Project 3

Group 10

March 28, 2018

Project Summary: In this project, we have created a classification engine for images of poodles, fried chicken and blueberry muffins. Our goal is to improve the prediction accuracy from the baseline model (GBM with decision stumps on 2000 SIFT features) and to enhance computational efficiency in terms of running time, storage and memory cost. We have examined feature descriptors such as cale-invariant feature transform (SIFT), histogram of oriented gradients (HOG), and local binary pattern (LBP), and evaluated classifiers including Gradient Boosting Machine (GBM), RBF Support Vector Machine (SVM), Random Forest (RF), XGBoost, Logistic Regression, Convolutional Neural Network (CNN).

Step 0: install/load packages and specify directories

Install and load packages.

```
if(!require("EBImage")){
  source("https://bioconductor.org/biocLite.R")
  biocLite("EBImage")
}
```

```
## Loading required package: EBImage
```

```
packages.used=c("gbm", "MASS", "OpenImageR", "jpeg", "ggplot2", "reshape2", "randomForest",
                "e1071", "xgboost")

# check packages that need to be installed.
packages.needed=setdiff(packages.used,
                        intersect(installed.packages()[,1],
                                  packages.used))
# install additional packages
if(length(packages.needed)>0){
  install.packages(packages.needed, dependencies = TRUE,
                   repos='http://cran.us.r-project.org')
}

## Loading packages
library("EBImage")
library("gbm")
```

```
## Loading required package: survival

## Loading required package: lattice

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.3
```

```
library("MASS")
library("OpenImageR")
```

```
##
## Attaching package: 'OpenImageR'
```

```
## The following objects are masked from 'package:EBImage':
##
##     readImage, writeImage
```

```
library("jpeg")
library("ggplot2")
library("reshape2")
library("randomForest")
```

```
## Warning: package 'randomForest' was built under R version 3.4.4
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
## The following object is masked from 'package:EBImage':
##
##     combine
```

```
library("e1071")
library("xgboost")
```

Set the working directory to the doc folder. Provide directories for raw images. Training set and test set are in different subfolders.

```
# setwd("../doc/")
experiment_dir <- "../data/" # This will be modified for different data sets.
img_train_dir <- paste(experiment_dir, "train/", sep="")
img_test_dir <- paste(experiment_dir, "test/", sep="")
```

Step 1: set up controls for evaluation experiments.

In this chunk, we have a set of controls for the evaluation experiments.

- (T/F) cross-validation on the training set
- (number) K, the number of CV folds
- (T/F) process features for images
- (T/F) run evaluation on an independent test set
- (T/F) process features for test set

```
run.cv=F # run cross-validation on the training set
K <- 5  # number of CV folds
run.feature=F # process features
run.test=F # run evaluation on an independent test set
run.test.feature = F # process features for test set
```

Which feature extraction method to perform

```
run.sift = F
run.hog = F
run.lbp = T
```

Which model to run

```
run.gbm = F
run.svm = F
run.xgboost = F
run.rf = T
run.logistic = F
```

Using cross-validation or independent test set evaluation, we compare the performance of different classifiers or classifiers with different specifications.

```
model_values <- seq(3, 11, 2) # depth for GBM
# model_labels = paste("GBM with depth =", model_values)

svm_values <- seq(0.01, 0.1, by = 0.02) # gamma for svm
# svm_labels = paste("SVM with gamma =", svm_values)

xgboost_values <- seq(0.1, 0.5, by = 0.1) # eta for xgboost
# xgboost_labels = paste("XGBoost with eta =", xgboost_values)
```

Step 2: construct visual feature and split training and testing sets

The baseline model uses scale-invariant feature transform (SIFT) algorithm to extract features, which have 2000 dimensions and are extremely over-weighted. We then have decided to explore a couple of other visual descriptors, including histogram of oriented gradients (HOG) and local binary patterns (LBP), which produce 55 dimensions and 59 dimensions respectively.

