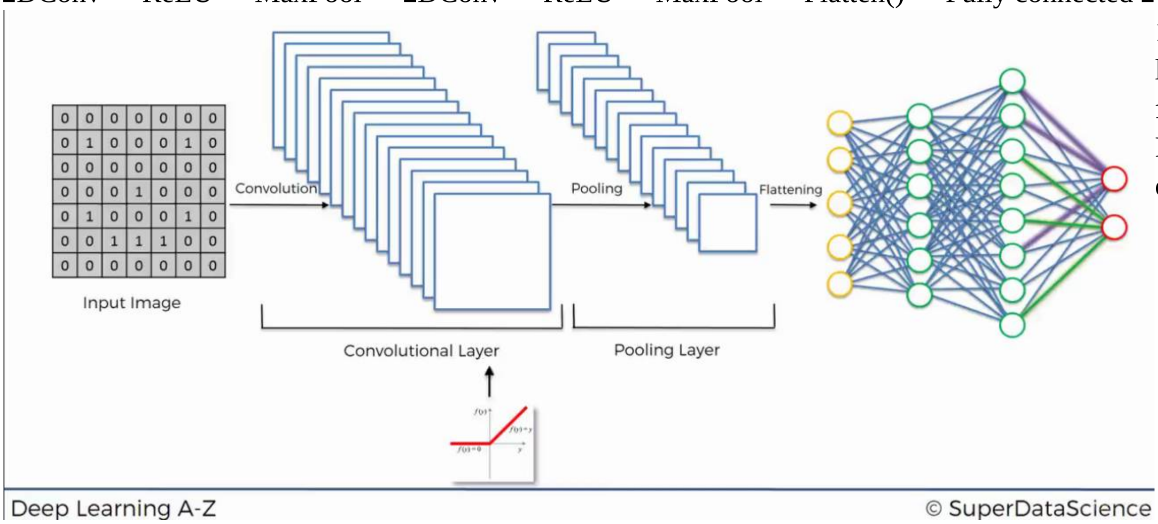
Below are important research of neural networks which will be implemented in the code:
Steps to create a neural network:
1. Learn a model that generates sensory data rather than classifying it. Eliminates the need for large amounts of labeled data.
2. Learn one layer of representation at a time using restricted boltzmann machines. This decomposes the overall learning task into multiple simpler tasks and eliminates the inference problems that arise in generative models.
3. Use a separate fine-tuning stage to improve the generative or discriminative abilities of the composite model.
A combination of these ideas leads to a novel and effective way of learning multiple layers of representation.
- Geoffrey E. Hinton
Optimization:
Steps to improve on a neural network from Geoffrey E. Hinton:
Allow higher-level feature detectors to communicate their needs to lower-level ones whilst also being easy to implement in layered networks of stochastic binary neurons that have activation states of 1 or 0 turned on with a probability that is a smooth non-linear function of the total input they receive.
Without the layer-by-layer learning, fine-tuning alone is hopelessly slow. Instead of fine-tuning the model to be a better at generating data, back-propagation can be used to fine-tune it to be better at discrimination. This works well.
To infer a probability distribution over the various possible settings of the hidden variables.
Gaussian distribution, Restricted Boltzmann Machines.
Learning feature detectors
The optimizer function in Kera's classifier.compile(optimizer, loss, metrics) is the algorithm you are going to use to find the optimal set of weights of the network. The "adam" optimizer using stochastic gradient descent algorithm that's efficient. What about the rmsprop? It computes the single gradient in batches and is slower. A sigmoid loss function is similar to logistic regression. After weight updates, the model uses metrics accuracy to improve the model's performance.

CNN Architecture:
2DConv -> ReLU -> MaxPool -> 2DConv -> ReLU -> MaxPool -> Flatten() -> Fully connected 2-layer neural network



128 neurons for the first layer -> ReLU -> 128 for hidden layer -> ReLU -> 3 neurons for output layer -> softmax

Learning/Training
The training process will use the cross-entropy error with activation functions of sigmoid or softmax. The softmax produces probability of the output. The starting loss, given at training, need to be consistent with the number of classes in the network. The training process will use stochastic gradient where the gradient is computed per input instead of in a batch. I will also try rmsprop, which is a batch training. I also forgot to use the prediction function if the output is 0/1 but that can be adjusted for a multi-class output. Here's an example from the "Deep-Learning in Python" on-line lecture that uses a simple ANN:

```
#Part 3: Making predictions and evaluating the model
#Predicting the test results
        y_prediction = classifier.predict(x_test_scaled)
```

y_prediction = (y_prediction > 0.5)
#neural network's final output will be true if the activation function is greater than 0.5, which means greater than 50% chance of leaving the bank
#Predicting a single new observation
new_prediction = classifier.predict(sc.transform(np.array([[0.0,0,600,1,40,3,60000,2,1,1,50000]])))
new_prediction = (new_prediction > 0.5)
#Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred) #so far we just split your dataset into a training set and a test set

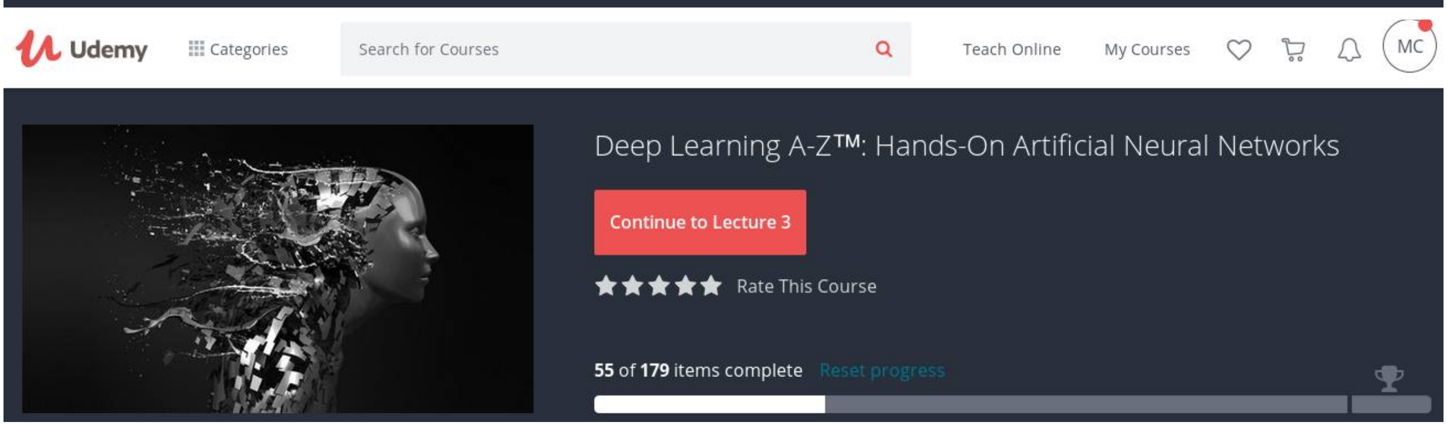
The variance problem of using validation sets is because validation sets can represent very different accuracy on another test, which is very inconsistent. Judging model on just one accuracy and one test set is not super relevant for knowing how well the model does in terms of loss, accuracy and generalization. The K-Fold Cross Validation will fix this variance problem because it splits the training set into 10 folds where k = 10 in 10 different iterations. Nine folds will represent the training set and 1 fold is to test the neural network. It is much more relevant because it takes the average.

