This chapter is under development!

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**]

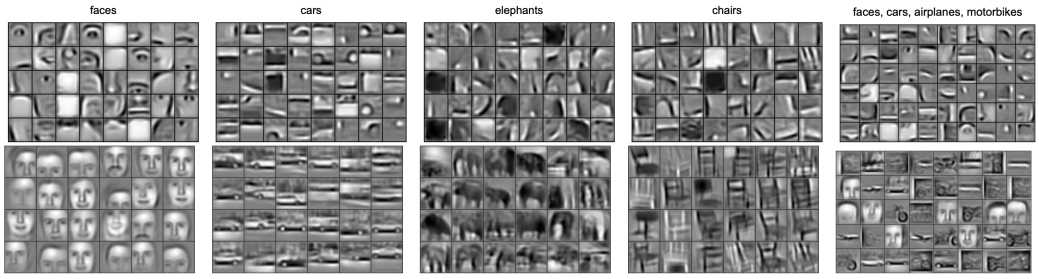


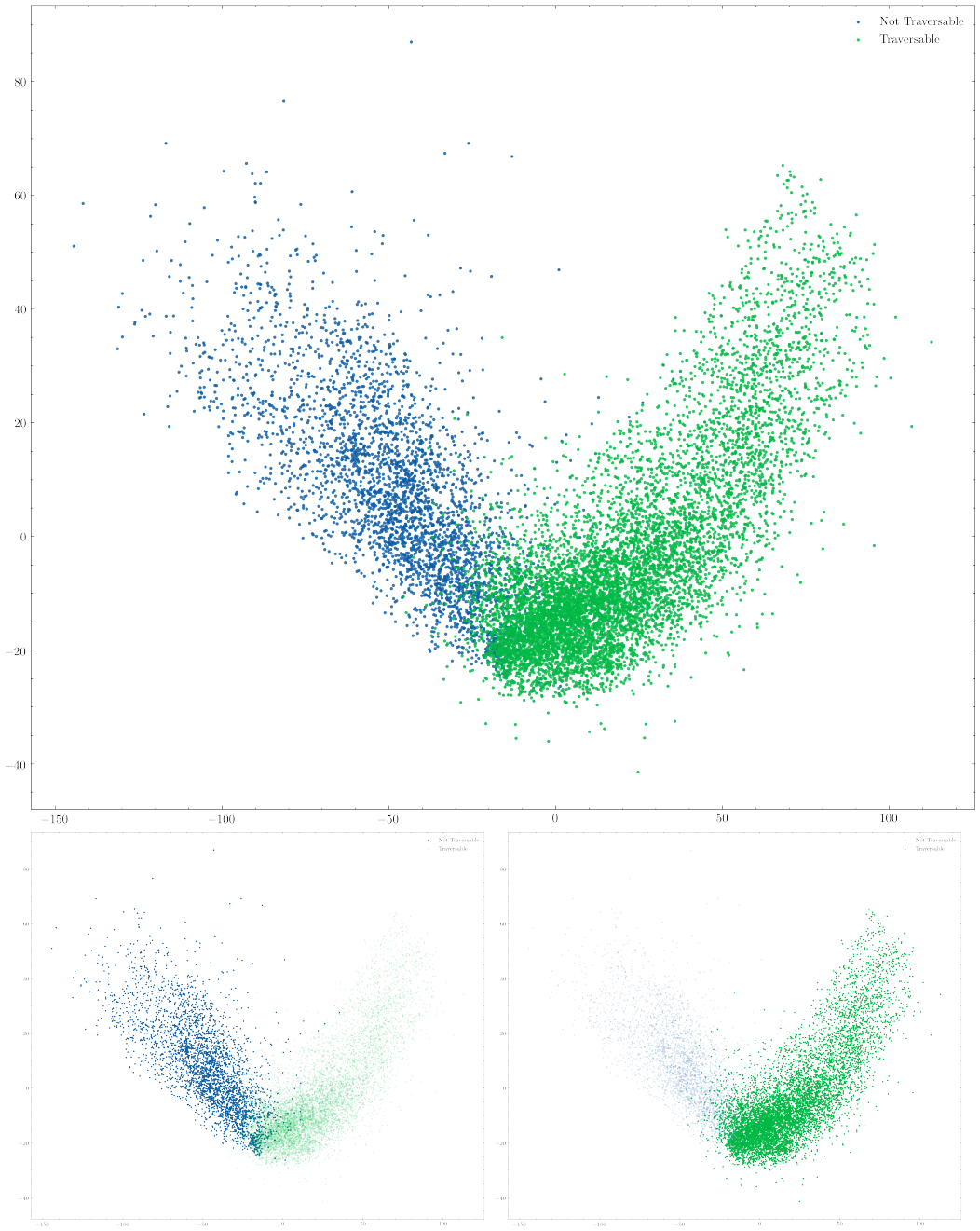
Figure 1.1. Figure from Lee et al. [**deepbelief**] paper where they shown for different classes the low-level features (up) and the high-level features (down) learned by a convolution neural network.

