Guilherme Reis March 22, 2018

Convolutional Neural Networks for Aircraft Model Identification

Abstract

"Plane-spotting" is the hobby of visually identifying aircraft makes and models, but can be very challenging to newcomers. We created a new 700GB, high-resolution dataset with high-fidelity labels through web scraping and built a Convolutional Neural Network capable of identifying aircraft make and model with 82% accuracy, compared to 24% for guessing the most common class, 88% for a human expert, and 8% estimated Bayes' error. Bias/variance analysis shows the network generalizes very well, performing stably around these values across all 22 target classes (even those underrepresented in the dataset) having trained for under 5 epochs.

Introduction

Different aircraft models can be distinguished due to visual characteristics such as aircraft size, wing span, presence of winglets, and shape of the aircraft body; furthermore, a sophisticated observer may exploit additional information, such as prior knowledge that a given airline only flies a particular type of aircraft. Crucially, airplanes don't significantly bend or deform, so given data from a wide variety of angles and sizes, this should theoretically be a great real-world distribution on which to exercise a convolution-based model. The motivation behind this project is to not only build a useful tool for plane-spotting, but also to attempt to understand how the network is learning, for instance by visualizing convolutional filters.

We scraped about 700GB of high-resolution images from popular plane-spotting websites, isolating their labels which are quite high-fidelity since users are experts, and data can be amended by the community. Building infrastructure capable of scraping, storing, and preprocessing tens of millions of images given limited time and computation resources was a significant challenge; nonetheless, the scraping infrastructure worked out excellently. We determined the median resolution of the images (812 x 1200) in the dataset to avoid excessive deformation, and scaled all images to this resolution with interpolation divided by a factor (6 for earlier iterations, and 4 for the final models to allow a larger input volume).

The input to our model is an image; the output is one of 22 target classes (an integer mapped to e.g. "Airbus_A380"). We built a Keras pipeline to shuffle and feed scraped images to the CNN and evaluate results. The optimizing metric is accuracy, given the nature of this specific classification task.

Related work

Unfortunately, there is little to no prior work on this specific problem. This is likely because it is a fairly niche hobby with no obvious commercial application. However, we did find some literature on aircraft identification for military purposes. For example, Hwang et al. (2018) developed a CNN for aircraft detection from top-down imagery produced by unmanned drones. Although the application is similar in goal, the input distribution is drastically different, as Hwang is limited to top-down aerial imagery. Nonetheless, Hwang achieves 89% accuracy on the task. Wu et al. (2015) apply CNNs towards the same goal using aerial imagery, finding it critical to use another algorithm before the CNN to extract regions of interest. Mash et al. (2016) uses closer-scale, non-aerial imagery, but again focus on military aircraft for autonomous aerial refueling.

Dataset Information

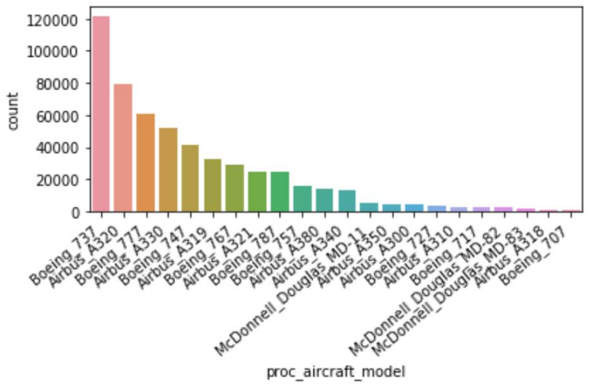

Fig. 1: Class label distributions

Due to compute cost and time limitations, we used only about half our scraped data: about 350GB of images, randomly split 95%/5%/5% into train/validation/test sets (485k images / 27k / 27k). For the milestone, where the model architecture was validated on just 30GB of data, the rationale for allowing an entire 5% for val and test sets was so that we could get enough resolution on accuracy numbers to confidently judge responses to changes in the model architecture; we kept the split after collecting more data because we increase the number of target classes and some are significantly underrepresented, yet we still wanted to ensure the accuracy numbers were meaningful for these classes; given the abundance of training data, there is no reason to decrease the 5% value. At first we attempted to use RGB images, but these were far too memory-consuming for little benefit in performance, so the models presented in this paper were trained and tested using grayscale images.