First few weeks of September:

Research on Neural Network's and programming in Python

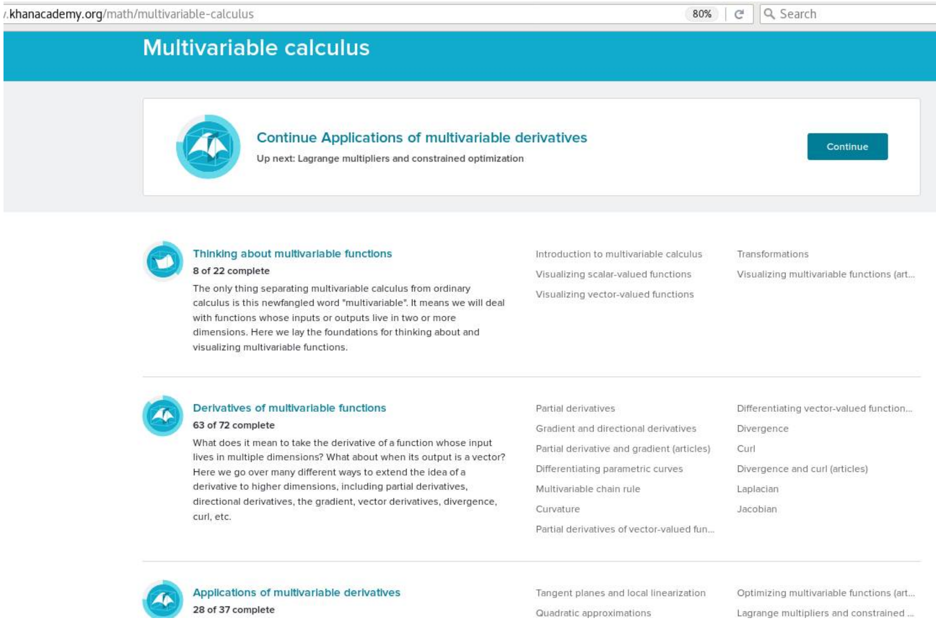
Paid $100 to go to an in-person group for deep learning, which uses the cloud to train on images of cats and dogs. The lecturer told me I should use Tensorflow or one of the popular libraries. Since I'm interested in extracting features of shapes for the neural network to learn, he told me that a convolutional network will do the job. This is because a convolutional neural network is designed to learn the pixels of images in a three dimensional output space. It does this by pooling and flattening the layers of a constant pixel size, or use padding if the size doesn't fit the dimensions of the image.

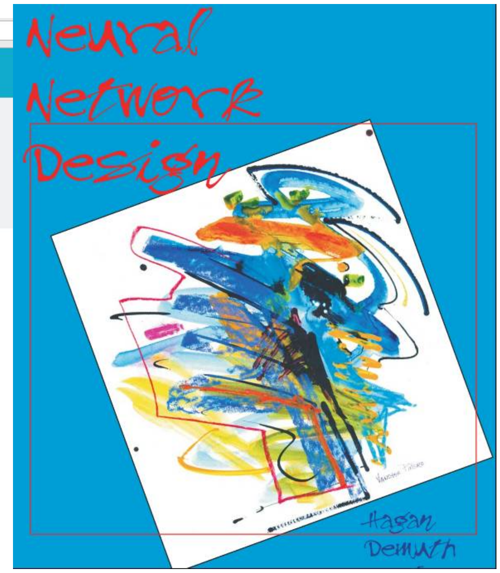
Last 3 weeks of September:



I spent this time taking udemy's online courses in learning the basics of python, first two week's of Andrew Ng's machine learning course. I have tried training a basic convolutional neural network of cats and dogs using the tutorial online but since my laptop doesn't have a Nvidia GPU I can't use GPU computation locally. It will take a couple of days just to get the output of the convolutional network.

First 3 weeks of October:





I decided to use the machine learning library Keras instead because it uses Tensorflow (in python 3) and Theano (in python 2) as backend. I spent 3 weeks reading Hagan's Neural Network Design book (2 weeks), reviewing on linear algebra (1 week) and learning and taking notes on multi-variable calculus on Kahn academy (1 week).

Week of October 23:

The baby AI image dataset is very old and has bugs in it. I wasn't able to extract the dataset by running their python program. So, I spent all this time creating my own dataset and preparing it for loading using pickle's serialization format into Google Cloud's Machine Learning Engine. I created my own python class called Draw.py, which uses multiprocessing of Pool workers in a class to draw images themselves as well as the intersection of images. Multiprocessing allows me to make as many images as possible by using parallel computing of 4 cores in a CPU.

draw.py

```
"""
Created on Fri Oct 27 18:41:03 2017
Draws images of shapes of circles, rectangles, squares, triangles
@author: maggie
"""
from __future__ import print_function
import cairo
import random
from multiprocessing import Pool

def draw_objects(object, filename):
    canvas = object(500,500)
    obj1 = canvas.background_color()
    canvas.fill_circle(obj1)
    obj2 = canvas.new_context()
    canvas.line_circle(obj2)
    obj3 = canvas.new_context()
    canvas.line_triangle(obj3)
    obj4 = canvas.new_context()
    canvas.fill_triangle(obj4)
    obj5 = canvas.new_context()
    canvas.line_rectangle(obj5)
    obj6 = canvas.new_context()
    canvas.fill_rectangle(obj6)
    obj7 = canvas.new_context()
    canvas.line_square(obj7)
    obj8 = canvas.new_context()
    canvas.fill_square(obj8)
    canvas.save_img(filename)

def draw_rectangle(object, filename):
    canvas = object(500,500)
    bg_obj = canvas.background_color()
    canvas.line_rectangle(bg_obj)
    context_obj = canvas.new_context()
    canvas.fill_rectangle(context_obj)
    canvas.save_img(filename)

def draw_circle(object, filename):
    canvas = object(500,500)
    bg_obj = canvas.background_color()
    canvas.fill_circle(bg_obj)
    context_obj = canvas.new_context()
    canvas.line_circle(context_obj)
```

