Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

FAKULTÄT
FÜR MATHEMATIK, INFORMATIK
UND NATURWISSENSCHAFTEN

Master Thesis

Identifying salient regions using point cloud gradient Class Activation Mapping

Dennis Struhs

3struhs@informatik.uni-hamburg.de
Studiengang Msc. Informatik
Matr.-Nr. 6535384
Fachsemester 5

Erstgutachter:	Prof. Dr. Simone Frintrop <frintrop@informatik.uni-hamburg.de>
Zweitgutachter:	Dr. Mikko Lauri <lauri@informatik.uni-hamburg.de>

Abgabe: 26.05.2020

Your brain does not manufacture thoughts.
Your thoughts shape neural networks.
– Deepak Chopra

Contents

List of Figures

List of important terms and abbreviations

Glossary

CAM Class Activation Mapping, introduced by [Zhou et al., 2016], is a method used to visualize discriminative regions in images. The resulting heat-map only shows regions that activate a user defined class.. 14

grad-CAM Gradient class activation mapping was suggested by [R. Selvaraju et al., 2017] and is a refined version of Class Activation Mapping to generate class label dependant saliency maps on 2D images.. iii, iv, 1, 3, 4, 11, 12, 16, 18, 21, 67, 68

LiDAR Is a scanning method that measures distances to a target using laser light to measure the reflected light with the help of a sensor. Differences in laser travel times and alterations in wavelengths can be used to compute 3D representations of the scanned object.. iii, 1

p-grad-CAM The novel implementation of the image based grad-CAM, proposed by this paper, which is able to find regions of interest for a particular user chosen class label working on point cloud data.. i, ii, iv–vii, 4, 25, 27, 28, 30, 32–37, 46, 50–69

point cloud An unordered set of points represented as XYZ coordinates in space.. 2, 3, 11, 40, 41

Pointnet Is a novel neuronal network proposed by [R. Qi et al., 2017] which is able to perform classification, part segmentation and semantic segmentation on point cloud data.. i, iii, 2–5, 7, 15, 16, 19, 21, 23, 39, 40, 57, 65, 68

Tensorflow A very commonly used framework to implement Neuronal Networks in Python and C++. It can be acquired at `https://www.tensorflow.org`. 3, 31, 67, 68

Acronyms

ASM Adaptive Saliency Maps. vi, vii, 56, 57, 62–69

CAM Class Activation Mapping. 14, 67

CNN Convolutional Neuronal Network. i, iii, 1, 3–6, 11–15, 27

DNN Deep Neuronal Network. 15

GCNN Graph Convolutional Neuronal Network. 11

LiDAR Light Detection and Ranging. 2

NiN Network in Network. 14

p-grad-CAM Point cloud gradient Class Activation Mapping. i, iv, vi, 21, 25–30, 32, 34, 36, 38, 39, 41, 51, 65, 68

ReLU Rectangular Linear Unit. 22

RGB Red Green Blue. 6, 23

WTA Winner-Take-All. 13

1 Introduction

In recent time Convolutional Neuronal Networks (CNN) became increasingly popular in many Computer Vision applications. Camera based visual perception applications process images with the help of CNN, to extract features into computer readable formats. Without CNN this would not be possible, as they play a major role in extracting important information from images via data-driven learning to handle different tasks. Unfortunately this is achieved by trading high performance for transparency and interpretability of the network. Approaches such as gradient Class Activation Mapping (grad-CAM) [R. Selvaraju et al., 2017], help closing the transparency gap. Grad-CAM is able to highlight the image areas that trigger a classification of a particular class label. However pixel data images are usually not well suited for dealing with depth information because if captured with a mono camera, they lack the spatial information to create a 3D metric space. It is possible to encode depth data as pixel color data to create depth images. In such a depth image each pixel contains the distance of each point to the camera center, usually measured in millimeters. However this still does not represent a full 3D metric space because the view is locked to the camera position and cannot be dynamically changed. There exist approaches to create depth images via cameras in a stereoscopic configuration and using depth estimation, like shown on Figure 1.1. While being an improvement it also suffers the same issue of not truly representing a full 3D environment and are yet again being restricted to the location and view of the camera.

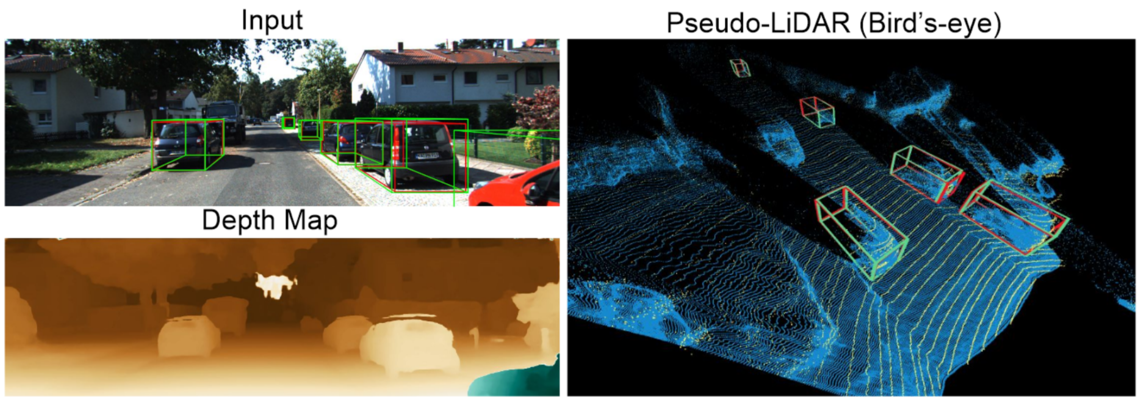


Figure 1.1: Creating a LiDAR like depth image using stereo cameras and depth estimation. [Wang et al., 2019]

Another option to represent depth information are so called point clouds[1]. Each point is represented as X,Y,Z coordinates in space. These point clouds yield the advantage of containing geometry information in a metric 3D space. Point clouds can easily be subject to translation, rotation and every other kind of real time 3D transformation. This is particularly helpful as any results from the network are also in a metric 3D space and thus all spatial information is preserved, unlike pixel data images. Point cloud data can be obtained by using Light Detection and Ranging (LiDAR) applications, stereoscopic images or structured light, which is used in the popular Kinect^{TM}sensor family. Despite these advantages, point clouds were less popular in common Machine Learning applications, compared to 2D image based applications.

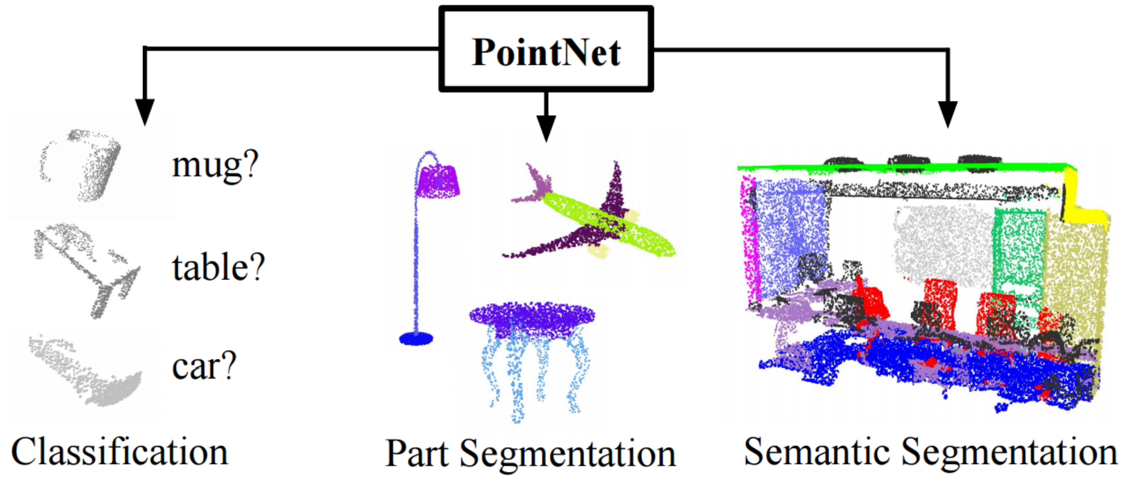


Figure 1.2: Available network types of the Pointnet framework. [R. Qi et al., 2017]

This changed when Pointnet [R. Qi et al., 2017] first proposed a network structure for object classification, semantic segmentation and part segmentation tasks using point clouds. Despite its recent inception it has inspired many other researchers to work with point clouds. Examples are Pointnet++ [Qi et al., 2017], Voxelnet [Zhou and Tuzel, 2018], Deep Kd-Networks [Klokov and Lempitsky, 2017], Frustum PointNets [Qi et al., 2018] and more. Pointnet currently displays over 1000 citations and counting. It has moved point clouds into the spotlight for many new interesting research projects, this paper included. The main contributions of this paper are as follows:

- A novel algorithm, that is able to highlight salient points, responsible for the classification result of a user defined target class label.
- A thorough explanation of the mathematical operations performed by our new approach and an ablation study to justify the design choices.
- An in depth discussion and test of our approach compared to a similar method that can find salient points in point clouds, to prove the validity of our method and to investigate the run-time performance of both algorithms.

[1] `https://pointcloudlibrary.github.io/`

1.1 Motivation

Convolutional Neuronal Networks (CNN) are so popular because of their versatility for a wide range of problems that are too complicated to be solved with hand-crafted features. Object classification is a particular popular task where CNN do a great job of solving the problem quickly and with high accuracy. Current state-of-the-art image classification networks can often even qualitatively outperform humans while being able to process large amounts of images very quickly. However a major issue of CNN is the aforementioned lack of transparency. The important decision making parts of Neuronal Networks are barely interpretable by humans because of the complexity and the usual huge amounts of network parameters. This makes it difficult to get an explanation of the decisions taken by the network, which is particularly dubious and even dangerous considering their increasingly growing influence on decision making on ever more delicate subjects. A bad decision could cause financial damage or, in the case of autonomous driving, even be lethal to other traffic participants. Point cloud based networks are not spared either and suffer from the very same issues even if their architecture differ from a CNN.

This lack of transparency is a major hindrance on trust in these new technologies, rightfully so if their behaviour may prove uncontrollable in certain situations. It would be nice to know which parts of an image or point cloud made the underlying network do a certain decision, so both developers and users gain a human readable insight about the inner workings of the network. For 2D image based networks there already exist various visualization approaches, that help to alleviate this lack of transparency. However for point clouds barely such visualization algorithms or frameworks exist at all. This leaves point cloud base applications in a lacking spot, as such valuable insights are just as valuable for point clouds as they are for pixel images.
Therefor our approach aims to provide a tool to increase both the transparency and provide the ability to gain a human readable clue of which points drive the classification decision making of a point cloud based classification network. It is inspired by the pixel image based gradient Class Activation mapping grad-CAM[2] [R. Selvaraju et al., 2017] approach. As a starting point the Pointnet[3] [R. Qi et al., 2017] network is used to extend it with a grad-CAM like visualization for point clouds, that shows the regions of interest from the perspective of the network for a particular class label. It is trained using the `modelnet40_ply_hdf5_2048`[4] database which contains 40 objects with different shapes represented as point cloud data. As the machine learning engine the Tensorflow[5] v1.8.0 framework, with CUDA[6] v9.0 and cudNN[7] v7.3.1 is used.

[2]`https://github.com/ramprs/grad-cam`
[3]`https://github.com/charlesq34/pointnet`
[4]`https://shapenet.cs.stanford.edu/media/modelnet40_ply_hdf5_2048.zip`
[5]`https://www.tensorflow.org/`
[6]`https://developer.nvidia.com/cuda-90-download-archive`
[7]`https://developer.nvidia.com/cudnn`

1.2 Thesis goal

This thesis pursues to achieve the following goals:

1. Investigate if the pixel image CNN based grad-CAM concept is also applicable to point clouds in 3D space using Pointnet.
2. Create a human readable visual representation of important points for the network via our novel algorithm named p-grad-CAM[8].
3. Discuss the design choices that were made for p-grad-CAM[8] during development.
4. Test the correctness of our implementation, by evaluating it from two perspectives:
   - **Testing against rotation**: This shows if the accuracy of our approach is affected by rotation or not. This is an important attribute because real world data objects are usually not perfectly aligned. Our algorithm therefor must not be affected in any way from randomly rotated objects in any direction.
   - **Ablation study**: It must be proven that grad-CAM indeed is able to find salient regions. In this test, points that were found important by grad-CAM are removed. If the relevant points were indeed found, a noticeable drop in accuracy would be expected. Further on it must be verified that our approach also yields better accuracy and loss results compared to just removing random points.
5. Compare our implementation with a state-of-the-art algorithm that also finds salient regions and measure their runtimes and accuracy. With the accuracy being measured by the amount of total points removed per algorithm. The fewer points necessary to make the network prediction deviate, the better.

[8]`https://github.com/Fragjacker/Pointcloud-grad-CAM`

2 Fundamentals

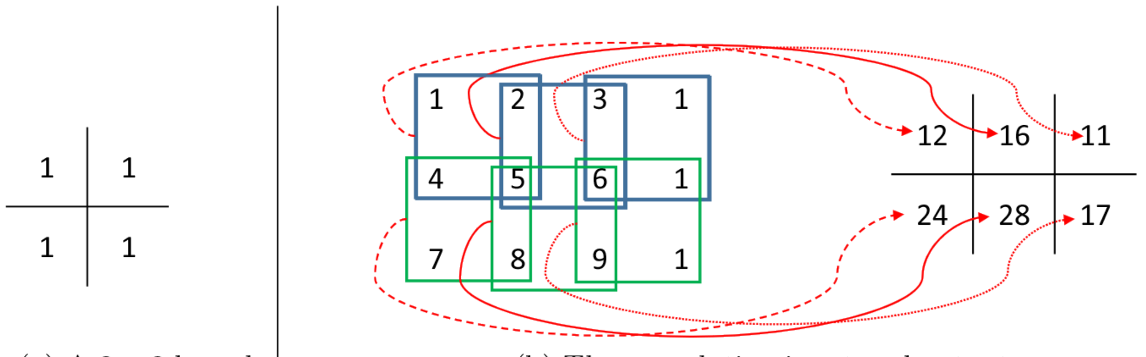
Neuronal Networks in general, as complex as they may be in practice, boil down to fairly simple mathematical concepts and operations. Such networks are often represented as matrices that can assume thousands if not millions of entries, each entry representing a single neuron of the network. Neurons are connected to each other via "synapses" that convey signals from one Neuron to another. These synaptic connections may change their topology, which are modelled via weight values stored in a matrix for each neuron. These weights ultimately determine the behavior of the network and are subject to change when a network is trained.

Every change of the neuron weights is modelled via a sequence of matrix multiplication which are deliberately modelled to produce the desired results defined by the update function. These matrices are the building blocks of so called layers that provide the actual functionality of the Neuronal Networks performing the machine learning tasks. The basic idea of this layering of Neuronal Networks is to have some input layer that represent the user input data, the hidden layers in between that perform the actual machine learning, using the input data and finally the output layer. The output layer is responsible to translate the data from the Neuronal Network back into a user readable result.

To achieve this, the dimensional complexity of the underlying hidden layer matrices needs to be reduced in such fashion, that the desired information emerges from it. One such method of dimensional reduction is called pooling. The idea behind pooling is to use the most important values, chosen by some a priori metric, and "pool" them into a single value to build a reduced matrix out of an initially larger one. Our approach makes use of pooling as well, hence several relevant pooling methods are introduced and discussed in this chapter.
There is also a short introduction to CNN and Pointnet, which are relevant because their concepts and algorithms are partly picked up and adapted by our approach as well. In addition there is a brief overview of the relevant averaging methods. Averaging is used by our proposed algorithm to compute a threshold value based on the input data to classify points into low and high scoring. This is important because our proposed algorithm works by removing the important high scoring points until the prediction deviates from the ground truth.