The only data normalization done is to center each pixel value in [0, 1]. We do not normalize by accounting for training mean or std. dev. because analysis of the dataset showed a wide variety of angles, and filter visualizations suggest the convolutional filters were able to account for these; normalizing by a training mean would harm underrepresented classes' performance because the training mean would be swamped by the majority of up-close images of large airplanes from a single class or two. Further below we show two sample images from the dataset.

Methods

We use a Convolutional Neural Network using a categorical cross-entropy loss, the non-linear ReLU activation, and the Adam optimizer. The Adam optimizer, popular within computer vision because of its versatility in highly non-convex optimization problems, can estimate both moments of the gradients to adaptively adjust the learning rate independently for each parameter. The cross-entropy loss is standard for this type of classification task, where we minimize:

$$H(p,q) = -\sum_{x} p(x)\log(q(x))$$

Wherein x is each input image, and the p and q functions return the one-hot encoded true labels and the predicted softmax distribution respectively. The best architecture is as follows:

```
Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(800,1200), padding='same')
                Conv2D(64, (3, 3), activation='relu', padding='same')
                Conv2D(64, (3, 3), activation='relu', padding='same')
                        MaxPooling2D(pool_size=(2, 2))

                              Dropout(0.5)
                Conv2D(128, (3, 3), activation='relu', padding='same')
                Conv2D(128, (3, 3), activation='relu', padding='same')
                        MaxPooling2D(pool_size=(2, 2))

                              Dropout(0.5)
                               Flatten()
                        Dense(64, activation='relu')
                              Dropout(0.5)
                    Dense(22_classes, activation='softmax')
```

Results

Model	**Train acc.**	**Val acc.**	**Test acc.**	**# epochs**
Guessing most common class	n/a	n/a	24%	n/a
Human expert (estimated)	n/a	n/a	82%	n/a
Bayes' error (estimated)	n/a	n/a	8%	n/a
Arch. B (30GB)	85%	73%	n/a	15
Arch. B (300GB, factor=4)	***84%***	***82%***	***82%***	***4***
Arch. B (300GB, factor=6)	79%	80%	79%	5
Arch. B (300GB, factor=4, padding=same)	81%	80%	80%	5
Arch. B (300GB, factor=6, padding=same)	80%	80%	79%	5

Using just 30GB of data, we iteratively performed bias/variance analysis to arrive at a custom CNN architecture and tuned hyperparameters, similar to the final architecture shown above. At that point, we achieved 85%/73% accuracy on train/val sets respectively. While it may not be immediately obvious whether the network suffered mostly from a bias or variance problem, we ultimately concluded it was largely a bias problem by changing two "knobs" – the number and size of filters, and dropout probabilities and locations. Based on the results of quick test runs manipulating these, a bias problem seemed more likely. By increasing dropout we were able significantly lower training accuracy without affecting validation accuracy excessively; simultaneously increasing parameterization did lead to an increase in validation accuracy. Because of this we concluded there was a bias problem, and more data was necessary, leading to training with approx. 10x the initial amount of data.

Training with the larger dataset yielded immediate performance improvements: after *just a single epoch*, we already achieve 60% validation accuracy. With 5 epochs, the final model achieves 84%/82% accuracy on train/val sets respectively[1]. Note how more vastly more data reduced overfitting.

Different batch sizes did not impact the network's learning speed (as measured in accuracy gain per epoch), so we simply used the most we could fit in GPU network for speed: 32 or 64 images per batch (depending on the resolution and padding), per GPU (we trained mostly on 4-8 GPUs in parallel, dividing into sub-batches of 32/64 and concatenating results on the CPU using Keras's multi_gpu_model).

However, several other hyperparameter and architecture choices did significantly affect the model's performance. Firstly, increasing the input resolution (i.e., reducing the factor divisor) improved performance, as would be expected since the model can absorb more information and match finer templates. There are of course two caveats: increased GPU memory consumption, and needing to train for more epochs to achieve equivalent performance (which is to be expected, as the input distribution space increases quadratically). Because of this, we largely stuck to a factor of 4, resulting in an (812//4, 1200//4) resolution, which was the most effective resolution we had computing resources for.

