

I. Definition

Project Overview

An [estimated 2 trillion photos](#) were shared online in 2015, with that number expected to continue to grow. Lacking efficient methods to examine and identify objects in these images they can only be understood by the text that is used to summarize or tag them not the actual image content.

Being able to identify different aspects of an image would provide considerable value to a brand seeking to connect with its users. This type of structured data is already [available for text](#) but what about images? There is significant value in a brand or company being able to understand when and where it's products and services are being shown in images online.

Is there a way to be able to identify the objects in an image to be able to know what brands are present in an image? What if a customer had an issue and posted an image without your company name in the text description? How would you could you identify and find such images?

In this project I created a Convolutional Neural Network that is capable of identifying brands in untagged/ unlabeled photos from a social media feed. The model I use implements a [previously trained](#) network and [transfer learning](#) to speed training. This project was [inspired](#) by a [number](#) of [different](#) sources.

Problem Statement

Can you [teach a computer](#) to recognize the brand logos of Nike and Altra in an unlabeled feed of images from Instagram?

In order to solve this problem I undertake the following:

1. Identify suitable existing Neural Networks trained for image recognition and classification
2. Implement the existing models locally and load pretrained weights
3. Obtain a dataset of images to retrain the network on the brands of interest
4. Reconfigure the final layers of the pre-existing models to classify the brands
5. Retrain the final layers of the new network using my dataset
6. Measure resulting performance
7. Optimize the model based on results

This project solves a classification problem. In its first incarnation discerning between 2 brands represents a *supervised binary classification* problem. In this case, selecting either [Nike](#) or [Altra](#) products are present.





Later, when a third category is introduced it becomes a *supervised multiclass classification* problem. [This website offers](#) a great layman's introduction to these concepts.

Metrics

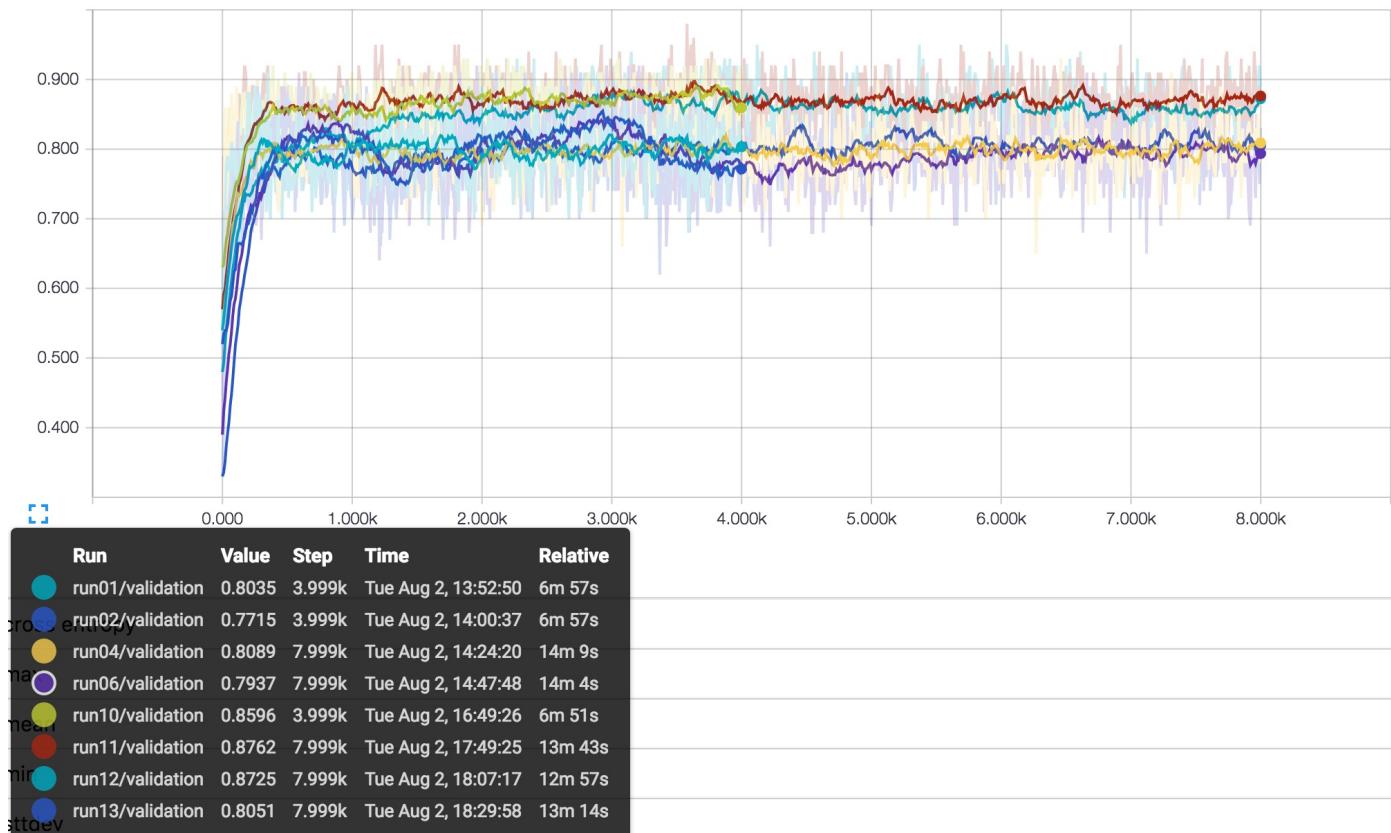
There are 2 metrics I use to gauge the performance of my model: accuracy and cross-entropy.

Accuracy

The accuracy is the primary measure of overall model performance. It is represented as a percentage of the number of images the model correctly identified divided by the total number of attempts.

Accuracy should improve as our model iterates, as shown below:

accuracy

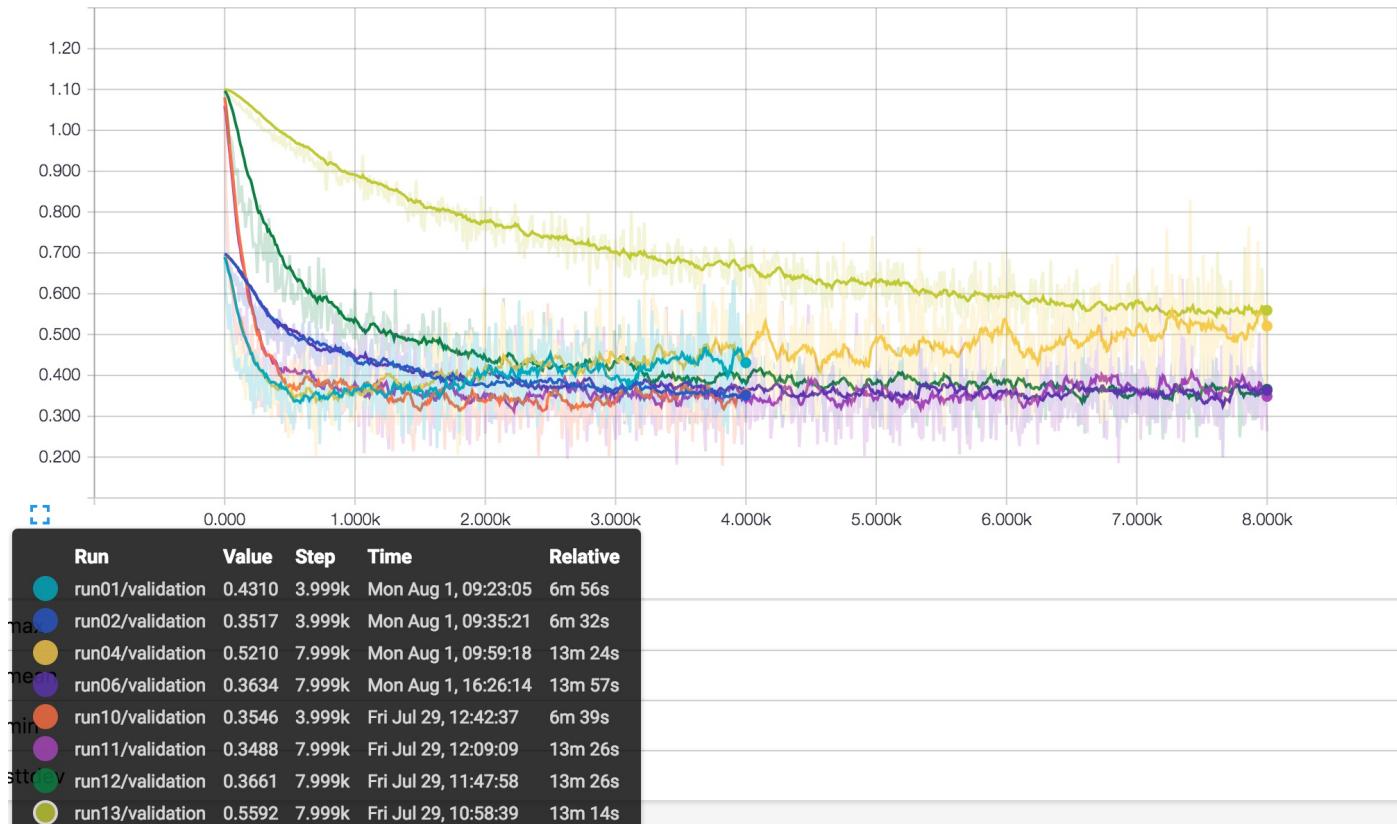


Cross-Entropy

Cross-entropy is a measure of a model's loss or cost. A high value represents a model doing a poor job. We seek to minimize the cross-entropy. A more in-depth discussion of cross-entropy is [available here](#)

In general, we expect to see cross-entropy decreasing as our model iterates, like this:

cross entropy



By using these 2 metrics we'll be able to quantify how well our model is performing via the accuracy but also how well the model is *learning* via the cross entropy.

