Machine Learning Basics

(HPRC)

Narendra Chaudhary

TEXAS A&M UNIVERSITY
High Performance Research Computing

Summer Computing Academy 2018

Machine Learning

- Recognizing objects in images. What is a dog ?
- Understanding language(Alexa, Siri, Cortana)
- Self driving cars
- Playing games (Chess, AlphaGo)
- Other applications where we have lot of data.
- Giant robots taking over the world ???

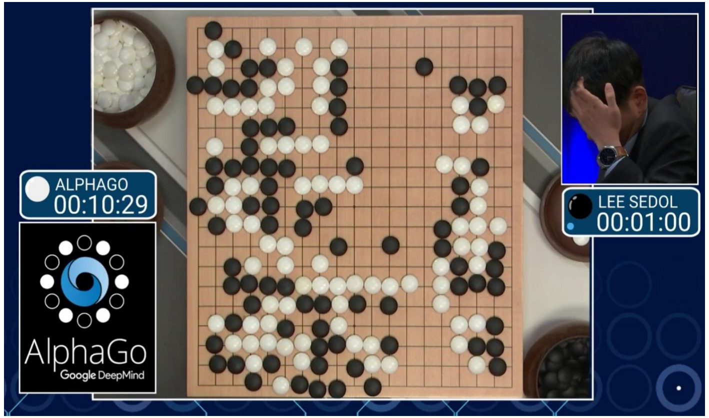


Image (to computer)

- What is an image to a computer ?
- Answer - Bunch of numbers just like everything else.
- Grayscale image - Sheet of numbers (2D array) ranging from 0 to 255.
- Lower number - dark area
- Higher number - bright area
- Color image - 3 different sheets of numbers (Red, Green and Blue). 3D array
- Image Dimension = (Height, Width, channels)

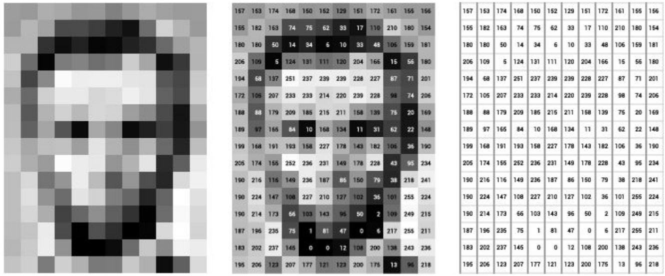
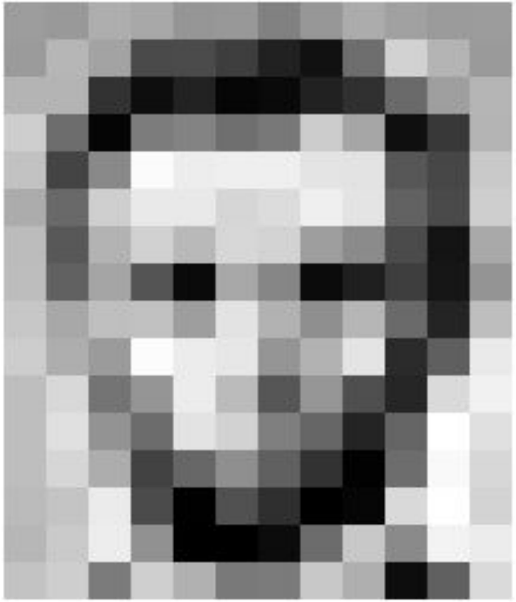


Image (numbers)