As we do not have the data with an independent test set, we have create our own testing data by random subsampling. We have used balanced partition to split the data, such that 80% as training set and 20% as testing set. In order to obain reproducible results, we have set seed to be 123.

```
# set up feature name
if(run.sift){
   feature_name <- "sift"
}
if(run.hog){
  feature_name <- "hog"
}
if(run.lbp){
  feature_name <- "lbp"
}

# run feature extraction
source("../lib/feature.R")
```

```
## Bioconductor version 3.6 (BiocInstaller 1.28.0), ?biocLite for help

## BioC_mirror: https://bioconductor.org

## Using Bioconductor 3.6 (BiocInstaller 1.28.0), R 3.4.3 (2017-11-30).

## Installing package(s) 'EBImage'
```

```
##
## The downloaded binary packages are in
##  /var/folders/8j/pkxd_zrs7ys47vj7s77fl1mh0000gn/T//Rtmp9TCa5o/downloaded_packages
```

```
## Old packages: 'curl', 'FactoMineR', 'lava', 'lme4', 'pdftools', 'rgl',
##   'spam'
```

```
features <- NULL
if(run.feature){
  features <- feature(experiment_dir,
                      run.sift = run.sift, run.hog = run.hog, run.lbp = run.lbp,
                      export = T)
}else{
  if("sift" %in% feature_name){
    load("../output/feature_SIFT.RData")
    sift2=read.csv("../data/SIFT_train.csv", header = F)
    dim(sift2)
    features <- sift
  }
  if("hog" %in% feature_name){
    load("../output/feature_HOG.RData")
    features <- hog
  }
  if ("lbp" %in% feature_name){
    features <- read.csv('../output/feature_LBP.csv',header = F)
  }
}

# constructing the feature path name
train_path <- paste0("../output/", feature_name, "_train.csv")
test_path <- paste0("../output/", feature_name, "_test.csv")

# split train and test sets
source("../lib/split_train_test.R")
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:survival':
##
##     cluster
```

```
data <- split_train_test(features)
write.csv(data$train, train_path)
write.csv(data$test, test_path)
#save(data$train, file=paste0("../output/", feature_name, "_train.RData"))
#save(data$test, file=paste0("../output/", feature_name, "_test.RData"))
```

Step 3: import training and testing sets.

```
# read test dataset
dat_test <- read.csv(test_path, header = T)
dat_test <- dat_test[,-1]

# read train dataset
```

```
dat_train <- read.csv(train_path, header = T)
dat_train <- dat_train[, 3:length(dat_train[1,])]
label_train <- read.csv(train_path, header = T)
label_train <- label_train[,2]
```

Step 4: train a classification model with training images

Call the train model and test model from library.

`train.R` and `test.R` are wrappers for all model training steps and classification/prediction steps. `train.R`

```
source("../lib/train.R")
source("../lib/test.R")
```

Model selection with cross-validation

- Do model selection by choosing among different values of training model parameters.

```
source("../lib/cross_validation.R")

if(run.cv){
  if(cv.gbm){
    err_cv <- array(dim=c(length(model_values), 2))
    for(k in 1:length(model_values)){
      cat("k=", k, "\n")
      err_cv[k,] <- cv.function(as.data.frame(dat_train), label_train, model_values[k], K, cv.gbm = T)
    }
  }

  if(cv.svm){
    err_cv <- array(dim=c(length(svm_values), 2))
    for(k in 1:length(svm_values)){
      cat("k=", k, "\n")
      err_cv[k,] <- cv.function(as.data.frame(dat_train), label_train, svm_values[k], K, cv.svm = T)
    }
  }

  if(cv.xgboost){
    err_cv <- array(dim=c(length(xgboost_values), 2))
    for(k in 1:length(xgboost_values)){
      cat("k=", k, "\n")
      err_cv[k,] <- cv.function(as.data.frame(dat_train), label_train, xgboost_values[k], K, cv.xgboost
    }
  }

  save(err_cv, file="../output/err_cv.RData")
}
```