II. Analysis

Data Exploration

2 datasets were used in this project. The majority of the images were downloaded from Instagram and represent the typical input image that could be expected from a direct feed of images flowing into Instagram at any given time. Tags were used to search for the Altra & Nike images. An example search and results can [be seen here](#).

An example Nike image



An example Altra image



First data set

Altra - 110 images (no cropped/ transformed images) Nike - 470 images
(200 full size and 270 cropped images)

Test Set #1	Training	Validation	Test
Nike			
Altra			

2nd data set

Altra - 481 images (240 full size and 241 crops) Nike - 608 images (260 full size and 348 cropped images) Neither - 453 images (452 full size)

Similar transformation were applied to this data set for cropping images. They were applied to both the Altra and Nike data sets.

The 'neither' category of images is comprised of downloading the bulk "firehose" of images from Instagram.

To supplement this data set, as well as test my models' performance on perturbations in input image shape, approximately 10% of the images are from alternative sources such as Pinboard, Facebook, or Google Image Search. Thus not all of the input images are square, as would be expected from Instagram alone, and their size varies.

Examples of difficult images

There are images that are difficult to extract information.

In real images, not just product shots, shoes tend to be relatively small and the logos are not easily readable.



Or the logos could be occluded by other objects in the scene, such as shoelaces in this example.



Or different designs call for morphing logos to match the shoes.



Occasionally there are multiple shoes of multiple brands in a single image.



Or there isn't one of our brands or even shoes in an image.



Or both of our brands of interest in a single image.



Algorithms and Techniques

The core of this project is the use of a Convolutional Neural Network.

CNNs and you

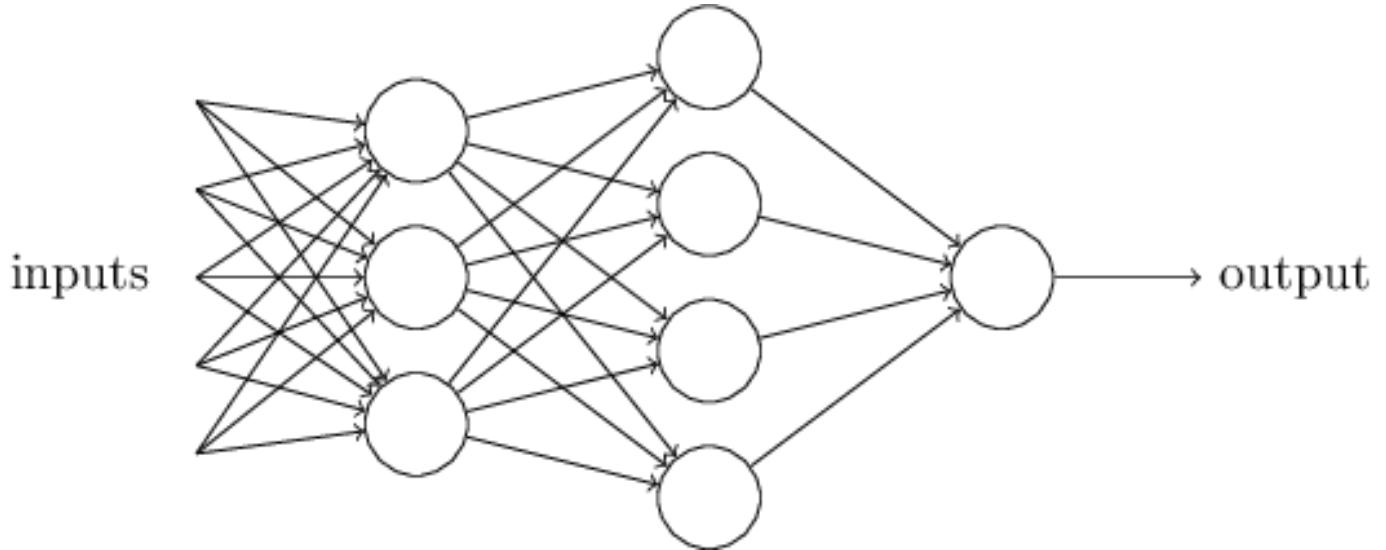
"No one teaches a child how to see"

(http://www.ted.com/talks/fei_fei_li_how_we_re_teaching_computers_to_understand_pictures)

Instead of better algorithms to explicitly define every permutation of logo that we would potentially see, machine learning and neural networks allow us to feed training images to a neural network and allow it to define for itself what the critical features are.

The first building block used in my solution was a neural network and deep learning.

[###todo] Explanation of a network

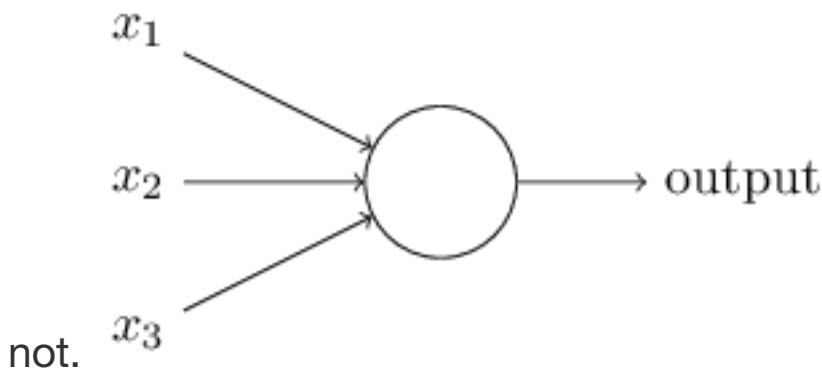


Convolutional neural networks are at the heart of these models. A way to reduce the dimensionality of the mathematics/ reduce computational complexity. Likely to have overfitting in convolved features. Pooling/ aggregating over a convolved feature, depends on mean/ max pooling

image layer -> convolved feature layer ->

Explanation of a perceptron [###todo]

Basic unit of a neural network is a neuron like node. It takes inputs and uses them along with weights to determine whether to "fire" its output or



$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

There's clearly a lot of information here to be learned and a variety of resources, [this is a good place](#) to start.

Layers -> input, hidden, output layers

Convolutional neural networks have been shown to be effective at image classification. ILSVR or ImageNet are both examples where CNNs achieve great performance and their [performance continues improving](#).

They approach, or occasionally exceed, [human performance](#) on similar benchmarks.

