Chapter 1

Interpretability

In this section, we will evaluate the model's prediction to better understand it. We will find if there are any features in the patches that can confuse it and if the model's output is robust. First, we will show how the model learn to correctly separate the features from the two clases, then we will introduce on technique used to highlight the region of the input image that contribute the most to the model predictions. Then, we will use it on the data from the *Quarry* test set to find out the patches were the model fails and analyze them.

Later, we will work with custom created patches with different features, walls, bumps, etc, to test the robustness of the model by comparing its predictions to the real data gathered from the simulator.

1.0.1 Features separability

In general, convolutional neural network learns to encode images by applying filters of increasing sizes at each layer. Usually, the first layers learn basic features, such us edges, while the final one encodes complex shapes. Those final informations are combined and mapped to the correct classes by one or more fully connected. For example, the following image was generated by plotting the learned features for different categories at different layers by Lee et al. [**deepbelief**]

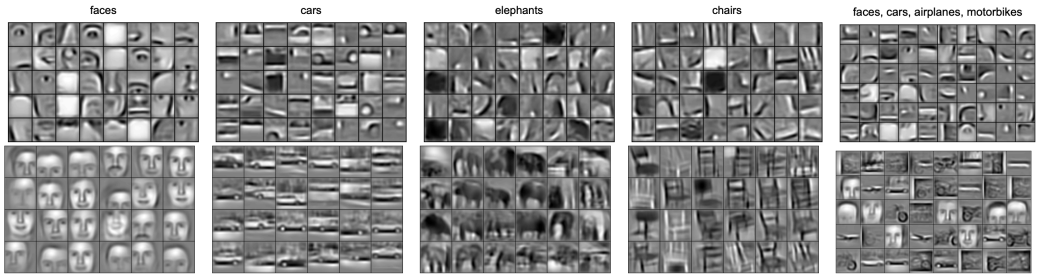

Figure 1.1. Figure from Lee et al. [**deepbelief**] paper where they shown for different classes the low-level features (up) and the high-level features (down) learned by a convolution neural network.

So, a correctly trained network should be able to separate those features based on

predicted classes. Intuitivelly, given two classes $\mathcal{A}$ and $\mathcal{B}$, for example *cat* and *dog*, the high-level features for each class should not be the same, otherwise the model may output a wrong prediction since the same feature is maped to different classes. Thus, visualizing the inputs features vectors against each class can give a good quality estimation of the model since highly separable features yield a correct learning procedure. Our model, MicroResNet, has a last convolution layer's features size of $[128, 3, 3]$. Since visualize a 128 dimension vector is impossible, we reduce its dimension to 2 by applying Principle Anyalisis Component (PCA) [**pca**]. The following figures shows the features of 11K images sampled from the train set label with their classes, *traversable* and *not traversable*.

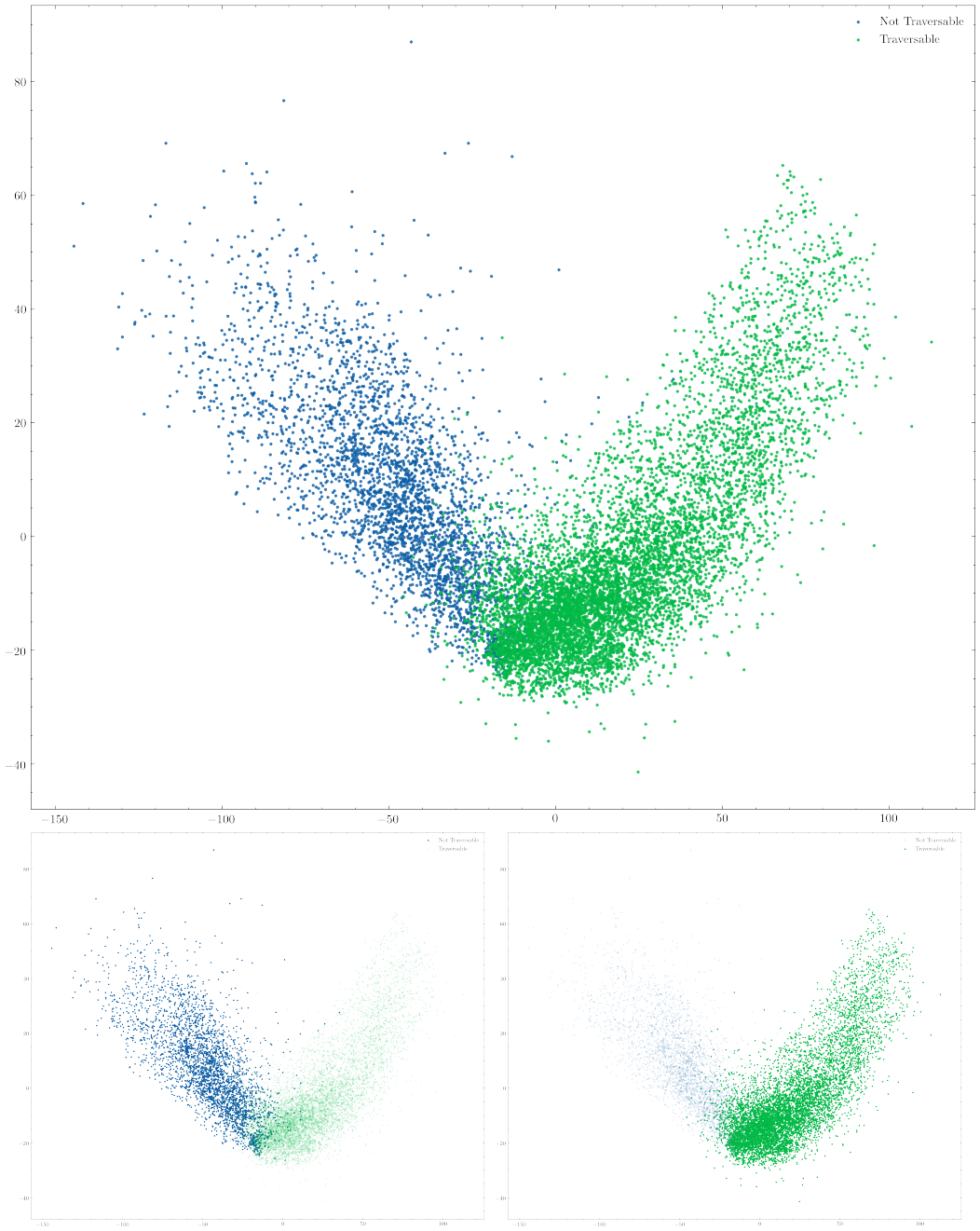

Figure 1.2. Principal Component Analysis

The two point clouds, left and right, are cleary separable meaning that the model was able to learn meaning features from the dataset. On the left there are the features of the not traversable images while on the right the ones for the traversable. One advatange of this procedure is that intrinsically, similar patches will be close to eachother helping to identify clusters of inputs. The following image shows the inputs based on their position

in the features space.

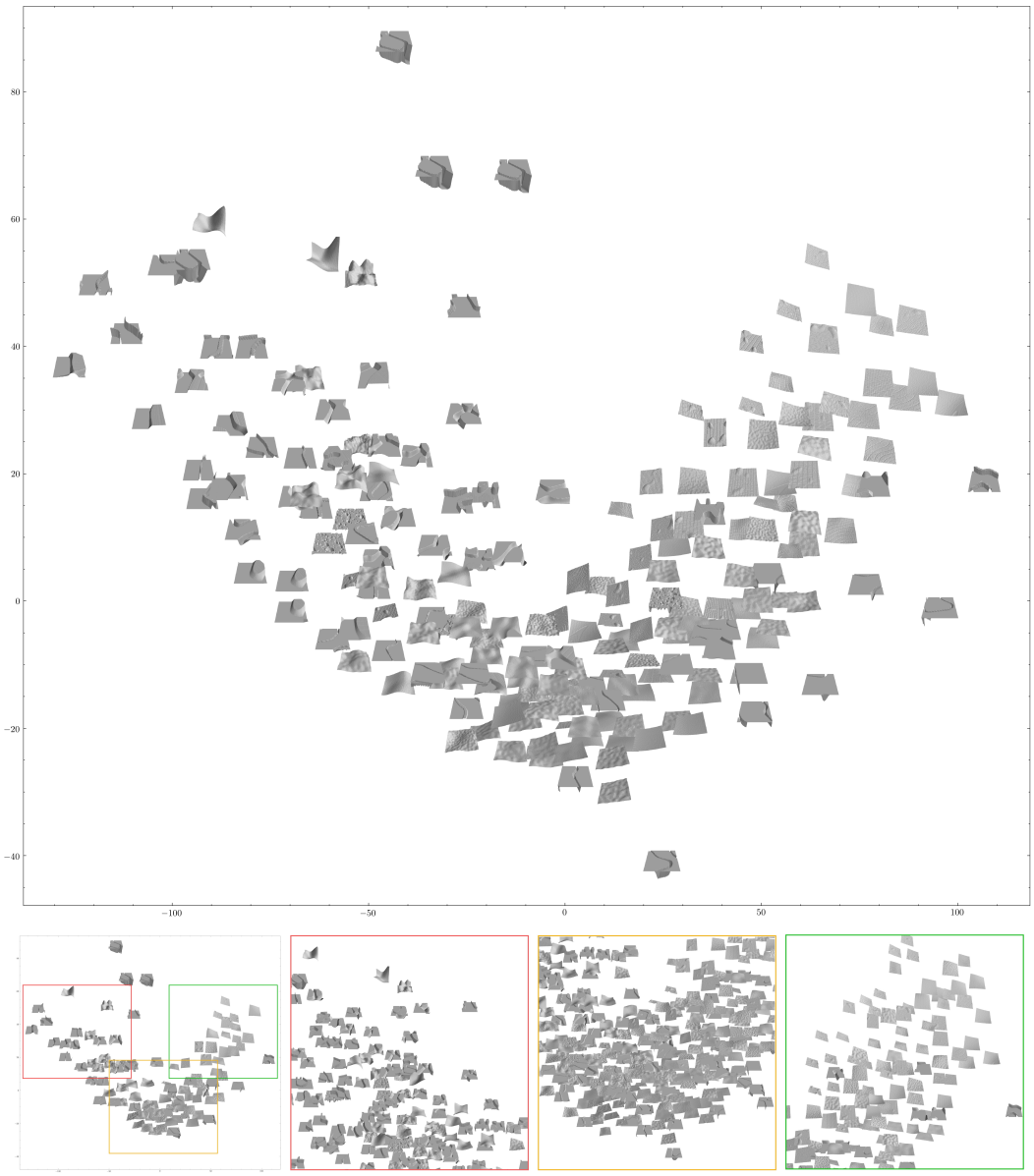

Figure 1.3. TODO

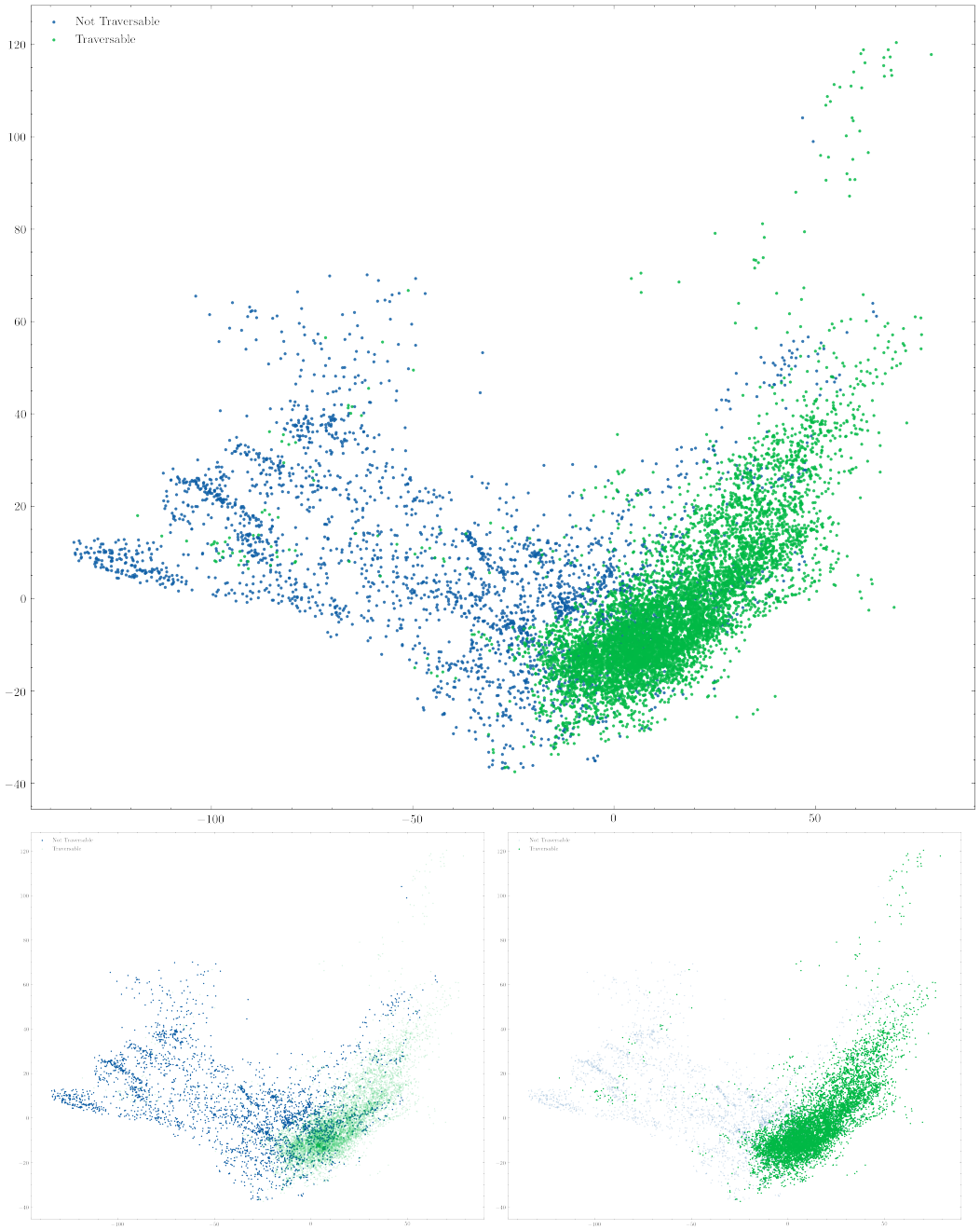

Figure 1.4. TODO

We can clearly see that patches with obstacles, bumps, bars and steps are correctly placed on the left side. Traversable inputs are on the right side, on the top part there are downhills. Goind down we can find little bumps and in the middle we have a big cloud of patches with rocks and small obstacles. Those patches are the most hard do classify since their features are the closest to the not traversable ones.

1.1 Grad-CAM

Gradient-weighted Class Activation Mapping (Grad-CAM) [**gradcam**] is a technique to produce ẗvisual explanationsf̈or convolutional neural networks. It highlights the regions of the input image that contribuite the most to the prediciton.

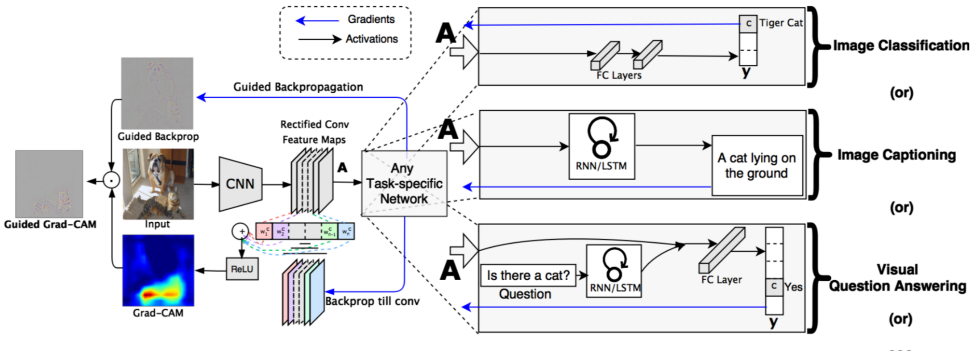

Figure 1.5. Grad-CAM procedure on an input image. Image from the original paper [**gradcam**].

In detail, the output with respect to a target class is backpropagate while storing the gradient and the output in the last convolution. Then, global average is applyed to the saved gradient keeping the channel dimension in order to get a 1-d tensor, this will represent the importance of each channel in the target convolutional layer. We each element of the convolutional layer outputs is multiplyed with the averaged gradients to create the grad cam. This whole procedure is fast and it is architecture independent.

In detail, the output with respect to a target class is backpropagate while storing the gradient and the output in the last convolution. Then global average is applyed the saved gradient keeping the channel dimension in order to get a 1-d tensor, this will represent the importance of each channel in the target convolutional layer. We each element of the convolutional layer outputs is multiplyed with the averaged gradients to create the grad cam. This whole procedure is fast and it is architecture independent.