Car Classification - CarNet

Rohit Patil
MEng. Robotics
University of Maryland
rpatil10@umd.edu

Anish Mitra
MEng. Robotics
University of Maryland
amitra12@umd.edu

Abstract

Object detection and classification applications have grown to become one of the most widely used application of machine learning, neural networks, and deep learning. In this project, we aim to develop a model that can identify the brand and model of cars, which can be used in multiple fields of object detection. Learning and prediction are performed on the Stanford Cars Dataset (SCD), which contains numerous images of cars with their respective labels of the brand, the model name, and the year of release. The project aims to create a new model by meshing ML-Decoders for classification with a family of models as the backbone network given by EfficientNet, which addresses the challenges of scaling up the model to any given target resource constraints.

1. Introduction

Ever increasing population of cars on road, including the introduction of new car brands and new car models in each brand calls for efficient ways of car brand/model classification. Applications vary from police surveillance, emergency response and accurate classification problem handed over by higher-level object classification ML models. In addition to this, with the increase in surveillance technology available for security and protection via CCTVs, drones, and even smartphones, it is important to have accurate and mobile applications to identify different brand and makes of cars. Hence, we see a requirement of a neural network that can perform high level classification, but be trained and run on systems with less powerful hardware. In this project, we have implemented an amalgamation of different methodologies by extracting the classification block from state of the art methods and fusing it with EfficientNet, where parameters can be modified to run the neural net on varying levels of system hardware available to us. We train and test on the Stanford Cars Dataset (hereby referred to as the *SCD*). The first challenge is to ensure that the fusion between the two is correct and there is no loss of data. The next challenge is to tune the parameters of the classifier block to account for the parameters and the output dimensions of the EfficientNet. The results do not beat the state-of-the-art model in terms of accuracy, but they can be trained and run on systems with lower CPUs/GPUs performace, unlike the SOTA models.

2. Related work

TResNet-L+ML-Decoder is considered as a SOTA for SCD Dataset. [1] ML-Decoder, as opposed to global average pooling, allows for better use of spatial data by predicting the existence of class labels through queries. The ML-Decoder is extremely effective and scales well to thousands of classes thanks to a redesign of the decoder architecture and the use of a novel group-decoding approach. ML-Decoder consistently offers a superior speed-accuracy trade-off than employing a larger backbone. Using the high-quality training code recommended in the A2 configuration, the model used the ImageNet dataset to test the effectiveness of the ML-Decoder for single-label classification. Additionally, ML-Decoder provides a better speed-accuracy trade-off than utilizing GAP (Global Average Pooling) with a bigger backbone. In a highly optimized single-label training environment, where ResNet50 obtains a baseline score of 79.7%, ML-Decoder may serve as a drop-in replacement for the GAP-based classification head and yet provide an additional boost, proving the usefulness and versatility.
EfficientNet architecture provides the capability to perform 3 modes of model scaling but keeping the number of parameters at a minimum at the same time. A compound scaling method [2] is proposed which allows uniform scaling all dimensions of depth, width and resolution (as shown in fig. 1) yet still having better accuracy and efficiency than previous ConvNets.

2.1. State of the art/baseline implementation

Table 1 gives the performance benchmarks of (State-of-the-art) SOTA models implemented to classify Stanford Car Dataset.

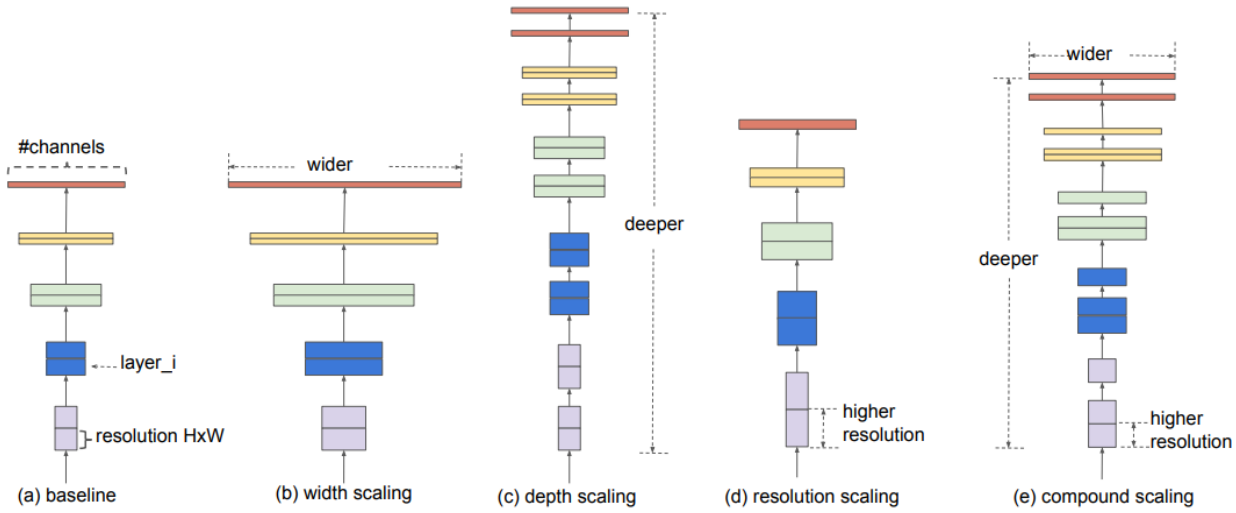

Figure 1. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.[2]