[###todo images that are difficult for humans to differentiate but computers get correct]

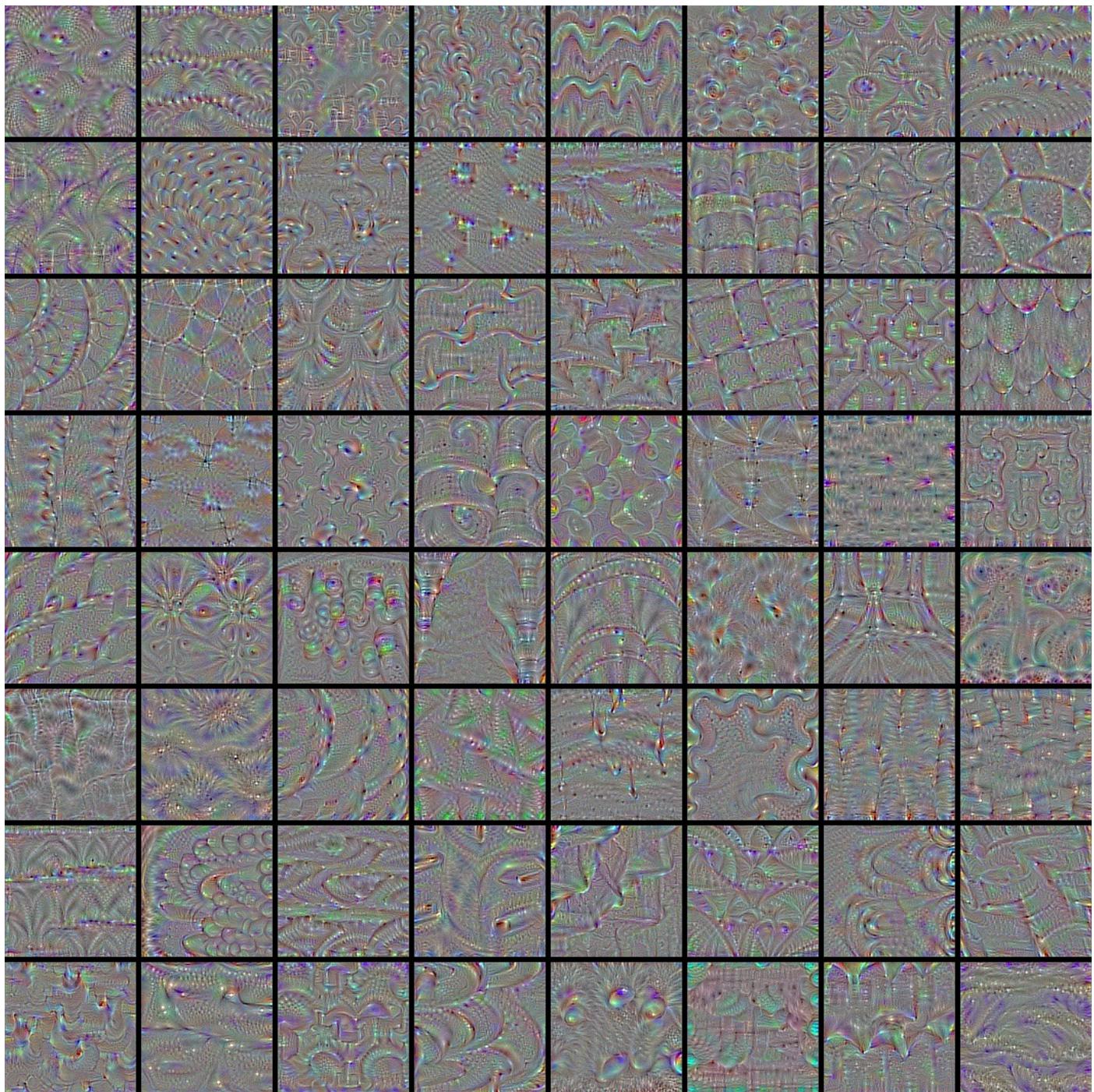
Breaking it down: [###todo - images of each section of the graph]

1. Input layers - decoding, resizing, flattening, etc to prepare images for the network
2. Decode the jpeg file, resize to a standardized size/ shape of 299x299x3 representing 299 pixels wide by 299 pixels tall by 3 layers (red, green, blue) deep

3. Convolutional layer
4. Pooling layer
5. Mixed layer
6. Softmax
7. Output layers

Exploratory Visualization

It is difficult to not anthropomorphize a computer algorithm that "sees" images. One way to help us to understand how a CNN "viewing" an image is to look at what it would define as an ideal example of a particular class. Examining different layers within our neural network, we can begin to appreciate how our network is building up features to identify objects.



In this case, here is what VGG16 would say is "99.999% a shoe"

Using this tutorial to examine what neural networks see

<https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html>

[###todo visualize layers of model] otherwise not sure this is relevant

Existing networks

In the case of the inception v3 network would take about 2 weeks to train

on 8x GPUs (~\$3k Nvidia K40s). Once trained the model and associated weights can be transferred to a smaller/ less expensive computer (in this case a MacbookPro) and used to classify images. The [Inception-v3 model](#) has over 27 million parameters and runs 5.7 billion floating point operations per inference

This approach allows us to "leverage" the resources/ computing power/ model complexity available to Google but used in our local environment.

The method I used to approach this problem utilized the significant work that large companies and research institutions have invested into this problem. I used 2 different networks to attempt this, VGG16 and Inception v3.

The VGG model was created by the Visual Geometry Group (VGG) at Oxford University and was [first described in 2015](#). Although there are multiple versions in , the one used in this work is 16-layers.

The Inception architecture [first appeared in a model called "GoogLeNet"](#) in 2014. Version 3 [was described](#) in 2015 and is the model used in this work.

Transfer Learning Pre-trained networks

Of particular interest to this project is [transfer learning](#).

Transfer learning is the ability to train a neural network in one scenario, say at Google in Mountain View, and then downloading the model and its associated weights to be reused in a different location/ time such as Austin, TX.

Additionally, pre-trained networks can be used. They have the ability to distinguish important features in images [despite being trained](#) on images that differ from the images that will be used in our implementation of the

network.

The ability to distinguish edges in images is universally applicable to classification tasks.

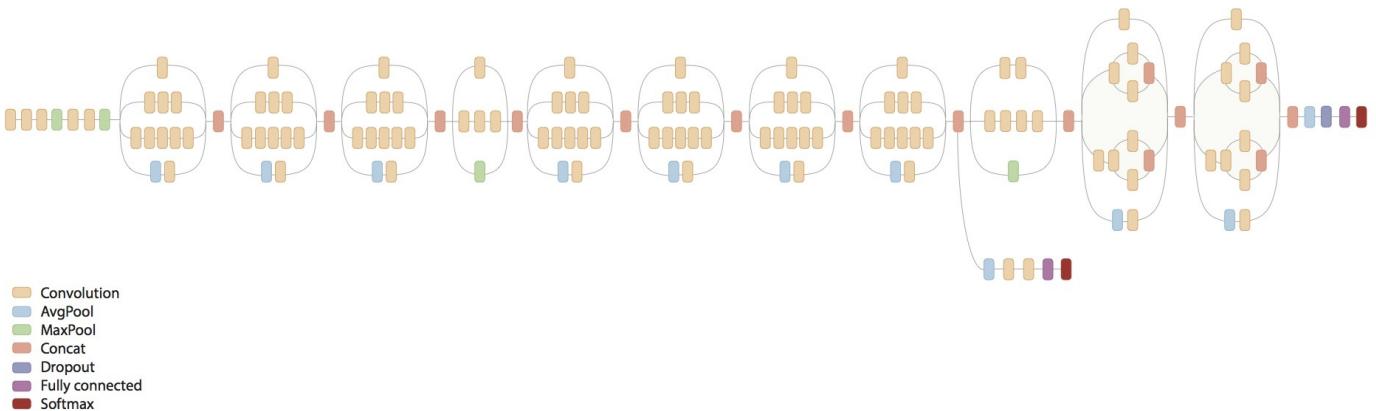
Layers of Inception- Pooling, 2D Convolution, etc.

1. Pooling layers look like this [###todo add more layer info]
2. Conv2D layers look like this

[###todo image of first few layers of a CNN]

Replacing the final layer of a network and replacing it with our preferred classifications allows us to leverage the power of a network that may have been trained for weeks on an array of high-powered GPUs and retraining it in a few minutes for our task.

This is what the Inception v3 network looks like:



1. Identify a suitable framework or library to build a pre-trained network

There are a number of different options for building and loading existing network architectures. [Caffe](#), [Keras](#), and [TensorFlow](#) are all excellent options.

For the purpose of my project I limited my choices to Keras &

TensorFlow. Their descriptions from their developers are provided below:

Keras

[Keras](#) is a minimalist, highly modular neural networks library, written in Python and capable of running on top of either TensorFlow or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

TensorFlow

[TensorFlow](#) is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that flow between them. This flexible architecture lets you deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device without rewriting code. TensorFlow also includes TensorBoard, a data visualization toolkit.

TensorFlow was originally developed by researchers and engineers working on the Google Brain team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research

2. Construct an existing network

Keras provides [a tutorial](#) on loading and utilizing the VGG16 architecture.

TensorFlow provides instructions for the reuse of the Inception v3 network.

3. Load pre-trained weights into the previously built network

The weights that effectively "remember" the training of a neural network are loaded. In some cases these weight files can be quite large. For the VGG16 model [the weight file](#) topped 550 mb

4. Modify the pre-existing model for our classification tasks

Remove the original model's top layers and replace with ones suitable to perform our classification tasks.

5. Update the bottlenecks

the penultimate layers of the model that perform the actual classification.

6. Retrain the network on my images of interest

I investigate 2 options- binary classification or a closed world problem where images of only Nike or Altra shoes are expected and an open world multiclass classification of images that could be of Nike or Altra shoes or some third category I've cleverly titled 'Neither'.

