Classification of Aircraft

Aircraft type classification compared with bilinear CNN model paper

Jeong Jae Hong,
Kyung Hee University, Computer Engineering
Suwon, Republic of Korea
show2626@nate.com

***Abstract*— Currently, aircraft images are mostly used for images of different shapes classification. For example, cifar10_data is used to classify aircraft, birds, horses, cars, cats and so on. But I think this. What if we could classify aircraft types? Recently, with the rapid development of optical technology, images of distant places can be easily photographed. Therefore, it will be possible to classify aircraft that are relatively far away. It would be the defense industry and civilian airlines. I believe they will be good examples in the future because they will buy large amounts of valuable products.**

But I had a lot of trouble collecting data. Fortunately, those who think like me have only published a paper on aircraft classification in 2013. In that paper, they use the Bilinear CNN model to classify 10,000 pictures of aircraft into 100 categories. However, the average accuracy is 48.69%, though it may be because of the lack of image data. So I try to use a different CNN model here for better accuracy.

Keywords—bilinear cnn; vgg16 cnn; aircraft; aircraft type

I. INTRODUCTION

The FGVC Aircraft paper classified the aircraft into four major categories. The first is to model the criteria as a specific classification. For example, Boeing 737-76J. The second is to divide into models that can not be easily identified by the naked eye. This is a task that can not be easily classified unless it is a professional. The third category is the same family. For example, the 737 Family, such as Boeing 737-700 or Boeing 737-7H4. Finally, it is classified according to the manufacturer. Manufacturers include Boeing and Airbus.

This paper focuses on classifying aircraft through a second class of variants. Because there were too few categories to categorize by manufacturer, and there was not enough data to classify more specifically, like the first method. However, it is not easy to classify 10,000 images into 100 categories. However, there is a difficulty in collecting data by search. In previous papers, it was said that collecting data could be done with the help of a collector who collects photographs by experts and hobbies. [Figure 1] shows the types of aircraft classified into 100 categories

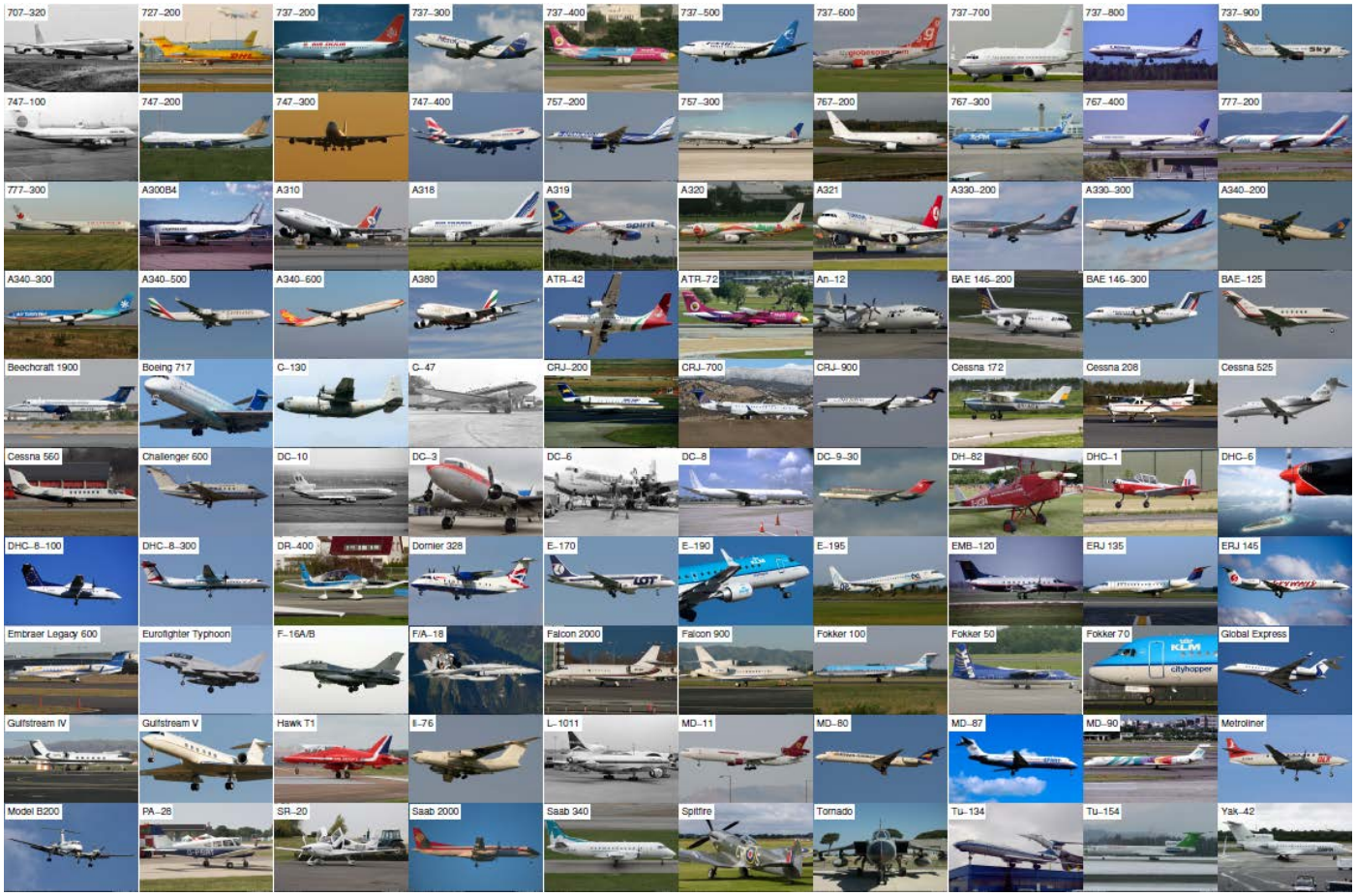
I would like to describe the images used in the following categories. Aircraft photographs have various features. First, aircraft are different in design depending on the size of the aircraft. It can range from small aircraft like jets to large aircraft like air force. The second is that it depends on the designation. Examples are private, civil and military. The third category is classification by purpose. Examples are transportation, carriers and fighters.

