In [1]:
```
%%time
!pwd
```

```
/c/Users/92312Wall time: 60 ms
```

In [ ]:
```
#%%install_ext https://raw.github.com/cpcloud/ipython-autotime/master/autot
#%load_ext autotime
```

In [2]:
```
cd d:\
```

```
d:\
```

In [3]:
```
cd d:\coding\python\
```

```
d:\coding\python
```

In [4]:
```
!pwd
```

```
/d/coding/python
```

In [5]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [6]:
```
!ls
```

```
01-Tensorflow Basics.ipynb
02-MNIST-with-Tensorflow.ipynb
03-Tensorflow with Estimators.ipynb
04-Tensorflow Project Exercise.ipynb
05-Tensorflow Project Exercise - Solutions-myediting.ipynb
05-Tensorflow Project Exercise - Solutions.ipynb
05-deep_learning_basics_and_computer_vision.ipynb
1.1. Linear Models â€” scikit-learn 0.23.2 documentation.pdf
1.10. Decision Trees â€” scikit-learn 0.23.2 documentation.pdf
1.11. Ensemble methods â€” scikit-learn 0.23.2 documentation.pdf
1.12. Multiclass and multilabel algorithms â€” scikit-learn 0.23.2 documenta
tion.pdf
1.13. Feature selection â€” scikit-learn 0.23.2 documentation.pdf
1.14. Semi-Supervised â€” scikit-learn 0.23.2 documentation.pdf
1.15. Isotonic regression â€” scikit-learn 0.23.2 documentation.pdf
1.16. Probability calibration â€” scikit-learn 0.23.2 documentation.pdf
1.17. Neural network models (supervised) â€” scikit-learn 0.23.2 documentati
on.pdf
1.1_notebook_quizz_v4.ipynb
1.2. Linear and Quadratic Discriminant Analysis â€” scikit-learn 0.23.2 docu
mentation.pdf
1.2_notebook_quizz_types.ipynb
1.3. Kernel ridge regression â€” scikit-learn 0.23.2 documentation.pdf
1.3_notebook_quizz_String_Operations.ipynb
1.4. Support Vector Machines â€” scikit-learn 0.23.2 documentation.pdf
1.5. Stochastic Gradient Descent â€” scikit-learn 0.23.2 documentation.pdf
1.6. Nearest Neighbors â€” scikit-learn 0.23.2 documentation.pdf
1.7. Gaussian Processes â€” scikit-learn 0.23.2 documentation.pdf
1.8. Cross decomposition â€” scikit-learn 0.23.2 documentation.pdf
1.9. Naive Bayes â€” scikit-learn 0.23.2 documentation.pdf
1.mp3
1.txt
1.wav
2.1_notebook_quizz_list_tuplesv4.ipynb
2.2_notebook_quizz_sets.ipynb
2.3_notebook_quizz_dictioary.ipynb
```

```
2329_3927_bundle_archive
2329_3927_bundle_archive.zip
3.1_notebook_quizz__Conditions_and_Branching.ipynb
3.2_notebook_quizz__Loops.ipynb
3.3_notebook_quizz_functions.ipynb
3.4_notebook_quizz_objects.ipynb
4.3_notebook_quizz_pandas.ipynb
4.4_notebook_quizz_pandas.ipynb
91783677_3145424692186895_360005382816399360_n.jpg
AirPassengers.csv
AirPassengers.ipynb
Animal_Classification
Animal_Classification.bmp
Animal_Classification.png
Appraisal Document.pdf
Assignment-1.ipynb
Assignment01_24_Sept_2020.ipynb
Assignment01_24_Sept_2020_final.ipynb
Building a Youtube Video Downloader using python.mp4
Celsius3.ipynb
Celsius4.ipynb
Celsius5.ipynb
Celsius_to_Fahrenheit.ipynb
CentralLimitTheorem.ipynb
Chinook-On-White-03.jpg
CoinToss.ipynb
CommViolPredUnnormalizedData.txt
Core-Temp-setup.exe
Crimes_version2_8jan.ipynb
Crimes_version3_8jan.ipynb
Crimes_version4_8jan.ipynb
Crimes_version9_8jan.ipynb
Crimes_version9_8jan_17_Oct_2020.ipynb
Crimes_version9_8jan_20-july-2020.ipynb
Detect_Faces.ipynb
Dogs_Cats.csv
EURUSD1m.csv
Ex1.txt
Ex2.txt
Example1.txt
Example2.txt
Example3.txt
INFLATION.csv
Image_Resize.ipynb
Image_Resolution.ipynb
Image_to_BlacknWhite.ipynb
Journey.wav
LRex01.ipynb
Lab-Loading+Graphs+in+NetworkX.ipynb
Language_Translation.ipynb
Language_translator.ipynb
MLfinalq4try.ipynb
Matplotlib-lib1.ipynb
Mid Exam.ipynb
Mid Tnt.ipynb
MidTerm-Copy1.ipynb
MidTerm.ipynb
OCR-Copy1.ipynb
OCR.ipynb
OOPS
PDF2AudioFile.py
PDF2AudioFile_paper01.py
PDF_splitter.py
PY0101EN-4-3-LoadData.ipynb
PY0101EN-5.2_API_2.ipynb
```

```
Pandas-Data-Science-Tasks-master
Pandas-Data-Science-Tasks-master.zip
PolynomialRegressionandPipelines.mp3
Progressive_bar.ipynb
Python_Thesaurus.py
Regular.csv
Regular.ipynb
SMSSpamCollection
SNA_graphs.py
Social_Network_Ads.csv
Some_New_Doc.pdf
SqueezeNet_v1.2-master.zip
TNTDemo-Copy1.ipynb
TNTDemo.ipynb
TextBlob.ipynb
Text_to_Speech.ipynb
Thinking-of-getting-a-cat.png
Tiny_URL.ipynb
UMICH_SI650_Sentiment_Classification.txt
US_Declaration.mp3
US_Declaration.pdf
US_Declaration_split.pdf
Unsupervised+Learning.ipynb
Untitled-Copy1.ipynb
Untitled.ipynb
Untitled1.ipynb
Untitled10.ipynb
Untitled11.ipynb
Untitled12-Copy1.ipynb
Untitled12.ipynb
Untitled13-Copy1.ipynb
Untitled13-Copy2.ipynb
Untitled13.ipynb
Untitled14.ipynb
Untitled15-Copy1.ipynb
Untitled15.ipynb
Untitled16-Copy1.ipynb
Untitled16.ipynb
Untitled17.ipynb
Untitled18.ipynb
Untitled19.ipynb
Untitled2.ipynb
Untitled20.ipynb
Untitled21.ipynb
Untitled22.ipynb
Untitled23.ipynb
Untitled24.ipynb
Untitled25.ipynb
Untitled26.ipynb
Untitled27.ipynb
Untitled28.ipynb
Untitled29.ipynb
Untitled3.ipynb
Untitled30.ipynb
Untitled31.ipynb
Untitled4.ipynb
Untitled41_30_Sept_2020.ipynb
Untitled42_30_Sept_2020.ipynb
Untitled43_30_Sept_2020.ipynb
Untitled44_14_sept_2020.ipynb
Untitled44_14_sept_2020b.ipynb
Untitled44_14_sept_2020c.ipynb
Untitled44_17_Sept_2020.ipynb
Untitled44_30_Sept_2020.ipynb
Untitled44_30_Sept_2020b.ipynb
```

```
Untitled45_17_sept_2020.ipynb
Untitled45_30_Sept_2020.ipynb
Untitled47_18_Sept_2020.ipynb
Untitled47_30_Sept_2020.ipynb
Untitled48_30_Sept_2020.ipynb
Untitled49_30_Sept_2020.ipynb
Untitled5.ipynb
Untitled6.ipynb
Untitled7.ipynb
Untitled8.ipynb
Untitled9.ipynb
Walkthrough.ipynb
WalkthroughTNTW8.ipynb
WebCAM.ipynb
Week 6 - Machine Learning.ipynb
Week 6 - Machine Learning.py
Week+1 (2).ipynb
Week+1.ipynb
Week+4.ipynb
__pycache__
a1.py
a10.py
a11.py
a12.py
a13.py
a14.py
a15.py
a16.py
a17.py
a18.py
a19 txt_file_to_audio_file.py
a2.py
a21.py
a22.py
a23.py
a24.py
a25.py
a26 text_line_to_audio.py
a27 text_multilines_to audio.py
a28 text_multilines_to audio.py
a29 combining_2_csv.py
a3.py
a30.py
a31 txt_to_json.py
a32 multiline_text_to_audio_file.py
a33.py
a34.py
a35.py
a36.py
a37.py
a38.py
a39.py
a4.py
a40.py
a41.py
a42.py
a43 yoputube-downloader.py
a43.py
a44 youtube_downloader.py
a45.py
a46.py
a47.py
a48.py
a49.py
a5.py
```

```
a50.py
a51.py
a52_udemy_2020_Complete_python_bootcamp.py
a53.py
a54.py
a55.py
a56.py
a57_pencil_sketch.py
a58.py
a6.py
a7.py
a8.py
a9.py
adspy_shared_utilities.py
assign05.ipynb
assignment05v1.ipynb
awscli-bundle.zip
ayaz-Copy1.ipynb
ayaz-Copy2.ipynb
ayaz.ipynb
bank_note_data.csv
card-pole-code-instagram.txt
cart-pole code.txt
celsius.ipynb
celsius1.ipynb
celsius2.ipynb
celsius_dataset.txt
cnn mnist-Copy1.ipynb
cnn mnist.ipynb
combined.csv
combined3.csv
combined4.csv
combined5.csv
crime-in-vancouver.zip
crime_ML_project-Copy1.ipynb
crime_ML_project.ipynb
crime_heatmap.html
crime_heatmap1.html
crimedata_csv_all_years_10july2020vuncuver
crimedata_csv_all_years_10july2020vuncuver.zip
crimes_dusra.ipynb
diabetes.csv
dogs-vs-cats-redux-kernels-edition.zip
file_03Sept2020.txt
final_stocks_project_23_Sept_2020.ipynb
first.pem
fruit_data_with_colors.txt
gaussian_02.ipynb
gaussian_answer.py
gaussian_class_full_implementation.ipynb
gaussian_class_full_implementation.py
gaussian_code_exercise.ipynb
hand_written_recognition-Copy2.ipynb
hand_written_recognition.ipynb
hello.mp3
ibm-credentials.env
ibmwatson_learning1.ipynb
insta_.ipynb
irfan_project.ipynb
irfan_project_15_Sept_2020.ipynb
irfan_project_30_Sept_2020.ipynb
iris.csv
kaggle.json
kaggle1.json
knn.py
```

```
l01c01_introduction_to_colab_and_python.ipynb
l02c01_celsius_to_fahrenheit.ipynb
l03c01_classifying_images_of_clothing (1).ipynb
l03c01_classifying_images_of_clothing.ipynb
labels.csv
lesson04.ipynb
math_func.py
miles_per_hours_regression_model_tf.ipynb
mobilenet_v2-0000.params
mobilenet_v2-symbol - Copy.json
mobilenet_v2-symbol.json
model-0000.params
model-symbol.json
model-symbol1.json
module2_cognitiveclass_python101.ipynb
movie_review.ipynb
mpg.csv
music_player.ipynb
music_player2.ipynb
my_new_file.txt
myfile
myfile.txt
numbers.txt
owlcreek.txt
pdf_file_split.py
plot_stock_market.ipynb
project_17_Sept_2020.ipynb
project_30_Sept_2020.ipynb
py0101EN_module3-5.ipynb
py101_module4-5.ipynb
pythontest.ipynb
reading_jason_file.ipynb
reaganomics.txt
regression.ipynb
resources
rps-test-set.zip
rps.zip
rps.zip.2
rps2.zip
sample.csv
sample1.csv
sample_submission.csv
screenshot_2020-10-31_at_14.30.09.png
shirt_class.ipynb
shirt_class1.ipynb
shirt_class1.py
shirt_class2.ipynb
shirt_class2.py
shirt_class3.ipynb
shirt_class3.py
shirt_class4.ipynb
shirt_class4.py
sinc plot 2.ipynb
sinc plot.ipynb
smallfile.json
smallfile.txt
speech_to_text.ipynb
split.pdf
split1.pdf
split11.pdf
stock_exchange
stock_exchange_project.ipynb
stock_exchange_project_30_Sept_2020.ipynb
stocks_21_sept_2020.ipynb
stocks_22_sept_2020.ipynb
```

```
stocks_22_sept_2020b.ipynb
sumvlistalues.py
sumvlistalues.pyc
tensorflow_training1.ipynb
test-400.jpg
test-600.jpg
test-600.png
test.txt
test.zip
testAudio.mp3
test_a35.py
test_math_func.py
text.png
text.txt
train
train.zip
udemyMLproject01.ipynb
voice.mp3
voice1.mp3
weight-height.csv
winequality-red.csv
```