2.1 Convolutional Neuronal Network (CNN)

In the field of deep learning, which is a subcategory of machine learning, the term convolutional neural network (CNN or ConvNet) defines a class of deep neural networks, commonly used in the context of analyzing visual 2D pixel data images. As the name suggests, CNN make use of the mathematical convolution operation. The architecture of CNN usually consist of an input layer, several hidden layers and one output layer. The hidden layers consist of multiple convolutional layers that convolve the input image via multiplication or other kinds of dot-product operations. The input data for an CNN usually consists of a tensor with order 3. In case of an image this would represent height, width and 3 Red Green Blue (RGB) color channels per pixel. It is possible to expand the order of the tensor to add more dimensions encoding additional data. Ultimately all tensor inputs are handled similarly by the CNN. Input data is sequentially passed through all layers starting with the input layer and ending with the output layer. In between exist the actual processing layers. These could each be either a convolution, pooling, normalization, fully connected or a loss layer. Albeit the convolutional layers are commonly just addressed as *convolutions*, the actual performed operation is the application of a $N \times N$ kernel, successively moved over all pixel values of an image, see Figure 2.1.



(a) A 2×2 kernel (b) The convolution input and output

Figure 2.1: Illustration of applying a CNN convolution kernel to an image. All values within the kernel frame are summed up into a single numerical value. It is possible to weight certain pixel indices if the coefficients in the kernel are changed into numbers other then 1. [Wu, 2017]

CNN technically are multi-layer perceptrons with fully connected layers. The term fully connected means that every neuron from one layer is connected to each neuron of the next layer, similar to a fully connected graph. This kind of neuronal network was originally inspired by the biological analogy of the visual cortex and its neuronal connection layout. The advantage of a CNN compared to traditional image processing techniques, is its ability to learn filters that had previously to be hand crafted. Therefor it requires a lot less pre-processing of the input image sets, which makes it able to work with a wider spectrum of image data and shorter runtimes, compared to traditional image processing methods. For a much deeper and more technical introduction to CNN see [Wu, 2017].

2.2 Pointnet

In the past, when researchers wanted to work with 3D point data they had to transform it either to voxels (pixels with a volume) or sets of images. This had the disadvantage of bloating the data sets in size and also caused side effect issues because information was lost or distorted during the transformation into another data structure. It was a persistent issue that hampered the potential of 3D point cloud data in neuronal networks. Pointnet [R. Qi et al., 2017] was amongst the first, if not the first novel network that directly worked with point clouds, without any intermediate conversion, as input data. It came with full support for object classification, part segmentation and scene semantic parsing tasks. It did not only show the proof of concept for point cloud based neuronal networks but also provided a highly efficient a network with high performance on par with state-of-the-art. Pointnet natively accepts point clouds as input data and outputs either a class label that characterize the input as a whole, or labels each point contained in a segmented point cluster.
The architecture of Pointnet itself however is comparatively simple because in the initial stages of the network processing pipeline, each input point is handled identical, yet independently. Each input point is defined at least by three coordinates (x, y, z), with the option to expand it with additional dimensions that represent local normals or any other desired local or global attribute. These point sets $\in \mathbb{R}^n$ come with the following properties:

- **Unordered**. This means that any network using point clouds must be invariant to $!N$ possible permutations of the input data.
- **Interacting**. Points form meaningful structures in spatial space and networks must be able to find local structures using nearby points.
- **Transformation invariant**. It requires that any network prediction must not be influenced by transformations such as translation or rotation.

The key-method that is necessary to make Pointnet work is the application of max pooling, which is explained in the next section. What basically happens is that Pointnet learns a collection of optimization criteria, which are able to localize important points that define the characteristic of the input and provide the reason for their selection. Ultimately the output layers compose those learnt optimal criteria into the description vector that classifies the entire point cloud object. The advantage of using point clouds directly, is that it can be subject to rigid or affine transformations at any given time because each point is transformed independently of each other and no information is lost due to intermediate file format conversions into other data types. This alone also provides an easy method to perform data augmentation because variations of point cloud data can be easily created by applying affine transformations to the input data before feeding it into the network during the learning phase.

2.3 Max Pooling

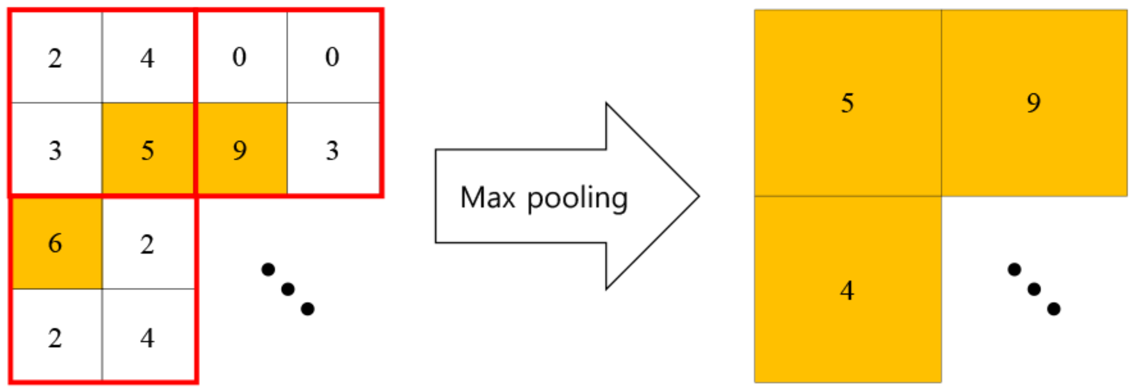


Figure 2.2: Max pooling reduces the feature vector values by picking the highest value. [Hyun et al., 2019]

Max pooling is amongst the most common of pooling operations. It takes the highest value from the sample range and returns it. It is the best approach to capture single very high scoring values that denote synaptic connections of great importance.

2.4 Stride Pooling

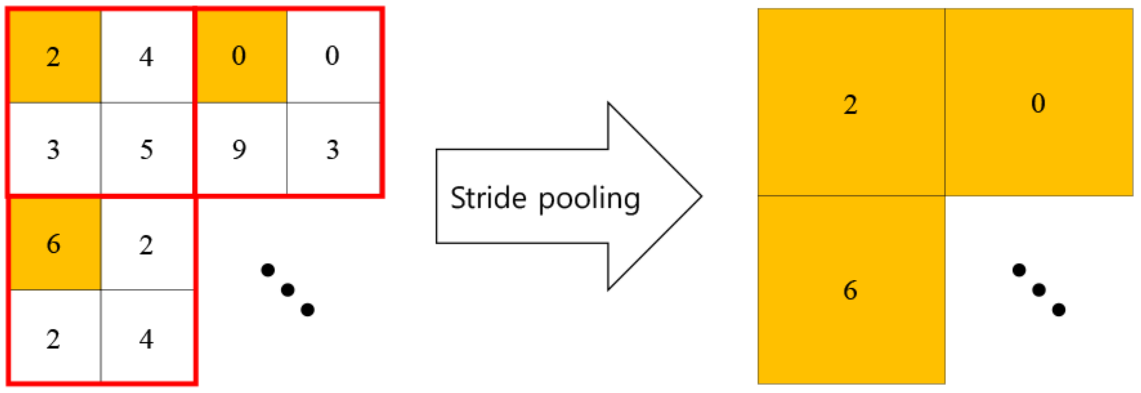


Figure 2.3: Stride pooling reduces the values by picking always the same element at index N. [Hyun et al., 2019]

Stride pooling chooses always the same index which is a pre defined constant index i. This pooling method is rather uncommon because it is inflexible and requires manual fine-tuning of i.

2.5 Average Pooling

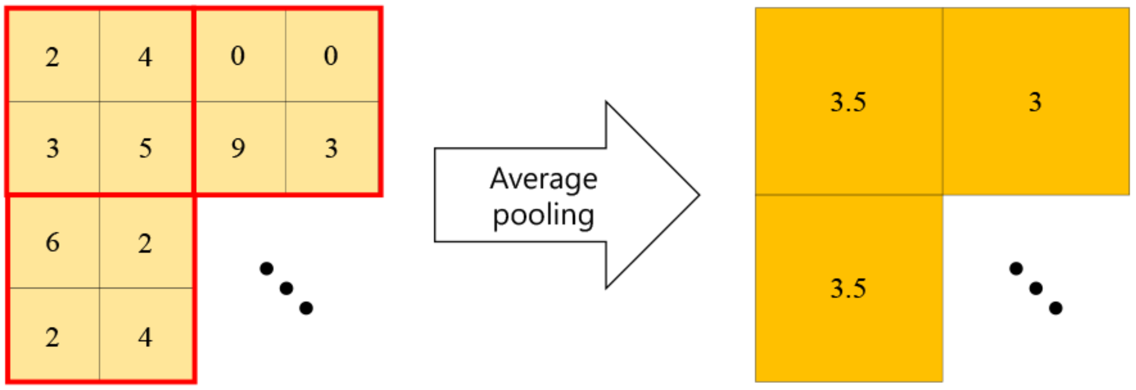


Figure 2.4: Average pooling reduces the feature vector entries by averaging them into one value. [Hyun et al., 2019]

Average pooling calculates the arithmetic mean value over all sample values and returns it. This approach is quite common but has the potential issue that it often produces floating point numbers as results, which may or may not be desired. The average will also not capture the few highest scoring values, if the majority of values are low.

2.6 Arithmetic mean value

The arithmetic mean is amongst the most common averaging methods. It is defined as:

$$v_{mean} = \frac{1}{n}\sum_{i=1}^{n} a_i = \frac{a_1 + a_2 + \cdots + a_{n-1} + a_n}{n} \tag{2.1}$$

2.7 Median value

The median value is another commonly used averaging technique. It is defined as the middle value of an ordered set. If the number of items in the set is even, the average of the two values next to the imaginary center are used. For example let there be the following list of values $14, 9, 2, 5$. These values first need to be sorted which yields $2, 5, 9, 14$. Since the numbers of value is even there is no middle value in the list. The values 5 and 8 are next to the center therefor the median would be the average of these values $\frac{5+9}{2} = 7$.

2.8 Midrange value

The midrange value is simply the arithmetic mean of the lowest and the highest value of a given set of values.

$$v_{mid} = \frac{1}{2}\big(\min(x) + \max(x)\big) \tag{2.2}$$

3 Related Work

The lack of transparency is a generally common issue in neuronal networks as a whole, which pretty much all networks suffer from. For example let us look at Graph Convolutional Neuronal Network (GCNN)s [Kipf and Welling, 2016] which are a variation of CNN which use node graphs instead of images for their tasks. Despite their difference over the architecture of the used neuronal network, the issue of lacking transparency remains the same. There exist several methods to increase transparency which are often designed for CNN because to their huge popularity. However their foundation mathematical concepts are often generic in nature and therefor yield the possibility of being applicable to other neuronal network types as well. However the question if they actually do also yield good results in other neuronal network types as well remains unanswered. The paper of [Pope et al., 2019] tackles these questions and shows how these concepts are not only still valid and applicable for GCNN but also useful to increase their overall explainability. They were able to deliver the proof of concept for classification tasks, using CAM and grad-CAM in visual scene graphs and molecular graphs. They provide a positive answer for their case to the question if grad-CAM can indeed be applied to a GCNN, raising the confidence that this may be possible as well with 3D point clouds. In the following sections the most common methods to increase transparency and explainability are briefly introduced, independant of their intended target network architecture.

3.1 Contrastive gradient-based Saliency Maps

The paper, proposed by [Simonyan et al., 2014], offers two main methods for visualization, which were designed to be used with CNN. They are named class model visualization and Image-Specific Class Saliency Visualisation. Both methods fall into the category of Contrastive gradient-based Saliency Maps [Pope et al., 2019] and are relevant because the base principles, which are employed in both methods, are the foundations for the more sophisticated approaches discussed later in this chapter.

3.1.1 Class Model Visualization

This method generates a colored image that visualizes the class scoring model that has been learned by the network. It represents the pixel shape templates for the convolution, that are linked to a certain class label. The base idea is that if $S_c(I)$ is the score of some class c, calculated by the classification layer of the network for some input image I, then the goal is to find a L_2 regularised Image, where the score $S_c(I)$ is maximized:

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2 \tag{3.1}$$

Here λ is the L_2 regularization parameter. The class model visualisation can then be found using back-propagation similar to the process used to optimise the layer weights. The main difference here is that the optimization uses the input image and leaves the learned layer weights untouched. The resulting image of the class model can be seen on Figure 3.1.

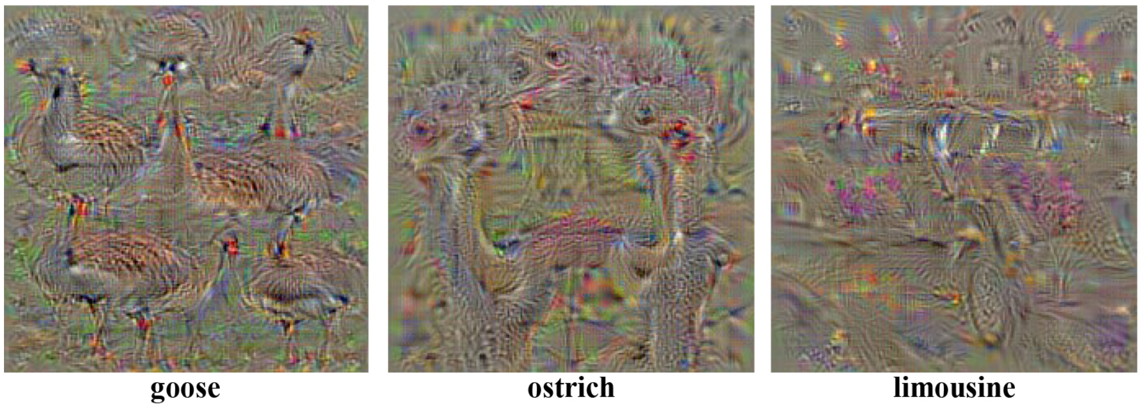


Figure 3.1: Example of learned class appearance model images by a CNN. [Simonyan et al., 2014]

3.1.2 Image-Specific Class Saliency Visualisation

This approach works by computing the score $S_c(I)$ via the gradient w of the class score with respect to the input image, with the desire to rank the pixels regarding their influence on the score $S_c(I_0)$. However this time the $S_c(I)$ score function is a non-linear function of I. Fortunately it can be approximated via the first-order Taylor expansion:

$$S_c(I) \approx w^T I + b \tag{3.2}$$

$$w = \left.\frac{\partial S_c}{\partial I}\right|_{I_0} \tag{3.3}$$

The result is a greyscale image with bright areas denoting salient regions, see Figure 3.2 on the facing page. This idea of using gradient back-propagation is very commonly used and was also picked up and refined by the grad-CAM approach, which will also be introduced in this chapter. Unfortunately this approach was specifically designed to be used in conjunction with CNN and is therefor, in it's current form, not applicable as a solution to be used with point clouds, since the architecture of the used networks are too different. Albeit the concept of using gradients to compute score values itself, is universally applicable in other networks. The grad-CAM and in conclusion our algorithm as well relies on this concept.

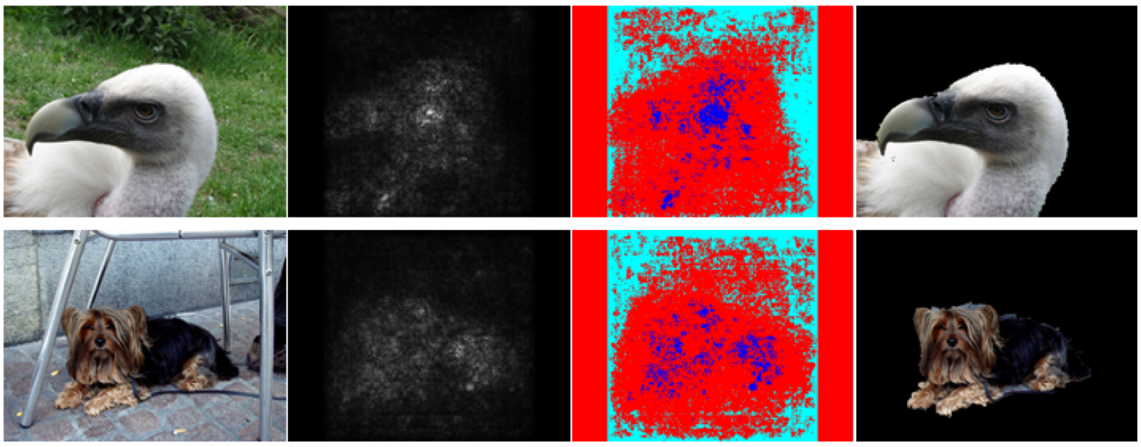


Figure 3.2: Contrastive gradients based Saliency Map, used to isolate the predicted class. [Simonyan et al., 2014]

3.2 Excitation Backpropagation

This method, introduced by [Zhang et al., 2016], works by identifying the neurons that are driving the current neuronal network task in a CNN. They collect the input stimuli from the output combined with a top-down Winner-Take-All (WTA) method to find the relevant neurons for a given data input. This combination of two methods is called Selective Tuning model, which is essential for their method. With the help of the Selective Tuning model, they are able to postulate a probabilistic WTA process, they call *Excitation Backpropagation*. This idea of finding relevant neurons in the network is also commonly used in many other approaches. One example is the critical routing paths method, introduced in the next section, which seeks to find important neurons in the network, yet with a different implementation. Another positive attribute is that *Excitation Backpropagation* works solely with the neurons and their connections inside the network. This makes it very viable for a portation to different networks types because it is generic and network type independent. This is one reason why this concept is also commonly used in various machine learning applications.