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

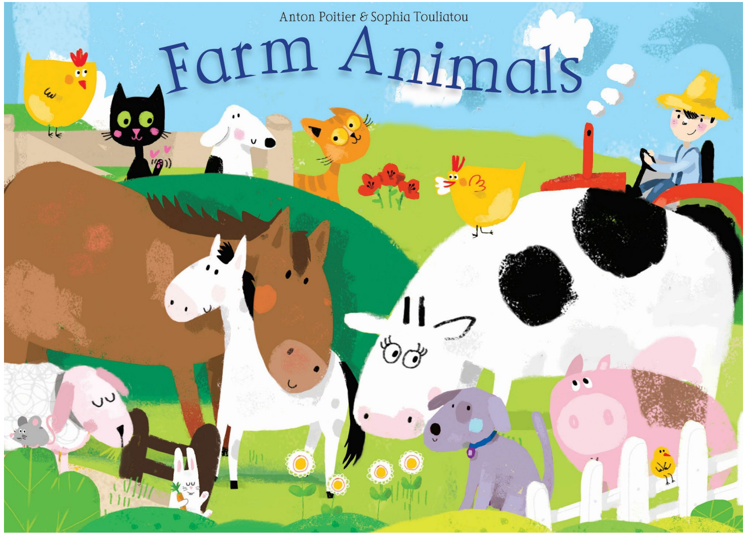
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Why Machine learning ?

- What is a horse ? Can you define it ?
  - 4 legs
  - A tail
- Very hard to write a program which can recognize objects in images. Like a horse, a car or a flower.
- Bunch of numbers == horse ???
- Humans and animals know instantly what a horse is
- Even though they can't define it

Learning

- We know horse, cat, lion because we learned it.
- From teachers and parents telling us what a horse is
- Children's book
- Learning by various images and their labels
- Supervised learning



Car

Cat

Horse

Supervised Machine Learning

- Supervised machine learning works the same way as learning from children's book
- What do I need to do supervised machine learning on computers ?
- A Machine brain - Model
- Training data - Images and their labels (Children's book)
- Methods (functions) in python to train
  - Libraries like keras, tensorflow, scikit-learn

Installation of Libraries

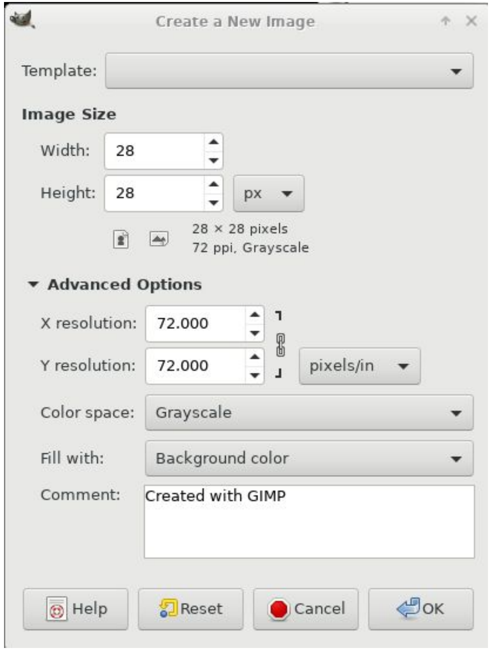
1. Google “Scientific Python for Raspberry Pi”. http://geoffboeing.com/2016/03/scientific-python-raspberry-pi/ follow instruction from step 3 of this link.
2. Open the terminal on raspberry pi and write the following commands.
3. sudo apt-get update
4. sudo apt-get upgrade
5. dpkg -l > ~/Desktop/packages.list
6. pip freeze > ~/Desktop/pip-freeze-initial.list
7. sudo apt-get install build-essential python-dev python-distlib python-setuptools python-pip python-wheel libzmq-dev libgdal-dev
8. sudo apt-get install xsel xclip libxml2-dev libxslt-dev python-lxml python-h5py python-numexpr python-dateutil python-six python-tz python-bs4 python-html5lib python-openpyxl python-tables python-xlrd python-xlwt cython python-sqlalchemy python-xlsxwriter python-jinja2 python-boto python-gflags python-googleapi python-httplib2 python-zmq libspatialindex-dev
9. sudo pip install bottleneck rtree
10. sudo apt-get install python-numpy python-matplotlib python-mpltoolkits.basemap python-scipy python-sklearn python-statsmodels python-pandas
11. sudo pip install jupyter
12. sudo pip install -U ipython
13. sudo pip install tensorflow==1.1.0
14. sudo pip install keras==2.0.6

