

When I ran the same code in google cloud, the machine only computed the 32 epochs for the first half of the dataset in the google cloud's regular machines, the machine returns an exit status of -9, which means that the program exited because there isn't enough memory in the machine to execute the program. I need to use a single GPU to run a simple convolutional neural network. I had to request for a quota of 2 Tesla K80 GPUs in the east-1 area from google cloud. The configuration file and output from google cloud to run using nvidia's GPU in the cloud is below. The above images represent a successful output running using GPU processing. The standard-gpu configuration only uses 1 GPU, the complex model uses 4 GPUs. There is no custom model for 2 GPUs. I will need to use google cloud's VM instance to create a machine running in 2 GPUS, install the necessary keras, tensorflow, nvidia dependencies on the machine if I decide to use 2 GPUS. But, for now, it is important to focus on getting a working architecture for the network. So, I'll use 1 GPU unless absolutely necessary for the network to develop a good generalization model. use a VM instance in the cloud. Below describes the commands to run the python code in google cloud. It requires an empty `__init__.py` file, `setup.py` file and remote linux commands.

CREATE METRIC

CREATE EXPORT

Filter by label or text search

Cloud ML Job, cnncopy\_train\_20171107\_16...All logsAny log levelJump to date

2017-11-07 EST

No older entries found matching current filter.

googlecloud\_config\_cnn.txt

load\_merge\_files.py

mer:

```
1 '''Cloud ML Engine package configuration.'''
2 from setuptools import setup, find_packages
3
4 setup(name='cnncopy',
5       version='1.0',
6       packages=find_packages(),
7       include_package_data=True,
8       description='MNIST MLP keras model on Cloud ML Engine',
9       author='Maggie Cao',
10      author_email='mahgieeee@hotmail.com',
11      license='MIT',
12      install_requires=[
13          'keras',
14          'h5py',
15          'pillow',
16          'joblib'],
17      zip_safe=False)
```

16:14:42.476

service

Validating job requirements...

16:14:42.759

service

Job creation request has been successfully validated.

16:14:42.985

service

Job cnncopy\_train\_20171107\_161426 is queued.

16:14:43.216

Waiting

for job to be provisioned.

16:19:34.788

Waiting

for TensorFlow to start.

16:22:11.452

master-replica-0

Running task with arguments: --cluster={"master": ["master-828a77db4d-0:2222"]}

16:22:11.694

master-replica-0

Running module trainer.cnncopy.

16:22:11.694

master-replica-0

Downloading the package: gs://cnninput\_dataset/cnncopy\_train\_20171107\_161426/pack

16:22:11.695

master-replica-0

Running command: gsutil -q cp gs://cnninput\_dataset/cnncopy\_train\_20171107\_161426/pack

16:22:12.816

master-replica-0

Installing the package: gs://cnninput\_dataset/cnncopy\_train\_20171107\_161426/pack

16:22:12.817

master-replica-0

Running command: pip install --user --upgrade --force-reinstall --no-deps cnncopy

16:22:13.173

master-replica-0

Processing ./cnncopy-1.0.tar.gz

16:22:13.433

master-replica-0

Building wheels for collected packages: cnncopy

16:22:13.434

master-replica-0

Running setup.py bdist\_wheel for cnncopy: started

16:22:13.668

master-replica-0

creating '/tmp/tmpzp2VZZip-wheel-/cnncopy-1.0-cp27-none-any.whl' and adding '.'

16:22:13.669

master-replica-0

adding 'trainer/cnncopy.py'

16:22:13.669

master-replica-0

adding 'trainer/\_\_init\_\_.py'

16:22:13.670

master-replica-0

adding 'cnncopy-1.0.dist-info/DESCRIPTION.rst'

16:22:13.670

master-replica-0

adding 'cnncopy-1.0.dist-info/metadata.json'

16:22:13.670

master-replica-0

adding 'cnncopy-1.0.dist-info/top\_level.txt'

16:22:13.670

master-replica-0

adding 'cnncopy-1.0.dist-info/WHEEL'

16:22:13.671

master-replica-0

adding 'cnncopy-1.0.dist-info/METADATA'

