CS4644/7643 Final Project Report
Vehicle Model Predictor

Venetis Pallikaras, Anirudh Prabakaran, Jin Woo, Tongdi Zhou
Georgia Institute of Technology

Abstract

Fine-Grained Image Classification (FGIC) problems have become popular as image detection advances from classifying different objects with completely different characteristics to distinguishing similar objects with minute discrepancies. In this paper, we tackle the problem of fine-grained vehicle classification by training a model using traditional CNNs such as Resnet-18 and Resnet-50, Bilinear CNNs, and Vision Transformers. The results showed that the CNN performed better at 100 make and model labels, but after that, the Bilinear CNN was able to perform better than the traditional CNN. In the case of make, model, and year labels, the Bilinear CNN was able to perform better in every circumstance, indicating that the Bilinear CNN could be more suitable for performing FGIC on more complex data compared to a regular CNN. Likewise, we were also able to show that a Vision Transformer model performed slightly worse than a Bilinear CNN model with make and model labels but performed much worse with the make, model, and year labels. For make and model, it performed worse than both ResNet-18 and ResNet-50 for the top 100 labels, but better in the top 300 labels. For make, model, and year, it performed better than both ResNet-18 and ResNet-50 in all cases.

1. Introduction

In this project, we have taken up a fine-grained classification of predicting a vehicle's make and model given an input image of a vehicle using various neural network architectures.

1.1. Applications

There are numerous applications for vehicle identification, including surveillance for crime prevention, traffic management, targeted advertisement, and even behavior analysis. With the meteoric rise of the Autonomous Driving industry, it becomes increasingly important to be able to recognize these vehicles in order to contribute to Intelligent Transport Systems (ITS). This can aid in the advancement of logistics, traffic management, and commuter safety.

1.2. Goal

The goal of this project is to experiment with various networks for the above task of vehicle identification. We want to build a model that is most suited for such fine-grained classifications. Additionally, since we plan to evaluate our models in a real-world setting, we want to analyze various aspects of the model including but not limited to accuracy, precision, recall, model size, inference time, and ability to handle various angles of the vehicle.

1.3. Related Works

The performance of image classification using deep learning has steadily improved over the years [13, 17, 20, 25]. ResNet has been used in various image classification projects. One such project that is slightly related to our project is the classification of vehicles, ranging from but not limited to bicycles, buses, cars, and motorcycles to be used in traffic surveillance systems [10]. ResNet is also used in biotech research such as classifying white blood cells [8] and malarial infected cells [19].

Vision transformer models are a recent advancement in image classification and have been used in a wide range of applications such as Covid-19 classifications from CT scans [4], breast ultrasound image classifications [7], and recycle waste classification [9].

Unlike generic image classification, Fine-Grained Image Classification (FGIC) aims to distinguish the minor differences of objects within the same category [24]. Our project, the classification of cars by make and model, falls under FGIC. FGIC has been applied in a wide range of applications such as identifying bird breeds [22], vehicle re-identification [11], and recognizing aircraft makes [16]. The main challenge of FGIC is to extract the fine-grained features that can differentiate highly similar objects.

An effective method to extract the fine-grained features is to use image segmentation [2, 3] to locate the key features, such as taillights, front grills, or side view mirrors. As a result, the deep network can learn more discriminative

features. However, these methods may require part annotation, which can be difficult and expensive. A better approach is to exploit the higher-order interactions between the features. For example, the covariance matrix-based representation characterizes the second-order feature interactions [23]. Based on this, the Bilinear CNN was proposed where images are represented by the outer product of the outputs from two separate CNNs [14,15]. The outer product produces the second-order interactions between features extracted by the two CNNs, and the covariance matrix is flattened before being fed to a fully connected layer for classification. Bilinear CNN was found to work well for FGIC, and subsequent work [6,12,18] was done to address the problem that the higher-order representation may largely increase the number of parameters, which can cause issues like overfitting.

2. Technical Approach

2.1. Method 1: Transfer Learning

For the baseline methods, we employ a transfer learning based approach where we fine-tuned a conv-net built on top of various backbone models.