Table 1. Benchmarks with state-of-the-art models

Model/Method	Stanford Car Dataset
	Accuracy
TResNet-L+ML-Decoder	96.41
EffNet-L2	95.13
API-Net	95.35
DenseNet161+MM+FRL	95.2
ELoPE	95.0
DF-GMM	94.8

3. Dataset

The Stanford Cars Dataset (SCD) or commonly known as *The Cars* dataset, is a collection of 16,185 images of cars that can be classified into 196 different classes. The dataset is divided into a training set of 8,144 images and a test set of 8,041 images, hence creating a rough 50-50 split for each class [3].

3.1. Data Preprocessing

The nature of the image data was such that there was no data removal or imputations required since there were no missing values in the dataset. We were able to extract the publicly available data from a *.tgz* file which contained the array of class names, one for each class. Another file contains the annotations for each image, which are essentially the bounding boxes for the area of interest in each image.

3.1.1 Data Augmentation

After the data has been extracted we perform randomized probabilistic transformations such as:

- Horizontal flip
- Rotation
- Random Affine Transform
- Sharpness adjustment
- Gray-scale conversion
- Perspective transform

3.1.2 File storage

The images are stored in separate folders with each folder labeled with the name of the respective classes. This allows easier access to data during training, validation, and testing by PyTorch libraries such as DataLoader and datasets.

3.2. Exploratory Data Analysis

As mentioned earlier, since there are no missing data, there is no need for imputing or deletion of data. After extracting the classes from the array, which were obtained in the form of a CSV file stored the *.tgz* file, each classname was then stripped apart to obtain the labels for each class. Each class contained 3 main labels namely, the brand, the model and the year of manufacturing. This proved to be particularly tricky, as some brand names would either be a single string, or could be a concatenation of two strings, which could be discerned by a human, but not by a program. For example, the difference between BMW and "Aston Martin" can be identified by a human, but by splitting the labels by spaces, Aston Martin is viewed as two different labels in the same class.

3.3. Input/Output of the model

3.3.1 Input

Input can be images of cars with various camera angles and scales. Ground truth or labeling for images is accessed through the Kaggle repository.

3.3.2 Output

Output from the neural network is a class label containing car brand, model and make year from total of 196 class labels. We also overlay actual class and predicted class labels over the images passing through a fully trained model.

3.4. Evaluation metric

The problem we are trying to solve, in the most fundamental sense, is a multi-class classification problem. For a binary classification problem, we have different metrics such as accuracy, precision, recall, ROC Curves to determine the performance of the model. For our application, we first create a confusion matrix that encompasses all the classes along with the number of True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN). Hence, we will end up with a *n x n* matrix, where 'n' is the number of classes (in our case, 196). From this we can calculate the Recall and Precision, which are given by:

$$Recall = \frac{TP}{TP + FN}$$
$$Precision = \frac{TP}{TP + FP}$$

From this we finally calculate the F1 score that is our main metric of determining the evaluation of performance of our model, which is given by:

$$F_1 = \frac{2}{recall^{-1} + precision^{-1}} = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

If we choose to penalize the False Negatives more i.e. we want to give more importance to Recall over Precision, we have a more generalized formula such that Recall is considered β times as important as precision [4]:

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$
$$F_\beta = \frac{(1 + \beta^2) \cdot TP}{(1 + \beta^2) \cdot TP + \beta^2 \cdot FN + FP}$$

In our case, for each epoch, we will calculate the F1 score for each class, and then average it out to calculate the F1 score for that particular epoch [5] as shown below. Based on this we can determine the epoch with the best accuracy and extract the parameters for that epoch.

$$F1(epoch_i) = \frac{1}{N} \sum_{x=x_i}^{x_n} F1(class = x)$$

4. Methods

4.1. Classifier Selection

The state-of-the-art (SOTA) model that achieves high accuracies as mentioned in Ridnik et al. [1], uses a backbone of TResNet-L with a MLDecoder classifier head to perform different types of classification, as shown in fig. 2.