Visualize cross-validation results.

```
if(run.cv){
  if(cv.gbm){
    load("../output/err_cv.RData")
    #pdf("../figs/cv_results.pdf", width=7, height=5)
```

```
    plot(model_values, err_cv[,1], xlab="Interaction Depth", ylab="CV Error",
         main="Cross Validation Error", type="n", ylim=c(0.25, 0.4))
    points(model_values, err_cv[,1], col="blue", pch=16)
    lines(model_values, err_cv[,1], col="blue")
    arrows(model_values,
           err_cv[,1]-err_cv[,2],
           model_values,
           err_cv[,1]+err_cv[,2],
           length=0.1,
           angle=90,
           code=3)
    #dev.off()
  }
  if(cv.svm){
    load("../output/err_cv.RData")
    #pdf("../figs/cv_results.pdf", width=7, height=5)
    plot(svm_values, err_cv[,1], xlab="SVM gamma values", ylab="CV Error",
         main="Cross Validation Error", type="n", ylim=c(0.30, 0.50))
    points(svm_values, err_cv[,1], col="blue", pch=16)
    lines(svm_values, err_cv[,1], col="blue")
    arrows(svm_values,
           err_cv[,1]-err_cv[,2],
           svm_values,
           err_cv[,1]+err_cv[,2],
           length=0.1,
           angle=90,
           code=3)
    #dev.off()
  }
  if(cv.xgboost){
    load("../output/err_cv.RData")
    #pdf("../figs/cv_results.pdf", width=7, height=5)
    plot(xgboost_values, err_cv[,1], xlab="XGBoost eta values", ylab="CV Error",
         main="Cross Validation Error", type="n", ylim=c(0.30, 0.50))
    points(xgboost_values, err_cv[,1], col="blue", pch=16)
    lines(xgboost_values, err_cv[,1], col="blue")
    arrows(xgboost_values,
           err_cv[,1]-err_cv[,2],
           xgboost_values,
           err_cv[,1]+err_cv[,2],
           length=0.1,
           angle=90,
           code=3)
    #dev.off()
  }

}
```

- List cross validation results and choose the "best" parameter value for each model and each feature.

```
source("../lib/cv_result.R")

cv.result <- cv_result()
```

Tune `depth` in GBM

```
cv.result[[1]]
```

```
##          depth = 3 depth = 5 depth = 7 depth = 9 depth = 11
## hog_gbm  0.4213333 0.4126667 0.4090000 0.4013333  0.4193333
## lbp_gbm  0.3670000 0.3520000 0.3350000 0.3353333  0.3390000
## sift_gbm 0.3096667 0.3103333 0.3033333 0.3066667  0.3040000
```

Tune `eta` in XGBoost

```
cv.result[[2]]
```

```
##              eta = 0.1 eta = 0.2 eta = 0.3 eta = 0.4 eta = 0.5
## hog_xgboost  0.3693333 0.3573333 0.3650000 0.3736667 0.3796667
## lbp_xgboost  0.2513333 0.2566667 0.2633333 0.2590000 0.2690000
## sift_xgboost 0.2490000 0.2536667 0.2576667 0.2693333 0.2643333
```

Tune `gamma` in SVM

```
cv.result[[3]]
```

```
##          gamma = 0.01 gamma = 0.03 gamma = 0.05 gamma = 0.07 gamma = 0.09
## hog_svm     0.3576667    0.3540000    0.3616667    0.3626667    0.4256667
## lbp_svm     0.2836667    0.2656667    0.2730000    0.2763333    0.2783333
## sift_svm    0.6306667    0.6836667    0.6793333    0.6880000    0.6793333
```

Tune `mtry` and `ntree` in Random Forest for feature hog

```
cv.result[[4]]
```

```
##           ntree = 1000 ntree = 1500 ntree = 2000 ntree = 2500
## mtry = 5     0.3720000    0.3773333    0.3663333    0.3646667
## mtry = 6     0.3750000    0.3760000    0.3586667    0.3693333
## mtry = 7     0.3630000    0.3766667    0.3810000    0.3690000
## mtry = 8     0.3716667    0.3823333    0.3636667    0.3630000
## mtry = 9     0.3703333    0.3830000    0.3703333    0.3776667
## mtry = 10    0.3713333    0.3750000    0.3860000    0.3670000
## mtry = 11    0.3720000    0.3833333    0.3786667    0.3716667
## mtry = 12    0.3750000    0.3810000    0.3906667    0.3670000
## mtry = 13    0.3730000    0.3780000    0.3806667    0.3783333
## mtry = 14    0.3730000    0.3816667    0.3780000    0.3993333
## mtry = 15    0.3743333    0.3866667    0.3786667    0.3643333
```