In [7]:
```
!chmod 600 ~/.kaggle/kaggle.json
```

In [8]:
```
#!pip install mxnet==1.5.1
import mxnet as mx
```

In [9]:
```
!ls -1 | wc -l
```

```
382
```

In [10]:
```
# number of files in train folder
!ls /d/coding/python/train -1 | wc -l
```

```
25000
```

SQUEEZENET

In [11]:
```
#!wget https://github.com/miaow1988/SqueezeNet_v1.2/blob/master/model-0000.p
```

In [12]:
```
#!wget https://github.com/miaow1988/SqueezeNet_v1.2/blob/master/model-symbo
```

In [13]:
```
#!wget https://github.com/KeyKy/mobilenet-mxnet/blob/master/mobilenet_v2-00(
```

In [14]:
```
#!wget https://github.com/KeyKy/mobilenet-mxnet/blob/master/mobilenet_v2-syn
```

In [15]:
```
!pip install mxnet
```

```
Requirement already satisfied: mxnet in c:\users\92312\anaconda3\envs\myenv\
lib\site-packages (1.5.0)
Requirement already satisfied: numpy<1.17.0,>=1.8.2 in c:\users\92312\anacon
da3\envs\myenv\lib\site-packages (from mxnet) (1.16.6)
Requirement already satisfied: requests<2.19.0,>=2.18.4 in c:\users\92312\an
aconda3\envs\myenv\lib\site-packages (from mxnet) (2.18.4)
Requirement already satisfied: graphviz<0.9.0,>=0.8.1 in c:\users\92312\anac
onda3\envs\myenv\lib\site-packages (from mxnet) (0.8.4)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in c:\users\92312\anaco
nda3\envs\myenv\lib\site-packages (from requests<2.19.0,>=2.18.4->mxnet) (3.
0.4)
```

```
Requirement already satisfied: idna<2.7,>=2.5 in c:\users\92312\anaconda3\en
vs\myenv\lib\site-packages (from requests<2.19.0,>=2.18.4->mxnet) (2.6)
Requirement already satisfied: urllib3<1.23,>=1.21.1 in c:\users\92312\anaco
nda3\envs\myenv\lib\site-packages (from requests<2.19.0,>=2.18.4->mxnet) (1.
22)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\92312\anaconda
3\envs\myenv\lib\site-packages (from requests<2.19.0,>=2.18.4->mxnet) (2020.
```

In [16]:
```
sym, arg_params, aux_params = mx.model.load_checkpoint('model', 0)
#pretrained/SqueezeNet_v1.2
#sym, arg_params, aux_params = mx.model.load_checkpoint('pretrained/Squeezel
```

In [17]:
```
!python --version
```

```
Python 3.6.12 :: Anaconda, Inc.
```

In [18]:
```
#!pip install matplotlib
```

In [19]:
```
mod = mx.mod.Module(symbol=sym, context=mx.cpu(), label_names=None)
mod.bind(for_training=False, data_shapes=[('data', (1,3,224,224))],
         label_shapes=mod._label_shapes)
mod.set_params(arg_params, aux_params, allow_missing=True)
```