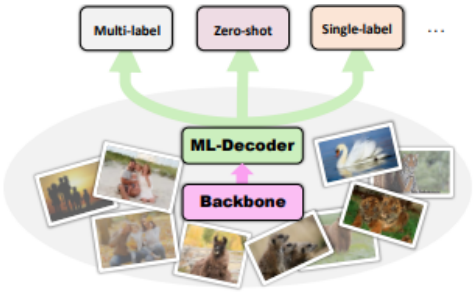

Figure 2. Simplified Representation of SOTA car classification model [1]

As shown in 1 the model with TResNet-L with ML-Decoder achieves an extremely high accuracy of 96.41% on the SCD. However, there are two main reasons for the high accuracy of this model:

- **Additional Training Data:** The work adds data from ImageNet and other datasets to the training data set, thereby increasing the number of features and fine-grained details that can be learned by the model. This significantly increases the accuracy while testing due to the increased number of minute features learned by the model.

- **Large Backbone Size:** TResNet-L is an extremely large backbone neural network that contains a significantly large number of parameters to be learned. Owing to this, TResNet-L cannot be trained on systems with lesser V-RAMs or lower GPUs and predominantly requires high-level computing powers.

Since our aim is to provide the best classification, but also ensure that the model can be trained on lower-end systems, we extract only the ML-Decoder block for classification. Using this ML-Decoder, we perform multiclass classification and omit zero-shot and single-label classification as it is irrelevant to our application.

4.2. Backbone Network Selection

The TResNet-L backbone being too large and computationally heavy, prompted the requirement for a more computationally robust and portable network. For our model to be trained and run on various systems with different hardware resource constraints, it was imperative that the backbone had the ability to be scaled without significant compromise in accuracy. It is also important to have lesser Floating Point Operations per Second (FLOPS) to ensure quicker inference. Lesser FLOPS results in lower requirements of computational power from the hardware. EfficientNet is a neural network that can be scaled according to the capabilities of the hardware system and the requirements of the application at hand. With the help of a compound scaling constant ϕ, the network width $w = \beta^{\phi}$, the network depth $d = \alpha^{\phi}$ and resolution $r = \gamma^{\phi}$ are scaled uniformly in a principled way [2]. Varying values of α, β, γ results in the creation of 8 networks ranging from EfficientNet-B0 to EfficientNet-B7. Fig. 3 compares the performance of EfficientNet architectures with existing ConvNets based on metrics such as Top-1 and Top-5 Accuracies[6], number of parameters to be trained[7], FLOPS etc. In this comparison all the models are trained and tested on ImageNet. [8][9][10]

Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPs	Ratio-to-EfficientNet
EfficientNet-B0	**77.1%**	**93.3%**	**5.3M**	**1x**	**0.39B**	**1x**
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M	4.9x	4.1B	11x
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M	2.6x	3.5B	8.9x
EfficientNet-B1	**79.1%**	**94.4%**	**7.8M**	**1x**	**0.70B**	**1x**
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M	7.6x	11B	16x
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M	4.3x	6.0B	8.6x
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M	3.0x	5.7B	8.1x
Xception (Chollet, 2017)	79.0%	94.5%	23M	3.0x	8.4B	12x
EfficientNet-B2	**80.1%**	**94.9%**	**9.2M**	**1x**	**1.0B**	**1x**
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M	5.2x	13B	13x
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M	6.1x	13B	13x
EfficientNet-B3	**81.6%**	**95.7%**	**12M**	**1x**	**1.8B**	**1x**
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M	7.0x	32B	18x
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M	7.7x	35B	19x
EfficientNet-B4	**82.9%**	**96.4%**	**19M**	**1x**	**4.2B**	**1x**
SENet (Hu et al., 2018)	82.7%	96.2%	146M	7.7x	42B	10x
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M	4.7x	24B	5.7x
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M	4.6x	23B	5.5x
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M	4.5x	23B	6.0x
EfficientNet-B5	**83.6%**	**96.7%**	**30M**	**1x**	**9.9B**	**1x**
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
EfficientNet-B6	**84.0%**	**96.8%**	**43M**	**1x**	**19B**	**1x**
EfficientNet-B7	**84.3%**	**97.0%**	**66M**	**1x**	**37B**	**1x**
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-

Figure 3. All EfficientNet models are scaled from baseline EfficientNet-B0 using different compound coefficient ϕ [11]. scaled EfficientNet models consistently reduce parameters and FLOPS by an order of magnitude (up to 8.4x parameter reduction and up to 16x FLOPS reduction) than existing ConvNets. [12]

For representation purposes, fig. 4 shows the EfficientNet-B0 baseline network. Our application deals with using ML-Decoder for enhanced classification, hence we can remove the 9th stage of the baseline network (which is the same operator for B0-B7 networks).

Our application deals with using ML-Decoder for enhanced classification, hence we can remove the 9th stage of the baseline network (which is the same operator for B0-B7 networks).

