Targets detection in radar signals

Raluca-Andreea GÎNGA

Faculty of Mathematics and Computer Science, Bucharest

December 29, 2021

Abstract

In this documentation, there are presented several solutions for detecting and classifying targets under radar interference based on 15.5 thousands of images. The approaches presented in the next few pages include methods like feature extraction using deep learning pretrained models followed by application of classical machine learning algorithms (which gave the worst performance), pretrained models (Dense Net and Xception) applied on slightly augmented data that provide some better results and another pretrained model, Efficient Net, used on a special form of data augmentation, called mixup, that brought the best performance overall.

1 Introduction

Autonomous vehicles are cars having the ability to perceive their surroundings using a variety of sensors, but also to analyze and process the received data, to understand them and to function properly in the end. At this moment, technology doesn't allow automobiles to drive themselves securely without a human driver. Modern cars, on the other hand, contain a number of features that can be included into the category of partial autonomy.

What does partial autonomy mean in sense of levels?

1. **Lack of automation**

   The driver is fully responsible for doing things like changing direction or speed.

2. **Driver-assistance**

   The car has basic automatic systems such as cruise control.

3. **Partial automation**

   The human being should always monitor the activity of the car. The car can use the steering system or controling the speed, but only in presence of the human driver.

4. **Conditional automation**

   It is containing facilities for the environment perception. The car is able to navigate alone most of the time. The driver has the responsibility to always be ready to regain the control of the car.

5. **High automation**

   The majority of the tasks could be done by the car itself. The driver may request to interveve occasionally.

6. **Autonomy**

   The car could do all of the navigation tasks without needing the human factor.

Today, state-of-the-art technology is approaching level 4, and some level 3 functions are being developed for production vehicles. The journey from scratch to a fully autonomous car requires continuous improvement in hardware and software.

Of all the sensors used in autonomous driving, the most varied range of data is obtained with the help of video cameras. They can have different shapes, so it can be obtained a full 360 degree view around the car.

Using radar sensors is a significant option for purchasing information that cannot be obtained by video cameras, it is versatile and has the advantage of operating regardless of the weather. The radar system is effective in detecting objects remotely so that the car has a detailed picture of the immediate environment. Radars mounted on different vehicles can interfere with one another, reducing thus the capacity of detection. In this regard, the proposed task was to develop some deep learning approaches in order to recognize the targets (people, other cars, barriers and so on) under radar interference.

The remainder of this documentation is organized as follows: section 2 describes the dataset and architectures used for this task. Section 3 describes the details of the deep learning approaches used, followed by their results on section 4. The last section (5) concludes the documentation about the implemented methods and comes with some further work that could be done in order to research more about this task.

2 Dataset and Hardware Environment

2.1 Overview of the dataset

The dataset of this task is consisting in:

- two .csv files that contain the images file names and one of them (train.csv) contains the label associated to them as well, whereas test.csv contains only the name of the image
- two folders (corresponding to train and test datasets) with the images

There are 15.500 training images and 5500 test images. All of these images have 55x128 pixels dimension.

In order to work on such a project and to ease the process of applying various Deep Learning techniques, the training set (based on the labels present in train.csv file) was split into 5 different folders corresponding to each class.

2.2 Hardware environment

In order to experiment, but also to train various models and try different approaches, I used 3 different work environments and I paralleled the work in several directions using those 3 hardware environments.

- NVIDIA GeForce GTX 1050 Ti with 4GB memory
- NVIDIA GeForce GTX 1650 with 4GB memory
- NVIDIA Tesla K80 with 12GB GPU memory provided by Google Colab