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**].

1.0.2 Train set

The following figures shows the features of 11K images sampled from the train set label with their classes, *traversable* and *not traversable*.

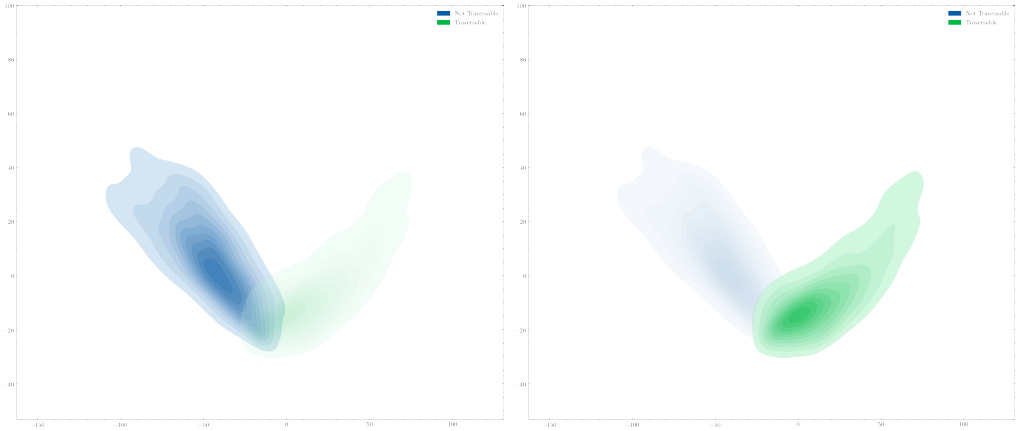


(a) Not Traversable (b) Traversable

Figure 1.2. Principal Component Analysis on the features space computed using the features from the last convolutional layers on the train dataset.

We can clearly distinguish two main clusters based on the labels, one on the left and one of the right. Those points are easily separable, even by human eyes, meaning that

the model was able to learn meaning features from the dataset. Moreover, if we plot the density center, we can see that most of the samples per class are not very close.



(a) Not Traversable (b) Traversable

Figure 1.3. Density plot for the points sampled from the train dataset in the features space. The centers of the cluster are not overlapping yielding a good separability and a correct learning.

We can also identify patches based on their position in the features space, for example, the points on the top left cloud, are patches with walls or big bumps. On the other hand, the hardest patch to classify, or in separate in the features space, are the ones in the middle. Those are the patches with a relatively flat ground and small obstacles. One advatange of this procedure is that intrinsically, similar patches will be close to eachother helping to identify clusters of inputs. We decided to not show all images on the same plot to avoid overcrowing the image. Instead, we clusted the points using K-Means with $k = 200$ clusters and then we took the patch that corresponded to the center point in each cluster. In this way, even if we are showing only a few inputs, we included all the meaningfull features. The following image shows the result.