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Figure 4. EfficientNet-B0 baseline network

4.3. CarNet

4.3.1 ML Decoder design

Multi-label classification also makes use of GAP-based heads. The use of average pooling, however, may not always be the most effective due to the need to recognize several objects of various sizes and positions. Since GAP-based classification heads have a fixed spatial pooling cost, they are intuitive, effective, and scale well with the number of classes. Nevertheless, they deliver poor results. While attention-based classification heads do enhance outcomes, they are frequently expensive, even for datasets with a limited number of classes, and virtually impractical for extreme classification scenarios. The original transformer-decoder served as the foundation for the ML-Decoder design, which has undergone two substantial revisions that greatly increase its scalability and efficiency. By removing the redundant self-attention block, it first transforms the quadratic reliance of the decoder on the number of input queries into a linear dependence. Second, ML-Decoder employs a novel group-decoding technique in which a fixed number of queries—instead of one per class—are interpolated to the actual number of classes by means of a new architectural component called group fully-connected. ML-Decoder, which uses group-decoding, also benefits from a fixed spatial pooling cost and scales well to thousands of classes. Its generalizability to previously unseen classes is enhanced by complementary query augmentation.

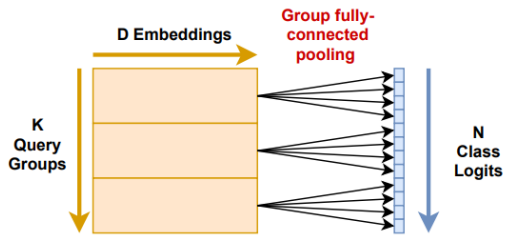

Figure 5. Group fully connected layer of ML-Decoder [1]

Group factor is defined by $g = \frac{N}{K}$. Logit L_i is generated

by group fully-connected using following operation:

$$L_i = (W_k \cdot Q_k)_j$$
$$\text{where: k} = i \textbf{ div } g, \text{j} = i \textbf{ mod } g$$

$W_k \in \mathbb{R}^{g \times D}$ is the k^{th} learnable projection matrix, and $Q_k \in \mathbb{R}^D$ is the k^{th} query. G_q are the input queries, for which flow of ML-Decoder with group decoding is given by:

$$\text{cross-attn:} \quad G_{q1} \longleftarrow \text{MultiHeadAttn}(G_q, E, E)$$
$$\text{feed-forward:} \quad G_{q2} \longleftarrow \text{FF}(G_{q1})$$
$$\text{group FC:} \quad \text{Logits} \longleftarrow \text{Group-FC}(G_{q2})$$

4.3.2 MBConv block design

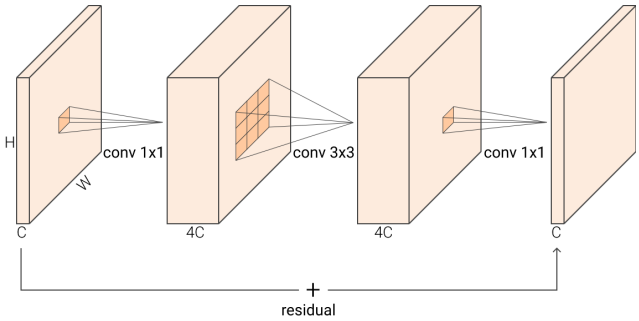

Figure 6. Inverted residuals blocks [13]

MBConv is a Inverted Linear BottleNeck layer with Depth-Wise Separable Convolution, which is a building block of MobileNetV2 network[14]. Inverted residual blocks are a type of building block used in some neural network architectures for image classification tasks. They are called "inverted" because they use a different structure than traditional residual blocks, with the shortcut connections (also called skip connections) coming after the block's layers instead of before them. Inverted residual blocks are used in MobileNetV2 to replace the MBConvBlock blocks used in MobileNetV1, and they are designed to be more efficient and effective at extracting features from the input image.

The structure of an inverted residual block typically consists of one or more depthwise convolutions, followed by one or more pointwise convolutions, and finally a skip connection that concatenates the output of the block with the input to the block. This structure allows the block to effectively capture both spatial and channel-wise relationships in the input image, resulting in improved performance on image classification tasks. Depth-Wise Separable Convolutions use a method to split a typical 3x3 convolution into two convolutions in order to reduce the number of parameters. One applies a single 3x3 filter to each input's channels, whilst the other applies a single 1x1 filter to all channels.

