dropout_28 (Dropout)         (None, 16, 16, 64)        0         
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 16, 16, 128)       73856     
_________________________________________________________________
batch_normalization_135 (Bat (None, 16, 16, 128)       512       
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 16, 16, 128)       147584    
_________________________________________________________________
batch_normalization_136 (Bat (None, 16, 16, 128)       512       
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0         
_________________________________________________________________
dropout_29 (Dropout)         (None, 8, 8, 128)         0         
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 8, 8, 256)         295168    
_________________________________________________________________
batch_normalization_137 (Bat (None, 8, 8, 256)         1024      
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080    
_________________________________________________________________
batch_normalization_138 (Bat (None, 8, 8, 256)         1024      
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 8, 8, 256)         590080    
_________________________________________________________________
batch_normalization_139 (Bat (None, 8, 8, 256)         1024      
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0         
_________________________________________________________________
dropout_30 (Dropout)         (None, 4, 4, 256)         0         
_________________________________________________________________
global_average_pooling2d_28  (None, 256)               0         
_________________________________________________________________
dense_101 (Dense)            (None, 32)                8224      
_________________________________________________________________
dropout_31 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_102 (Dense)            (None, 10)                330       
=================================================================
Total params: 1,748,650
Trainable params: 1,746,346
Non-trainable params: 2,304
_________________________________________________________________
```

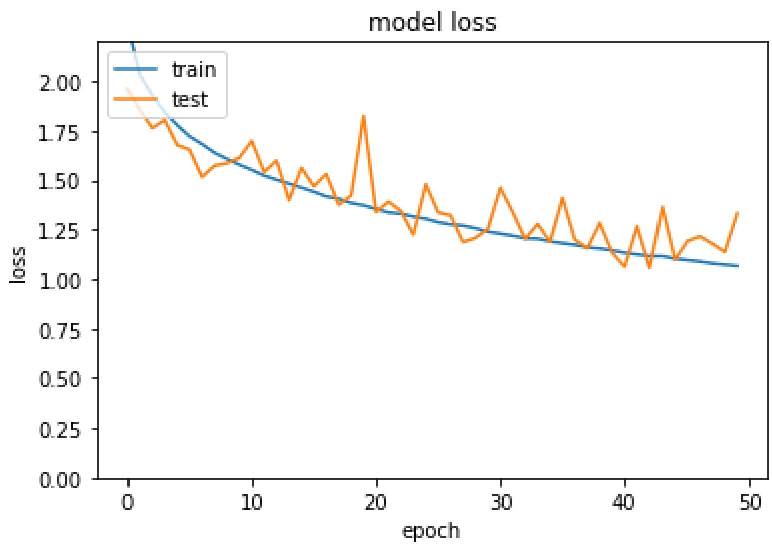
In [103]:
```
BATCH_SIZE = 128
MAX_EPOCH = 50
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.SGD(lr=1e-3),
              metrics=['accuracy'])
history_SGD_4 = model.fit(x_train_VGG, y_train_VGG,
          batch_size=BATCH_SIZE,
          epochs=MAX_EPOCH,
          verbose=1,
          validation_data=(x_test_VGG, y_test_VGG))
score = model.evaluate(x_test_VGG, y_test_VGG, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

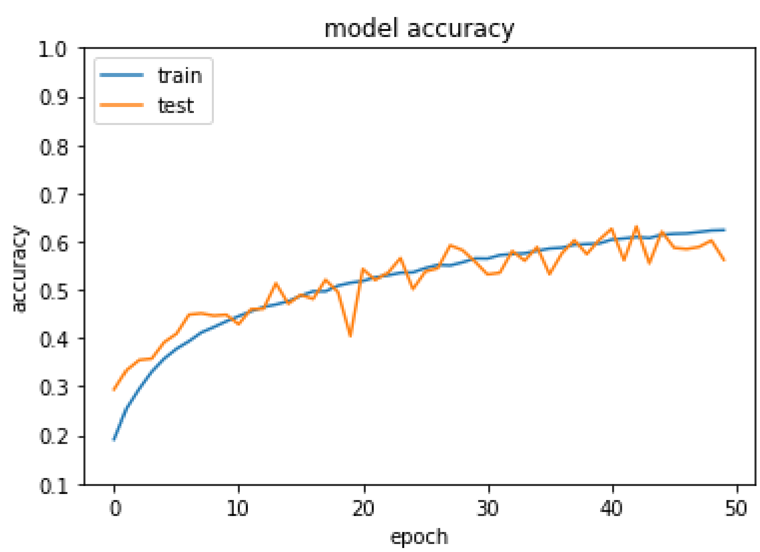
```
Train on 50000 samples, validate on 10000 samples
Epoch 1/50
50000/50000 [==============================] - 15s 301us/step - loss:
2.2862 - acc: 0.1914 - val_loss: 1.9612 - val_acc: 0.2939
Epoch 2/50
50000/50000 [==============================] - 12s 241us/step - loss:
2.0321 - acc: 0.2555 - val_loss: 1.8533 - val_acc: 0.3342
Epoch 3/50
50000/50000 [==============================] - 12s 241us/step - loss:
1.9277 - acc: 0.2953 - val_loss: 1.7633 - val_acc: 0.3548
Epoch 4/50
50000/50000 [==============================] - 12s 238us/step - loss:
1.8424 - acc: 0.3305 - val_loss: 1.8035 - val_acc: 0.3575
Epoch 5/50
50000/50000 [==============================] - 12s 242us/step - loss:
1.7782 - acc: 0.3575 - val_loss: 1.6776 - val_acc: 0.3914
Epoch 6/50
50000/50000 [==============================] - 12s 239us/step - loss:
1.7208 - acc: 0.3780 - val_loss: 1.6545 - val_acc: 0.4093
Epoch 7/50
50000/50000 [==============================] - 12s 241us/step - loss:
1.6820 - acc: 0.3936 - val_loss: 1.5162 - val_acc: 0.4487
Epoch 8/50
50000/50000 [==============================] - 12s 241us/step - loss:
1.6386 - acc: 0.4118 - val_loss: 1.5732 - val_acc: 0.4513
Epoch 9/50
50000/50000 [==============================] - 12s 242us/step - loss:
1.6075 - acc: 0.4229 - val_loss: 1.5855 - val_acc: 0.4465
Epoch 10/50
50000/50000 [==============================] - 12s 240us/step - loss:
1.5779 - acc: 0.4347 - val_loss: 1.6130 - val_acc: 0.4485
Epoch 11/50
50000/50000 [==============================] - 12s 238us/step - loss:
1.5516 - acc: 0.4454 - val_loss: 1.6987 - val_acc: 0.4287