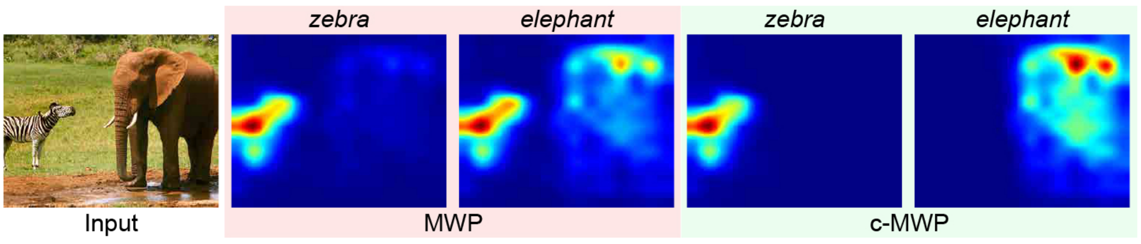


Figure 3.3: Salient regions found by excitation backpropagation. [Zhang et al., 2016]

3.3 CAM and critical routing paths

The original idea for the Class Activation Mapping (CAM) was proposed by [Zhou et al., 2016]. They achieved the visualization of class-discriminative regions in images, specifically for a Convolutional Neuronal Network (CNN). The approach is based upon the observation, that convolutional units inside of CNN naturally act as object detectors even without supervision about the location of objects. This useful attribute is unfortunately lost if these convolutional units are used with fully-connected layers in the network. This issue can be avoided if fully-convolutional networks are used instead. That approach also has the advantage of reducing the number of parameters of the network. The fully convolutional Network in Network (NiN)[Zhou et al., 2016] for example also uses global average pooling to prevent over-fitting during training. It turned out that another advantage of global average pooling is, that with a little tweaking, the network can retain localization ability of the convolutional units until the last layer. This allows to easily find important regions in images, see Figure 3.4. Unfortunately this entire approach is tailored towards image based CNNs. A different method, suggested by [Wang et al.,

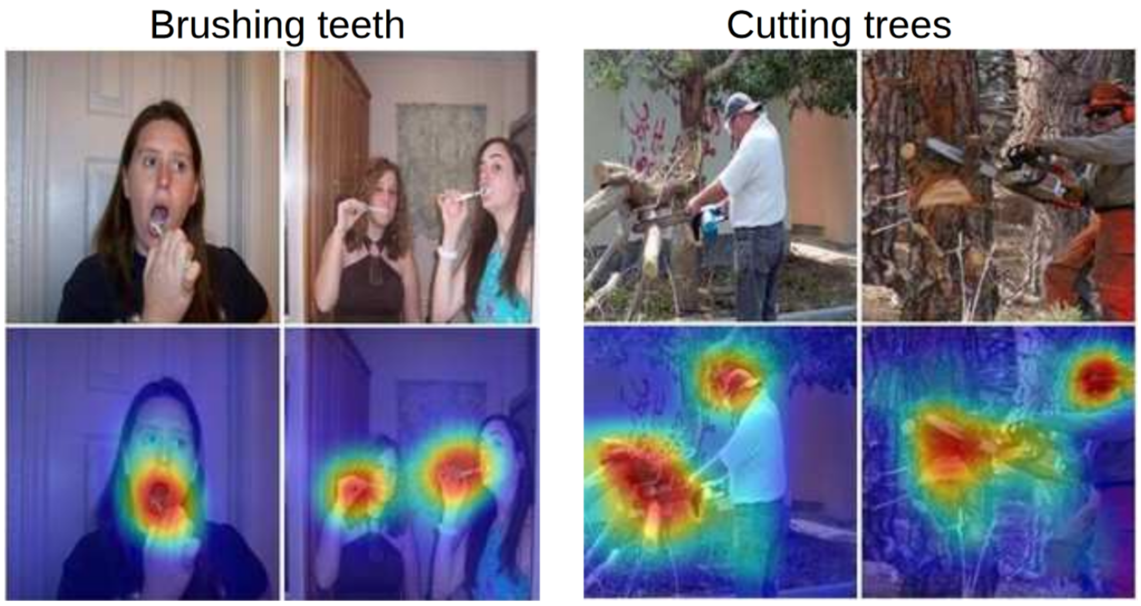
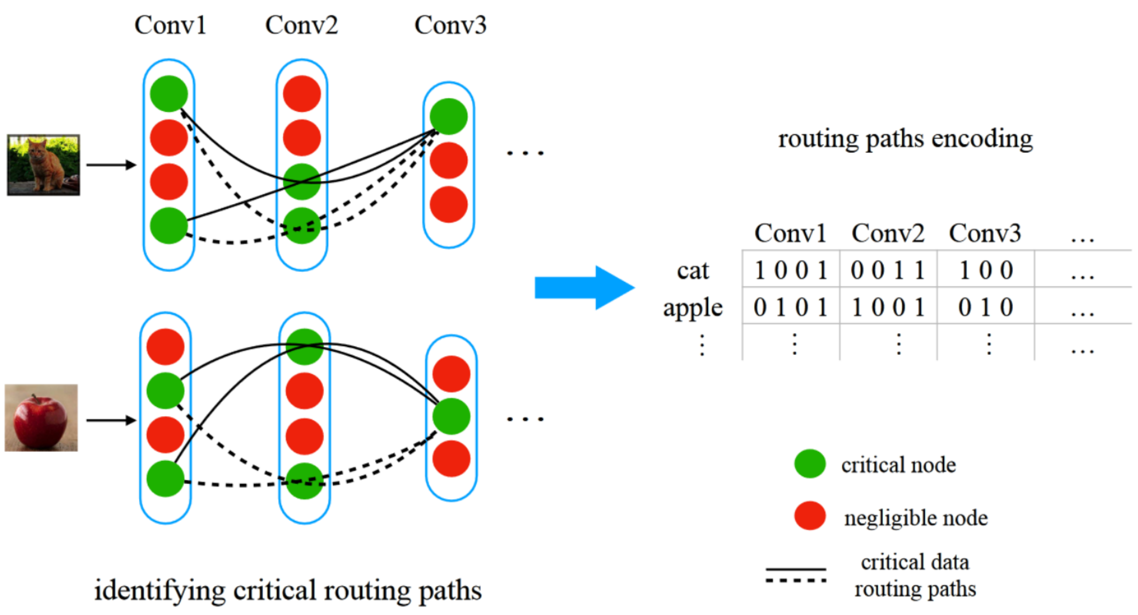


Figure 3.4: Application of CAM on a CNN to identify discriminative regions in images. [Zhou et al., 2016]

2018], works by identifying nodes in a neuronal network, that are important for the decision making process, see Figure 3.5 on the facing page. These critical nodes are defined as important layer result outlets. As such if their contribution would be filtered out completely, the performance of the network would drop immensely. This is accomplished by slightly changing the network via adding scalar control gates. These control gates are put behind each layer to use their output for learning the optimal routing paths for all individual data samples fed into the network. It is another interesting approach that does not require retraining the network as well. This approach however is designed towards architectures used for image based Deep Neuronal Network (DNN) applications. It is also not designed per-se to yield a visual representation of salient regions. Its purpose is to identify adversarial image samples by using their novel approach. It would be interesting however to see how much this approach could be adapted to perform similar tasks with point clouds. This is not yet in the scope of this paper.



	Conv1	Conv2	Conv3	...
cat	1 0 0 1	0 0 1 1	1 0 0	...
apple	0 1 0 1	1 0 0 1	0 1 0	...
⋮	⋮	⋮	⋮	...

Figure 3.5: Critical data routing paths and the encoding feature of the routing paths. [Wang et al., 2018]

3.4 grad-CAM

The work of [R. Selvaraju et al., 2017], which our approach was mainly inspired by, generates a visualization of discriminative regions, e.g. regions of importance for a certain class label, in images by summing up the class score "weights" of all feature maps and then filter negative scores. This successively generates gradients over all involved layers. The resulting heat-map only shows regions that activate a user defined class. Regions with a red color denote higher scores for a given class, whereas blue regions denote lower scores, see Figure 3.6 on the next page. Another advantage of this approach, is that even though it was originally designed for CNN, it could be possibly adapted to be used with point cloud data, due to the generic nature of the approach.
Another advantage of this approach is, that the network does not need to be retrained in order to fetch the relevant score weights because they are imbued inside the feature maps, which are retained via the gradients. We are particularly interested in this quality of the approach and seek to combine it with the Pointnet, see Figure 3.7, network suggested by [R. Qi et al., 2017] in order to create a similar heat-map but for point cloud applications. However since Pointnet does not contain feature maps but a feature vec-

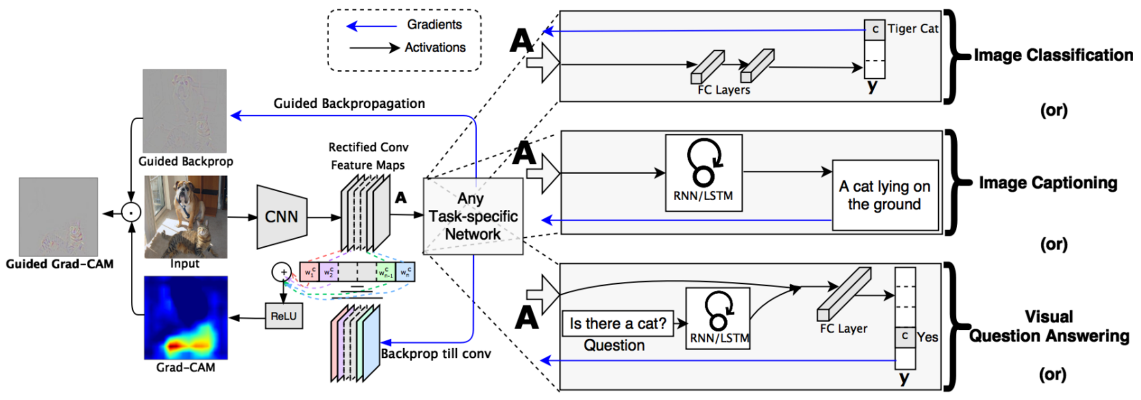


Figure 3.6: Computation of the gradient Class Activation Mapping (grad-CAM). [R. Selvaraju et al., 2017]

ture, that is passed through the network, it is required to adjust the grad-CAM approach to fit the specifications of the Pointnet architecture.

Pointnet was chosen because it provides methods and algorithms to perform deep learning on unordered point sets. It is very robust and yields very high performance that reaches up to state of the art quality or better.

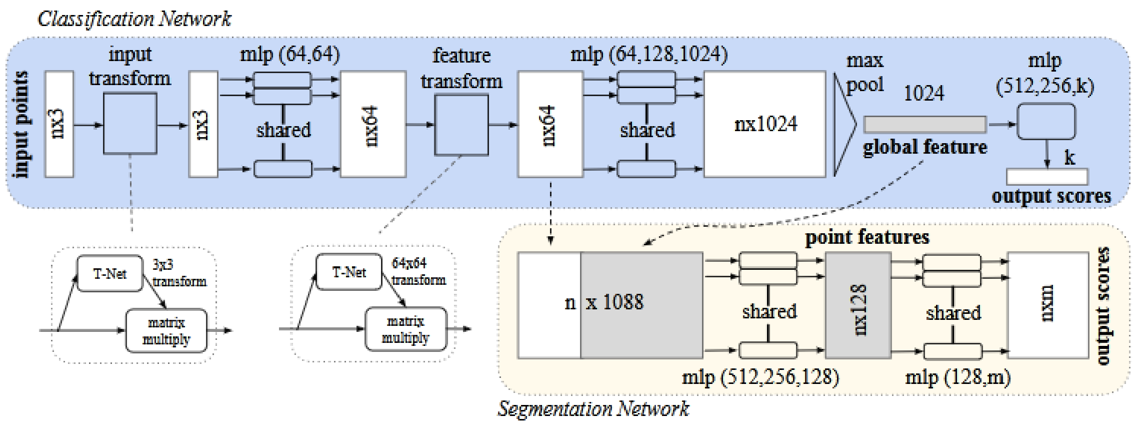


Figure 3.7: **Relevant Pointnet architecture:** The object classification network takes n points from a point cloud as input and performs input and feature transformations. Finally it aggregates the point features via max pooling. The output is classification scores for k classes. [R. Qi et al., 2017]

For our approach we use the basic idea of the grad-CAM approach by [R. Selvaraju et al., 2017] and combine it with the Pointnet [R. Qi et al., 2017] network, to build our own implementation that yields similar qualities as the original grad-CAM method but for point clouds. However as the two networks are different in their architecture, this needs to be accounted. Whereas the original grad-CAM uses feature maps to obtain their weights, Pointnet does not have such feature maps but uses feature vectors instead. This is due the nature of the point cloud which are just N points in space, represented as X, Y, Z coordinates in spatial 3D space. That is a handy attribute because the entire relevant information of the input data can be encoded in a single vector that contains the coordinates for every point existing in space. No further analysis or treatment, like convolution, is required to isolate the relevant information from the input data, compared to pixel data images.

3.5 Saliency Maps

A similar fashioned concept to build Saliency Maps for point clouds, introduced by [Zheng et al., 2018], works by computing the loss for each point in the point cloud. Our approach in comparison is basically "asking" the network for the scores of all points simultaneously, whereas saliency maps probe the reaction of the network removing the points causing the highest prediction loss. They then use the resulting point clouds, to verify the correctness of their approach, as adversarial samples. (see Figure 3.8). Another

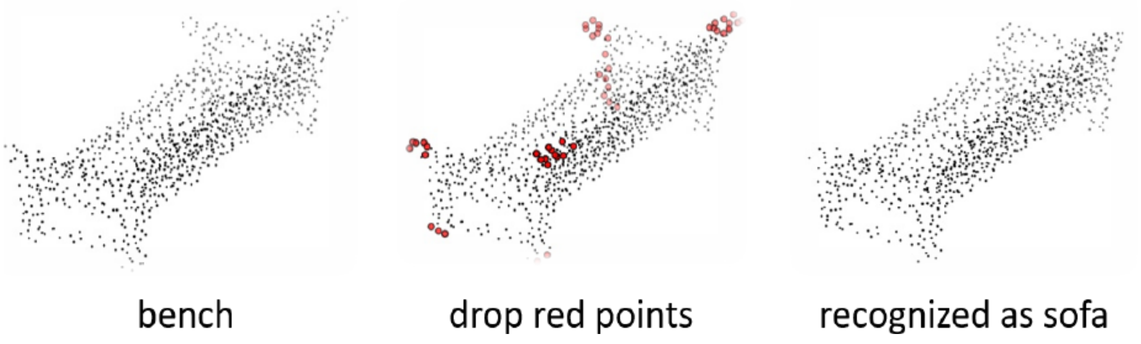


Figure 3.8: Dropping the highest score ranked points identified by the Saliency Map changes the prediction outcome. [Zheng et al., 2018]

difference in our approach compared to theirs is that they use point shifting instead of dropping points. This goes by the intuition that the outwards located points encode the shape information of the object. Therefor moving those points inwards should have similar results on the classification results as the dropping of points (see Figure 3.9 on the next page). Another advantage this approach yields is the complete avoidance of changing tensors during the computation and evaluation of the salience maps, since all the tensors keep their shapes as no points are removed and only change their position.
Based on this point shifting method they are able to approximate the loss contribution for each point by the gradient of the loss, monitoring the change of the prediction loss of the manipulated point cloud objects. In each step of their algorithm the n-highest ranking, with n being number specified a priori by the user, points are removed. This successively yields a loss-ranking of the original unchanged point cloud, which can be used to paint each point to reflect its importance, see Figure 3.10.