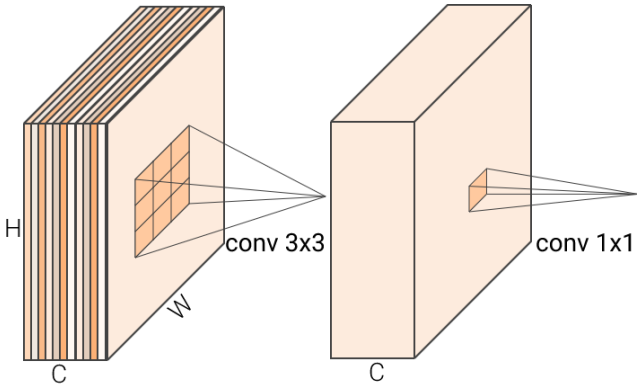

Figure 7. Depth-wise separable convolution [13]

4.3.3 CarNet : EfficientNet + ML Decoder

EfficientNet is a convolutional neural network (CNN) architecture that is designed to be efficient and effective at image classification tasks. The architecture of EfficientNet is based on a combination of three key design principles: scaling, depthwise separable convolutions, and compound scaling. The scaling principle is used to adapt the network architecture to different input image sizes. This allows the network to be used on images of different sizes without needing to design separate network architectures for each size. The depthwise separable convolution principle is used to improve the efficiency of the network. Depthwise separable convolutions are a type of convolutional layer that decomposes a standard convolution into two separate operations: a depthwise convolution and a pointwise convolution. This allows the network to use fewer parameters and computational resources while still effectively extracting features from the input image. EfficientNet extensively uses MBConv blocks to extract features of the images, which is explained earlier. We combine this capability to fuse with ML Decoder so as to obtain higher accuracies in predicting multilabel & multiclass classification as seen in SCD. We choose the EfficientNet B5 model as our backbone, which has approximately 30 million parameters and ML Decoder is connected by 2048 inputs from later, and has 7 million parameters. The selection of the backbone network was based on available computing power and VRAM in GPU's.

5. Experiments

5.1. CarNet(with EfficientNet B3 as the backbone)

CarNet(B3) was trained on a local laptop machine whose specifications are Intel 12th Gen 12800H 16GB RAM and Nvidia RTX 3060 6GB VRAM. CarNet(B3) has around 19 million trainable parameters, takes input images of resolution 300 x 300, and batch-size 16. Trained with a static learning rate of 0.0001 for 20 epochs to achieve 86% of validation accuracy, which took around 1 hour 14 mins to

Table 2. CarNet Layers Architecture using EfficientNet-B5 backbone with MLDecoder Classifier

Stage	Operation	Resolution	Channels	Parameters
1	Input	456 x 456	3	-
2	Zero-padding	457 x 457	3	-
3	Conv2dStaticSamePadding	228 x 228	48	2,050,296
4	BatchNorm2d	228 x 228	48	96
5	MemoryEfficientSwish	228 x 228	48	-
6	(MBConvBlock) x 3	228 x 228	24	2,940 + 1,206 + 1,206
7	(MBConvBlock) x 5	114 x 114	40	13,046 + (27,450) * 4
8	(MBConvBlock) x 5	57 x 57	64	37,098 + (73,104) * 4
9	(MBConvBlock) x 7	29 x 29	128	91,664 + (256,800) * 6
10	(MBConvBlock) x 7	29 x 29	176	306,048 + (496,716) * 6
11	(MBConvBlock) x 9	15 x 15	304	632,140 + (1,441,644) * 8
12	(MBConvBlock) x 3	15 x 15	512	1,792,268 + (3,976,320) * 2
13	Identity	15 x 15	512	-
14	Conv2dStaticSamePadding	15 x 15	2048	1,048,576
15	BatchNorm2d	15 x 15	2048	4,096
16	MLDecoder (TransformerDecoder)	16 x 768	100	7,319,748
17	CarNet Output Layer	1 x 196	-	-
Total Parameters = **37,709,532**				