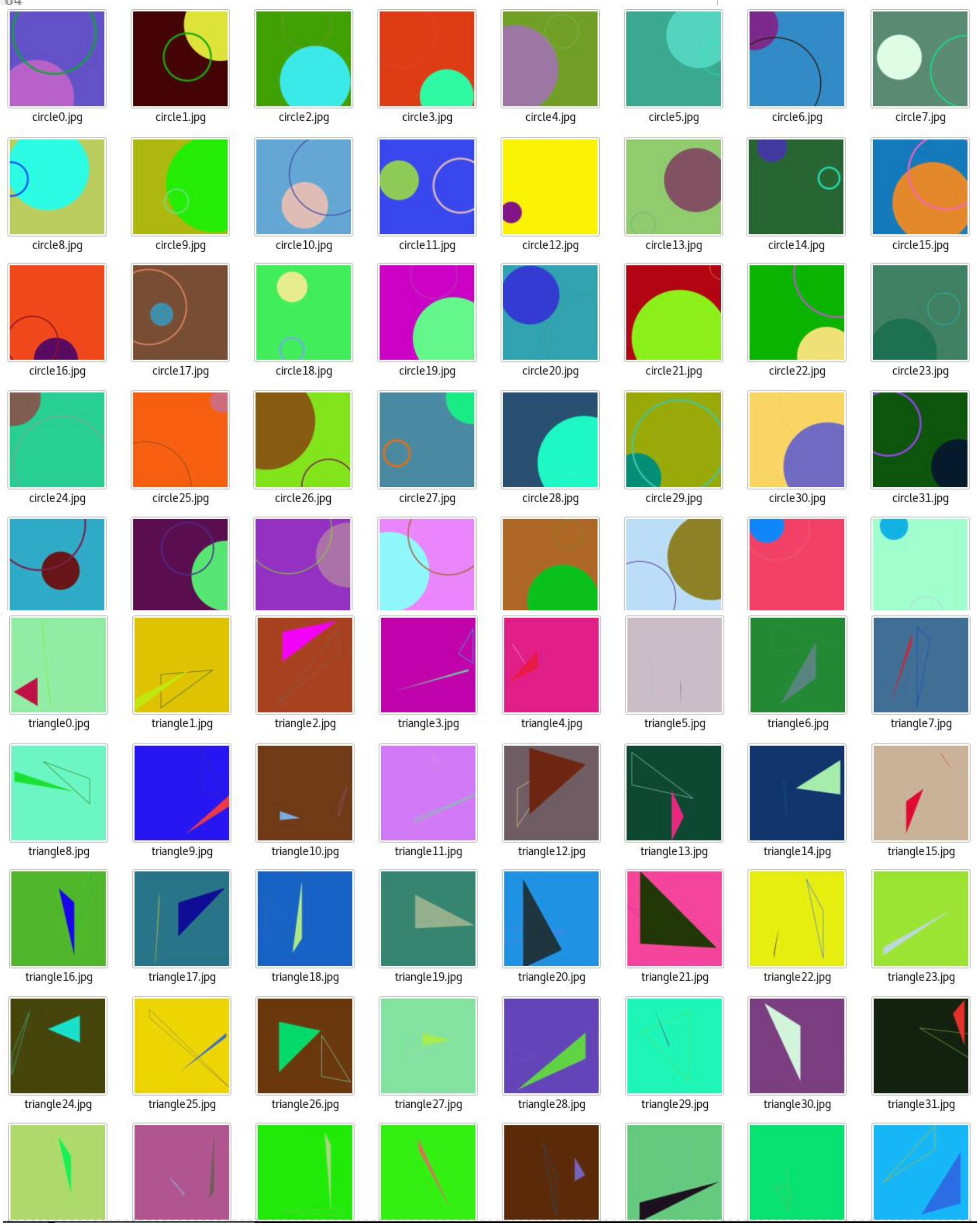
```
    canvas.save_img(filename)

def draw_square(object, filename):
    canvas = object(500,500)
    bg_obj = canvas.background_color()
    canvas.line_square(bg_obj)
    context_obj = canvas.new_context()
    canvas.fill_square(context_obj)
    canvas.save_img(filename)

def draw_triangle(object, filename):
    canvas = object(500,500)
    bg_obj = canvas.background_color()
    canvas.line_triangle(bg_obj)
    context_obj = canvas.new_context()
    canvas.fill_triangle(context_obj)
    canvas.save_img(filename)

class Draw(object):

    def __init__(self, canvas_width, canvas_height):
        import numpy as np
        self.canvas_width = canvas_width
        self.canvas_height = canvas_height
        self.data = np.zeros ((self.canvas_width, self.canvas_height, 4),
                              dtype = np.uint64)
        self.surface = cairo.ImageSurface.create_for_data(self.data,
                                                          cairo.FORMAT_ARGB32,
                                                          self.canvas_width,
                                                          self.canvas_height)

    def run(self):
        p = Pool(processes=4)

        for x in range(2000):
            p.apply_async(draw_triangle, (Draw,str(x)) )
        p.close()
        p.join()

    def new_context(self):
        return cairo.Context(self.surface)
```

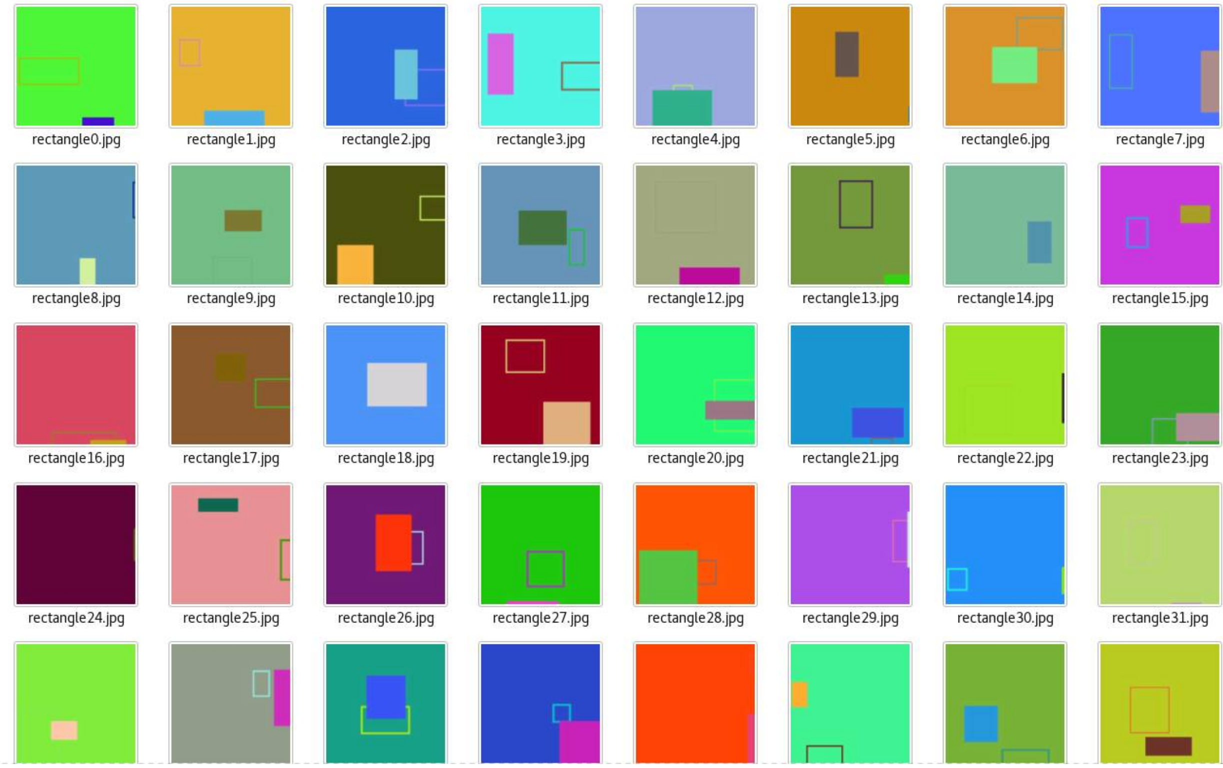
```
    #the next image drawn on background color must have
    #the parameter in background_color
    def background_color(self):
        r = random.uniform(0,1)
        g = random.uniform(0,1)
        b = random.uniform(0,1)
        name = self.new_context()
        name.set_source_rgb(r,g,b)
        name.paint()
        return name

    def fill_circle(self, object):
        import math
        r = random.uniform(0,1)
        g = random.uniform(0,1)
        b = random.uniform(0,1)
        xc = random.randint(10,500)
        yc = random.randint(10,500)
        radius = random.randint(50,250)
        object.arc(xc, yc, radius, 0, 2*math.pi)
        object.set_source_rgb(r, g, b)
        object.fill()

    def line_circle(self, object):
        import math
        r = random.uniform(0,1)
        g = random.uniform(0,1)
        b = random.uniform(0,1)
        xc = random.randint(10,500)
        yc = random.randint(10,500)
        radius = random.randint(50,250)
        w = random.uniform(0,10)
        object.arc(xc, yc, radius, 0, 2*math.pi)
        object.set_line_width(w)
        object.set_source_rgb(r, g, b)
        object.stroke()

    def fill_rectangle(self, object):
        r = random.uniform(0,1)
        g = random.uniform(0,1)
        b = random.uniform(0,1)
        x = random.randint(10,500)
        y = random.randint(10,500)
        width = random.randint(50,250)
        height = random.randint(50,250)
        object.rectangle(x, y, width, height)
        object.set_source_rgb(r, g, b)
        object.fill()

    def line_rectangle(self, object):
        r = random.uniform(0,1)
        g = random.uniform(0,1)
        b = random.uniform(0,1)
        x = random.randint(10,500)
        y = random.randint(10,500)
        w = random.uniform(0,10)
        width = random.randint(50,250)
        height = random.randint(50,250)
        object.rectangle(x, y, width, height)
        object.set_line_width(w)
        object.set_source_rgb(r, g, b)
        object.stroke()

    def fill_square(self, object):
        r = random.uniform(0,1)
        g = random.uniform(0,1)
        b = random.uniform(0,1)
        x = random.randint(10,500)
        y = random.randint(10,500)
        width = random.randint(50,250)
        object.rectangle(x, y, width, width)
        object.set_source_rgb(r, g, b)
        object.fill()

    def line_square(self, object):
        r = random.uniform(0,1)
        g = random.uniform(0,1)
        b = random.uniform(0,1)
        x = random.randint(10,500)
        y = random.randint(10,500)
        w = random.uniform(0,10)
        width = random.randint(50,250)
        object.rectangle(x, y, width, width)
        object.set_line_width(w)
        object.set_source_rgb(r, g, b)
        object.stroke()

    def fill_triangle(self, object):
        r = random.uniform(0,1)
        g = random.uniform(0,1)
        b = random.uniform(0,1)
        x = random.randint(10,500)
        y = random.randint(10,500)
        x1 = random.randint(10,500)
        y1 = random.randint(10,500)
        y2 = random.randint(10,500)
        object.move_to(x,y)
        object.line_to(x, y1)
        object.line_to(x1, y2)
        object.line_to(x, y)
        object.set_source_rgb(r, g, b)
        object.fill()

    def line_triangle(self, object):
        r = random.uniform(0,1)
        g = random.uniform(0,1)
        b = random.uniform(0,1)
        x = random.randint(10,500)
        y = random.randint(10,500)
        x1 = random.randint(10,500)
        y1 = random.randint(10,500)
        y2 = random.randint(10,500)
        w = random.uniform(0,3)
        object.move_to(x,y)
        object.line_to(x, y1)
        object.line_to(x1, y2)
        object.line_to(x, y)
        object.set_line_width(w)
        object.set_source_rgb(r, g, b)
        object.stroke()

    def save_img(self, filename):
        print (filename)
        dir = "test_set/triangle/"
        intersection = "triangle."
        self.surface.write_to_png(dir + intersection + filename + ".png")
if __name__ == '__main__':
    d = Draw(500, 500)
    d.run()
```