Tune `mtry` and `ntree` in Random Forest for feature lbp

```
cv.result[[5]]
```

```
##           ntree = 1000 ntree = 1500 ntree = 2000 ntree = 2500
## mtry = 5     0.2856667    0.2933333    0.3120000    0.3136667
## mtry = 6     0.2993333    0.2986667    0.3113333    0.3063333
## mtry = 7     0.3083333    0.3130000    0.2943333    0.3066667
## mtry = 8     0.3036667    0.3056667    0.2970000    0.3046667
## mtry = 9     0.3013333    0.3113333    0.2956667    0.3063333
## mtry = 10    0.3063333    0.3096667    0.2946667    0.3123333
## mtry = 11    0.3040000    0.3013333    0.3020000    0.3023333
## mtry = 12    0.3033333    0.3043333    0.3063333    0.2930000
## mtry = 13    0.3033333    0.3100000    0.3040000    0.2963333
## mtry = 14    0.3050000    0.2806667    0.2990000    0.3040000
## mtry = 15    0.2930000    0.3096667    0.2986667    0.2916667
```

```
# min(cv.result[[5]])
```

Step 5: train entire training set, compare prediction error and running time

Train models with the entire training set for each feature using the selected model (model parameter) via cross-validation. Feed the final training model with the completely holdout testing data.

```
source("../lib/model_compare.R")

run.compare <- F
compare_result <- compare.model(run.compare)
```

Compare models using SIFT feature

```
compare_result[[1]]
```

```
##          X         gbm        svm   logistic           rf xgboost
## 1 tm_train 403.5300000 50.9900000 38.4700000 1469.3500000 632.720
## 2  tm_test   0.0500000  4.4000000  0.3000000    0.5000000   0.140
## 3    error   0.3266667  0.5916667  0.3266667    0.2866667   0.235
```

Compare models using hog feature

```
compare_result[[2]]
```

```
##          X    gbm   svm logistic         rf xgboost
## 1 tm_train 18.236 2.201    0.482 77.8070000  36.404
## 2  tm_test  0.006 0.177    0.007  0.3060000   0.101
## 3    error  0.395 0.330    0.395  0.4216667   0.355
```

Compare models using lbp feature

```
compare_result[[3]]
```

```
##          X        gbm   svm  logistic         rf    xgboost
## 1 tm_train 21.1490000 1.899 0.3260000 62.3550000 50.5570000
## 2  tm_test  0.0200000 0.163 0.0080000  0.2030000  0.1960000
## 3    error  0.3083333 0.255 0.2316667  0.2716667  0.2483333
```

Since logistic model with LBP features has lowest running time and lowest prediction error, we choose our final model to be lbp+logistic. Compared to baseline model (sift+gbm), the training time decreases to 0.33s from 403.53s and prediction accuracy increased by 9.5% from 67.3% to 76.8%.

CNN model

1. Import necessary packages

In [1]:
```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd
import random
import time
%matplotlib inline
```

```
C:\Anaconda\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion
of the second argument of issubdtype from `float` to `np.floating` is depreca
ted. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

2. Read and proprocess data

a) create the path list of images and labels

In [2]:
```
def form_readlist():
    images_labels_df = pd.read_csv('../data/label_train.csv')[['img', 'label'
]]
    filenamelist = []
    filelabellist = []
    for index in images_labels_df.index:
        filenamelist.append('../data/images/' +
                            images_labels_df.iloc[index, 0][4:] + '.jpg')
        filelabellist.append(images_labels_df.iloc[index, 1]-1)
    return(filenamelist, filelabellist)
```

In [3]:
```
filenamelist, filelabellist = form_readlist()
```

b) define the transformation and standardilization of images, transform the labels into one hot vectors

In [4]:
```
def _parse_function_for_train(filenamelist, filelabellist):
    image_string = tf.read_file(filenamelist)
    image_decoded = tf.image.decode_jpeg(image_string, channels = 3)
    image_resized = tf.image.resize_images(image_decoded, [128, 128])
    image_resized = image_resized / 255.0

