Utilizing ResNet in a car classification task

Final project for *Neural Network and Deep Learning PhD. Course*

Andrea Iommi - 690005 (andrea.iommi@phd.unipi.it)

August 19, 2025

1 Project description

This report presents the work for the final project for the *Neural Network and Deep Learning* course. The project involves developing a Convolutional Neural Network (CNN) to recognize a set of car models, particularly brands and types. I conducted various experiments with different parameters and model architectures to determine the optimal configuration for solving the task. Additionally, I explored adversarial attacks on the trained models and developed a web application to make the model's training interactive. Experimental results highlight how the ResNet-18 is complex enough to solve the task well, but with an imperceptible input variation (adversarial example), its performance drops

I adopted the PyTorch[1] Python framework to solve the task. PyTorch is a framework for building deep learning models. Written in Python, it's relatively easy for most machine learning developers to learn and use. It is distinctive for its support for multiple GPUs and its auto-differentiation technique, which enables computation graphs to be modified on the fly. This makes it a popular choice for fast experimentation and prototyping (for further details, see [2]).

The work is organized as follows: in the section 2, I outline the architecture deployed by briefly introducing the latent concepts behind the models. Then, in section 3, I propose a variation of the proposed dataset. In section 4, I describe the training pipeline, motivating the choices embraced, and I discuss the results obtained. An adversarial attack is illustrated in section 5. Finally, in the section 6, I propose a minimal web interface for training and utilizing models with a graphical UI.

2 Architectures utilized

The architecture used for this project was the *ResNet* [5]. In particular, ResNet-18 and ResNet-34. They are small and medium models of the ResNet family and are widely used in medical or other specialized domains [1, 6, 10]. For

[1] https://pytorch.org/

[2] https://www.nvidia.com/en-us/glossary/pytorch/

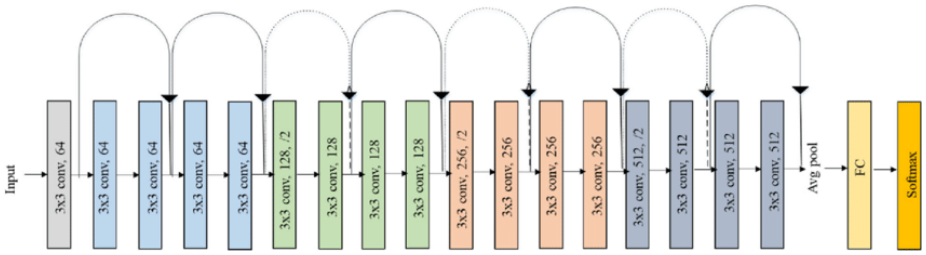


Figure 1: ResNet-18 Architecture

example, Reset-18 contains 18 trainable blocks (see figure 1) and is built using *residual blocks*, allowing the model to learn identity mappings.

The principle of the residual block is to allow the network to learn not the whole mapping of input to output, but the residual (i.e., the input minus the desired output). These skip connections prevent network degradation (i.e., the gradient vanishing) and enable deeper architectures to be learned by improving gradient flow. Although not very deep, ResNet-18 and ResNet-34 achieve a good trade-off between accuracy, efficiency, and computational cost. These capabilities allow the model to be integrated extensively in many applications that require fast training and inference.

3 Stanford Cars dataset

The most used dataset for car classification is the *Stanford Cars dataset* [8] available on Kaggle. It contains $16,185$ images of 196 classes of cars. The data is split into $8,144$ training images and $8,041$ testing images, where each class has been split roughly $50-50$. However, it was chosen to adopt a more recent variation [3]. The last one adopts the following updates concerning the original dataset: *(i)* the train set is further split into training and validation, in a $2:1$ ratio, *(ii)* the training and validation images are augmented using *albumentations* [2]. The amount of augmentation is calibrated so that in the final output, we have roughly an equal number of cars per class. In particular, the argumentations utilized are: *Horizontal Flipping, Random Brightness Contrast, Random Gamma variation, Rotating, RGB Shift, Adding Gaussian Noise, Channel Shuffling, Grayscaling, and Blurring.* This variant comprises $86,430$ training images, $27,977$ validation images, and 8041 test images, totalling $122,448$ labelled examples. Figure 2 shows an example of a car in the training set.

The dataset contains images of varying dimensions, so a preprocessing step was required to make the dataset applicable. During the training phase, each image underwent a rescaling operation. The utilized model imposes a specific dimension on the input. ResNet requires a $400 \times 400 \times 3$ input tensor, repre-

[3]The dataset can be downloaded from here

Figure 2: On the right, an argumented (rotated) Audi S4 Sedan 2012, on the left, an Audi R8 Coupe 2012 extracted from the training set.

senting the width, height, and number of channels, respectively. In addition to making the training faster, a normalization was applied (i.e., $\mu = 0.5$ and $\sigma = 0.5$) by exploiting the built-in method in PyTorch.

4 Experiments

4.1 Training configurations

To train our image classifier, I adopted a standardized pipeline across all experimental configurations. Specifically, I investigated the ResNet-18 and ResNet-34 [5], both of which were pretrained on ImageNet [3] and fine-tuned on our dataset. I also evaluated the performance of these models in an unpretrained setting.

All models were trained using the *Adam* [7] with Weight decay optimizer (`AdamW` in PyTorch), which integrates the weight decay in the Adam optimizer during the gradient updating. Empirical studies show an improved regularization compared to the standard Adam optimizer [9] (old-but-gold paper).

The initial learning rate was scheduled using a *Cosine Annealing* Learning Rate Scheduler (`CosineAnnealingLR`), which gradually reduces the learning rate following a cosine curve, encouraging more stable convergence in later epochs.

Each model was trained for a maximum of 15 epochs. But, to prevent overfitting, I implemented early stopping with a patience of 5 epochs. The training was interrupted if the validation performance did not improve for 5 consecutive epochs. However, in the experimental part (see section 4.3), I did not encounter this scenario.

Regarding the loss function, a standard cross-entropy loss was adopted (`CrossEntropyLoss`), suitable for multi-class classification tasks. I assessed the model's performance with a classical technique: *hold-out*. I trained the model with the training set, then I selected the best parameter configuration on the validation set. The model achieving the best validation performance was selected for final evaluation on the test set. The metric integrated in the

evaluation was the `f1_score`[4]. The rationale behind this choice is that F1 is a metric that evaluates the overall model's performance, taking into account both precision and accuracy. By assigning a low metric value if the model fails to classify images that belong to minority classes correctly.

4.2 Architectures

In the experiments, I investigated the performance of the ResNet architecture by varying the model settings. I am interested in investigating two phenomena: the impact of pretraining and the frozen weights. Therefore, to lead the experiment, I chose 4 different types of configuration: *(i)* The ResNet without the pretraining, i.e, with a random initialization, *(ii)* a pretrained model, *(iii)* a pretrained model with the first 6-th layer frozen from the gradient update, and finally *(iv)* a pretrained model with the first 8-th layer frozen. In detail, with *(i)* and *(ii)*, I evaluated how much the pretraining impacts the model performance. With *(iii)* and *(iv)*, I investigated the impact of freezing the layers. The 6-th frozen means only half of the convolutional layer can be updated. Instead, 8-th layers means that all convolutional layers are fixed (fixed feature extractor) and only the last layer (classifier) was updated. These combinations were explored for both architectures, i.e, ResNet-18 and ResNet-34.

Conf.	*Description*	*Intuition*
(i)	No pretrained	Random initialization
(ii)	Pretrained on ImageNet [3]	Task specialization
(iii)	Pretrained with first 6-th frozen layers	Half backpropagation
(iv)	Pretrained with first 8-th frozen layers	Frozen feature extractor

Table 1: Model's parameters

4.3 Results

Figures 3 and 4 illustrate the experimental results. The charts outline two distinct clusters of configurations: group *(i)* and *(iv)*, and group *(ii)* and *(iii)*, which exhibit similar behaviours within their respective groups.

The first group, *(i)* and *(iv)*, achieves poor performance (see Table 2 for more details). I hypothesize two different causes: for the configuration *(i)*, a random initialization of the weights necessitates more epochs to get good results. The feature extractors, i.e., the convolutional layers, require a huge amount of data to become discriminative and thus learn patterns. The configuration *(iv)* was limited to fine-tune only the classifier, i.e, the last layer, and this converged to a bad performance. Even if the model was pretrained (and thus the feature extractors had learnt patterns), the overall network remains too general and incapable of specializing toward a specific task.

[4]https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

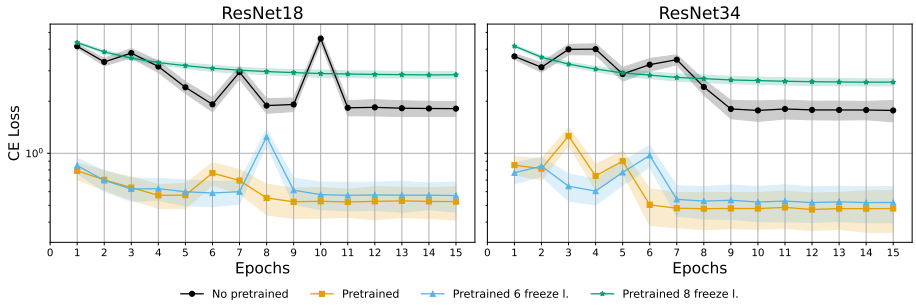


Figure 3: Cross-entropy loss (CE Loss) in the **validation set** through the epochs in different model configurations. The lines identify the mean over all batches, and the shadows represent ± 1 standard deviation.

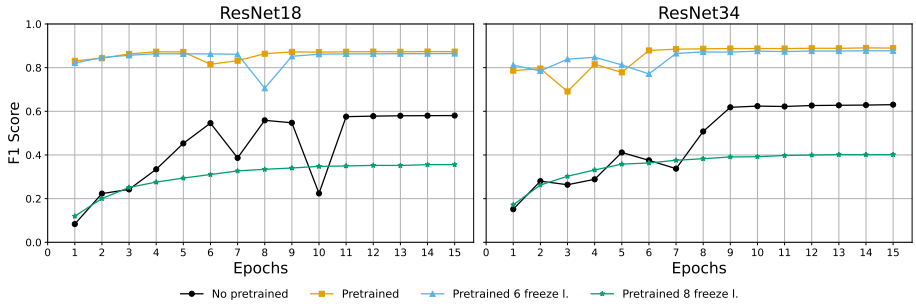


Figure 4: F1 score in the **validation set** through the epochs in different model configurations.

Conversely, group *(ii)* and *(iii)* performed significantly better. These configurations involved freezing only the first six layers, which appears to have little negative impact on the model's ability to adapt to the task. One possible explanation can be attributed to the fact that the gradient in the initial layer becomes too small to impact, even if the residual blocks and skill connections attempt to mitigate this.

Concerning the model's architecture, no significant performance difference was observed between ResNet-18 and ResNet-34. The smaller network was sufficient for solving the classification task. Nonetheless, the highest F1 score overall was achieved by the ResNet-34 model with pretrained weights and no frozen layers (see Table 2 for more details). Table 3 instead shows the model performance in the test set for ResNet-34.

Name	*TR loss* $(10^{-4})\downarrow$	*TR F1* $\uparrow$	*VL loss* $\downarrow$	*VL F1* $(10^{-1})\uparrow$
R18 No pretrained	7.76	0.99	1.81	5.80
R18 Pretrained	1.44	0.99	0.52	8.73
R18 Pre. 6 layer	1.57	**1.00**	0.56	8.64
R18 Pre. 8 layer	15398.47	0.75	2.84	3.55
R34 No pretrained	4.38	0.99	1.76	6.30
R34 Pretrained	0.97	0.99	**0.47**	**8.89**
R34 Pre. 6 layer	**0.77**	**1.00**	0.51	8.77
R34 Pre. 8 layer	12025.17	0.82	2.57	4.01

Table 2: The results of training different ResNet architectures and model hyperparameters.

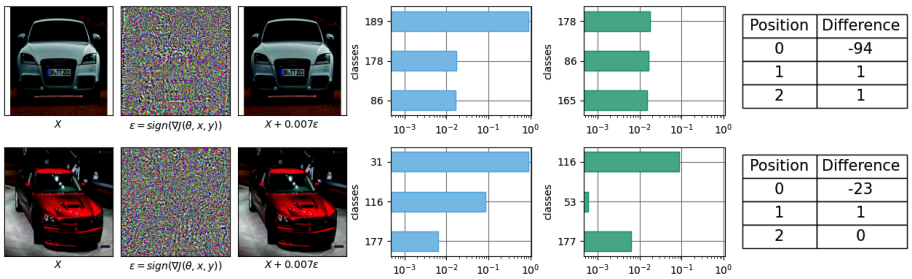
5 Adversarial attack

The *Adversarial attacks* represent a set of techniques adopted to induce the classifier to misclassify by introducing a little variation in the original input[4]. I do not go into detail in this project, but I provide an example of how an adversarial attack can be performed in our car classifier. I adopted the simplest one: *Fast Gradient Sign Method* (FGSM). FGSM works by using the gradients of the neural network to create an adversarial example. Since the method has access to the model's gradients belongs to the white-box attack. For an input image, the method uses the gradients of the loss with respect to the input image to create a new image that maximizes the loss. Mathematically speaking, we have:

$$x_{adv} = x + \epsilon \text{ sing}(\nabla_x J(\theta, x, y)) \tag{1}$$

It is worth noting that the gradients are taken with respect to the input (i.e., the image) and not the parameters θ. This is done because the objective is to create an image that maximises the loss. A method to accomplish this is to find how much each pixel in the image contributes to the loss value and add a perturbation accordingly. This works quickly because it is easy to get how each input pixel contributes to the loss by using the chain rule and finding the required gradients.

Figure 5 is a demonstration of adversarial example generation applied to ResNet-18. By adding a small tensor whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, I can change ResNet's classification of the image. For each row, the first image represents the original image, the second is the noise, and the third is the adversarial example. Then we have two bar charts: the first represents the top-3 class distribution for the original image, and the second one identifies the top-3 classes for the adversarial example. Finally, there is a table that identifies the position change of the top-3 classes predicted with the original image. As we can see, the distribution significantly changes. The model's confidence declines with the adversarial example, leading to uncertainty in the classification. Table 3



Position	Difference
0	-94
1	1
2	1

Position	Difference
0	-23
1	1
2	0

Figure 5: Adversarial attack example. In the first row, the original image's predicted class is 189 (Audi TT Hatchback 2011) with high confidence. However, with the adversarial attack, the class 189 drops 94th positions, while the classes 178 and 86 rise by one position.

Input	F1 ↑	Accuracy ↑	Uncertain H
Original images	**0.9021**	**0.9024**	**1.4531 ± 0.8601**
Adversarial images	0.1346	0.1315	2.6242 ± 0.8007

Table 3: ResNet-34's performance on the **test set**, with and without the adversarial attack. We also calculated the uncertainty utilizing the Shannon entropy, defined as $H(\boldsymbol{x}) = -\sum_{i=0}^{|\boldsymbol{x}|} p(x_i) log(p(x_i))$.

shows the model performance of the pretrained ResNet-34 with and without the adversarial attack.

6 Web Interface

Figure 6 shows the training and inference web interfaces. It is designed to train, predict, and monitor the performance of a ResNet-18. In the project, it is applied to a car dataset, but its utilization is suitable for any image classification tasks.

Training The train interface is divided into three main sections. The top section provides users with the ability to set various hyperparameters related to the dataset, the model, and the training configuration. For instance, the user can specify the number of output classes (in our project was 196), and define the folder containing the dataset images. Additionally, the user can choose whether to use a pretrained version of ResNet-18 or to train the network from scratch. The optimization settings can also be configured, including the optimizer, the learning rate, and the batch size. General training parameters such as the number of epochs, the early stopping patience, and the target GPU are also editable. Moreover, the user can define a folder where the results will be saved. Once all parameters are set, the training process is initiated by pressing the

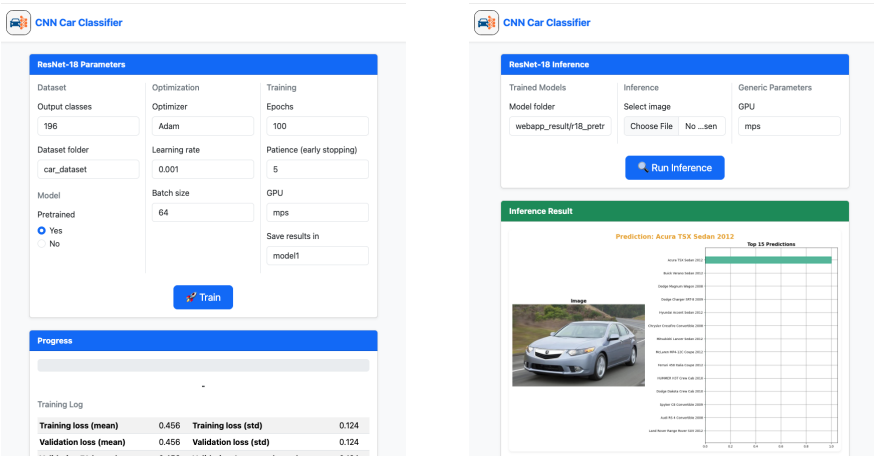


Figure 6: Web interface for a generic image classification task with the ResNet-18.

"Train" button. Below the main configuration panel, there is a progress bar indicating the current training status and a log section that displays real-time feedback from the training process, such as loss values and performance metrics across epochs.

Inference The inference's web interface represents a tool for testing the trained models. After choosing the model to utilize, the user can load any image, and the loaded model classifies the image into one of the predefined classes. During the image loading, the script verifies the correct format size and eventually resizes the input to adhere to the model's input dimension. The image passes through the model, and finally, the script returns the top-15 classes with their probability distribution.

7 Conclusion

This project explored the application of Convolutional Neural Networks (CNNs), in particular ResNet-18 and ResNet-34, for the task of classifying car brands and types. Using the PyTorch framework, various experiments were conducted to identify the optimal hyperparameters.

An interesting aspect of the project involved the impact of pre-training and layer freezing on the models. The results showed that pre-trained models, particularly ResNet-34 with non-frozen layers, achieved the best performance in terms of F1-score. Configurations with random weight initialization or with a high number of frozen layers showed poorer performance. This points out that feature extractors require a large amount of data to learn discriminative patterns (in the downstream task), and excessive freezing limits the model's ability

to specialize in the specific task. No significant performance difference was observed between ResNet-18 and ResNet-34, indicating that the smaller network was sufficient for effectively solving the classification task.

In the project, I investigated adversarial attacks, demonstrating how small perturbations in the input can lead the classifier to misclassify. An example of a Fast Gradient Sign Method (FGSM) attack on ResNet-18 was illustrated. Finally, I developed a web application with an interactive interface to train and utilize ResNet-18 models for image classification tasks.

Use of generative tools During the preparation of this work, I used Grammarly in order to improve my grammar and spelling checking. After using these tools, I reviewed and edited the content as needed.

References

[1] Swarnambiga Ayyachamy et al. "Medical image retrieval using Resnet-18". In: *Medical imaging 2019: imaging informatics for healthcare, research, and applications*. Vol. 10954. SPIE. 2019, pp. 233–241.

[2] Alexander Buslaev et al. "Albumentations: fast and flexible image augmentations". In: *Information* 11.2 (2020), p. 125.

[3] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[4] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples". In: *arXiv preprint arXiv:1412.6572* (2014).

[5] Kaiming He et al. "Deep residual learning for image recognition. 10.48550". In: *arXiv preprint arXiv.1512.03385* (2015).

[6] Enbiao Jing et al. "ECG heartbeat classification based on an improved ResNet-18 model". In: *Computational and Mathematical Methods in Medicine* 2021.1 (2021), p. 6649970.

[7] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: `1412.6980 [cs.LG]`. URL: `https://arxiv.org/abs/1412.6980`.

[8] Jonathan Krause et al. "3d object representations for fine-grained categorization". In: *Proceedings of the IEEE international conference on computer vision workshops*. 2013, pp. 554–561.

[9] Anders Krogh and John Hertz. "A simple weight decay can improve generalization". In: *Advances in neural information processing systems* 4 (1991).

[10] Yi Zhao et al. "Deep learning classification by ResNet-18 based on the real spectral dataset from multispectral remote sensing images". In: *Remote sensing* 14.19 (2022), p. 4883.