Alternative Installation

- Download bash file. https://github.com/narendrachaudhary51/Teach_machine_learning/blob/master/installation.sh
- Open terminal
- Run ./installation.sh command

Activity 1 - Create a character image by gimp

- Open terminal and write “sudo apt-get install gimp”
- Create a folder rename it to “Teach_machine_learning”.
- CD to the folder and write command “gimp” in the terminal.
- Go to file->New
- In the window, write Width = 28 and Height = 28
- In Advanced options pick colorspace as “Grayscale”
- Zoom in, make black background color and write a number(0-9) with white brush at the center of image.
- Export as PNG image and save it in the folder.
- We will try to recognize this number using machine learning.



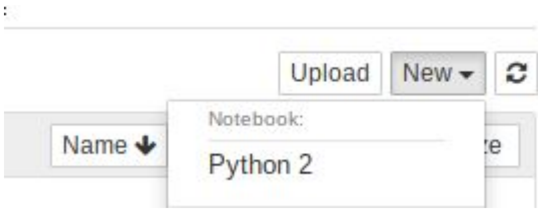
Activity 2 - Training a simple model

In this activity, we will learn following things

- All reference code. You can clone it using git clone command. https://github.com/narendrachaudhary51/Teach_machine_learning
- Import important libraries
- How to load handwritten number dataset MNIST?
- Check dataset and learn about train and test dataset. Reshape data.
- Train a simple model (machine brain) on this dataset
- Check how good the trained model is ?
- How much time it takes ?
- How to predict the label of images ?
- Can it recognize image with number created in Activity 1.

Activity 2 - Training a simple model

- In terminal, write this command. “jupyter notebook --no-browser”
- Copy the generated url in the web browser
- Start a new python2 notebook. From top right.
- In first cell import the following
- Try to run code in jupyter




```
In [1]: import time
        from PIL import Image
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.linear_model import LogisticRegression
        %matplotlib inline
```

Important libraries

Python Imaging library

- You can import a function or module using “from” instead of entire library

```
In [1]: import time
        from PIL import Image
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.linear_model import LogisticRegression
        %matplotlib inline
```

Numpy library

- Used for multidimensional array.
- Import as “np”
- Array operation functions
- Python only had lists earlier

matplotlib.pyplot library

- Used for making plots
- Showing images
- Import as “plt”

scikit-learn library

- Machine learning library
- We only import LogisticRegression model from linear models

Activity 2 - Training a simple model

- Import the handwritten numbers training dataset (MNIST) from keras library.
- The data is split into training set and test set.
- X arrays contains images while Y arrays contain labels (numpy arrays)

```
In [2]: from keras.datasets import mnist
        (x_train, y_train), (x_test, y_test) = mnist.load_data()      # import dataset

Using TensorFlow backend.
```

- x_train, y_train like exercise in the book
- x_test, y_test like exam. Model should not know what will come in the exam.

Activity 2 - Training a simple model

- Check the shape of data
- Change its type
- Normalize - Image number in range (0,1).
- data shape -
  60000 = number of images
- Image width = 28
- Image height = 28
- 10000 test images and labels

In [3]:
```
print("Training data shape: ", x_train.shape)
print("Training labels shape: ", y_train.shape)
print("Test data shape: ", x_test.shape)
print("Test labels shape: ", y_test.shape)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255                                  #make range (0-1)
x_test /= 255                                   #make range (0-1)
```

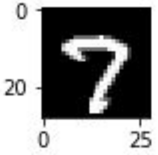
```
('Training data shape: ', (60000, 28, 28))
('Training labels shape: ', (60000,))
('Test data shape: ', (10000, 28, 28))
('Test labels shape: ', (10000,))
```