Transfer Learning is an idea in Machine Learning, where a model developed for a task is reused as the starting point for a model on a different task. This approach has proven to be extremely useful in Deep Learning, since users often do not have high compute power to train deep neural networks with large datasets. With transfer learning, users can now reuse a network that has already been pre-trained on a large and exhaustive dataset (typically done by organizations with high compute power). The key idea why transfer learning works very well in CV applications is that, early layers of deep convolutional neural nets learn small patterns and shapes while the final layers learn more complex objects like a car. Hence, by reusing these initial layers and modifying the final few layers, we can achieve very good performance for our specific problem statement even with small datasets.

In this project, the backbone models we chose as the feature extractors are ResNet-18, ResNet-50 and MobileNetV2. Figure 1 shows an example of an architecture typically used for Transfer Learning.

2.2. Method 2: Bilinear CNN

Figure 2 shows the basic structure of a bilinear CNN, where an image is processed by two CNNs, and the outputs from the two CNNs are fed into a matrix outer product followed by average pooling to generate the bilinear features [14, 15]. The class predictions are obtained by passing the bilinear representations through a linear and softmax

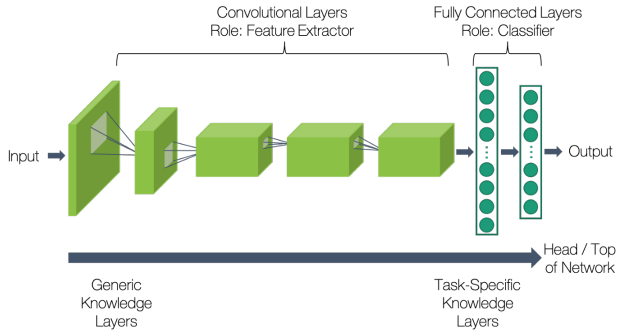

Figure 1. Architecture used for Transfer Learning

layer.

The effectiveness of bilinear CNNs in classifying fine-grained recognition tasks stems from their capability to exploit the second-order statistics of the features. The resulting high-dimensional bilinear representation contains rich information about the interactions of the features extracted by the two CNNs, which enables the network to capture useful localized features.

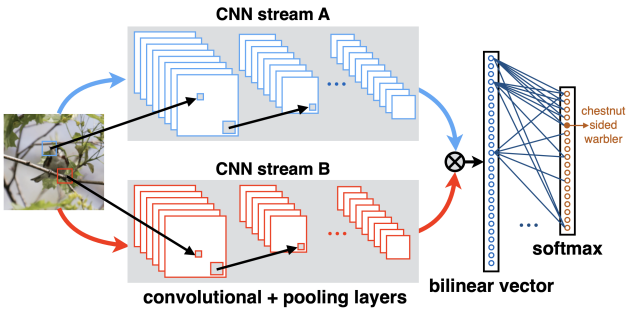

Figure 2. Image classification using a Bilinear CNN

The Bilinear CNN can be represented by a quadruple $\mathcal{B} = (f_A, f_B, \mathcal{P}, \mathcal{C})$ where f_A and f_B are the CNN feature functions, $\mathcal{P}$ is the pooling function, and $\mathcal{C}$ is the classification function. The CNN feature function $f : \mathcal{L} \times \mathcal{I} \rightarrow \mathbb{R}^{K \times D}$ maps an image $I \in \mathcal{I}$ and a location $l \in \mathcal{L}$ to a $K \times D$ feature. The features from the two CNNs are combined using a outer product given by

$$\text{bilinear } (l, I, f_A, f_B) = f_A(l, I)^T f_B(l, I)$$

Finally, the pooling layer combines the bilinear features across different locations and can be represented by

$$\Phi(I) = \sum_{l \in \mathcal{L}} \text{bilinear } (l, I, f_A, f_B) = \sum_{l \in \mathcal{L}} f_A(l, I)^T f_B(l, I)$$

Here, we use a sum pooling. Finally, the pooled features are passed to a fully connected layer for classification. Since the Bilinear CNN classifier is a directed acyclic

graph (DAG), we can use back-propagation to train the network. We can use loss functions such as cross-entropy. In terms of back-propagation, suppose the outputs of the two CNNs are matrices A and B, and the bilinear representation is $\mathbf{x} = A^T B$, by chain rule, we have

$$\frac{d\ell}{dA} = B\left(\frac{d\ell}{d\mathbf{x}}\right)^T, \quad \frac{d\ell}{dB} = A\left(\frac{d\ell}{d\mathbf{x}}\right)$$

Figure 3 shows the flow of gradient in a Bilinear CNN at the bilinear representation generation stage. Note that we apply a normalization where we pass the bilinear representation through a signed square-root followed by a l_2 normalization.

