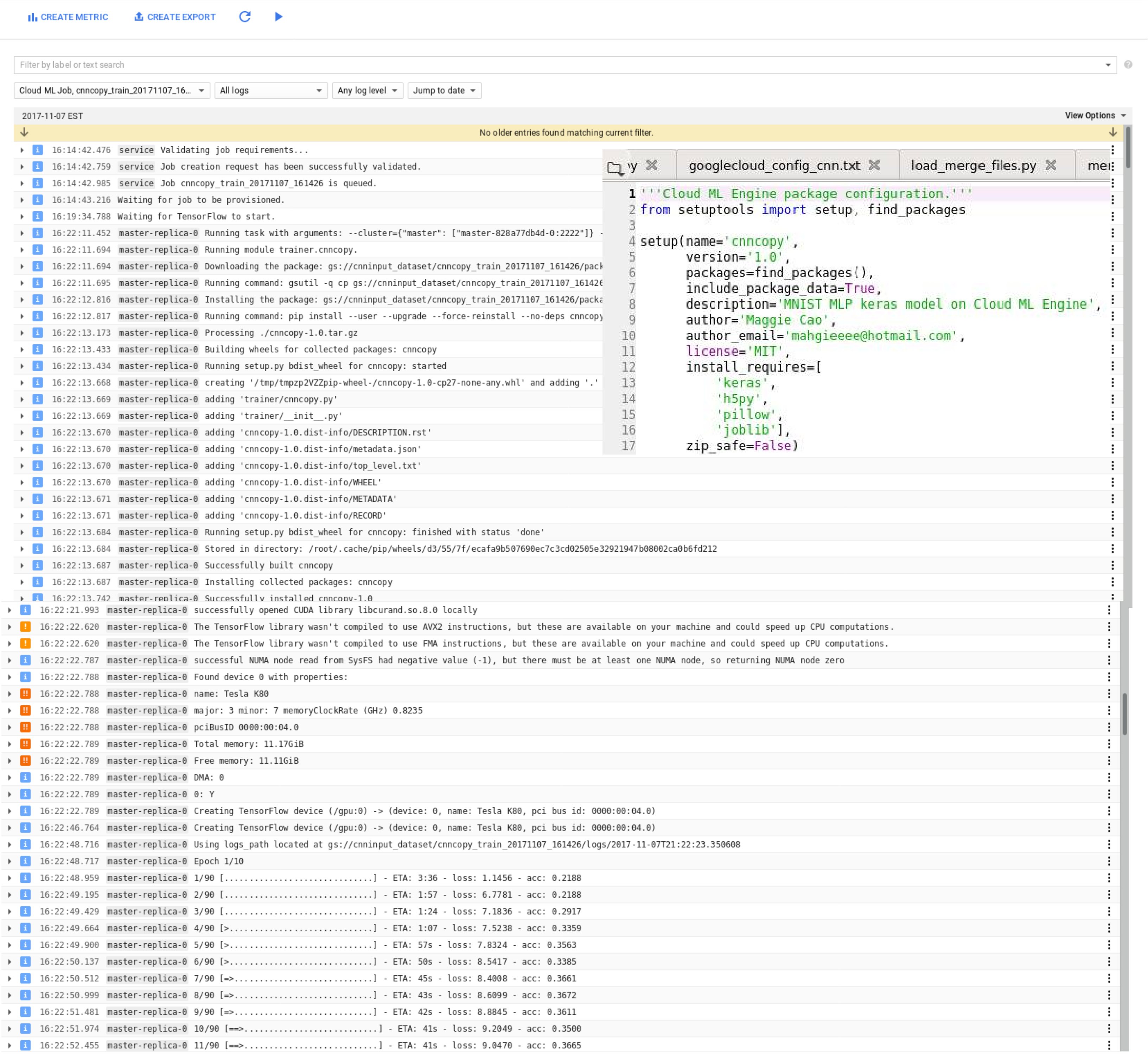
When I ran the same code in google cloud, the machine only computed the 32 epochs for the first half of the dataset in the google cloud's regular machines, the machine returns an exit status of -9, which means that the program exited because there isn't enough memory in the machine to execute the program. 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.



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/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 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:

```
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://cnninput_dataset/data/random_shapes.pkl', '--job-dir', u'gs://cnn…
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 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…
16:42:24.882 Finished tearing down TensorFlow.
16:43:09.789 Job failed.
```

```python
def train_model(loader = 'random_shapes.pkl', job_dir = './', **args):
    from keras.models import Sequential
    from keras.layers import Conv2D
    from keras.layers import MaxPooling2D
    from keras.layers import Flatten
    from keras.layers import Dense
    from keras.preprocessing.image import ImageDataGenerator
    from datetime import datetime # for filename conventions
    from tensorflow.python.lib.io import file_io # for better file I/O
    #import h5py # for saving the model
    import joblib

    #set the loggining path for ML Engine logging to storage bucket
    logs_path = job_dir + '/logs/' + datetime.now().isoformat()
    print('Using logs_path located at {}'.format(logs_path))

    with open(loader, 'rb') as f:
        save = joblib.load(f)
        train_shape_dataset = save['train_shape_dataset']
        train_y_dataset = save['train_y_dataset']
        train_shape_halfdataset = save['train_shape_halfdataset']
        train_y_halfdataset = save['train_y_halfdataset']
        #test_shape_dataset = save['test_shape_dataset']
        #test_y_dataset = save['test_y_dataset']
        #rectangle_ylabel = save['rectangle_ylabel']
        del save  # hint to help gc free up memory

    # Initialising the CNN, adding a layer
    classifier = Sequential()

    # Step 1 - Convolution
    classifier.add(Conv2D(32, (3, 3), input_shape = (300, 300, 3), activation

    # Step 2 - Pooling
    classifier.add(MaxPooling2D(pool_size = (2, 2)))

    # Adding a second convolutional layer
    classifier.add(Conv2D(32, (3, 3), activation = 'relu'))

    classifier.add(MaxPooling2D(pool_size = (2, 2)))

    # Step 3 - Flattening
    classifier.add(Flatten())
    #Dense function is used to add a fully connected layer at the end
    # Step 4 - Full connection
    classifier.add(Dense(units = 128, activation = 'relu'))
    classifier.add(Dense(units = 3, activation = 'softmax'))

    # Compiling the CNN #change 'binary_crossentropy to categorical'
    classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])

    # Part 2 - Fitting the CNN to the images

    #augmentation configuration to prevent overfitting
    datagen = ImageDataGenerator(rescale = 1./255,
                                 shear_range = 0.2,
                                 zoom_range = 0.2,
                                 horizontal_flip = True)

    #augmentation configuration for rescaling test images
    test_datagen = ImageDataGenerator(rescale = 1./255)

    #circle_train = np.array(circle_train)
    #circle_ylabel = np.array(circle_ylabel)

    datagen.fit(train_shape_dataset)

    batches = 0
    #flow() creates batches of randomly transformed images
    for x_batch, y_batch in datagen.flow(train_shape_dataset,
                                         train_y_dataset,
                                         batch_size = 32,
                                         save_to_dir = "shapes/train/",
                                         save_prefix = "shapes",
                                         save_format = "jpeg"):
        #datagen.fit(x_batch) #or (x_batch, y_batch for output of classes)
        classifier.fit(x_batch, y_batch)
        batches += 1
        if batches >= len(train_shape_dataset) / 32:
            break #without break, generator will loop indefinitely

    datagen.fit(train_shape_halfdataset)

    batches1 = 0
    #flow() creates batches of randomly transformed images
    for x_batch1, y_batch1 in datagen.flow(train_shape_halfdataset,
                                           train_y_halfdataset,
                                           batch_size = 32,
                                           save_to_dir = "shapes/train/",
                                           save_prefix = "half_shapes",
                                           save_format = "jpeg"):
        #datagen.fit(x_batch) #or (x_batch, y_batch for output of classes)
        classifier.fit(x_batch1, y_batch1)
        batches1 += 1
        if batches >= len(train_shape_halfdataset) / 32:
            break #without break, generator will loop indefinitely

    datagen.fit(rectangle_train)

    batches2 = 0
    '''#flow() creates batches of randomly transformed images
    for x_batch2, y_batch2 in datagen.flow(rectangle_train,
                                           rectangle_ylabel,
                                           batch_size = 32,
                                           save_to_dir = "shapes/",
                                           save_prefix = "rectangles",
                                           save_format = "jpeg"):
        #datagen.fit(x_batch) #or (x_batch, y_batch for output of classes)
        classifier.fit(x_batch2, y_batch2)
        batches2 += 1
        if batches >= len(rectangle_train) / 32:
            break #without break, generator will loop indefinitely'''


    #classifier.summary() ?

    # Save the model locally
    classifier.save('model.h5')

    # Save the model to the Cloud Storage bucket's jobs directory
    with file_io.FileIO('classifiermodel.h5', mode='r') as input_f:
        with file_io.FileIO(job_dir + '/model.h5', mode='w+') as output_f:
            output_f.write(input_f.read())
```

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 versions 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:

```python
def train_model(train_file = 'gs://cnninput_dataset/data/random_shapes.pkl',
                job_dir = './',
                **args):
    # set the loggining path for ML Engine logging to storage bucket
    logs_path = job_dir + '/logs/' + datetime.now().isoformat()
    print('Using logs_path located at {}'.format(logs_path))

    # need tensorflow to open file descriptor in order for google cloud to
    # process it (instead of 'with open(loader, 'rb' as f:')
    with file_io.FileIO(train_file, mode='r') as f:
        # joblib loads compressed files consistenting of large datasets
        # efficiently.
        save = joblib.load(f)
        train_shape_dataset = save['train_shape_dataset']
        train_y_dataset = save['train_y_dataset']
        #train_shape_halfdataset = save['train_shape_halfdataset']
        #train_y_halfdataset = save['train_y_halfdataset']
        validate_shape_dataset = save['validate_shape_dataset']
        validate_y_dataset = save['validate_y_dataset']
        del save  # hint to help gc free up memory

    # Initialising the CNN by adding a simple sequential layer
    classifier = Sequential()

    # Step 1:
    # Sequential layer consists of Convolution of type 3 by 3 convolutional
    # window with 32 output filters(dimensionality of output space) for each
    # input image uses reLU layers, which is a
    # ''a nonlinear layer, network with relu is trained faster without creating
    # a decrease in accuracy  @ adeshpande3.github.io''
    classifier.add(Conv2D(32, (3, 3), input_shape = (300, 300, 3), activation =

    # Step 2:
    # Max Pooling downsamples the number pixels per neuron and create a max
    # number that describes those features in a pool_size of 2 by 2
    # change pool size from (2,2) to (8,8) to (4,4)
    classifier.add(MaxPooling2D(pool_size = (4, 4)))

    # Adding a second convolutional layer, which is the same as the first one
    classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
    # change pool size from (2,2) to (8,8)
    classifier.add(MaxPooling2D(pool_size = (4, 4)))
    # Dropout layers at the second convolutional layer before flattening
    classifier.add(Dropout(0.25))

    # Step 3: Flattening the convolutional layers for input into a fully
    # connected layer
    classifier.add(Flatten())

    # Step 4:
    # Fully connected: Dense function is used to add a fully connected
    # 3 layer perceptron at the end
    classifier.add(Dense(units = 128, activation = 'relu'))
    # dropout at the first layer perceptron
    classifier.add(Dropout(0.25))
    # adding second hidden layer - remove if accuracy decreases or loss increases
    classifier.add(Dense(units = 128, activation = 'relu'))
    classifier.add(Dropout(0.35))
    # softmax classifier as an activation from the last layer perceptron
    # units represent number of output classes
    # the output classes are triangle, rectangle, square, circle
    classifier.add(Dense(units = 4, activation = 'softmax'))

    # Compiling the CNN:
    # check if optimizer adam is good, categorical_crossentropy is for
    # multi-class network, multilabel with intersection needs binary_crossentropy
    # and sigmoid activations
    # change from adam to rmsprop
    classifier.compile(optimizer = 'adam',
                       loss = 'categorical_crossentropy',
                       metrics = ['accuracy'])

    # Part 2:
    # Feeding CNN the input images and fitting the CNN
    # CNN uses data augmentation configuration to prevent overfitting
    # datagen augmentation is for training data input
    datagen = ImageDataGenerator(rescale = 1./255,
                                 shear_range = 0.2,
                                 zoom_range = 0.2,
                                 horizontal_flip = True)

    # augmentation configuration for rescaling images used for validation
    validate_datagen = ImageDataGenerator(rescale = 1./255)
    # the test set data augmentation only rescales the images
    # is this enough to test the network correctly? if you want a more manual
    # representation of fitting the input data use for loop
    validate_datagen.fit(validate_shape_dataset)
    validate_generator = datagen.flow(validate_shape_dataset,
                                      validate_y_dataset,
                                      batch_size = 32)

    # the code below fits the training data that is loaded by pickle file
    # to prevent memory error, 1/2 of the number of data inputs are feed first
    # an epoch define the input being run once from
    # the architecture of the cnn is:
    # 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

    # compute quantities required for featurewise normalization
    datagen.fit(train_shape_dataset)
    #early_stopping = EarlyStopping(monitor = 'val_loss', patience = 2)
    # fits the model on batches with real-time data augmentation
    train_generator = datagen.flow(train_shape_dataset,
                                   train_y_dataset,
                                   batch_size = 32)
    classifier.fit_generator(train_generator, #train generator
                             steps_per_epoch = len(train_shape_dataset) / 32,
                             epochs = 20,
                             validation_data =  validate_generator,
                             validation_steps = 300)

    '''#early stopping prevent overfitting after the second half
    early_stopping = EarlyStopping(monitor = 'val_loss', patience = 2)
    # feed the same data generator the other half of the dataset
    datagen.fit(train_shape_halfdataset)
    train_generator_half = datagen.flow(train_shape_halfdataset,
                                        train_y_halfdataset,
                                        batch_size = 32)
    classifier.fit_generator(train_generator_half,
                             steps_per_epoch = len(train_shape_halfdataset) / 32,
                             epochs = 20,
                             callbacks = [early_stopping],
                               validation_data =  validate_generator,
                               validation_steps = 300)'''

     #evaluate the model
     score = classifier.evaluate(validate_shape_dataset,
                                 validate_y_dataset,
                                 verbose = 0)
     print ("Test loss: ", score[0])
     print ("Test accuracy", score[1])

     classifier.save('model.h5')

     # Save the model to the Cloud Storage bucket's jobs directory
     with file_io.FileIO('model.h5', mode='r') as input_f:
         with file_io.FileIO(job_dir + '/model.h5', mode='w+') as output_f:
             output_f.write(input_f.read())


if __name__ == '__main__':
     # Parse the input arguments for common Cloud ML Engine options
     parser = argparse.ArgumentParser()
     parser.add_argument('--train-file',
                         help='local path of pickle file')
     parser.add_argument('--job-dir',
                         help='Cloud storage bucket to export the model')
     args = parser.parse_args()
     arguments = args.__dict__
     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.