In [20]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import cv2
import numpy as np
# define a simple data batch
from collections import namedtuple
Batch = namedtuple('Batch', ['data'])
```

In [21]:
```
# list the last 10 layers
all_layers = sym.get_internals()
all_layers.list_outputs()[-10:]
```

Out[21]:
```
['fire9_concat_output',
 'dropout0_output',
 'conv10_conv_weight',
 'conv10_conv_bias',
 'conv10_conv_output',
 'conv10_relu_output',
 'pool10_output',
 'flatten0_output',
 'softmax_label',
 'softmax_output']
```

In [22]:
```
fe_sym = all_layers['flatten0_output']
fe_mod = mx.mod.Module(symbol=fe_sym, context=mx.cpu(), label_names=None)
fe_mod.bind(for_training=False, data_shapes=[('data', (1,3,224,224))])
fe_mod.set_params(arg_params, aux_params)
```

In [23]:
```
def get_features(img):
    fe_mod.forward(Batch([mx.nd.array(img)]))
    features = fe_mod.get_outputs()[0].asnumpy()
    return features
```

In [24]:
```
def get_image(url, show=False):
    if url.startswith('http'):
        # download and show the image
        fname = mx.test_utils.download(url)
    else:
        fname = url
    img = cv2.cvtColor(cv2.imread(fname), cv2.COLOR_BGR2RGB)
    if img is None:
         return None
    if show:
         plt.imshow(img)
         plt.axis('off')
    # convert into format (batch, RGB, width, height)
    img = cv2.resize(img, (224, 224))
    img = np.swapaxes(img, 0, 2)
    img = np.swapaxes(img, 1, 2)
    img = img[np.newaxis, :]
    return img
```

In [25]:
```
def predict(url):
    img = get_image(url, show=True)
    # compute the predict probabilities
    mod.forward(Batch([mx.nd.array(img)]))
    prob = mod.get_outputs()[0].asnumpy()
    # print the top-5
    prob = np.squeeze(prob)
    a = np.argsort(prob)[::-1]
    for i in a[0:5]:
        print('probability=%f, class=%s' %(prob[i], labels[i]))
```

In [26]:
```
img = get_image('https://icatcare.org/app/uploads/2018/07/Thinking-of-gettir
features = get_features(img)
print("{}\n shape: {}".format(features,features.shape))
```

```
[[ 2.731812    3.900898    4.886676    6.9023705   5.5878525   5.5976615
   3.629975    5.6956005   5.783275    5.9859443  10.328435    7.2491193
  10.0898     11.63578     5.579315   10.29896     9.659982    9.824672
   5.9974594   9.134588   10.065866   10.166415    9.808081    6.2899804
  13.240796    1.9723076   7.6495514   3.7121732   2.7841308   6.293435
   6.945307    8.631692    7.296452    6.8019      6.523342    5.698733
   8.282391    3.8308432  10.181626    8.768396   12.96276     9.041883
   9.2452     10.779444   11.305762    5.7258286  11.200753   10.964641
   3.4149983   5.8307905   7.5116653   9.025102    3.9738123   3.9767184
   8.057209    4.7041106   8.180445    4.212362    7.9470005   5.90094
   7.274863    9.651379    6.8402634   3.008466    4.3598137   4.1929665
   8.628107    3.9853954   5.85128     7.3413982   7.4339786   6.6312594
   8.691613    8.439705    8.843549    9.059312    9.449073   12.678202
  11.697754    7.8836465   7.3444376   6.4519577   6.810825   11.757463
  10.983698    7.6246886   5.5038695   7.0347304  12.915622    8.201405
   3.994503    5.410097    6.8564887   6.9246025   9.8086815   6.9710774
   4.193385    3.50595     1.6024156   4.7404633   3.3018112   2.9084263
   2.415487    5.3541484   8.748272    5.6931205   5.78507     6.12905
   5.1263065   5.9972253   7.4542603   6.640498   11.303707    7.1208987
   9.347334    8.703756    4.862454   12.2739525   6.180233    4.480562
   5.6459785   1.0452358   6.122117    5.4823976  11.299999    4.1871943
   7.71949     2.9370356   4.467441    4.9877315   5.1120286   6.40134
   9.139218    8.804351    6.144144    2.6501622   4.686039    3.7774026
  10.005825    8.510311    7.5478544   7.3255296   7.3307242   3.8225973
   4.816144    6.866867    4.5165176   7.426783    6.511624    3.6083217
   7.267048   10.171129    7.746411    5.281473    8.630497    5.626667
```

6.5024347	13.557568	9.849359	4.837752	4.333935	3.7696323
5.3609295	2.2633295	2.98707	2.4315627	3.9928892	2.2981822
3.342407	6.635318	2.192593	8.712072	4.9626703	8.373492
4.807754	2.604943	4.978068	3.2130494	7.1132894	1.5388218
2.2567573	3.458329	2.7490795	4.9099455	4.1809993	3.6504633
5.125884	7.6492615	5.345399	4.6102123	3.795777	2.694023
6.086537	5.387833	1.9049263	8.116256	7.423189	3.3709116
5.399277	6.287826	3.9034204	6.057797	4.3366733	7.391601
5.458247	3.5410907	0.8116387	4.913134	3.0773628	2.524288
3.4502816	5.5048203	5.5890627	3.111317	1.8214488	4.929328
4.658477	5.163583	3.2834659	3.5414617	2.3487625	1.1612093
4.3987837	7.205919	7.160168	5.134451	3.4099333	7.2843556
1.0771722	3.4878016	7.6779838	6.426784	6.5381455	2.581391
4.5315685	5.8846464	6.235899	7.2709856	3.3144867	5.621662
4.957663	2.6886454	7.497139	3.8266103	2.0137925	6.833844
5.250486	2.5354953	10.352083	6.991752	9.562757	5.328881
7.0152397	12.107493	7.2554445	2.0420218	2.8452964	4.244531
5.744788	5.4434156	1.5472522	3.3997128	6.343063	9.78134
8.96338	3.3579895	1.9317473	1.5271807	5.0281773	7.896086
7.7716556	10.337724	9.567745	7.9217057	7.9268084	5.3693123
6.6163344	10.761529	11.807285	9.075543	11.988959	22.696743
22.162323	12.080123	12.826471	21.835653	14.643059	20.231909
13.029128	13.020741	10.732145	10.547263	15.827932	7.679771
1.9486969	1.970214	3.0648887	3.9313178	8.43166	9.492557
6.441335	7.1024995	7.843514	9.124733	6.7014875	3.3743446
7.671623	9.547771	8.999785	10.56783	9.413501	11.694575
8.665098	8.841963	5.5585203	8.441075	9.334013	8.717619
10.550848	8.072866	7.4475703	5.9206634	12.910718	5.2324095
10.362941	6.979834	11.146726	10.812782	6.1894917	4.91079
15.443866	17.415869	11.733546	11.387103	3.4203036	11.020241
5.445361	3.4969065	6.3759475	6.3474693	13.639748	7.980881
6.675926	6.94456	7.724025	5.3664584	4.7508082	6.613133
4.9604473	6.077958	9.4021	7.6230917	10.184332	10.919795
4.3083596	9.011375	9.799607	9.7259245	5.6023235	5.6934505
7.5121617	4.707308	7.7173448	7.038019	2.5001907	8.207554
7.1257854	7.286927	4.8646	4.4201236	7.9287863	7.410235
3.3168452	4.6729064	6.1844077	4.931931	2.6935425	7.101244
2.4658318	2.89273	5.3334613	2.9348166	4.585412	7.929239
6.5128403	4.9168253	4.0626335	6.639361	2.1379588	9.514823
7.6458187	10.895908	5.6789308	5.7382855	8.675856	7.4208903
4.531518	9.101163	9.213087	11.0596075	8.179051	9.600113
8.79123	2.6726408	5.024301	2.4073465	12.150897	0.76296633
0.5038563	7.698403	4.4655457	6.7847123	10.82722	4.090029
6.0966706	8.620804	7.540544	3.5552733	13.943615	9.3506775
7.003283	11.929674	6.1212196	3.899696	5.960222	1.7010775
7.056834	4.255964	8.233021	7.9206142	8.374679	7.7616105
10.318233	6.185758	14.05891	12.324538	4.4500723	3.7979531
11.232958	10.459604	9.267933	13.071587	5.993655	9.3386965
3.3234677	7.087869	10.93612	4.7873716	8.666861	4.634499
5.0674467	13.847441	11.273701	13.505605	11.774431	11.9611635
10.948316	17.558033	7.588045	10.792004	3.4737248	13.2257595
4.9713297	14.716063	10.0185995	3.4421515	3.9179456	4.786414
3.2410672	8.168965	9.91004	4.8734303	3.3967478	16.299444
10.226325	5.2889423	3.468615	8.962406	13.654785	7.991586
4.363833	8.174878	5.608153	6.835205	3.5675814	7.301462
5.185373	10.017095	11.80642	9.212149	8.4429245	5.80141
9.667006	12.208717	11.963175	7.345216	9.081502	9.427847
8.753898	7.2916155	1.8946753	11.637853	9.241547	13.928189
13.06165	15.367341	12.914505	12.344783	11.604444	8.127106
2.5657392	2.1852176	15.490841	10.774006	16.341034	11.205923
9.197771	5.786351	4.4183025	9.605479	8.044161	11.222432
5.2172	5.975121	10.967843	3.1460893	5.208506	9.072665
7.3603683	7.686628	7.032531	3.9155047	10.085672	6.1941204
13.652328	6.0107265	2.6337419	4.9069314	7.181142	14.790271
2.8501515	7.726963	13.043632	15.613863	2.583749	9.394105