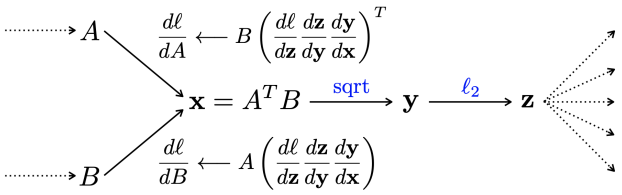

Figure 3. Gradient flow in a Bilinear CNN

In practice, the CNNs are pre-trained on dataset like the ImageNet and are truncated before the fully connected layers to obtain the features. In addition, the two CNNs can be either the same or different. If the two CNNs are of the same architecture, due to the symmetry, they have the same weights. As a result, we only need to implement one CNN, which alleviates the computational complexity of the overall model. If the two CNNs are different, and they produce different feature maps, resizing may be required for the features to generate the bilinear representation. During the training, we adopt a two-phase procedure described in [1]. In phase 1, we fix the pre-trained CNNs and only train the fully connected layer using a relatively large learning rate. In phase 2, we train the entire network using a small learning rate for a large number of epochs.

The high-dimensional bilinear representation is largely redundant. A possible optimization is to use principal component analysis to project the bilinear representation onto a lower dimension before passing it to the fully connected layer. Such projections can still be expensive in computation. Alternatively, we can project the features from the two CNNs to lower dimensions before generating the bilinear CNN. However, this approach can degrade the accuracy significantly [5, 15]. A third approach is to only reduce the dimension of the features from one CNN, which has been found to work well in practice [15].

2.3. Method 3: Vision Transformer

ViT aims at bringing "attention" to the image classification world. In order to be able to feed an image to a transformers, we need to split it to smaller "patches" and flatten those. We then create embeddings for those patches just as we would in NLP, but we also add positional encodings indicating the original location of the patch/embedding (eg. first patch). Essentially image patches work to a similar way to tokens in NLP tasks.We then feed the embeddings to a classic transformer encoder architecture. and finally to a fully connected softmax layer in order to perform classification.

2.4. Method 4: Aggregated by Brands

A second approach that we tried, was to aggregate all images to just car brand folders, ignoring the subtypes in order to reduce the number of classes. We then split again our data to train,validation and test in similar fashion to the previous approach. We then proceeded to use data augmentations techniques either via Albumentations or via Torchvision. The latter had a wider variety of augmentations and it was significantly faster, thus it was preferred over Torchvision. Following this, we used and plan to use a variety of pre-trained architectures such as Resnet and Densenet, while modifying the number of layers within them. For example, we tried both Resnet18 and Resnet101. For the time being for this approach we only used accuracy as an evaluation metric, but we plan to add more such as F1-score and precision and recall. Training/Validation curves as well as metrics and statistics were tracked using the Weight and Biases library. Our Loss function was CrossEntropyLoss In the future we plan to explore the Sweeps functionality of WB for better parameter optimization. Google Collab was used for training and validation, thus the GPU model and computer characteristics weren't fixed due to the nature of Collab.

2.5. Evaluation Procedure and Metrics

We used 80% of our dataset for training, 10% for validation and used the remaining 10% as the independent test set. Our training and validation dataloaders were configured to have a batch size of 16 for faster and efficient training. Our test dataloaders are configure to have a batch size of 1 (inference will be done one image at a time). For evaluation of all our models, we will be using the same standardized set of metrics:

- Accuracy - Ratio of correct predictions to total predictions.
- Precision - Weight-averaged precision across all labels.
- Recall - Weight-averaged recall across all labels.
- Inference time - Since our end-goal is to try and evaluate this on real-driving videos. We want to prioritize

models with least inference time so that it can work in real-time.

3. Dataset and Data Preprocessing

3.1. Dataset

We are using an established dataset, the VMMRdb [21], for this project. It contains 291,752 images of vehicles, which provides us with a great scale. The dataset contains 9,170 classes, where the class label consists of the make, the model, and the year (ranging from 1950 to 2016) of individual vehicles. The vehicle images are of great diversity. They were taken by different individuals and different imaging devices, they also contain different angles of the vehicles. The images are in .jpg format and have a size ranging from 20 to 100 KB. The images are organized in folders where each folder contains vehicles of the same make, model, and year.

