Chris Kaschner 2016-08-8

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 and not the actual image content.

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? This type of structured data is already available for text but what about images? 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 create a Convolutional Neural Network (CNN) 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 whether 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.

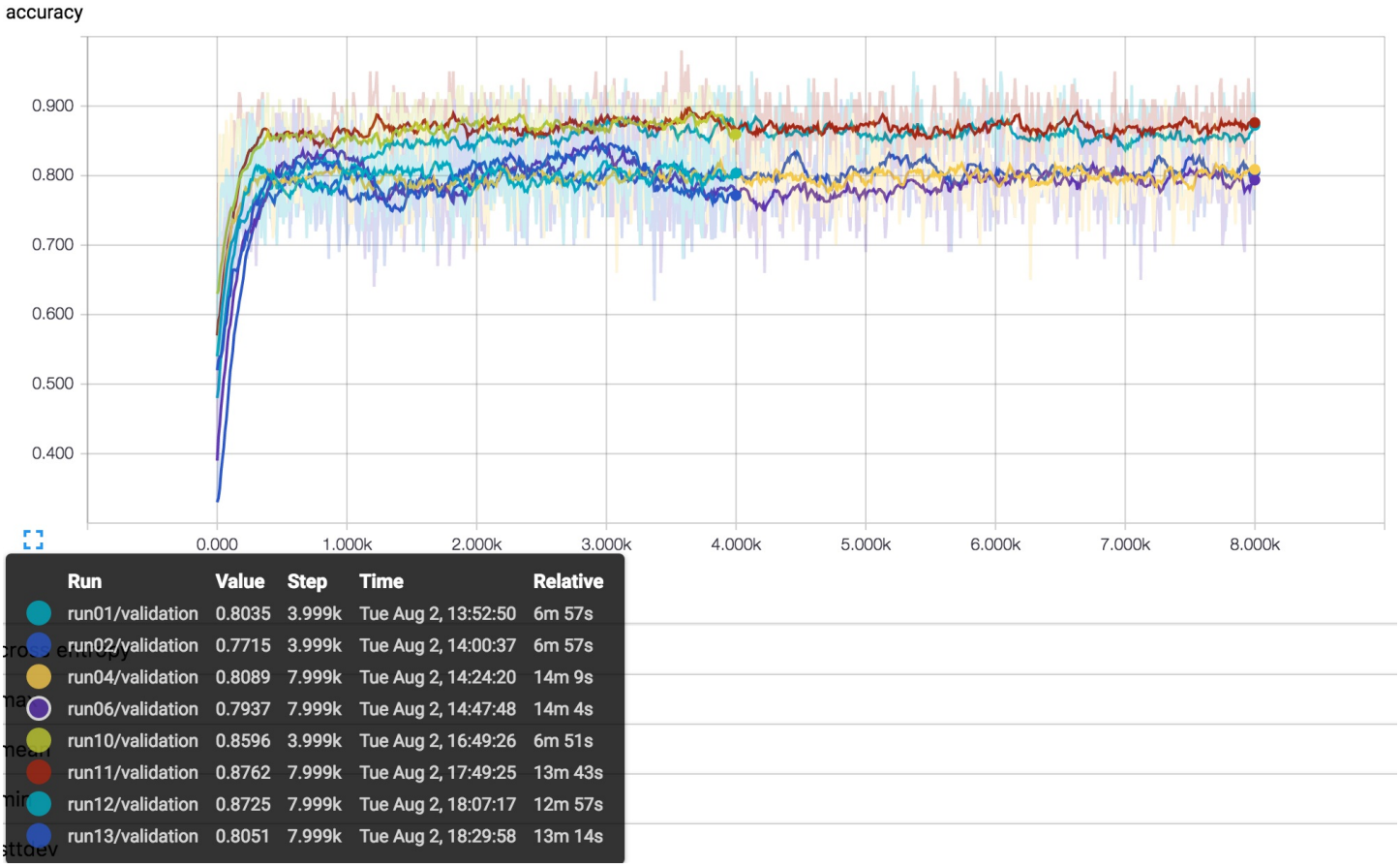
Metrics

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.

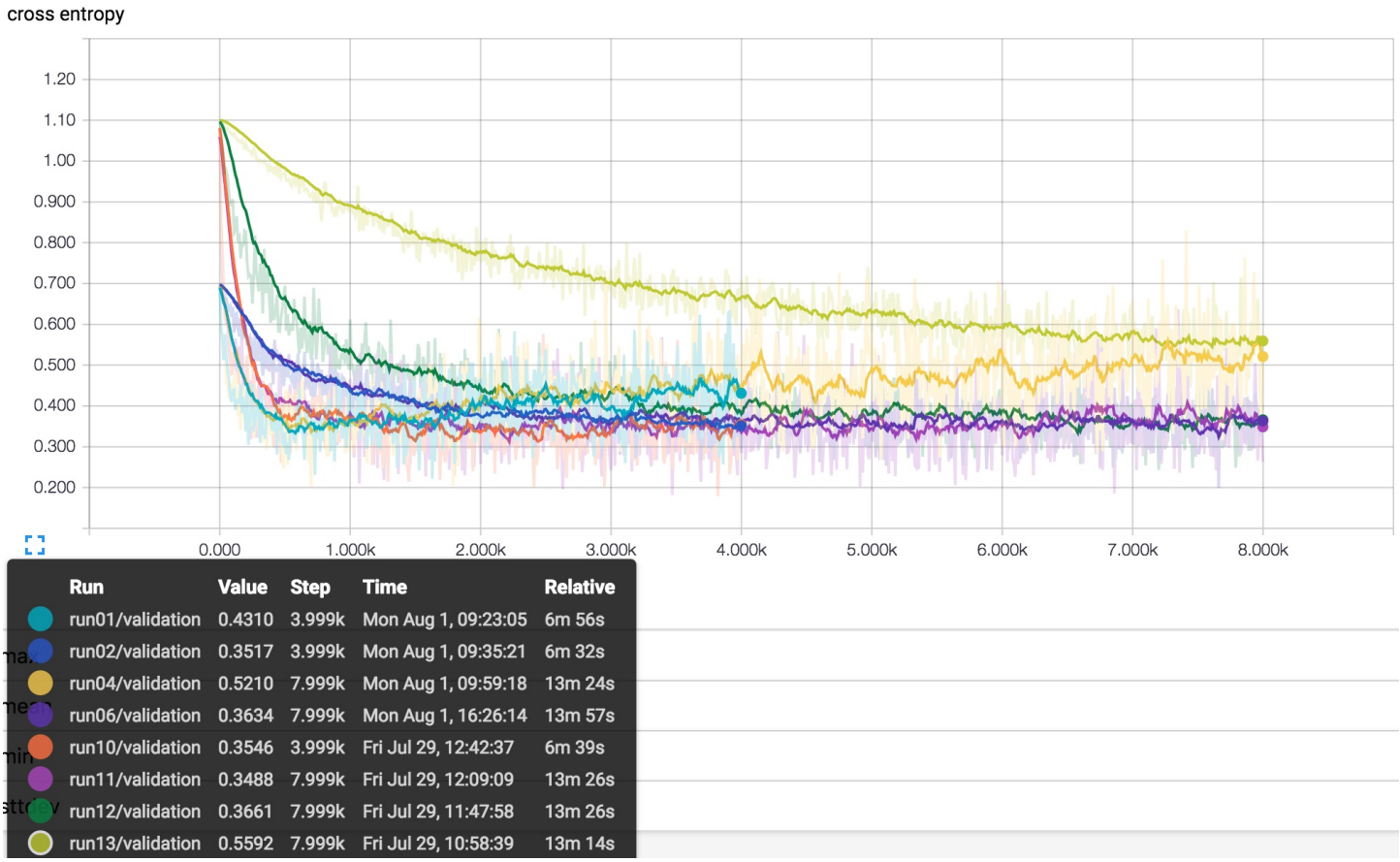
Accuracy should improve as our model iterates, as shown below:



Cross-Entropy

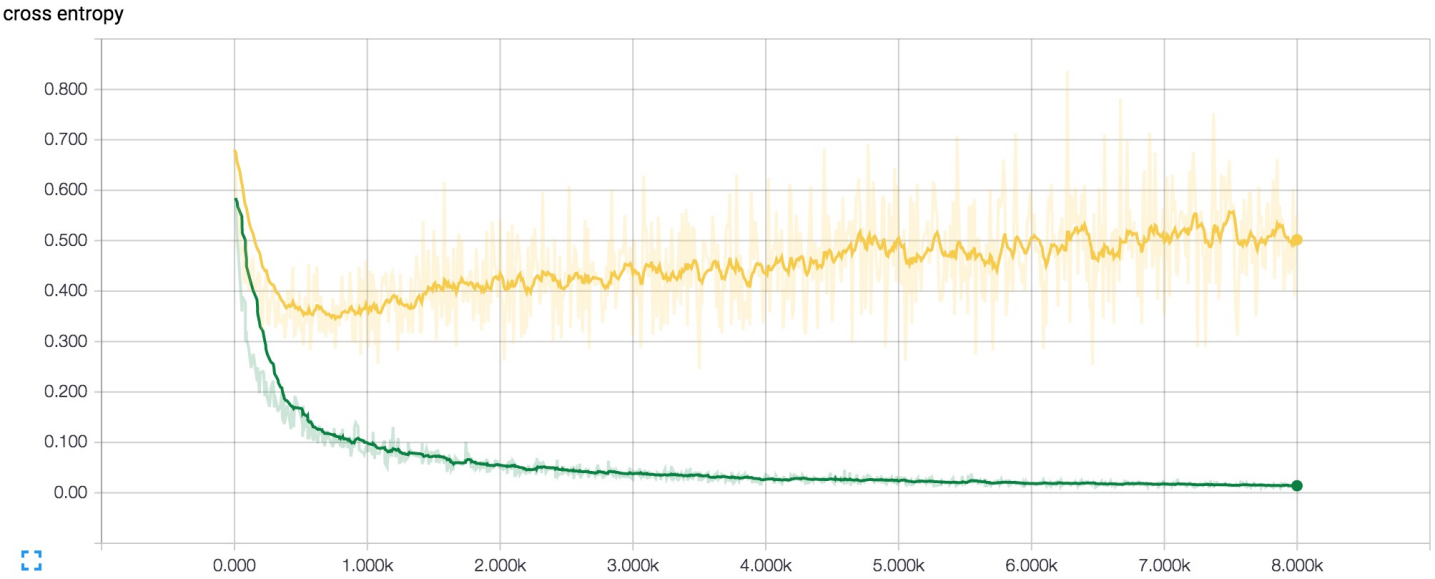
Cross-entropy is a measure of a model's loss or cost. A high value represents a model doing a poor job thus we seek to minimize the cross-entropy. A more in-depth discussion of cross-entropy is available here

