Detect targets in radar signals

Tudor Andrei Dumitrascu

December 28, 2021

1 Introduction

Radars are useful in for autonomous driving because they can be used to detect object (e.g. cards, buses, pedestrians or bikes) around the car. Therefore it's important to make correct predictions. In this case, Deep Learning can be used to detect such object from the resulting radar outputs. The goal of this project is to present and showcase the results of using multiple DL methods to count the number of objects given scenarios detected by a radar.

2 Data

The data is split into in a training set with give labels and a testing set which is used to create the submission. The training set images are found in a folder, and along side it, a csv document which contains the label for each file. The data is loaded using the PyTorch Dataset and Dataloader. The Dataset objects are used by the Dataloaders. The processing is defined in the Dataset class, the images are loaded from disk using OpenCV, the channel order is then changed from B,G,R to R,G,B, and finally is passed trough the transformation pipeline. The transformation pipeline is defined using the Torchvision Library Paszke et al. (2019) . The first step in this pipeline are `ToTensor()` which changes to order of the image channels from HxWxC to CxHxW and normalizes the values in the [0,1] interval, the second step is the `Padding` operation which transforms the images from 128x55 to 128x64. The change is size allows for easier computation of dimension when using Convolution operations.

3 Approaches

In the development of the project, multiple approaches were used and all of them are using convolutional neural networks or transformers. An important is to select the output type. Since the task is to count the number of objects in the image, it can be seen as an regression task where the

output values must is a discrete value in the [1,5] interval. At the same time, this can be seen as a classification task. In my implementation, I have used both approaches in order to obtain the best result.

3.1 CNN

The first one is a simple convolutional neural network which uses different Conv2d layers, along with a ReLU activation function. For regularization, batch normalization was used, which prevented the model from overfitting.

3.2 ADNet

The ADNet is a subset of the MVRSS architecture from Ouaknine, Newson, Pérez, et al. (2021). The original architecture handles semantic segmentation from Range-Azimuth-Doppler radar images. In this project, I have taken the branch that handles only the Azimuth part, and used it as the backbone of the model.

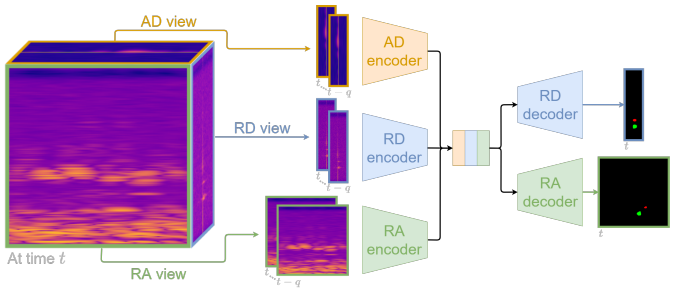

Figure 1: Multi-View Radar Semantic Segmentation

3.3 ViT

The vision transformer was introduced in Dosovitskiy et al. (2021). The goal of this approach was to bring the ubiquitous transformer from the NLP domain into Computer Vision application. The proposed architecture can be with and without convolutional layers. In this implementation I used the convolution-less model. The intuition behind this approach is that the original image is split into patches, which are passed trough the encoder block. I the encoder part, the embeddings are passed trough the global self attention layers and the local MLP layers. The architecture performs really well on large datasets, whereas on smaller ones it doesn't outperform ResNet. Moreover, the author (Tolstikhin et al. 2021, 2) points out performance greatly increases on large datasets (>14M images). One improvement in this direction could be made by pre-training the transformer on the Ouaknine, Newson, Rebut, et al. (2021) dataset, and the fine-tuned to the downstream task.

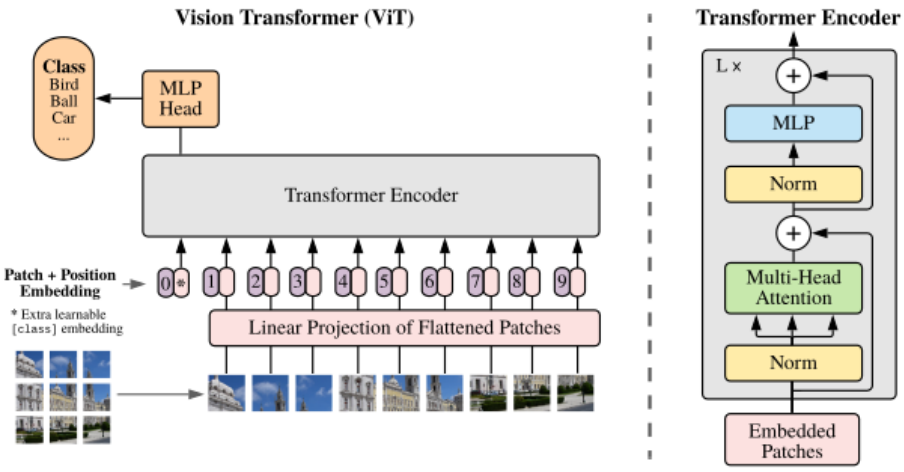

Figure 2: Vision Transformer

3.4 ConvMixer and ConvRepeater

The ConvMixer architecture was introduced in Anonymous (2022), as a combination between Dosovitskiy et al. (2021) and Tolstikhin et al. (2021). The author explores the hypothesis that the performance of the vision transformers relies in the patch-based representation, by directly operating on the patches extracted from the input images. In the original implementation, the residual is added to the output of the first convolution, keeping the number of channels consistent across the ConvMixer Layers. In addition to that, I have used another type of ConvMixer Layer, named ConvRepeater, which concatenates the residual information to the output of the first convolution, therefore doubling the number of channels, hoping to gather more information from the initial image.

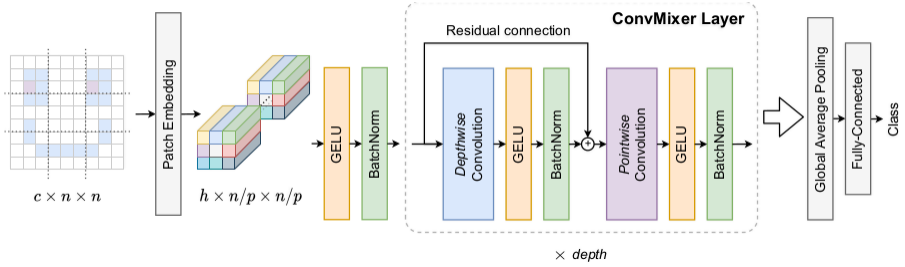

Figure 3: ConvMixer Architecture

4 Cross Validation

Since there is no defined validation dataset, in order to properly evaluate the upper mentioned models, the KFold Cross Validation technique was split. For each fold, a section of the training set is kept back and used for evaluating the model. This procedure was done 5 times across the whole

Model	MAE
CNN_C	1.427
CNN_R	1.937
ConvMixer_C	1.540
ConvMixer_R	1.912
ConvRepeater_C	1.801
ConvRepeater_R	1.934
ADNet_C	1.781
ADNet_R	1.973
ViT_C	1.511
ViT_R	1.718

Table 1: MAE Score across the models

dataset and the average MAE was calculated for each model. By training the same architecture for classification and regression we can choose the best approach for each architecture.

5 Training routine

The train routine was defined using the Falcon and The PyTorch Lightning team (2019) library which removes a lot of the boilerplate code. The models were trained for a maximum of 100 epochs, with a batch size of the maximum availability and the maximum number of workers.

- Model Checkpoint - saves a checkpoint of the model when the lowest validation loss is achieved;
- Early Stopping - stops the training after the validation loss hasn't decreased.

5.1 Hyper Parameters

In the training loop, the Adam optimizer was used along with a scheduler was used that reduces the learning rate once the validation loss reached a plateau. The learning rate in the beginning is `0.003`. The scheduler reduces the learning rate by a factor of `.05` after a certain amount of epoch where the validation loss stagnates.

Model	Val Accuracy
Random Chance	0.200
CNN_C	0.180
ConvMixer_C	0.498
ConvRepeater_C	**0.516**
ADNet_C	0.456
ViT_C	0.286

Table 2: Validation Accuracy

6 Conclusion

The best models are presented in the Table 2, with the best model being highlighted. The outcome of the models is not the most desirable and needs much improvement, during training it was observed that some of the classes are not being correctly learned.

Bibliography

Anonymous. 2022. "Patches Are All You Need?" In *Submitted to the Tenth International Conference on Learning Representations*. https://openreview.net/forum?id=TVHS5Y4dNvM.

Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, et al. 2021. "An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale." *ICLR*.

Falcon, William, and The PyTorch Lightning team. 2019. *PyTorch Lightning* (version 1.4). https://doi.org/10.5281/zenodo.3828935.

Ouaknine, Arthur, Alasdair Newson, Patrick Pérez, Florence Tupin, and Julien Rebut. 2021. "Multi-View Radar Semantic Segmentation." In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 15671–80.

Ouaknine, Arthur, Alasdair Newson, Julien Rebut, Florence Tupin, and Patrick Pérez. 2021. "CARRADA Dataset: Camera and Automotive Radar with Range- Angle- Doppler Annotations." In *2020 25th International Conference on Pattern Recognition (ICPR)*, 5068–75. https://doi.org/10.1109/ICPR48806.2021.9413181.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, et al. 2019. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In *Advances in Neural Information Processing Systems 32*, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, 8024–35. Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Tolstikhin, Ilya, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, et al. 2021. "MLP-Mixer: An All-MLP Architecture for Vision." https://arxiv.org/abs/2105.01601.