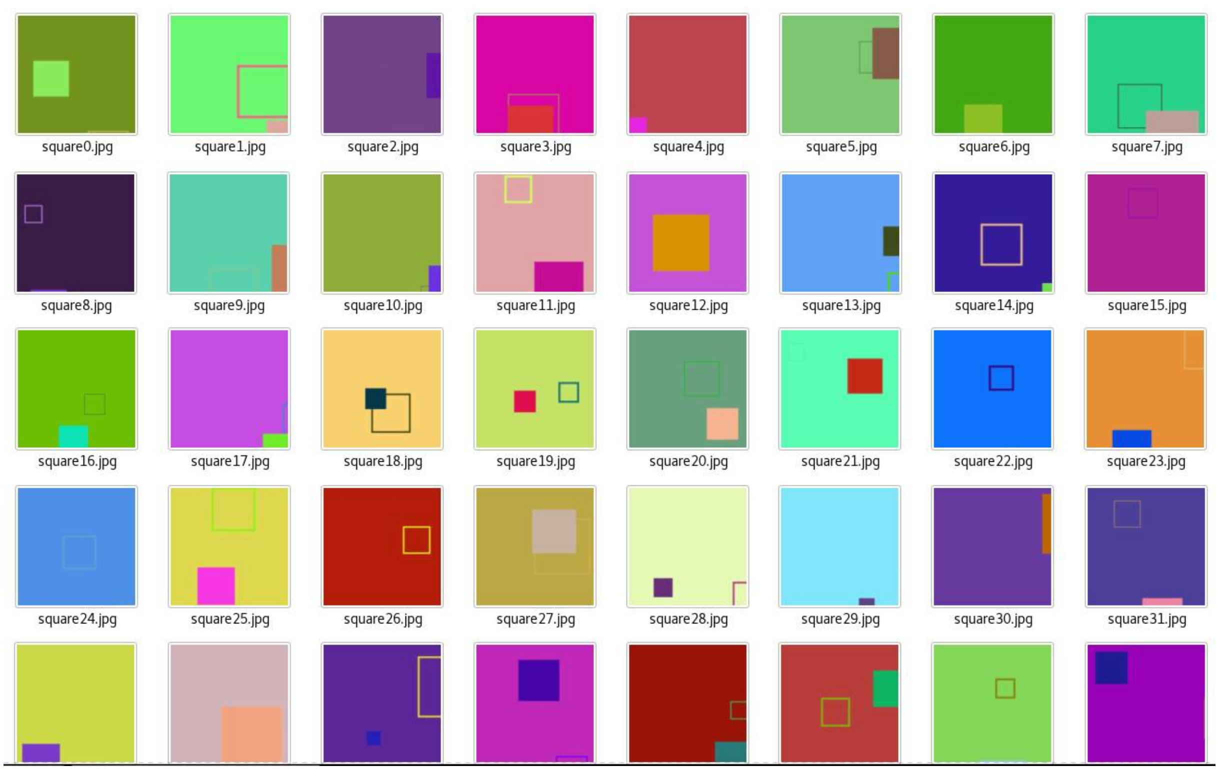
This file reduces the image's quality to reduce the file size:

```
"""
Created on Thu Oct 26 20:50:49 2017

@author: maggie
"""
from __future__ import print_function
from __future__ import division
from PIL import Image
import glob
import pickle
import scipy.misc
import numpy as np
from multiprocessing import Lock
from multiprocessing import Pool

def init(lock):
    global childs_lock
    childs_lock = lock

"""each pool worker gets original img data to reduce file size"""
def reduce_images(image_path):
    childs_lock.acquire()
    img = Image.open(image_path)
    childs_lock.release()
    basewidth = 300
    percent = (basewidth / float(img.size[0]))
    hsize = int((float(img.size[1]) * float(percent)))
    img = img.resize((basewidth, hsize), Image.ANTIALIAS) #ANTIALIAS reserves quality
    x_train = np.array(img)
    #x_train = np.array(img, dtype = np.uint8) #a numpy array with data type CV_8UC1
    #x_train = x_train[ : , : , 0] #slice out the color dimension
    print (x_train.shape)
    img.close()
    return x_train

#global storage variable for both main and pool of workers
result_list = []

"""result(data) is called whenever process_images(path) returns a result
result_list is modified by main process not by pool of workers"""
def result(data):
    result_list.append(data)
```

```python
#create empty pickle_file first then append to file
output = open (pickle_file, 'wb')
output.close()

def result(data):
    output = open (pickle_file, 'ab')
    print ("in pickle file: " , pickle_file)
    pickle.dump(data, output, pickle.HIGHEST_PROTOCOL)
    output.close()

if __name__ == '__main__':

    shape_path = "test_set/circle1/"
    lock = Lock()
    p = Pool(processes=4, initargs = (lock, ), initializer = init)
    #for shapes in shape_path:
    for image_path in glob.glob(shape_path + "*jpg"):
        p.apply_async(process_images, (image_path, shape_path), callback = result)
    p.close() # no more tasks
    p.join() #wrap up current tasks