For the target label, we first consider only the make and model of the vehicles. This reduces the number of labels to 1175. We then advance to classifying the make, model, and year of vehicles. We expect the latter task is much harder because annual versions of cars are usually not significantly different. For example, Honda Civic 2007 looks almost the same as Honda Civic 2008. Additionally, since our dataset is extremely imbalanced (as plotted in the dataset analysis section), we experiment with multiple values for the number of target labels. For example, we choose the top-100, top-200 and top-300 most commonly occurring make-model instances in our dataset. Table 1 summarizes the various dataset configurations used and also gives an idea of the imbalance we are dealing with in this task.

Top-X	Dataset Size	Most Frequent	Least Frequent
100	184985	honda civic (6895)	mitsubishi lancer (774)
200	231245	honda civic (6895)	ford escort (315)
300	254784	honda civic (6895)	acura cl (169)

Table 1. Various dataset configurations used

3.2. Data Preprocessing

In the original dataset, samples are organized by their most granular labels, that is, the make, the model, and the year. In our data preprocessing, we have three goals. The first goal is to convert the string labels to integer labels for easy processing. Instead of using "Chevrolet impala 2012", we use a tuple of 3 integers to represent this label. The second goal is to have the ability to control the granularity of the dataset. To achieve this, we developed multiple mapping functions that map labels to tuples and tuples to labels. With the integer labels, we can easily randomize or split the samples. Figure 4 shows six random samples drawn from the dataset.

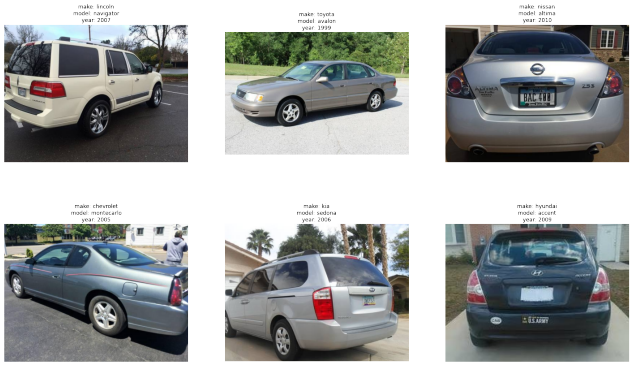

Figure 4. Six samples of the dataset

The final goal of our data preprocessing is to visualize the sample size for different label granularities. That is, we would like to see which label has more samples when compared to other labels. This helps us to understand the dataset imbalance and helps us to decide which labels to use. Figure 5 shows the 18 car brands with the most amount of samples in the dataset.

Figure 5. The treemap of the 18 car brands with the most amount of samples

Figure 6 and figure 7 show the data treemaps when we use make+model and make+model+year as labels. We can see that as we increase the granularity, the sample size for each class decreases, which makes the classification more

challenging. However, we can also observe that the class imbalance is alleviated as we use more granular labels, which may improve the classification performance.