The second alteration to the model was to use zero-padding for convolutions (padding='same' in Keras) so that the volume sizes between layers is preserved and we don't lose information around the edges; this also significantly helped, because it avoided excessive dimensionality reduction through the network.

Thirdly, we noted that the number of filters we used greatly impacted the network. For example, doubling the number of filters in the last convolutional layer worsens performance, as does reducing the number of early filters. The network was equally sensitive to changes to the 3x3 kernel size. We hypothesize this is because the earlier layers are very focused on extracting the orientation or general size/shape of the airplane, whereas later layers may be looking for finer features (which exist, but are harder to learn, and redundant for *most* airplane types).

To gain some understanding of what or how the CNN was learning, we attempted two kinds of visualization using the keras-vis library. For both, we replace the final softmax layer's activation function with a linear function for better visualization. We first tried Class Activation Maps (grad-CAM), a technique which uses the gradients flowing into the last convolutional layer to find the input image that most relevant to classifying on each class (e.g., training on MNIST will result in images that approximately show the different digits). However, we ran into significant difficulties due to the amount of data/time to train; this was compounded by incompatibilities between keras-vis and the multi-GPU model that were morose to fix.

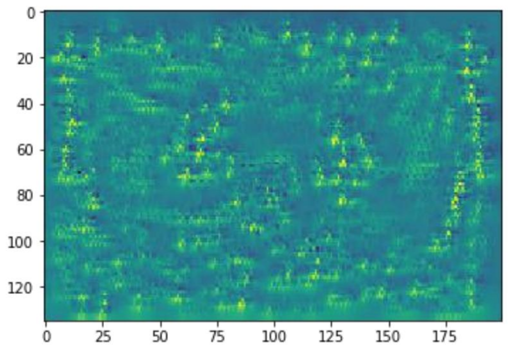

Fig. 2: Input image that maximizes one of the early convolutional filters (typical image)

Instead, we performed Activation Maximization in which we optimize the input image to maximize each filter's activation, producing (ideally) a visual representation of the template each filter has learned to match. Unfortunately, the filters are not particularly informative. We offer two

[1] We were unfortunately unable to train for longer due to the high cost for GPU computing power.

hypotheses as to why: the maximizing image is very dissimilar from the training distribution, so the visualization is too strange without additional regularization applied during its creation[2]; or perhaps the varied positions of airplanes within the image result in overlapping, useful but uninterpretable templates.

Furthermore, we performed error analysis to understand our model's shortcomings by performing manual error analysis of about 30 images per class (on the model trained with 30GB data). The primary conclusions were that the model struggled when the image is not the exterior of an airplane (e.g. cockpit) or multiple airplanes are present in the image, which we then incorporated into our revised Bayes' and human error estimates; and that the model struggled when airplanes were distant (hence, small) when far in the sky. This suggests a direction for future work: to use, for instance, a separate image classifier/segmenter to create a tight bounding box around each airplane and crop aggressively; this would allow for either a smaller (memory-wise) model, or a model of increased complexity at the current size; more importantly, it would remove vast amount of unnecessary information from each image.