```
 8.6676035   0.41363326  4.146782    9.67872     9.279223    8.796541
16.4047     11.5193615   5.3688045   1.984938    7.5876045   6.9396825
14.072974    4.313797    4.247334    2.5274694  14.33964    14.996346
 5.0891495   2.2578015  10.512023    7.1637583   6.479706    0.6981036
 8.151602    2.126832   19.057438    0.35773605  7.0367427   0.7631183
 3.4614582   6.594459   13.109664    7.9887624   4.068507    7.1986895
 8.804501    3.9188902  11.875959   12.563932    1.2866049  10.684025
10.414188    9.318348    6.4337487  10.807209   13.9467     13.667538
10.968394    0.31435058  8.297131    7.201411    5.2648506   9.962434
12.99909     6.0167027   6.755425    4.2900424  12.476635    9.821191
13.725561    3.7854533  12.910076    1.9558824   8.380436    8.893562
 2.7058542   5.410047    7.5020194   5.542324    9.119854    9.73772
13.117701   10.940124   11.175711    1.406536   14.934693   15.472118
11.472244    2.485258   14.423505    2.9746714   2.8564594  11.77143
 6.1318235  11.445189    9.001336    9.262844    2.1180592   9.955632
 7.700891   13.581739    8.4076      9.292692    3.1939998   3.232419
 5.744683   17.093956    9.243409    3.5197144   3.9177282   9.475567
13.095526    5.068781   10.414359   10.499538    5.8909526   8.429688
 3.668364    9.282007    1.9323313   4.94462     8.410939   10.613595
 3.3637278   2.1542652   9.364938    9.538076   11.225167    2.438679
 9.073262    6.7250767   5.564789    7.6732173   6.279336    3.7686012
 1.9405166  10.818338    9.91537     1.2044476   5.1299253  12.860839
 4.7520866  11.391399   13.650659   10.387297   14.147362   12.050399
 6.563175    2.0673943   5.9914737   8.105741    6.44125     8.340171
 3.8919117   4.5356436  12.865544    5.7256727   3.525284    9.760831
15.003664    7.7152114   7.3014693   6.4310384  14.705738    3.274754
 8.713832    5.6867223   8.028869    3.5077872   5.1369505   7.5494065
12.989616    8.225569   13.512066   16.23734     8.576904    4.9031487
10.672872    7.348828   11.329117    1.7260668   4.011467   11.259492
 9.169684   11.627292   15.071683    9.140865    1.1424189  17.970129
 9.407949    2.7270186  18.140486    9.003847    2.5072134  10.935851
 4.9892483  11.738513    0.52019525  7.8873      4.24844    11.841047
11.201463    7.455634    7.2534976   8.160788    7.77645     4.2174435
 5.5322914   7.0204196   4.0031576  10.046073    8.734598   15.279126
 9.053204    2.3913186   5.732251   18.224155    7.5134535   3.9214163
 5.300208    3.0420492   6.3857594   8.329537   11.639782   17.700714
 8.512334    8.652159    7.3959537   7.1831346   5.4304075  13.718815
 6.6245522  15.252483   14.832651    6.6380954  11.767881   16.180178
 8.354548    7.03131    11.555989   13.561806    7.1899133   3.1991372
 3.8116345   2.8905377  11.651769   15.387638   14.040853    6.565446
13.462716   12.679194   10.703344   11.839084    6.4391747   7.229307
 6.290086   10.939685   17.566397    7.924627    5.3297095   7.7819676
 3.5846536  13.153733    7.882102    4.1145      1.4778726   0.9619494
11.64157     6.8478184  10.172747    4.869986   10.389148    8.457949
 6.292947   11.046795    4.5261507   9.975329    2.8246858   8.585759
 7.7183886   3.7410016   9.822101    7.828371    2.135351    4.3781753
 6.6460547   9.450028    8.374489    1.8673855   6.908634    7.6942377
10.442543    1.6138124   4.992465    5.0941358   5.5209208   3.5012095
11.521548   13.694515    4.702955    6.5822797  11.46124     2.659943
11.959797   11.355201    5.2679415   3.2046177  10.093636   12.032429
10.127256    2.4071126   6.0026193  11.668372    5.411328    9.999648
 9.526372    2.4523509   7.007791   16.54697     1.5456221   8.567241
 3.0186105  13.096735    8.174237   12.758465    9.848466    9.25708
 0.9611106   5.522036    2.5690813   2.928036   10.095376    8.211961
 3.408937    2.8309448   7.2162156   9.619771    0.22439492 10.548249
10.733221    7.318069   11.090988   11.2479925   6.309042    7.397608
 6.753635   17.710535    8.317594   11.241346    7.7849264  11.963484
 3.5570066   8.755915    5.8203      6.24455     9.4146185   8.909975
11.53808     6.4253025  17.255009   13.751666   11.622288   15.469
 4.2377596   7.7361646  10.365317   13.240438   18.205475   14.377346
 9.415743   10.363938    3.6586401   9.33291     9.287964    9.225867
10.726657    2.3643987   3.2395902   2.7732954   7.17731    10.607796
13.280322    7.78        3.3202124  10.017651    6.189192    6.967744
 5.7646823   8.264954    2.065885    9.877998    9.344263   11.44254
 5.910218    4.1088095   5.1819386   2.406357    4.4888825   4.482684
```

```
 3.3455498  12.267348    6.3040047   5.797586    5.4890037   2.9057486
 8.029715    7.3118124   5.331028    6.7807636   7.269804    7.031609
 6.9609904   8.205075    6.2120514   9.704171    4.3887997   9.3089285
 5.3341084   6.456188    5.7876287   4.358955    7.141377    4.1641693
 8.003896    8.619182    2.5742772   2.4626918   3.8087516   5.987409
10.418254    9.594039   18.342419   11.671423    2.6376793   9.242506
 2.542373    6.4728875   5.514851    5.139831    1.7740526   3.924323
 5.7800474   5.5538397   2.673251    7.5682654   8.417677    2.7631457
10.067946    9.356203    5.7668056  13.894631    7.128602    8.015947
 4.268326    7.279767    3.921046    4.57002     6.206504    5.756426
```

In [27]:
```
!pwd
```

```
/d/coding/python
```

In [28]:
```
# from https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data
from os import listdir
from os.path import isfile, join
import os


mypath = join(os.getcwd(),'train')

cats_imgs = [join(mypath,f) for f in listdir(mypath) if f.startswith('cat')]
dogs_imgs = [join(mypath,f) for f in listdir(mypath) if f.startswith('dog')]
```

In [29]:
```
print("cats: {} and dogs: {}".format(len(cats_imgs),len(dogs_imgs)))
```