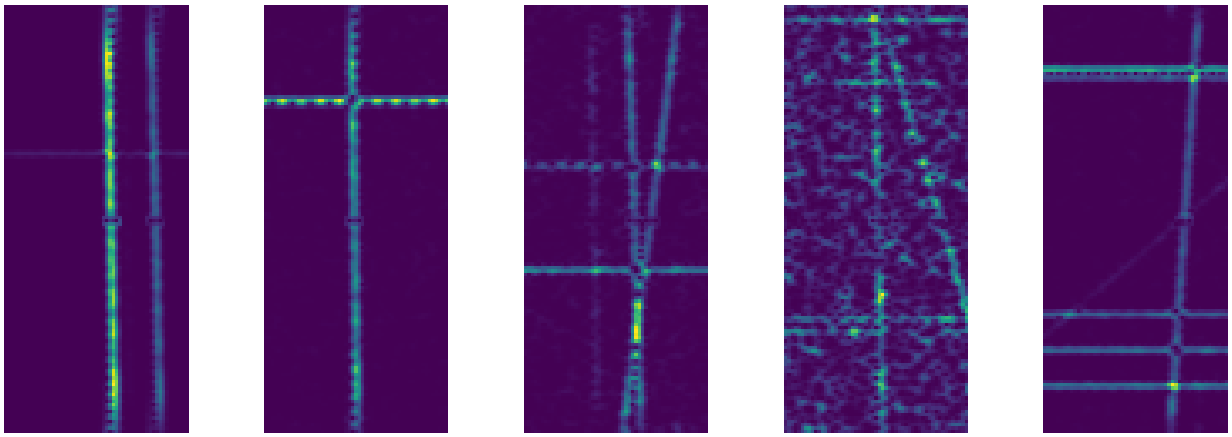


Figure 1: Sample of 5 images belonging to 5 different classes (1 belongs to the first image, 2 to the second and so on)

3 Approaches

3.1 Feature Extraction using Pretrained Models

In computer vision, transfer learning represents a popular approach since it allows us to develop in a quick manner correct and accurate models.

In this approach, 3 feature extraction models were used, VGG16[5], Xceptio[1] and a concatenation of the features extracted from the two previously mentioned models. Then, to reduce the high computational cost and high training time of neural networks, there were used 3 classic Machine Learning algorithms: XGBoost, Logistic Regression and Linear SVC. No data augmentation was used here, so low result is quite expected in this case. This method, as it will be seen in the next chapter (4), provides the weakest performance on validation data.

3.2 Pretrained Models

This approach consisted of using pre-trained models, adding new layers and freezing certain layers. The pre-trained models selected were DenseNet 169 and Xception. Other models such as ResNet 50 have also been tried, but a high overfitting trend has been observed and these models have been abandoned in order not to consume existing resources and to focus more on the models that seemed to bring better results. For these two pretrained models, there was applied a data augmentation form of rescaling the images and random rotating them with random angles between -30 degrees and 30 degrees.

3.2.1 DenseNet 169

A first architecture (pretrained model) used was Dense Convolutional Network with 169 layers brought in 2017 by Huang et. al [2].

The architecture used for this task contained:

- DenseNet169 model, freezing the first 55 layers
- Global Average Pooling 2D that takes the average of all values across a tensor of $input\_width \times input\_height \times input\_channels$
- Dense layer of 5 corresponding to the 5 classes the images belong with softmax activation

Layers	Output Size	DenseNet 169
Convolution	112×112	7×7 conv, stride 2
Pooling	56×56	3×3 max pool, stride 2
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv
	28×28	2×2 average pool, stride 2
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv
	14×14	2×2 average pool, stride 2
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$
Transition Layer (3)	14×14	1×1 conv
	7×7	2×2 average pool, stride 2
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$
Classification Layer	1×1	7×7 global average pool
	5	5 D fully-connected, softmax

Figure 2: DenseNet 169 Architecture

Hyperparameters Using Bayesian Optimization:

- Adam optimizer with a learning rate of 0.0001 provided the best results

3.2.2 Google Xception

Xception model was researched and brought by one of the Google researchers, involving Depth-wise Separable Convolutions. This kind of model is separated into 3 flows: *entry flow, middle flow with 8 repetitions of the same block* and *exit flow.*