16:22:13.671

master-replica-0

adding 'cnncopy-1.0.dist-info/RECORD'

16:22:13.684

master-replica-0

Running setup.py bdist\_wheel for cnncopy: finished with status 'done'

16:22:13.684

master-replica-0

Stored in directory: /root/.cache/pip/wheels/d3/55/7f/ecafa9b507690ec7c3cd02505e32921947b08002ca0b6fd212

16:22:13.687

master-replica-0

Successfully built cnncopy

16:22:13.687

master-replica-0

Installing collected packages: cnncopy

16:22:13.742

master-replica-0

Successfully installed cnnconv-1.0

16:22:21.993

master-replica-0

successfully opened CUDA library libcudand.so.8.0 locally

16:22:22.620

master-replica-0

The TensorFlow library wasn't compiled to use AVX2 instructions, but these are available on your machine and could speed up CPU computations.

16:22:22.620

master-replica-0

The TensorFlow library wasn't compiled to use FMA instructions, but these are available on your machine and could speed up CPU computations.

16:22:22.787

master-replica-0

successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero

16:22:22.788

master-replica-0

Found device 0 with properties:

16:22:22.788

master-replica-0

name: Tesla K80

16:22:22.788

master-replica-0

major: 3 minor: 7 memoryClockRate (GHz) 0.8235

16:22:22.788

master-replica-0

pciBusID 0000:00:04:0

16:22:22.789

master-replica-0

Total memory: 11.17GiB

16:22:22.789

master-replica-0

Free memory: 11.11GiB

16:22:22.789

master-replica-0

DMA: 0

16:22:22.789

master-replica-0

0: Y

16:22:22.789

master-replica-0

Creating TensorFlow device (/gpu:0) -> (device: 0, name: Tesla K80, pci bus id: 0000:00:04:0)

16:22:46.764

master-replica-0

Creating TensorFlow device (/gpu:0) -> (device: 0, name: Tesla K80, pci bus id: 0000:00:04:0)

16:22:48.716

master-replica-0

Using logs\_path located at gs://cnninput\_dataset/cnncopy\_train\_20171107\_161426/logs/2017-11-07T21:22:23.350608

16:22:48.717

master-replica-0

Epoch 1/10

16:22:48.959

master-replica-0

1/90 [.....] - ETA: 3:36 - loss: 1.1456 - acc: 0.2188

16:22:49.195

master-replica-0

2/90 [.....] - ETA: 1:57 - loss: 6.7781 - acc: 0.2188

16:22:49.429

master-replica-0

3/90 [.....] - ETA: 1:24 - loss: 7.1836 - acc: 0.2917

16:22:49.664

master-replica-0

4/90 [>.....] - ETA: 1:07 - loss: 7.5238 - acc: 0.3359

16:22:49.900

master-replica-0

5/90 [>.....] - ETA: 57s - loss: 7.8324 - acc: 0.3563

16:22:50.137

master-replica-0

6/90 [>.....] - ETA: 50s - loss: 8.5417 - acc: 0.3385

16:22:50.512

master-replica-0

7/90 [=>.....] - ETA: 45s - loss: 8.4008 - acc: 0.3661

16:22:50.999

master-replica-0

8/90 [=>.....] - ETA: 43s - loss: 8.6099 - acc: 0.3672

16:22:51.481

master-replica-0

9/90 [=>.....] - ETA: 42s - loss: 8.8845 - acc: 0.3611

16:22:51.974

master-replica-0

10/90 [=>.....] - ETA: 41s - loss: 9.2049 - acc: 0.3500

16:22:52.455

master-replica-0

11/90 [=>.....] - ETA: 41s - loss: 9.0470 - acc: 0.3665

This network is compiled using CUDA's K80 GPU. I learned that the output of the losses are wrong for the first successful compilation of the convolutional neural network in the google cloud. According to <http://cs231n.github.io/>, the starting loss of the CNN is  $-\ln(1/\text{num\_of\_classes})$ . For 3 classes,  $-\ln(0.33) = 1.1086626245$ , which should be the starting loss of the network. However, my starting loss ranges from 6-8%.