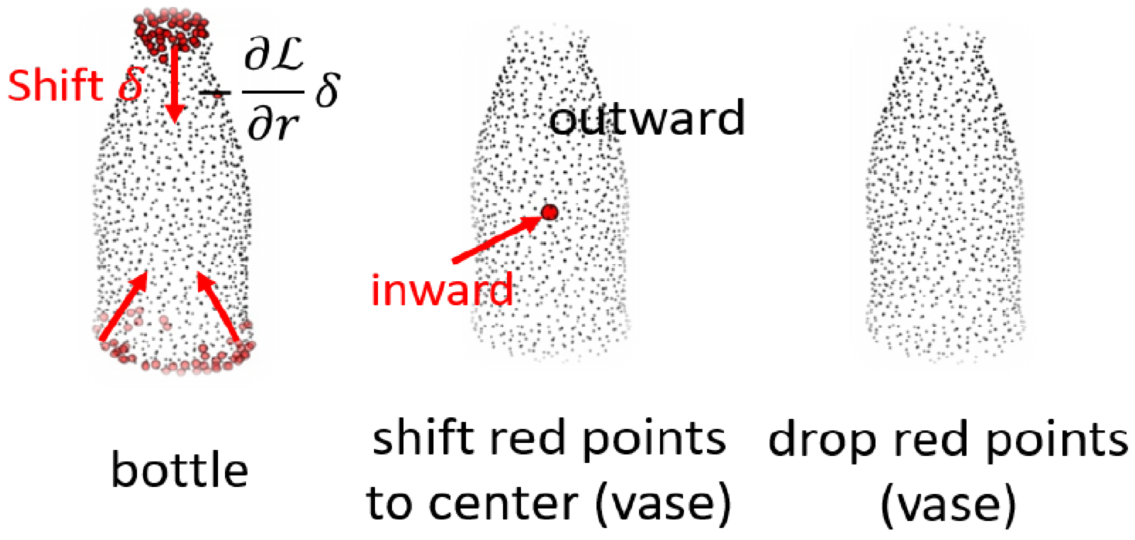

Shift δ
$-\frac{\partial \mathcal{L}}{\partial r}\delta$
outward
inward
bottle
shift red points to center (vase)
drop red points (vase)

Figure 3.9: Achieve similar results as point dropping by shifting the points into the spherical coordinate center of the point cloud. [Zheng et al., 2018]

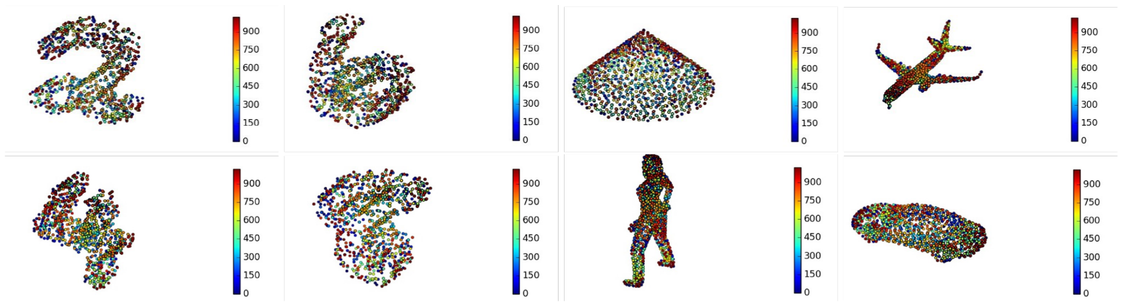


Figure 3.10: Visualization of the Saliency Map, similar to our point cloud grad-CAM approach, colored by their significance in the score ranking. [Zheng et al., 2018]

Since we pursue a similar goal, it would be interesting to see how our approach compares to the Saliency Maps method. This similarity of goals and the usage of the same Pointnet network, which is used by our approach as well, makes it a prime candidate for a comparison of our results with theirs. To achieve this a similar approach, using adversarial samples to attempt deviating the prediction of the network away from the ground truth, is pursued. Since our approach seeks to find the points of interest for the network, the premise of our tests would be that the prediction should change in a similar way as the salience approach does, by removing points with high importance score rankings. If our approach does indeed correctly find important points, it we would expected that our approach yields comparable visual results to the ones produced by the Saliency Maps approach, see Figure 3.11 below.

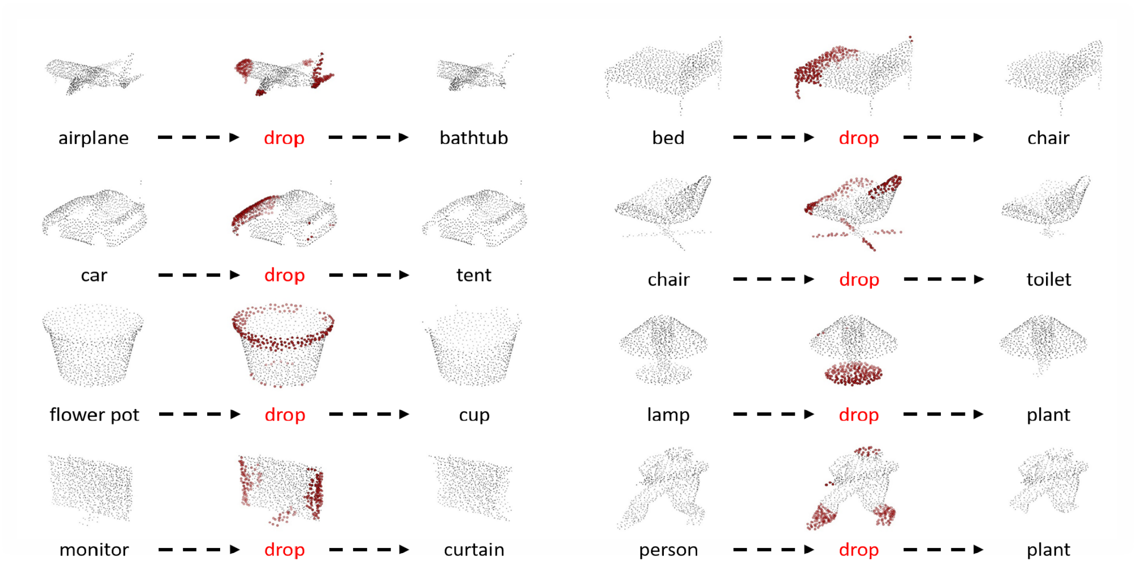


Figure 3.11: Dropping the points with the highest scores. Left image shows the original prediction before the point removal. The right image shows the changed prediction after the removal of the highest ranked points. [Zheng et al., 2018]

4 Approach

The goal of this approach is, to apply the concept of grad-CAM, described in [R. Selvaraju et al., 2017], to the point cloud data based Pointnet network from [R. Qi et al., 2017]. This novel algorithm shall be named Point cloud gradient Class Activation Mapping (p-grad-CAM) to reflect its origin. Since a close as possible replication of the original grad-CAM concept is desired, it is of virtue for this this new algorithm to obtain the score values for all N points of the point cloud. This is necessary to create a point cloud equivalent of feature map back-propagation. Fortunately all important feature and input transforms which make up the contributions of individual points, are embedded in the feature vector. It can be obtained by extracting the $N \times 1024$ feature vector, right before it is fed to the max pooling operation (See Figure 4.1 below). This is important because the feature vector gets reduced after the max pooling operation, losing dimensions and therefor potentially valuable information. After obtaining the score values, the influence of unwanted

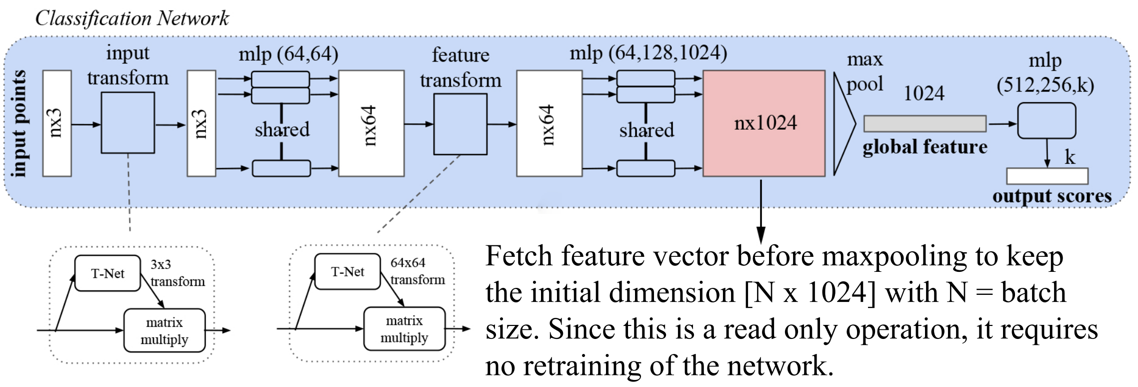


Figure 4.1: Extraction of the feature vector before the maxpooling operation in the Pointnet object classification network architecture. [R. Qi et al., 2017]

class labels are filtered out in the class activation vector of shape $[1, 40]$, by multiplying it with a one hot vector. The desired label of interest must be set a priori by the user. It is an integer value that corresponds to line numbers of the available shapes, specified inside the `shape_names.txt` file, which is supplied by the `modelnet40_ply_hdf5_2048`[1] data base. In the next step the gradient of the score is computed for the class c, y^c of the extracted feature vector, with respect to the filtered target class activation vector A_i^k, i.e. $\frac{\partial y^c}{\partial A_i^k}$. The resulting tensor is of dimension $[N, 1024]$. To obtain the score tensor, that contains the score for each input point of point cloud set N, similar to the grad-CAM

[1] `https://shapenet.cs.stanford.edu/media/modelnet40_ply_hdf5_2048.zip`

approach, pooling is applied which yields a tensor of dimension N. The relevant pooling methods are explained in more detail in section 5.2.1 on page 30. The results of the applied operations yield the condensed final score for each point of the point cloud object:

$$w^c = \overbrace{\max\Big(\underbrace{\frac{\partial y^c}{\partial A^k}}_{\text{Gradients}} \Big)}^{\text{Max Pooling}} : i \in [1, 1024] \tag{4.1}$$

With the score weights available it is now possible to weight the contribution of each point, from the extracted feature vector, towards the desired class activation label only. To achieve this, the original unaltered feature vector, of shape $[N, 1024]$, is multiplied with the average pooled score tensor w^c of shape $[1, N]$. In the following step, all score contribution values are summed up $p_{N,1} \dots p_{N,1024}$ for each row of the feature vector and multiplied by the result of the corresponding score w^c for each point. This is done to have a single value for each point, that represents its cumulative score contribution in the scene which can then be weighted using the results of w^c. To do this the feature vector is multiplied with the score, so that only the relevant feature vector contributions for the current class activation label remain. Now since the dimensions of the score vector do not match the feature vector ones, it needs first to be transposed and vertically expanded by copying the row values into 1024 element big columns. The resulting matrix contains the values of interest on its main diagonal, which are extracted into a new score vector:

$$M(p_N, w_N^c) = \underbrace{\begin{bmatrix} p_{1,1} & \dots & p_{1,1024} \\ \vdots & \ddots & \vdots \\ p_{N,1} & \dots & p_{N,1024} \end{bmatrix}}_{\text{Feature vector } [N,1024]} \cdot \underbrace{\begin{bmatrix} w_1^c & \dots & w_N^c \\ \vdots & \ddots & \vdots \\ w_1^c & \dots & w_N^c \end{bmatrix}}_{\text{Score tensor } [1024,N]} = \underbrace{\begin{bmatrix} w_1^c \cdot \sum_i^{1024} p_{1,i} & \dots & w_N^c \cdot \sum_i^{1024} p_{1,i} \\ \vdots & \ddots & \vdots \\ w_1^c \cdot \sum_i^{1024} p_{N,i} & \dots & w_N^c \cdot \sum_i^{1024} p_{N,i} \end{bmatrix}}_{\text{Score matrix [N,N]}} \tag{4.2}$$

$$= \underbrace{\begin{bmatrix} w_1^c \cdot \sum_i^{1024} p_{1,i} & \dots & w_N^c \cdot \sum_i^{1024} p_{1,i} \\ \vdots & \ddots & \vdots \\ w_1^c \cdot \sum_i^{1024} p_{N,i} & \dots & w_N^c \cdot \sum_i^{1024} p_{N,i} \end{bmatrix}}_{\text{Extract main diagonal values [N,N]}} \mapsto \underbrace{\begin{bmatrix} w_1^c \cdot \sum_i^{1024} p_{1,i} \\ \vdots \\ w_N^c \cdot \sum_i^{1024} p_{N,i} \end{bmatrix}}_{\text{Score values [1,N]}} = T(p_N, w_N^c) \tag{4.3}$$

The final score value vector now contains the corresponding score of each point in the point cloud with respect to the chosen class label of interest. In the final step unwanted negative values are removed, by applying a Rectangular Linear Unit (ReLU) on the tensor. Doing this, removes any scores, that are not related to the selected class label and therefor do not influence the decision making of the network positively. This score tensor is then used to color the point cloud points in a similar heat-map like fashion as in the grad-CAM approach. Points with a red color indicate a high score while points with a green color indicate low scores. Black points indicate scores that are zero, this means that there were no gradients flowing back from the network. The design choices for this approach will be tested and discussed in more detail in Chapter 5.2 on page 30 via an ablation study.

4.1 Point coloration

Since the approach to compute the score weights yields arbitrary numerical values, it is of virtue to find a way to transform this information into usable color value ranges. The Open3d[2] library for Python provides the point cloud class that handles rendering and coloration of points. The color information is represented as Red Green Blue (RGB) information, which must assume floating point values between one and zero. To translate the score weights into Open3d compliant color values, the maximum score value, found in the gradient, is used to colorize the points by using the following normalization formulas:

$$red = \frac{T(p_N, w_N^c)}{max\big(T(p, w^c)\big)} \qquad green = 1 - \frac{T(p_N, w_N^c)}{max\big(T(p, w^c)\big)} \qquad : red, green \in \mathbb{R}[0,1] \quad (4.4)$$

Here $T(p_N, w_N^c)$ is the current score value for the current point and $max\big(T(p, w^c)\big)$ is the highest found score value of the entire score vector. These formulas are used to achieve a linear color value mapping for the two color channels *red* and *green*, with value ranges between zero and one. A red coloration denotes a low score and green shows high score values. Points with zero scores, which thus don't contribute to the networks decision making, are colored black. Currently there is the issue that huge outlier values dominate lower values, see Figure 4.2, because if $max\big(T(p, w^c)\big)$ is much bigger then $T(p_N, w_N^c)$ the majority of points will be of green color. To fix this a threshold is applied to all non zero values which serves as a ceiling for all scores. This results in a more appealing color gradient, as smaller values are also considered more important now, as shown on Figure 4.3 on the next page.

4.2 Application and verification

It is possible to apply our approach to any kind of point cloud neuronal classification network, as long as it allows for extraction of the feature vector in a similar way as Pointnet does. Any more complex systems which use point clouds for computer vision classification task may benefit from this, including robotics, autonomous driving or any other

[2]`http://www.open3d.org`

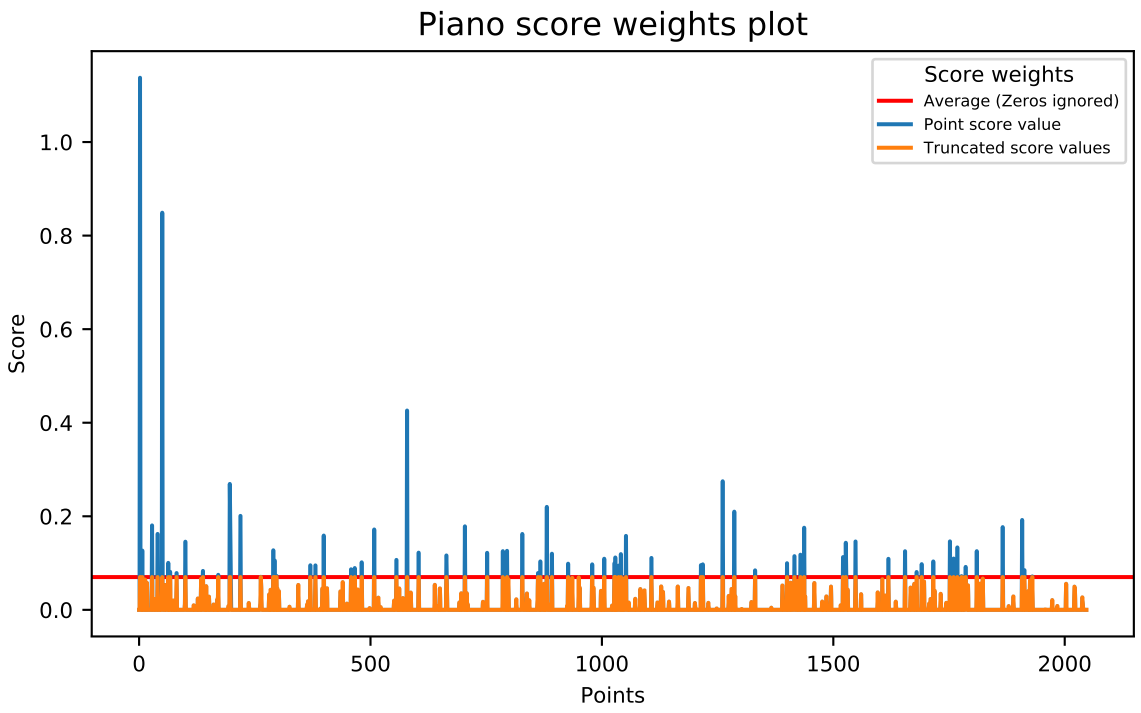


Figure 4.2: Plot of un-truncated (Blue) and truncated (Orange) gradient score values for Piano object with 2048 points. The average truncation threshold is shown in red.