    #Use filp and rotation to carry out the data augmentation of training data
    image_resized = tf.image.random_flip_left_right(image_resized)
    rotate_or_not = random.choice((True, False))
    if rotate_or_not:
        radians = np.random.uniform(-0.262, 0.262)
        image_resized = tf.contrib.image.rotate(image_resized, radians)

    label = tf.one_hot(filelabellist, depth = 3)
    return(image_resized, label)
```

In [5]:
```
def _parse_function_for_validation(filenamelist, filelabellist):
    image_string = tf.read_file(filenamelist)
    image_decoded = tf.image.decode_jpeg(image_string, channels = 3)
    image_resized = tf.image.resize_images(image_decoded, [128, 128])
    image_resized = image_resized / 255.0
    label = tf.one_hot(filelabellist, depth = 3)
    return(image_resized, label)
```

c) seperate the data into training set and validation set

In [6]:
```
partitions = [0] * len(filelabellist)
testfilesize = int(len(filelabellist)/5)
partitions[:testfilesize] = [1] * testfilesize
random.shuffle(partitions)
```

In [7]:
```
trainfilelist, testfilelist = tf.dynamic_partition(data = filenamelist,
                                                   partitions = partitions,
                                                   num_partitions = 2)
trainlabellist, testlabellist = tf.dynamic_partition(data = filelabellist,
                                                     partitions = partitions,
                                                     num_partitions = 2)
```

d) use the Dataset API to feed the data in CNN model(we shuffle and batch the data)

In [8]:
```
dataset_train = tf.data.Dataset.from_tensor_slices((trainfilelist,
                                                    trainlabellist))
dataset_train = dataset_train.map(_parse_function_for_train)
dataset_train = dataset_train.shuffle(buffer_size = 3000).repeat().batch(100)
iterator_train = dataset_train.make_one_shot_iterator()
one_batch_train = iterator_train.get_next()

dataset_validation = tf.data.Dataset.from_tensor_slices((testfilelist, testlab
ellist))
dataset_validation = dataset_validation.map(_parse_function_for_validation)
dataset_validation = dataset_validation.batch(testfilesize)
iterator_validation = dataset_validation.make_one_shot_iterator()
one_batch_validation = iterator_validation.get_next()
```

3. define the structure of the CNN model

a) Initialization and definitions of functions of CNN model

In [9]:
```
learning_rate = 1e-3
```

In [10]:
```
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return(tf.Variable(initial))

def bias_variable(size):
    initial = tf.constant(0.1, shape=size)
    return(tf.Variable(initial))

def conv2d(x, W):
    return(tf.nn.conv2d(x, W, strides = [1, 1, 1, 1], padding = 'SAME'))

