CS230

AcId: Aircraft Identification

Emilio Botero*
Department of Aeronautics and Astronautics
Stanford University
ebotero@stanford.edu

Abstract

In this work we classify images of common airliners by manufacturer. We do this by creating a dataset of nearly 100,000 images of various airplanes to learn the unique characteristics of the manufacturers. By applying transfer learning to VGG-16 network applied to ImageNet we are able to achieve a high degree of accuracy >98% compared to using a simple convolutional neural net.

1 Introduction

For the vast majority of travelers Aircraft Identification (AcId) is a rather trite subject. However, to those in the aviation industry it is a critical matter. Taxi instructions from controllers to pilots, ground crew handling, and even airline matters of branding are crucially linked to identifying the airframer of a particular airplane. The author will momentarily digress here and discuss a recent anecdote. A Boeing colleague was recently undergoing recurrent training when the training film erroneously contained Airbus imagery. This can be demoralizing especially when the topic is how their products are superior... how are they if the employees can't tell.

How does one tell which airplane is which if the employees at the manufacturers are prone to egregious errors? The answer is in the fine details much like a face recognition task. Although the overall configurations are the same the fine feature differ. Stack Exchange contains a guide in helping the novice discern these details (avi [2014]). Of the branding differences, we find things such as dorsal fin differences, nose shape, nacelle shaping, and more. To a trained plane spotter these are obvious.

Plane-spotting has become an ever-growing hobby with thousands of aviation photographers cataloging different aircraft as a game much like a scavenger hunt combined with the beauty of millions of dollars' worth of aluminum hurtling through the air. The crux of this project is to get a sufficiently large dataset. We leverage the plane spotters' community as it has many deep repositories of labeled images. In this project we use the jetphotos.com repository. Jetphotos has over 3 million cropped and labeled images.

2 Related work

Aircraft imagery in machine learning has been studied before, however not in the context as explained here. Common image libraries often contain images of aircraft, such as ImageNet (Deng et al. [2009]) which contain 44 images of all types of aircraft. However, few studies have done explicit classification. For example, studies have been conducted to identify whether an aircraft is within a satellite imageChen et al. [2013].

*Student CS230 Winter 2018

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

A fairly recent study used an SVM to classify airplanes, but had a poor accuracy of 48.7% (Maji et al. [2013]). One issue with that study was that it was used to identify specific models. Therefore knowledge similar gained from a specific manufacturer cannot be used to help inform decisions. Furthermore, their training set was limited to only 10,000 images.

In this work we go beyond these studies to examine a high variance problem with many images to learn from.

3 Dataset and Features

Obtaining imagery from jetphotos is not trivial. In fact, it may be the most challenging aspect of the project. To obtain the imagery a python code was made using Scrapy (Myers and McGuffee [2015]). This package handles all of the http protocol parts of the process. In the setup process, one cannot just download at will all the photos desired. Unfortunately, that would overload the server and believe it to think it was undergoing a DDOS attack. Therefore, a small delay was put into place. It took about a day of experimenting to find the setting required that doesn't take too long but doesn't make the web-server stop responding.

By downloading the thumbnails of the images, rather than the full images we can both tractably download all images and train a network. The thumbnails are 400x225 pixels and RGB channels .jpgs. After scraping the data set contains almost 50,000 images of Airbus aircraft and an additional 50,000 images of Boeing aircraft we have a full dataset. Each takes over 36 hours to obtain. Examples of typical images can be seen in Figures 1 and 2.

Figure 1: Airbus Product (jet)

Figure 2: Boeing Product (jet)

While we have a large binary labeled dataset. We're not quite done. The largest files and the smallest files must be removed. The reason being is that the large files contain cityscapes with an airplane as a background, and the small files contain blue sky with a faint silhouette of an airplane. Neither of which is sufficient for even a trained spotter to classify. Examples of small image can be seen in Figure 3 as well as large data images in Figure 4.

Once complete we have a large, yet high variance dataset that makes this challenging. All photos are taken at various angles, and each manufacturer has a large product catalog that looks significantly different, yet similar.