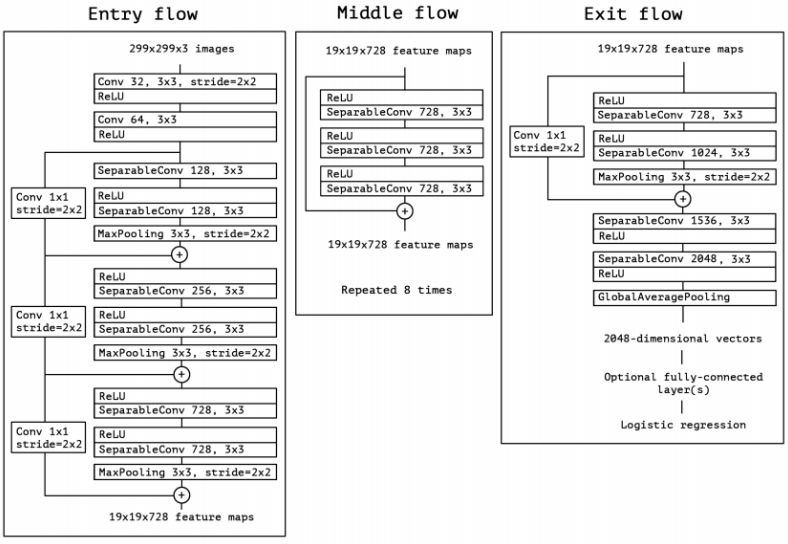


Figure 3: Xception Architecture

The architecture used for this task contained:

- Xception full model without freezing any layer
- Global Average Pooling 2D
- Dense layer of 4096 neurons with relu activation

- Dense layer of 5 neurons corresponding to the 5 classes the images belong with softmax activation

Hyperparameters Using Bayesian Optimization:

- Adding a new Dense layer of 4096 gave slightly better results than no adding at all or other values of that specific dense layer
- Adam optimizer with a learning rate of 0.001 provided the best results

3.3 Mixup

This approach is a more innovative one and less explored in image classification field that brought the best results for targets detection of radar signals.

Usually, we input a data point X into a neural network with parameters, obtaining a predicted class/label along with the true label. We also have a loss function that compares what is the output with the true label and then trying to make that loss smaller. Briefly, we want to adjust our parameters such as the next time, the output of X, to be a little closer to the reality. We are calling this technique *Empirical Risk Minimization (ERM),* because our data point X comes from some data distribution $\mathscr{D}$ like the space of all natural images, but what we actually have is a dataset of a finite amount of data that we can sample X and y from, so instead of minimizing our true risk, we minimize the empirical risk. What is the problem of ERM? The problem is that we can get overly confident about our data points and nothing else and thus, it will hurt the generalization.

The paper of Zhang et al.[6] proposes to train these classifiers on all the data points that are between some two random samples. In mixup technique, it is starting by taking two samples from the dataset and mixing them together, resulting in a data point that's in between the other two samples. This mixing is done in a linear fashion and what is really important is that it is applied both on the feature vectors and the target labels. Coefficient λ is sampled randomly, however, it's always going to be in $[0, 1]$ interval.

$$\tilde{x} = \lambda x_i + (1 - \lambda)\ x_j, \qquad x_i, x_j \text{ - raw input vectors}$$
$$\tilde{y} = \lambda y_i + (1 - \lambda)\ y_j, \qquad y_i, y_j \text{ - one-hot encoded classes}$$
$$(x_i, x_j)\,, (y_i, y_j\ ) \text{ two samples randomly drawn from the dataset}$$
$$\lambda \in [0, 1]$$

The model will learn thus to basically smoothly interpolate. The authors claim that input mixup can help improve generalization of the models applied and that it can yield smoother estimates of uncertainty.

Example 3.1 On the toy dataset that we see below, it is shown what mixup can do. So, in a classic model, we have the orange and the green data points and blue is basically where the classifier believes it's class one (orange one) being a hard border there. The hard border is sort of a problem in itself because if we have the data dispersed and there are green points on the hard border, the ERM model is absolutely sure those points belong to the orange class. When using mixup, the border is much more fuzzy and in the middle (between the delimitation of those two classes) the classifier isn't that sure about the class a data point belongs to, so that's kind of a more desirable situation.