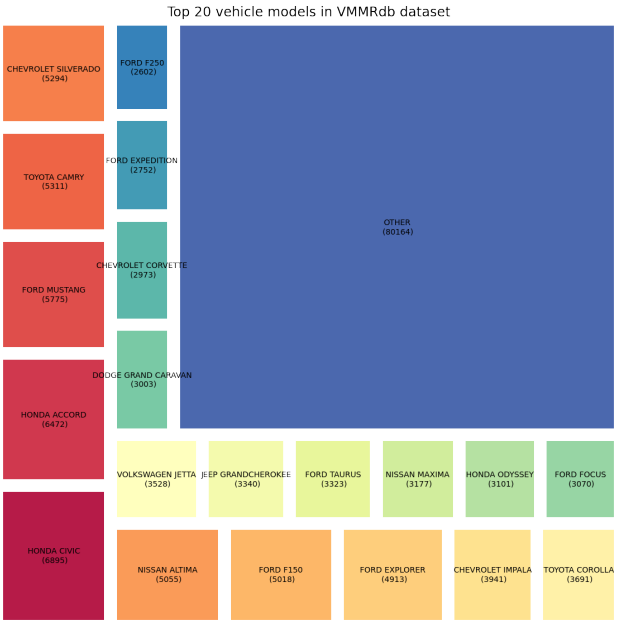

Figure 6. The treemap of the 20 car models with the most amount of samples

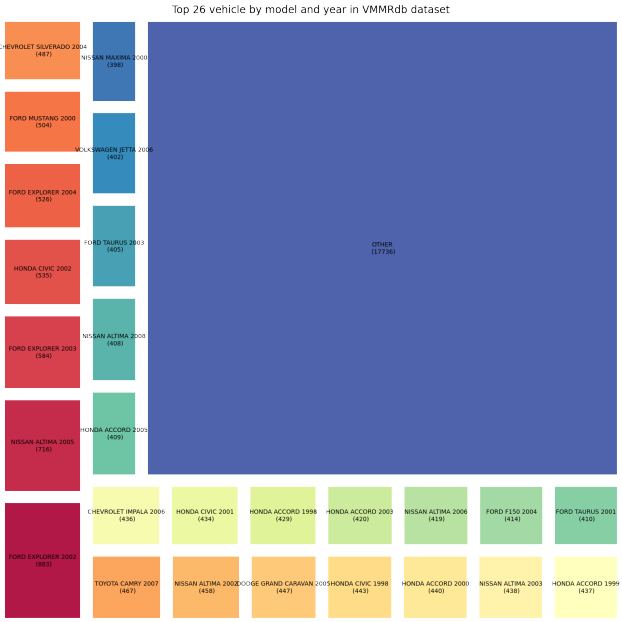

Figure 7. The treemap of the 26 car models (different years) with the most amount of samples

In addition, we use various data augmentation and normalization methods. We resize all the images to the same size of 224×224. We utilize random horizontal flips where images chosen at random are replicated and flipped horizontally. We also normalize the training data using deterministic mean and standard deviation. These methods increase the variety of data samples while keeping them standardized for the training process.

4. Experiments and Results

4.1. Method 1: Transfer Learning

For all the models in this section, we train them for 10 epochs, use an SGD optimizer with a learning rate of 0.001 and momentum of 0.9. We decay the learning rate by gamma every 7 steps. We used CrossEntropyLoss with weight distribution (to account for the class imbalance in our dataset) as our loss function. The models were trained on an NVIDIA Quadro P6000 GPU.

Table 2 summarizes accuracy and inference time from our baseline methods. Here, accuracy refers to the accuracy computed when evaluating on the test dataloaders with a batch size of 1. Inference time, refers to the average time taken per image for the final trained model to perform inference. Table 3 summarizes precision, recall and F1 for our baseline methods. All the metrics are weighted averages over the different labels used.

We see how the model performance is gradually reducing as we increase the size and number of labels of the dataset. This makes sense as the model is finding it difficult to accurately distinguish between classes, especially the ones with low number of training instances. Our end goal is to beat these baselines with our Bilinear CNN, since they are known to work well for FGIC tasks.

Figure 8 shows examples of some sample predictions by this method on the test set. Although the model predicts pretty accurately, we see an example of a wrong prediction in the 2nd image. A "Honda Civic" was wrongly classified as a "Honda Civiccoupe". In the next method, we hope the Bilinear CNNs can better capture such fine differences in images.

Backbone	Size (MB)	Top-X	Accuracy	Time (s)
ResNet-18	45	100	94.97	0.0039
		200	84.57	0.0082
		300	81.01	0.0080
ResNet-50	96	100	97.30	0.0078
		200	89.10	0.0132
		300	87.44	0.0131

Table 2. Accuracy and Inference time for make+model

Figure 8. Sample predictions for make+model

Backbone	Top-X	Precision	Recall	F1
ResNet-18	100	95.5	94.96	95.08
	200	86.31	84.57	84.87
	300	84.48	81.0	81.79
ResNet-50	100	97.54	97.29	97.34
	200	90.57	89.09	89.46
	300	89.26	87.43	87.84

Table 3. Precision, Recall and F1 for make+model.