def max_pool_2x2(x):
    return(tf.nn.max_pool(x, ksize = [1, 2, 2, 1], strides = [1, 2, 2, 1],
                          padding = 'SAME'))
```

b) define the structure of CNN model

In [11]:
```
x = tf.placeholder(tf.float32, shape=[None, 128, 128, 3], name='x')
y_true_one_hot = tf.placeholder(tf.float32, shape=[None, 3], name='y_')
y_true_label = tf.argmax(y_true_one_hot, axis=1)
```

In [12]:
```
W_conv1 = weight_variable([4, 4, 3, 32])
b_conv1 = bias_variable([32])
c_conv1 = tf.nn.bias_add(conv2d(x, W_conv1), b_conv1)
c_pool1 = max_pool_2x2(c_conv1)
o_conv1 = tf.nn.relu(c_pool1)
```

In [13]:
```
W_conv2 = weight_variable([4, 4, 32, 32])
b_conv2 = bias_variable([32])
c_conv2 = tf.nn.bias_add(conv2d(o_conv1, W_conv2), b_conv2)
c_pool2 = max_pool_2x2(c_conv2)
o_conv2 = tf.nn.relu(c_pool2)
```

In [14]:
```
W_conv3 = weight_variable([4, 4, 32, 64])
b_conv3 = bias_variable([64])
c_conv3 = tf.nn.bias_add(conv2d(o_conv2, W_conv3), b_conv3)
c_pool3 = max_pool_2x2(c_conv3)
o_conv3 = tf.nn.relu(c_pool3)
```

In [15]:
```
o_conv3_flat = tf.reshape(o_conv3, [-1, 16 * 16 * 64])
W_fc1 = weight_variable([16 * 16 * 64, 128])
b_fc1 = bias_variable([128])
c_fc1 = tf.nn.bias_add(tf.matmul(o_conv3_flat, W_fc1), b_fc1)
o_fc1 = tf.nn.relu(c_fc1)
```

In [16]:
```
keep_prob = tf.placeholder(tf.float32, name = 'keep_prob')
o_fc1_drop = tf.nn.dropout(o_fc1, keep_prob)
```

In [17]:
```
W_fc2 = weight_variable([128, 3])
b_fc2 = bias_variable([3])
o_fc2 = tf.nn.bias_add(tf.matmul(o_fc1_drop, W_fc2), b_fc2)
y = tf.nn.softmax(o_fc2)
y_pred_label = tf.argmax(y, axis = 1)
```

c) define the loss function, optimizer and accuracy of the model

In [18]:
```
cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2(logits=o_fc2,
                                                           labels=y_true_one_h
ot)
cost = tf.reduce_mean(cross_entropy)
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)

correct_prediction = tf.equal(y_pred_label, y_true_label)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