```

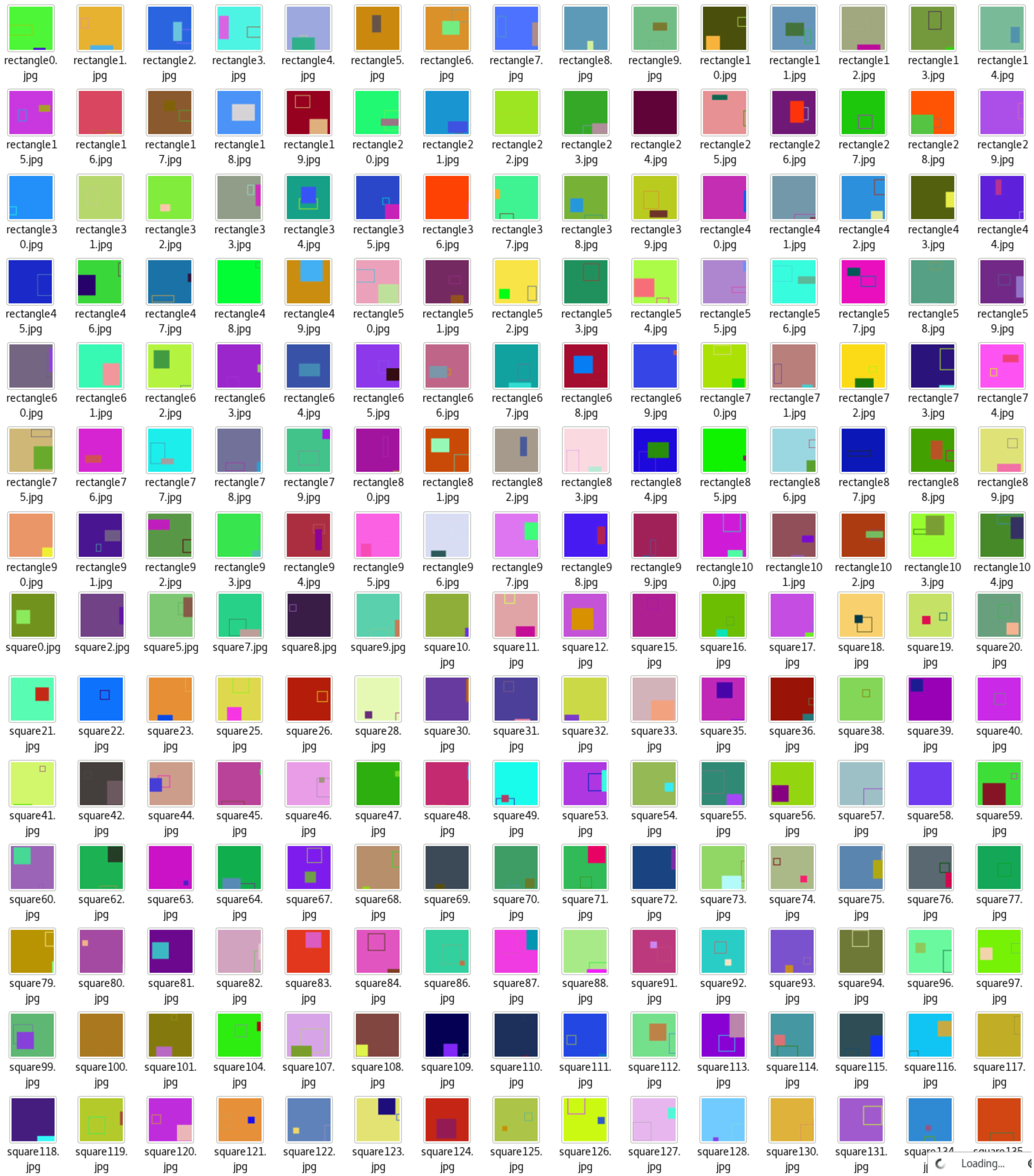






Below are more examples of the training dataset in JPEG extension:


triangle0.jpg
triangle1.jpg
triangle2.jpg
triangle3.jpg
triangle4.jpg
triangle5.jpg
triangle6.jpg
triangle7.jpg
triangle8.jpg
triangle9.jpg
triangle10.jpg
triangle11.jpg
triangle12.jpg
triangle13.jpg
triangle14.jpg
triangle15.jpg
triangle16.jpg
triangle17.jpg
triangle18.jpg
triangle19.jpg
triangle20.jpg
triangle21.jpg
triangle22.jpg
triangle23.jpg
triangle24.jpg
triangle25.jpg
triangle26.jpg
triangle27.jpg
triangle28.jpg
triangle29.jpg
triangle30.jpg
triangle31.jpg
triangle32.jpg
triangle33.jpg
triangle34.jpg
triangle35.jpg
triangle36.jpg
triangle37.jpg
triangle38.jpg
triangle39.jpg
triangle40.jpg
triangle41.jpg
triangle42.jpg
triangle43.jpg
triangle44.jpg
triangle45.jpg
triangle46.jpg
triangle47.jpg
triangle48.jpg
triangle49.jpg
triangle50.jpg
triangle51.jpg
triangle52.jpg
triangle53.jpg
triangle54.jpg
triangle55.jpg
triangle56.jpg
triangle57.jpg
triangle58.jpg
triangle59.jpg
triangle60.jpg
triangle61.jpg
triangle62.jpg
triangle63.jpg
triangle64.jpg
triangle65.jpg
triangle66.jpg
triangle67.jpg
triangle68.jpg
triangle69.jpg
triangle70.jpg
triangle71.jpg
triangle72.jpg
triangle73.jpg
triangle74.jpg
triangle75.jpg
triangle76.jpg
triangle77.jpg
triangle78.jpg
triangle79.jpg
triangle80.jpg
triangle81.jpg
triangle82.jpg
triangle83.jpg
triangle84.jpg
triangle85.jpg
triangle86.jpg
triangle87.jpg
triangle88.jpg
triangle89.jpg
triangle90.jpg
triangle91.jpg
triangle92.jpg
triangle93.jpg
triangle94.jpg
triangle95.jpg
triangle96.jpg
triangle97.jpg
triangle98.jpg
triangle99.jpg
triangle100.jpg
triangle101.jpg
triangle102.jpg
triangle103.jpg
triangle104.jpg
circle0.jpg
circle1.jpg
circle2.jpg
circle3.jpg
circle4.jpg
circle5.jpg
circle6.jpg
circle7.jpg
circle8.jpg
circle9.jpg
circle10.jpg
circle11.jpg
circle12.jpg
circle13.jpg
circle14.jpg
circle15.jpg
circle16.jpg
circle17.jpg
circle18.jpg
circle19.jpg
circle20.jpg
circle21.jpg
circle22.jpg
circle23.jpg
circle24.jpg
circle25.jpg
circle26.jpg
circle27.jpg
circle28.jpg
circle29.jpg
circle30.jpg
circle31.jpg
circle32.jpg
circle33.jpg
circle34.jpg
circle35.jpg
circle36.jpg
circle37.jpg
circle38.jpg
circle39.jpg
circle40.jpg
circle41.jpg
circle42.jpg
circle43.jpg
circle44.jpg
circle45.jpg
circle46.jpg
circle47.jpg
circle48.jpg
circle49.jpg
circle50.jpg
circle51.jpg
circle52.jpg
circle53.jpg
circle54.jpg
circle55.jpg
circle56.jpg
circle57.jpg
circle58.jpg
circle59.jpg
circle60.jpg
circle61.jpg
circle62.jpg
circle63.jpg
circle64.jpg
circle65.jpg
circle66.jpg
circle67.jpg
circle68.jpg
circle69.jpg
circle70.jpg
circle71.jpg
circle72.jpg
circle73.jpg
circle74.jpg
circle75.jpg
circle76.jpg
circle77.jpg
circle78.jpg
circle79.jpg
circle80.jpg
circle81.jpg
circle82.jpg
circle83.jpg
circle84.jpg
circle85.jpg
circle86.jpg
circle87.jpg
circle88.jpg
circle89.jpg
circle90.jpg
circle91.jpg
circle92.jpg
circle93.jpg
circle94.jpg
circle95.jpg
circle96.jpg
circle97.jpg
circle98.jpg
circle99.jpg
circle100.jpg
circle101.jpg
circle102.jpg
circle103.jpg
circle104.jpg
circle105.jpg
circle106.jpg
circle107.jpg
circle108.jpg
circle109.jpg
circle110.jpg
circle111.jpg
circle112.jpg
circle113.jpg
circle114.jpg
circle115.jpg
circle116.jpg
circle117.jpg
circle118.jpg
circle119.jpg