```
cats: 12500 and dogs: 12500
```

In [30]:
```
!pip install sklearn
```

```
Requirement already satisfied: sklearn in c:\users\92312\anaconda3\envs\myen
v\lib\site-packages (0.0)
Requirement already satisfied: scikit-learn in c:\users\92312\anaconda3\env
s\myenv\lib\site-packages (from sklearn) (0.23.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\92312\anacon
da3\envs\myenv\lib\site-packages (from scikit-learn->sklearn) (2.1.0)
Requirement already satisfied: joblib>=0.11 in c:\users\92312\anaconda3\env
s\myenv\lib\site-packages (from scikit-learn->sklearn) (0.17.0)
Requirement already satisfied: scipy>=0.19.1 in c:\users\92312\anaconda3\env
s\myenv\lib\site-packages (from scikit-learn->sklearn) (1.5.4)
Requirement already satisfied: numpy>=1.13.3 in c:\users\92312\anaconda3\env
s\myenv\lib\site-packages (from scikit-learn->sklearn) (1.16.6)
```

In [31]:
```
# Grid search cross validation
from sklearn.model_selection import GridSearchCV
#grid={"C":np.logspace(-3,3,7), "penalty":["l1","l2"]}# l1 lasso l2 ridge
#logreg=LogisticRegression()
#logreg_cv=GridSearchCV(logreg,grid,cv=10)
#logreg_cv.fit(X_train,y_train)
#print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
#print("accuracy :",logreg_cv.best_score_)
```

In [32]:
```
from sklearn.model_selection import GridSearchCV
def GS(x_train,y_train):
    grid={"C":np.logspace(-3,3,7), "penalty":["l1","l2"]}# l1 lasso l2 ridge
    logreg=LogisticRegression()
    logreg_cv=GridSearchCV(logreg,grid,cv=10)
    logreg_cv.fit(x_train,y_train)
    return logreg_cv.best_params_,logreg_cv.best_score_
    #print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params
    #print("accuracy :",logreg_cv.best_score_)
```

N=10

In [33]:
```
%%time
import time
"the code you want to test stays here"

start10 = time.time()
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
Nmax = 12
cats_features = [get_features(get_image(img)).ravel() for img in cats_imgs[
dogs_features = [get_features(get_image(img)).ravel() for img in dogs_imgs[

Y_cats = np.array(Nmax * [1])
Y_dogs = np.array(Nmax * [0])

X_cvd = np.vstack([cats_features,dogs_features])
Y_cvd = np.vstack([Y_cats,Y_dogs]).ravel()

X_train10, X_test10, y_train10, y_test10 = train_test_split(X_cvd, Y_cvd, ra

lg10 = LogisticRegression().fit(X_train10, y_train10)

print("Test set score: {:.2f}".format(lg10.score(X_test10, y_test10)))
end10 = time.time()
print("time consumed by logisitic regression {:.2f} min {:.2f} sec".format(
# Grid Search

start10cv=time.time()
print()
print('Grid Search')
best_params10,best_score10=GS(X_train10,y_train10)
print("tuned hpyerparameters :(best parameters) {}".format((best_params10)))
print("accuracy :{:.2f}".format((best_score10)))
end10cv=time.time()
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end10
print()
```

```
Test set score: 0.83
time consumed by logisitic regression 0.00 min 1.78 sec

Grid Search
tuned hpyerparameters :(best parameters) {'C': 0.001, 'penalty': 'l2'}
accuracy :0.95
time consumed by Grid search 0.00 min 2.26 sec

Wall time: 4.04 s
```

In [ ]:

N=100

In [34]:
```
%%time
start100 = time.time()
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
Nmax = 100
cats_features = [get_features(get_image(img)).ravel() for img in cats_imgs[
dogs_features = [get_features(get_image(img)).ravel() for img in dogs_imgs[

Y_cats = np.array(Nmax * [1])
Y_dogs = np.array(Nmax * [0])

X_cvd = np.vstack([cats_features,dogs_features])
Y_cvd = np.vstack([Y_cats,Y_dogs]).ravel()

X_train100, X_test100, y_train100, y_test100 = train_test_split(X_cvd, Y_cvd

lg100 = LogisticRegression().fit(X_train100, y_train100)

print("Test set score: {:.2f}".format(lg100.score(X_test100, y_test100)))
end100 = time.time()
print("time consumed by logisitic regression {:.2f} min {:.2f} sec".format(

# Grid Search
start100cv = time.time()
print()
print('Grid Search')
best_params100,best_score100=GS(X_train100,y_train100)
print("tuned hpyerparameters :(best parameters) ",best_params100)
print("accuracy : {:.2f}".format(best_score100))
end100cv=time.time()
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end10
print()
```

```
Test set score: 0.88
time consumed by logisitic regression 0.00 min 15.01 sec

Grid Search
tuned hpyerparameters :(best parameters)  {'C': 0.1, 'penalty': 'l2'}
accuracy : 0.97
time consumed by Grid search 0.00 min 5.56 sec

Wall time: 20.6 s
```

In [ ]:

N=500

In [35]:
```
%%time
start500 = time.time()
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
Nmax = 500
cats_features = [get_features(get_image(img)).ravel() for img in cats_imgs[
dogs_features = [get_features(get_image(img)).ravel() for img in dogs_imgs[

Y_cats = np.array(Nmax * [1])
Y_dogs = np.array(Nmax * [0])

X_cvd = np.vstack([cats_features,dogs_features])
Y_cvd = np.vstack([Y_cats,Y_dogs]).ravel()

X_train500, X_test500, y_train500, y_test500 = train_test_split(X_cvd, Y_cvd

lg500 = LogisticRegression().fit(X_train500, y_train500)

print("Test set score: {:.2f}".format(lg500.score(X_test500, y_test500)))
end500 = time.time()
print("time consumed by logisitic regression {:.2f} min {:.2f} sec".format(

# Grid Search
start500cv = time.time()
print()
print('Grid Search')
best_params500,best_score500=GS(X_train500,y_train500)
print("tuned hpyerparameters :(best parameters) ",best_params500)
print("accuracy : {:.2f}".format(best_score500))
end500cv=time.time()
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end50
print()
```

```
Test set score: 0.94
time consumed by logisitic regression 1.00 min 33.52 sec

Grid Search
tuned hpyerparameters :(best parameters)  {'C': 100.0, 'penalty': 'l2'}
accuracy : 0.96
time consumed by Grid search 0.00 min 16.24 sec

Wall time: 1min 49s
```

In [ ]:

N=1000

In [36]:
```
%%time
start1000 = time.time()
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
Nmax = 1000
cats_features = [get_features(get_image(img)).ravel() for img in cats_imgs[
dogs_features = [get_features(get_image(img)).ravel() for img in dogs_imgs[

Y_cats = np.array(Nmax * [1])
Y_dogs = np.array(Nmax * [0])

X_cvd = np.vstack([cats_features,dogs_features])
Y_cvd = np.vstack([Y_cats,Y_dogs]).ravel()

X_train1000, X_test1000, y_train1000, y_test1000 = train_test_split(X_cvd, Y

lg1000 = LogisticRegression().fit(X_train1000, y_train1000)

print("Test set score: {:.2f}".format(lg1000.score(X_test1000, y_test1000)))
end1000 = time.time()
print("time consumed by logisitic regression {:.2f} min {:.2f} sec".format(

# Grid Search
start1000cv = time.time()
print()
print('Grid Search')
best_params1000,best_score1000=GS(X_train1000,y_train1000)
print("tuned hpyerparameters :(best parameters) ",best_params1000)
print("accuracy : {:.2f}".format(best_score1000))
end1000cv=time.time()
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end1(
```

```
Test set score: 0.95
time consumed by logisitic regression 3.00 min 32.60 sec

Grid Search
tuned hpyerparameters :(best parameters)  {'C': 0.001, 'penalty': 'l2'}
accuracy : 0.95
time consumed by Grid search 0.00 min 30.92 sec
Wall time: 4min 3s
```

In [ ]:

N=5000

In [37]:
```
%%time
start5000 = time.time()
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
Nmax = 5000
cats_features = [get_features(get_image(img)).ravel() for img in cats_imgs[
dogs_features = [get_features(get_image(img)).ravel() for img in dogs_imgs[

Y_cats = np.array(Nmax * [1])
Y_dogs = np.array(Nmax * [0])

X_cvd = np.vstack([cats_features,dogs_features])
Y_cvd = np.vstack([Y_cats,Y_dogs]).ravel()