The output of the program is below, the compilation error is because of the syntax error in model.save so I wasn't able to save the model in the cloud directory. The test loss (3.63) is fewer than the next keras output as well as a higher accuracy (61%) than the next output, because the next version of the CNN file is being trained on 3033 images in total (1/3 of the total training set due to memory error since I increased the number of output classes from 3 to 4). The validation data was organized wrongly in this network because it was just fit into the network. Training should consist of a validation set and training set. This version of the CNN is trained using stochastic gradient training on a total of 6200 images, with half

16:22:52.946	master-replica-0	12/90	[====>.....]	ETA: 40s	loss: 9.0906	acc: 0.3698
16:22:53.441	master-replica-0	13/90	[====>.....]	ETA: 39s	loss: 9.1079	acc: 0.3726
16:22:53.934	master-replica-0	14/90	[====>.....]	ETA: 39s	loss: 8.8979	acc: 0.3884
16:22:54.423	master-replica-0	15/90	[====>.....]	ETA: 38s	loss: 8.7307	acc: 0.4000
16:22:54.905	master-replica-0	16/90	[====>.....]	ETA: 37s	loss: 8.5082	acc: 0.4160
16:22:55.391	master-replica-0	17/90	[====>.....]	ETA: 37s	loss: 8.5411	acc: 0.4136
16:22:55.887	master-replica-0	18/90	[====>.....]	ETA: 36s	loss: 8.3572	acc: 0.4149
16:22:56.387	master-replica-0	19/90	[====>.....]	ETA: 36s	loss: 8.0705	acc: 0.4194
16:22:56.874	master-replica-0	20/90	[====>.....]	ETA: 35s	loss: 7.7490	acc: 0.4141
16:22:57.371	master-replica-0	21/90	[====>.....]	ETA: 35s	loss: 7.4609	acc: 0.4077
16:22:57.868	master-replica-0	22/90	[====>.....]	ETA: 34s	loss: 7.1898	acc: 0.4034
16:22:58.349	master-replica-0	23/90	[====>.....]	ETA: 34s	loss: 6.9293	acc: 0.4022
16:22:58.824	master-replica-0	24/90	[====>.....]	ETA: 33s	loss: 6.6899	acc: 0.3997
16:22:59.318	master-replica-0	25/90	[====>.....]	ETA: 32s	loss: 6.4653	acc: 0.4012
16:22:59.802	master-replica-0	26/90	[====>.....]	ETA: 32s	loss: 6.2582	acc: 0.4026
16:23:00.281	master-replica-0	27/90	[====>.....]	ETA: 31s	loss: 6.0662	acc: 0.4028
16:23:00.775	master-replica-0	28/90	[====>.....]	ETA: 31s	loss: 5.8884	acc: 0.4051
16:23:01.258	master-replica-0	29/90	[====>.....]	ETA: 30s	loss: 5.7216	acc: 0.4052
16:23:01.814	master-replica-0	30/90	[====>.....]	ETA: 30s	loss: 5.5683	acc: 0.4021
16:23:02.054	master-replica-0	31/90	[====>.....]	ETA: 29s	loss: 5.4356	acc: 0.4020
16:23:02.537	master-replica-0	32/90	[====>.....]	ETA: 28s	loss: 5.3016	acc: 0.4031
16:23:03.021	master-replica-0	33/90	[====>.....]	ETA: 28s	loss: 5.1739	acc: 0.4023
16:23:03.514	master-replica-0	34/90	[====>.....]	ETA: 27s	loss: 5.0556	acc: 0.3987
16:23:03.997	master-replica-0	35/90	[====>.....]	ETA: 27s	loss: 4.9434	acc: 0.3945
16:23:04.482	master-replica-0	36/90	[====>.....]	ETA: 26s	loss: 4.8363	acc: 0.3974
16:23:04.966	master-replica-0	37/90	[====>.....]	ETA: 26s	loss: 4.7379	acc: 0.3959
16:23:05.447	master-replica-0	38/90	[====>.....]	ETA: 25s	loss: 4.6428	acc: 0.3929
16:23:05.931	master-replica-0	39/90	[====>.....]	ETA: 25s	loss: 4.5538	acc: 0.3925
16:23:06.423	master-replica-0	40/90	[====>.....]	ETA: 24s	loss: 4.4677	acc: 0.3897