7. Evaluate model performance

Using training, validation, and testing values for accuracy and cross-entropy

8. Optimize hyperparameters and compare performance

1. Select the optimum solution

2. download existing pre-trained model and weights [Keras](#) and [TensorFlow](#) both offer frameworks for creating the CNNs used, transferring weights, and popping layers to retrain.
3. remove/pop the final layers that are used to classify ImageNet images (1000 classes)
4. Replace final layers with ones that suit my needs, in this case 3 classes
5. Retrain the network on images that I provide
6. Measure performance and vary hyperparameters seeking to maximize the models performance

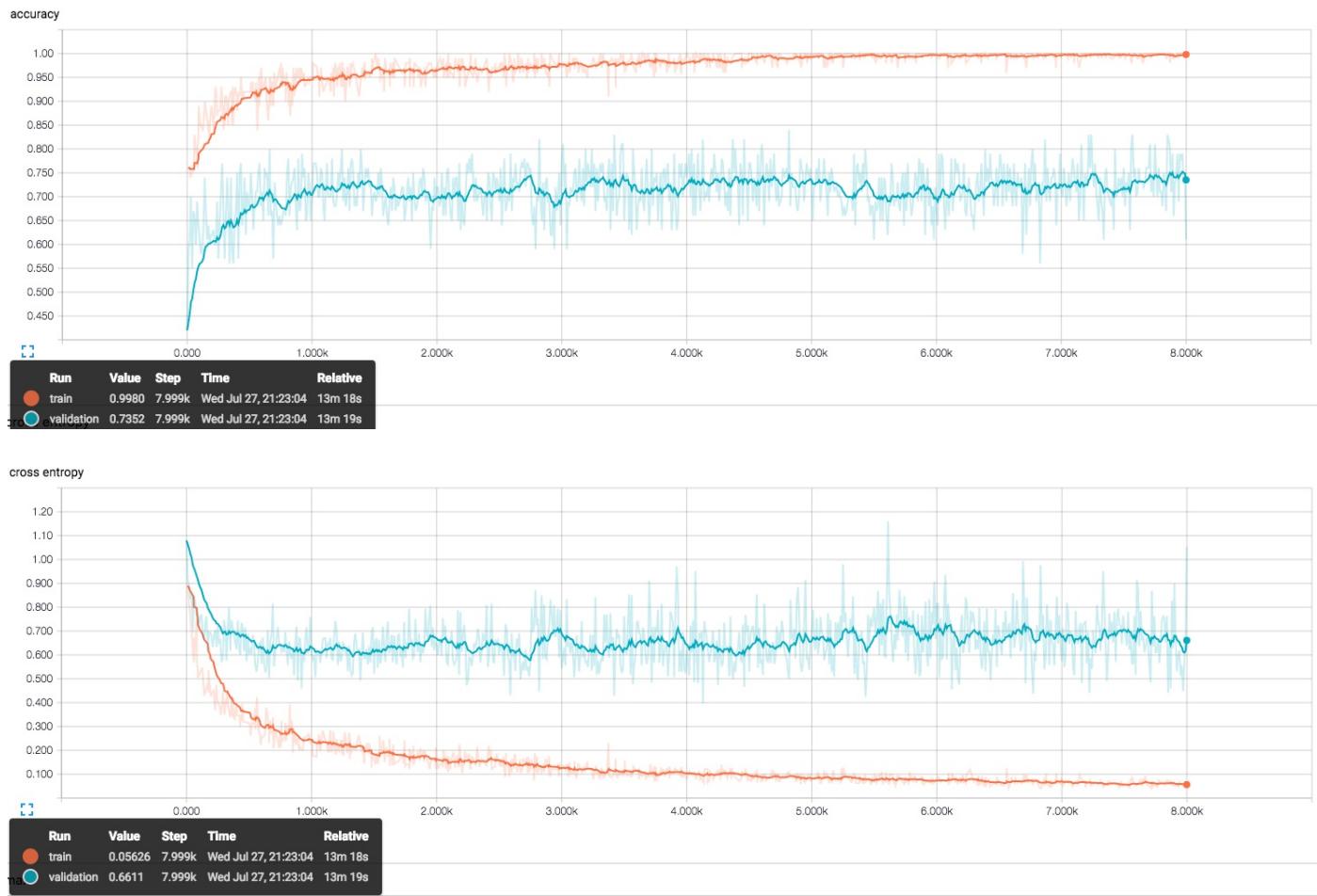
Splitting Data into Training, Validation, and Test sets

To reduce and mitigate overfitting, we split our images into 3 categories.

1. Training Data - 80% of the images are used for the model to learn from
2. Validation Data - 10% of the images are used for periodic "checking" of our model during training
3. Test Data - 10% of the images are used only once to predict real world results of our model against images it has never seen.

Training and Validation information can be viewed during training runs via the TensorBoard application and are shown throughout this report. The test accuracy is reported as a single percentage datapoint at the conclusion of a successful training run of the model.

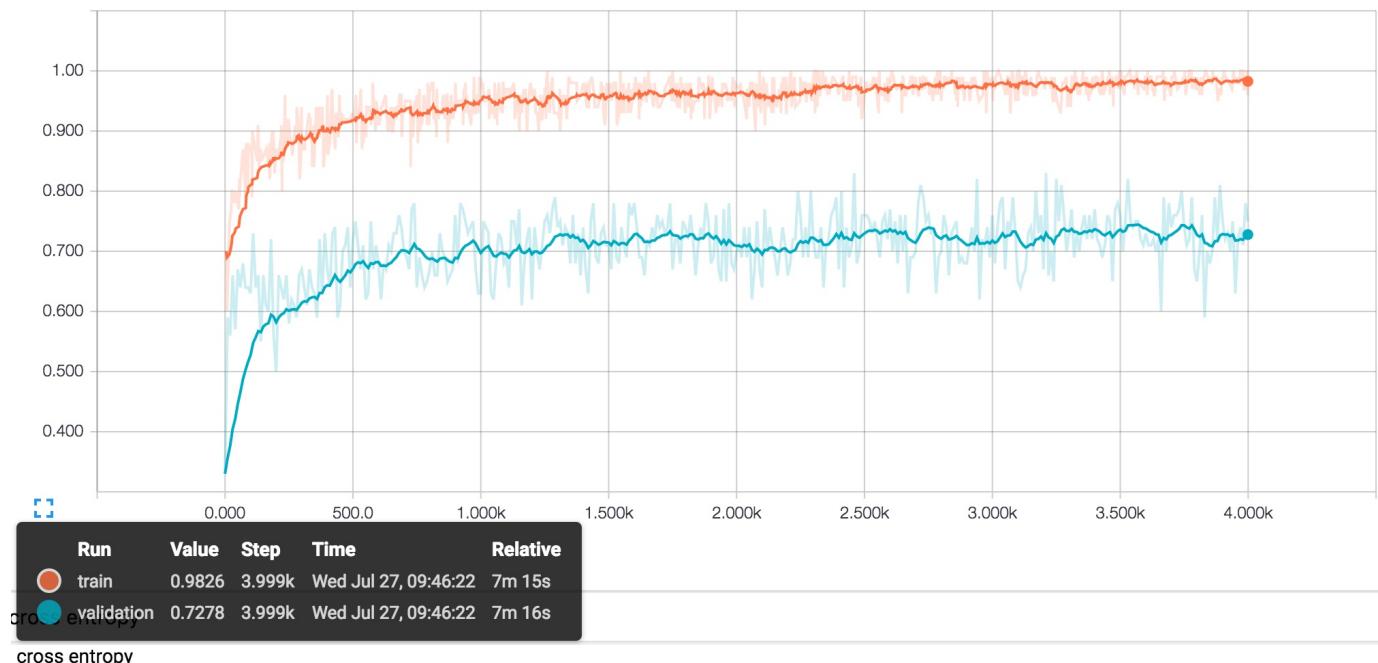
Overfitting, a model's "over-learning" of features in the training set.