X_train5000, X_test5000, y_train5000, y_test5000 = train_test_split(X_cvd, Y

lg5000 = LogisticRegression().fit(X_train5000, y_train5000)

print("Test set score: {:.2f}".format(lg5000.score(X_test5000, y_test5000)))
end5000 = time.time()
print("time consumed by logisitic regression {:.2f} min {:.2f} sec".format(

# Grid Search
start5000cv = time.time()
print()
print('Grid Search')
best_params5000,best_score5000=GS(X_train5000,y_train5000)
print("tuned hpyerparameters :(best parameters) ",best_params5000)
print("accuracy : {:.2f}".format(best_score5000))
end5000cv=time.time()
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end50
```

```
Test set score: 0.95
time consumed by logisitic regression 20.00 min 12.87 sec

Grid Search
tuned hpyerparameters :(best parameters)  {'C': 0.001, 'penalty': 'l2'}
accuracy : 0.97
time consumed by Grid search 1.00 min 31.52 sec
Wall time: 21min 44s
```

In [ ]:

N=12500

In [38]:
```
%%time
start12500 = time.time()
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
Nmax = 12500
cats_features = [get_features(get_image(img)).ravel() for img in cats_imgs[
dogs_features = [get_features(get_image(img)).ravel() for img in dogs_imgs[

Y_cats = np.array(Nmax * [1])
Y_dogs = np.array(Nmax * [0])

X_cvd = np.vstack([cats_features,dogs_features])
Y_cvd = np.vstack([Y_cats,Y_dogs]).ravel()

X_train12500, X_test12500, y_train12500, y_test12500 = train_test_split(X_cv

lg12500 = LogisticRegression().fit(X_train12500, y_train12500)

print("Test set score: {:.2f}".format(lg12500.score(X_test12500, y_test12500
end12500 = time.time()
print("time consumed by logisitic regression {:.2f} min {:.2f} sec".format(

# Grid Search
start12500cv = time.time()
print()
print('Grid Search')
best_params12500,best_score12500=GS(X_train12500,y_train12500)
print("tuned hpyerparameters :(best parameters) ",best_params12500)
print("accuracy : {:.2f}".format(best_score12500))
end12500cv=time.time()
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end12
```

```
Test set score: 0.97
time consumed by logisitic regression 39.00 min 50.22 sec

Grid Search
tuned hpyerparameters :(best parameters)  {'C': 0.001, 'penalty': 'l2'}
accuracy : 0.97
time consumed by Grid search 3.00 min 29.30 sec
Wall time: 43min 19s
```

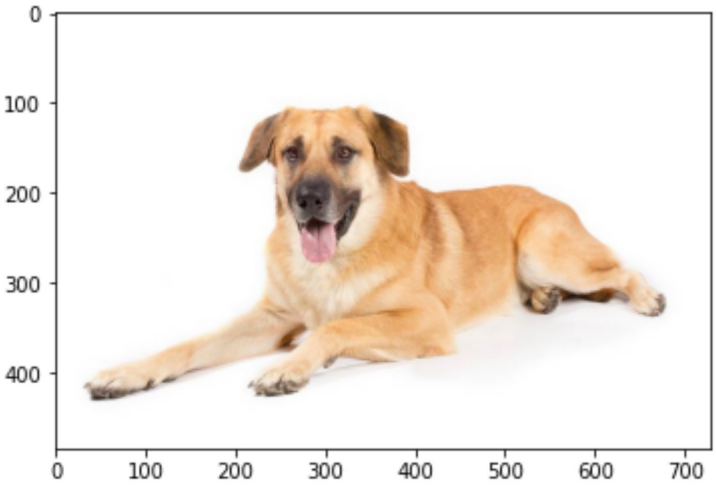
In [ ]:

In [39]:
```
print()
print('N=10')
print("Test set score: {:.2f}".format(lg10.score(X_test10, y_test10)))
print("tuned hpyerparameters :(best parameters) ",best_params10)
print("accuracy :{:.2f}".format((best_score10)))
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end1(
print()
print('N=100')
print("Test set score: {:.2f}".format(lg100.score(X_test100, y_test100)))
print("tuned hpyerparameters :(best parameters) ",best_params100)
print("accuracy : {:.2f}".format(best_score100))
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end1(
print()
print('N=500')
print("Test set score: {:.2f}".format(lg500.score(X_test500, y_test500)))
print("tuned hpyerparameters :(best parameters) ",best_params500)
print("accuracy : {:.2f}".format(best_score500))
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end5(
print()
print('N=1000')
print("Test set score: {:.2f}".format(lg1000.score(X_test1000, y_test1000)))
print("tuned hpyerparameters :(best parameters) ",best_params1000)
print("accuracy : {:.2f}".format(best_score1000))
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end1(
print()
print('N=5000')
print("Test set score: {:.2f}".format(lg5000.score(X_test5000, y_test5000)))
print("tuned hpyerparameters :(best parameters) ",best_params5000)
print("accuracy : {:.2f}".format(best_score5000))
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end5(
print()
print('N=12500')
print("Test set score: {:.2f}".format(lg12500.score(X_test12500, y_test1250(
print("tuned hpyerparameters :(best parameters) ",best_params12500)
print("accuracy : {:.2f}".format(best_score12500))
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end12
```

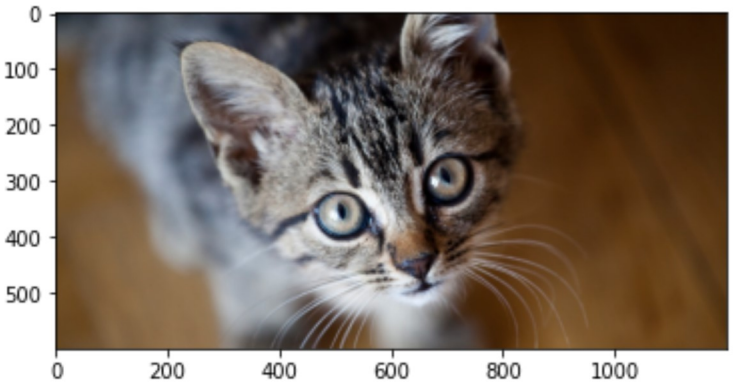
```
N=10
Test set score: 0.83
tuned hpyerparameters :(best parameters)  {'C': 0.001, 'penalty': 'l2'}
accuracy :0.95
time consumed by Grid search 0.00 min 2.26 sec

N=100
Test set score: 0.88
tuned hpyerparameters :(best parameters)  {'C': 0.1, 'penalty': 'l2'}
accuracy : 0.97
time consumed by Grid search 0.00 min 5.56 sec

N=500
Test set score: 0.94
tuned hpyerparameters :(best parameters)  {'C': 100.0, 'penalty': 'l2'}
accuracy : 0.96
time consumed by Grid search 0.00 min 16.24 sec

N=1000
Test set score: 0.95
tuned hpyerparameters :(best parameters)  {'C': 0.001, 'penalty': 'l2'}
accuracy : 0.95
time consumed by Grid search 0.00 min 30.92 sec
```

```
N=5000
Test set score: 0.95
tuned hpyerparameters :(best parameters)  {'C': 0.001, 'penalty': 'l2'}
accuracy : 0.97
time consumed by Grid search 1.00 min 31.52 sec