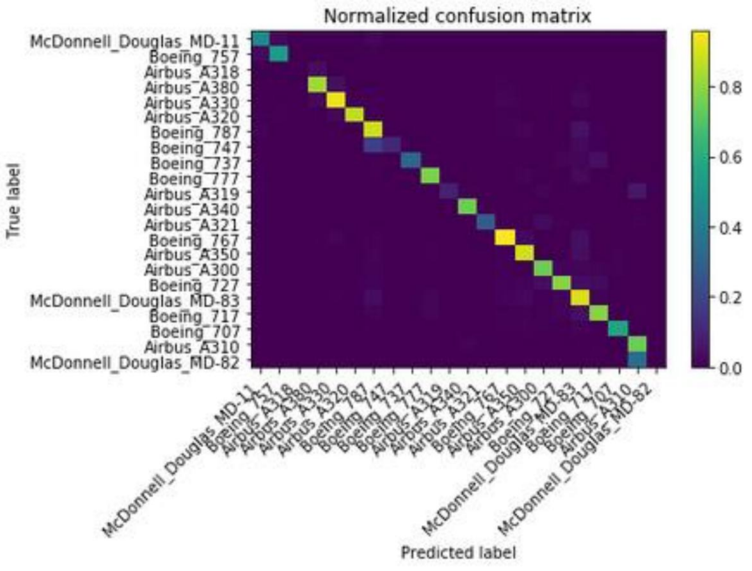

Fig. 3: Confusion matrix for best model

Lastly, since we have a multi-class classification problem with unbalanced class distributions, we analyzed the confusion matrix for our final, best classifier. The confusion matrix shows very strong performance across all classes; interestingly, it tends to get confused by very similar-looking airplanes (e.g. A310 vs. MD-82). Two sample images from the dataset picturing these two airplanes are shown below. Performance is low for the A318 class for two reasons: it is severely undersampled in the dataset as it is not a very common plane; it is difficult to distinguish from an A319 or A320 without a good understanding of the scale/aspect ratio of the images.

We did not perform AUC or precision/recall analysis as their power is reduced in non-binary classification tasks such as these.

Conclusion

Overall, the CNN architecture performs very strongly on a problem with little to no prior academic research, achieving an accuracy equivalent to a human expert, far surpassing an untrained person and a mere 10 p.p. away from estimated Bayes' error. Thanks to the error analyses performed, two directions are clear for future work: firstly, to more aggressively crop images, particularly to remove

[2] Regularization of the input image we are looking for – *not* regularization of the model's parameters!

extraneous (non-airplane) pixels; secondly, to oversample certain challenging classes as well as classes where the confusion matrix indicates areas for improvement.

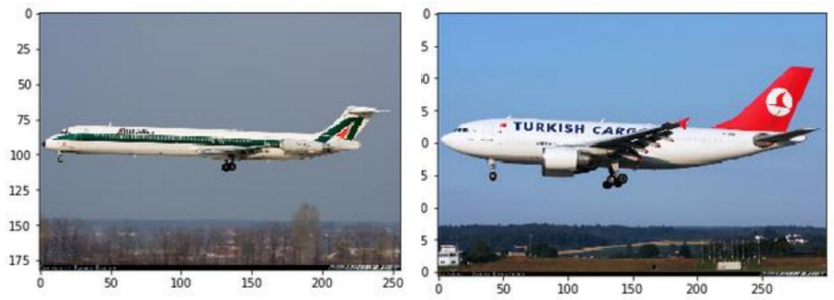

Fig. 4: Sample images from dataset; MD-82 (left) and A310 (right)

Code

https://github.com/guifereis/230-Aircraft-Model-Recognition-CNN/

Acknowledgements & References

Thank you to Amazon AWS for donating compute power used to train these models.

Contributions

All the work here has been done by myself alone, except where cited. Small snippets of code taken from documentation or other sources not included here have been cited in the source code (Git repo).

References

Chollet, François. “Keras.” 2015, github.com/keras-team/keras.

Hwang, Sunyou, et al. “Aircraft Detection Using Deep Convolutional Neural Network in Small Unmanned Aircraft Systems.” *2018 AIAA Information Systems-AIAA Infotech @ Aerospace*, July 2018, doi:10.2514/6.2018-2137.

Mash, Robert, et al. “Toward Aircraft Recognition with Convolutional Neural Networks.” *2016 IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS)*, 2016, doi:10.1109/naecon.2016.7856803.

Raghavendra, Kotikalapudi, and Others. “Keras-Vis.” raghakot.github.io/keras-vis/.

Selvaraju, Ramprasaath R., et al. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization.” *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, doi:10.1109/iccv.2017.74.

Wu, Hui, et al. “ Fast Aircraft Detection in Satellite Images Based on Convolutional Neural Networks.” *Fast Aircraft Detection in Satellite Images Based on Convolutional Neural Networks - IEEE Conference Publication*, ieeexplore.ieee.org/document/7351599/.