I had problems loading the images to a pickle file because I originally stored the images as a dictionary which represents in a string format. Numpy wants a float object, so I decided to use Python's list data structure to store all the numpy arrays.

```
maggie@debian:~/Downloads/Convolutional_Neural_Networks$ python trainer/cnncopy.py --job-dir ./ --train-file compressedimages.pkl
Using TensorFlow backend.
Using logs_path located at .//logs/2017-10-19T19:34:32.050048
Traceback (most recent call last):
  File "trainer/cnncopy.py", line 145, in <module>
    train_model(**arguments)
  File "trainer/cnncopy.py", line 105, in train_model
    save_format = 'jpeg'):
  File "/home/maggie/anaconda3/lib/python3.6/site-packages/keras/preprocessing/image.py", line 461, in flow
    save_format=save_format)
  File "/home/maggie/anaconda3/lib/python3.6/site-packages/keras/preprocessing/image.py", line 774, in __init__
    self.x = np.asarray(x, dtype=K.floatx())
  File "/home/maggie/anaconda3/lib/python3.6/site-packages/numpy/core/numeric.py", line 531, in asarray
    return array(a, dtype, copy=False, order=order)
ValueError: could not convert string to float: "{'dataset/training_set/cats/cat.175.jpg': <PIL.Image.Image image mode=RGB size=300x226 at 0x7F760CECD860>, 'dataset/training_set
/cats/cat.2192.jpg': <PIL.Image.Image image mode=RGB size=500x474 at 0x7F760CECDFD0>, 'dataset/training_set/cats/cat.2429.jpg': <PIL.Image.Image image mode=RGB size=499x480 at
0x7F760CECDF28>, 'dataset/training_set/cats/cat.2652.jpg': <PIL.Image.Image image mode=RGB size=378x498 at 0x7F760CECDEB8>, 'dataset/training_set/cats/cat.3026.jpg': <PIL.Image
.Image image mode=RGB size=499x375 at 0x7F760CECDE48>, 'dataset/training_set/cats/cat.1317.jpg': <PIL.Image.Image image mode=RGB size=360x449 at 0x7F760CECDDD8>, 'dataset/train
ing_set/cats/cat.224.jpg': <PIL.Image.Image image mode=RGB size=500x374 at 0x7F760CECDD68>, 'dataset/training_set/cats/cat.3110.jpg': <PIL.Image.Image image mode=RGB size=499x3
75 at 0x7F760CECDCF8>, 'dataset/training_set/cats/cat.975.jpg': <PIL.Image.Image image mode=RGB size=384x383 at 0x7F760CECDC88>, 'dataset/training_set/cats/cat.2902.jpg': <PIL.
Image.Image image mode=RGB size=500x374 at 0x7F760CECDC18>, 'dataset/training_set/cats/cat.997.jpg': <PIL.Image.Image image mode=RGB size=500x320 at 0x7F760CECDF60>, 'dataset/t
raining_set/cats/cat.497.jpg': <PIL.Image.Image image mode=RGB size=469x303 at 0x7F760CECDEF0>, 'dataset/training_set/cats/cat.2623.jpg': <PIL.Image.Image image mode=RGB size=4
99x274 at 0x7F760DA2EDA0>, 'dataset/training_set/cats/cat.1246.jpg': <PIL.Image.Image image mode=RGB size=359x270 at 0x7F760DA2ECC0>, 'dataset/training_set/cats/cat.3521.jpg':
<PIL.Image.Image image mode=RGB size=276x225 at 0x7F760CB7CBA8>, 'dataset/training_set/cats/cat.955.jpg': <PIL.Image.Image image mode=RGB size=400x235 at 0x7F760CB7CB38>, 'data
set/training_set/cats/cat.1301.jpg': <PIL.Image.Image image mode=RGB size=499x375 at 0x7F760CB7CAC8>, 'dataset/training_set/cats/cat.171.jpg': <PIL.Image.Image image mode=RGB s
ize=312x280 at 0x7F760CB7CA58>, 'dataset/training_set/cats/cat.2079.jpg': <PIL.Image.Image image mode=RGB size=405x403 at 0x7F760CB7C9E8>, 'dataset/training_set/cats/cat.1736.j
pg': <PIL.Image.Image image mode=RGB size=229x448 at 0x7F760CB7C978>, 'dataset/training_set/cats/cat.2542.jpg': <PIL.Image.Image image mode=RGB size=211x250 at 0x7F760CB7C908>,
 'dataset/training_set/cats/cat.3488.jpg': <PIL.Image.Image image mode=RGB size=500x374 at 0x7F760CB7C898>, 'dataset/training_set/cats/cat.1319.jpg': <PIL.Image.Image image mod
e=RGB size=500x374 at 0x7F760CB7C828>, 'dataset/training_set/cats/cat.1683.jpg': <PIL.Image.Image image mode=RGB size=369x402 at 0x7F760CB7C7B8>, 'dataset/training_set/cats/cat
.2299.jpg': <PIL.Image.Image image mode=RGB size=500x335 at 0x7F760CB7C748>, 'dataset/training_set/cats/cat.3699.jpg': <PIL.Image.Image image mode=RGB size=140x92 at 0x7F760CB7
C6D8>, 'dataset/training_set/cats/cat.1356.jpg': <PIL.Image.Image image mode=RGB size=500x374 at 0x7F760CB7C668>, 'dataset/training_set/cats/cat.1908.jpg': <PIL.Image.Image ima
ge mode=RGB size=240x179 at 0x7F760CB7C5F8>, 'dataset/training_set/cats/cat.1053.jpg': <PIL.Image.Image image mode=RGB size=500x374 at 0x7F760CB7C588>, 'dataset/training_set/ca
ts/cat.537.jpg': <PIL.Image.Image image mode=RGB size=450x367 at 0x7F760CB7C518>, 'dataset/training_set/cats/cat.1397.jpg': <PIL.Image.Image image mode=RGB size=499x375 at 0x7F
760CB7C4A8>, 'dataset/training_set/cats/cat.881.jpg': <PIL.Image.Image image mode=RGB size=500x395 at 0x7F760CB7C438>, 'dataset/training_set/cats/cat.2151.jpg': <PIL.Image.Imag
e image mode=RGB size=349x265 at 0x7F760CB7C3C8>, 'dataset/training_set/cats/cat.464.jpg': <PIL.Image.Image image mode=RGB size=319x240 at 0x7F760CB7C358>, 'dataset/training_se
t/cats/cat.2610.jpg': <PIL.Image.Image image mode=RGB size=374x500 at 0x7F760CB7C2E8>, 'dataset/training_set/cats/cat.1191.jpg': <PIL.Image.Image image mode=RGB size=399x499 at
 0x7F760CB7C278>, 'dataset/training_set/cats/cat.1880.jpg': <PIL.Image.Image image mode=RGB size=126x141 at 0x7F760CB7C208>, 'dataset/training_set/cats/cat.703.jpg': <PIL.Image
.Image image mode=RGB size=407x500 at 0x7F760CB7C198>, 'dataset/training_set/cats/cat.517.jpg': <PIL.Image.Image image mode=RGB size=321x500 at 0x7F760CB7C128>, 'dataset/traini
ng_set/cats/cat.3942.jpg': <PIL.Image.Image image mode=RGB size=349x262 at 0x7F760CB7CDD8>, 'dataset/training_set/cats/cat.563.jpg': <PIL.Image.Image image mode=RGB size=499x47
6 at 0x7F760CB7CE80>, 'dataset/training_set/cats/cat.552.jpg': <PIL.Image.Image image mode=RGB size=375x499 at 0x7F760CB7CF28>, 'dataset/training_set/cats/cat.1239.jpg': <PIL.I
mage.Image image mode=RGB size=500x465 at 0x7F760CB7CF98>, 'dataset/training_set/cats/cat.3774.jpg': <PIL.Image.Image image mode=RGB size=500x374 at 0x7F760CB7CC18>, 'dataset/t
raining_set/cats/cat.2820.jpg': <PIL.Image.Image image mode=RGB size=255x192 at 0x7F760CB7CBE0>, 'dataset/training_set/cats/cat.3613.jpg': <PIL.Image.Image image mode=RGB size=
320x323 at 0x7F760CEF4128>, 'dataset/training_set/cats/cat.958.jpg': <PIL.Image.Image image mode=RGB size=425x201 at 0x7F760CEF4198>, 'dataset/training_set/cats/cat.2712.jpg':
```

