Report on CNN/Scattering classification comparison

Alberto Carli Gabriele Roncolato Leonardo Zecchin

Contents

Chapter 1

Introduction

This document illustrates the project valid for the Visual Intelligence course of the academic year 2022/2023.
The assignment tests the knowledge gained in class by going on to apply signal analysis methods, in particular we classified signals using **Convolutional Neural Networks** and using wavelet theory, specifically **Wavelet Scattering**.
We wrote the code for both types of classifications and then compared the results and checked the veracity of the results.
For the entire project, we followed the guidelines of laboratory classes.

Chapter 2

Dataset

Chapter 3

Objectives

The objective of this project was to explore the use of scattering transforms to improve the performance of convolutional neural networks on a given dataset. Specifically, we aimed to compare the performance of a CNN trained on the original dataset to one trained on the scattering decomposition of the data.

We then visualized the filters learned by the CNN and compared them to the ones in the scattering network to gain insight into the types of features that were extracted. By accomplishing these objectives, we hoped to gain a better understanding of how the CNN learns which features to extract.

Chapter 4

CNN Classification

4.1 What is a CNN

A Convolutional Neural Network (**CNN**) is a type of deep learning algorithm commonly used for image recognition and computer vision tasks. It is designed to automatically learn and extract relevant features from input images through convolutional and pooling layers, followed by fully connected layers that produce output predictions.
The `CNN_128x128` architecture consists of four **convolutional** layers and three fully connected layers. The first layer takes an input image with `input_channel` number of channels, and the output of the last layer is a vector with `num_classes` elements representing the probability of each class.

4.2 Architecture

```
CNN_128x128(
  (conv1): Conv2d(3, 32, kernel_size=(9, 9), stride=(1, 1))
  (conv2): Conv2d(32, 32, kernel_size=(9, 9), stride=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1))
  (conv4): Conv2d(64, 64, kernel_size=(5, 5), stride=(1, 1))
  (drop1): Dropout1d(p=0.1, inplace=False)
  (flat): Flatten(start_dim=1, end_dim=-1)
  (fc1): Linear(in_features=576, out_features=256, bias=True)
  (drop2): Dropout(p=0.1, inplace=False)
  (fc2): Linear(in_features=256, out_features=32, bias=True)
  (fc3): Linear(in_features=32, out_features=2, bias=True)
)
```

4.2.1 Convolutional Layers

The first convolutional layer has 32 output channels, while the second convolutional layer has 64 output channels. The third and fourth convolutional layers also have 64 output channels, but they use a smaller kernel size of 5x5 instead of 9x9 used in the first two layers. All convolutional layers have a stride of 1 and use a **Rectified Linear Unit** (**ReLU**) activation function.

4.2.2 Dropout

The dropout layer is used to prevent overfitting in the model. In this architecture, we use a dropout rate of 0.1.

4.2.3 Flatten Layer

The flatten layer is used to convert the output of the last convolutional layer into a vector (*tensor*), which can be used as input to the fully connected layers. In this architecture, the flatten layer converts the output of the last convolutional layer into a vector of size 576.

4.2.4 Fully Connected Layers

The fully connected layers consist of three layers, with 256, 32 and `num_classes` neurons respectively. All fully connected layers use a ReLU activation function except for the last layer which uses the **softmax function** to output the class probabilities.

4.3 Training the Model

To train the model, we pass the input images through the CNN and compute the loss function (in this case **cross-entropy**) between the predicted class probabilities and the true labels. We use **stochastic gradient descent** to optimize the model weights, with a *learning rate* of 0.001 and a *momentum* of 0.9. The model is trained for a fixed number of epochs, and the best model is selected based on its performance on a validation set.

Chapter 5

Scattering+NN Classification

5.1 Wavelet Scattering

5.2 Neural Network

The `NN_128x128` class defines a Neural Network (NN) for classification tasks. The network expects input with a specified number of channels, `input_channel`, and the number of output classes, `num_classes`. The class inherits from the `nn.Module` class, which is a base class for all neural network modules in PyTorch.

5.2.1 Architecture

The NN architecture defined in the NN_128x128 class consists of only **fully connected layers**. There are no convolutional layers that are typically used in a CNN. Instead, the data is flattened using the `nn.Flatten()` layer and passed to the fully connected layers.

The architecture of the network consists of three fully connected layers. The first layer, `self.fc1`, has 256 output neurons, and the input size is 243. The second layer, `self.fc2`, has 32 output neurons, and the third and final layer, `self.fc3`, has `num_classes` output neurons.

Between the second and third layer, there is a dropout layer, `self.drop2`, with a dropout rate of 0.1. The dropout layer helps to prevent overfitting of the model.

The activation function used in the network is **ReLU**, which is applied after each fully connected layer. The last layer uses the **softmax** function to produce probabilities for each of the `num_classes` output neurons.

Chapter 6

Performances comparison

Chapter 7

Filters

Chapter 8

Conclusions