Figure 3: Small Data Image Example (jet)

Figure 4: Large Data Image Example (jet)

4 Methods

For this project two different network architectures were used. The first was the starter code provided. The reason to start with that code was used mostly as a proof of concept as well as used as a reference point. Further discussion of the results comparing both methods is in the following section.

4.1 Simple CNN

This code was the starter code provided for the class given in Python Tensorflow (Abadi et al. [2015]). This was modified to use the native resolution of 400x225 pixels from the basic 64x64 pixels given. Further it was modified to become a binary classification from numerous labels.

The basic structure of this model consisted of 16 channels in one block containing conv->batch norm->relu->maxpool->dense->dense->softmax. The loss function is softmax cross entropy. The weights were initialized from a Xavier initializer.

This network was relatively inexpensive to run, and could be run on a laptop overnight.

4.2 VGG-16 with Transfer Learning

For this final algorithm we use the VGG-16 algorithm (Simonyan and Zisserman [2014]) trained on ImageNet weights (Deng et al. [2009]). This was coded using Keras Chollet et al. [2015]. Using a guide on transfer learning in Keras as a starting point Jay [2017] we tuned and trained the network.

Significant deviations from the guide were made. Due to the ease of prototyping the code was executed on using Google Colabs. However, using Google Colabs restricts the use to short running scripts. Therefore, speed of convergence was a prime factor. This weighed into the choice of layers to train, the optimizer used, and the hyperparameter tuning as discussed in the following section. One may wonder if given the resources of the course, putting oneself in a resource constrained situation makes sense? I contend that resources are always restricted, even within academia, solving problems in a time conscience manner is absolutely important.

Therefore, in performing transfer learning, many layers were held frozen. The first 14 layers were held the same, including all convolutional filters. However, the dense fully connected layers were able to be modified. Two additional fully connected layers of 1024 hidden units, with dropout were added. A final softmax is then applied at the end. So now the optimizer just has to learn the weights associated with the final 5 dense layers. Again this was done to reduce the time retraining the network. The loss function was a softmax cross entropy. In this case as a binary classification problem we could have used a sigmoid, but it was built for the potential of expanding the use.

5 Experiments/Results/Discussion

5.1 Simple CNN

Here we initially trained the Simple CNN on a subset of the data. We train on a set of 10,000 images with a dev set of 300 images. This was done to see how well the data generalizes. One of the hardest questions in deep learning is how much data do I need? This allows us to answer this question as well as see how this type of architecture performs.

We can see the convergence plots for both loss and accuracy in Figures 5 and 6. The network quickly converges to low loss in 10 epochs. Further training was not performed as the loss was already quite low. When looking at both the loss and accuracy we see that it has reached a quite low value. This indicates that it is overfit to the training set. However, the results are far better than the accuracy expected by a literature review. Because of the computational time associated with running this problem we now look at running using the second option, transfer learning.

5.2 VGG-16 with Transfer Learning

Now we pivot and look at the results of training the VGG-16 network as described prior. Because this network is slightly larger, we try to save time by running downscaled samples of vehicles. We use human intuition to judge how much downscaling can be performed. Figures 7, 8, and 9 show the

Figure 5: Simple CNN Loss

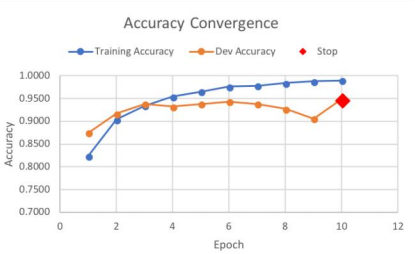

Figure 6: Simple CNN Accuracy

original resolution, a half scale, and quarter scale. The quarter scale is too coarse for a human, and was tried with the algorithm and could not converge. The half scale is the goldilocks zone, it has a quarter of the parameters yet it is still readable by a human. This is used for the remainder of this project.

Figure 7: 400 Pixel Width

Figure 8: 200 Pixel Width

Figure 9: 100 Pixel Width

We expand the data available to the VGG-16 algorithm. We obtain a full set of 100,000 images for training including 509 dev set images. This takes significantly longer to train.

At first the Adam optimizer was used, however, after two Epochs little progress was made. Instead Stochastic Gradient Descent was used. The learning rate was 0.01, other learning rates were tried, but this seemed to work the best at running the first Epoch.