Epoch 12/50
50000/50000 [==============================] - 12s 247us/step - loss:
1.5231 - acc: 0.4556 - val_loss: 1.5415 - val_acc: 0.4600
Epoch 13/50
50000/50000 [==============================] - 12s 243us/step - loss:
1.5020 - acc: 0.4643 - val_loss: 1.5993 - val_acc: 0.4606
Epoch 14/50
50000/50000 [==============================] - 12s 245us/step - loss:
1.4826 - acc: 0.4696 - val_loss: 1.3991 - val_acc: 0.5136
Epoch 15/50
50000/50000 [==============================] - 12s 241us/step - loss:
1.4623 - acc: 0.4765 - val_loss: 1.5623 - val_acc: 0.4711
Epoch 16/50
50000/50000 [==============================] - 12s 245us/step - loss:
1.4412 - acc: 0.4876 - val_loss: 1.4687 - val_acc: 0.4897
Epoch 17/50
50000/50000 [==============================] - 12s 242us/step - loss:
1.4177 - acc: 0.4970 - val_loss: 1.5310 - val_acc: 0.4811
Epoch 18/50
50000/50000 [==============================] - 12s 241us/step - loss:
1.4064 - acc: 0.4969 - val_loss: 1.3754 - val_acc: 0.5208
Epoch 19/50
50000/50000 [==============================] - 12s 240us/step - loss:
```

```
1.3845 - acc: 0.5085 - val_loss: 1.4238 - val_acc: 0.4955
Epoch 20/50
50000/50000 [==============================] - 12s 241us/step - loss:
1.3724 - acc: 0.5144 - val_loss: 1.8246 - val_acc: 0.4044
Epoch 21/50
50000/50000 [==============================] - 12s 246us/step - loss:
1.3544 - acc: 0.5183 - val_loss: 1.3383 - val_acc: 0.5432
Epoch 22/50
50000/50000 [==============================] - 12s 246us/step - loss:
1.3365 - acc: 0.5257 - val_loss: 1.3913 - val_acc: 0.5201
Epoch 23/50
50000/50000 [==============================] - 12s 243us/step - loss:
1.3299 - acc: 0.5300 - val_loss: 1.3439 - val_acc: 0.5356
Epoch 24/50
50000/50000 [==============================] - 12s 240us/step - loss:
1.3156 - acc: 0.5353 - val_loss: 1.2246 - val_acc: 0.5657
Epoch 25/50
50000/50000 [==============================] - 12s 247us/step - loss:
1.3045 - acc: 0.5362 - val_loss: 1.4800 - val_acc: 0.5014
Epoch 26/50
50000/50000 [==============================] - 12s 239us/step - loss:
1.2868 - acc: 0.5449 - val_loss: 1.3365 - val_acc: 0.5379
Epoch 27/50
50000/50000 [==============================] - 12s 239us/step - loss:
1.2759 - acc: 0.5515 - val_loss: 1.3220 - val_acc: 0.5447
Epoch 28/50
50000/50000 [==============================] - 12s 242us/step - loss:
1.2697 - acc: 0.5504 - val_loss: 1.1870 - val_acc: 0.5920
Epoch 29/50
50000/50000 [==============================] - 12s 243us/step - loss:
1.2563 - acc: 0.5570 - val_loss: 1.2085 - val_acc: 0.5820
Epoch 30/50
50000/50000 [==============================] - 12s 241us/step - loss:
1.2392 - acc: 0.5650 - val_loss: 1.2537 - val_acc: 0.5582
Epoch 31/50
50000/50000 [==============================] - 12s 239us/step - loss:
1.2289 - acc: 0.5646 - val_loss: 1.4619 - val_acc: 0.5319
Epoch 32/50
50000/50000 [==============================] - 12s 243us/step - loss:
1.2186 - acc: 0.5718 - val_loss: 1.3392 - val_acc: 0.5353
Epoch 33/50
50000/50000 [==============================] - 12s 241us/step - loss:
1.2058 - acc: 0.5741 - val_loss: 1.2020 - val_acc: 0.5802
Epoch 34/50
50000/50000 [==============================] - 12s 241us/step - loss:
1.2034 - acc: 0.5755 - val_loss: 1.2782 - val_acc: 0.5603
Epoch 35/50
50000/50000 [==============================] - 12s 240us/step - loss:
1.1898 - acc: 0.5808 - val_loss: 1.1882 - val_acc: 0.5885
Epoch 36/50
50000/50000 [==============================] - 12s 243us/step - loss:
1.1809 - acc: 0.5855 - val_loss: 1.4110 - val_acc: 0.5323
Epoch 37/50
50000/50000 [==============================] - 12s 244us/step - loss:
1.1726 - acc: 0.5872 - val_loss: 1.1987 - val_acc: 0.5755
Epoch 38/50
50000/50000 [==============================] - 12s 238us/step - loss:
```

```
1.1600 - acc: 0.5933 - val_loss: 1.1558 - val_acc: 0.6027
Epoch 39/50
50000/50000 [==============================] - 12s 243us/step - loss:
1.1542 - acc: 0.5948 - val_loss: 1.2833 - val_acc: 0.5739
Epoch 40/50
50000/50000 [==============================] - 12s 238us/step - loss:
1.1450 - acc: 0.5959 - val_loss: 1.1337 - val_acc: 0.6033
Epoch 41/50
50000/50000 [==============================] - 12s 247us/step - loss:
1.1324 - acc: 0.6039 - val_loss: 1.0621 - val_acc: 0.6264
Epoch 42/50
50000/50000 [==============================] - 12s 245us/step - loss:
1.1246 - acc: 0.6063 - val_loss: 1.2674 - val_acc: 0.5613
Epoch 43/50
50000/50000 [==============================] - 12s 246us/step - loss:
1.1163 - acc: 0.6092 - val_loss: 1.0566 - val_acc: 0.6312
Epoch 44/50
50000/50000 [==============================] - 12s 241us/step - loss:
1.1160 - acc: 0.6072 - val_loss: 1.3641 - val_acc: 0.5547
Epoch 45/50
50000/50000 [==============================] - 12s 244us/step - loss:
1.1031 - acc: 0.6141 - val_loss: 1.0987 - val_acc: 0.6205
Epoch 46/50
50000/50000 [==============================] - 12s 245us/step - loss:
1.0957 - acc: 0.6160 - val_loss: 1.1913 - val_acc: 0.5870
Epoch 47/50
50000/50000 [==============================] - 12s 245us/step - loss:
1.0886 - acc: 0.6167 - val_loss: 1.2157 - val_acc: 0.5844
Epoch 48/50
50000/50000 [==============================] - 12s 243us/step - loss:
1.0789 - acc: 0.6196 - val_loss: 1.1787 - val_acc: 0.5887
Epoch 49/50
50000/50000 [==============================] - 12s 242us/step - loss:
1.0727 - acc: 0.6228 - val_loss: 1.1375 - val_acc: 0.6022
Epoch 50/50
50000/50000 [==============================] - 12s 246us/step - loss:
1.0658 - acc: 0.6236 - val_loss: 1.3320 - val_acc: 0.5622
Test loss: 1.3320011768341065
Test accuracy: 0.5622
```

In [172]:
```
# summarize history for loss
plt.plot(history_SGD_4.history['loss'])
plt.plot(history_SGD_4.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.ylim(0,2.2)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for accuracy
plt.plot(history_SGD_4.history['acc'])
plt.plot(history_SGD_4.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.ylim(0.1,1)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





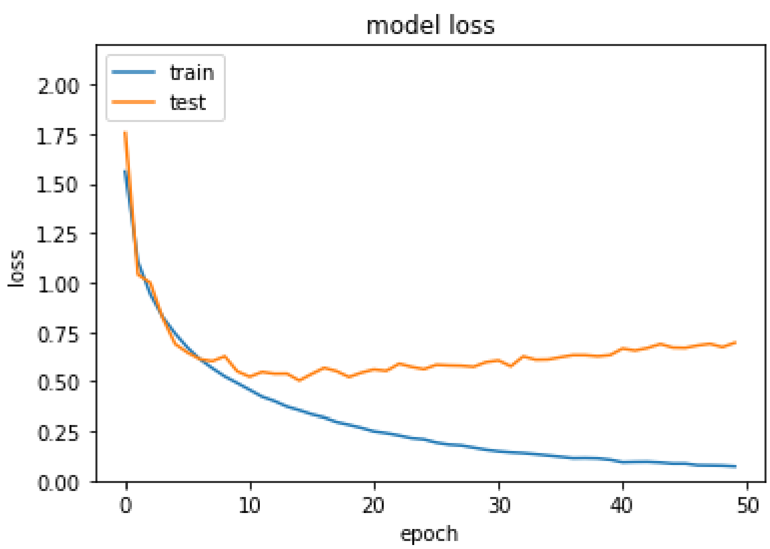
In [98]:
```
BATCH_SIZE = 128
MAX_EPOCH = 50
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adagrad(),
              metrics=['accuracy'])
history_Adagrad = model.fit(x_train_VGG, y_train_VGG,
          batch_size=BATCH_SIZE,
          epochs=MAX_EPOCH,
          verbose=1,
          validation_data=(x_test_VGG, y_test_VGG))
score = model.evaluate(x_test_VGG, y_test_VGG, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

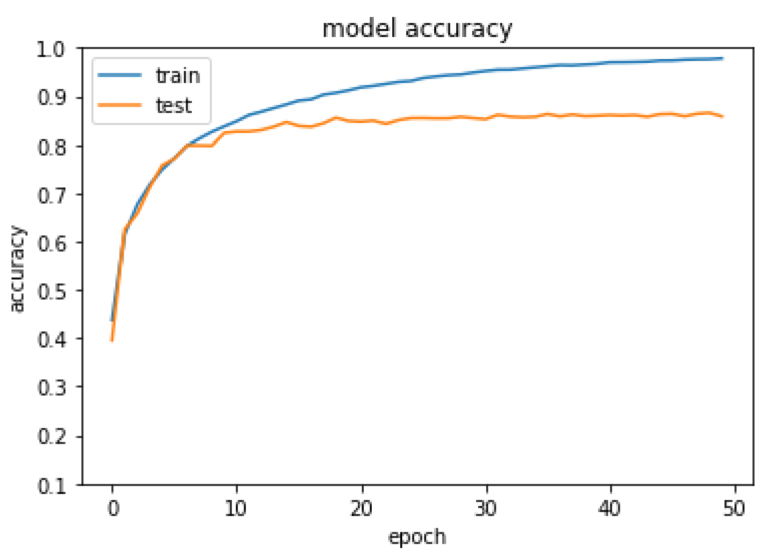
```
Train on 50000 samples, validate on 10000 samples
Epoch 1/50
50000/50000 [==============================] - 14s 288us/step - loss:
1.5609 - acc: 0.4375 - val_loss: 1.7541 - val_acc: 0.3955
Epoch 2/50
50000/50000 [==============================] - 12s 238us/step - loss:
1.1060 - acc: 0.6158 - val_loss: 1.0400 - val_acc: 0.6258
Epoch 3/50
50000/50000 [==============================] - 12s 238us/step - loss:
0.9401 - acc: 0.6768 - val_loss: 0.9982 - val_acc: 0.6581
Epoch 4/50
50000/50000 [==============================] - 12s 237us/step - loss:
0.8251 - acc: 0.7177 - val_loss: 0.8195 - val_acc: 0.7123
Epoch 5/50
50000/50000 [==============================] - 12s 238us/step - loss:
0.7415 - acc: 0.7489 - val_loss: 0.6885 - val_acc: 0.7575
Epoch 6/50
50000/50000 [==============================] - 12s 238us/step - loss:
0.6700 - acc: 0.7727 - val_loss: 0.6448 - val_acc: 0.7715
Epoch 7/50
50000/50000 [==============================] - 12s 240us/step - loss:
0.6114 - acc: 0.7975 - val_loss: 0.6118 - val_acc: 0.7984
Epoch 8/50
50000/50000 [==============================] - 12s 242us/step - loss:
0.5661 - acc: 0.8126 - val_loss: 0.6019 - val_acc: 0.7983
Epoch 9/50
50000/50000 [==============================] - 12s 239us/step - loss:
0.5244 - acc: 0.8262 - val_loss: 0.6264 - val_acc: 0.7978
Epoch 10/50
50000/50000 [==============================] - 12s 239us/step - loss:
0.4912 - acc: 0.8374 - val_loss: 0.5505 - val_acc: 0.8239
Epoch 11/50
50000/50000 [==============================] - 12s 241us/step - loss:
0.4571 - acc: 0.8479 - val_loss: 0.5229 - val_acc: 0.8268
Epoch 12/50
50000/50000 [==============================] - 12s 242us/step - loss:
0.4224 - acc: 0.8610 - val_loss: 0.5471 - val_acc: 0.8268
Epoch 13/50
50000/50000 [==============================] - 12s 239us/step - loss:
0.4002 - acc: 0.8677 - val_loss: 0.5382 - val_acc: 0.8299
Epoch 14/50
50000/50000 [==============================] - 12s 242us/step - loss:
0.3726 - acc: 0.8752 - val_loss: 0.5386 - val_acc: 0.8367
Epoch 15/50
50000/50000 [==============================] - 12s 239us/step - loss:
0.3540 - acc: 0.8825 - val_loss: 0.5038 - val_acc: 0.8461
Epoch 16/50
50000/50000 [==============================] - 12s 243us/step - loss:
0.3337 - acc: 0.8905 - val_loss: 0.5370 - val_acc: 0.8383
Epoch 17/50
50000/50000 [==============================] - 12s 240us/step - loss:
0.3181 - acc: 0.8932 - val_loss: 0.5678 - val_acc: 0.8365
Epoch 18/50
50000/50000 [==============================] - 12s 242us/step - loss:
0.2957 - acc: 0.9029 - val_loss: 0.5513 - val_acc: 0.8437
Epoch 19/50
50000/50000 [==============================] - 12s 242us/step - loss:
```

```
0.2817 - acc: 0.9067 - val_loss: 0.5216 - val_acc: 0.8553
Epoch 20/50
50000/50000 [==============================] - 12s 240us/step - loss:
0.2665 - acc: 0.9117 - val_loss: 0.5434 - val_acc: 0.8485
Epoch 21/50
50000/50000 [==============================] - 12s 241us/step - loss:
0.2487 - acc: 0.9180 - val_loss: 0.5597 - val_acc: 0.8471
Epoch 22/50
50000/50000 [==============================] - 12s 238us/step - loss:
0.2399 - acc: 0.9210 - val_loss: 0.5528 - val_acc: 0.8492
Epoch 23/50
50000/50000 [==============================] - 12s 243us/step - loss:
0.2279 - acc: 0.9252 - val_loss: 0.5891 - val_acc: 0.8426
Epoch 24/50
50000/50000 [==============================] - 12s 237us/step - loss:
0.2146 - acc: 0.9293 - val_loss: 0.5728 - val_acc: 0.8503
Epoch 25/50
50000/50000 [==============================] - 12s 235us/step - loss:
0.2086 - acc: 0.9312 - val_loss: 0.5611 - val_acc: 0.8544
Epoch 26/50
50000/50000 [==============================] - 12s 238us/step - loss:
0.1922 - acc: 0.9374 - val_loss: 0.5833 - val_acc: 0.8546
Epoch 27/50
50000/50000 [==============================] - 12s 238us/step - loss:
0.1822 - acc: 0.9405 - val_loss: 0.5801 - val_acc: 0.8539
Epoch 28/50
50000/50000 [==============================] - 12s 239us/step - loss:
0.1780 - acc: 0.9431 - val_loss: 0.5786 - val_acc: 0.8541
Epoch 29/50
50000/50000 [==============================] - 12s 238us/step - loss:
0.1668 - acc: 0.9445 - val_loss: 0.5734 - val_acc: 0.8570
Epoch 30/50
50000/50000 [==============================] - 12s 240us/step - loss:
0.1559 - acc: 0.9486 - val_loss: 0.5968 - val_acc: 0.8545
Epoch 31/50
50000/50000 [==============================] - 12s 243us/step - loss:
0.1481 - acc: 0.9518 - val_loss: 0.6054 - val_acc: 0.8521
Epoch 32/50
50000/50000 [==============================] - 12s 238us/step - loss:
0.1427 - acc: 0.9545 - val_loss: 0.5759 - val_acc: 0.8612
Epoch 33/50
50000/50000 [==============================] - 12s 239us/step - loss:
0.1391 - acc: 0.9546 - val_loss: 0.6258 - val_acc: 0.8574
Epoch 34/50
50000/50000 [==============================] - 12s 239us/step - loss:
0.1335 - acc: 0.9571 - val_loss: 0.6089 - val_acc: 0.8561
Epoch 35/50
50000/50000 [==============================] - 12s 240us/step - loss:
0.1271 - acc: 0.9594 - val_loss: 0.6101 - val_acc: 0.8570
Epoch 36/50
50000/50000 [==============================] - 12s 240us/step - loss:
0.1205 - acc: 0.9617 - val_loss: 0.6217 - val_acc: 0.8629
Epoch 37/50
50000/50000 [==============================] - 12s 239us/step - loss:
0.1135 - acc: 0.9639 - val_loss: 0.6323 - val_acc: 0.8580
Epoch 38/50
50000/50000 [==============================] - 12s 239us/step - loss:
```

```
0.1143 - acc: 0.9635 - val_loss: 0.6320 - val_acc: 0.8618
Epoch 39/50
50000/50000 [==============================] - 12s 240us/step - loss:
0.1126 - acc: 0.9651 - val_loss: 0.6266 - val_acc: 0.8585
Epoch 40/50
50000/50000 [==============================] - 12s 237us/step - loss:
0.1056 - acc: 0.9666 - val_loss: 0.6324 - val_acc: 0.8594
Epoch 41/50
50000/50000 [==============================] - 12s 238us/step - loss:
0.0928 - acc: 0.9698 - val_loss: 0.6656 - val_acc: 0.8607
Epoch 42/50
50000/50000 [==============================] - 12s 237us/step - loss:
0.0945 - acc: 0.9700 - val_loss: 0.6556 - val_acc: 0.8597
Epoch 43/50
50000/50000 [==============================] - 12s 238us/step - loss:
0.0951 - acc: 0.9705 - val_loss: 0.6680 - val_acc: 0.8607
Epoch 44/50
50000/50000 [==============================] - 12s 240us/step - loss:
0.0914 - acc: 0.9712 - val_loss: 0.6884 - val_acc: 0.8572
Epoch 45/50
50000/50000 [==============================] - 12s 240us/step - loss:
0.0861 - acc: 0.9733 - val_loss: 0.6704 - val_acc: 0.8625
Epoch 46/50