10.31.17:

The training set consists of a total of 6,200 images. Before being serialized into a pickle file, the training set is organized in a tuple structure (numpy array, y_label). The numpy array is the data array processed by the PIL module in (300, 300, 3) format. The numpy array represents the matrix in float32 of the image. The y_label represents the target values of the shapes, which is the expected output of the convolutional neural network. Keras requires categorical crossentropy loss to be computed with categorical encodings. The categorical one hot encoding transfers integers (0...number of classes) into binary format. My y_label is a series of categorical hot encodings of 0, 1, 2 in binary format of three classes (circles, rectangles and squares, triangle).

I had to change the numpy array data structure from a default float to float32 bit since the loading of the pickle files in the default float structure consumes too much memory in megabytes per file. The difference almost reduced the entire file size from 3.0 GB (without compression) to 1.7 G.B. The pickle files are too huge, so I have to reduce the quality and size of each image to reduce the pickle files. Pickle loads and image creation of the shapes are created using multiprocessing of independent Pool workers. I have been trying to figure out how to create a pickle file, organize numpy arrays and store them in a huge list, dump that huge list using joblib. Use memmap to store large numpy arrays because it's inefficient for the list to increase in data memory allocation in list comprehension of pickle loading. The file below create (numpy arrays, y_label) tuples and stores them in a pickle file.

The short-term goal is to train the shapes individually first and then figure out how to get the model to generalize on the “intersection” of shapes either by using recurrent convolutional neural networks or multi-label output using supervised learning. How will the network learn? I need to adjust the architecture of the CNN. The multi-label output is simpler and much easier. This requires sigmoid activation and loss = binary_crossentropy at the output layer for multi-label output to work.

draw.py | googlecloud_config_cnn.txt | load_merge_files.py

```python
from __future__ import print_function
from PIL import Image
import glob
import pickle
import numpy as np
from multiprocessing import Lock
from multiprocessing import Pool
from keras.utils import to_categorical

"""to make lock and queue storage global to all child workers
def init(lock):
    global childs_lock
    childs_lock = lock

def process_images(image_path, shape_path):
        shape_y = None
        if shape_path == "test_set/circle1/":
            shape_y = 0
        elif shape_path == "test_set/rectangle1/":
            shape_y = 1
        elif shape_path == "test_set/triangle1/":
            shape_y = 2
     elif shape_path == "test_set/square1/":
            shape_y = 3

        ylabel = to_categorical(shape_y, num_classes = 4)
        ylabel = np.reshape(ylabel, (4))
        print ("new shape", ylabel.shape)
        print (ylabel)
        childs_lock.acquire()
        img = Image.open(image_path)
        childs_lock.release()

        np_img = np.array(img, dtype = [('img_info', np.float16)])
        '''y_label = np.empty(None, dtype = [('y_class',np.int8)])
        y_label.fill(shape_y)'''
        # Add another dimension 1 image number for keras to process
        #np_img = np_img.reshape( (-1, ) + np_img.shape)
        img.close()
        return np_img['img_info'], ylabel

#global storage variable for both main and pool of workers
pickle_file = 'test_circle.pkl'
if __name__ == '__main__':

    circle_path = "test_set/rectangle/"
    lock = Lock()
    p = Pool(processes=4, initargs = (lock, ), initializer = init)

    for image_path in glob.glob(circle_path + "*png"):
        p.apply_async(reduce_images, (image_path,), callback = result)


    p.close() # no more tasks
    p.join() #wrap up current tasks

    output = open ('test_rectangle.pkl', 'wb')
    for x in result_list:
        pickle.dump(x, output, -1)
    output.close()

    name = []
    num_files = 2000
    for i in range(num_files):
        name.append("test_set/rectangle1/rectangle" + str(i) + ".jpg")

    #save resized data to a folder
    with open('test_rectangle.pkl', 'rb') as pkl_file:
        data1 = [pickle.load(pkl_file) for i in range(num_files)]
    for i in range(num_files):
        scipy.misc.imsave(name[i], data1[i])
```

This file merges all the pickled files that each represents the individual shape data and their y_labels from training, validation and testing set.

draw.py | googlecloud_config_cnn.txt | load_merge_files.py | merge_files.py

```python
from __future__ import print_function
import pickle
import joblib
#import numpy as np
#from tempfile import mkdtemp
#import os.path as path

def load_train_or_test(files):
    with open(files, 'rb') as f:
        try:
            print ("Opening files")
            print (files)
            while True:
                yield pickle.load(f) #python version 2.7
        except EOFError:
            pass

if __name__ == '__main__':

    #include (shape array,y_labels) as a tuple returned by the pickle
    circle_dataset = [item for item in load_train_or_test ("circle.pkl")]
    triangle_dataset = [item for item in load_train_or_test ("triangle.pkl")]
    rectangle_dataset = [item for item in load_train_or_test ("rectangle.pkl")]
    square_dataset = [item for item in load_train_or_test ("square.pkl")]

    #merge the individual shape data into one train_data
    train_data = circle_dataset + triangle_dataset + rectangle_dataset + square_dataset

    validation_circle_dataset = [item for item in load_train_or_test ("validate_circle.pkl")]
    validation_triangle_dataset = [item for item in load_train_or_test ("validate_triangle.pkl")]
    validation_rectangle_dataset = [item for item in load_train_or_test ("validate_rectangle.pkl")]
    validation_square_dataset = [item for item in load_train_or_test ("validate_square.pkl")]

    validation_data = validation_circle_dataset + validation_triangle_dataset + validation_rectangle_dataset + validation_square_dataset

    pickle_file = 'shape_data.pkl'
    try:
        f = open(pickle_file, 'wb')
        save = {#'train_shape_dataset': train_shape_dataset,
                'train_data': train_data,
                'validation_data': validation_data,
                }
        #pickle.dump(save, f, pickle.HIGHEST_PROTOCOL)
        joblib.dump(save, f, compress = True)
        f.close()
    except Exception as e:
        print('Unable to save data to', pickle_file, ':', e)
        raise
```