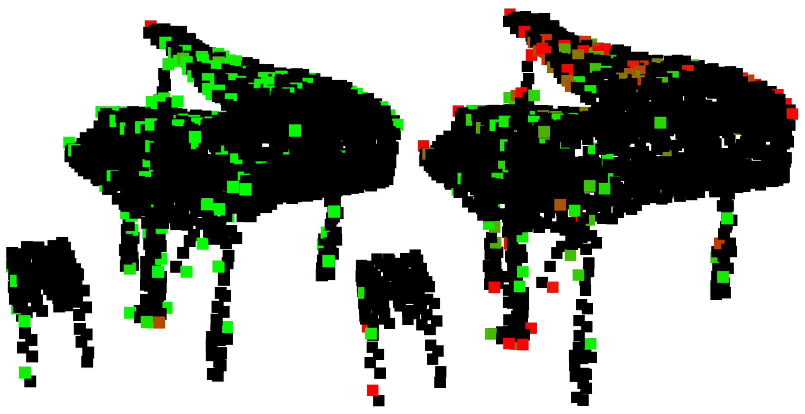


Figure 4.3: Comparison of gradient visualization, for Piano object, with (right picture) and without average value truncation (left picture). Black pixels denote zero scores and thus are not important for classification of the selected label at all. Importance of points range from green (less important) to red (very important) for decision making of the network.

kind of point cloud based neuronal network classification application. P-grad-CAM can be used in classification tasks to find those points in the point cloud, that are of highest importance from the perspective of the network, contributing the most towards the classification decision. Not only does this alone yield a deeper insight upon the rationale of the network itself, it also allows to create specialized point sets to allow a deeper investigation of our approach.

To verify the effectiveness of p-grad-CAM a two step test process is enacted. In the first step the network will be tested against rotations in random directions, using the original unprocessed point cloud. This is necessary to establish a ground truth, which allows for comparison with the test results. In the second step a new test set of points is built, using the salient points that were computed by the p-grad-CAM algorithm. In the first test set, all zero-score points are removed. In the second test set all values beyond a certain threshold are removed. These sets will then be inverted and only the zero-score points are kept etc. The resulting test sets are then each fed back into the network to monitor the change of the networks accuracy, loss and the classification prediction. All sets are then compared to the accuracy, loss and prediction of test sets, where equally many randomly chosen points were removed. The results will be plotted to display the networks performance for each point set variation.

This approach allows to get a direct feedback on the correctness of our approach. The premise is that the network accuracy is reliant on the existence of these important points for a correct prediction. So by removing the important points, a drop in accuracy should occur. The opposite should be true for removing the zero score points, the prediction accuracy should not change at all. If these premises can hold against a test set of randomly removed points, then it was successfully proven, that the important points for the decision making of the network have indeed been found.

4.3 Assembling the p-grad-CAM algorithm

With the base idea of the approach outlined, the draft of the p-grad-CAM algorithm can now be set up. The base idea is that every point with a weight greater then zero and over a certain threshold is removed from the point cloud. The residual point cloud object is then successively fed back into the network until the prediction deviates from the ground truth. This yields the draft for the final algorithm as shown on Listing 1 on the following page. This final algorithm can then be applied to find the salient regions of the input point cloud with respect to an user chosen class label. It is a self evaluating fire-and-forget algorithm that requires no further intervention of the user during runtime. The chosen points are colored after their importance ranking as described in the previous Sections. The result of the algorithm, using the maxpooled greater then zero threshold for the `thresholdMode` parameter, is shown on Figure 4.4 on the next page.

Algorithm 1: Point cloud gradient Class Activation Mapping (p-grad-CAM)

```
Data: pointclouds_pl, labels_pl, sess, poolingMode, thresholdMode,
      numDeletePoints
Result: Heatmap visualization of important/unimportant points.
1  initialization;
2  while ground truth == prediction do
3      pointWeights ← computeHeatGradient(pointclouds_pl, labels_pl);
4      pointclouds_pl, importantPoints ← delete_above_threshold( pointWeights,
         pointclouds_pl, thresholdMode );
5      weightArray.extend( importantPoints );
6      for i to range(BATCH_SIZE) do // evaluate for prediction change
7          rotPC ← rotate_point_cloud_XYZ(pointclouds_pl);
8          prediction, loss ← sess.run(rotPC, labels_pl);
9          prediction ← np.argmax( prediction, 1 );
10         correct ← np.sum( prediction == labels_pl );
11         total_correct += correct;
12         total_seen += 1;
13         loss_sum += loss * BATCH_SIZE;
14         accuracy = total_correct / float( total_seen );
15     end
16 end
17 draw_residualPointcloud(pointclouds_pl);
18 draw_pointcloudHeatmap(pointclouds_pl, weightArray);
```

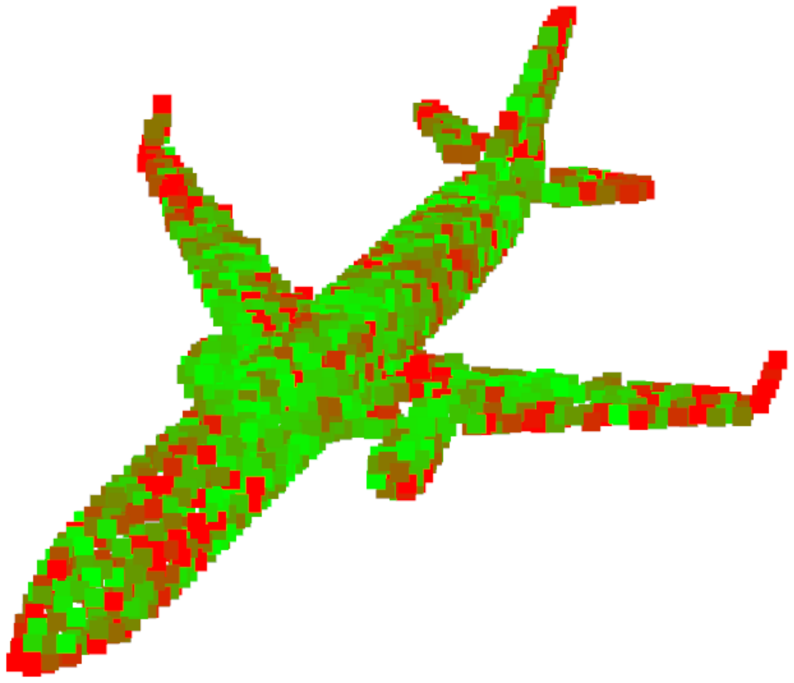


Figure 4.4: Heat map plot of an airplane after 7 iterations removing every value greater then zero. A total of 1685 out of 2048 points has been removed before the accuracy dropped to zero and the prediction changed. These 1685 important points were collected over each iteration, with about ∼200-300 points removed each iteration and finally drawn to screen.

5 Point cloud gradient Class Activation Mapping (p-grad-CAM)

In this section a crude first prototype of the p-grad-CAM algorithm is applied to Pointnet using the `modelnet40_ply_hdf5_2048` database. It is being used to find and visualize the salient regions for a particular class label. The visualization is achieved by coloring each point in the point cloud using the score values $T(p, w^c)$, calculated from the extracted feature vector from the deep network, which are then transformed into usable color values. It is also necessary for the user to specify the desired class label before running the algorithm, as it currently defaults to label 1 (airplane). The class label is an integer value that represents a verbose label string from the `shape_names.txt` file, which is included in the `modelnet40_ply_hdf5_2048` database. The number corresponds to the line in the `shape_names.txt` file where the particular label of desire is located at. For example if we were interested in the label "person", which is located at line 25, one would enter the number 25 for the variable `testLabel`. The gradients will now be computed with respect to that particular class label. The results of the prototype algorithm can be seen on Figure 5.1 on the following page.
The first thing that strikes the eye, when looking closer to the result, is the large amount of black points. These black points indicate score weights with value zero and thus had no relevance for the network classification decision. This implies, that the network would have made the same classification decision even if those points did not exist. The other interesting result is that the red important points are rather scarce, compared to the total amount of points. When looking at the plot on Figure 5.2, which shows of the score values for each point, this becomes even more visible. From a total amount of 2048 points there were 61 points that were greater then the red average threshold and about 8 values, which were significantly bigger in value than all others.

The same phenomenon of few very high outlying values, that dominate the other values by orders of magnitude, was observed in other cases too. When looking at the distribution of important red pixels it appears that they accumulate on the wingtips and the front region of the airplane. It is amazing how they appear to naturally gather in regions that would be considered as a good feature in terms of computer vision. Those are regions that would be also subject of interest for a 2D feature point extraction algorithm using CNN. Which is peculiar since no CNN or 2D feature extraction algorithm was applied here, as everything is based solely on (x, y, z) coordinate points in 3D spatial space. Considering all these observations, this raises two interesting questions that will be investigated in more detail at the next Section:

1. Does the removal of the black zero score points impact the accuracy and prediction of the network negatively or will it have no impact at all?
2. Does the removal of the high scoring points indeed affect the accuracy of the decision in a negative fashion or not at all?

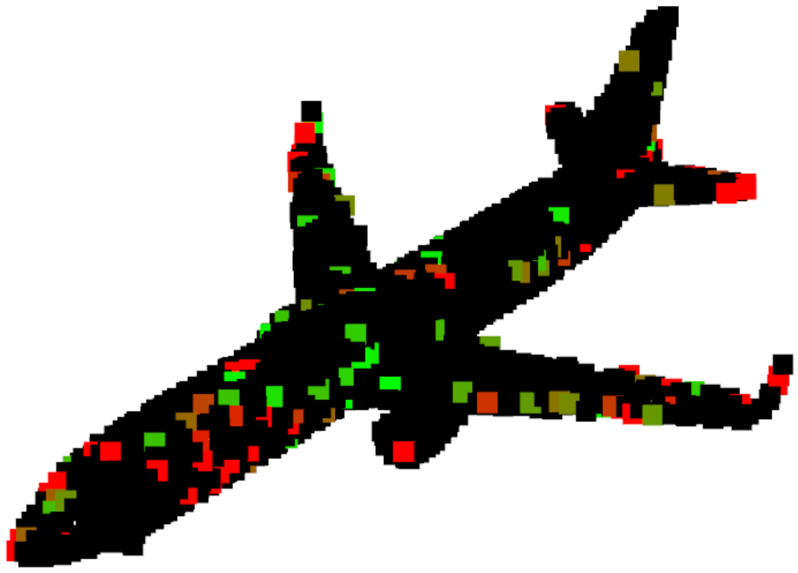


Figure 5.1: Point cloud gradient Class Activation Mapping algorithm prototype applied on **modelnet40_ply_hdf5_2048** using the class label 1 (airplane).

5.1 Using p-grad-CAM to build test sets

With the help of the test sets that were described in the previous section, it is now possible to perform an in-depth testing of p-grad-CAM. These test sets can be selectively created to test for certain premises of behavior that the algorithm should show. Since the importance of points is now known we can specifically remove points that are important or unimportant for the network classification decision making. Those modified point clouds are then successively fed back into the same network for as long as the prediction equals the ground truth. To test the validity of p-grad-CAM, special test sets have been constructed, to answer the prominent questions on the previous page. The first test set will have every point removed, that has scores greater then zero. The results, using the point score heat map as illustrated on Figure 5.1, are shown in Figure 5.3 on the facing page.
The difference is very subtle but if compared to Figure 5.1 all colored points were indeed removed. This is especially visible on the wingtips and the white holes in the front of the plane where many non zero score points were located.

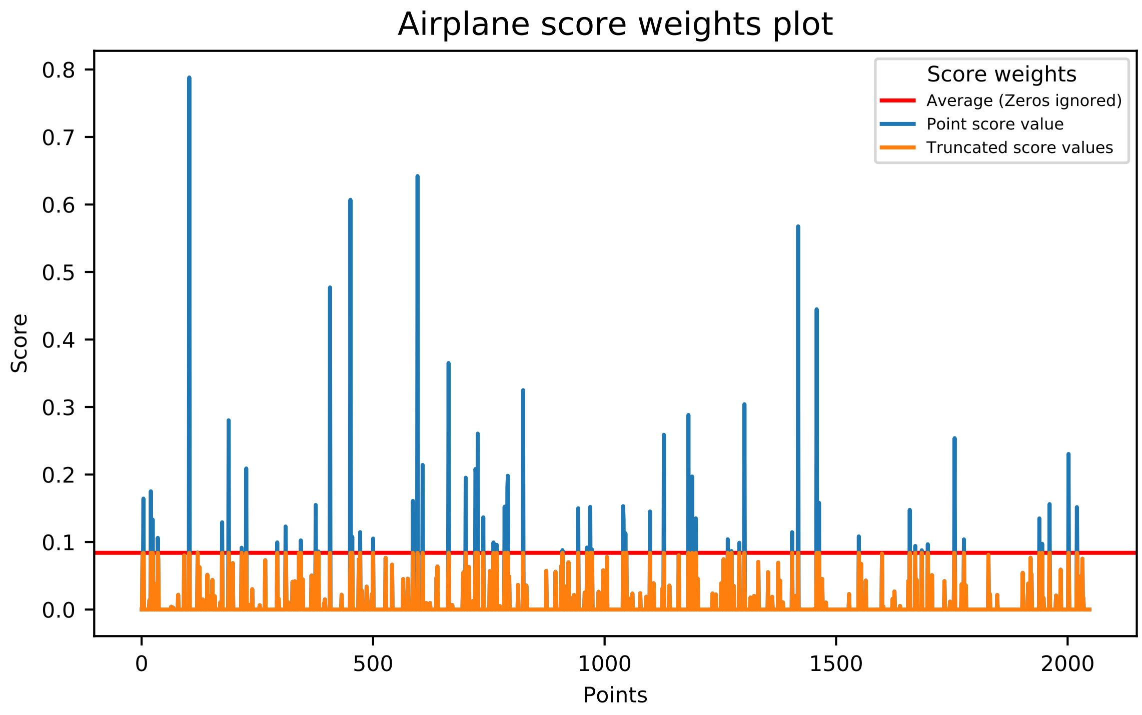


Figure 5.2: Score values of the p-grad-CAM algorithm prototype for each of the 2048 points using the class label 1 (aircraft).

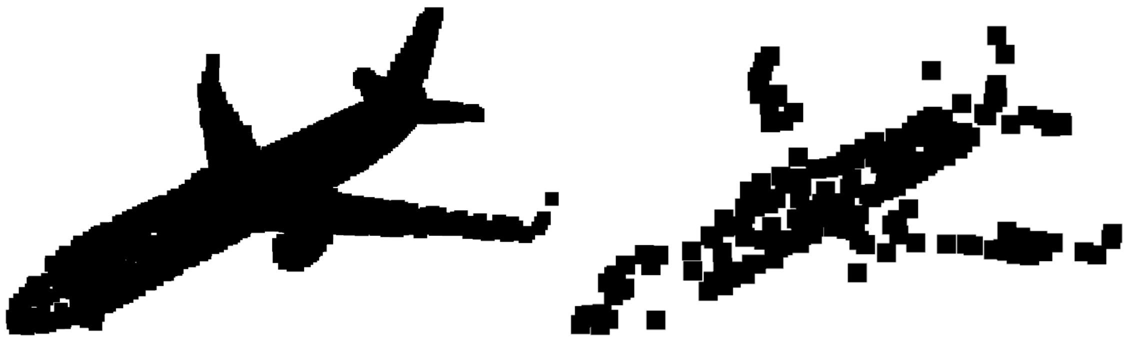


Figure 5.3: Comparison between two airplane point clouds, where all scores greater zero were removed and only scores equal zero stay (left image) and one where all scores equal zero were removed with only scores greater than zero that stay (right image).