50000/50000 [==============================] - 12s 236us/step - loss:
0.0861 - acc: 0.9736 - val_loss: 0.6689 - val_acc: 0.8632
Epoch 47/50
50000/50000 [==============================] - 12s 238us/step - loss:
0.0778 - acc: 0.9753 - val_loss: 0.6811 - val_acc: 0.8582
Epoch 48/50
50000/50000 [==============================] - 12s 240us/step - loss:
0.0768 - acc: 0.9760 - val_loss: 0.6894 - val_acc: 0.8634
Epoch 49/50
50000/50000 [==============================] - 12s 240us/step - loss:
0.0756 - acc: 0.9764 - val_loss: 0.6727 - val_acc: 0.8653
Epoch 50/50
50000/50000 [==============================] - 12s 239us/step - loss:
0.0719 - acc: 0.9779 - val_loss: 0.6950 - val_acc: 0.8580
Test loss: 0.6949959478616714
Test accuracy: 0.858
```

In [173]:
```
# summarize history for loss
plt.plot(history_Adagrad.history['loss'])
plt.plot(history_Adagrad.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.ylim(0,2.2)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for accuracy
plt.plot(history_Adagrad.history['acc'])
plt.plot(history_Adagrad.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.ylim(0.1,1)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





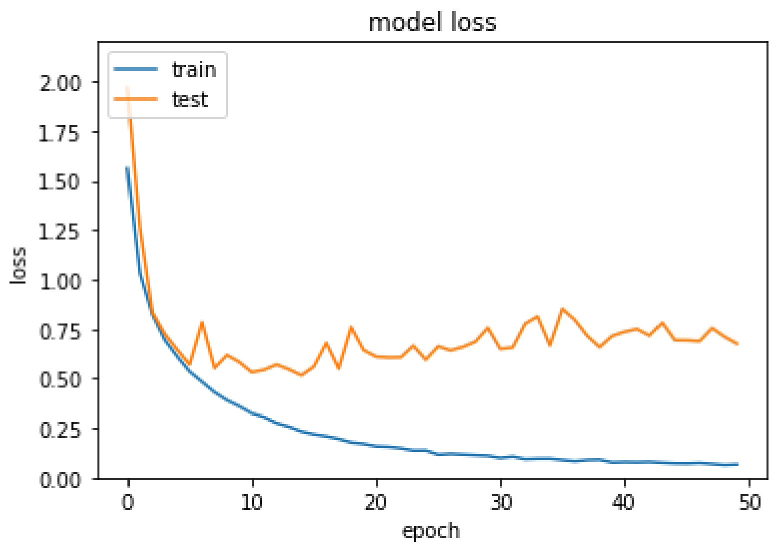
In [105]:
```
BATCH_SIZE = 128
MAX_EPOCH = 50
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Nadam(),
              metrics=['accuracy'])
history_Nadam = model.fit(x_train_VGG, y_train_VGG,
          batch_size=BATCH_SIZE,
          epochs=MAX_EPOCH,
          verbose=1,
          validation_data=(x_test_VGG, y_test_VGG))
score = model.evaluate(x_test_VGG, y_test_VGG, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

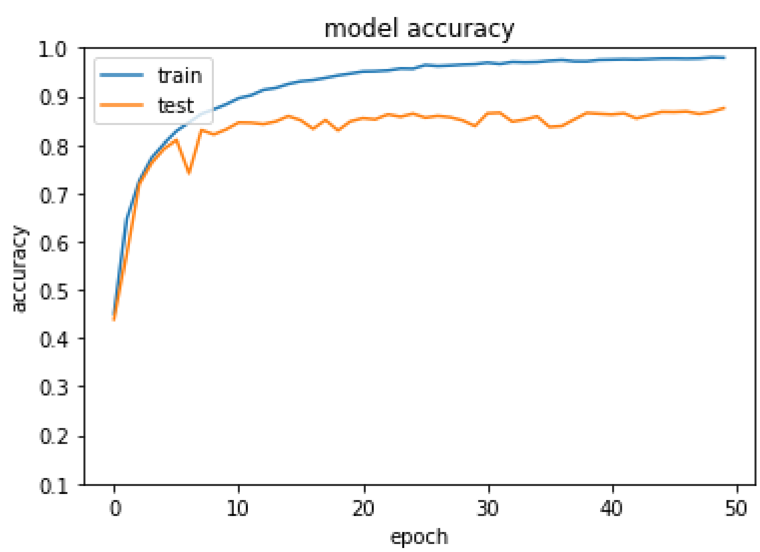
```
Train on 50000 samples, validate on 10000 samples
Epoch 1/50
50000/50000 [==============================] - 16s 322us/step - loss:
1.5625 - acc: 0.4508 - val_loss: 1.9672 - val_acc: 0.4389
Epoch 2/50
50000/50000 [==============================] - 12s 249us/step - loss:
1.0304 - acc: 0.6474 - val_loss: 1.2605 - val_acc: 0.5755
Epoch 3/50
50000/50000 [==============================] - 13s 253us/step - loss:
0.8213 - acc: 0.7260 - val_loss: 0.8337 - val_acc: 0.7186
Epoch 4/50
50000/50000 [==============================] - 13s 254us/step - loss:
0.6931 - acc: 0.7735 - val_loss: 0.7237 - val_acc: 0.7626
Epoch 5/50
50000/50000 [==============================] - 13s 256us/step - loss:
0.6096 - acc: 0.8024 - val_loss: 0.6445 - val_acc: 0.7918
Epoch 6/50
50000/50000 [==============================] - 13s 251us/step - loss:
0.5340 - acc: 0.8278 - val_loss: 0.5692 - val_acc: 0.8096
Epoch 7/50
50000/50000 [==============================] - 13s 254us/step - loss:
0.4824 - acc: 0.8454 - val_loss: 0.7835 - val_acc: 0.7410
Epoch 8/50
50000/50000 [==============================] - 12s 249us/step - loss:
0.4305 - acc: 0.8631 - val_loss: 0.5519 - val_acc: 0.8297
Epoch 9/50
50000/50000 [==============================] - 13s 254us/step - loss:
0.3902 - acc: 0.8726 - val_loss: 0.6172 - val_acc: 0.8205
Epoch 10/50
50000/50000 [==============================] - 13s 253us/step - loss:
0.3593 - acc: 0.8830 - val_loss: 0.5819 - val_acc: 0.8310
Epoch 11/50
50000/50000 [==============================] - 13s 251us/step - loss:
0.3242 - acc: 0.8955 - val_loss: 0.5308 - val_acc: 0.8452
Epoch 12/50
50000/50000 [==============================] - 13s 254us/step - loss:
0.3042 - acc: 0.9016 - val_loss: 0.5435 - val_acc: 0.8447
Epoch 13/50
50000/50000 [==============================] - 13s 251us/step - loss:
0.2739 - acc: 0.9128 - val_loss: 0.5698 - val_acc: 0.8419
Epoch 14/50
50000/50000 [==============================] - 13s 254us/step - loss:
0.2558 - acc: 0.9168 - val_loss: 0.5450 - val_acc: 0.8475
Epoch 15/50
50000/50000 [==============================] - 12s 249us/step - loss:
0.2322 - acc: 0.9249 - val_loss: 0.5153 - val_acc: 0.8585
Epoch 16/50
50000/50000 [==============================] - 13s 253us/step - loss:
0.2181 - acc: 0.9305 - val_loss: 0.5596 - val_acc: 0.8500
Epoch 17/50
50000/50000 [==============================] - 13s 252us/step - loss:
0.2080 - acc: 0.9330 - val_loss: 0.6785 - val_acc: 0.8322
Epoch 18/50
50000/50000 [==============================] - 12s 249us/step - loss:
0.1945 - acc: 0.9375 - val_loss: 0.5489 - val_acc: 0.8504
Epoch 19/50