Activity 2 - Training a simple model

- Let’s check one of the training image out of the 60000

```
Image_number = 52                                    # pick a image number
plt.figure(figsize = (1,1))
plt.imshow(x_train[Image_number], cmap = 'gray')             # Show the grayscale image
print("Image_label:", y_train[Image_number])
```

```
('Image_label:', 7)
```



- Image is of number 7 and it has the label number 7.
- We are good to proceed

Activity 2 - Training a simple model

- Reshape the data
- Data reshaped to two dimensions
- Create a model (your little machine brain)

```
x_train = x_train.reshape(60000,28*28)      # change the shape of Training images
x_test = x_test.reshape(10000,28*28)        # Change the shape of Testing images

print("Training data shape: ", x_train.shape)
print("Training labels shape: ", y_train.shape)
print("Test data shape: ", x_test.shape)
print("Test labels shape: ", y_test.shape)
```

```
('Training data shape: ', (60000, 784))
('Training labels shape: ', (60000,))
('Test data shape: ', (10000, 784))
('Test labels shape: ', (10000,))
```

```
train_samples = 5000
model = LogisticRegression(C=50. / train_samples,
                           multi_class='multinomial',
                           penalty='l2', solver='sag', tol=0.1)
```

Activity 2 - Training a simple model

- Train using model.fit function
- fit function requires training images and labels as inputs
- Train on subset of training image. For example - we train on 20000 images in the following code. Also try on 100, 500, 1000, 10000.
- Check the training time by time.time() function

```
start = time.time()  # To calculate training time, Start

model.fit(x_train[:20000],y_train[:20000])          # Training function

end = time.time()   # To calculate training time, Stop
print ('Training time', end - start)
```

```
('Training time', 40.37853693962097)
```

Activity 2 - Training a simple model

- After training, check your score(accuracy) on test set.

```
score = model.score(x_test,y_test)                    # Check accuracy on test set
print(score)
```

```
0.9146
```

- Use model.predict function to predict label of one of the test image.
- You will reshape the data again.

```
Test_image_number = 52                # Test image number
print("Correct Image_label:", y_test[Test_image_number])          # Correct Image label

Predicted_label = model.predict(x_test[Test_image_number].reshape(1,28*28))
print("Predicted Label: ", Predicted_label)

plt.figure(figsize = (1,1))
plt.imshow(x_test[Test_image_number].reshape(28,28), cmap = 'gray') # reshape image and show
```

```
('Correct Image_label:', 5)
('Predicted Label: ', array([5], dtype=uint8))
<matplotlib.image.AxesImage at 0x64c4f370>
```

Activity 2 - Training a simple model

- Now your model (Machine brain) can find the number in an image.
- Import the gimp image you created into code
- Use model.predict function to predict the label of the image.
- Does it work ?

```
gimp_image = np.array(Image.open('gimp_image.png'))

plt.figure( figsize = (1,1))
plt.imshow(gimp_image, cmap = 'gray')
print("Predicted Label: ", model.predict(gimp_image.reshape(1,28*28)))
```

Activity 3: Train a convolutional neural network

- Perform the similar tasks of Activity 2
- Use a more complicated convolutional neural network model, in place of simple model
- Data is reshaped differently, normalization
- More intense use of keras library functions
- Keras library uses tensorflow library at backend
- Goal is to improve accuracy
- Also check the difference in training time

Activity 3: Train a convolutional neural network

- In last activity, score and accuracy was low.
- This was the consequence of simple model (Like an ant brain).
- In this activity, we will use a neural network model which is more complicated. (Like an animal brain).
- We will follow process similar to last activity. So first import libraries.