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

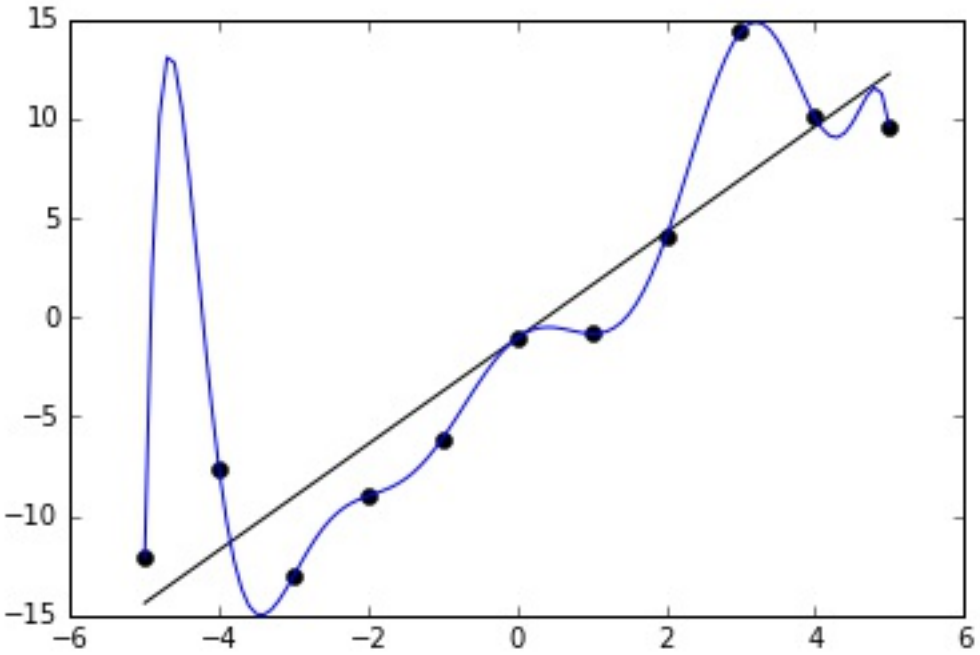


Run	Value	Step	Time	Relative
run01/validation	0.4310	3.999k	Mon Aug 1, 09:23:05	6m 56s
run02/validation	0.3517	3.999k	Mon Aug 1, 09:35:21	6m 32s
run04/validation	0.5210	7.999k	Mon Aug 1, 09:59:18	13m 24s
run06/validation	0.3634	7.999k	Mon Aug 1, 16:26:14	13m 57s
run10/validation	0.3546	3.999k	Fri Jul 29, 12:42:37	6m 39s
run11/validation	0.3488	7.999k	Fri Jul 29, 12:09:09	13m 26s
run12/validation	0.3661	7.999k	Fri Jul 29, 11:47:58	13m 26s
run13/validation	0.5592	7.999k	Fri Jul 29, 10:58:39	13m 14s

If cross-entropy begins diverging the model is overfitting and some changes are required. This can be seen in Run04:



Overfitting occurs when a model is describing randomness and/ or noise in a dataset instead of the underlying relationship.



In the above chart, the blue line represents an overly complex polynomial function that perfectly describes a linear relationship between the points.

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.

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.

Examples of Altra and Nike logos

First data set

310 images were downloaded from Instagram as square format .jpegs with an edge length of 640 pixels. They represented 200 images in the Nike category and 110 images in the Altra category.

Second data set

The second dataset was made up of the entirety of the first dataset supplemented by additional images in each category. In total it constituted 240 Altra, 260 Nike, and 453 'neither' images.

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 and their resolutions vary.

Examples of difficult images

Within the dataset there are a number of images that are difficult to classify. Here are some examples and explanations of those images in the dataset.

Logos tend to be relatively small in actual images

Logos can be occluded by other objects in the scene

Both brands of interest could be in the same photo

Logos can be morphed to match product designs

Multiple brands in a single image

Neither brand of interest is present

Algorithms and Techniques

"No one teaches a child how to see" and in a similar way, instead of defining algorithms to explicitly define every permutation of logo that we would potentially see, neural networks allow us to feed training images to a model and allow it to define for itself what features are critical to classifying our images.

Convolutional neural networks are a subtype of neural network that are extremely effective at image classification.

They approach, or occasionally exceed, human performance on similar benchmarks.

The method I used to approach this problem utilized the significant work that large companies and research institutions have invested into this problem in the form of reusable networks and architectures. Instead of creating a CNN from scratch I was able to select a suitable network from a number of freely available options

and for my project I have selected 2 different networks: VGG16 and Inception v3.

VGG16

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 the one used in this work is 16-layers.

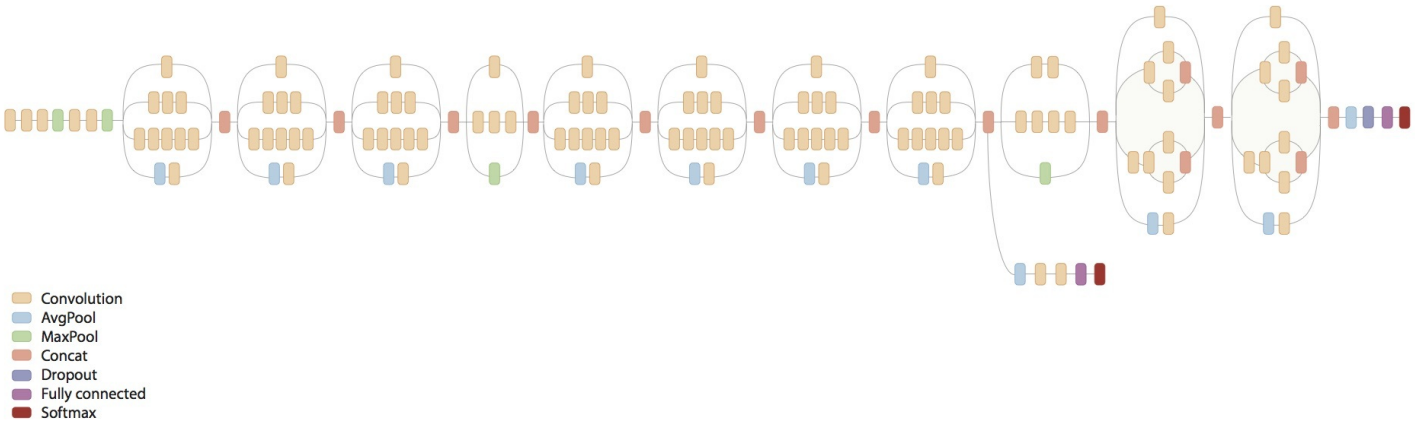
Inception

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

Training a large and complex CNN is a nontrivial task. For example, training the Inception v3 network which has over 27 million parameters, would take approximately 2 weeks to train on $50k worth of computer hardware and would need several *million* images.

Inception v3 network



By using transfer learning a model that was created at Google and trained on their enormous image library can be used for our image classification task of ~500 images total. This technique relies on the inherent ability of neural networks that have been trained on one classification task to be effective) at recognizing features in new tasks.

Replacing the final layer of a previously trained 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.

Process to implement

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

   There are a number of different options for building and 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 and 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