Cloud ML Job, cnncopy\_train\_20171107\_16... All logs Any log level Jump to date

2017-11-07 EST		View Options
16:40:31.456	master-replica-0	29/37 [====>.....] - ETA: 3s - loss: 0.8362 - acc: 0.6218
16:40:31.920	master-replica-0	30/37 [====>.....] - ETA: 3s - loss: 0.8344 - acc: 0.6198
16:40:32.375	master-replica-0	31/37 [====>.....] - ETA: 3s - loss: 0.8322 - acc: 0.6210
16:40:32.829	master-replica-0	32/37 [====>.....] - ETA: 2s - loss: 0.8322 - acc: 0.6182
16:40:33.293	master-replica-0	33/37 [====>.....] - ETA: 2s - loss: 0.8327 - acc: 0.6184
16:40:33.757	master-replica-0	34/37 [====>.....] - ETA: 1s - loss: 0.8312 - acc: 0.6195
16:40:34.233	master-replica-0	35/37 [====>.....] - ETA: 1s - loss: 0.8309 - acc: 0.6196
16:40:34.707	master-replica-0	36/37 [====>.....] - ETA: 0s - loss: 0.8332 - acc: 0.6181
16:40:34.856	master-replica-0	37/37 [====>.....] - ETA: 0s - loss: 0.8354 - acc: 0.6174
16:40:34.856	master-replica-0	38/37 [====>.....] - 17s 460ms/step - loss: 0.8323 - acc: 0.6208
16:40:45.688	master-replica-0	Traceback (most recent call last): File "/usr/lib/python2.7/runpy.py", line 162, in _run_module_as_main "__main__", fname, loader, pkg_...
16:40:46.117	master-replica-0	Test loss: 3.633359828
16:40:46.117	master-replica-0	Test accuracy 0.61
16:40:46.249	master-replica-0	Command '['python', '-m', 'u'trainer.cnncopy', 'u'--train-file', 'u'gs://cnncopy_dataset/data/random_shapes.pkl', '--job-dir', 'u'gs://cnncopy_...]
16:40:46.250	master-replica-0	Module completed; cleaning up.
16:40:46.250	master-replica-0	Clean up finished.
16:41:12.919	master-replica-0	The replica master 0 exited with a non-zero status of 1. Termination reason: Error. Traceback (most recent call last): File "/usr/lib/python2.7/runpy.py", line 162, in _run_module_as_main "__main__", fname, loader, pkg_...
16:42:24.882	master-replica-0	Finished tearing down TensorFlow.
16:43:09.789	master-replica-0	Job failed.

the dataset in one generator and the other half in another generator, fitting into the CNN using real-data augmentation. The validation data, consisting of roughly 1800 images, isn't incorporated with the 2 training generators. The validation data is just incorporated separately as an additional generator. This just means that Keras will treat it as an additional input batch of data. Below is my CNN file in Keras of the above output:

```

8 def train_model(loader = 'random_shapes.pkl', job_dir = './', **args):
9     from keras.models import Sequential
10    from keras.layers import Conv2D
11    from keras.layers import MaxPooling2D
12    from keras.layers import Flatten
13    from keras.layers import Dense
14    from keras.preprocessing.image import ImageDataGenerator
15    from datetime import datetime # for filename conventions
16    from tensorflow.python.lib.io import file_io # for better file I/O
17    #import h5py # for saving the model
18    import joblib
19
20    #set the logging path for ML Engine logging to storage bucket
21    logs_path = job_dir + '/logs/' + datetime.now().isoformat()
22    print('Using logs_path located at {}'.format(logs_path))
23
24    with open(loader, 'rb') as f:
25        save = joblib.load(f)
26        train_shape_dataset = save['train_shape_dataset']
27        train_y_dataset = save['train_y_dataset']
28        train_shape_halfdataset = save['train_shape_halfdataset']
29        train_y_halfdataset = save['train_y_halfdataset']
30        #test_shape_dataset = save['test_shape_dataset']
31        #test_y_dataset = save['test_y_dataset']
32        #rectangle_ylabel = save['rectangle_ylabel']
33        del save # hint to help gc free up memory
34
35    # Initialising the CNN, adding a layer
36    classifier = Sequential()
37
38    # Step 1 - Convolution
39    classifier.add(Conv2D(32, (3, 3), input_shape = (300, 300, 3), activation='relu'))
40
41    # Step 2 - Pooling
42    classifier.add(MaxPooling2D(pool_size = (2, 2)))
43
44    # Adding a second convolutional layer
45    classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
46
47    classifier.add(MaxPooling2D(pool_size = (2, 2)))
48
49    # Step 3 - Flattening
50    classifier.add(Flatten())
51
52    #Dense function is used to add a fully connected layer at the end
53    # Step 4 - Full connection
54    classifier.add(Dense(units = 128, activation = 'relu'))
55    classifier.add(Dense(units = 3, activation = 'softmax'))
56
57    # Compiling the CNN #change 'binary_crossentropy to categorical'
58    classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
59
60    # Part 2 - Fitting the CNN to the images
61
62    #augmentation configuration to prevent overfitting
63    datagen = ImageDataGenerator(rescale = 1./255,
64                                shear_range = 0.2,
65                                zoom_range = 0.2,
66                                horizontal_flip = True)
67
68    #augmentation configuration for rescaling test images
69    test_datagen = ImageDataGenerator(rescale = 1./255)
70
71    #circle_train = np.array(circle_train)
72    #circle_ylabel = np.array(circle_ylabel)
73
74    datagen.fit(train_shape_dataset)
75
76    batches = 0
77    #flow() creates batches of randomly transformed images
78    for x_batch, y_batch in datagen.flow(train_shape_dataset,
79                                         train_y_dataset,
80                                         batch_size = 32,
81                                         save_to_dir = "shapes/train/",
82                                         save_prefix = "shapes",
83                                         save_format = "jpeg"):
84        #datagen.fit(x_batch) #or (x_batch, y_batch for output of classes)
85        classifier.fit(x_batch, y_batch)
86        batches += 1
87        if batches >= len(train_shape_dataset) / 32:
88            break #without break, generator will loop indefinitely
89
90    datagen.fit(train_shape_halfdataset)
91
92    batches1 = 0
93    #flow() creates batches of randomly transformed images
94    for x_batch1, y_batch1 in datagen.flow(train_shape_halfdataset,
95                                         train_y_halfdataset,
96                                         batch_size = 32,
97                                         save_to_dir = "shapes/train/",
98                                         save_prefix = "half_shapes",
99                                         save_format = "jpeg"):
100        #datagen.fit(x_batch) #or (x_batch, y_batch for output of classes)
101        classifier.fit(x_batch1, y_batch1)
102        batches1 += 1
103        if batches1 >= len(train_shape_halfdataset) / 32:
104            break #without break, generator will loop indefinitely
105
106    datagen.fit(rectangle_train)
107
108    batches2 = 0
109    #flow() creates batches of randomly transformed images
110    for x_batch2, y_batch2 in datagen.flow(rectangle_train,
111                                         rectangle_ylabel,
112                                         batch_size = 32,
113                                         save_to_dir = "shapes/",
114                                         save_prefix = "rectangles",
115                                         save_format = "jpeg"):
116        #datagen.fit(x_batch) #or (x_batch, y_batch for output of classes)
117        classifier.fit(x_batch2, y_batch2)
118        batches2 += 1
119        if batches2 >= len(rectangle_train) / 32:
120            break #without break, generator will loop indefinitely''
121
122    #classifier.summary() ?
123
124    # Save the model locally
125    classifier.save('model.h5')
126
127    # Save the model to the Cloud Storage bucket's jobs directory
128    with file_io.FileIO('classifiermodel.h5', mode='r') as input_f:
129        with file_io.FileIO(job_dir + '/model.h5', mode='w+') as output_f:
130            output_f.write(input_f.read())
131

```



I don't know how the memory error occurred during pickle loading in the next version of the CNN. The data input that failed on memory error consists of 2100 circles, 2450 squares, 2100 triangles and 2450 squares for the training set, which is a total of 9100 images. The validation set contains 900 circles, 1050 rectangles, 1050 squares and 900 triangles, which is a total of 3900 images. The loading of the validation set is probably where the memory error occurred because joblib won't be able to load a large numpy array of images greater than approximately 3500 images. In addition, the loss is greater and accuracy is less because this version uses rmsprop, where the gradient is computed on a batch of data instead of for every input data. Rmsprop optimizer divides the gradient by a running average of its recent magnitude. According to fchollet, the designer of Keras, it is recommended to leave the parameters of the optimizer at their default values, except the learning rate which could be freely tuned. Fchollet also says that this optimizer is usually a good choice for recurrent neural networks, but I'm using a CNN network so I will go back to either Adam or SGD as optimizer. The test dataset consists of 2000 images for each shape. Here is the modification of the first Keras code:

```

18 def train_model(train_file = 'gs://cnninput_dataset/data/random_shapes.pkl',
19                 job_dir = './',
20                 **args):
21     # set the logging path for ML Engine logging to storage bucket
22     logs_path = job_dir + '/logs/' + datetime.now().isoformat()
23     print('Using logs_path located at {}'.format(logs_path))
24
25     # need tensorflow to open file descriptor in order for google cloud to
26     # process it (instead of 'with open(loader, 'rb' as f:'))
27     with file_io.FileIO(train_file, mode='r') as f:
28         # joblib loads compressed files consisting of large datasets
29         # efficiently.
30         save = joblib.load(f)
31         train_shape_dataset = save['train_shape_dataset']
32         train_y_dataset = save['train_y_dataset']
33         #train_shape_halfdataset = save['train_shape_halfdataset']
34         #train_y_halfdataset = save['train_y_halfdataset']
35         validate_shape_dataset = save['validate_shape_dataset']
36         validate_y_dataset = save['validate_y_dataset']
37         del save # hint to help gc free up memory
38
39     # Initialising the CNN by adding a simple sequential layer
40     classifier = Sequential()
41
42     # Step 1:
43     # Sequential layer consists of Convolution of type 3 by 3 convolutional
44     # window with 32 output filters(dimensionality of output space) for each
45     # input image uses ReLU layers, which is a
46     # 'a nonlinear layer, network with relu is trained faster without creating
47     # a decrease in accuracy @ adeshpande3.github.io'
48     classifier.add(Conv2D(32, (3, 3), input_shape = (300, 300, 3), activation = 'relu'))
49
50     # Step 2:
51     # Max Pooling downsamples the number pixels per neuron and create a max
52     # number that describes those features in a pool_size of 2 by 2
53     # change pool size from (2,2) to (8,8) to (4,4)
54     classifier.add(MaxPooling2D(pool_size = (4, 4)))
55
56     # Adding a second convolutional layer, which is the same as the first one
57     classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
58     # change pool size from (2,2) to (8,8)
59     classifier.add(MaxPooling2D(pool_size = (4, 4)))
60     # Dropout layers at the second convolutional layer before flattening
61     classifier.add(Dropout(0.25))
62
63     # Step 3: Flattening the convolutional layers for input into a fully
64     # connected layer
65     classifier.add(Flatten())
66
67     # Step 4:
68     # Fully connected: Dense function is used to add a fully connected
69     # 3 layer perceptron at the end
70     classifier.add(Dense(units = 128, activation = 'relu'))
71     # dropout at the first layer perceptron
72     classifier.add(Dropout(0.25))
73     # adding second hidden layer - remove if accuracy decreases or loss increases
74     classifier.add(Dense(units = 128, activation = 'relu'))
75     classifier.add(Dropout(0.35))
76     # softmax classifier as an activation from the last layer perceptron
77     # units represent number of output classes
78     # the output classes are triangle, rectangle, square, circle
79     classifier.add(Dense(units = 4, activation = 'softmax'))
80
81     # Compiling the CNN:
82     # check if optimizer adam is good, categorical_crossentropy is for