In [1]:
```
import numpy as np
import math
import matplotlib.pyplot as plt
import keras
from keras.models import Sequential,load_model
from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D, BatchNormalization
from keras.callbacks import ModelCheckpoint, EarlyStopping
from PIL import Image
import time
%matplotlib inline
```

Activity 3: Train a convolutional neural network

- Import the MNIST dataset
- Split training and test dataset
- Check their shape

```
In [2]: # the data, split between train and test sets
        from keras.datasets import mnist
        (x_train, y_train), (x_test, y_test) = mnist.load_data()

        print("Training data shape: ", x_train.shape)
        print("Training labels shape: ", y_train.shape)
        print("Test data shape: ", x_test.shape)
        print("Test labels shape: ", y_test.shape)

        Training data shape:  (60000, 28, 28)
        Training labels shape:  (60000,)
        Test data shape:  (10000, 28, 28)
        Test labels shape:  (10000,)
```

Activity 3: Train a convolutional neural network

- This time reshape training and test image data into 4-dimensional tensor
- Data Shape = (image number, height, width, channels)
- Normalize data image data

In [3]:

```
# Define some variables
img_rows, img_cols = 28, 28          # input image dimensions
num_classes = 10                     # 10 classes
input_shape = (img_rows, img_cols, 1)  # shape = (height, width, channels)

x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

Activity 3: Train a convolutional neural network

- Use the keras utilities to convert labels into “one hot vector”
- Training label data shape changes from (10000,) to (10000,10)
- Size of vector same as number of categories.
- 1 at index of the label and 0 everywhere.

```
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

print("Training data shape: ", x_train.shape)
print("Training labels shape: ", y_train.shape)
print("Test data shape: ", x_test.shape)
print("Test labels shape: ", y_test.shape)
print(y_test[0])
```

```
Training data shape:  (60000, 28, 28, 1)
Training labels shape:  (600000, 10)
Test data shape:  (10000, 28, 28, 1)
Test labels shape:  (100000, 10)
[ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

Activity 3: Train a convolutional neural network

- Simple convolutional neural network code.
- Copy the code at this point.
- Batch size - According to memory
- Epochs - How many times train model on dataset

```
In [4]: batch_size = 128
        epochs = 12

        model = Sequential()
        model.add(Conv2D(32, kernel_size=(3, 3),
                         activation='relu',
                         input_shape=input_shape))
        model.add(Conv2D(64, (3, 3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))
        model.add(Flatten())
        model.add(Dense(128, activation='relu'))
        model.add(Dropout(0.5))
        model.add(Dense(num_classes, activation='softmax'))

        model.compile(loss=keras.losses.categorical_crossentropy,
                      optimizer=keras.optimizers.Adadelta(),
                      metrics=['accuracy'])
```

Activity 3: Train a convolutional neural network

- This peace of code saves the model to 'MNIST_ConvNet.h5' model file.
- The code will save the model whenever the accuracy improves on validation set.
- It will also stop the training early if accuracy doesn't improve.

In [5]:
```
# Save the model according to the conditions
checkpoint = ModelCheckpoint('MNIST_ConvNet.h5', monitor='val_acc', verbose=1, save_best_only=True,
                             save_weights_only=False, mode='auto', period=1)
early = EarlyStopping(monitor='val_acc', min_delta=0, patience=10, verbose=1, mode='auto')
```

Activity 3: Train a convolutional neural network

- The "model.fit" function with batch size, epochs, validation data (test data here) and callbacks as input.
- The output shows the progress of training, accuracy and loss.
- Training will run till your lunch. You can stop it after that.

```
In [*]: model.fit(x_train, y_train,
                  batch_size=batch_size,
                  epochs=epochs,
                  verbose=1,
                  validation_data=(x_test, y_test),
                  callbacks = [checkpoint, early])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
 2816/60000 [>.............................] - ETA: 329s - loss: 1.6492 - acc: 0.4805
```

Activity 3: Train a convolutional neural network

- After training, load the saved model
- Use this model to get accuracy on test set.
- Use model.evaluate function.
- Use model.predict to predict label of image
- Input image should be reshaped to 4-dimension.

```
In [7]: model = load_model('MNIST_ConvNet.h5')
```

```
In [8]: score = model.evaluate(x_test, y_test, verbose=0)
        print('Test loss:', score[0])
        print('Test accuracy:', score[1])

Test loss: 0.0551423161302
Test accuracy: 0.9826
```