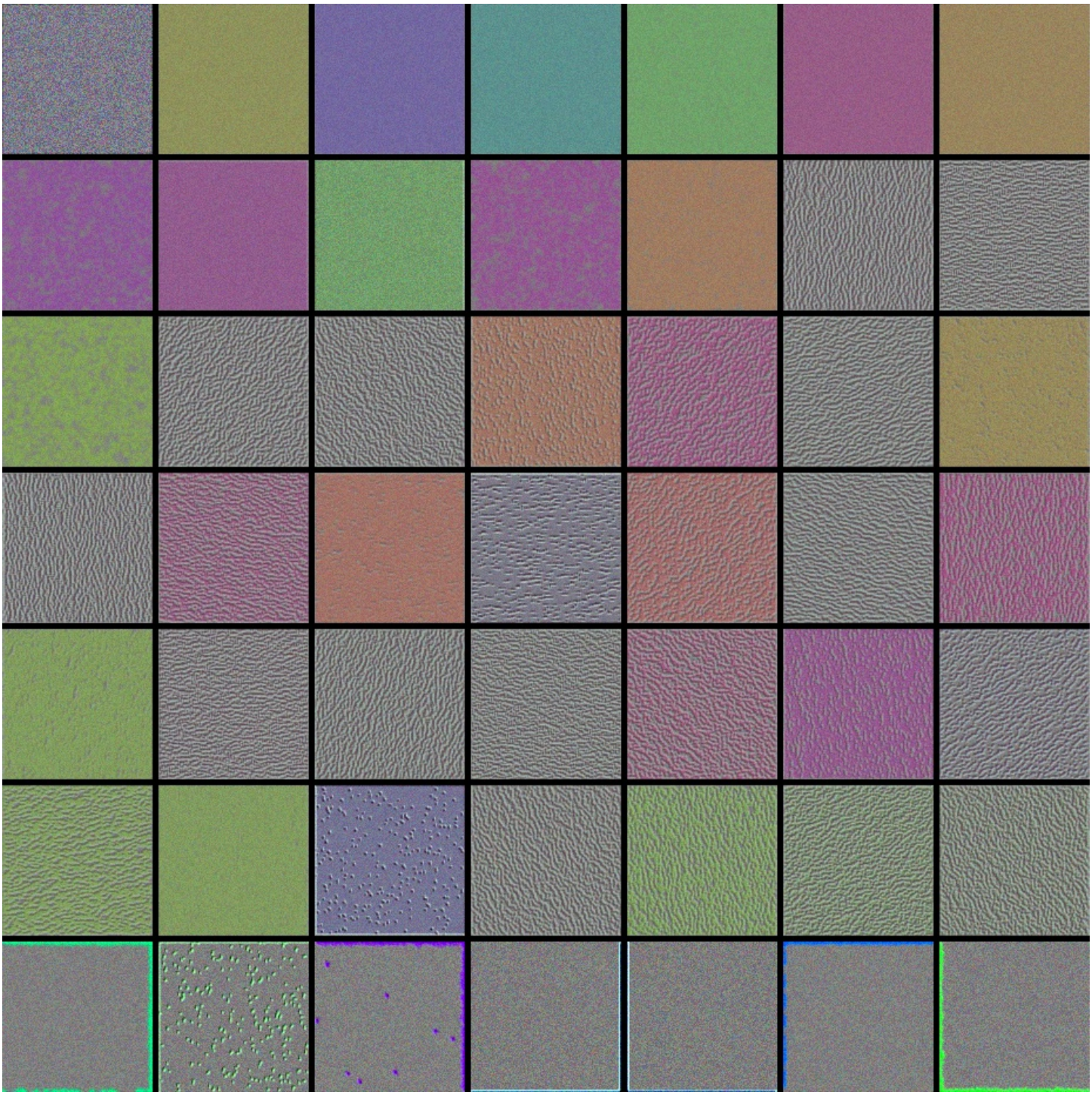
8. Optimize hyperparameters and compare performance

1. Select the optimum solution

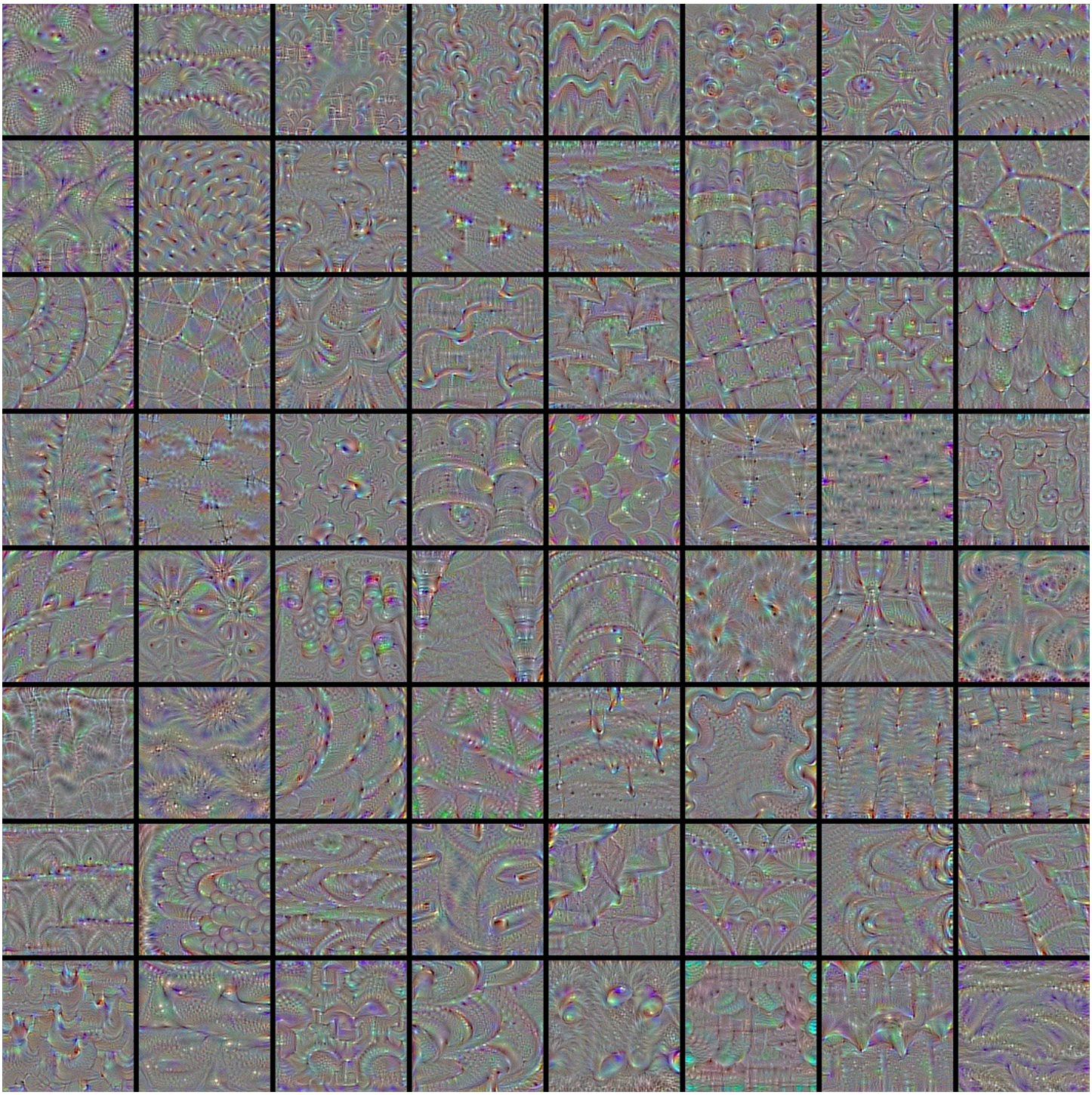
Exploratory Visualization

It is important to remember that computers do not "see" images in the same way as humans. One way to help us to understand how a CNN is "viewing" an image is to visualize the inputs that would maximize different layers of our network. Using the VGG16 model and the process described here we can see what a neural network would characterize as an ideal input.