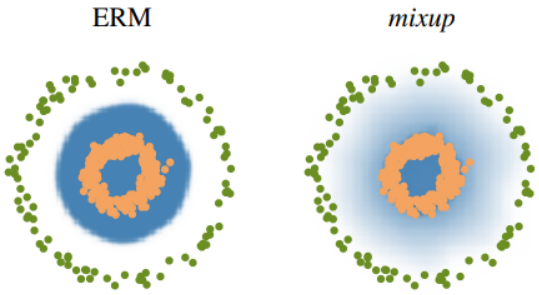


Figure 4: Effect of *mixup* ($\lambda = 1$) on toy problem. Green is represented by class 0, whereas orange in represented by class 1. Blue shading indicated the probability of a class 1 given the data points x

3.3.1 Workflow: Libraries, Augmentation and Models

For this approach, I used some several new libraries that provided the best results:

- PyTorch Lightning [1]- which is a Pytorch lightweight wrapper for Deep Learning researchers that aims to help creating models that are faster, easy to work with and more reliable
- Albumentations [2] - similarly to Image Data Generator provided by Tensorflow, this library is also designed for image augmentation. The transformations made were: resizing the images to 224x224 pixels, horizontal flipping, vertical flipping, random rotating, normalizing and converting to tensor
- Timm (Py**T**orch **Im**age **M**odels) [3] - a library that is stocking a wide variety of State of the Art models.
- Test Time Augmentation[4](Ttach library [4]) which aims to do some random modifications on the validation/test images and create some data augmentation copies on the flow of each image, returning in the end an esemble of the predictions made on the augmented images.

The architectures used for this approach were:

- Efficient Net V2 & Mobile Net V2 models were used and created using Timm library, adding some fully connected layers with 'relu' activation
- Accuracy as performance metric
- Combination of Cross Entropy loss with the mixup technique for training dataset & Cross Entropy loss for the validation dataset
- Adam with weight decay (AdamW) has been chosen as an optimizer
- Cosine annealing warm restarts has been used as a learning rate scheduler
- For a 2% increase in validation accuracy, it was also used test-time augmentation with horizontal, vertical flippings and rotations of 0, 90, 180 or 270 degrees

All of these approaches were used on Google Colab, since it's more powerful to train and because it was seen the potential of this approach and tried on so many Kaggle competitions.

[1] https://www.pytorchlightning.ai/
[2] https://albumentations.ai/
[3] https://github.com/rwightman/pytorch-image-models
[4] https://github.com/qubvel/ttach

4 Results

4.1 Feature Extraction using Pretrained Models

As already said in chapter 3 (Feature Extraction using Pretrained Models subsection), VGG16, Xception and combination of both models were used in order to extract the features from images, followed by application of 3 classical Machine Learning models: XGBoost, Logistic Regression and Linear SVC. Model predicting of features for all images lasted like 2 seconds/image. Having a total of 21000 images (training + validation), it lasted like 700 minutes (~12 hours per model). The results obtained are listed in the following table:

	VGG16			Xception			VGG16 + Xception		
Model	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
XGBoost	0.40	0.39	0.40	0.40	0.39	0.40	0.40	0.39	0.40
Logistic Regression	0.41	0.40	0.40	0.41	0.40	0.40	0.41	0.40	0.40
Linear SVC	0.38	0.37	0.37	0.35	0.38	0.35	0.35	0.39	0.35

Table 1: Results of XGBoost, Logistic Regression and Linear SVC on pretrained models

4.2 Pretrained Models

In the following table and figures, there are presented the results obtained for DenseNet169 and Xception models.

Model	Total params	Time per epoch (on average)	Total time (hour)	Hardware Environment
DenseNet169	12.651.205	55 minutes	13.8 hours	NVIDIA GeForce GTX 1050 Ti
Xception	29.274.669	25 minutes	6.3 hours	NVIDIA GeForce GTX 1650

Table 2: DenseNet169 and Xception training time

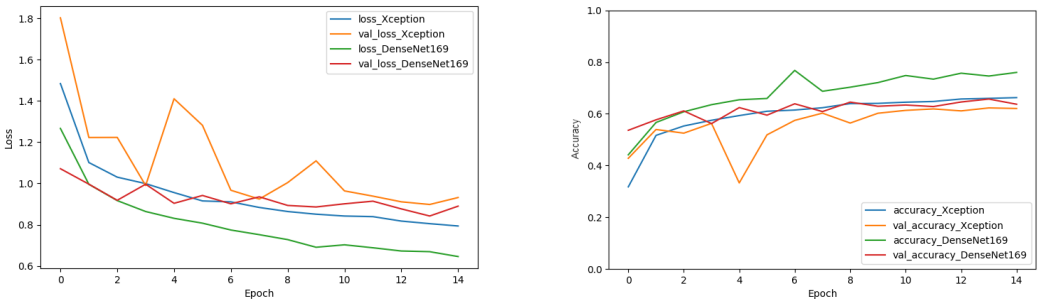


Figure 5: 5a) Loss on training and validation on Xception and DenseNet169 models.
5b) Accuracy on training and validation on Xception and DenseNet169 models.

4.3 Mixup

In the table from below, there are presented the results with and without applying test-time augmentation on the validation dataset. along with the training time spent for each model.

The results for Mixup technique applying only albumentation are listed in the figures below that showcase the loss and accuracy on validation dataset using MobileNet and EfficientNetV2.

Table 3: Models and training time

Model	Total params (MB)	Time per epoch (on average)	Total time (hours)	Hardware Environment
MobileNetV2	4.0	2.5 minutes	2.1 hours	NVIDIA Tesla K80
EfficientNetV2	24.4	7.1 minutes	5.9 hours	NVIDIA Tesla K80

	MobileNetV2			EfficientNetV2		
Data augmentation method	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Only albumentation	0.69	0.70	0.69	0.74	0.74	0.73
Albumentation + Test time augmentation	0.71	0.73	0.70	0.76	0.77	0.76

Table 4: Models and their results with and without test-time augmentation

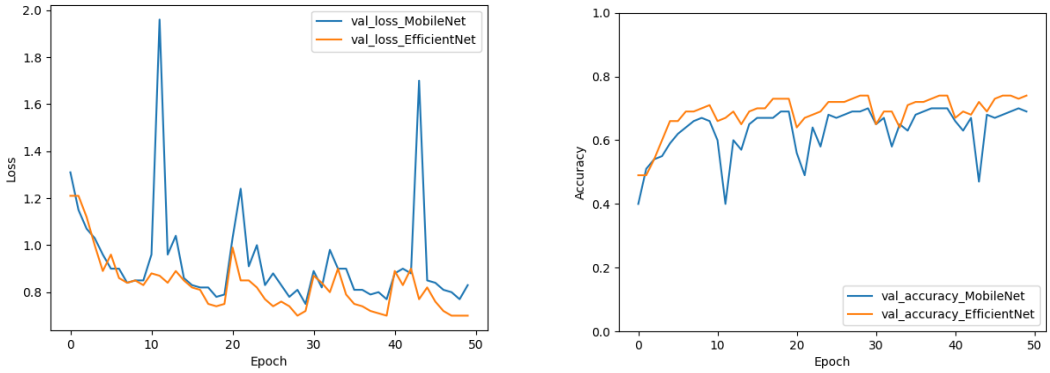


Figure 6: Loss and accuracy validation history on MobileNet and EfficientNetV2 models.

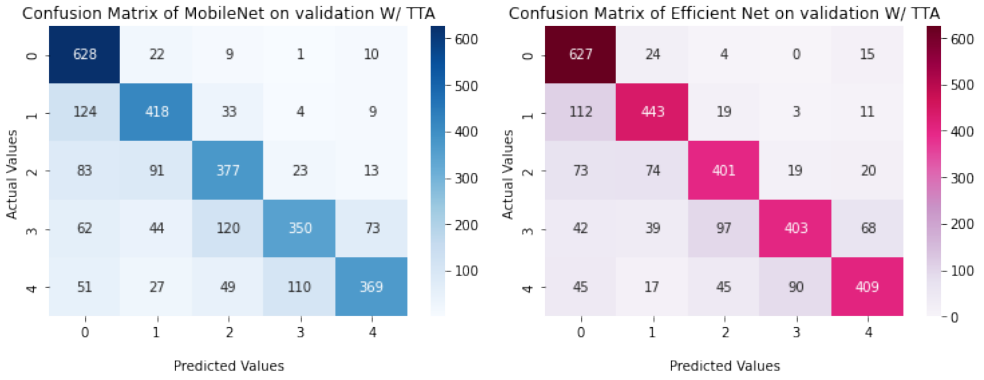
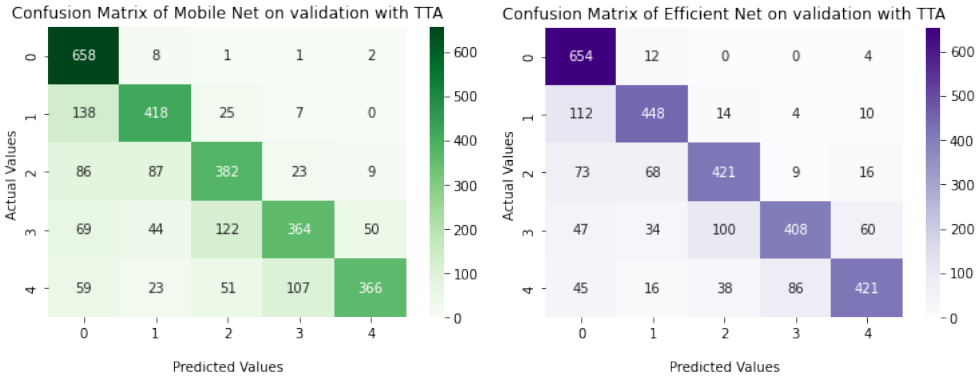


Figure 7: Confusion Matrix for Mobile Net and Efficient Net without&with Test-time augmentation.



5 Conclusion and Further Work

In the current project, it was solved the problem posed by "Detect targets of radar signals" competition on Kaggle[5]. There were applied various methods, including the application of pretrained models as feature extraction followed by machine learning algorithms, application of pretrained models (DenseNet169 and Xception) with and without freezing the layers and finally, mixup technique with special forms of augmentation given by albumentations and test-time augmentation at the final stage. The enormous potential was given by the former mentioned method, Mixup and Efficient Net V2 model, experimenting with different parameters of λ in mixup sample and obtaining the same results. The best result on accuracy was provided by Efficient Net V2 model, on the test dataset on Kaggle obtaining a score of 0.76363 on public leaderbord on 25% of the test data and 0.77260 on the private leaderbord, so a better improvement on the other 75% of the data tested that ensures the generality of the model.

As a potential future work, Mixup method can be fully exploited and it can be applied other models as well, such as ResNet, FixEfficientNet, Xception; The current model that provided the best results, EfficientNetV2, could be also trained for more than 50 epochs with AdamW, SGD or other optimizers as well. Due to the limited resources that Colab provides (more precisely the time limited to 12 hours), the models could be trained on the Cloud in AWS or using more powerful GPUs that have no time limit.

[5] https://www.kaggle.com/c/detect-targets-in-radar-signals

Bibliography

[1] Francois Chollet, *Xception: Deep Learning with Depthwise Separable Convolutions*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017

[2] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger, *Densely Connected Convolutional Networks*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017

[3] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, NIPS'12: Proceedings of the 25th International Conference on Neural Information Processing Systems, 2012

[4] Divya Shanmugam, Davis Blalock, Guha Balakrishnan, John Guttag, *Better Aggregation in Test-Time Augmentation*, arXiv:2011.11156, 2020

[5] Karen Simonyan, Andrew Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, International Conference on Learning Representations, 2015

[6] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, David Lopez-Paz, *mixup: Beyond Empirical Risk Minimization*, ICLR 2018 Conference Blind Submission, 2018

Table of Contents