83     # multi-class network, multilabel with intersection needs binary_crossentropy
84     # and sigmoid activations
85     # change from adam to rmsprop
86     classifier.compile(optimizer = 'adam',
87                       loss = 'categorical_crossentropy',
88                       metrics = ['accuracy'])
89
90     # Part 2:
91     # Feeding CNN the input images and fitting the CNN
92     # CNN uses data augmentation configuration to prevent overfitting
93     # datagen augmentation is for training data input
94     datagen = ImageDataGenerator(rescale = 1./255,
95                                 shear_range = 0.2,
96                                 zoom_range = 0.2,
97                                 horizontal_flip = True)
98
99     # augmentation configuration for rescaling images used for validation
100    validate_datagen = ImageDataGenerator(rescale = 1./255)
101
102    # the test set data augmentation only rescales the images
103    # is this enough to test the network correctly? if you want a more manual
104    # representation of fitting the input data use for loop
105    validate_datagen.fit(validate_shape_dataset)
106    validate_generator = datagen.flow(validate_shape_dataset,
107                                    validate_y_dataset,
108                                    batch_size = 32)
109
110    # the code below fits the training data that is loaded by pickle file
111    # to prevent memory error, 1/2 of the number of data inputs are feed first
112    # an epoch define the input being run once from
113    # the architecture of the cnn is:
114    # 2DConv -> ReLU -> MaxPool -> 2DConv -> ReLU -> MaxPool -> Flatten() ->
115    # Fully connected 2-layer neural network
116    # 128 neurons for the first layer -> ReLU -> 128 for hidden layer -> ReLU
117    # -> 3 neurons for output layer -> softmax
118
119    # compute quantities required for featurewise normalization
120    datagen.fit(train_shape_dataset)
121    #early stopping = EarlyStopping(monitor = 'val_loss', patience = 2)
122    # fits the model on batches with real-time data augmentation
123    train_generator = datagen.flow(train_shape_dataset,
124                                  train_y_dataset,
125                                  batch_size = 32)
126    classifier.fit_generator(train_generator, #train generator
127                           steps_per_epoch = len(train_shape_dataset) / 32,
128                           epochs = 20,
129                           validation_data = validate_generator,
130                           validation_steps = 300)
131
132    '''#early stopping prevent overfitting after the second half
133    early_stopping = EarlyStopping(monitor = 'val_loss', patience = 2)
134    # feed the same data generator the other half of the dataset
135    datagen.fit(train_shape_halfdataset)
136    train_generator_half = datagen.flow(train_shape_halfdataset,
137                                       train_y_halfdataset,
138                                       batch_size = 32)
139    classifier.fit_generator(train_generator_half,
140                           steps_per_epoch = len(train_shape_halfdataset) / 32,
141                           epochs = 20,
142                           callbacks = [early_stopping],
143                           validation_data = validate_generator,
144                           validation_steps = 300)'''
145
146    #evaluate the model
147    score = classifier.evaluate(validate_shape_dataset,
148                              validate_y_dataset,
149                              verbose = 0)
150
151    print ("Test loss: ", score[0])
152    print ("Test accuracy", score[1])
153
154    classifier.save('model.h5')
155
156    # Save the model to the Cloud Storage bucket's jobs directory
157    with file_io.FileIO('model.h5', mode='r') as input_f:
158        with file_io.FileIO(job_dir + '/model.h5', mode='w+') as output_f:
159            output_f.write(input_f.read())
160
161    if __name__ == '__main__':
162        # Parse the input arguments for common Cloud ML Engine options
163        parser = argparse.ArgumentParser()
164        parser.add_argument('--train-file',
165                            help='local path of pickle file')
166        parser.add_argument('--job-dir',
167                            help='Cloud storage bucket to export the model')
168        args = parser.parse_args()
169        arguments = args.__dict__
170        train_model(**arguments)

```

The loss at the beginning of training for this CNN modification is correct because  $-\ln(1/4) = 1.3862943611$ , which is very close as indicated in the output of the CNN.