From the first convolutional layer:



From the last convolutional layer:

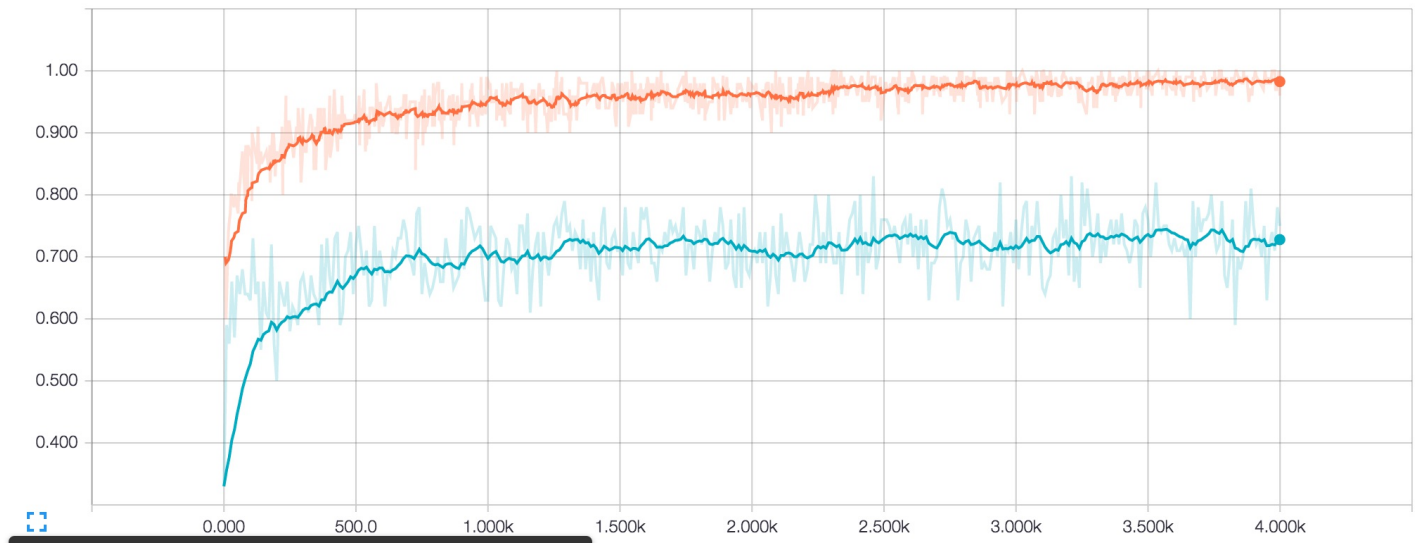


Examining different layers within our neural network, we can begin to appreciate how our network is building up features to identify objects. We can also start to understand what a network can "see".

Benchmark

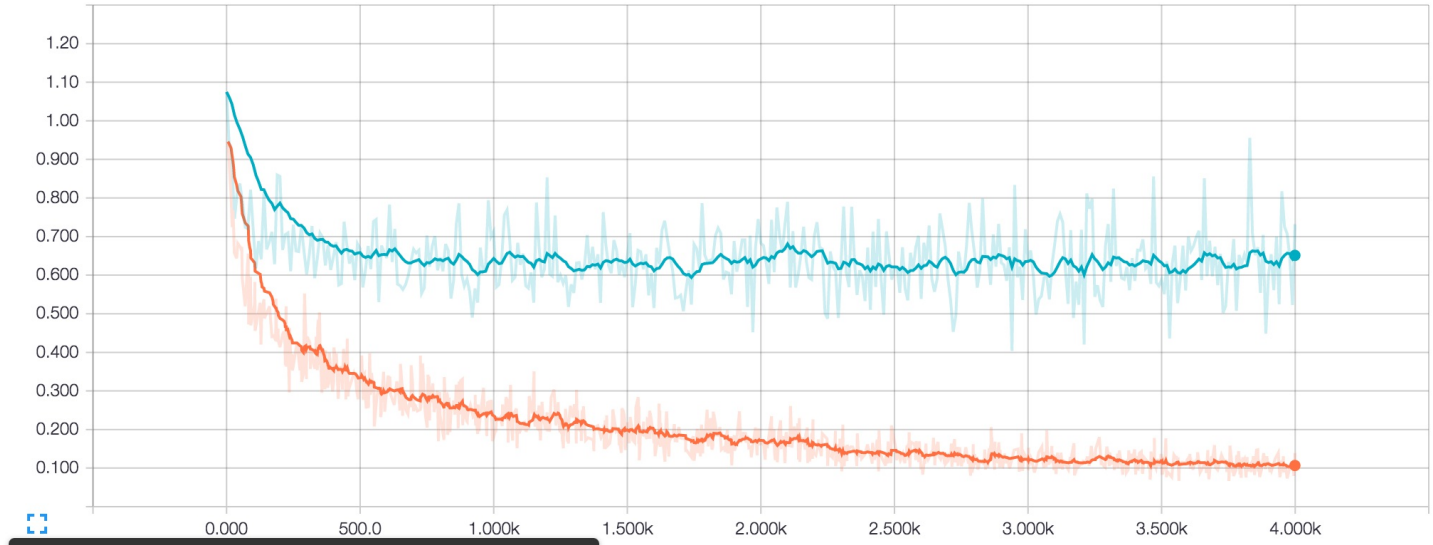
Running the initial CNN with default values as seen from this tutorial yields a result of 87.8% test accuracy on the simplest form of binary classification- only selecting Nike or Altra and the smaller our two datsets

accuracy



Run	Value	Step	Time	Relative
train	0.9826	3.999k	Wed Jul 27, 09:46:22	7m 15s
validation	0.7278	3.999k	Wed Jul 27, 09:46:22	7m 16s

cross entropy



Run	Value	Step	Time	Relative
train	0.1067	3.999k	Wed Jul 27, 09:46:22	7m 15s
validation	0.6512	3.999k	Wed Jul 27, 09:46:22	7m 16s

How does it do on the difficult images we identified previously?