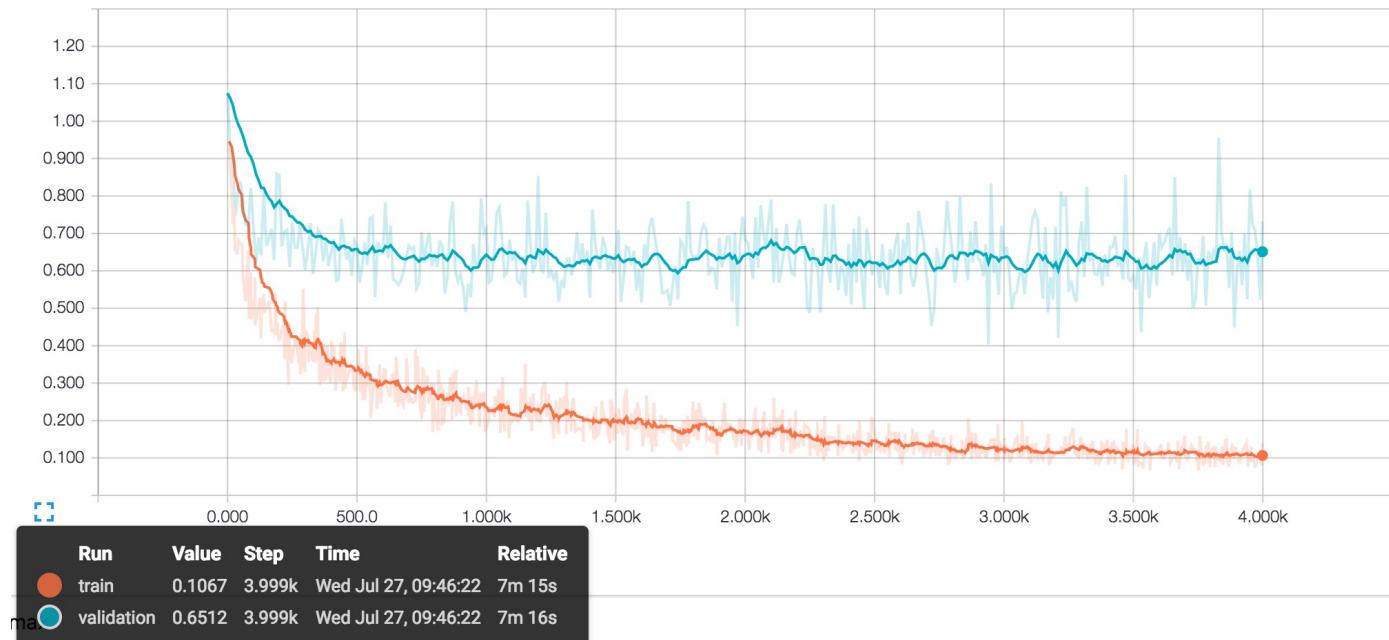
Benchmark

Running the initial CNN with default values as seen from [this tutorial](#) yields a result of 87.8% test accuracy on the simplest form of binary classification- only selecting Nike or Altra and the smaller our two datasets

accuracy



cross entropy



how does it do on the following?



altra (score = 0.98203) nike (score = 0.01797)



altra (score = 0.99800) nike (score = 0.00200)



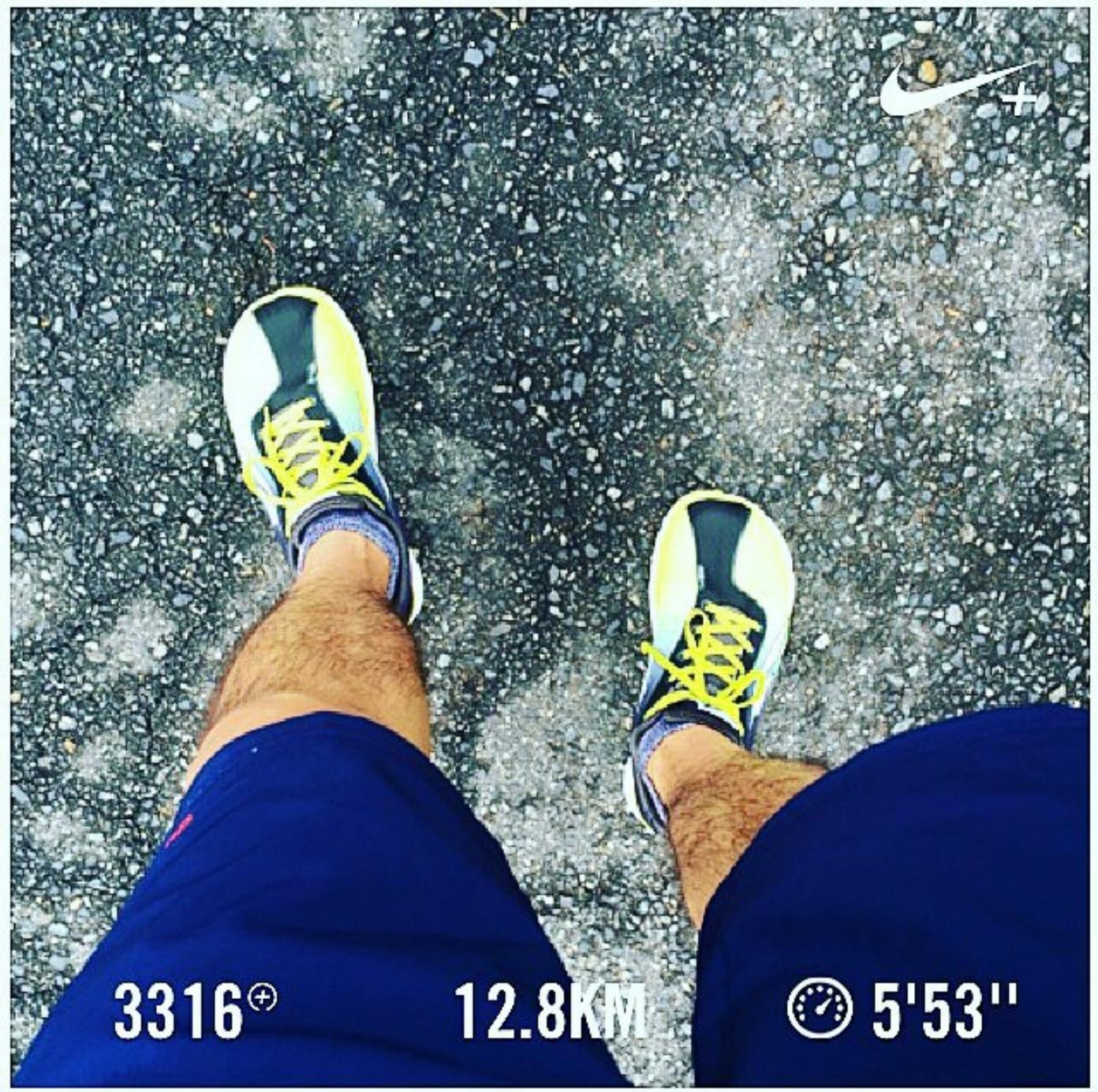
nike (score = 0.91221) altra (score = 0.08779)



nike (score = 0.98963) altra (score = 0.01037)



altra (score = 0.85372) nike (score = 0.14628)



altra (score = 0.96446) nike (score = 0.03554)



nike (score = 0.80665) altra (score = 0.19335)

III. Methodology

Data Preprocessing

Premodel Level - Image transformations

One way to improve my data set is to crop the images in a way that

highlights the features that I'm interested in extracting. Cropping the images of shoes so that the logo is the most prominent aspect of the image helps clearly define the features that need to be recognized in the images.

The following image provides an example- the original image contains 3 samples of the Nike logo. By training on the original image as well as the 2 cropped versions of just the logo increases the number of training images as well as our models' ability to differentiate the Nike logo in images.



Adobe Lightroom was used to load, crop, and export images in both the Altra and Nike categories.

Model Level

1. Images randomly split into training, validation, and testing sets
2. Resized so that the longest edge is 299 pixels wide/ tall
3. Reshaped into 299x299x3 array for Preprocessing

Implementation

In order to reconstruct these results locally, follow the TensorFlow [tutorial](#) on transfer learning and retraining. Once you have that setup, replace the existing `retrain.py` file with the version [here](#). [###todo github link for retrain.py] and the dataset available [here](#)[###todo link to dataset].