4.2. Method 2: Bilinear CNN

4.2.1 Bilinear CNN with ResNet-18

For the Bilinear CNN using ResNet-18, the model was trained in two phases using a Nvidia V100 on Google Cloud. The first phase consists of only training the last FC layer. The model was trained for 30 epochs in this phase, with a base learning rate of 1.0 and a weight decay of 1e-8. The scheduler used was the ReduceLROnPlateau in Pytorch, and it would reduce the learning rate by a factor of 0.1 for every 3 epochs the model saw no improvement.

The second phase is when all the layers are fine-tuned. The model was trained for 30 epochs in this phase, with a base learning rate of 1e-2 and a weight decay of 1e-5. The scheduler was also ReduceLROnPlateau using the same parameters as phase 1.

For this Bilinear CNN, the model was trained on make and model labels to compare to the previous ResNet models. We expected the Bilinear CNN to perform at least equal to ResNet-18 since it used ResNet-18 as the backbone for it's two CNNs.

Table 4 summarizes the resulting accuracies and inference times for the Bilinear CNN model that was trained using ResNet-18 as the two CNNs. Table 5 shows the resulting precision, recall, and F1 score for this Bilinear CNN. Compared to 2, the BCNN has a slight disadvantage in accuracy for the top 100 make and model labels. This disadvantage continues to show in its lower F1 score. However, the gap widens in the BCNN's favor once the training data becomes more complex. In the Top 200 and 300 labels, the BCNN has a smaller accuracy drop than ResNet-18 and ResNet-50. This is also reflected in the F1 scores. Interestingly, the BCNN had a lower F1 score than ResNet-50 by less than half a percent, although the accuracy was over two percent higher. The results here do show that the BCNN could have the potential to perform more complex FGIC better than a regular CNN.

Backbone	Top-X	Accuracy	Time (s)
BCNN (ResNet-18)	100	93.69	0.0025
BCNN (ResNet-18)	200	91.42	0.0026
BCNN (ResNet-18)	300	89.81	0.0026

Table 4. Accuracy and Inference time for Bilinear CNN using 2 ResNet-18

Backbone	Top-X	Precision	Recall	F1
BCNN (ResNet-18)	100	93.75	93.31	93.49
BCNN (ResNet-18)	200	90.98	90.17	90.44
BCNN (ResNet-18)	300	87.33	87.16	87.05

Table 5. Precision, Recall and F1 for Bilinear CNN using 2 ResNet-18

One final note of interest is that the BCNN had a consistent inference time of around 0.0026 seconds for this project regardless of the complexity of the training data as shown in 4 compared to 2, where the inference time does increase as complexity increases. The machines used to train the different models are not consistent, so it's possible that hardware differences were the key factors that led to the discrepancy in inference time.

4.2.2 Bilinear CNN with VGG-16

In this section, we use a bilinear CNN with two VGG-16s to classify vehicles by make, model, and year. The model is trained on the Google Cloud Platform with an Nvidia V100 GPU. The VGG-16 network is truncated to remove the fully connected layers and the final pooling layer to obtain the features. There are 512 features from each VGG-16 and the resulting bilinear representation has a dimension of 512^2, which is then passed into a fully connected layer with the number of classes as the output dimension. The training process is largely the same as in Section 4.2.1. The only differences are that the fully connected layers are trained for 55 epochs and the entire network is trained for 25 to 30 epochs in the fine-tuning phase. We trained the bilinear CNN for the top 100 to 500 classes of make, model, and year, and the results are shown in Table 6. We can see that the testing accuracy is fairly consistent for different numbers of classes.

There is no significant degradation in performance as we increase the number of classes, which shows bilinear CNN's ability to capture fine details between classes.

Top-X	Accuracy	Precision	Recall	F1
100	62.79	63.52	62.98	62.93
200	55.75	56.37	55.75	55.47
300	55.74	56.78	55.73	55.37
400	54.42	55.14	54.41	53.98
500	52.75	53.77	52.75	52.37

Table 6. Testing statistics of the bilinear CNN for classifying vehicles by make, model, and year

4.3. Method 3: Vision Transformer

For this section, we used Hugging Face to train our models and more particularly the "Google-ViT-Base", "Facebook-Convtext" and "AAraki-Cifer10-Finetuned". The first two are pretrained using ImageNet while the last one is a pretrained ImageNet "Google-Vit Base" model which was also trained on the Cifar10 dataset. The one that achieved the highest performance in all metrics was the model from Google. It should also be noted that we used "Weight and Biases" to track the metrics as well as the experiments overall. We can see the relevant results bellow:

Backbone	Top-X	Accuracy	Time (s)
Google ViT - Base	100	93.28	0.01
Google ViT - Base	200	90.01	0.01
Google ViT - Base	300	88.81	0.01

Table 7. Accuracy and Inference time for Google ViT

Backbone	Top-X	Precision	Recall	F1
Google ViT - Base	100	92.75	92.86	92.2
Google ViT - Base	200	90.28	89.46	89.07
Google ViT - Base	300	87.68	87.81	87.58

Table 8. Precision, Recall and F1 for Google ViT

4.4. Method 3: Aggregated by Brands

At Table 9 we can see the resulting accuracy for the second model approach. More evaluations metrics will be added for the final presentation.

Although the number of classes has been reduced the accuracy has decreased compared to the first method. This originates from the fact that our data have images from cars that date from 1970 to 2020. Thus, it was hard for the model to distinguish the brand since car shape and looks would change drastically for the same brand/class throughout the years.

Backbone	Accuracy
ResNet-18	65
ResNet-101	69
DenseNet	71

Table 9. Accuracy for dataset with aggregated brands

4.5. Discussion

Table 10 shows the testing accuracy of different deep learning methods when we use make+model labels. At 100 classification classes, ResNet-50 has the highest accuracy due to its great expressiveness enabled by the deep architecture. The Bilinear CNN has comparable performance as the ResNet18. As we increase the number of classes, all models perform worse since the differences between the classes decrease, which makes the classification more difficult. However, Bilinear CNN has the least amount of performance degradation, achieving an accuracy of 89.81% with 300 classes. This is a result of Bilinear CNN's capability of discerning fine-grained features through their higher-order interactions in the bilinear vector.

	Top 100	Top 200	Top 300
MobileNets	81.33%	—	—
ResNet-18	94.97%	84.57%	81.01%
ResNet-50	97.30%	89.10%	87.44%
Bilinear CNN (ResNet-18)	93.69%	91.42%	89.81%
Vision Transformer	93.28%	90.01%	88.81%

Table 10. Testing accuracy of different deep learning methods for make+model labels

Table 11 shows the model performance when we use make+model+year as labels. We can see due to the difficulty in classifying years, all the models perform much worse than in Table 1. However, even with a shallower network (VGG-16), Bilinear CNN outperforms ResNet-50 by 5% across all scenarios and outperforms Resnet-18 by almost 10% in some cases. This illustrates Bilinear CNN's superiority in fine-grained image recognition.

5. Conclusion

We conclude that depending on the granularity of the labels, different models should be used for image classification. With less granular labels or a relatively small number of classes, ResNets are preferred for their performance and easy training process. As the labels get more granular or when the number of classes is large, Bilinear CNNs can be used to leverage their ability to detect the subtle differences in fine features of different classes.

	Top 100 Classes	Top 200 Classes	Top 300 Classes	Top 400 Classes	Top 500 Classes
ResNet-18	54.50%	47.56%	46.90%	43.34%	42.22%
ResNet-50	55.60%	50.64%	49.34%	47.12%	46.75%
Bilinear CNN (VGG-16)	62.79%	55.75%	55.74%	54.42%	52.75%
Vision Transformer	58.67%	56.66%	53.94%	51.28%	49.99%

Table 11. Testing accuracy of different deep learning methods for make+model+year labels

For future work, we can explore how bilinear CNNs perform when we use two different CNNs. We would like to compare and contrast the performance and computational efficiency between bilinear CNNs that use two identical CNNs and the ones with different CNNs. In addition, we would like to investigate various dimensional reduction schemes for bilinear representation, described in Section 2.2, and how they affect the network performance in terms of different metrics.