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

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

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

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

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

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.

Original Image

Cropped Areas
#1 & #2 from Original

Adobe Lightroom was used to load, crop, and export images in both the Altra and Nike categories. The resulting images were used to supplement both datasets.

Model Level

The model itself handles data preprocessing steps including:

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

The following steps were performed to create the final model:

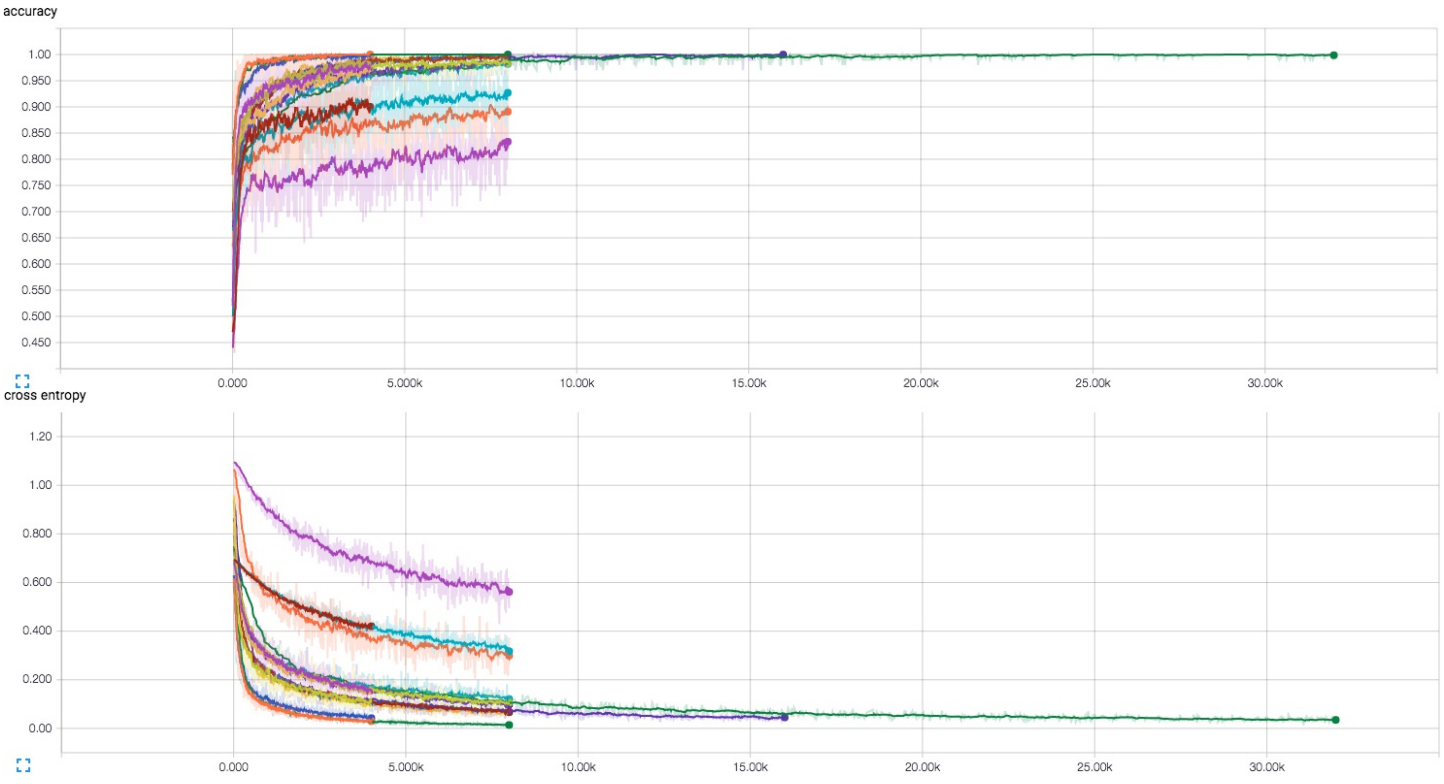
1. Construct the Inception v3 network in TensorFlow
2. Load pre-trained weights into the previously built network
3. Modify the pre-existing model for our classification tasks
4. Update the bottlenecks
5. Retrain the network on images of interest
6. Record & Evaluate model performance

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].

Initial Results

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

Training



Validation

Refinement

Our benchmark test accuracy of 87.8% only describes the case of binary classification. By adding the items below my model achieves a similar level of accuracy with a more complex multiclass classification problem set before it.

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. To implement, I added a third image category title "neither" comprised of a random selection of images that represented anythings that was not a Nike or Altra item.

Additional images

The most straightforaward way to improve results from a neural network is to increase the amount of data being used to train that network. The second dataset was added to increase the number of images in the training pool.

Hyperparameter Search

Once my initial network was functioning there are a number of different hyperparamters available that can be varied to potentially improve model performance. They include:

- Learning Rate (η)

    - How big of a step to take along the gradient of descent.
- Number of iterations
    - How many training steps to take.
- Optimizer Type
    - Stochastic Gradient Descent (SGD) - most common.
    - Adaptive Gradient or "Adagrad" that adapts the learning rate and utilizes an accumulator value (δ) that is typically between 0.01 and 0.1

IV. Results

Model Evaluation and Validation

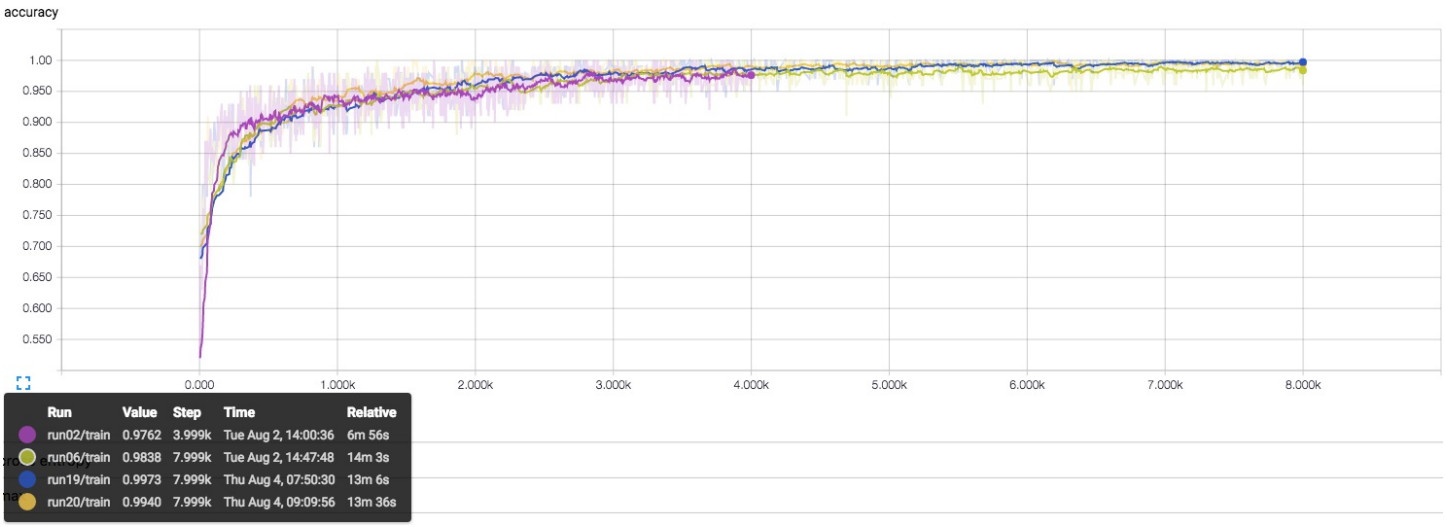
Approximately 20 different runs were conducted across a number of different hyperparameters.

Ranked by test accuracy, here are the top results.

Top Results					
Run #	Test Accuracy	Iterations	Learning Rate (η)	Data Set	Classificatio
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

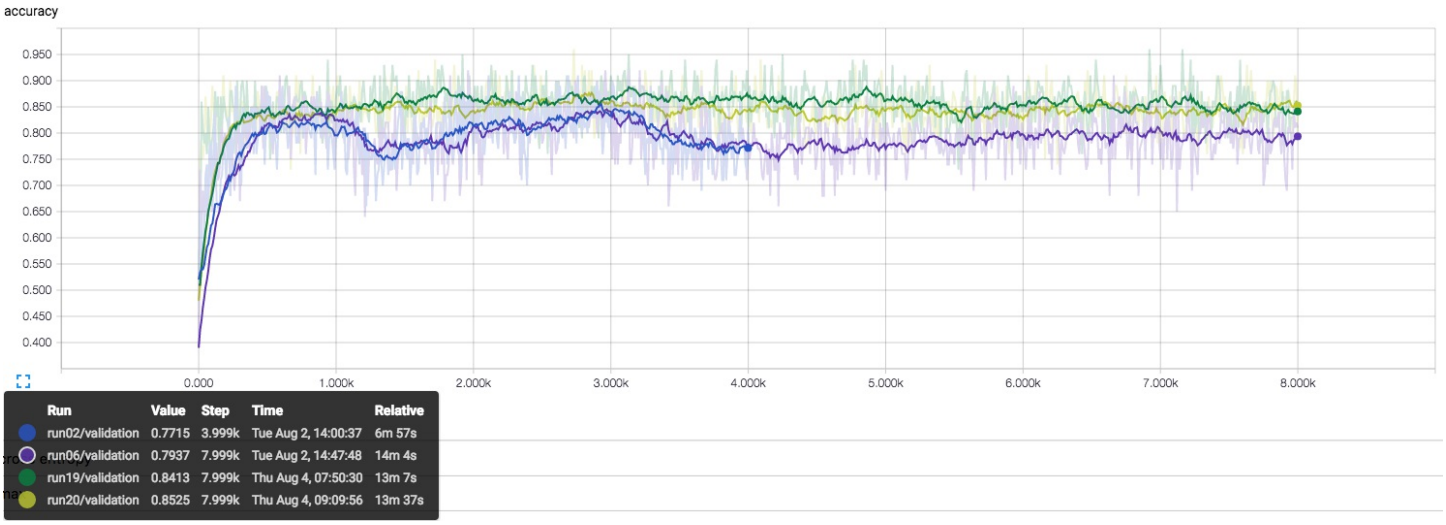
Comparing the training and validation scores amongst the top results:

Training Accuracy



Run	Value	Step	Time	Relative
run02/train	0.9762	3.999k	Tue Aug 2, 14:00:36	6m 56s
run06/train	0.9838	7.999k	Tue Aug 2, 14:47:48	14m 3s
run19/train	0.9973	7.999k	Thu Aug 4, 07:50:30	13m 6s
run20/train	0.9940	7.999k	Thu Aug 4, 09:09:56	13m 36s

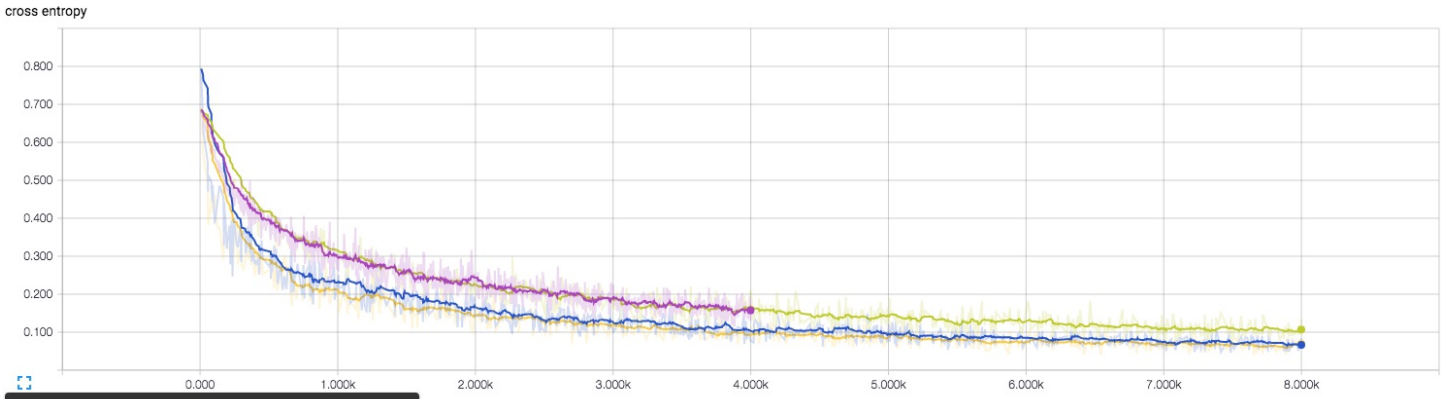
Validation Accuracy



Run	Value	Step	Time	Relative
run02/validation	0.7715	3.999k	Tue Aug 2, 14:00:37	6m 57s
run06/validation	0.7937	7.999k	Tue Aug 2, 14:47:48	14m 4s
run19/validation	0.8413	7.999k	Thu Aug 4, 07:50:30	13m 7s
run20/validation	0.8525	7.999k	Thu Aug 4, 09:09:56	13m 37s

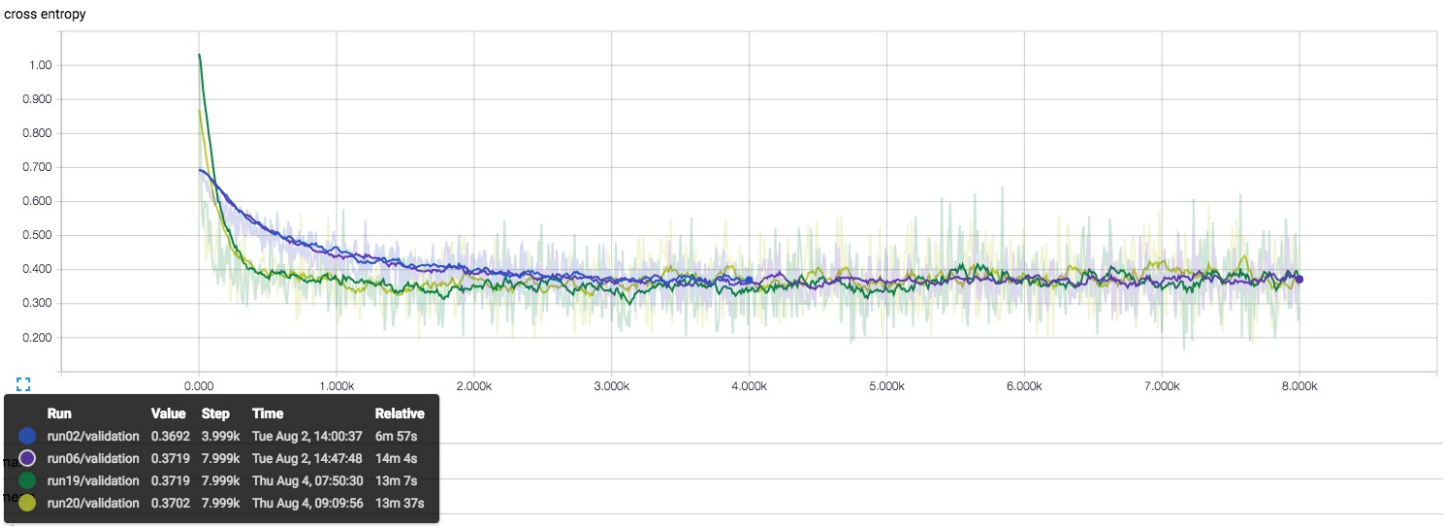
We can also compare cross-entropy between models

Training Cross-Entropy



Run	Value	Step	Time	Relative
run02/train	0.1573	3.999k	Tue Aug 2, 14:00:36	6m 56s
run06/train	0.1069	7.999k	Tue Aug 2, 14:47:48	14m 3s
run19/train	0.06654	7.999k	Thu Aug 4, 07:50:30	13m 6s
run20/train	0.06927	7.999k	Thu Aug 4, 09:09:56	13m 36s

Validation Cross-Entropy



Justification

Top Results Binary

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

Top Results Multiclass

Top Results					
Run #	Test Accuracy	Iterations	Learning Rate (η)	Data Set	Classificatio
20	88.6%	8,000	0.001	Second	Multiclass

Revisiting Difficult Photos

Despite improvements to our model, we can see from the below that there remains images that are difficult to

classify. Although these results do not show 100% accuracy, for my purposes I prefer that the model "fails" to the neither category (a false negative). Previously, in the binary classification model, the model would fail to one of the brands of interest (false positive).

Binary

altra (score = 0.88364)
nike (score = 0.11636)

altra (score = 0.97544)
nike (score = 0.02456)

altra (score = 0.90633)
nike (score = 0.09367)

nike (score = 0.93357)
altra (score = 0.06643)

nike (score = 0.51287)
altra (score = 0.48713)

nike (score = 0.66159)
altra (score = 0.33841)

Multiclass

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

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

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

neither (score = 0.49912)
nike (score = 0.49754)
altra (score = 0.00334)

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

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

V. Conclusion

Free-Form Visualization

We started with a discussion of how being able to identify problems with your products would permit a company to contact and resolve issues for customers. So how does our model perform on images of shoes that may require intervention?

neither (score = 0.58464)
nike (score = 0.23102)
altra (score = 0.18433)

altra (score = 0.92383)
nike (score = 0.07578)
neither (score = 0.00039)

neither (score = 0.55865)
nike (score = 0.35871)
altra (score = 0.08264)

nike (score = 0.87319)
neither (score = 0.11433)
altra (score = 0.01249)

nike (score = 0.52935)
altra (score = 0.46793)
neither (score = 0.00272)

neither (score = 0.74700)
nike (score = 0.25135)
altra (score = 0.00166)

From these results I would say that the model does a good job identifying the brands in the above photos and would provide the ability to find a customers image without any other information.

Reflection

Steps to create a solution:

1. Identify a suitable framework, TensorFlow, to build a pre-trained network
2. Construct the existing Inception v3 network
3. Create a dataset of images from Instagram
4. Load pre-trained weights into the previously built network
5. Modify the pre-existing model for our classification tasks
6. Update the bottlenecks
7. Retrain the network on images of interest

8. Evaluate model performance
9. Optimize hyperparameters and compare performance
10. 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 be 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. 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. 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. Fortunately the Keras documentation is excellent and provided an alternative method for visualizations.

expectations

My final model performs better than I expected. The accuracy of predictions was nearly 90% for multiclass classification which was beyond my expectation. Especially in light of the relatively small dataset provided for training. ~90% accuracy with ~500 images across 3 classes is an excellent result.

Improvement

- implement a web based interface to this model. Bonus point if you could provide additional information in

the event an image was mis-classified. 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.
- 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?