From there, the following steps will be performed:

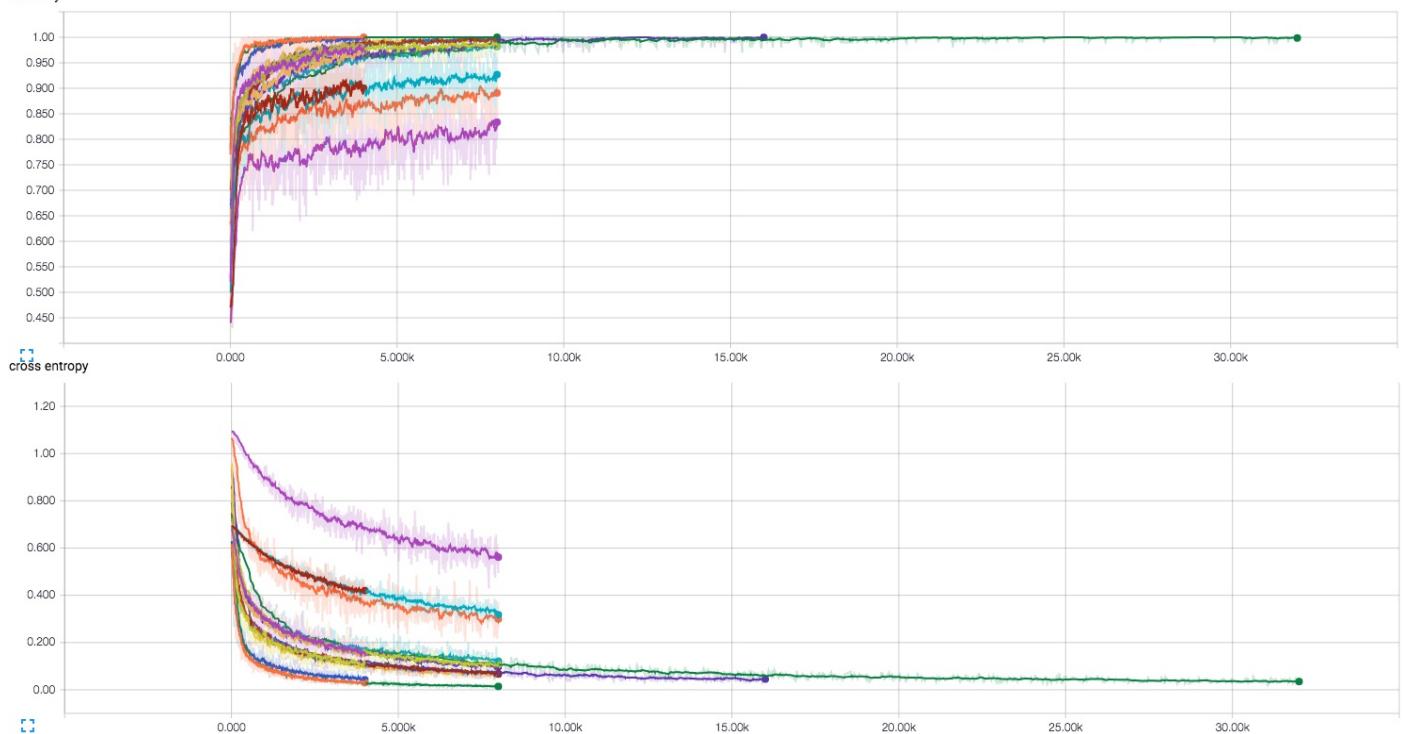
1. Construct the Inception v3 network in TensorFlow
2. Load pre-trained weights into the previously built network
3. Modify the pre-existing model for our classification tasks
4. Update the bottlenecks
5. Retrain the network on images of interest
6. Record & Evaluate model performance

Initial Results

18 runs total are shown below. Some underperforming trials are omitted for clarity and brevity.

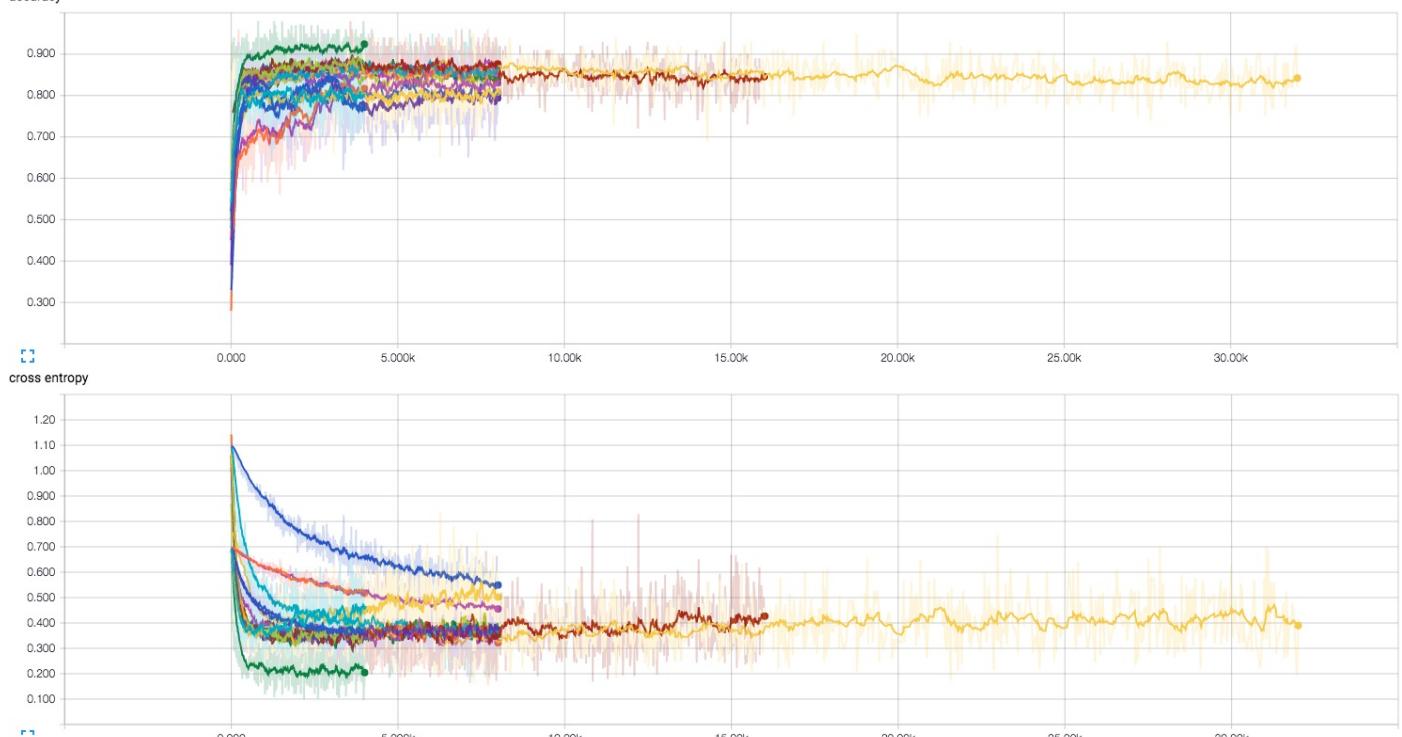
Training

accuracy



Validation

accuracy



<!-- In this section, the process for which metrics, algorithms, and techniques that you implemented for the given data will need to be clearly documented. It should be abundantly clear how the implementation was carried out, and discussion should be made regarding any complications that occurred during this process.
Questions to ask yourself when writing this section:

- Is it made clear how the algorithms and techniques were implemented with the given datasets or input data?
- Were there any complications with the original metrics or techniques that required changing prior to acquiring a solution?
- Was there any part of the coding process (e.g., writing complicated functions) that should be documented? -->

Refinement

Not just binary classification:

Importantly I also wanted to differentiate between the brands of interest and other shoes/ brands. I.E. I didn't want to categorize everything as either Nike or Altra, but rather be able to know that an image did not contain one of the logos I was tracking.

Additional images

Optimizer

- Gradient Descent
- Adaptive Gradient or Adagrad has an implementation in TensorFlow

Hyperparameter Search

- Learning Rate [###todo add math symbol here]
- Number of iterations
- With/ Without 'Neither' category
- Optimizer
 - Gradient Descent
 - Adagrad most implementations start with an accumulator of 0.01) or 0.1