50000/50000 [==============================] - 13s 254us/step - loss:
```

```
0.1771 - acc: 0.9426 - val_loss: 0.7583 - val_acc: 0.8288
Epoch 20/50
50000/50000 [==============================] - 13s 252us/step - loss:
0.1701 - acc: 0.9465 - val_loss: 0.6427 - val_acc: 0.8476
Epoch 21/50
50000/50000 [==============================] - 13s 255us/step - loss:
0.1578 - acc: 0.9507 - val_loss: 0.6095 - val_acc: 0.8542
Epoch 22/50
50000/50000 [==============================] - 13s 255us/step - loss:
0.1551 - acc: 0.9513 - val_loss: 0.6053 - val_acc: 0.8517
Epoch 23/50
50000/50000 [==============================] - 13s 254us/step - loss:
0.1478 - acc: 0.9530 - val_loss: 0.6065 - val_acc: 0.8621
Epoch 24/50
50000/50000 [==============================] - 13s 252us/step - loss:
0.1367 - acc: 0.9570 - val_loss: 0.6644 - val_acc: 0.8571
Epoch 25/50
50000/50000 [==============================] - 13s 251us/step - loss:
0.1373 - acc: 0.9565 - val_loss: 0.5943 - val_acc: 0.8636
Epoch 26/50
50000/50000 [==============================] - 13s 254us/step - loss:
0.1159 - acc: 0.9642 - val_loss: 0.6615 - val_acc: 0.8552
Epoch 27/50
50000/50000 [==============================] - 12s 247us/step - loss:
0.1199 - acc: 0.9620 - val_loss: 0.6411 - val_acc: 0.8590
Epoch 28/50
50000/50000 [==============================] - 12s 246us/step - loss:
0.1165 - acc: 0.9633 - val_loss: 0.6589 - val_acc: 0.8561
Epoch 29/50
50000/50000 [==============================] - 12s 245us/step - loss:
0.1133 - acc: 0.9650 - val_loss: 0.6851 - val_acc: 0.8496
Epoch 30/50
50000/50000 [==============================] - 13s 251us/step - loss:
0.1101 - acc: 0.9660 - val_loss: 0.7552 - val_acc: 0.8379
Epoch 31/50
50000/50000 [==============================] - 12s 248us/step - loss:
0.0992 - acc: 0.9691 - val_loss: 0.6487 - val_acc: 0.8642
Epoch 32/50
50000/50000 [==============================] - 13s 253us/step - loss:
0.1064 - acc: 0.9664 - val_loss: 0.6558 - val_acc: 0.8654
Epoch 33/50
50000/50000 [==============================] - 12s 248us/step - loss:
0.0931 - acc: 0.9706 - val_loss: 0.7756 - val_acc: 0.8471
Epoch 34/50
50000/50000 [==============================] - 12s 249us/step - loss:
0.0957 - acc: 0.9698 - val_loss: 0.8121 - val_acc: 0.8513
Epoch 35/50
50000/50000 [==============================] - 12s 249us/step - loss:
0.0959 - acc: 0.9704 - val_loss: 0.6664 - val_acc: 0.8580
Epoch 36/50
50000/50000 [==============================] - 13s 250us/step - loss:
0.0888 - acc: 0.9730 - val_loss: 0.8507 - val_acc: 0.8360
Epoch 37/50
50000/50000 [==============================] - 12s 247us/step - loss:
0.0827 - acc: 0.9747 - val_loss: 0.7951 - val_acc: 0.8380
Epoch 38/50
50000/50000 [==============================] - 12s 247us/step - loss:
```

```
0.0889 - acc: 0.9719 - val_loss: 0.7148 - val_acc: 0.8517
Epoch 39/50
50000/50000 [==============================] - 12s 241us/step - loss:
0.0905 - acc: 0.9718 - val_loss: 0.6577 - val_acc: 0.8651
Epoch 40/50
50000/50000 [==============================] - 12s 236us/step - loss:
0.0767 - acc: 0.9748 - val_loss: 0.7139 - val_acc: 0.8634
Epoch 41/50
50000/50000 [==============================] - 12s 249us/step - loss:
0.0786 - acc: 0.9754 - val_loss: 0.7351 - val_acc: 0.8618
Epoch 42/50
50000/50000 [==============================] - 13s 251us/step - loss:
0.0776 - acc: 0.9760 - val_loss: 0.7491 - val_acc: 0.8643
Epoch 43/50
50000/50000 [==============================] - 13s 252us/step - loss:
0.0792 - acc: 0.9756 - val_loss: 0.7160 - val_acc: 0.8537
Epoch 44/50
50000/50000 [==============================] - 13s 251us/step - loss:
0.0751 - acc: 0.9766 - val_loss: 0.7801 - val_acc: 0.8607
Epoch 45/50
50000/50000 [==============================] - 13s 252us/step - loss:
0.0719 - acc: 0.9776 - val_loss: 0.6928 - val_acc: 0.8673
Epoch 46/50
50000/50000 [==============================] - 12s 248us/step - loss:
0.0712 - acc: 0.9777 - val_loss: 0.6921 - val_acc: 0.8669
Epoch 47/50
50000/50000 [==============================] - 12s 248us/step - loss:
0.0738 - acc: 0.9773 - val_loss: 0.6879 - val_acc: 0.8681
Epoch 48/50
50000/50000 [==============================] - 12s 249us/step - loss:
0.0689 - acc: 0.9779 - val_loss: 0.7538 - val_acc: 0.8630
Epoch 49/50
50000/50000 [==============================] - 12s 248us/step - loss:
0.0637 - acc: 0.9800 - val_loss: 0.7102 - val_acc: 0.8673
Epoch 50/50
50000/50000 [==============================] - 12s 248us/step - loss:
0.0663 - acc: 0.9794 - val_loss: 0.6751 - val_acc: 0.8749
Test loss: 0.6751188353247941
Test accuracy: 0.8749
```

In [174]:
```
# summarize history for loss
plt.plot(history_Nadam.history['loss'])
plt.plot(history_Nadam.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.ylim(0,2.2)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for accuracy
plt.plot(history_Nadam.history['acc'])
plt.plot(history_Nadam.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.ylim(0.1,1)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





In [107]:
```
BATCH_SIZE = 128
MAX_EPOCH = 50
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.RMSprop(),
              metrics=['accuracy'])
history_RMSprop = model.fit(x_train_VGG, y_train_VGG,
          batch_size=BATCH_SIZE,
          epochs=MAX_EPOCH,
          verbose=1,
          validation_data=(x_test_VGG, y_test_VGG))
score = model.evaluate(x_test_VGG, y_test_VGG, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/50
50000/50000 [==============================] - 16s 329us/step - loss:
1.5991 - acc: 0.4330 - val_loss: 1.6797 - val_acc: 0.4591
Epoch 2/50
50000/50000 [==============================] - 12s 245us/step - loss:
1.0683 - acc: 0.6324 - val_loss: 1.0910 - val_acc: 0.6369
Epoch 3/50
50000/50000 [==============================] - 12s 238us/step - loss:
0.8587 - acc: 0.7119 - val_loss: 0.9857 - val_acc: 0.6677
Epoch 4/50
50000/50000 [==============================] - 12s 241us/step - loss:
0.7303 - acc: 0.7604 - val_loss: 0.7986 - val_acc: 0.7431
Epoch 5/50
50000/50000 [==============================] - 12s 239us/step - loss:
0.6381 - acc: 0.7930 - val_loss: 0.7648 - val_acc: 0.7548
Epoch 6/50
50000/50000 [==============================] - 12s 241us/step - loss:
0.5716 - acc: 0.8161 - val_loss: 0.6257 - val_acc: 0.7989
Epoch 7/50
50000/50000 [==============================] - 12s 238us/step - loss:
0.5098 - acc: 0.8353 - val_loss: 0.6098 - val_acc: 0.8039
Epoch 8/50
50000/50000 [==============================] - 12s 240us/step - loss:
0.4670 - acc: 0.8488 - val_loss: 0.5470 - val_acc: 0.8191
Epoch 9/50
50000/50000 [==============================] - 12s 243us/step - loss:
0.4266 - acc: 0.8628 - val_loss: 0.5741 - val_acc: 0.8242
Epoch 10/50
50000/50000 [==============================] - 12s 247us/step - loss:
0.3908 - acc: 0.8749 - val_loss: 0.6792 - val_acc: 0.7996
Epoch 11/50
50000/50000 [==============================] - 12s 249us/step - loss:
0.3563 - acc: 0.8867 - val_loss: 0.7010 - val_acc: 0.8056
Epoch 12/50
50000/50000 [==============================] - 13s 252us/step - loss:
0.3306 - acc: 0.8944 - val_loss: 0.6557 - val_acc: 0.8216
Epoch 13/50
50000/50000 [==============================] - 12s 248us/step - loss:
0.3031 - acc: 0.9023 - val_loss: 0.6659 - val_acc: 0.8140
Epoch 14/50
50000/50000 [==============================] - 12s 243us/step - loss:
0.2829 - acc: 0.9088 - val_loss: 0.6305 - val_acc: 0.8356
Epoch 15/50
50000/50000 [==============================] - 12s 247us/step - loss:
0.2636 - acc: 0.9149 - val_loss: 0.7954 - val_acc: 0.7886
Epoch 16/50
50000/50000 [==============================] - 12s 248us/step - loss:
0.2476 - acc: 0.9224 - val_loss: 0.7604 - val_acc: 0.8160
Epoch 17/50
50000/50000 [==============================] - 12s 248us/step - loss:
0.2309 - acc: 0.9254 - val_loss: 0.5690 - val_acc: 0.8586
Epoch 18/50
50000/50000 [==============================] - 12s 246us/step - loss:
0.2167 - acc: 0.9315 - val_loss: 0.6179 - val_acc: 0.8488
Epoch 19/50
50000/50000 [==============================] - 12s 248us/step - loss:
```

```
0.2015 - acc: 0.9362 - val_loss: 0.6492 - val_acc: 0.8466
Epoch 20/50
50000/50000 [==============================] - 12s 247us/step - loss:
0.1989 - acc: 0.9383 - val_loss: 0.7450 - val_acc: 0.8202
Epoch 21/50
50000/50000 [==============================] - 12s 247us/step - loss:
0.1876 - acc: 0.9416 - val_loss: 0.6410 - val_acc: 0.8440
Epoch 22/50
50000/50000 [==============================] - 12s 249us/step - loss:
0.1725 - acc: 0.9467 - val_loss: 0.6248 - val_acc: 0.8521
Epoch 23/50
50000/50000 [==============================] - 12s 248us/step - loss:
0.1636 - acc: 0.9493 - val_loss: 0.7388 - val_acc: 0.8244