II. IN THE PREVIOUS PAPER

A. Bilinear CNN Models

Bilinear CNN Models literally means putting images in two CNN A and CNN B at the same time. The result is multiplied by the outer product and pooled to obtain a bilnear vector. After that, it refers to sorting the bilnear vector by putting softmax.

This model is usually used for fine-grained visual recognition (FGVR). First, CNN A outputs an output including a certain size and position through an input, and CNN B outputs an output including a predetermined size and a position through the same input. The feature outputs are then combined through the matrix outer product at each location. This is called the bilinear feature combination. The classification is then carried out via softmax. The important point here is the back propagation part. Until now, the CNN models simply had one direction, but since this model uses two CNNs at the same time, a different back propagation must be applied to each CNN. It is necessary to apply the differential corresponding to each CNN for the slope descent in the backward propagation calculation. The model is shown in the [Figure2].



[Figure1. Types of aircraft classified into 100 categories]

B. *Image preprocessing*

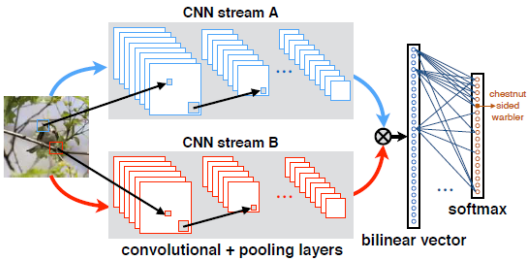
The image data was preprocessed using the tflearn(build_hdf5_image_dataset) module and converted to h5 file. First, the train data set is made up of 6001 pieces of 448x448 size, the validation data set is 666 pieces of 224x224 size, and finally the test data set is 3333 pieces of 224x224 size. Note that the train data set has more images than the other sets, but the size is twice as large, so the train set size is about 15GB. I am sorry that I can not directly turn the data shuffle in my environment (16GB of memory) because I load the entire h5 file in memory when I study it in that paper. It was difficult to load 15 GB into the memory as a whole, but I ended up shuffling.

C. *Momentum optimizer*

In that paper, that used Momentum as an optimizer. We set the learning rate to 0.9 and the momentum to 0.9.

D. *Measurements.*

Accuracy was calculated from 3333 test data, and an average of 48.69% was obtained. The data used in the tests were almost the same for each variant. However, the DR-400, the most well-learned model, came out with 94.1%, and the least learned variant was the 737-300 with 6.1%. Most variants that were not categorized were Boeing 7x7. This is because most of them are hard to distinguish by eye.