6. Team Contribution

- Venetis Pallikaras:
  - Experimented with various preprocessing and data augmentation methods
  - Trained models using make dataset and trained transformers for the rest datasets
- Anirudh Prabakaran:
  - Project setup. Implemented the dataset class and other utility methods.
  - Training, testing and evaluation of all baseline methods on make+model and make+model+year datasets.
- Jin Woo:
  - Build bilinear CNN with two ResNet-18 networks and trained the model with make+model labels
- Tongdi Zhou:
  - Data analysis and visualization
  - Build bilinear CNN with two VGG-16 networks and trained the model with make+model+year labels

References

[1] Steve Branson, Grant Van Horn, Serge Belongie, and Pietro Perona. Bird species categorization using pose normalized deep convolutional nets, 2014. 3

[2] Yin Cui, Yang Song, Chen Sun, Andrew Howard, and Serge Belongie. Large scale fine-grained categorization and domain-specific transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4109–4118, 2018. 1

[3] Yin Cui, Feng Zhou, Yuanqing Lin, and Serge Belongie. Fine-grained categorization and dataset bootstrapping using deep metric learning with humans in the loop. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1153–1162, 2016. 1

[4] Xiaohong Gao, Yu Qian, and Alice Gao. Covid-vit: Classification of covid-19 from ct chest images based on vision transformer models, 2021. 1

[5] Yang Gao, Oscar Beijbom, Ning Zhang, and Trevor Darrell. Compact bilinear pooling, 2015. 3

[6] Yang Gao, Oscar Beijbom, Ning Zhang, and Trevor Darrell. Compact bilinear pooling. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 317–326, 2016. 2

[7] Behnaz Gheflati and Hassan Rivaz. Vision transformers for classification of breast ultrasound images. In *2022 44th Annual International Conference of the IEEE Engineering in Medicine Biology Society (EMBC)*, pages 480–483, 2022. 1

[8] Mehdi Habibzadeh, Mahboobeh Jannesari, Zahra Rezaei, Hossein Baharvand, and Mehdi Totonchi. Automatic white blood cell classification using pre-trained deep learning models: Resnet and inception. In *Tenth international conference on machine vision (ICMV 2017)*, volume 10696, pages 274–281. SPIE, 2018. 1

[9] Kai Huang, Huan Lei, Zeyu Jiao, and Zhenyu Zhong. Recycling waste classification using vision transformer on portable device. *Sustainability*, 13(21), 2021. 1

[10] Heechul Jung, Min-Kook Choi, Jihun Jung, Jin-Hee Lee, Soon Kwon, and Woo Young Jung. Resnet-based vehicle classification and localization in traffic surveillance systems. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017. 1

[11] Sultan Daud Khan and Habib Ullah. A survey of advances in vision-based vehicle re-identification. *Computer Vision and Image Understanding*, 182:50–63, 2019. 1

[12] Shu Kong and Charless Fowlkes. Low-rank bilinear pooling for fine-grained classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 365–374, 2017. 2

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017. 1

[14] TsungYu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear convolutional neural networks for fine-grained visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 40(6):1309–1322, 2017. 2

[15] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear cnn models for fine-grained visual recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 1449–1457, 2015. 2, 3

[16] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013. 1

[17] Hieu Pham, Zihang Dai, Qizhe Xie, and Quoc V Le. Meta pseudo labels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11557–11568, 2021. 1

[18] Ninh Pham and Rasmus Pagh. Fast and scalable polynomial kernels via explicit feature maps. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 239–247, 2013. 2

[19] A. Sai Bharadwaj Reddy and D. Sujitha Juliet. Transfer learning with resnet-50 for malaria cell-image classification. In *2019 International Conference on Communication and Signal Processing (ICCSP)*, pages 0945–0949, 2019. 1

[20] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[21] Faezeh Tafazzoli, Hichem Frigui, and Keishin Nishiyama. A large and diverse dataset for improved vehicle make and model recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 1–8, 2017. 4

[22] Grant Van Horn, Steve Branson, Ryan Farrell, Scott Haber, Jessie Barry, Panos Ipeirotis, Pietro Perona, and Serge Belongie. Building a bird recognition app and large scale dataset with citizen scientists: The fine print in fine-grained dataset collection. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 595–604, 2015. 1

[23] Lei Wang, Jianjia Zhang, Luping Zhou, Chang Tang, and Wanqing Li. Beyond covariance: Feature representation with nonlinear kernel matrices. In *Proceedings of the IEEE international conference on computer vision*, pages 4570–4578, 2015. 2

[24] Xiu-Shen Wei, Yi-Zhe Song, Oisin Mac Aodha, Jianxin Wu, Yuxin Peng, Jinhui Tang, Jian Yang, and Serge Belongie. Fine-grained image analysis with deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. 1

[25] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017. 1