We can see the convergence plots for both loss and accuracy in Figures 10 and 11 on the VGG network. The loss function quickly decreases at first and slowly plateaus. We run until the dev set starts to diverge significantly. By running epochs sequentially and saving regularly we are able to get around Google Colabs computational limitations imposed. While the loss function is not quite as low (more samples), the accuracy is excellent.

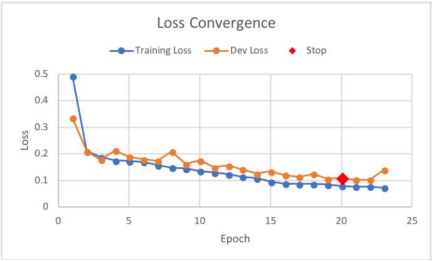

Figure 10: VGG-16 Loss

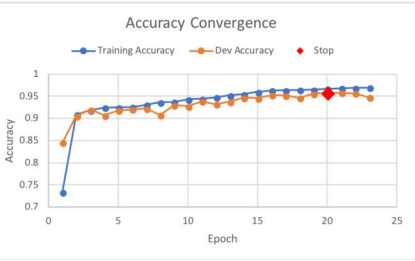

Figure 11: VGG-16 Accuracy

We can compare the accuracy of both networks on a held out test set of 568 images, where roughly half and half are of their respective category. We found that the Simple CNN has an accuracy of 80.3%. However the VGG-16 network has an accuracy of over 98%. A further breakdown of the accuracy is shown in 1. We see the algorithm even out performs the dev set. These sets are chosen at random, so no real reason why this is the case can be determined as both sets were also checked to ensure the contained images didn't have any spurious images.

Table 1: Accuracy of VGG-16 on Test Set

	Accuracy
Airbus	99.30%
Boeing	97.17%
Overall	**98.24%**

6 Conclusion/Future Work

In this report we create convolutional neural networks to perform binary image classification on two of the most popular aircraft manufacturers. By scraping 100,000 images we were able to train two separate neural networks to perform the task. By performing transfer learning from VGG-16 trained on ImageNet we obtained an accuracy of the test set of over 98%.

Tweaking hyperparameters allowed us to quickly obtain such an accurate solution. With further work we can expand this to classify further manufacturers. Furthermore, the approach attempted in this report allows one to classify airplanes that have yet to appear. We could expand this to match the model of aircraft as well. Given further computational resources we could easily train a robust and accurate classifier

One may wonder what the value of classifying a yet to exist aircraft is. The first reason may be for competitive analysis. The second reason may to allow designers to continue their subtle branding. For example, the images shown in Figures 12 and 13 show conceptual designs published by their respective manufacturer. When analyzed in our tool we find that the first concept is judged to be an Airbus correctly with a confidence of 100%! However, the second concept is incorrectly judged to be an Airbus with a confidence of 99%. The trained eye can see why after a close examination, the nose profile is that of an Airbus. This indicates that even the artistic designer follied and broke from typical nose profile to that of the competitor. Finally, maybe my colleague at Boeing can use this automatically to check their publications....

Figure 12: Future Airbus Concept

Figure 13: Future Boeing Concept

7 Contributions

This project was performed solo.

References

How can i tell apart an airbus from a boeing?, Mar 2014. URL `https://aviation.stackexchange.com/questions/1436/how-can-i-tell-apart-an-airbus-from-a-boeing`.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

Xueyun Chen, Shiming Xiang, Cheng-Lin Liu, and Chun-Hong Pan. Aircraft detection by deep belief nets. In *Pattern Recognition (ACPR), 2013 2nd IAPR Asian Conference on*, pages 54–58. IEEE, 2013.

Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013.

Daniel Myers and James W McGuffee. Choosing scrapy. *Journal of Computing Sciences in Colleges*, 31(1):83–89, 2015.

Aviation photos - 3 million on jetphotos. URL `https://www.jetphotos.com/`.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

François Chollet et al. Keras. `https://keras.io`, 2015.

Prakash Jay. Transfer learning using keras – prakash jay – medium, Apr 2017. URL `https://medium.com/@14prakash/transfer-learning-using-keras-d804b2e04ef8`.