[Figure2. Bilinear CNN Model]

III. IN THIS PAPER

But I have some doubts about the Bilinear CNN model. First, I was wondering if it would be better to produce a bilinear vector by combining the results from different CNN models. Each CNN model extracts features and classifies them according to the results. At this time, I think that if two different CNN models extract different features, they would be meaningless if they are mixed together. Second, as a result, the use of the Bilinear CNN model is not famous. If performance was acceptable and good, then others would have experimented with the model. However, the performance was not very good compared with the complexity of the model.

Therefore, I try to classify it through simple and most widely used VGG16 model using the data set used in the above experiment. VGG16 is a model created by the Visual Geometry Group at the University of Oxford and has been awarded the localization and classification Sections 1 and 2 at ImageNet Challenge 2014. Not only is the image classification performance good, but also because it discloses the weight used in the classification, it is recognized as more meaningful academic achievement.

VGG16 has a simpler structure than I thought. As a general CNN structure, a set consisting of convolution layer and pooling layer is layered, and at the end, a vector having 4096 items is output, and classification is performed based on this. A more detailed explanation will be given below. From now on I will describe the parts that I have worked on in this paper.

A. Image preprocessing

- The image preprocessing part was very important. I had to keep failing 10 times for the first time because it was not done properly in Image preprocessing. Since the size of a given data set was one each, Image preprocessing was performed as follows. First, I loaded the image via cv2. The longer side of length and width is multiplied by 256, the shorter side is saved as x, and the image is resized by using interpolation method with longer value as x value and shorter value as 256. After that, it was converted to numpy float32, and the preprocessing was completed using randint, cropped and average imagenet values. This method is a different preprocessing method than existing papers. This is different from tflearn in that it only resizes and divides each image by 255.

B. Use tfrecord to store data

- In the previous article, that used h5py to save it as an h5 file. In this method, all the train data are loaded in the memory and there is a problem because of excessive memory use but there is a big problem. Since the data set used in this experiment is small, it is necessary to repeatedly learn by shuffling the data. However, there is a problem when shuffling train data loaded in memory. It was an extra memory for shuffle, which was very inefficient.

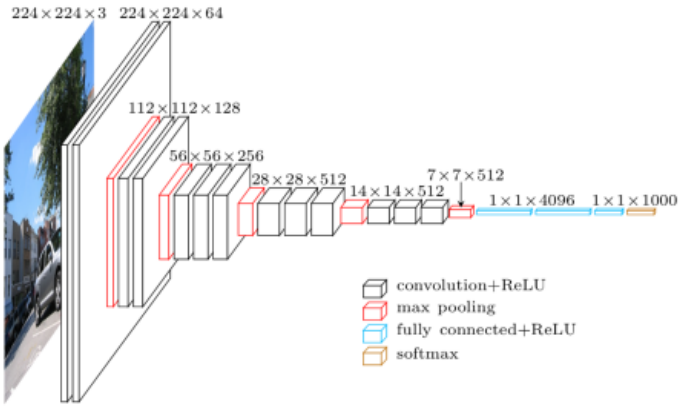
- So I decided to use Tfrecord as an alternative. The Tfrecord file is a binary data format for storing learning data of tensor flow, and it is stored in Google's Protocol Buffer format by serializing the data to a file. Also, since the labels are also stored together, it is possible to avoid the inconvenience by bringing the labels too. The advantages of Tfrecord are: The name of the file can be loaded into the queue first. At this time, since the epcoch is repeatedly designated, data can be loaded more systematically. It then reads the serialized data through Tfrecord. Transforms the serialized data into the original image shape through reshape. Next, the most important shuffle_batch makes it easy to shuffle. You can also specify batch_size. After that, convert the label by one_hot encoding and return it to complete data storage and retrieval. Also, shuffle_batch showed about 10 times faster performance than using h5py with 4 threads. (Training data set: 6001, Validation data set: 666, Test data set: 3333)

C. VGG16 Model and pre-trained parameters for VGG16 in TensorFlow

- The VGG16 model was easily available. At first, I got the vgg16 model file in the simplest way and imported and applied it. However, the weight value was applied randomly at first, and it took quite a long time to learn properly. Therefore, it was shown that the weights file which is used in other experiments and optimized is supported in npz format and the learning is converged at least three times faster than learning. I then saved it in my own npz file storage format and experimented with changing each parameter value.

