Classification Accuracies of Watermarked Images Using a VGG Type Neural Network Model

Ryan Bass & Josh Ohm & Elizabeth Sheetz

Introduction

The objective of this project is to implement and compare existing Neural Networks used in image classification. More specifically, the effect of digital watermarks on the classification accuracies of the models will be studied. In this project, Tensorflow will be used to apply digital watermarks on a test set from ImageNet. Then, the classification accuracies of various neural network models on the watermarked images will be analyzed and compared.

Dataset

We will be using the ImageNet dataset for this project. ImageNet, as its name implies, is an image database that organizes images into classes according to the WordNet hierarchy, with each class containing over five hundred nodes on average[2]. ImageNet provides over 1.3 million images of 1000 classes for the purpose of testing machine learning and deep learning algorithms, and it is generally the most frequently used dataset for deep learning researchers. In this project, we will be using 32 different images.

Network Architecture

The VGG Neural Network Architecture will be used during this project. The name VGG is derived from the name of its designer, the Oxford Visual Geometry Group. The group released the white paper in 2017a . [4]

VGG utilizes 3x3 convolution and 2x2 pooling operations . Inputs go directly to the first convolutional layers neurons, then max pooling of the the wights occurs. Max pooling is often grouped in with with the Convolution Layer it is summing. Thus we simply call this a hidden layer. A nice graphic of this can be shown in figure 1 and figure 2.

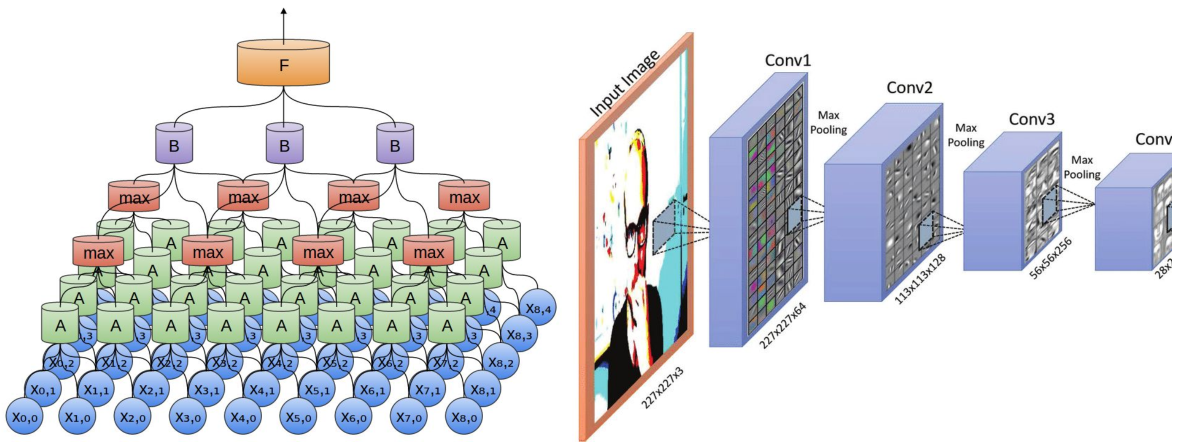


Figure 1: An example of a 2D CNN with Max pooling (Left) [5] . Visualization of a CNN for use in Image Recognition (Right) [6]

Table 1

Number Of Inputs	Hidden Layers	Filter Size, Number	Activation Function
256x256x3	65536	8x8	Softmax

Tensorflow will be used on a NVIDIA JETSON TK1 Development Board.

Methods

Cost Function - A "Style" Cost Function will be used. This was chosen due its common use by image classifiers. The "Style" an image is determined based on the correlation between activations across channels. This will be implemented using multinomial logistic regression, or more commonly referred to as *softmax regression*. Softmax regression applies nonlinearity to the output of the network, then calculates the cross entropy between the normalized predictions and the label index. In turn it gives a decisive output from 0-1.

Initializing weights and biases - Weights and biases are implemented at random during the learning process. This is due to the fact the network does not need to know exactly where something is in reference to the larger image in order to detect it. These weights and biases will be different for each class.

Batch Size - 32 Images

In general, the following steps are required in order to complete this experiment:

1) Download jpg files from a source

2) Apply watermarks to the files and create a separate directory saving the watermarked images
3) Convert from jpg to Tensor DataSet
4) Do any other manipulation/preparation required by the NN's (eg. divide by 255 to scale the images)
5) Acquire/implement various pre-trained NN's
6) Test the NNs with non-watermarked data.
7) Test the NNs with watermarked data.
8) Compare the results of 6-7 and make conclusions.

<u>Watermarking Process</u> - The program used to watermark images in this project is named "watermarking.py". First, the modules 'Image', 'ImageDraw', and 'ImageFont' are imported from Python Image Library (PIL), and the modules 'time' and 'os' are also imported. Next, the input and output paths for the image, the text used in the watermark, and the font path, which are all used in the watermarking process, are defined. The program then loads the image from the input path and makes it editable. After the text, position, color and font of the watermark are all defined, the watermark is drawn onto the image and the watermarked image is displayed. Finally, the watermarked image is saved to the output path.

Results

In general, we were able to successfully complete steps 1-3 marked above. The original image used in watermarking.py and the watermarked image are shown in figures 3 and 4, respectively. The outputs of the tensorflow tutorial that was implemented are shown in figures 5, 6, and 7.

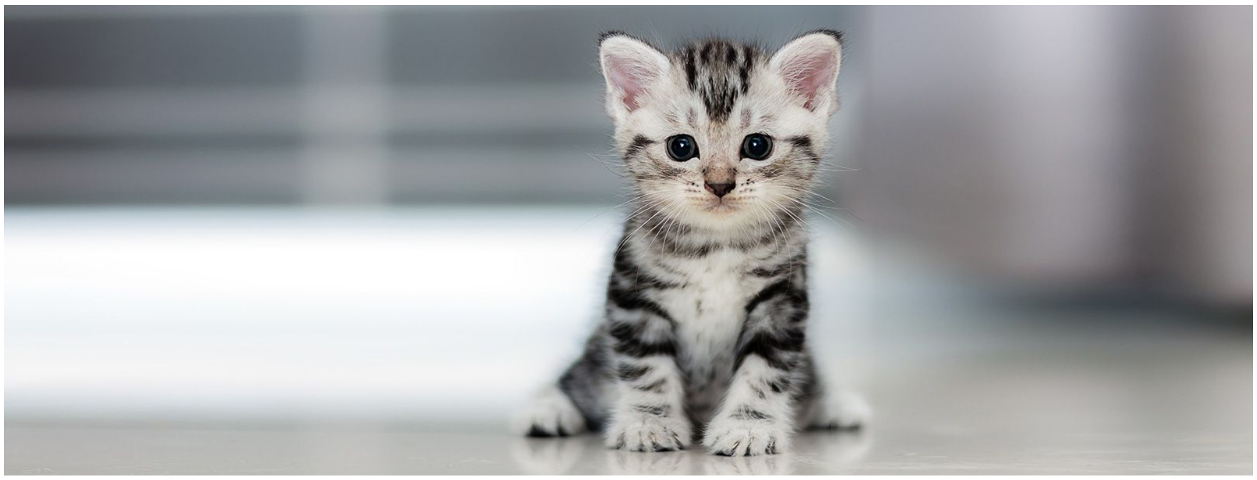


Figure 3: Original Image

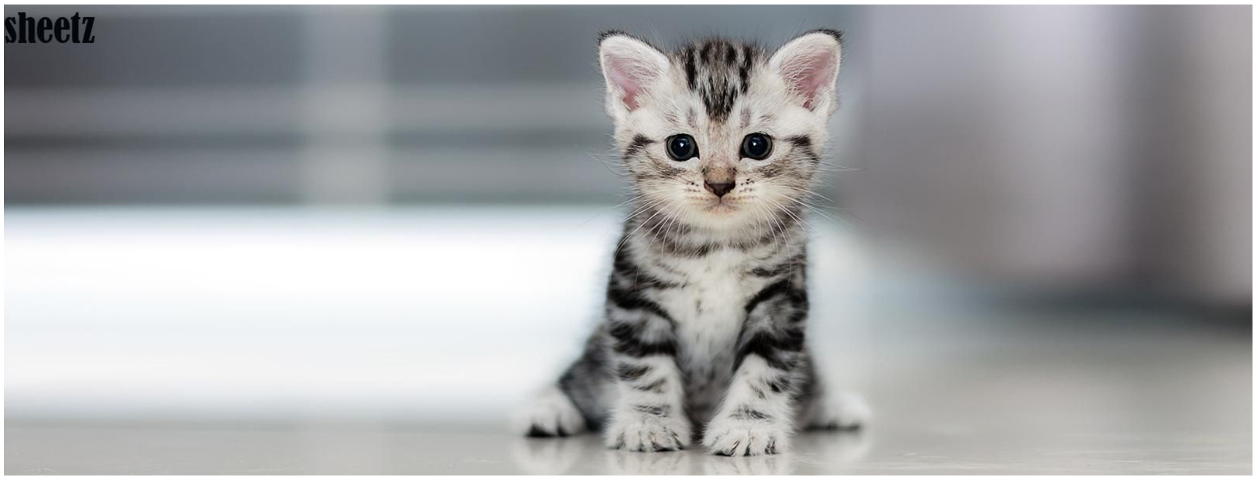


Figure 4: Watermarked Image

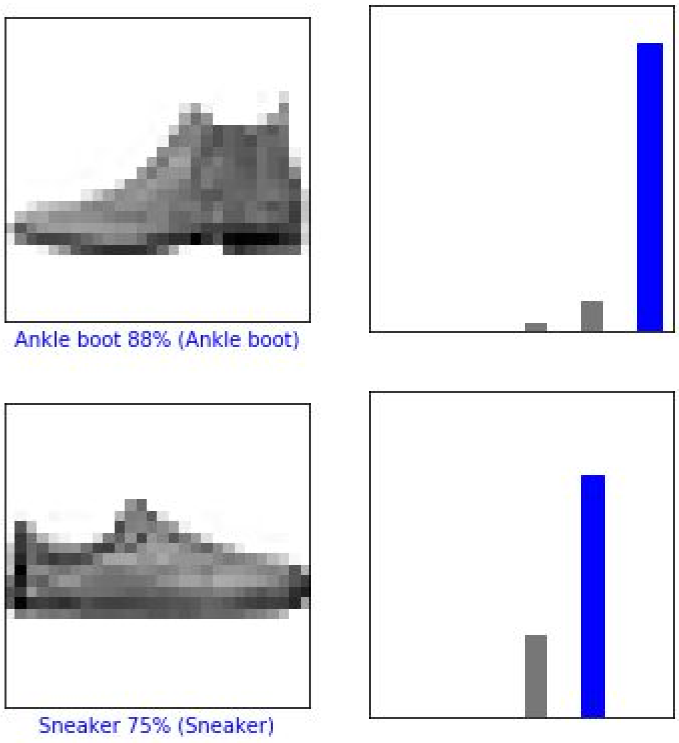


Figure 5: Default Output of tensorflow_example.py (1/3)

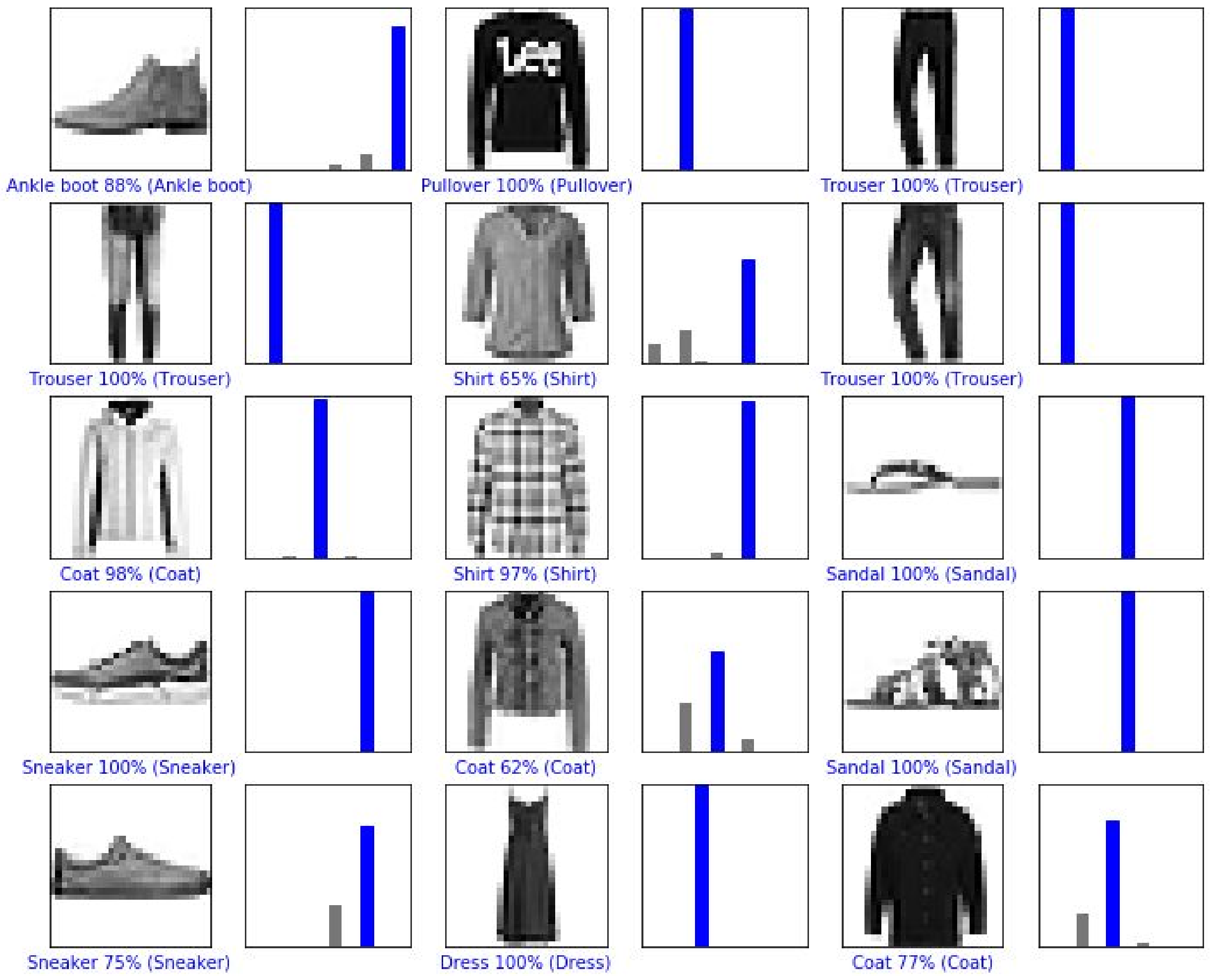


Figure 6: Default Output of tensorflow_example.py (2/3)

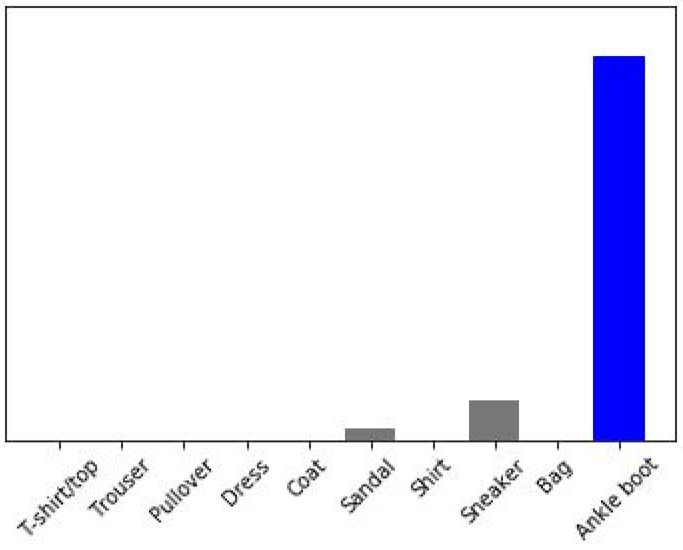
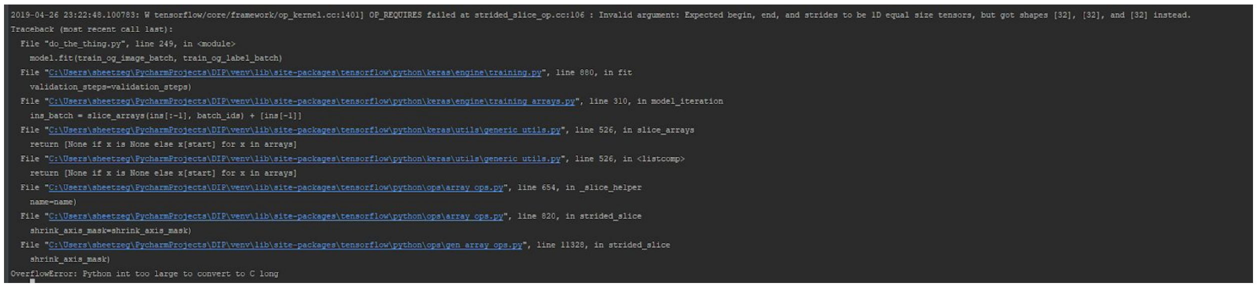


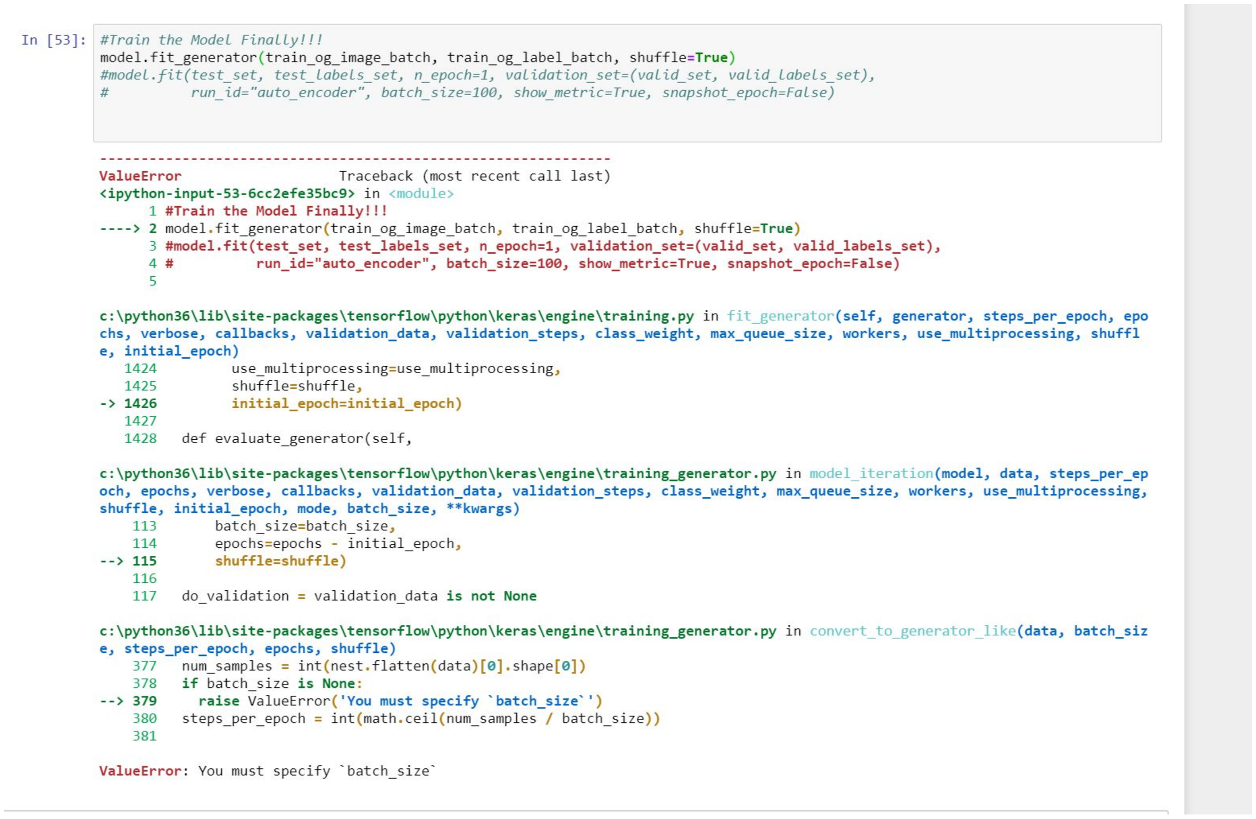
Figure 7: Default Output of tensorflow_example.py (3/3)

We were not able to successfully implement NN's. We believe there must either be an error in the data setup or the NN configuration. Initially, we used the function 'model.fit' and received the error in figure 8. Upon suggestion, we changed 'model.fit' to 'model.fit_generator', but when we did this, we got the value error in figure 9 that states "You must specify 'batch_size'". We tried adding a specific value for batch_size, but we got type error in figure 10 stating that "fit_generator() got an unexpected keyword argument 'batch_size'".



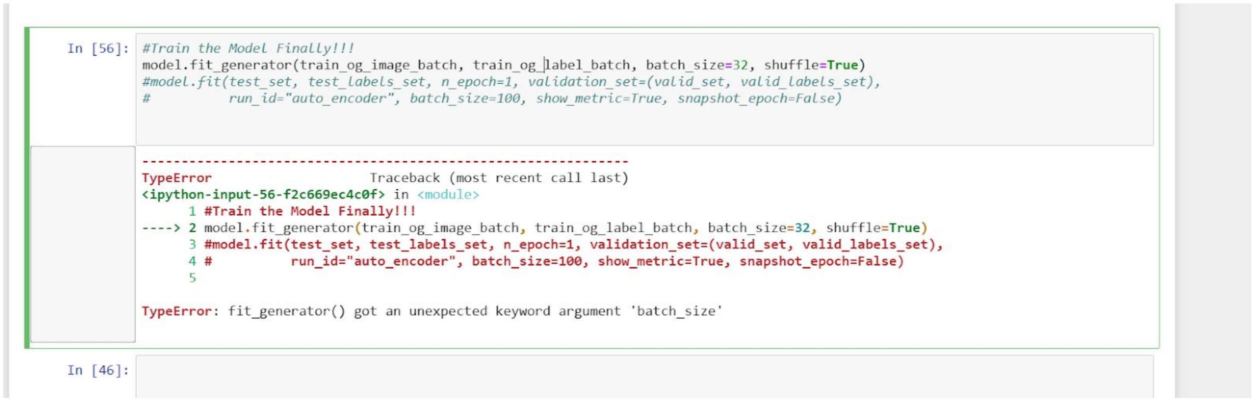
```
2019-04-26 23:22:48.100783: W tensorflow/core/framework/op_kernel.cc:1401] OP_REQUIRES failed at strided_slice_op.cc:106 : Invalid argument: Expected begin, end, and strides to be 1D equal size tensors, but got shapes [32], [32], and [32] instead.
Traceback (most recent call last):
  File "do_the_thing.py", line 249, in <module>
    model.fit(train_og_image_batch, train_og_label_batch)
  File "C:\Users\sheetzeg\PycharmProjects\DIP\venv\lib\site-packages\tensorflow\python\keras\engine\training.py", line 880, in fit
    validation_steps=validation_steps)
  File "C:\Users\sheetzeg\PycharmProjects\DIP\venv\lib\site-packages\tensorflow\python\keras\engine\training_arrays.py", line 310, in model_iteration
    ins_batch = slice_arrays(ins[:-1], batch_ids) + [ins[-1]]
  File "C:\Users\sheetzeg\PycharmProjects\DIP\venv\lib\site-packages\tensorflow\python\keras\utils\generic_utils.py", line 526, in slice_arrays
    return [None if x is None else x[start] for x in arrays]
  File "C:\Users\sheetzeg\PycharmProjects\DIP\venv\lib\site-packages\tensorflow\python\keras\utils\generic_utils.py", line 526, in <listcomp>
    return [None if x is None else x[start] for x in arrays]
  File "C:\Users\sheetzeg\PycharmProjects\DIP\venv\lib\site-packages\tensorflow\python\ops\array_ops.py", line 654, in _slice_helper
    name=name)
  File "C:\Users\sheetzeg\PycharmProjects\DIP\venv\lib\site-packages\tensorflow\python\ops\array_ops.py", line 820, in strided_slice
    shrink_axis_mask=shrink_axis_mask)
  File "C:\Users\sheetzeg\PycharmProjects\DIP\venv\lib\site-packages\tensorflow\python\ops\gen_array_ops.py", line 11328, in strided_slice
    shrink_axis_mask)
OverflowError: Python int too large to convert to C long
```

Figure 8: Overflow Error using model.fit



```
In [53]: #Train the Model Finally!!!
         model.fit_generator(train_og_image_batch, train_og_label_batch, shuffle=True)
         #model.fit(test_set, test_labels_set, n_epoch=1, validation_set=(valid_set, valid_labels_set),
         #          run_id="auto_encoder", batch_size=100, show_metric=True, snapshot_epoch=False)

         ---------------------------------------------------------------
         ValueError                    Traceback (most recent call last)
         <ipython-input-53-6cc2efe35bc9> in <module>
               1 #Train the Model Finally!!!
         ----> 2 model.fit_generator(train_og_image_batch, train_og_label_batch, shuffle=True)
               3 #model.fit(test_set, test_labels_set, n_epoch=1, validation_set=(valid_set, valid_labels_set),
               4 #          run_id="auto_encoder", batch_size=100, show_metric=True, snapshot_epoch=False)
               5

         c:\python36\lib\site-packages\tensorflow\python\keras\engine\training.py in fit_generator(self, generator, steps_per_epoch, epo
         chs, verbose, callbacks, validation_data, validation_steps, class_weight, max_queue_size, workers, use_multiprocessing, shuffl
         e, initial_epoch)
            1424         use_multiprocessing=use_multiprocessing,
            1425         shuffle=shuffle,
         -> 1426         initial_epoch=initial_epoch)
            1427
            1428   def evaluate_generator(self,

         c:\python36\lib\site-packages\tensorflow\python\keras\engine\training_generator.py in model_iteration(model, data, steps_per_ep
         och, epochs, verbose, callbacks, validation_data, validation_steps, class_weight, max_queue_size, workers, use_multiprocessing,
         shuffle, initial_epoch, mode, batch_size, **kwargs)
             113       batch_size=batch_size,
             114       epochs=epochs - initial_epoch,
         --> 115       shuffle=shuffle)
             116
             117   do_validation = validation_data is not None

         c:\python36\lib\site-packages\tensorflow\python\keras\engine\training_generator.py in convert_to_generator_like(data, batch_siz
         e, steps_per_epoch, epochs, shuffle)
             377   num_samples = int(nest.flatten(data)[0].shape[0])
             378   if batch_size is None:
         --> 379     raise ValueError('You must specify `batch_size`')
             380   steps_per_epoch = int(math.ceil(num_samples / batch_size))
             381

         ValueError: You must specify `batch_size`
```

Figure 9: Value Error for model.fit_generator without 'batch_size'



```
In [56]: #Train the Model Finally!!!
         model.fit_generator(train_og_image_batch, train_og_label_batch, batch_size=32, shuffle=True)
         #model.fit(test_set, test_labels_set, n_epoch=1, validation_set=(valid_set, valid_labels_set),
         #          run_id="auto_encoder", batch_size=100, show_metric=True, snapshot_epoch=False)

         ---------------------------------------------------------------
         TypeError                      Traceback (most recent call last)
         <ipython-input-56-f2c669ec4c0f> in <module>
               1 #Train the Model Finally!!!
         ----> 2 model.fit_generator(train_og_image_batch, train_og_label_batch, batch_size=32, shuffle=True)
               3 #model.fit(test_set, test_labels_set, n_epoch=1, validation_set=(valid_set, valid_labels_set),
               4 #          run_id="auto_encoder", batch_size=100, show_metric=True, snapshot_epoch=False)
               5

         TypeError: fit_generator() got an unexpected keyword argument 'batch_size'

In [46]:
```

Figure 10: Type Error for model.fit_generator with 'batch_size' specified

Because we were not able to figure out and resolve the problem with our code, we were unable to complete step 5, and therefore could not test and compare the NNs with watermarked data. That being said, we were able to successfully implement the initial TensorFlow tutorial and train and test a model. The code used for this is in "tensorflow_example.py".

References

1. http://digitalwatermarkingalliance.org/digital-watermarking-works/
2. http://www.image-net.org/
3. https://www.tensorflow.org/tutorials/images/deep_cnn
4. http://www.robots.ox.ac.uk/~vgg/practicals/cnn/index.html
5. http://colah.github.io/posts/2014-07-Conv-Nets-Modular/
6. https://www.researchgate.net/publication/318798243_Artificial_Intelligent_System_for_Automatic_Depression_Level_Analysis_through_Visual_and_Vocal_Expressions
7. https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/
8. https://datascience.stackexchange.com/questions/9850/neural-networks-which-cost-function-to-use