N=12500
Test set score: 0.97
tuned hpyerparameters :(best parameters)  {'C': 0.001, 'penalty': 'l2'}
accuracy : 0.97
time consumed by Grid search 3.00 min 29.30 sec
```

In [40]:
```
import matplotlib.image as mpimg

dog_test_path = join(os.getcwd(),'Chinook-On-White-03.jpg')
cat_test_path = join(os.getcwd(),'Thinking-of-getting-a-cat.png')

img = mpimg.imread(dog_test_path)
imgplot = plt.imshow(img)
plt.show()
```



In [41]:
```
img = mpimg.imread(cat_test_path)
imgplot = plt.imshow(img)
plt.show()
```



In [42]:
```
features_out = get_features(get_image(dog_test_path))

prob = lg1000.predict_proba(features_out)
pred = lg1000.predict(features_out)

if pred[0] == 1:
    fpred = 'cat'
else:
    fpred = 'dog'

print("prob: {} and prediction: {}".format(prob,fpred))
```

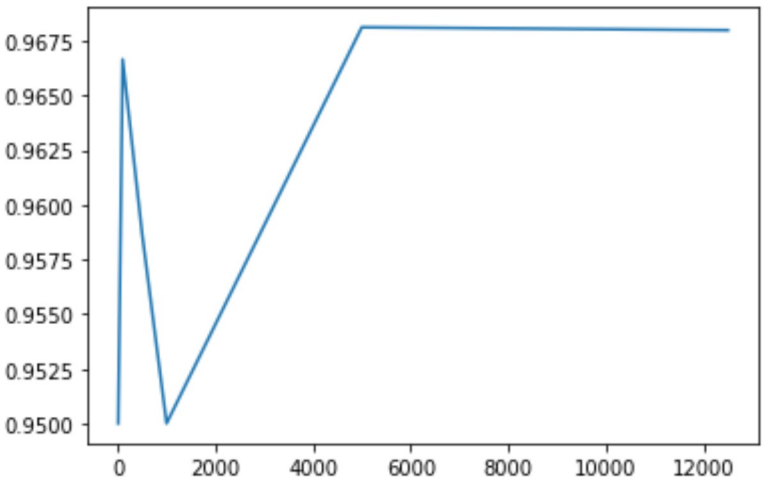
```
prob: [[1.00000000e+00 4.90122721e-18]] and prediction: dog
```

In [43]:
```
features_out = get_features(get_image(cat_test_path))

prob = lg1000.predict_proba(features_out)
pred = lg1000.predict(features_out)

if pred[0] == 1:
    fpred = 'cat'
else:
    fpred = 'dog'

print("prob: {} and prediction: {}".format(prob,fpred))
```

```
prob: [[3.33066907e-15 1.00000000e+00]] and prediction: cat
```

In [45]:
```
x1=[10, 100, 500, 1000, 5000, 12500]
y1=[best_score10, best_score100, best_score500, best_score1000, best_score5(
#y2=[best_score10b, best_score100b, best_score500b, best_score1000b, best_s
plt.plot(x1,y1)
```

Out[45]:
```
[<matplotlib.lines.Line2D at 0x14e259ff320>]
```



MOBILENET

In [46]:
```
sym2, arg_params2, aux_params2 = mx.model.load_checkpoint('mobilenet_v2', 0)
```

In [47]:
```
mod2 = mx.mod.Module(symbol=sym2, context=mx.cpu(), label_names=None)
mod2.bind(for_training=False, data_shapes=[('data', (1,3,224,224))],
          label_shapes=mod2._label_shapes)
mod2.set_params(arg_params2, aux_params2, allow_missing=True)
```

In [48]:
```
# list the last 10 layers
all_layers2 = sym2.get_internals()
all_layers2.list_outputs()[-10:]
```

Out[48]:
```
['conv6_4_bn_moving_var',
 'conv6_4_bn_output',
 'relu6_4_output',
 'pool6_output',
 'fc7_weight',
 'fc7_bias',
 'fc7_output',
 'fc7_flatten_output',
 'prob_label',
 'prob_output']
```

In [49]:
```
all_layers2.list_outputs()[-3]
```

Out[49]:
```
'fc7_flatten_output'
```

In [50]:
```
fe_sym2 = all_layers2['fc7_flatten_output']
fe_mod2 = mx.mod.Module(symbol=fe_sym2, context=mx.cpu(), label_names=None)
fe_mod2.bind(for_training=False, data_shapes=[('data', (1,3,224,224))])
fe_mod2.set_params(arg_params2, aux_params2)
```

In [51]:
```
def Task2(Nmax):
    cats_features = [get_features(get_image(img)).ravel() for img in cats_ir
    dogs_features = [get_features(get_image(img)).ravel() for img in dogs_ir

    Y_cats = np.array(Nmax * [1])
    Y_dogs = np.array(Nmax * [0])

    X_cvd = np.vstack([cats_features,dogs_features])
    Y_cvd = np.vstack([Y_cats,Y_dogs]).ravel()
    return X_cvd,Y_cvd
```

N=10

In [52]:
```
%%time
start10b = time.time()
X_cvd10,Y_cvd10=Task2(12)
X_train10b, X_test10b, y_train10b, y_test10b = train_test_split(X_cvd10, Y_c
lg10b = LogisticRegression().fit(X_train10b, y_train10b)
print("Test set score: {:.2f}".format(lg10b.score(X_test10b, y_test10b)))
end10b = time.time()
print("time consumed by logisitic regression {:.2f} min {:.2f} sec".format(

# Grid Search
start10bcv=time.time()
print()
print('Grid Search')
best_params10b,best_score10b=GS(X_train10b,y_train10b)
print("tuned hpyerparameters :(best parameters) {}".format((best_params10b))
print("accuracy :{:.2f}".format((best_score10b)))
end10bcv=time.time()
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end10
print()
```

```
Test set score: 0.83
time consumed by logisitic regression 0.00 min 1.77 sec

Grid Search
tuned hpyerparameters :(best parameters) {'C': 0.001, 'penalty': 'l2'}
accuracy :0.95
time consumed by Grid search 0.00 min 2.18 sec

Wall time: 3.95 s
```

N=100

In [53]:
```
%%time
start100b = time.time()
X_cvd100,Y_cvd100=Task2(100)
X_train100b, X_test100b, y_train100b, y_test100b = train_test_split(X_cvd10(
lg100b = LogisticRegression().fit(X_train100b, y_train100b)
print("Test set score: {:.2f}".format(lg100b.score(X_test100b, y_test100b)))
end100b = time.time()
print("time consumed by logisitic regression {:.2f} min {:.2f} sec".format(

# Grid Search
start100bcv=time.time()
print()
print('Grid Search')
best_params100b,best_score100b=GS(X_train100b,y_train100b)
print("tuned hpyerparameters :(best parameters) {}".format((best_params100b)
print("accuracy :{:.2f}".format((best_score100b)))
end100bcv=time.time()
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end10
print()
```

```
Test set score: 0.88
time consumed by logisitic regression 0.00 min 13.80 sec

Grid Search
tuned hpyerparameters :(best parameters) {'C': 0.1, 'penalty': 'l2'}
accuracy :0.97
time consumed by Grid search 0.00 min 5.23 sec
```

N=500

In [54]:
```
%%time
start500b = time.time()
X_cvd500,Y_cvd500=Task2(500)
X_train500b, X_test500b, y_train500b, y_test500b = train_test_split(X_cvd50(
lg500b = LogisticRegression().fit(X_train500b, y_train500b)
print("Test set score: {:.2f}".format(lg500b.score(X_test500b, y_test500b)))
end500b = time.time()
print("time consumed by logisitic regression {:.2f} min {:.2f} sec".format(

# Grid Search
start500bcv=time.time()
print()
print('Grid Search')
best_params500b,best_score500b=GS(X_train500b,y_train500b)
print("tuned hpyerparameters :(best parameters) {}".format((best_params500b)
print("accuracy :{:.2f}".format((best_score500b)))
end500bcv=time.time()
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end50(
print()
```

```
Test set score: 0.94
time consumed by logisitic regression 1.00 min 8.03 sec

Grid Search
tuned hpyerparameters :(best parameters) {'C': 100.0, 'penalty': 'l2'}
accuracy :0.96
time consumed by Grid search 0.00 min 14.57 sec

Wall time: 1min 22s
```

N=1000

In [55]:
```
%%time
start1000b = time.time()
X_cvd1000,Y_cvd1000=Task2(1000)
X_train1000b, X_test1000b, y_train1000b, y_test1000b = train_test_split(X_cv
lg1000b = LogisticRegression().fit(X_train1000b, y_train1000b)
print("Test set score: {:.2f}".format(lg1000b.score(X_test1000b, y_test1000b
end1000b = time.time()
print("time consumed by logisitic regression {:.2f} min {:.2f} sec".format(

# Grid Search
start1000bcv=time.time()
print()
print('Grid Search')
best_params1000b,best_score1000b=GS(X_train1000b,y_train1000b)
print("tuned hpyerparameters :(best parameters) {}".format((best_params1000b
print("accuracy :{:.2f}".format((best_score1000b)))
end1000bcv=time.time()
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end10(
print()
```