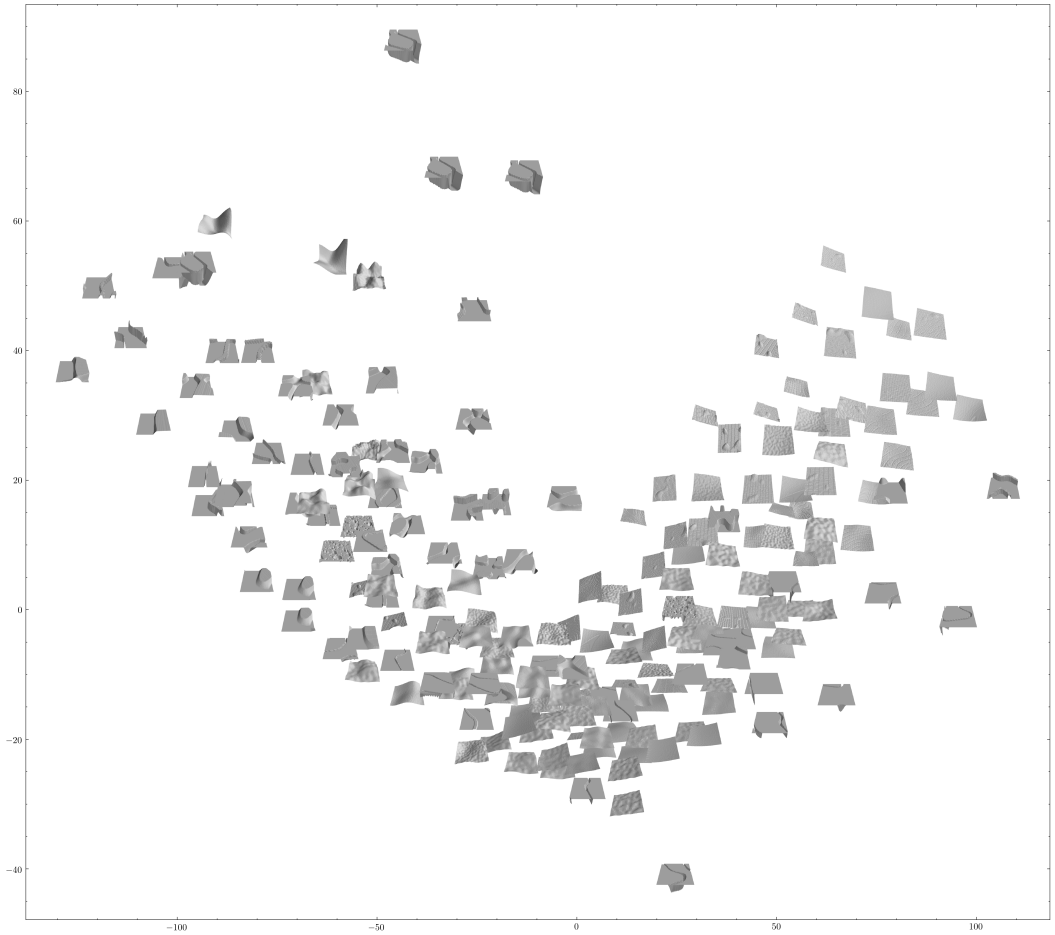


Figure 1.4. Patches corresponding of the points computer on the features space computed using the features from the last convolutional layers on the train dataset. Similar grounds are close to each other.

Patches with similar features are close to eachother. On the left-top side, we can distinguish highgly non traversable patches with walls/bumps in front of the robot. Going down, we start to encoter patches with ligher obstacles. On the platoue, there are traversable patches with small obstacles such as light bumps. It is important to notice that those patches are the closest ones to the non traversable ones, meaning that those are the most harder inputs for the model to classify. Going up on the right side, we see some small steps. Finally, on the top, we find all the downhill patches, the easiest ones to traverse.

1.0.3 Test set

We can apply the same procedure to evaluate the test set. This dataset is harder and present challanging situation for the robot, so we expected some points to be mixedup.

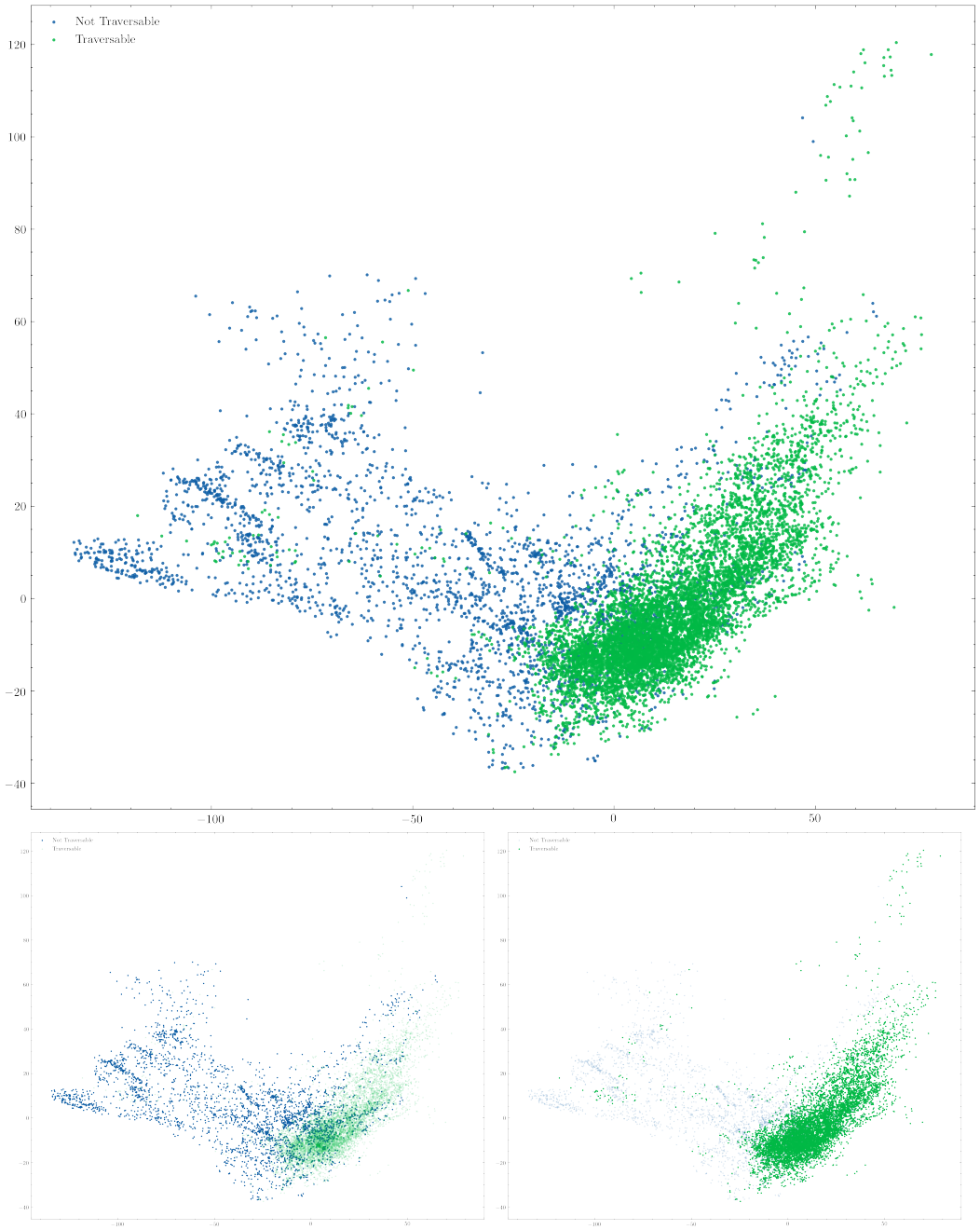
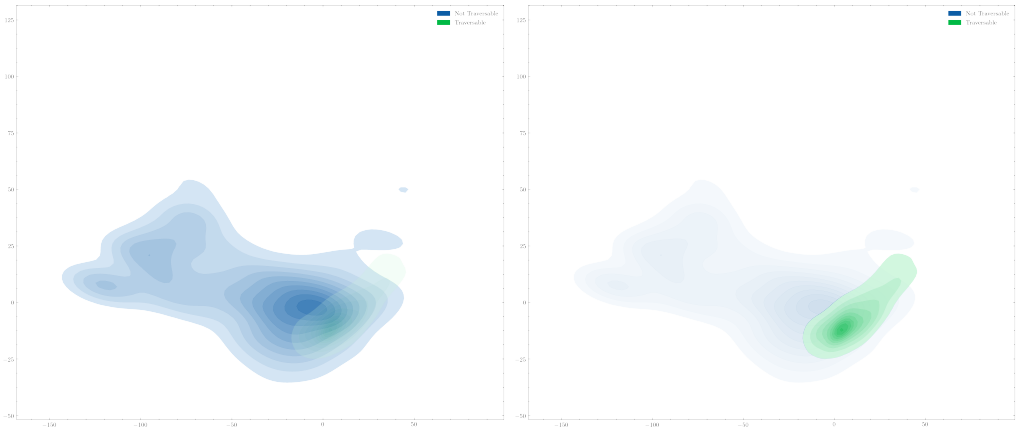


Figure 1.5. TODO

Interesting, the traversable patches are really close to each other, while the others spans a very big surface. This means that there are lots of different terrains with different features that are not traversable. The traversable points are clustered near the center. Moreover, some points are not perfect separable. We plotted the density for each class to better understand where the most points are clusted,



(a) Not Traversable (b) Traversable

Figure 1.6. Density plot for the points sampled from the test dataset in the features space. The centers of the cluster are not overlapping yielding a good separability and a correct learning.

The two centers are really close to each other, this explain the lower AUC score than the validation set. We can also visualize the patches to better understand which inputs are causing the model to fail.

add patches for the test set!

In the following section we will take a deep look at the test set to find which patches confuse the model. Most probably, those samples will be located between the two clusters center where the difference between classes' features is minimum.

1.1 Grad-CAM

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

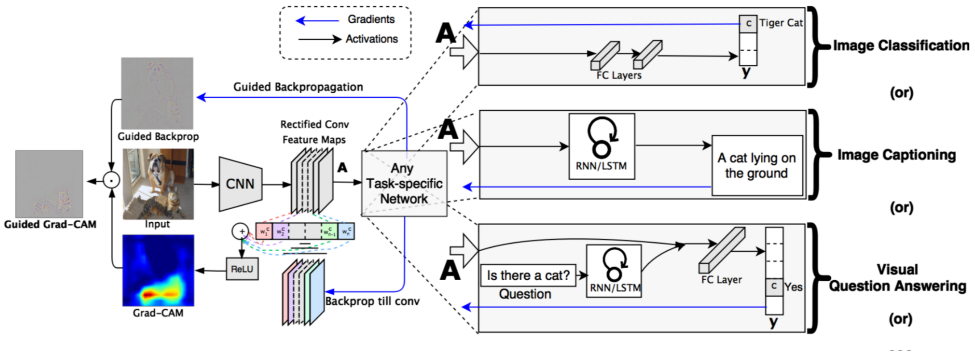


Figure 1.7. Grad-CAM procedure on an input image. Image from the original paper [**gradcam**].

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.