- VGG-16 is as follows. It literally means to use 16 layers. The input should contain an image of 224x224x3. In the first Conv, we used 3x3x64 filters twice. Again, all activation functions use relu, add bias value, and proceed to pooling [2x2] at the end of Conv. The result is 112x112x128. The second Conv2 uses a 3x3x128 filter to produce a result of 112x112x128. The third Conv used a 3x3x256 filter three times and produced a result of 28x28x256. The fourth Conv used a 3x3x512 filter three times, resulting in 14x14x512. In the last Fifth Conv, we used 3x3x512 filters 3 times to get 7x7x512 results. It is composed of softmax which can convert it into 1x1x4096 in one line and then classify it into 100 kinds.

- It was easy to use because it was well documented in Google how to load the previous weight value into VGG16. However, there was no detailed description of how to learn and store the weight values separately. Therefore, in VGG model structure, the parameter name values are stored separately, and the code to save as npz file is added through sess.run. This task seemed simple at first glance, but it was complicated and difficult. I took a short time to learn the 20epoch because the learning speed was as fast as I said earlier. Therefore, we could save the weight value every 20 epochs and easily back up overfitting.



[Figure3. VGG-16 Model]

D. Adam optimizer

- Adam is an optimization algorithm similar to RMSProp, which directly estimates parameter updates from the mean and variance of the gradient and adds a bias adjustment. The learning rate is set to 0.0001.

E. Experiment environment

- CPU: AMD Ryzen5 1600 six core, 3.20GHz / RAM : 16GB / GPU : NVIDIA GeForce GTX 1060 6GB / OS: Windows10

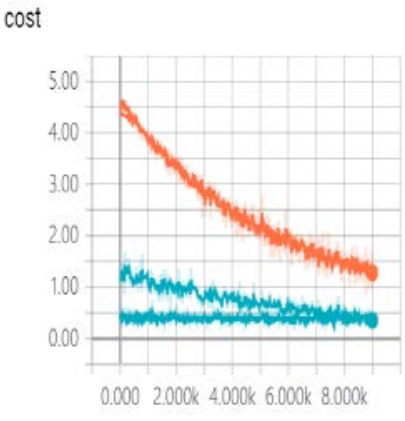
F. Experiment result

- As a result of my experiments, the accuracy of 46.33% was obtained by adjusting 1536 pieces in 3333 test data sets. This was 2% less accurate than the previous paper. I thought that it would be okay if I had about 46% in 100 categories, but I wondered if this was exactly the learning. So, in the previous article, 16 of the models that were well learned (DR-400, Eurofighter Typhoon, F-16, Cessna 172, SR-20, BAE-125, DH- 1, Il-76, An-12, Falcon 900, PA-28, and Spitfire, 534 pictures, It was saved as “testset” tfrecord.) were selected and extracted from the test data set. The result was surprisingly 82.39% accuracy. If you look at the models below [Figure 4], the features are noticeably different. Comparisons between models such as the Boeing 7xx say that they could hardly be matched exactly.

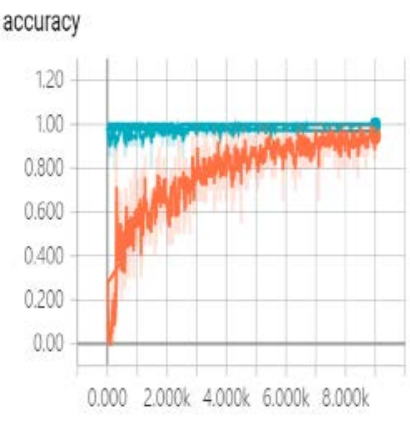
IV. CONCLUSION

Even if that do not use the Bilinear CNN model, I could achieve some similar results with the VGG16 model. If I have a chance, Google Image Crawler has the desire to categorize more and more models to grow your data. I also tried k-fold to save less data, but I wanted to apply it if I had a chance next time.

[Figure4. Well classified aircraft types]



[Figure5. Cost value]



[Figure6. Accuracy value]

In Figure 5 and 6, For the cost value and the accuracy value, the orange color is the result of the initial 20 epoch, and the blue color is then recorded after the weight is saved and then again by 20 epoch.

REFERENCES

[1] University of Massachusetts, Amherst "Bilinear CNN Models for Fine-grained Visual Recognition" Tsung-Yu Lin Aruni RoyChowdhury Subhransu Maji, 2015

[2] Subhransu Maji, Esa Rahtu Juho Kannala, Matthew Blaschko, Andrea Vedaldi "Fine-Grained Visual Classification of Aircraft", 2013

[3] Karen Simonyan, Andrew Zisserman "VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE OGNITION"

[4] This paper refers to a part of the contents of the corresponding github and informs that the same part is very small (https://github.com/abhaydoke09/Bilinear-CNN-TensorFlow)