Epoch 24/50
50000/50000 [==============================] - 12s 249us/step - loss:
0.1571 - acc: 0.9511 - val_loss: 0.7895 - val_acc: 0.8358
Epoch 25/50
50000/50000 [==============================] - 12s 245us/step - loss:
0.1533 - acc: 0.9543 - val_loss: 0.7159 - val_acc: 0.8476
Epoch 26/50
50000/50000 [==============================] - 12s 248us/step - loss:
0.1446 - acc: 0.9566 - val_loss: 0.7043 - val_acc: 0.8384
Epoch 27/50
50000/50000 [==============================] - 12s 246us/step - loss:
0.1375 - acc: 0.9582 - val_loss: 0.6664 - val_acc: 0.8552
Epoch 28/50
50000/50000 [==============================] - 12s 245us/step - loss:
0.1335 - acc: 0.9597 - val_loss: 0.6555 - val_acc: 0.8522
Epoch 29/50
50000/50000 [==============================] - 12s 249us/step - loss:
0.1286 - acc: 0.9607 - val_loss: 0.6874 - val_acc: 0.8434
Epoch 30/50
50000/50000 [==============================] - 12s 247us/step - loss:
0.1251 - acc: 0.9623 - val_loss: 0.8071 - val_acc: 0.8376
Epoch 31/50
50000/50000 [==============================] - 12s 246us/step - loss:
0.1198 - acc: 0.9648 - val_loss: 0.6501 - val_acc: 0.8632
Epoch 32/50
50000/50000 [==============================] - 12s 247us/step - loss:
0.1179 - acc: 0.9658 - val_loss: 0.7089 - val_acc: 0.8425
Epoch 33/50
50000/50000 [==============================] - 12s 246us/step - loss:
0.1129 - acc: 0.9656 - val_loss: 0.8513 - val_acc: 0.8277
Epoch 34/50
50000/50000 [==============================] - 12s 247us/step - loss:
0.1117 - acc: 0.9664 - val_loss: 0.9202 - val_acc: 0.8273
Epoch 35/50
50000/50000 [==============================] - 12s 242us/step - loss:
0.1104 - acc: 0.9679 - val_loss: 0.6386 - val_acc: 0.8636
Epoch 36/50
50000/50000 [==============================] - 12s 246us/step - loss:
0.1033 - acc: 0.9694 - val_loss: 0.7566 - val_acc: 0.8524
Epoch 37/50
50000/50000 [==============================] - 12s 249us/step - loss:
0.1005 - acc: 0.9713 - val_loss: 0.7331 - val_acc: 0.8556
Epoch 38/50
50000/50000 [==============================] - 12s 245us/step - loss:
```

```
0.1003 - acc: 0.9713 - val_loss: 0.7536 - val_acc: 0.8563
Epoch 39/50
50000/50000 [==============================] - 12s 249us/step - loss:
0.0994 - acc: 0.9720 - val_loss: 0.7282 - val_acc: 0.8556
Epoch 40/50
50000/50000 [==============================] - 12s 248us/step - loss:
0.0982 - acc: 0.9712 - val_loss: 0.7373 - val_acc: 0.8599
Epoch 41/50
50000/50000 [==============================] - 12s 247us/step - loss:
0.0936 - acc: 0.9731 - val_loss: 0.7248 - val_acc: 0.8597
Epoch 42/50
50000/50000 [==============================] - 12s 245us/step - loss:
0.0882 - acc: 0.9742 - val_loss: 0.7132 - val_acc: 0.8597
Epoch 43/50
50000/50000 [==============================] - 12s 243us/step - loss:
0.0904 - acc: 0.9737 - val_loss: 0.7408 - val_acc: 0.8550
Epoch 44/50
50000/50000 [==============================] - 12s 246us/step - loss:
0.0857 - acc: 0.9748 - val_loss: 0.8124 - val_acc: 0.8380
Epoch 45/50
50000/50000 [==============================] - 12s 243us/step - loss:
0.0914 - acc: 0.9738 - val_loss: 0.7211 - val_acc: 0.8587
Epoch 46/50
50000/50000 [==============================] - 12s 247us/step - loss:
0.0830 - acc: 0.9759 - val_loss: 0.8645 - val_acc: 0.8535
Epoch 47/50
50000/50000 [==============================] - 12s 250us/step - loss:
0.0845 - acc: 0.9761 - val_loss: 0.6940 - val_acc: 0.8693
Epoch 48/50
50000/50000 [==============================] - 12s 245us/step - loss:
0.0858 - acc: 0.9757 - val_loss: 0.7605 - val_acc: 0.8647
Epoch 49/50
50000/50000 [==============================] - 12s 242us/step - loss:
0.0846 - acc: 0.9764 - val_loss: 0.8083 - val_acc: 0.8552
Epoch 50/50
50000/50000 [==============================] - 12s 248us/step - loss:
0.0771 - acc: 0.9781 - val_loss: 0.8476 - val_acc: 0.8480
Test loss: 0.8475979769527913
Test accuracy: 0.848
```

In [175]:
```
# summarize history for loss
plt.plot(history_RMSprop.history['loss'])
plt.plot(history_RMSprop.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.ylim(0,2.2)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for accuracy
plt.plot(history_RMSprop.history['acc'])
plt.plot(history_RMSprop.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.ylim(0.1,1)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```