This file uses memory mapping to store large numpy arrays, and randomize the data arrays. It then stores all the data in a compressed pickle file for Google Cloud to load. Google Cloud uses python 2, so the CNN loader file will also use python 2.

draw.py | googlecloud_config_cnn.txt | load_merge_files.py

```python
from __future__ import print_function
import joblib
import pickle
import numpy as np
from tempfile import mkdtemp
import os.path as path

'''returns individual list data info and y label data in numpy arrays'''
def get_data(shape_temp_file, label_temp_file, dataset):
    #use memory mapping to store large datasets
    temp_filename = path.join(mkdtemp(), shape_temp_file)
    train_shape_dataset = np.memmap(temp_filename, dtype = np.float16, mode = 'w+', shape = (300, 300, 3))
    temp_filename1 = path.join(mkdtemp(), label_temp_file)
    train_y_dataset = np.memmap(temp_filename1, dtype = np.float16, mode = 'w+', shape = (3))

    train_shape_dataset = [x[0] for x in dataset]
    #convert list back to np array for keras to process
    train_shape_dataset = np.array(train_shape_dataset)
    print ("in get_data function for dataset")
    print (train_shape_dataset.shape)
    train_y_dataset = [x[1] for x in dataset]
    train_y_dataset = np.array(train_y_dataset)
    print (train_y_dataset.shape)

    return train_shape_dataset, train_y_dataset

if __name__ == '__main__':

    pickle_file = 'shape_data.pkl'
    np.random.seed(135)
    with open(pickle_file, 'rb') as f:
        #save = pickle.load(f)
        save = joblib.load(f)
        train_data = save['train_data']
        validation_data = save['validation_data']
        del save  # hint to help gc free up memory

    #shuffle the tuple (shape_info, y_label) dataset
    np.random.seed(135)
    np.random.shuffle(train_data)

    #split list in half
    train_data_half = train_data[ :: 3]
    #train_data_other_half = train_data[1 :: 2]
    validation_data_half = validation_data[ :: 3]

    train_shape_dataset, train_y_dataset = get_data('shapes.dat', 'shapes_y.dat', train_data_half)
    #train_shape_halfdataset, train_y_halfdataset = get_data('shapes.dat', 'shapes_y.dat', train_data_other_half)
    validate_shape_dataset, validate_y_dataset = get_data('validate_shapes.dat', 'validate_shapes_y.dat', validation_da

    print ("in main: 1/half train", train_shape_dataset.shape)
    print ("in main: 1/half y_label", train_y_dataset.shape)
    #print ("in main: 2/half train", train_shape_halfdataset.shape)
    #print ("in main: 2/half y_label", train_y_halfdataset.shape)
    print ("in main: validate", validate_shape_dataset.shape)
    print ("in main: validate y_label", validate_y_dataset.shape)

    pickle_file = 'random_shapes.pkl'
    try:
        f = open(pickle_file, 'wb')
        save = {'train_shape_dataset': train_shape_dataset,
                'train_y_dataset': train_y_dataset,
                #'train_shape_halfdataset': train_shape_halfdataset,
                #'train_y_halfdataset': train_y_halfdataset,
                'validate_shape_dataset': validate_shape_dataset,
                'validate_y_dataset': validate_y_dataset,
                }
        #pickle.dump(save, f, pickle.HIGHEST_PROTOCOL)
        joblib.dump(save, f, compress = True)
        f.close()
    except Exception as e:
        print('Unable to save data to', pickle_file, ':', e)
        raise
```

11.3-11.5.17:

Google cloud works locally but had errors of loading pickle file remotely on google cloud because the Cloud Compute Engine doesn't recognize python's file descriptor. I need to use tensorflow's open method, need to set gs:// for every input file data for Google Cloud to recognized it. Here are the steps to run the CNN loader file in Google Cloud:

draw.py   googlecloud_config_cnn.txt

```
gsutil cp -r trainer/cloudml-gpu.yaml gs://cnninput_dataset/trainer/cloudml-gpu.yaml
gsutil cp -r trainer/__init__.py gs://cnninput_dataset/trainer/__init__.py

data folder
gsutil cp -r data/random_shapes.pkl gs://cnninput_dataset/data/random_shapes.pkl

bucket folder
gsutil cp -r setup.py gs://cnninput_dataset/setup.py


export BUCKET_NAME=cnninput_dataset
export JOB_NAME="cnncopy_train_$(date +%Y%m%d_%H%M%S)"
export JOB_DIR=gs://$BUCKET_NAME/$JOB_NAME
export REGION=us-east1

train on machine locally
gcloud ml-engine local train \
  --job-dir $JOB_DIR \
  --module-name trainer.cnncopy \
  --package-path ./trainer \
  -- \
  --train-file ./data/random_shapes.pkl

submit a job to cloud ML engine
gcloud ml-engine jobs submit training $JOB_NAME \
    --job-dir $JOB_DIR \
    --runtime-version 1.0 \
    --module-name trainer.cnncopy \
    --package-path ./trainer \
    --region $REGION \
    --config trainer/cloudml-gpu.yaml \
    -- \
    --train-file gs://$BUCKET_NAME/data/random_shapes.pkl

submit a job to cloud ML engine
gcloud ml-engine jobs submit training $JOB_NAME \
    --job-dir $JOB_DIR \
    --runtime-version 1.0 \
    --module-name trainer.cnncopy \
    --package-path ./trainer \
    --region $REGION \
    -- \
    --train-file gs://$BUCKET_NAME/data/random_shapes.pkl
```

11.6.17:

There is an memory error when running on Google Cloud's regular CPU after one set of 10 epochs for the first half of the dataset. There is not enough memory allocated and training took 1 hour, which is too slow. I decided to use yaml configuration to run on a single NVIDIA K80 GPU processor on Google Cloud Compute Engine.

11.7.17:

I executed this with no errors in Google Cloud with GPU computing on a validation set 1000 images and training set of 6000 images with roughly 60 percent accuracy, 3 percent error rate in 3 series of 10 epochs per training set each. The learning model is able to be saved. Google Cloud automatically plots the gradient on Tensorboard. The reason the error rate is so high and accuracy is low is because there are alot of background samples that the CNN intakes as pool sizes. Background colored samples are data that contains no linear information - unimportant numpy array figures. so when the network does the maxpool of background samples near the 'important line samples', if the background samples are in greater distributation than the amount of important line samples, maxpool will label that area as background sample which makes the neurons increase the weights for backgrounds instead of the contour images itself.

11.8.17:

I increased the y-label output from 3 classes to 4 classes. Keras does the automatic shuffle at every epoch in fit_generator. I changed the architecture of the CNN, add drop out layers that might drop out neurons that have no data of contour characteristics being drawn or do some cropping of batches that do not consist of contour information beforehand. I increased the pool size of the CNN and changed it from adam optimizer to rms optimizer. The CNN will do fit the generator model from data augmentation in 20 epochs with validation and training inputs inputted. I also implemented the validation set correctly during the fitting of the network with real data augmentation. The CNN does poorly during training, with an accuracy of 59 percent and 6 percent loss. This is because I used 3,000 images to train the dataset, which is 1/3 of the total training set, which might not contain evenly distributed images of each type of shape. I reduced the total training set by a third because I want to focus on getting the architecture of the CNN right and there is memory error at the Tesla K80 GPU from the loading of the images since the validation data increased by twice as much as the previous one.

11.9.17:

Trying to figure out how to redesign the architecture of my CNN by looking back on the research I did in Neural Network Design. I also need to create my own data generator (augmentation) function that crops large scaled images to reduce unnecessary background sampling of images in Pooling. I don't want to separate the contours and filling of the images from the background because the background plays an important part in the composition of the entire image object. Such images that need to be cropped, where the dotted lines represent the cropping location, in a generator function are: