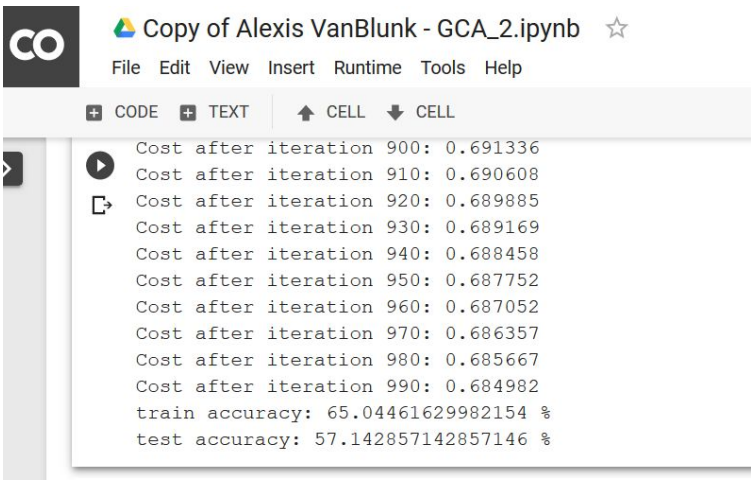
Attempts to improve accuracy:

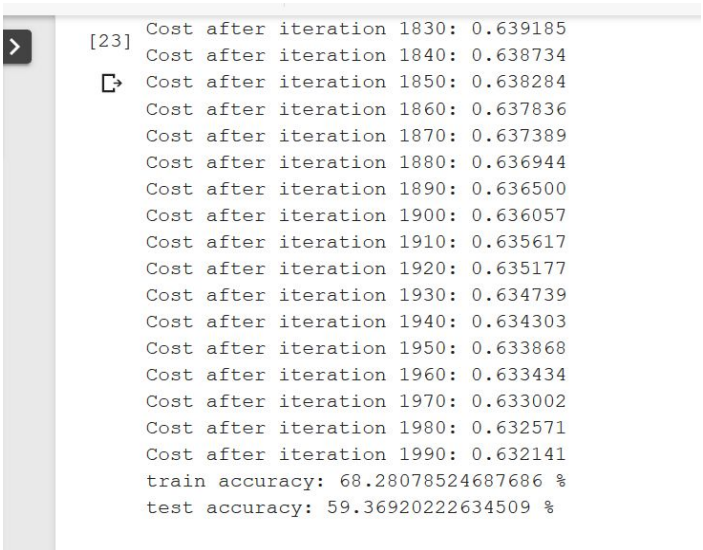
Import new training images

After figuring out how to connect the new array to our existing array, we ran into the problem of having too much data. This caused colab to crash because it did not have enough RAM to handle the additional 9000 images.

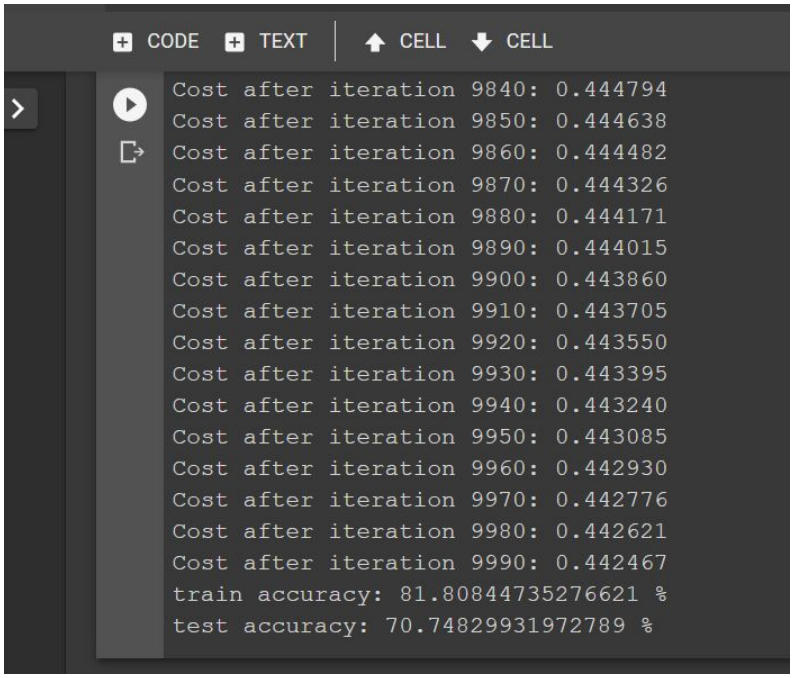
We decided to use 4000 new photos, bringing the total training photos to 8,405. We ran into problems because we did not add 4000 new zeros to our labels. Therefore, we were getting the error that the arrays were not the same shape. Once we finally were able to run the new code, we discovered that it actually decreased accuracy.



Increasing the number of iterations from 1000 to 2000 helped increase the accuracy from its first value.



Finally, we ran the test for 10,000 iterations to see how accurate we could get our code. This improved train accuracy by about 4% and test accuracy by 0.7%.

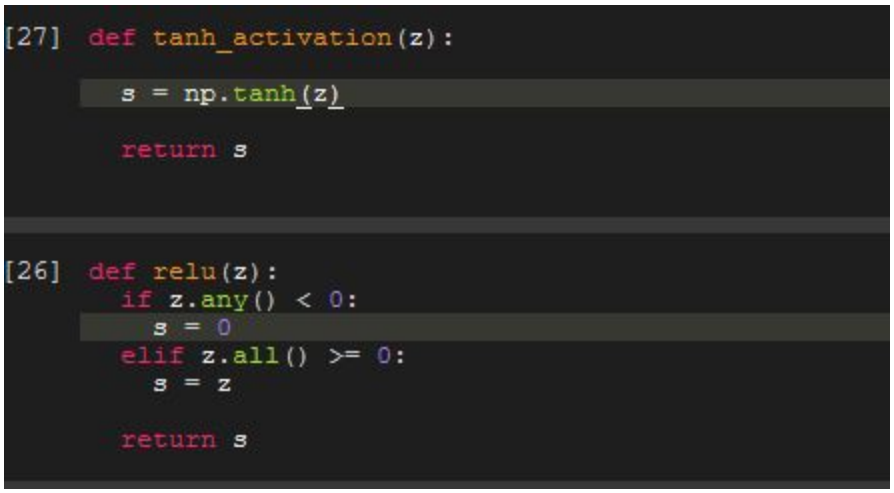


Changing Learning Rate

Increasing the learning rate from 0.002 to 0.003 caused the costs to not converge to one value then decrease, but continue oscillating between high and low values.

Changing Activation Function

An attempt was made to change the activation function for the model. Two different functions were used, the hyperbolic tangent (tanh) and the rectified linear unit (ReLU).



Because the hyperbolic tangent functions ranges between -1 and 1, the labels on the Y predictions had to be changed. For tanh, a value over 0 was positive, and a value under 0 was negative. For ReLU, a value over 0 was positive and a value equaling 0 was negative. However,

both functions caused a decrease in the effectiveness of the model. The hyperbolic tangent was very effective in training, but failed in the test. The ReLU was the opposite, with a strong test performance but a lacking train accuracy. For control, both tests were ran with 200 iterations each.

```
train accuracy: 74.91486946651531 %
test accuracy: 31.787260358688926 %
```

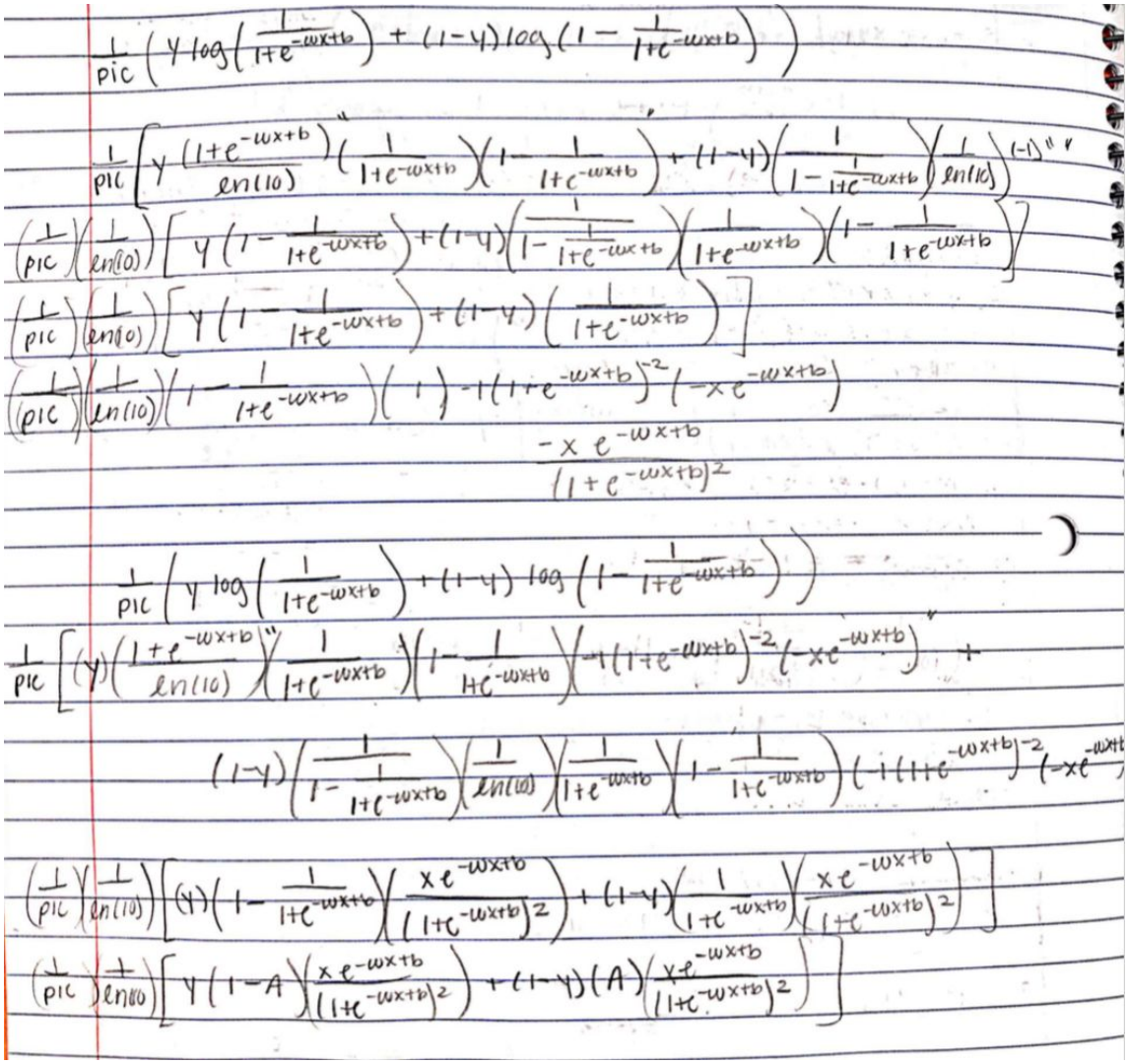
```
train accuracy: 25.153234960272414 %
test accuracy: 68.46011131725417 %
```

Further research led to the understanding that for this type of model, a sigmoid function is the best activation, which lines up with our results.

Derive the derivative of the cost function, dw and db:

The derivatives of the cost function with respect to w and b are necessary for minimizing the error associated with the cost function. In this case, the cross-entropy error (aka Log Loss) is used rather than the mean-squared error typically used for linear regression models. The cross-entropy error is designed specifically for probability outputs, which makes it appropriate for our model calculating the probability whether an image is a cat. Log-Loss makes is easy to analyze the function for both y=0 and y=1 and repeatedly adjust the weights to minimize the cost. The model will use weights that make the derivatives closest to zero, thus minimizing the error.

Two attempts of deriving dw are presented below. The derivation proved rather difficult and is thus incomplete, however the calculations emphasized the importance of the chain rule. The sigmoid function has the simplified derivative $\sigma'(x)=\sigma(x)(1-\sigma(x))$. This derivative is important to use in the chain rule and helps in the process of minimizing the cost.



Derivative Research Sources:
https://towardsdatascience.com/backward-propagation-for-feed-forward-networks-afdf9d038d21
https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html#cost-function

In a couple of paragraphs, summarize what you have learned. Discuss the limitations of the approach.

We've learned that a single neuron approach is rather limited compared to multi-neuron deep learning. There was not much that could be done to improve the code provided, seeing as it was very limited in scope. We conclude the sigmoidal function is the most applicable activation function. While the recognition accuracy exceeds the baseline 50%, the model is unsophisticated and there are likely better approaches to deep learning than the code provided. Our most successful tactic in improving this code was substantially increasing the number of iterations while adding images to the training. However, this only increased our accuracy by a minor increment of 0.7%. Overall, we learned image recognition with a single neuron approach

is restrictive. We anticipate multi-neuron deep learning is better suited for more advanced magnitudes of image recognition.

Image recognition is not done this way in production environments! Discuss how it is done by data scientists today.

Data scientists follow a general process to identify graphical data. First, they will gather an array of data that will be used in the process. The single image that is inputted will be preprocessed in a fashion that eliminates noise or unneeded information/space that surrounds an image. This is eliminated to help the program focus on key features within the RGB pixel values. Next, the image will be stripped of most of its color and replaced with vector values that will appear on an image that highlights features of the graphic. This is completed so that only the needed information, the lines on the image, is used and the program does not become distracted by color or other information that may be displayed. After the key features are identified, the scientist will build a model that will be able to label the image to fit within a specific set of criteria by reading the vector values for planes and surfaces that positions each image into a particular class based upon its usage of planes or surfaces. A network will be formed based upon the training the algorithm receives when scientists run the process over thousands of images. The model is similar to the neural network paths within the human brain and allows for detailed analysis of the picture. Once the test portion is completed, the scientists can now use the code to recognize images.

https://www.datascience.us/use-image-recognition/