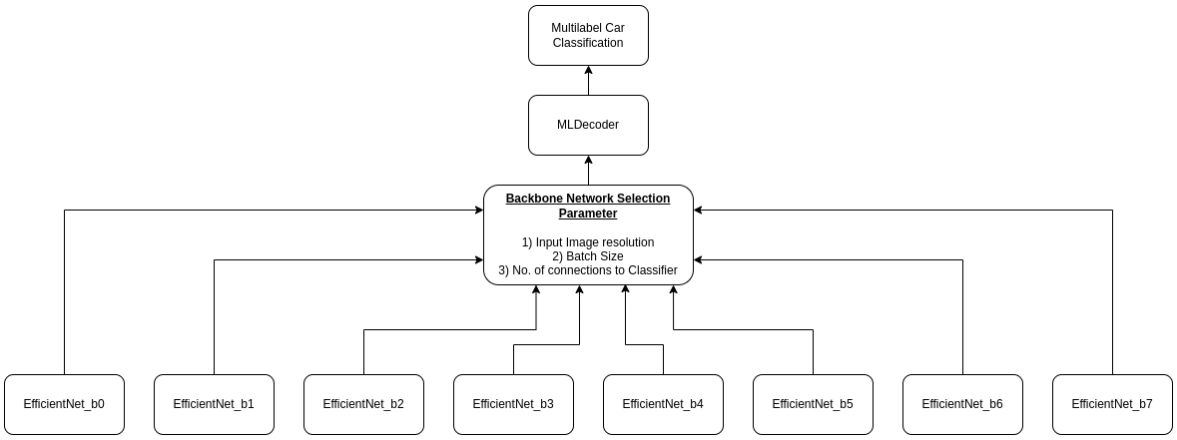

Figure 8. Simplified representation of CarNet

complete. A dropout of 0.3 for the backbone and 0.1 for the ML-Decoder classifier was employed. Executed multiple training of the network while hyper-tuning parameters that include input image resolution, batch size, learning rate, dropout rate, loss with regularization, optimizer selection, dataset augmentation transforms, and activations functions. As seen in fig. 9 initial validation accuracy is higher than training accuracy due to well well-augmented dataset and combine dropout rate. After epoch 7, accuracies start diverging and plateauing after a few epochs. Corresponding behaviour can be seen for the validation loss across the epochs as shown in fig. 10.

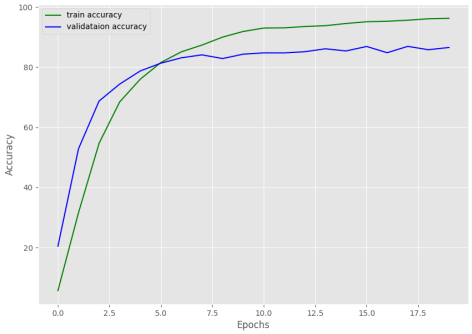

Figure 9. CarNet(B3) Validation accuracy vs epochs

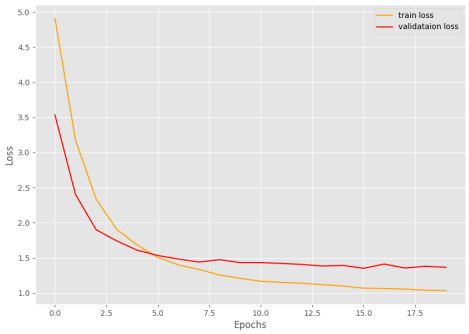

Figure 10. CarNet(B3) Validation loss vs epochs

5.2. Upscaling network - CarNet(with EfficientNet B5 as the backbone)

CarNet(B5) was trained on Google Colab Pro using premium GPUs on the cloud, namely Nvidia V100 32GB VRAM. CarNet(B5) has 37.7 million trainable parameters, takes input images of resolution 456 x 456, and batch-size 16. A dropout of 0.4 for the backbone and 0.1 ML-Decoder classifier was employed. Used label smoothing cross-entropy loss (mean reduction) shown in eq. 1, AdamW optimizer with weight decay of 0.2, and step learning rate reduction methods to enhance performance and accuracy. Weight decay acts as a regularization parameter. Pytorch development framework was utilized to develop all the network layers and training scripts [15] [16] [17] [18]. The network was trained with a step learning rate starting with 0.0001(gamma reduction 0.1 for every 6 epochs, as seen in fig. 13) for 16 epochs to achieve 90.9% multi-class validation accuracy (as seen in fig. 11) and 0.91 multi-class F1 score (as seen in fig. 14).

$$\text{Label smoothing cross entropy loss} = (1 - \exp)ce(i) + \exp \sum \frac{ce(j)}{N} \quad (1)$$

where $ce(i)$: Standard cross-entropy loss ; N: Number of classes

For representation purposes, after performing testing of the model, we have taken a random sample from each of the 196 classes to illustrate the output of the model. Fig. 15 shows the 14 x 14 grid of a random sample prediction taken from each class. Correctly predicted samples have all the text on the image in green, whereas images with the wrong predictions have “predicted label” displayed in red colour.

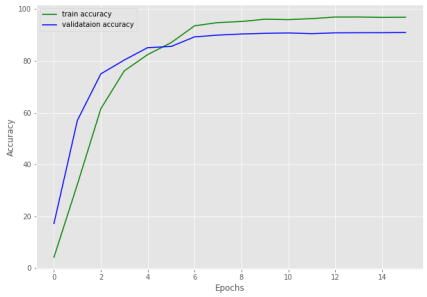

Figure 11. CarNet(B5) Validation accuracy vs epochs

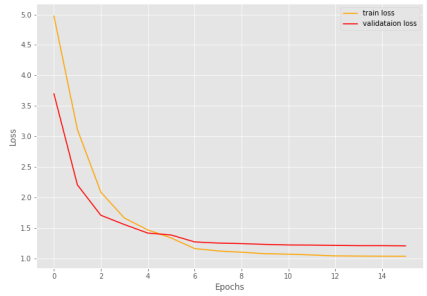

Figure 12. CarNet(B5) Validation loss vs epochs

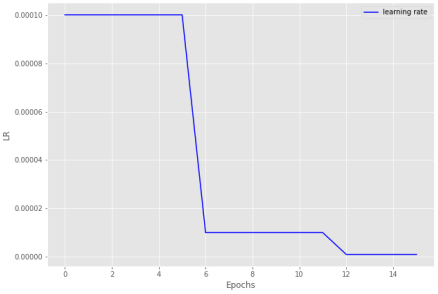

Figure 13. CarNet(B5) Learning rate vs epochs

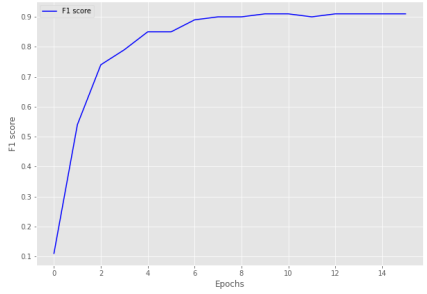

Figure 14. CarNet(B5) F1 score vs epochs

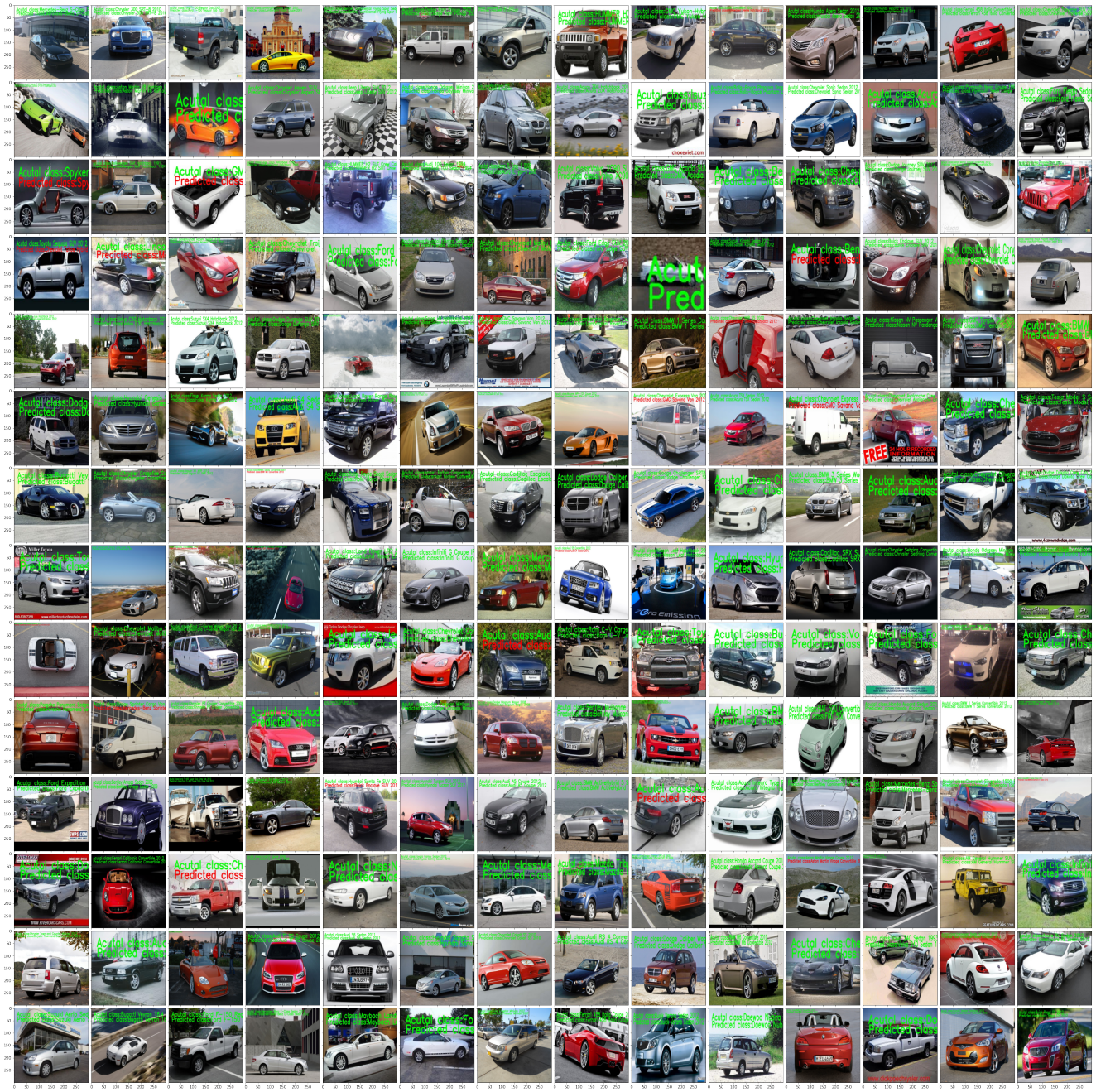

Figure 15. Prediction/Classification on a random image of each class using CarNet(B5)

6. Limitations

Although our method opens up the avenue of performing car brand detection by highly motivated peers of the deep learning community who do not have access to supercomputers, there is a compromise on the final accuracy of the model. In our case, where we swap the backbone network using transfer learning, only a few hyperparameters can be fiddled around with to achieve better accuracy due to the limitation of the hardware. The unavailability of more VRAM even prevents the usage of higher EfficientNet backbone architectures, which require higher GPU specifications. As mentioned in the previous section, training and analyses was performed locally on a machine laptop with Nvidia RTX 3060 GPU, which is already powerful and expensive hardware for many to procure. Despite these specifications, we were able to train the CarNet model with only EfficientNet-B3 as the backbone to achieve 86% validation

accuracy. If the accuracy was to be improved with the same system, tuning of more hyperparameters would be required. This proves extremely cumbersome and is not the optimum approach to finding the best model. For each small change in hyperparameter values, training times are very long. This prevents us from using the traditional method of creating a grid of all possible combinations of hyper parameter values to eventually find the best model with highest accuracy.

7. Challenges Faced

- Fusing two different deep learning architectures proved quite challenging, especially given the volume of data that we had to work with and the complex features/dimensions we had to take into consideration.
- Hardware limitations prevented us from tuning many hyperparameters at once, but instead focusing on tuning only a few parameters at a time.
- Long wait times due to hardware limitations, preventing us from performing many forward passes for different parameter values
- To work around hardware limitations, we tried making use of the free Google Colab credits, but due to university administrative issues, we were not able to link those credits to the projects.
- **Going above and beyond:** We purchased Google cloud credits to ensure that we have extra compute credits to access premium GPUs. This allowed us to perform scaling of our model and note the comparisons between using different EfficientNet backbone architectures. We were determined to provide satisfactory results, even if it meant compromising on deadlines.

8. Conclusion

In our project, we tried to fuse two different deep learning architectures taken from the classification block of TResNet-L + MLDecoder implementation that achieves 96.41% accuracy on ImageNet, and the backbone of EfficientNet architectures that require training of significantly lesser number of parameters to predict the brand, model and year of cars from the Stanford Cars Dataset. The idea behind this was to give rise to a scalable model that could be run on different systems which have varying availability of hardware resources such as VRAM, GPUs, CPUs etc. We were able to generate a model with a backbone (EfficientNet-B3) that trained 19 million parameters on a local laptop machine, providing an accuracy of 86%. To further demonstrate the scalability, we modified the model with a higher end backbone (EfficientNet-B5) that trained 37 million parameters on Google Colab server with premium GPUs to provide a model with accuracy of 90.9%. However, due to lack of hardware resources, we could not perform extensive hyperparameter tuning to achieve breakthrough accuracies. With the hardware available to us, using everyday laptops, we were able to train a model that cannot usually be trained on the same system using other wider or deeper neural network architectures.

References

[1] Tal Ridnik, Gilad Sharir, Avi Ben-Cohen, Emanuel Ben-Baruch, and Asaf Noy. Ml-decoder: Scalable and versatile classification head. *arXiv preprint arXiv:2111.12933*, 2021.

[2] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. 2019.

[3] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[4] BEXGBoost. Comprehensive guide on multiclass classification metrics, Jun 2021.

[5] Baeldung. F-1 score for multi-class classification, Nov 2022.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[7] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2016.

[8] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[9] François Chollet. Xception: Deep learning with depthwise separable convolutions, 2016.

[10] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning, 2016.

[11] Saining Xie, Ross Girshick, Piotr Dollar, Z. Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. pages 5987–5995, 07 2017.

[12] Xingcheng Zhang, Zhizhong Li, Chen Change Loy, and Dahua Lin. Polynet: A pursuit of structural diversity in very deep networks. pages 3900–3908, 07 2017.

[13] Francesco Zuppichini. Residual, bottleneck, inverted residual, linear bottleneck, mbconv explained, Oct 2021.

[14] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. 2018.

[15] mrT23 Alibaba-MIIL. Alibaba-miil/ml$_decoder$: $Officialpytorchimplementationof"ml-decoder$: $Scalableandversatileclassificationhead"(2021), Apr2022.$

[16] Matkalinowski. Matkalinowski/efficientnet-on-stanford-cars-dataset: My implementation of efficientnet's with benchmarks on stanford cars dataset., Nov 2021.

[17] Luke Melas. Lukemelas/efficientnet-pytorch: A pytorch implementation of efficientnet and efficientnetv2 (coming soon!), Apr 2021.

[18] Jesús Utrera. Stanford car dataset by classes folder, Jul 2018.