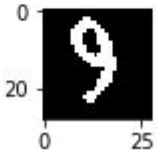
```
In [17]: Test_image_number = 41

         print("Image_label:", y_test[Test_image_number].argmax())
         print("Predicted Label: ", model.predict(x_test[Test_image_number].reshape(1,28,28,1)).argmax())
         plt.figure(figsize = (1,1))
         plt.imshow(x_test[Test_image_number].reshape(28,28), cmap = 'gray')
```

Activity 3: Train a convolutional neural network

- Predict the label of gimp image you created.
- This was example of simple machine learning you can do on raspberry pi.

In [12]:
```
gimp_image = np.array(Image.open('gimp_image.png'))
plt.figure( figsize = (1,1))
plt.imshow(gimp_image, cmap = 'gray')
print("Predicted Label: ", model.predict(gimp_image.reshape(1,28,28,1)/255).argmax())
```

```
Predicted Label:  9
```



Activity 4: Using Imagenet Models

- We only saw small grayscale images.
- What about color images ?
- What about more complicated object ?
- What about that dog and horse ??
- Training deep models on large color images takes lot of time.
- Facilities like HPRC needed
- Good thing - Many researchers have already trained models on 1000 Imagenet classes
- You can use those models on Pi

Activity 4: Using Imagenet Models

- Import needed libraries
- Models available for only some image dimensions
- We use width = 224, height = 224.
- Import a trained model named Mobilenet

```
In [1]: import numpy as np
        import math
        import matplotlib.pyplot as plt
        import keras
        from PIL import Image
        %matplotlib inline

Using TensorFlow backend.
```

```
In [2]: img_width, img_height = 224, 224
```

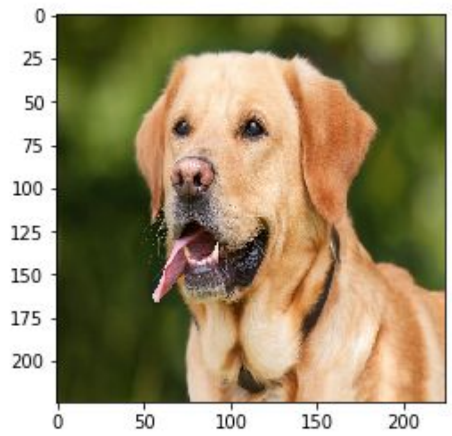
```
In [4]: # import a model trained with Imagenet
        model = keras.applications.mobilenet.MobileNet(input_shape=(img_width, img_height, 3), alpha=0.5,
                                                       include_top=True, weights='imagenet')

Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.6/mobilenet_5_0_224_tf.h5
5365760/5577668 [===========================>..] - ETA: 0s
```

Activity 4: Using Imagenet Models

- Download the image of the dog from internet
- Open the image, convert to RGB
- Resize the image to (224,224) and convert to numpy array
- Plot the image
- Change the shape to make it 4-dimensional. (1, 224, 224, 3)
- Normalize it.

```
In [4]:
#IM = Image.open(data_dir + '/roses/3903276582_fe05bf84c7_n.jpg')
IM = Image.open('dog-1210559_960_720.jpg').convert("RGB")
npim = np.array(IM.resize((img_height, img_width), Image.NEAREST))
print(npim.shape)
plt.imshow(npim)
npim = npim.reshape((1,img_height, img_width,3))/255.0
```

```
(224, 224, 3)
```



Activity 4: Using Imagenet Models

- Use model.predict function to get the label probability
- Use keras utility of decode_prediction to get names of top three labels
- Model predicts ~83 % chance that it is a Labrador_retriever
- Play with other images
- Take images from your phone, move them to Pi by email or other way
- Try to predict their labels.

```
In [5]: preds = model.predict(npim)
        print (keras.applications.mobilenet.decode_predictions(preds, top=3)[0])

[('n02099712', 'Labrador_retriever', 0.82896763), ('n02087394', 'Rhodesian_ridgeback', 0.094310038), ('n02099601', 'golden_retr
iever', 0.052263264)]
```

Activity 5: Transfer learning

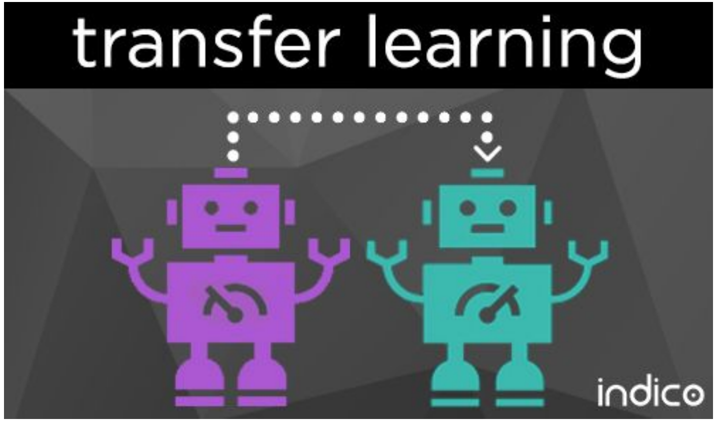
- We still can't get our own categories.
- What if I want to use machine learning to label different flower photos ?
- We may not have such classes in 1000 Imagenet classes
- Also less images available for training
- Don't have cluster or GPUs to train.
- We only have a Pi
- Use transfer learning

Daisy

Rose

Activity 5: Transfer learning

- Transfer learning procedure
- Download a Imagenet model without the top layer
- Make these layers non-trainable
- Add some new layers
- Train the network with new layers
- We basically transfer the knowledge learned by Imagenet models to our new model.



Activity 5: Transfer learning

- Let’s first get the flower dataset
- Copy following link in web browser and save
- http://download.tensorflow.org/example_images/flower_photos.tgz
- Extract the data
- Do ls command in flower_photos folder
- The command should display
- Check images in the folders
- Do they have same size ??

```
daisy/
dandelion/
roses/
sunflowers/
tulip/
LICENSE.txt
```

dandelion

Activity 5: Transfer learning

- Import the libraries and functions
- Specify some variables and flower_photos directory

```
import numpy as np
import math
import matplotlib.pyplot as plt
import keras
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential,load_model, Model
from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D, BatchNormalization
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, TensorBoard, EarlyStopping
from PIL import Image
%matplotlib inline
```

```
img_width, img_height = 224, 224
data_dir = "/home/pi/Documents/Teach_machine_learning/flower_photos"
batch_size = 8
epochs = 10
nb_train_samples = 3000
nb_validation_samples = 300
```

Activity 5: Transfer learning

- Import a model trained on Imagenet without include_top.
- This will download the model
- Make layers non-trainable

```python
# import a model trained with Imagenet
model = keras.applications.mobilenet.MobileNet(input_shape=(img_width, img_height, 3), alpha=0.5,
                                               include_top=False, weights='imagenet')
```

```python
# Freeze the layers which you don't want to train.
for layer in model.layers:
    layer.trainable = False
```

Activity 5: Transfer learning

- Adding extra layers
- The final layer output should be equal to number of classes
- In flower_phots there are 5 classes of flowers
- The model_final is our new model
- Compile model_final

```
#Adding custom Layers
x = model.output
x = Flatten()(x)
x = Dense(256, activation="relu")(x)
x = Dropout(0.5)(x)
x = Dense(256, activation="relu")(x)
predictions = Dense(5, activation="softmax")(x)

# creating the final model
model_final = Model(input = model.input, output = predictions)
```

```
# compile model
model_final.compile(loss = "categorical_crossentropy",
                    optimizer = keras.optimizers.adam(lr=0.0001),
                    metrics=["accuracy"])
```

Activity 5: Transfer learning

- Generator code
- Copy this code
- Reads the images from folders and resizes them at training
- Training generator and validation generator
- No need to read data by yourself

```
datagen = ImageDataGenerator(
rescale = 1./255,
horizontal_flip = True,
fill_mode = "nearest",
zoom_range = 0.3,
width_shift_range = 0.3,
height_shift_range=0.3,
rotation_range=30)

train_generator = datagen.flow_from_directory(data_dir,
                                              target_size=(img_width, img_height),
                                              shuffle=True, seed=13,
                                              class_mode='categorical',
                                              batch_size=batch_size)
validation_generator = datagen.flow_from_directory(data_dir,
                                              target_size=(img_width, img_height),
                                              shuffle=True, seed=13,
                                              class_mode='categorical',
                                              batch_size=batch_size)
```

```
Found 3670 images belonging to 5 classes.
Found 3670 images belonging to 5 classes.
```

Activity 5: Transfer learning

- Checkpoint and early stopping same as before
- We use the fit_generator function this time.
- Let training run for whole night or stop after some epochs

```
# Save the model according to the conditions
checkpoint = ModelCheckpoint("transfer_mobilenet_split.h5", monitor='val_acc', verbose=1, save_best_only=True,
                             save_weights_only=False, mode='auto', period=1)
early = EarlyStopping(monitor='val_acc', min_delta=0, patience=10, verbose=1, mode='auto')
```

```
# Train the model
model_final.fit_generator(
    train_generator,
    epochs = epochs,
    samples_per_epoch = nb_train_samples,
    validation_data = validation_generator,
    nb_val_samples = nb_validation_samples,
    callbacks = [checkpoint, early])
```

Activity 5: Transfer learning

- Load the model
- Some extra code specific to mobilenet

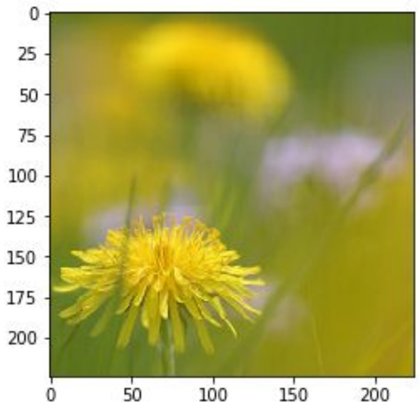
```
from keras.utils.generic_utils import CustomObjectScope

with CustomObjectScope({'relu6': keras.applications.mobilenet.relu6,'DepthwiseConv2D': keras.applications.mobilenet.Depthwi
    model_final = load_model('transfer_mobilenet_split.h5')
```

Activity 5: Transfer learning

- Read a image
- Resize it
- Plot it
- Change to numpy array
- Normalize it and change the shape to 4-dimension

```
IM = Image.open(data_dir + '/dandelion/14060367700_fe87e99b6a_m.jpg')

npim = np.array(IM.resize((img_height, img_width), Image.NEAREST))
print(npim.shape)
plt.imshow(npim)
npim = npim.reshape((1,img_height, img_width,3))/255.0
```

```
(224, 224, 3)
```



Activity 5: Transfer learning

- Use the predict function to predict the label probabilities
- Plot these probabilities
- Now you can play with other images
- Try your image
- Which flower are you ?

```
prob = model_final.predict(npim)
print(prob[0,:])
labels = ('daisy', 'dandelion', 'roses', 'sunflowers', 'tulips')
y_pos = np.arange(len(labels))
plt.bar(y_pos, prob[0,:])
plt.xticks(y_pos, labels)
plt.ylabel('Probability')
plt.title('Which flower ?')

plt.show()
```

```
[  3.08370392e-04   9.99653459e-01   2.43296336e-06   3.29769864e-05
   2.76349988e-06]
```