For the second test set, all points with score values equal to zero were removed. Only points with scores greater then zero were kept. This yields the most drastic difference when compared to Figure 5.1. However despite the majority of points being removed, it is remarkable how the reduced point cloud visually still resembles an airplane. Another important question that now needs to be answered is which kind of threshold should be used to select important points and which kind of pooling method should p-grad-CAM use. Because there exist many possible candidates, it requires a design choice study to figure out, which threshold or pooling method yields the qualitative best result. This design choice study will be conducted next.

5.2 Design choice study preparation

In the following sections, the design choices of both our proposed p-grad-CAM algorithm and the point coloration approach, will be discussed. In particular the p-grad-CAM pooling method, important for the score calculation and the visualization threshold method, used to filter out unimportant points, are subject for a deeper inspection. Each of them will be compared to similar commonly used methods and measured for potential increase of visual fidelity and accuracy. To gain a meaningful metric to determine which pooling or threshold method fits the best, different sets of point cloud objects are generated using each proposed method candidate and run the p-grad-CAM with them to measure their impact on the network accuracy. The threshold and pooling method which yields the highest accuracy is then chosen for the final implementation of the p-grad-CAM algorithm. First the possible pooling candidates are subject for investigation, followed by a deeper look at which threshold method fits the best.

5.2.1 p-grad-CAM pooling method candidates

Pooling operations play an important role in deep networks. They help reducing the size of feature maps or respectively feature vectors. This allows for training and testing with a reduced strain on computation power because it reduces the complexity of the network. It helps by reducing the receptive field for each neuron, which allows them to cover a larger portion of the input. This improves the overall perceptive ability of the network. Another helpful property is the reduction of the negative influence on results by noise and distortion. Pooling naturally flattens values into a more representative state and remove outliers and noise. Essentially pooling improves the networks perceptive power while reducing the strain on computational resources. It is an important part of any modern day deep network but is also very situational. This means that the commonly used pooling methods are not the "one size fits all" kind and thus require testing and evaluation to figure out which method fits best for the each individual network.
The pooling methods, which will be taken into consideration for testing, are average, max and stride pooling. There actually exist much more methods like universal pooling

[Lee et al., 2016], Mixed average-max pooling, gated pooling, tree pooling [Hyun et al., 2019] etc. However they are not considered for the design choice study because most of them require to be trained alongside the network, which defeats the purpose of our approach working on an existing network, without any retraining necessary. The investigated pooling methods are stride, average and max pooling since they are either easy to implement or even have their own inbuilt functions in Tensorflow. The first pooling variant that will be looked at is max pooling, followed by the not so popular stride pooling and finally average pooling. Stride pooling, despite it being uncommon, was used here because it does not require any retraining or change of the network architecture to work and was considerably easy to implement.

Max Pooling

In this approach all values from each row of the gradient tensor are fed into the max operator, to reduce the tensor from shape [N,1024] to [N]:

$$w_c = \overbrace{\max_N(\underbrace{\frac{\partial y_c}{\partial A_i}}_{\text{Gradients}})}^{\text{Max Pooling}} \quad : \quad i \in [1, 1024], N \in [1, \text{num\_points}] \tag{5.1}$$

This is achieved in Tensorflow via the `reduce_max()` function. The max pool operation is more in line with the Pointnet architecture, since it also uses a max pooling operation to reduce the feature vector for further processing (see Figure 4.1 on page 21). The resulting score vector, showing the scores for each point of the test object, is shown on Figure 5.6 on page 34. It is interesting to see that from the three suggested approaches, max pooling yields overall higher score values. The next best competitor, average pooling, yields similar visual results, the score values however are weaker in amplitude.

Stride Pooling

This method choses always the number from index *i* in the score tensor:

$$w_c = \overbrace{f_{sp}(\underbrace{\frac{\partial y_c}{\partial A_i}}_{\text{Gradients}})}^{\text{Stride Pooling}} : i \in [1, 1024] \tag{5.2}$$

This is the most simple and computationally least expensive operation since it is little more then a simple read operation. This approach has no dedicated Tensorflow function hence it had to be implemented manually as `tensorflow.squeeze(tensorflow.map_fn(lambda x: x[i:i+1], maxgradients))` with $i \in [0, num\_points]$. The `x[i:i+1]` statement however requires the manual adjustment for the stride pooling index *i* for each run of program. Unfortunately it requires a lot of fine tuning to find some *i* that actually yields any valuable information. Figure 5.4 below shows the score tensor, after some trial and error for the stride index, where exactly one point got any score information at all and all other points scores were zero. The reason for this is that the dynamic distribution of the score values is simply ignored since always the same score is taken into consideration. Therefor potentially valuable information will not be taken into consideration to compute the scores for all points.

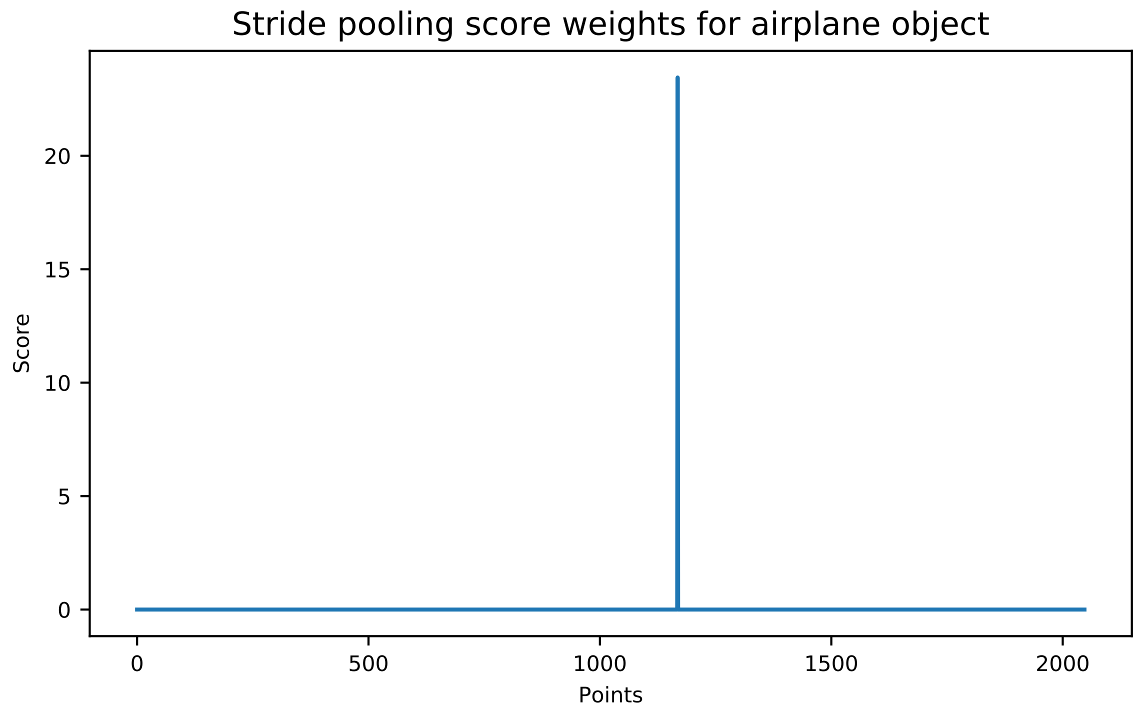


Figure 5.4: Score value plot of a stride pooled p-grad-CAM.

In conclusion the stride pooling approach is not suited for the p-grad-CAM algorithm. It simply does not fit the requirement for the algorithm to require as few human intervention and manual value evaluation as possible, yet yielding good results.

Average Pooling

This approach calculates the arithmetic mean over all 1024 values per neuron for each point in the point cloud. The score vector function then looks as follows:

$$w_c = \overbrace{\frac{1}{1024}\sum_{i}^{1024}}^{\text{Average Pooling}} \underbrace{\frac{\partial y_c}{\partial A_i}}_{\text{gradients}} \tag{5.3}$$

This pooling operation is also used in the original grad-CAM [R. Selvaraju et al., 2017] approach in order to reduce the feature vector down to single score value per neuron. It can be applied in Tensorflow via the `reduce_mean()` function. The result score value plot, using average pooling, for each point can be seen on Figure 5.5. When compared to the max pooled variant it apparently yields values with lesser amplitude. Also some points that were rather non important in the max pooled variant are suddenly very important. This is particularly peculiar and requires further investigation if either average or max pooling yield the best results with respect to the Pointnet architecture, which will be conducted in later chapters.

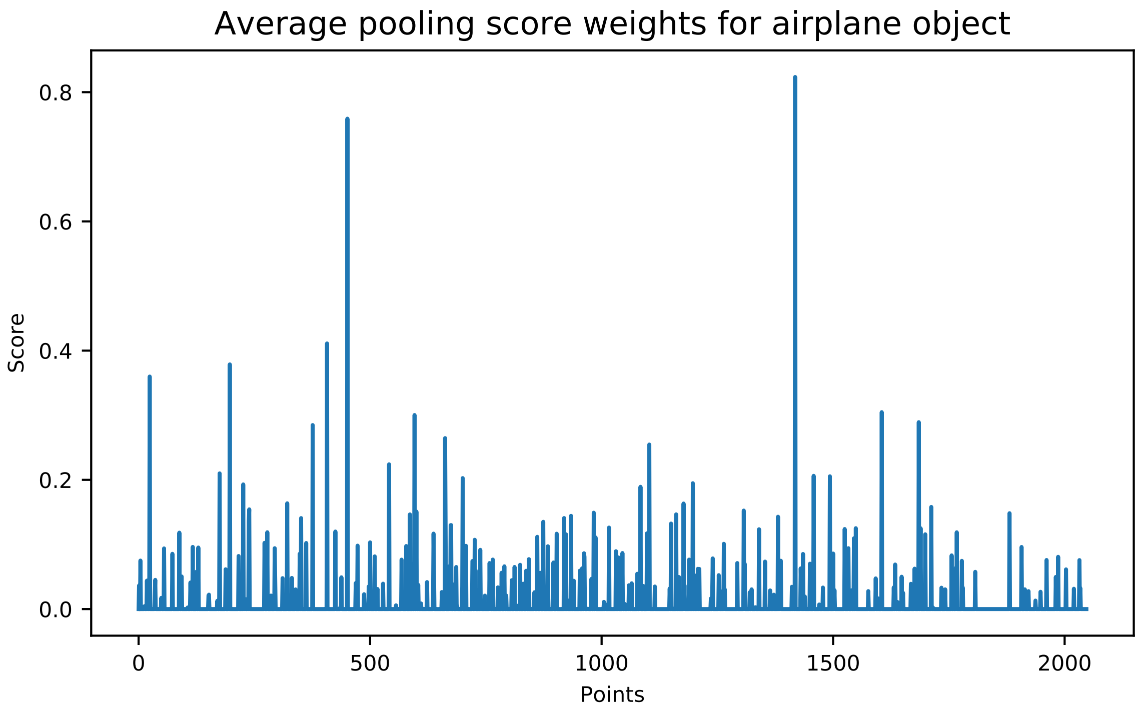


Figure 5.5: Score value plot of the **average pooled** p-grad-CAM for the Airplane object (2048 points).

5.2.2 Visualization thresholding method candidates

To visualize the network scores, each point in the point cloud is colored in such a fashion, that it is both appealing to the eyes and yields relevant information. Unfortunately having score values that are greater then the other score values by one or two orders of magnitude is no rarity. Since the coloration of the points uses the highest found score value, those huge outlier values drastically skew the coloration result. Instead of continuous gradient, only the few points of high score values are colored red (quite important), while the majority of lower value points are colored green (quite unimportant). To remedy this, a threshold is used to clip huge score values to even out the coloration, this results in a better distribution range of colors, see Figure 4.3 on page 24. There are a lot of possible methods to calculate such a threshold value to yield a better coloration of the heat map. The most common ones in the following will be subject of deeper investigation. Each thresholding method will be tested with both the average and max pooled variant of our p-grad-CAM algorithm in order to determine which approach yields the

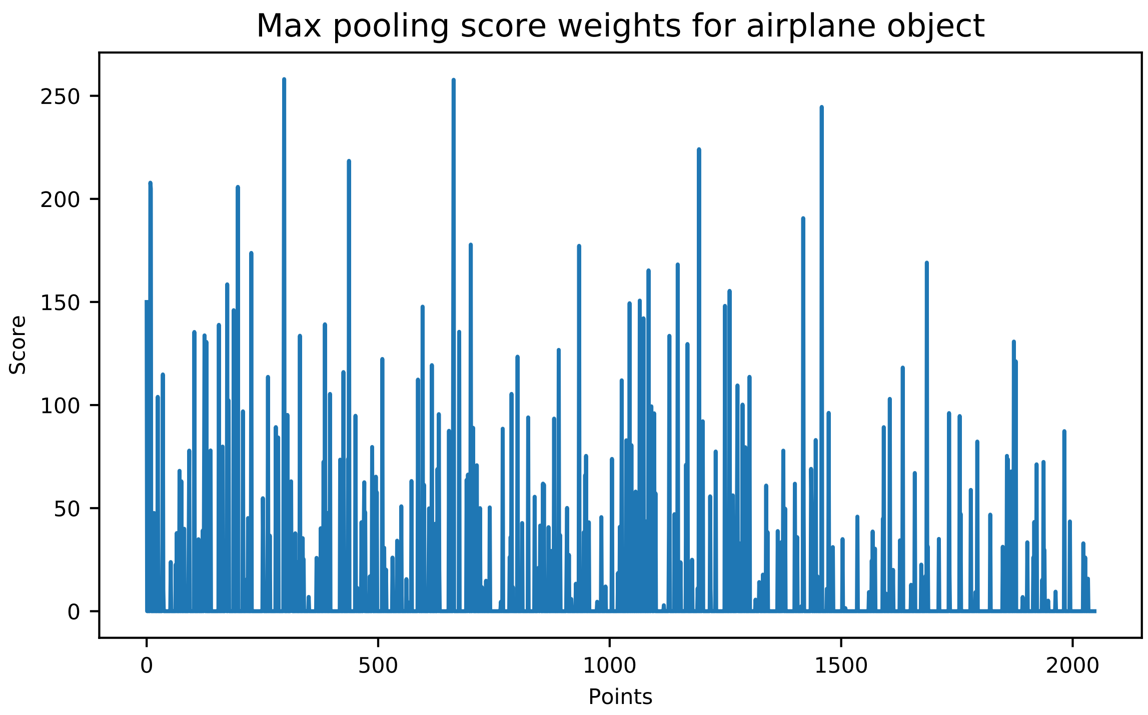


Figure 5.6: Score value plot of the **max pooled** p-grad-CAM for the Airplane object (2048 points).

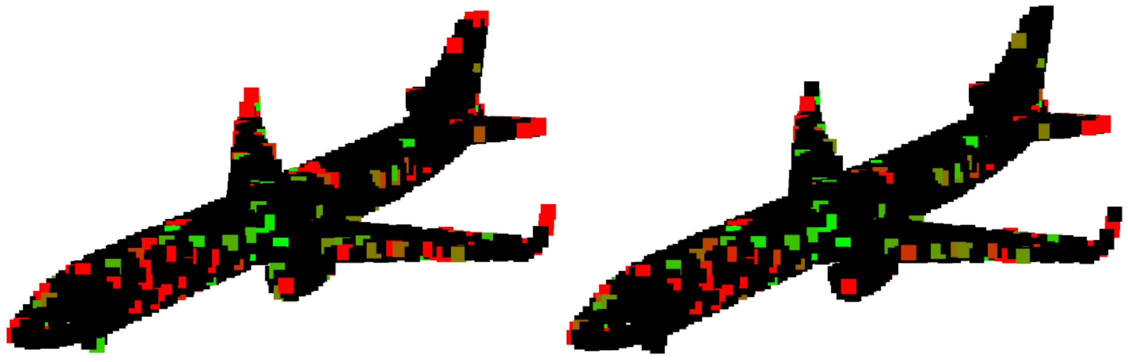


Figure 5.7: Comparing the visual result of the heat map for the max pooled (left image) and average pooled (right image) p-grad-CAM.

best visual result. In Chapter 6 on page 39 the findings from this chapter will be used to generate test sets to determine if these results can be backed by empirical data.

Arithmetic mean

The arithmetic mean is calculated by summing up the score of each point and divide it through the total amount of points. The result is a floating point number which determines the ceiling for the highest allowed score value. This threshold is only used for point coloration to remove the huge impact of outlier values upon it.

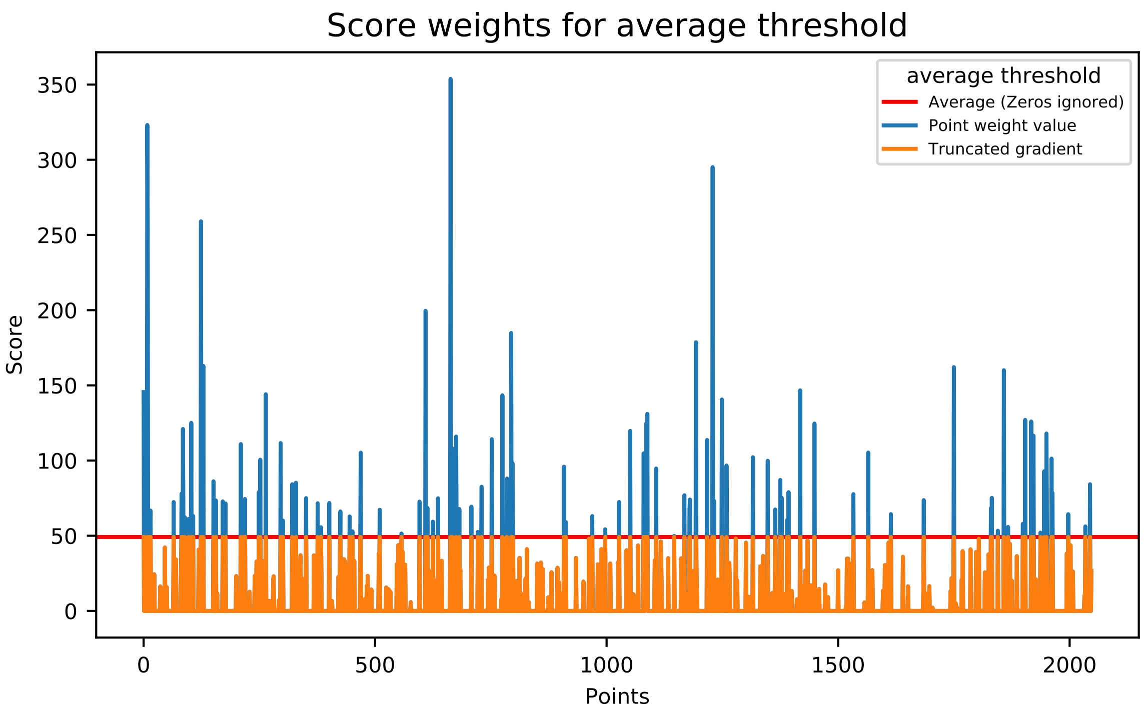


Figure 5.8: Airplane score value plot of **arithmetic mean** averaged p-grad-CAM.

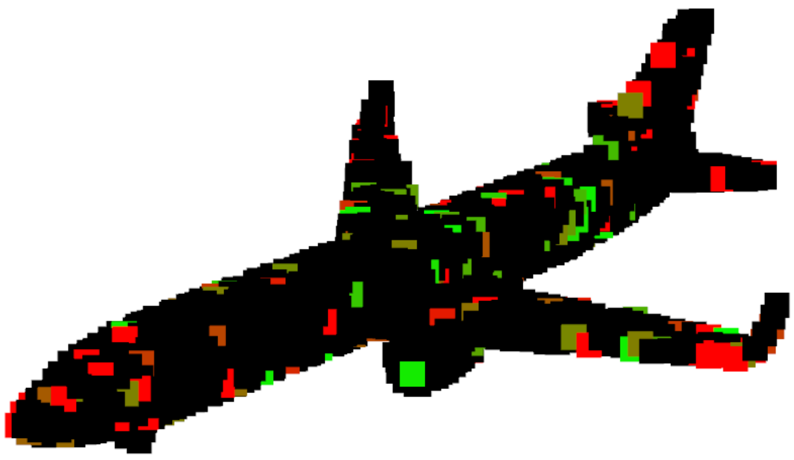


Figure 5.9: Airplane heat map for **arithmetic mean** thresholded score matrix.

Median value

The median approach chooses the value, situated in the middle of all score values after they have been sorted. The threshold is again used for the coloration of the points to remove the huge impact of outlier values upon the point colorization. The results on the impact of this method are shown below.

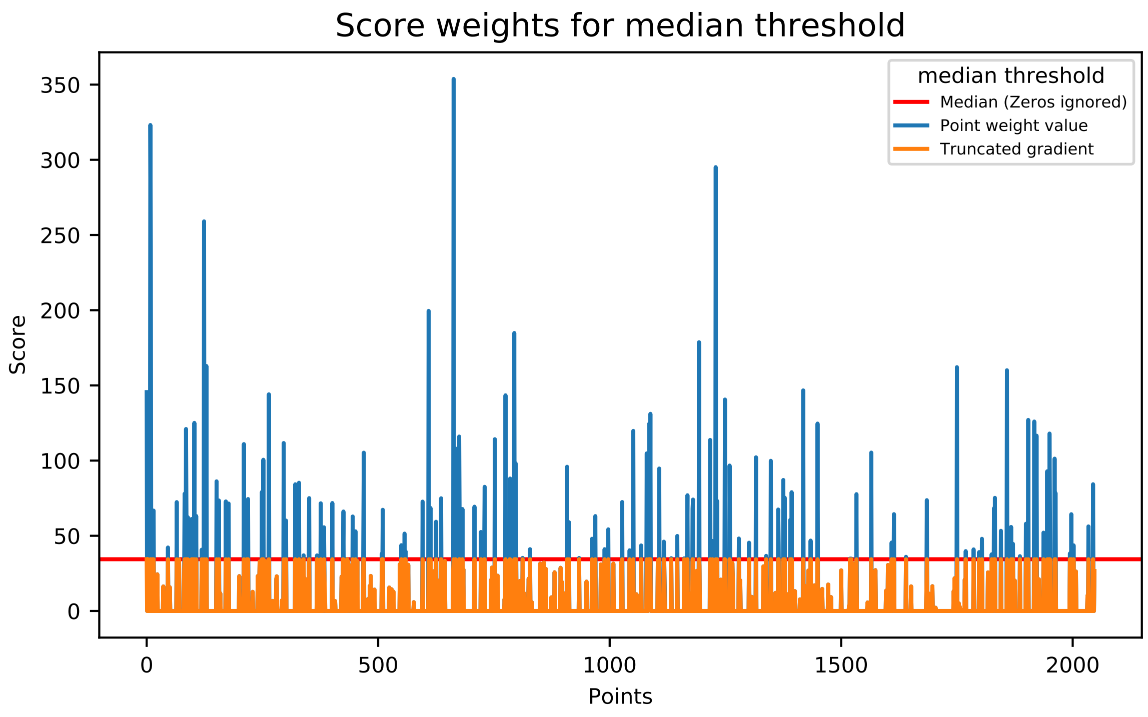


Figure 5.10: Airplane score value plot of **median** averaged p-grad-CAM.

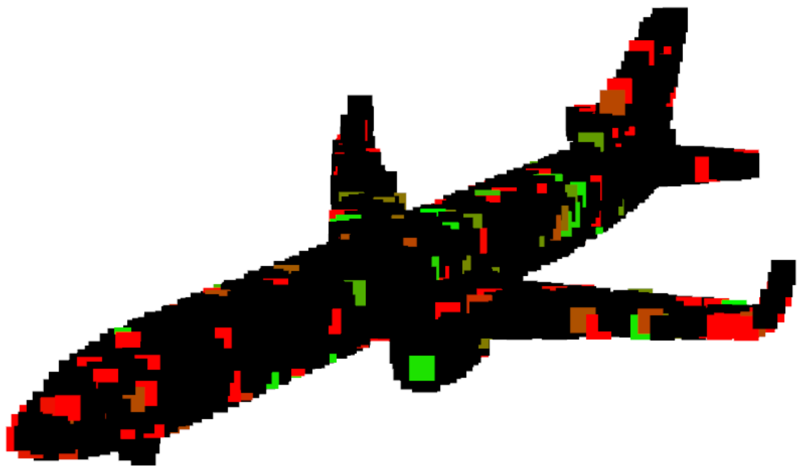


Figure 5.11: Airplane heat map for **median** thresholded score matrix.

Mid-range value

For the mid range method the single highest and the lowest score is summed up and divided by two. From all methods, mid range yields the highest threshold which favors few values with high scores over the majority of low scores. This threshold works the best when intending to select the most important points for removal but not so well for coloring the points. The reason is that high score value points once again dominate the lower scores. This phenomenon can also be observed on Figure 5.13.

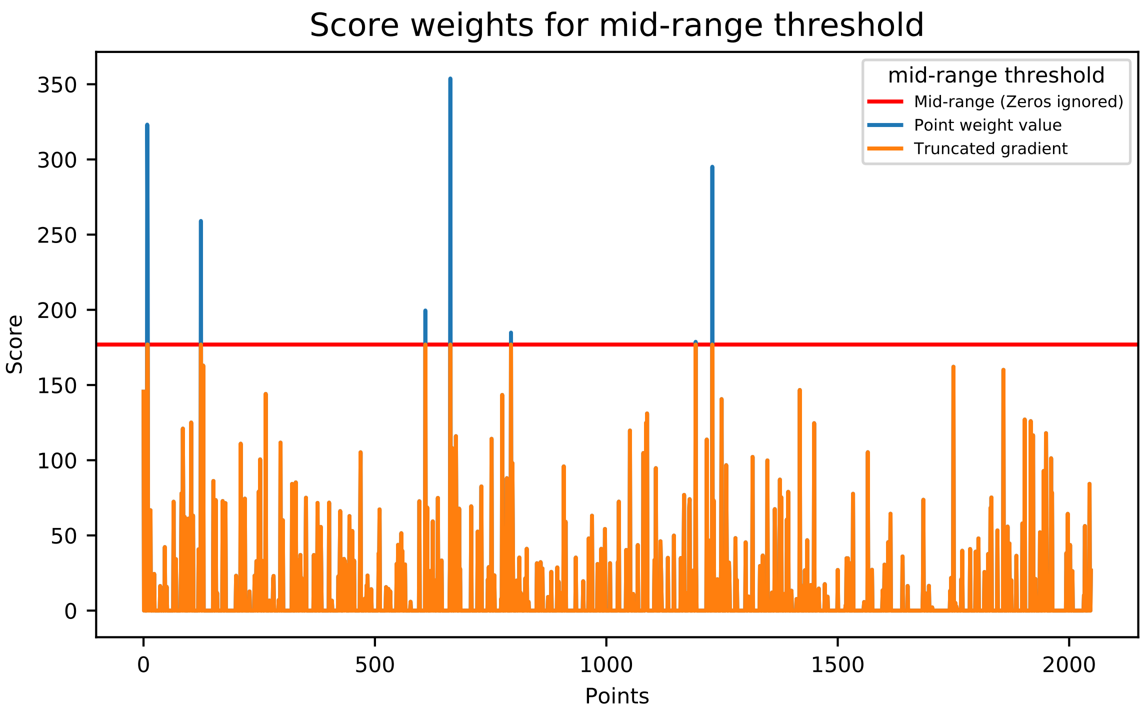


Figure 5.12: Airplane score value plot of **mid-range** averaged p-grad-CAM.

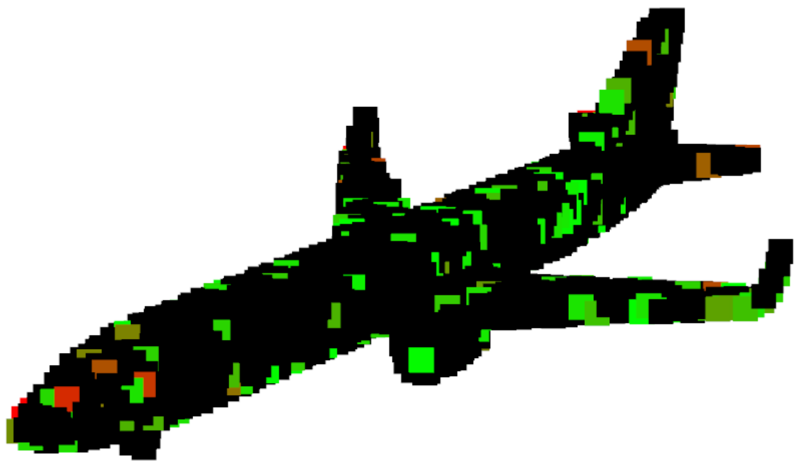


Figure 5.13: Airplane heat map for **mid-range** thresholded score matrix.

6 Evaluation

In this chapter the knowledge from the previous chapter is used to specifically test the correctness and performance of the Point cloud gradient Class Activation Mapping (p-grad-CAM) algorithm. There will be two major tests that the network will undergo. The first test uses the shapes from the `modelnet40_ply_hdf5_2048` data set and randomly rotates them around the XYZ axis. We specifically came up with this test after it turned out that the original Pointnet network code only rotated the shapes around the Y-axis for training data augmentation. The test with XYZ-axis rotated data, which the network was originally not trained for, is important to see if any changes to the prediction accuracy occur. In that case it would be required that the network is retrained with XYZ rotated input data.

The second test is directly derived from the desire to verify the correctness of the p-grad-CAM approach. To clarify if our approach is indeed able to find the relevant areas of interest, a specific test set of point cloud data is built with varying amounts of points, that will be dropped and then evaluated for their impact on the networks prediction accuracy. These test sets are created in such a way, that only those points are removed, that were identified by our approach as important or unimportant for the decision making of the network. The premise of the test is, that with the removal of important points a drop in the prediction accuracy and ultimately a wrong prediction would be expected. This is a logical conclusion since the points that had previously driven the network towards a specific decision are now gone. The inversion of this premise is also of interest to look at. It would would be expected that no drop in prediction accuracy should occur at all, if only points that were completely unimportant for the networks classification decision are removed. Both tests will be conducted in the following sections.

6.1 Testing against XYZ rotation

A very peculiar find surfaced during a deeper look over the original Pointnet code, which performs data augmentation via rotating a point cloud in a random direction. It turned out, that it rotates point cloud objects only around the Y-Axis. This became evident when looking closely over `line 16` in Listing 7.1 on page 73 and `line 15` in Listing 7.2 on page 73, where there is only one Y-axis rotation matrix present in both functions. This appears to be an oversight at best or a mistake at worst. This potential issue can be easily rectified by adding the missing rotation matrices to make it capable of random rotation around the XYZ-Axis. This however means that the network needs to be retrained in order for the improved rotation data augmentation code to be captured by the network. To confirm that this is indeed a necessity, the original unchanged Pointnet is tested, which was trained only for Y-Axis rotation. This should reliably verify if XYZ-Axis rotation does indeed yield worse results which makes the Y-Axis only rotation indeed be an issue that requires fixing. For this purpose, during the networks evaluation stage, a batch of XYZ-Rotated point cloud objects will be passed to the network to see if there occured any significant change of accuracy, loss or the prediction. The results can be observed in Table 6.1 below.

Original network handling different rotation methods		
Rotation method	Average accuracy	Mean loss
Y-Axis Rotation	1.0	$2.1928470232524 \cdot 10^{-5}$
XYZ-Axis Rotation	0.252	5.45134687423706

Table 6.1: Change of the average accuracy and mean loss of XYZ-Axis and Y-Axis rotated airplane point cloud object consecutively averaged over 1000 iterations. Using the original unchanged network.

The accuracy for an XYZ rotated airplane drops down to $\sim$ 25% from 100% with the network classifying it wrongly as stairs. This phenomenon occurs using the original unchanged Pointnet network, trained with the Y-Axis rotation code. Therefor a fix is indeed necessary, see Listing 7.3 on page 74 for details, so that XYZ rotated data could next be used to generate new training data to retrain Pointnet against XYZ-Rotation, even though it was originally only trained for Y-Axis rotations. This should increase the robustness for any potential real world data, which is rarely rotated solely around the Y-Axis. The results of the newly retrained network for XYZ-Axis rotation are shown on Table 6.2 below.