IV. Results

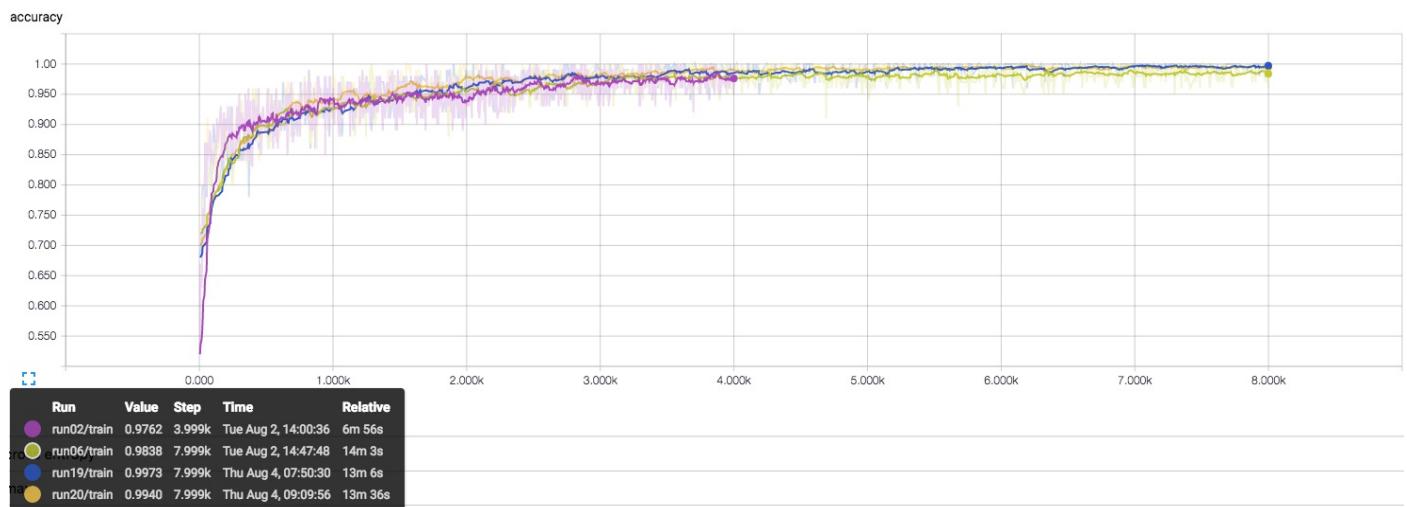
Model Evaluation and Validation

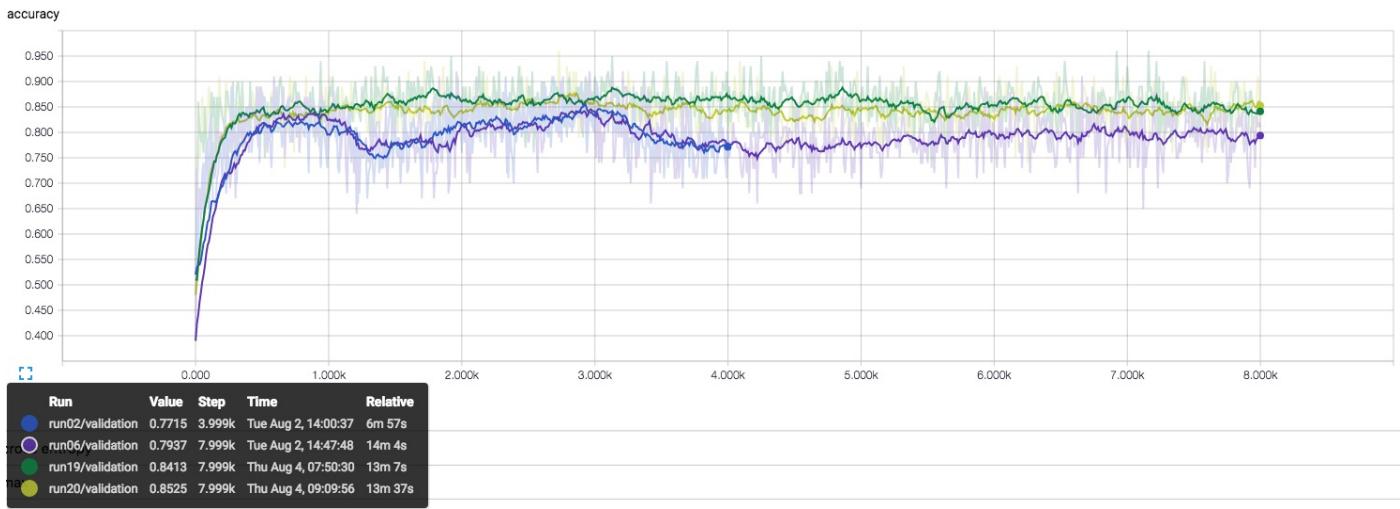
Approximately 20 different runs were conducted across a number of different hyperparameters.

Ranked by test accuracy, here are the top results.

Top Results					
Run #	Test Accuracy	Iterations	Learning Rate (η)	Data Set	Classification
6	97.4%	8,000	0.001	First	Binary
2	96.8%	4,000	0.001	First	Binary
20	88.6%	8,000	0.001	Second	Multiclass
19	87.4%	8,000	0.001	Second	Multiclass

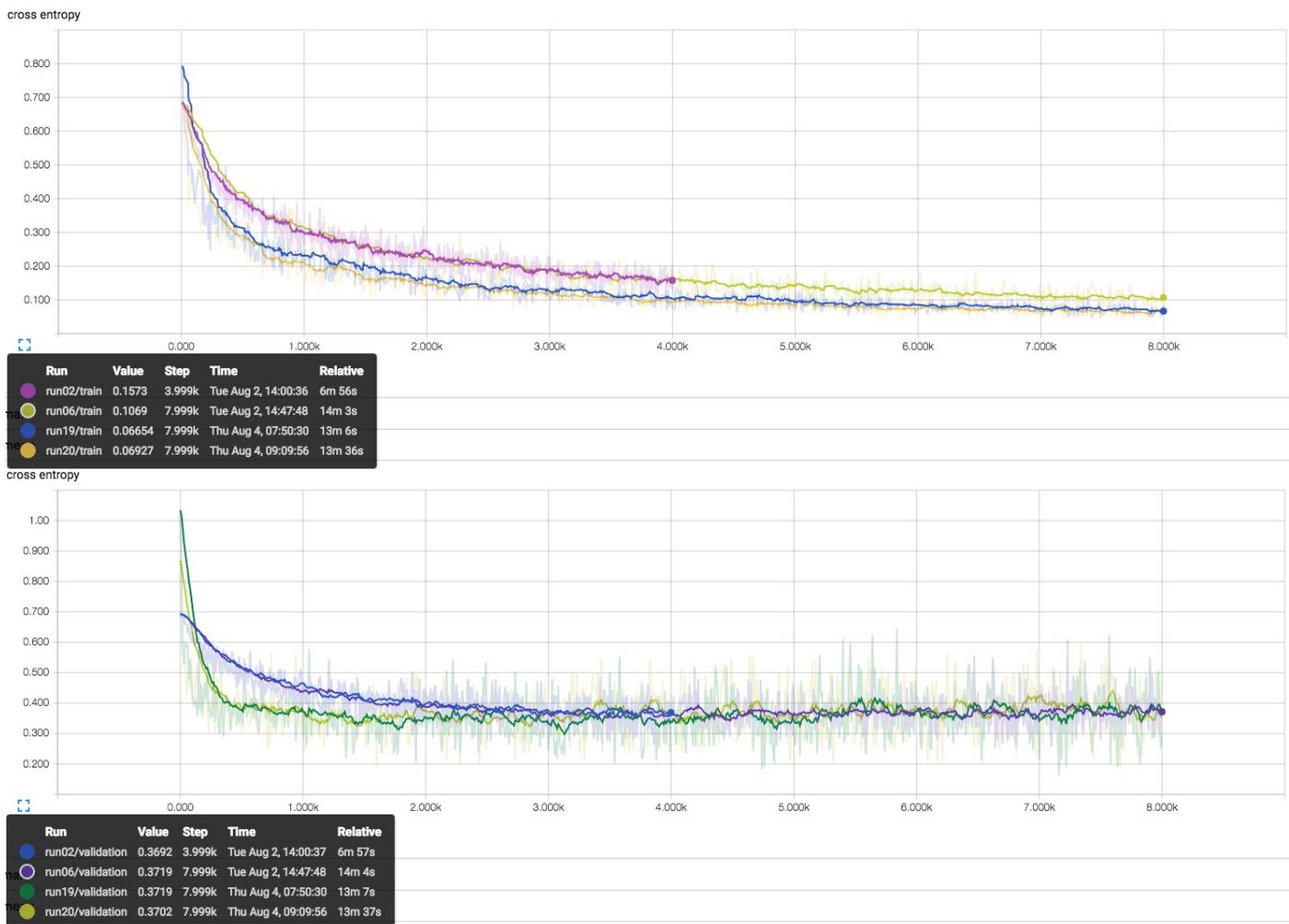
That only tells a portion of the story however. Comparing the training and validation scores can be instructive





This shows us that although we get superior results from these at least two of the models (# 2 and 4) are saturating/ no longer learning, etc.

We can also compare cross-entropy between models



From the above information, and selecting for the model that uses multiclass classification, this is the ideal set of hyperparameters for our model.

Top Results					
Run #	Test Accuracy	Iterations	Learning Rate (η)	Data Set	Classification
6	97.4%	8,000	0.001	First	

Here are the graphs of its performance compared to the previous top-runners. [###todo graph of *the best model*]

With this finished model, we are able to compare against some difficult images from earlier:

[###todo difficult images from earlier, previous classification and current]

[###todo Other sensitivity analysis?]

Justification

Our final model has the following form

[###todo simplified graphic of model]

Revisiting Difficult Photos

[###todo Need to re-copy output graph to update these once you settle on a model]



neither (score = 0.99816) nike (score = 0.00158) altra (score = 0.00025)



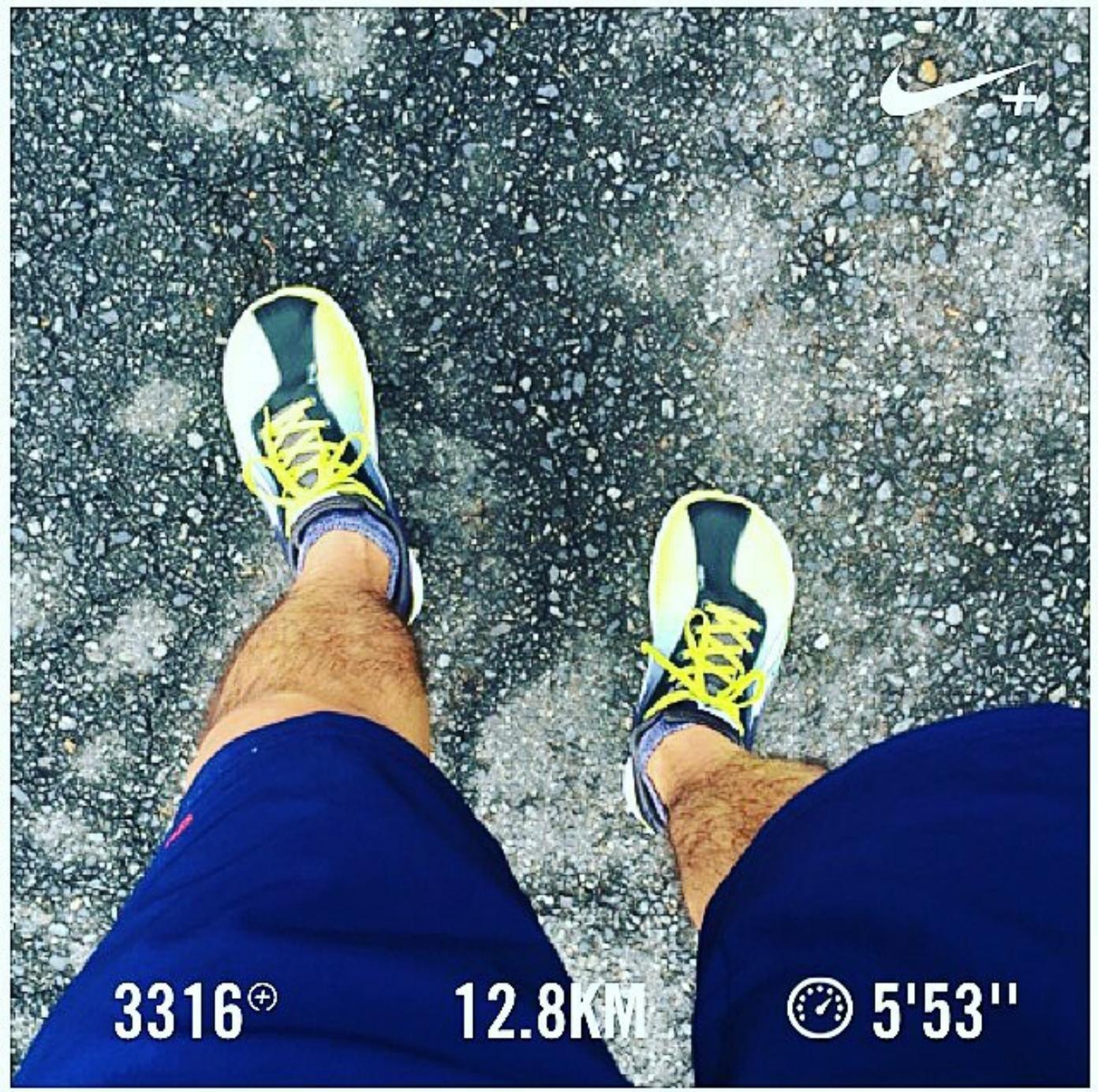
neither (score = 0.98079) altra (score = 0.01201) nike (score = 0.00720)



nike (score = 0.56717) neither (score = 0.42952) altra (score = 0.00331)



neither (score = 0.87823) nike (score = 0.10636) altra (score = 0.01541)



neither (score = 0.97332) altra (score = 0.02171) nike (score = 0.00497)



neither (score = 0.85841) nike (score = 0.13508) altra (score = 0.00651)

In this section, your model's final solution and its results should be compared to the benchmark you established earlier in the project using some type of statistical analysis. You should also justify whether these results and the solution are significant enough to have solved the problem posed in the project. Questions to ask yourself when writing this section:

- *Are the final results found stronger than the benchmark result*

reported earlier?

- *Have you thoroughly analyzed and discussed the final solution?*
- *Is the final solution significant enough to have solved the problem?*

V. Conclusion

(approx. 1-2 pages)

Free-Form Visualization

[###todo insert images of torn up shoes and my models performance identifying them]

In this section, you will need to provide some form of visualization that emphasizes an important quality about the project. It is much more free-form, but should reasonably support a significant result or characteristic about the problem that you want to discuss. Questions to ask yourself when writing this section:

- *Have you visualized a relevant or important quality about the problem, dataset, input data, or results?*
- *Is the visualization thoroughly analyzed and discussed?*
- *If a plot is provided, are the axes, title, and datum clearly defined?*

Reflection

Steps to create a solution:

1. Identify a suitable framework or library to build a pre-trained network
2. Construct an existing network
3. Load pre-trained weights into the previously built network
4. Modify the pre-existing model for our classification tasks

5. Update the bottlenecks
6. Retrain the network on my images of interest
7. Evaluate model performance
8. Optimize hyperparameters and compare performance
9. Select the optimum solution

There were a number of new, and now that they have been solved, interesting problems that I had to work through in the course of this project.

Interesting

Machine learning is unique in that the work that is being done at its highest levels- like that performed at Google or research institutions can has been released for public use. What other field allows for the almost immediate implementation of cutting/ bleeding edge technologies by a new practitioner? The idea of [democratizing](#) artificial intelligence is a call to action to its practitioners.

Applying what I have learned to a real world dataset proved immensely empowering and rewarding. Being able to use my coursework on things found "out in the wild" was a powerful way to implement things that I had studied on an actual problem and not just a "toy" dataset.

TensorFlow & Keras are both popular tools in the machine learning community right now and I'm glad I got the opportunity to apply them to a project and gain a better understanding of them.

Difficulties

TensorFlow and Keras were both new resources for me to work with, and both provided great functionality once I was able to learn and implement their features. Unfortunately they are not without a learning

curve and in some cases bugs. Why a retrained network Tracking down and understanding why TensorBoard would not display some of my logged training runs turned out to be an [interesting issue](#) with how paths are specified.

Visualizing neural networks proved to be one of the more difficult aspect of the project. There are a number of ways to Attempting to build Caffe for its abilities to create visualizations of different layers of a network was quite difficult. There are a number of dependencies and local customizations needed. While I will continue to sort through them, they ultimately proved too time consuming to utilize in this project.

expectations

My final model performs better than I expected. The accuracy of predictions

In this section, you will summarize the entire end-to-end problem solution and discuss one or two particular aspects of the project you found interesting or difficult. You are expected to reflect on the project as a whole to show that you have a firm understanding of the entire process employed in your work. Questions to ask yourself when writing this section:

- *Have you thoroughly summarized the entire process you used for this project?*
- *Were there any interesting aspects of the project?*
- *Were there any difficult aspects of the project?*
- *Does the final model and solution fit your expectations for the problem, and should it be used in a general setting to solve these types of problems?*

Improvement

- implement a web based interface to this model. Bonus point if you could provide additional information in the event an image was misclassified. Remotely using the model would be beneficial in lieu of downloading my Python code and running locally. Additionally, a web based implementation could be extended to mobile devices as well, negating the need for a mobile based version.
- TensorFlow can be run on a mobile device, extending this work to an iOS or Android device would allow the model to be run remotely.
- Comparing my results to another baseline, such as [PCANet](#) would be a good gut check of my results.
- *Were there algorithms or techniques you researched that you did not know how to implement, but would consider using if you knew how?*
- The process of removing and re-architecting the VGG16 architecture in Keras became unwieldy.
- Visualizing the outputs/ weights of different levels within the CNN
 - [like this](#)
 - [or this](#)
- Deeper [sensitivity analysis](#)
- The network could always be improved by adding additional training images.

Before submitting, ask yourself. . .

- Does the project report you've written follow a well-organized structure similar to that of the project template?
- Is each section (particularly **Analysis** and **Methodology**) written in a clear, concise and specific fashion? Are there any ambiguous terms or phrases that need clarification?
- Would the intended audience of your project be able to understand your analysis, methods, and results?

- Have you properly proof-read your project report to assure there are minimal grammatical and spelling mistakes?
- Are all the resources used for this project correctly cited and referenced?
- Is the code that implements your solution easily readable and properly commented?
- Does the code execute without error and produce results similar to those reported?