```
Test set score: 0.95
time consumed by logisitic regression 2.00 min 29.71 sec
```

```
Grid Search
tuned hpyerparameters :(best parameters) {'C': 0.001, 'penalty': 'l2'}
accuracy :0.95
time consumed by Grid search 0.00 min 21.35 sec
```

N=5000

In [56]:
```
%%time
start5000b = time.time()
X_cvd5000,Y_cvd5000=Task2(5000)
X_train5000b, X_test5000b, y_train5000b, y_test5000b = train_test_split(X_cv
lg5000b = LogisticRegression().fit(X_train5000b, y_train5000b)
print("Test set score: {:.2f}".format(lg5000b.score(X_test5000b, y_test5000b
end5000b = time.time()
print("time consumed by logisitic regression {:.2f} min {:.2f} sec".format(i

# Grid Search
start5000bcv=time.time()
print()
print('Grid Search')
best_params5000b,best_score5000b=GS(X_train5000b,y_train5000b)
print("tuned hpyerparameters :(best parameters) {}".format((best_params5000b
print("accuracy :{:.2f}".format((best_score5000b)))
end5000bcv=time.time()
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end50
print()
```

```
Test set score: 0.95
time consumed by logisitic regression 13.00 min 5.90 sec

Grid Search
tuned hpyerparameters :(best parameters) {'C': 0.001, 'penalty': 'l2'}
accuracy :0.97
time consumed by Grid search 1.00 min 22.17 sec

Wall time: 14min 28s
```

N=12500

In [57]:
```
%%time
start12500b = time.time()
X_cvd12500,Y_cvd12500=Task2(12500)
X_train12500b, X_test12500b, y_train12500b, y_test12500b = train_test_split
lg12500b = LogisticRegression().fit(X_train12500b, y_train12500b)
print("Test set score: {:.2f}".format(lg12500b.score(X_test12500b, y_test125
end12500b = time.time()
print("time consumed by logisitic regression {:.2f} min {:.2f} sec".format(

# Grid Search
start12500bcv=time.time()
print()
print('Grid Search')
best_params12500b,best_score12500b=GS(X_train12500b,y_train12500b)
print("tuned hpyerparameters :(best parameters) {}".format((best_params12500
print("accuracy :{:.2f}".format((best_score12500b)))
end12500bcv=time.time()
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end12
print()
```

```
Test set score: 0.97
time consumed by logisitic regression 33.00 min 46.91 sec

Grid Search
tuned hpyerparameters :(best parameters) {'C': 0.001, 'penalty': 'l2'}
accuracy :0.97
time consumed by Grid search 3.00 min 31.02 sec

Wall time: 37min 17s
```

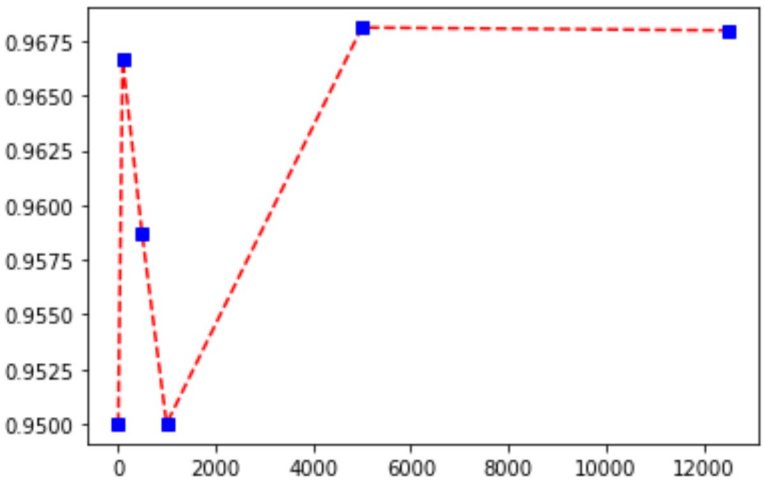
In [58]:
```
print()
print('N=10')
print("Test set score: {:.2f}".format(lg10b.score(X_test10b, y_test10b)))
print("tuned hpyerparameters :(best parameters) ",best_params10b)
print("accuracy :{:.2f}".format((best_score10b)))
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end1(
print()
print('N=100')
print("Test set score: {:.2f}".format(lg100b.score(X_test100b, y_test100b)))
print("tuned hpyerparameters :(best parameters) ",best_params100b)
print("accuracy : {:.2f}".format(best_score100b))
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end1(
print()
print('N=500')
print("Test set score: {:.2f}".format(lg500b.score(X_test500b, y_test500b)))
print("tuned hpyerparameters :(best parameters) ",best_params500b)
print("accuracy : {:.2f}".format(best_score500b))
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end5(
print()
print('N=1000')
print("Test set score: {:.2f}".format(lg1000b.score(X_test1000b, y_test1000k
print("tuned hpyerparameters :(best parameters) ",best_params1000b)
print("accuracy : {:.2f}".format(best_score1000b))
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end1(
print()
print('N=5000')
print("Test set score: {:.2f}".format(lg5000b.score(X_test5000b, y_test5000k
print("tuned hpyerparameters :(best parameters) ",best_params5000b)
print("accuracy : {:.2f}".format(best_score5000b))
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end5(
print()
print('N=12500')
print("Test set score: {:.2f}".format(lg12500b.score(X_test12500b, y_test125
print("tuned hpyerparameters :(best parameters) ",best_params12500b)
print("accuracy : {:.2f}".format(best_score12500b))
print("time consumed by Grid search {:.2f} min {:.2f} sec".format(int((end12
```

```
N=10
Test set score: 0.83
tuned hpyerparameters :(best parameters)  {'C': 0.001, 'penalty': 'l2'}
accuracy :0.95
time consumed by Grid search 0.00 min 2.18 sec

N=100
Test set score: 0.88
tuned hpyerparameters :(best parameters)  {'C': 0.1, 'penalty': 'l2'}
accuracy : 0.97
time consumed by Grid search 0.00 min 5.23 sec

N=500
Test set score: 0.94
tuned hpyerparameters :(best parameters)  {'C': 100.0, 'penalty': 'l2'}
accuracy : 0.96
time consumed by Grid search 0.00 min 14.57 sec

N=1000
Test set score: 0.95
tuned hpyerparameters :(best parameters)  {'C': 0.001, 'penalty': 'l2'}
accuracy : 0.95
time consumed by Grid search 0.00 min 21.35 sec
```

```
N=5000
Test set score: 0.95
tuned hpyerparameters :(best parameters)  {'C': 0.001, 'penalty': 'l2'}
accuracy : 0.97
time consumed by Grid search 1.00 min 22.17 sec

N=12500
Test set score: 0.97
tuned hpyerparameters :(best parameters)  {'C': 0.001, 'penalty': 'l2'}
accuracy : 0.97
time consumed by Grid search 3.00 min 31.02 sec
```

In [61]:
```
x1=[10, 100, 500, 1000, 5000, 12500]
y1=[best_score10, best_score100, best_score500, best_score1000, best_score5(
y2=[best_score10b, best_score100b, best_score500b, best_score1000b, best_sc(
plt.plot(x1,y1,'r--',x1,y2,'bs')
```

Out[61]:
```
[<matplotlib.lines.Line2D at 0x14e39c277b8>,
 <matplotlib.lines.Line2D at 0x14e39c27828>]
```



In [67]:
```
print('Accuracy of Best score of Grid Search')
for i in range(len(x1)):
    print("{} {}   SqueezeNET: {:.2f} MobileNET: {:.2f}".format(i,x1[i],y1[i
```

```
Accuracy of Best score of Grid Search
0 10   SqueezeNET: 0.95 MobileNET: 0.95
1 100   SqueezeNET: 0.97 MobileNET: 0.97
2 500   SqueezeNET: 0.96 MobileNET: 0.96
3 1000   SqueezeNET: 0.95 MobileNET: 0.95
4 5000   SqueezeNET: 0.97 MobileNET: 0.97
5 12500   SqueezeNET: 0.97 MobileNET: 0.97
```

The values are found to be almost same for both cases as shown in the graph above and the calculations above

In [ ]: