Prof.Student'sAlessandro Giusti Student'sCo-AdvisorR. Omar Chavez-Garcia

This chapter is under development!

# Chapter 1

# Interpretability

Understanding the model's strength, robustness and, limitations is crucial to gain a better understanding of its outputs. This is even more important when working with controllers that can be deployed in real scenarios. In this section, we evaluated the quality of our traversability estimator with different methodology. First, we showed that the model has correctly learned grounds' features and was able to separable terrains based on them. Second, we utilized the challenging Quarry dataset to visualize the most traversable, the lest traversable and the misclassified patches. Utilizing a special method, we determined that the model always looked at the correct features in the ground even if when it fails. Lastly, we crafted patches with different unique features, such as walls, bumps, etc, to test the robustness of the model by comparing its outputs to the real data gathered from the simulator.

### 1.0.1 Features separability

In general, convolutional neural networks learn to encode images by applying filters of increasing size at each layer. Usually, the first layers learn basic features, such as edges, while the final one encodes complex shapes. The final convolutional layer's outputs are usually referred to as features space, consequently, a feature vector is just the output of the last layer for a given image. Those last features are combined and mapped to the correct classes by one or more fully connected. The following image help to visualize the different features learned at each layer. It was generated by plotting the learned features for different categories at different layers by Lee et al. [**deepbelief**].

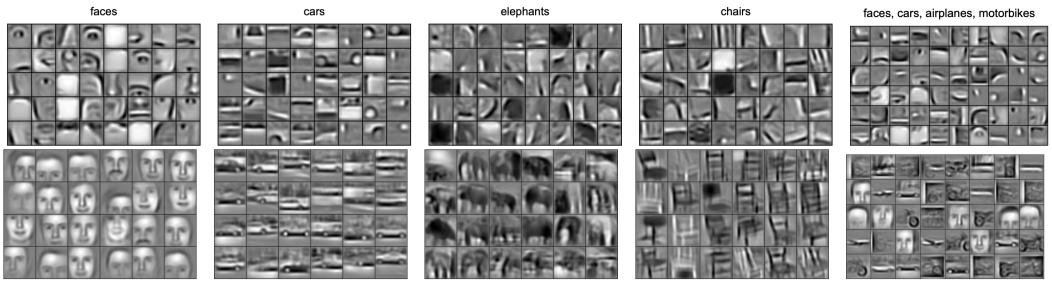
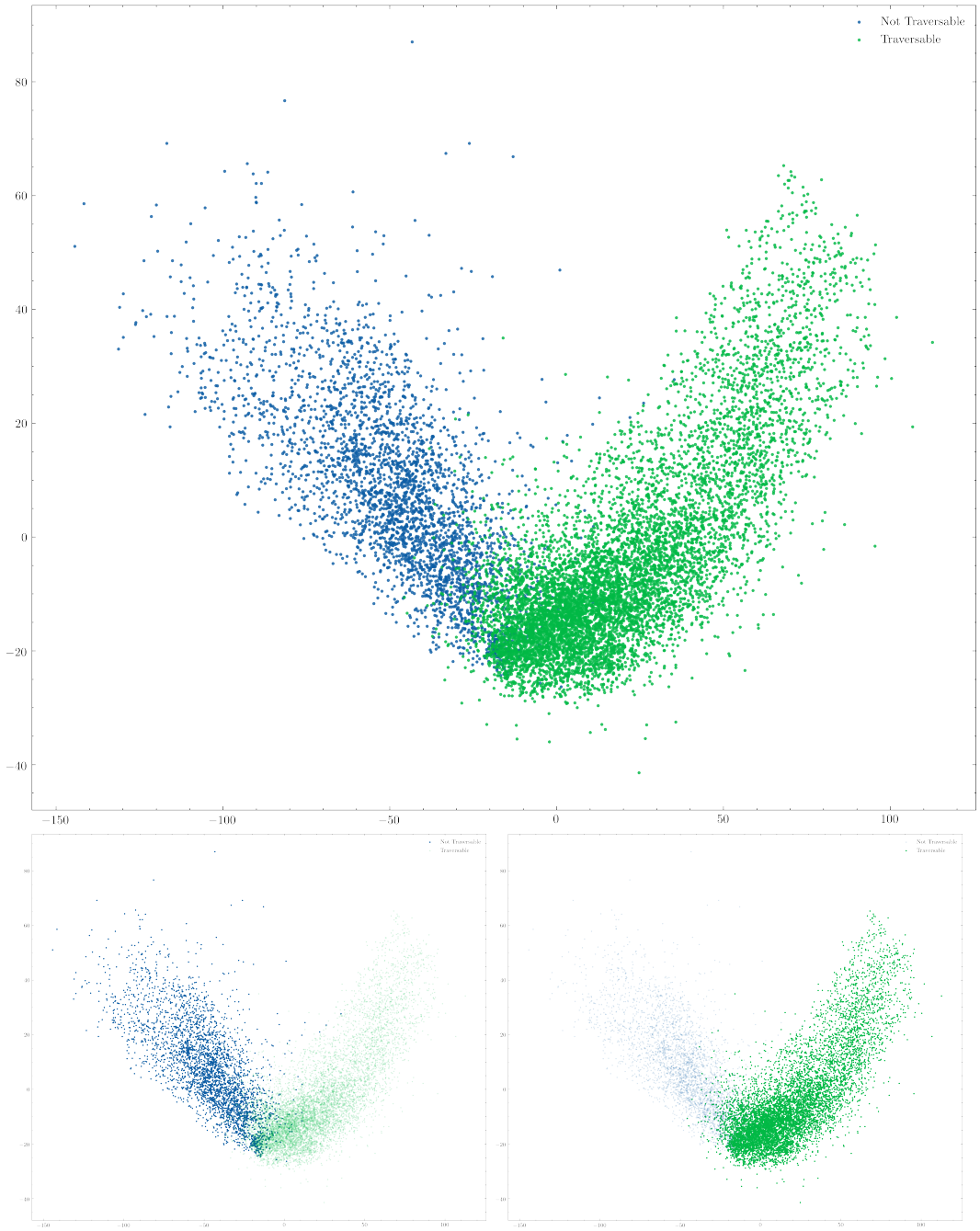



Figure 1.1. Figure from Lee et al. [**deepbelief**] paper where they showed for different classes the low-level features (up) and the high-level features (down) learned by a convolution neural network.

So, a correctly trained network should be able to separate those features based on predicted classes. Intuitively, given two classes $\mathscr{A}$ and $\mathscr{B}$, for example, *cat* and *dog*, the high-level features for each class should not be the same, otherwise, the model may misclassify the input due to the overlap of different classes' features. One technique to discover the degree of separability of our network is to directly visualize the inputs features vectors for each class. Unfortunately, like most models, our network, MicroResNet, has a high dimensional feature space. Each patch is mapped to a $[128, 3, 3]$ vector. We cannot directly visualize a 128 dimension space, we reduced each feature vector dimension to a two-dimension by applying Principle Analysis Component (PCA) [**pca**] to visualize it. We investigate the features space of the model booth in the train and test set.

### 1.0.2 Train set

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

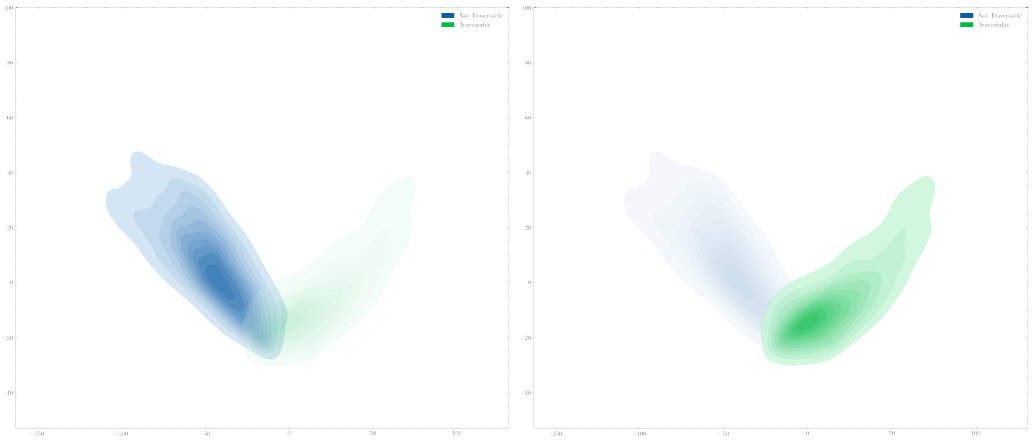



(a) Not Traversable (b) Traversable

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

We can clearly recognize two main clusters based on the labels' color, 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. To be sure the center of each class' point cloud is not overlapping we plotted the density of each cluster.



(a) Not Traversable (b) Traversable

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

Clearly, there is some distance between the centers. Furthermore, we can directly plot the patch corresponding to each feature vector to identify clusters of patches based on their position. Intuitively, if similar inputs are close to each other in the features space then the model also learned to effectively encode terrains features. We decided to not show all images on the same plot to avoid overcrowding the image. Instead, we clustered 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 meaningful features. The following image shows the result.

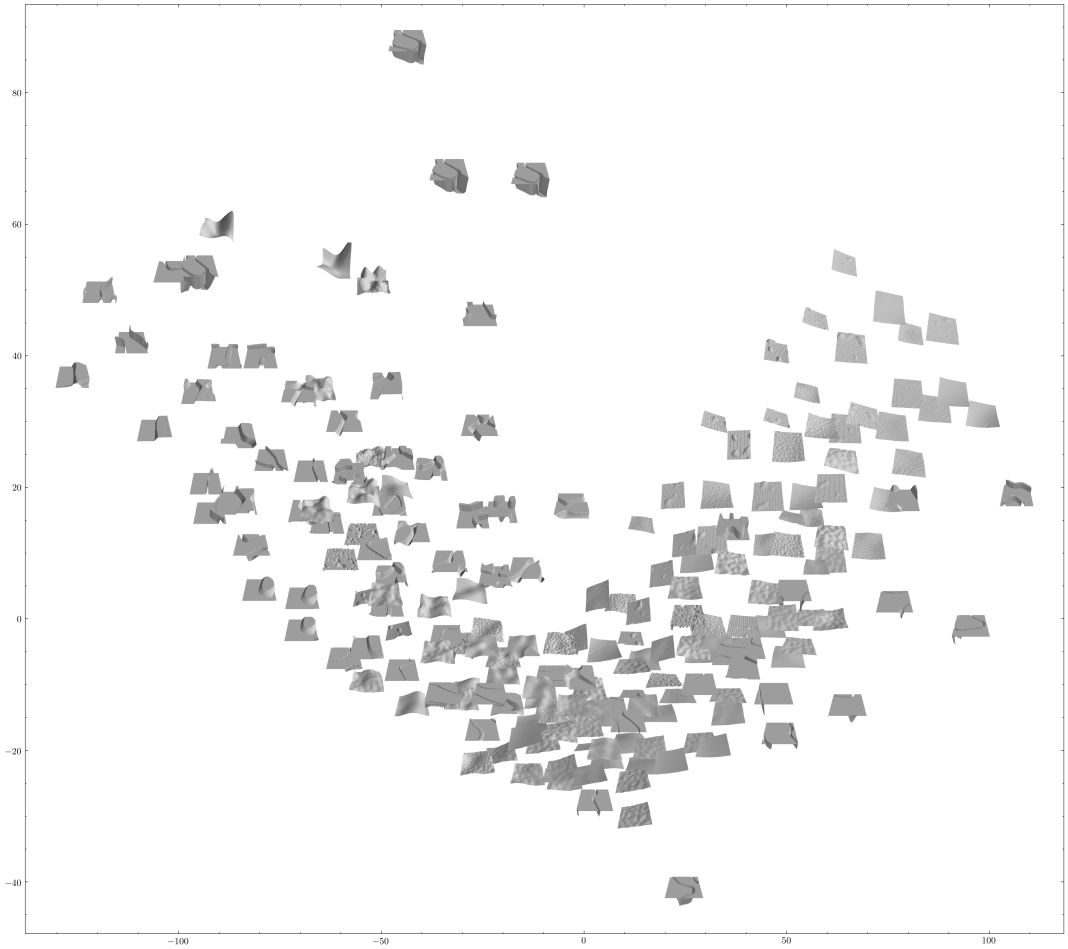



Figure 1.4. Patches plotted using the coordinate of the features vector obtained from the last convolutional layer's output and then reduced using PCA to a two-dimensional vector. Similar grounds are close to each other.

Definitely, patches with similar features are close to each other yielding a quality features encoding. On the left-top side, we can distinguish highly untraversable patches with walls/bumps in front of the robot. Going down, we encounter patches with smaller obstacles. On the plateau, there are traversable patches with small obstacles such as light bumps. Importantly, those patches are the closest ones to the not traversable ones, so they were the hardest to separate, thus, to classify. Going up on the right side, we see some grounds with small steps. Finally, on the top, we find all the downhill patches, the simplest ones to traverse.

### 1.0.3 Test set

We can apply the same procedure on the test set. Since it is a real world quarry, this dataset is harder than the train set and present challenging situations for the robot. The following image shows the features space after reducing its dimension to two using PCA.

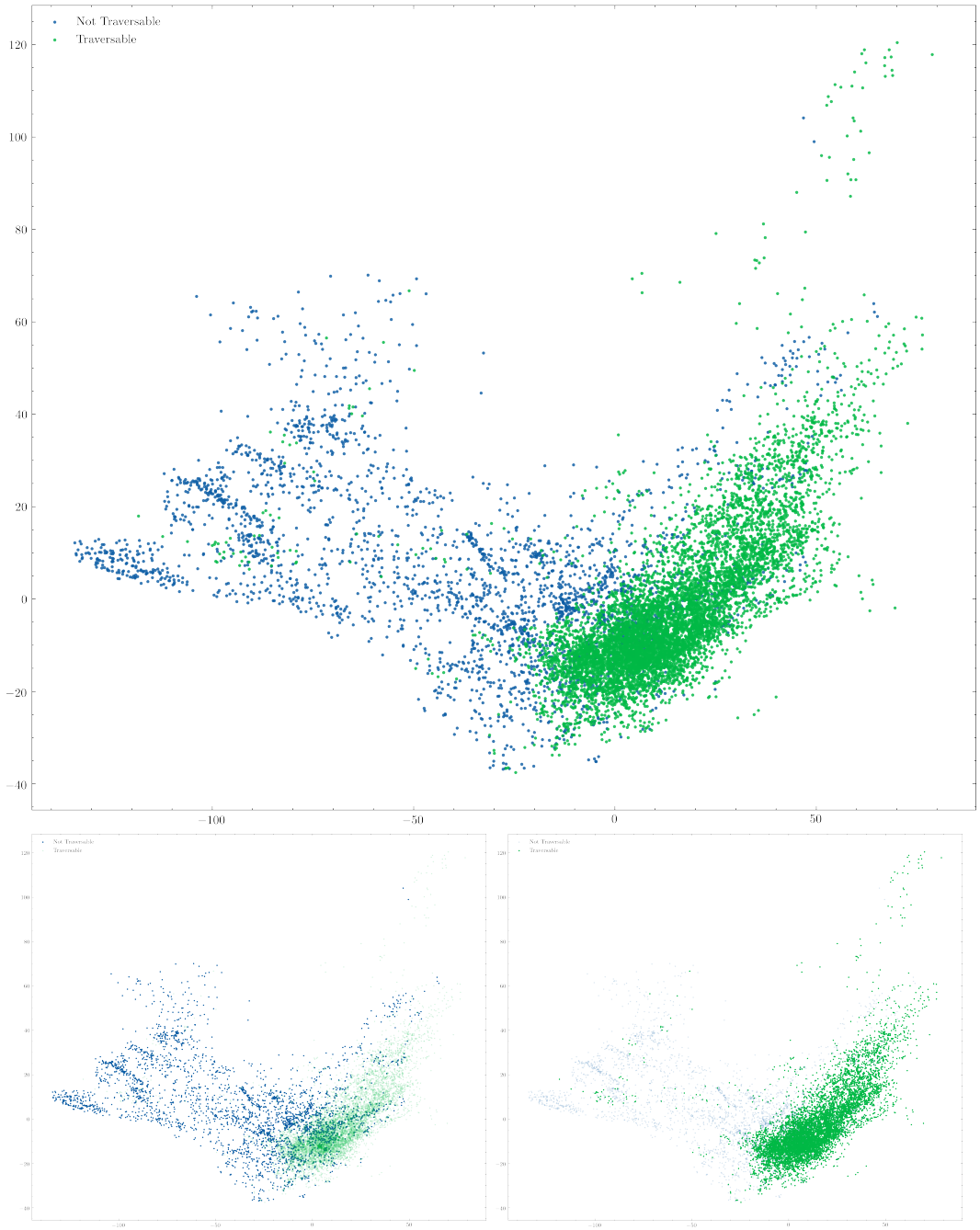
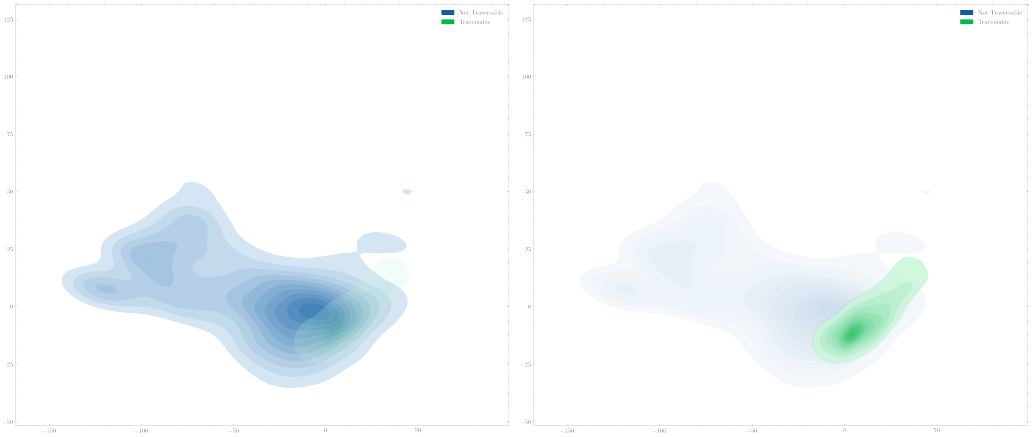



Figure 1.5. TODO

Interesting, the traversable patches are very near to each other, while the others span a very big surface. This suggests that there are many not traversable terrains with different features. The traversable points are clustered near the center, this

implies that most of them share similar features. We plotted the density for each class to better understand where the most points are mapped.



(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 correct learning.

The two centers are really close to each other, making those samples harder to separate and some not traversable points are mixed up with the traversable ones. This explains the elevated number of false negative that lower down the AUC score on this dataset. We can also visualize the patches by plotting them using their features coordinates

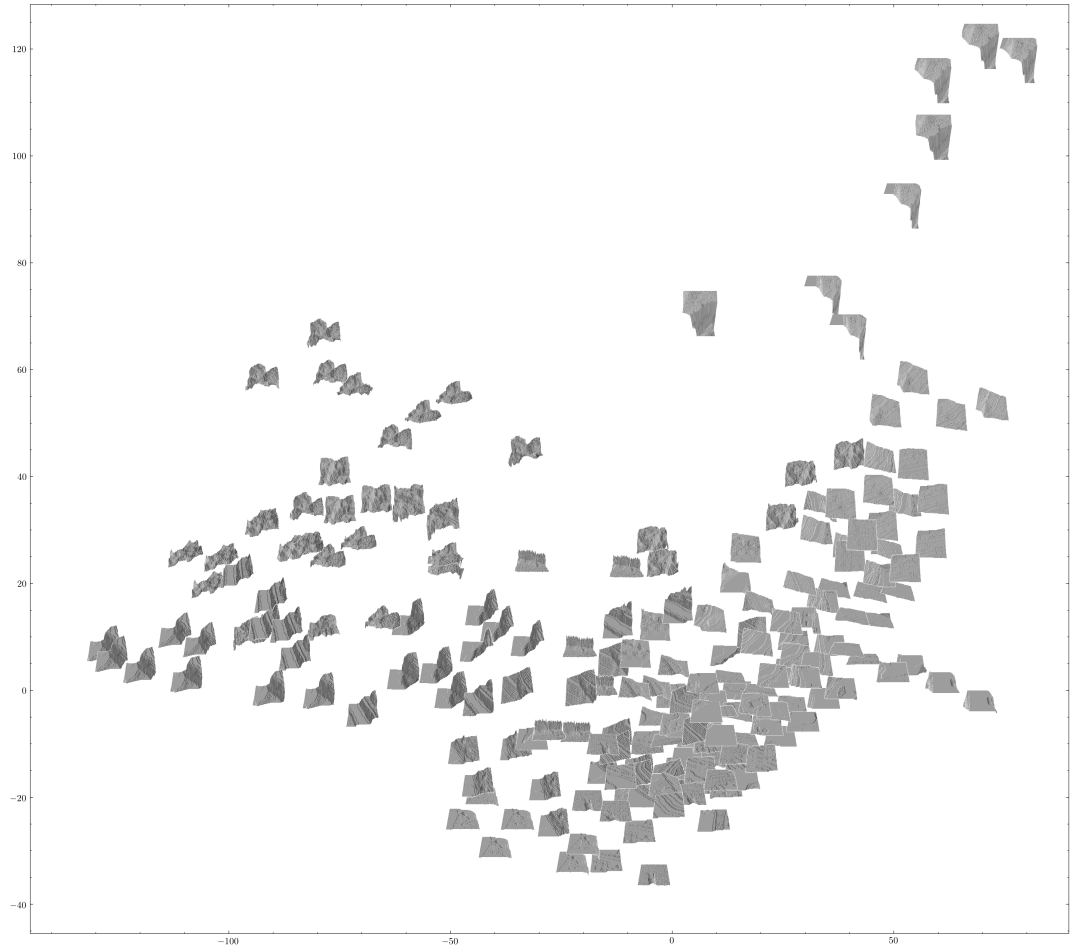



Figure 1.7. Patches that correspond to coordinates in the features space of the last convolutional layers on the test dataset. Similar grounds are close to each other.

**??**

On the top left, from the not traversable cloud, we can see patches with a high level of bumps. Going down we find surfaces with huge walls in front of the robot while going close to the center we start to see all the traversable patches. Those samples have not too steep slopes. If we move to the density center, green double shown in figure **??**, we encounter lots of flat patches with little obstacles. Going up on the right branch we find downhill and on the top there are falls.

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 ẗvisual explanationsf̈or convolutional neural networks. It highlights the regions of the input image that contribute the most to the predictions.

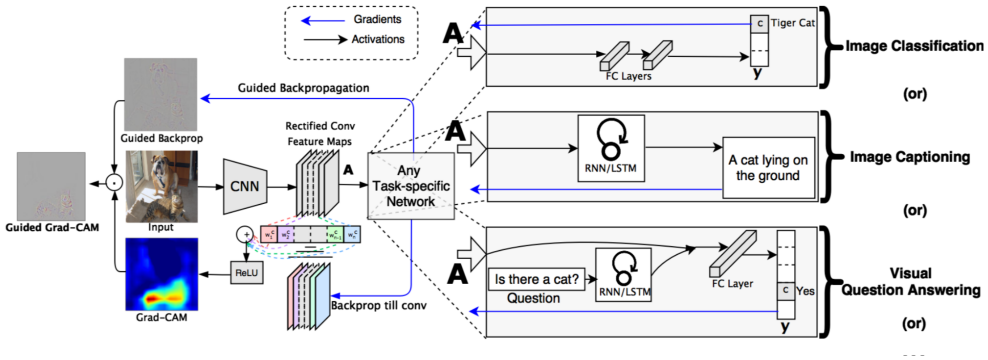



Figure 1.8. Grad-CAM procedure on an input image. Image from the original paper [**gradcam**].

In detail, the output with respect to a target class is backpropagated while storing the gradient and the output in the last convolution. Then, a global average is applied 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. After, each element of the convolutional layer outputs is multiplied with the averaged gradients to create the grad cam. This whole procedure is fast and it is architecture independent.