4. Train the model

In [19]:
```
time_list = []
training_accuracy_list = []
validation_accuracy_list = []
with tf.Session() as session:
    tf.global_variables_initializer().run()
    start_time = time.time()
    validation_x, validation_y = session.run(one_batch_validation)
    for i in range(500):
        train_x_batch, train_y_batch = session.run(one_batch_train)
        if i % 10 == 0:
            train_accuracy = accuracy.eval(feed_dict = {x: train_x_batch,
                                                        y_true_one_hot: train_
y_batch,
                                                        keep_prob: 1.0})
            validation_accuracy = accuracy.eval(feed_dict = {x: validation_x,
                                                             y_true_one_hot: v
alidation_y,
                                                             keep_prob: 1.0})
            print('step %d, training accuracy %g, validation accuracy %g, time
 %.3f sec'
                  %(i, train_accuracy, validation_accuracy, time.time() - star
t_time))

            time_list.append(time.time() - start_time)
            training_accuracy_list.append(train_accuracy)
            validation_accuracy_list.append(validation_accuracy)

        optimizer.run(feed_dict = {x: train_x_batch,
                                   y_true_one_hot: train_y_batch,
                                   keep_prob: 0.7})
```

```
step 0, training accuracy 0.34, validation accuracy 0.366667, time 26.423 s
ec
step 10, training accuracy 0.3, validation accuracy 0.343333, time 53.039 s
ec
step 20, training accuracy 0.52, validation accuracy 0.458333, time 79.485
sec
step 30, training accuracy 0.55, validation accuracy 0.505, time 122.250 se
c
step 40, training accuracy 0.53, validation accuracy 0.595, time 148.641 se
c
step 50, training accuracy 0.75, validation accuracy 0.633333, time 191.292
sec
step 60, training accuracy 0.61, validation accuracy 0.655, time 218.177 se
c
step 70, training accuracy 0.63, validation accuracy 0.691667, time 244.583
sec
step 80, training accuracy 0.66, validation accuracy 0.688333, time 286.976
sec
step 90, training accuracy 0.65, validation accuracy 0.678333, time 313.303
sec
step 100, training accuracy 0.65, validation accuracy 0.696667, time 355.93
0 sec
step 110, training accuracy 0.77, validation accuracy 0.751667, time 382.33
1 sec
step 120, training accuracy 0.87, validation accuracy 0.748333, time 425.12
8 sec
step 130, training accuracy 0.78, validation accuracy 0.761667, time 451.55
9 sec
step 140, training accuracy 0.71, validation accuracy 0.731667, time 477.81
0 sec
step 150, training accuracy 0.75, validation accuracy 0.791667, time 520.32
6 sec
step 160, training accuracy 0.81, validation accuracy 0.78, time 546.738 se
c
step 170, training accuracy 0.86, validation accuracy 0.783333, time 589.25
4 sec
step 180, training accuracy 0.8, validation accuracy 0.751667, time 615.659
sec
step 190, training accuracy 0.81, validation accuracy 0.775, time 641.986 s
ec
step 200, training accuracy 0.84, validation accuracy 0.791667, time 684.63
9 sec
step 210, training accuracy 0.86, validation accuracy 0.785, time 710.801 s
ec
step 220, training accuracy 0.79, validation accuracy 0.755, time 753.188 s
ec
step 230, training accuracy 0.86, validation accuracy 0.81, time 779.641 se
c
step 240, training accuracy 0.8, validation accuracy 0.803333, time 822.482
sec
step 250, training accuracy 0.81, validation accuracy 0.831667, time 848.87
3 sec
step 260, training accuracy 0.8, validation accuracy 0.805, time 875.186 se
c
step 270, training accuracy 0.86, validation accuracy 0.835, time 917.822 s
ec
step 280, training accuracy 0.83, validation accuracy 0.836667, time 944.13
```

```
5 sec
step 290, training accuracy 0.86, validation accuracy 0.795, time 986.715 s
ec
step 300, training accuracy 0.88, validation accuracy 0.815, time 1013.043
sec
step 310, training accuracy 0.81, validation accuracy 0.786667, time 1039.5
29 sec
step 320, training accuracy 0.84, validation accuracy 0.82, time 1082.001 s
ec
step 330, training accuracy 0.87, validation accuracy 0.851667, time 1108.3
08 sec
step 340, training accuracy 0.89, validation accuracy 0.855, time 1151.103
sec
step 350, training accuracy 0.92, validation accuracy 0.848333, time 1177.6
81 sec
step 360, training accuracy 0.88, validation accuracy 0.856667, time 1220.3
32 sec
step 370, training accuracy 0.91, validation accuracy 0.86, time 1247.227 s
ec
step 380, training accuracy 0.88, validation accuracy 0.85, time 1273.656 s
ec
step 390, training accuracy 0.9, validation accuracy 0.816667, time 1316.22
2 sec
step 400, training accuracy 0.85, validation accuracy 0.8, time 1342.906 se
c
step 410, training accuracy 0.92, validation accuracy 0.868333, time 1385.5
00 sec
step 420, training accuracy 0.86, validation accuracy 0.87, time 1412.530 s
ec
step 430, training accuracy 0.91, validation accuracy 0.876667, time 1440.1
10 sec
step 440, training accuracy 0.96, validation accuracy 0.84, time 1484.208 s
ec
step 450, training accuracy 0.88, validation accuracy 0.818333, time 1510.6
38 sec
step 460, training accuracy 0.94, validation accuracy 0.838333, time 1554.0
37 sec
step 470, training accuracy 0.93, validation accuracy 0.86, time 1580.931 s
ec
step 480, training accuracy 0.96, validation accuracy 0.865, time 1624.132
sec
step 490, training accuracy 0.95, validation accuracy 0.873333, time 1651.1
91 sec
```

5. Visualization

In [21]:
```
fig1 = plt.figure()
ax = fig1.add_subplot(1, 1, 1)
ax.plot(time_list, training_accuracy_list, color = 'green',
        label = 'Training Accuracy')
ax.plot(time_list, validation_accuracy_list, color = 'red',
        label = 'Validation Accuracy')
ax.legend(loc = 'best')
ax.set_xlabel('training time')
ax.set_ylabel('accuracy')
ax.set_title('CNN Model')
```

Out[21]:
```
Text(0.5,1,'CNN Model')
```