Retrained fixed network handling different rotation methods		
Rotation method	Average accuracy	Mean loss
Y-Axis Rotation	1.0	0.000107754705823027
XYZ-Axis Rotation	1.0	0.000229689321713522

Table 6.2: Change of the average accuracy and mean loss of XYZ-Axis and Y-Axis rotated airplane point cloud object, consecutively averaged over 1000 iterations. Using the new fixed and retrained network.

The change in both the accuracy and the loss of the improved network are dramatic. Not only did the original accuracy for the Y-Axis rotation stay the same but the accuracy for XYZ-Axis rotation achieves the same result now as well. Their loss functions however are not absolutely identical, which is shown on Table 6.2. Even though both losses are sufficiently low now, the loss for the XYZ rotated point cloud is still slightly greater compared to point clouds that were rotated only around the Y-Axis. Apparently the reduction of two degrees of freedom on only Y-Axis rotated objects causes a slightly lower loss. The gained results so far are however good enough to be used for the actual ablation study, which will be conducted in the next section.

6.2 Ablation study

This test idea is directly built upon the information, provided by our p-grad-CAM algorithm and the resulting weights for each point of the point cloud object. Since the points of importance are now known, it is possible to precisely manipulate the point cloud object to verify the premise of our approach; finding the points that the network finds important for its classification decision. To do this, various test sets are created where points have been deliberately dropped based upon the values of their respective score weights $T(p, w^c)$. These test sets will be built to test both average and max pooling to procure the weights for each point. These points will then be selected for dropping, by applying a arithmetic mean (average), median, midrange threshold. On top of that there are two extra tests, which remove all points with scores greater than zero or just drop random points. This is done to filter more important points over less important ones. The resulting point clouds are then fed back into the network and evaluated for their accuracy, loss and prediction changes. If the points of importance were indeed found, there should be a measurable drop in accuracy, raise of the loss and change of the prediction from correct to incorrect. The following sections will look in particular at the verification of this claim.

6.2.1 Removing important points

In this test only important points, i.e. having a score weight $T(p, w^c)$ greater than zero and greater then a certain threshold, are removed. Internal tests have shown, that not all important points are found in a single run of our point cloud grad-CAM approach. This means that there exist more points of importance in the point cloud, yet only a small subset of critical points drive the networks decision, unless all of these critical points are removed. A similar observation has been reported by [Zheng et al., 2018]. To account for this, the p-grad-CAM algorithm will be applied repeatedly over consecutively shrinking point sets, where in each iteration all found points of importance are removed and the result be used for the next iteration, until enough points were removed to make the prediction deviate from the ground truth. The arithmetic mean, median and max range threshold methods, introduced in Section 5.2.2 on page 33, are used for choosing the points to be dropped. For each point removed by our algorithm there are equally many randomly chosen points removed, to investigate for their impact on the accuracy, loss and prediction of the network to quantify their impact. This should also yield the answer to the question over which threshold method works the best and if our algorithm actually performs better then randomly removing points.

To achieve this, every possible combination of pooling and threshold method needs to be investigated. There are two pooling methods being average and maxpooling, five point selection threshold methods with random, greater-then-zero, arithmetic mean, median and mid range plus two tests for important and unimportant points to be removed, that need to be take into consideration. This totals to 20 test sets per point cloud shape. With the total amount of 40 available shapes this would require 800 test runs to account for all possible shapes. For the sake of simplicity these tests were only conducted using the airplane, bed and bench shapes to have a first impression of possible emerging trends. Each of the tests will be ran on our point cloud grad-CAM approach using a test set of 500 instances of the same point cloud object, randomly rotated around the XYZ axis and the results averaged into a single result. Since the important points are removed in the test set, the best method would be expeacted to be the first one to drop significantly in accuracy, while simultaneously reaching high loss values, compared to the other methods. The results of our efforts are shown on Figure 6.1 and Figure 6.2 on the next page.

The accuracy plot, shown in Figure 6.1, confirms the interesting observation mentioned earlier, that deleting the most important points does not immediately impact accuracy. If they did, there should have been a visible drop in accuracy happening from the first iteration on but that isn't the case. In conclusion this means that there exists a subset of important points that supersede each other and once the points with the highest importance weight are removed, points with a prior low weight suddenly become important and still provide a solid prediction accuracy. However after about 3-5 iterations the first drops in accuracy can be observed, with the greater-than-zero threshold being the first one to drop. The second best thresholding method appears to be the maxpooled median which yields over all better results then average pooling.

This makes the greater-than-zero threshold currently the best one out of all the approaches in total, since we were looking for a method that would be the first to drop in accuracy. This observation goes in line with the loss plot in Figure 6.2 where the non zero threshold method is the first one with a visible increase of loss as well. These are desirable results which conform with our initial premise of a low accuracy and high loss, which means that important points were indeed found and removed. Another peculiar observation is the requirement of a considerable amount of iterations before visible drops in accuracy and loss have occurred. A significant drop of these metrics was expected to happen much earlier. However with the greater-than-zero threshold method taking the win in this experiment, it would be interesting to see if it still performs just as good with the inversion of this test premise, removing unimportant points instead of the important points, which will be investigated in Section 6.2.2 on page 46.

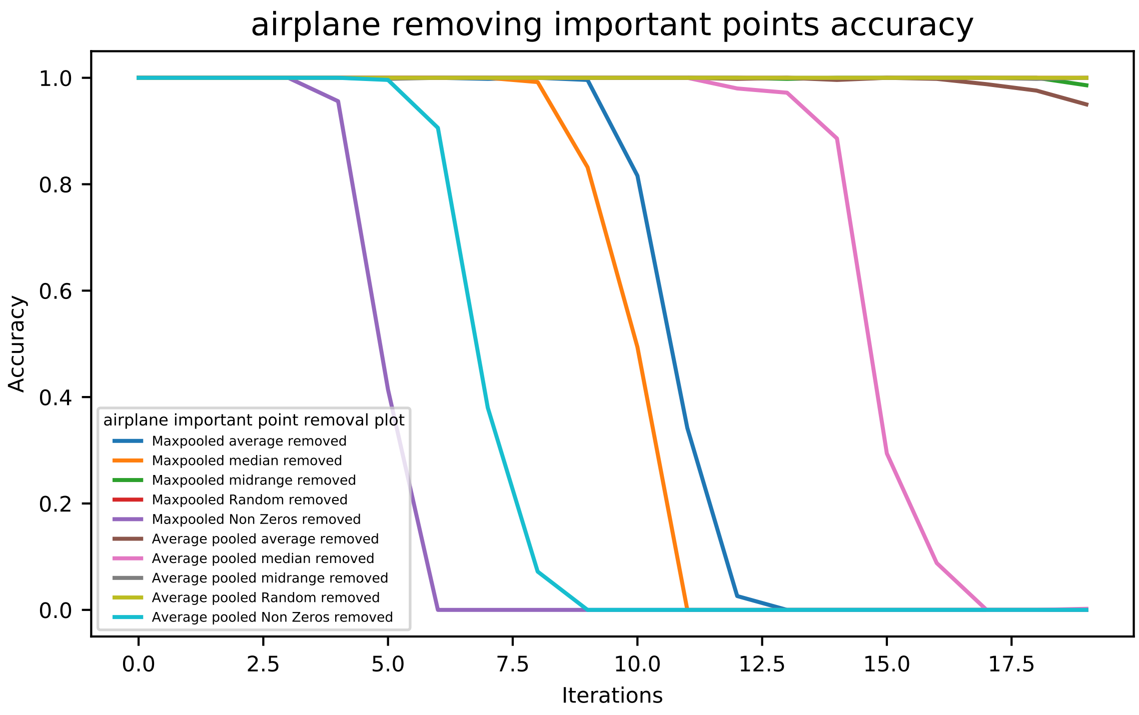


Figure 6.1: Airplane accuracy plot for the removal of important points for each pooling variant (maxpooled vs average-pooled) and each threshold method (random, greater-than-zero, arithmetic mean (average), median and mid range). Each iteration is the average value calculated over 500 batches of the same point cloud object. Once eligible points were removed the respective result point cloud is fed back into the next iteration.

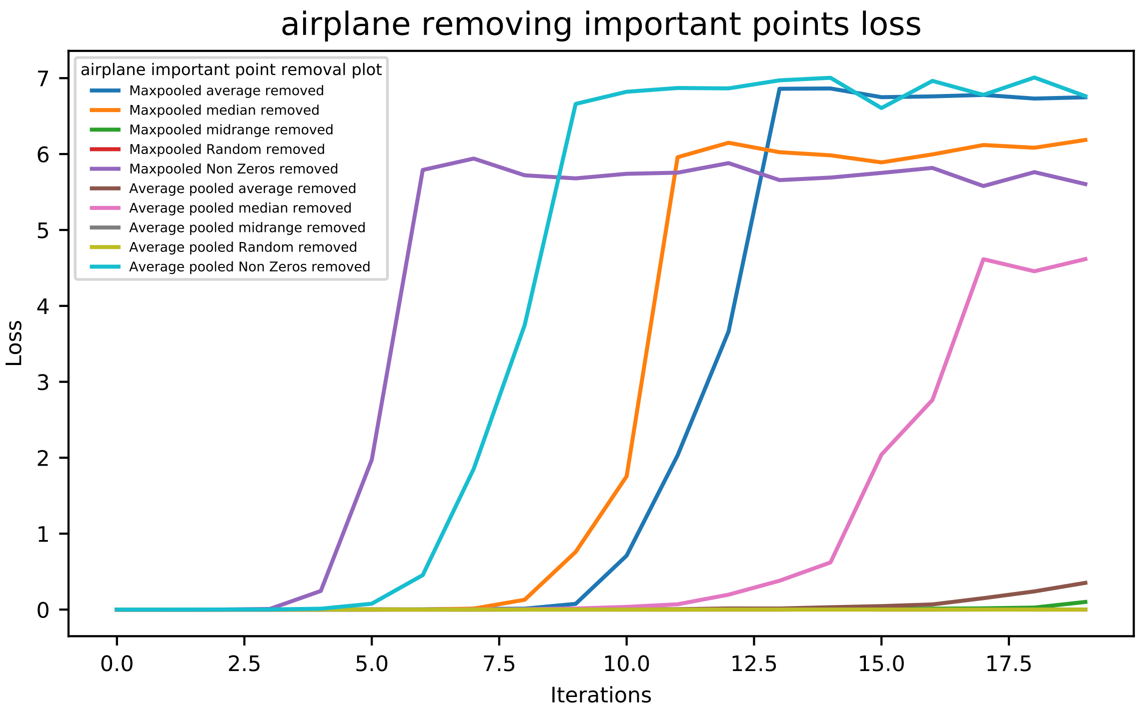


Figure 6.2: Airplane loss plot for the removal of important points for each pooling variant and each threshold method using 500 batches for each iteration.

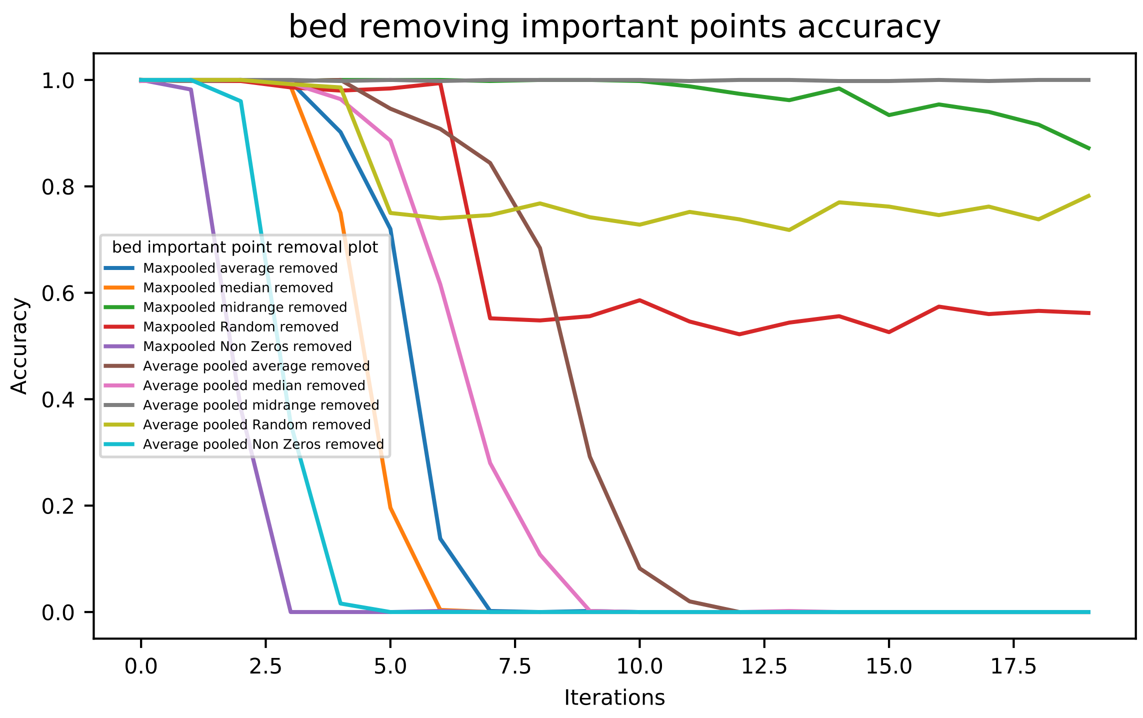


Figure 6.3: Bed accuracy plot for the removal of important points for each pooling variant and each threshold method with 500 batches for every iteration.

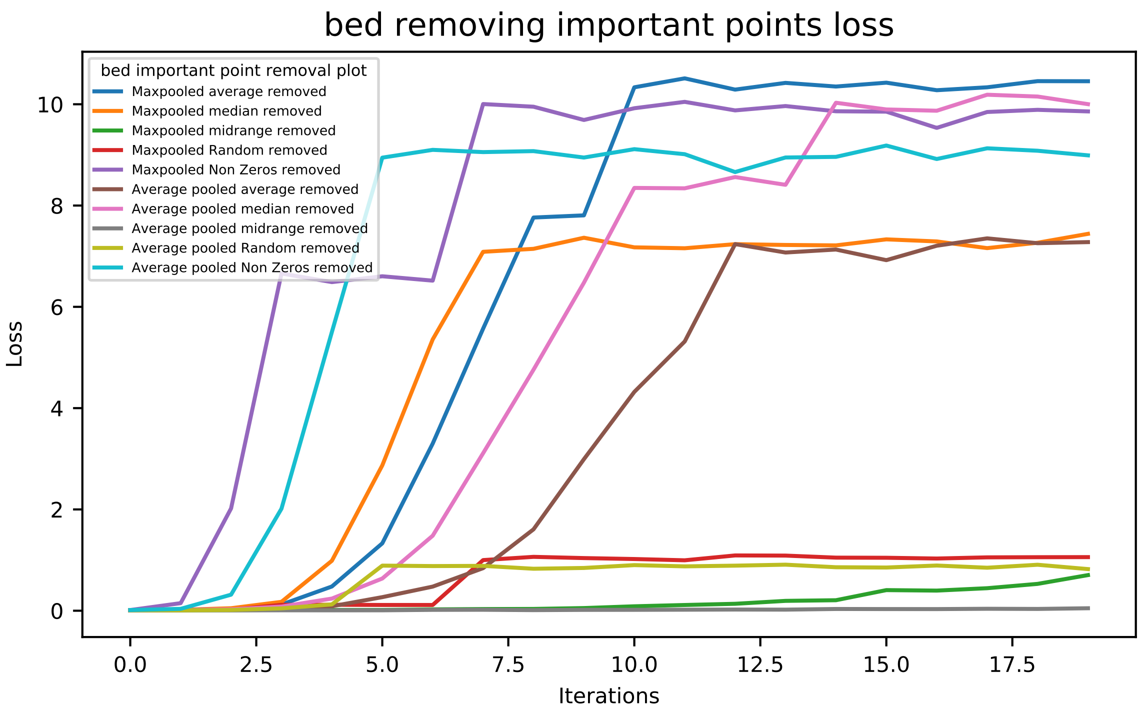


Figure 6.4: Bed loss plot for the removal of important points for each pooling variant and each threshold method with 500 batches in every iteration.

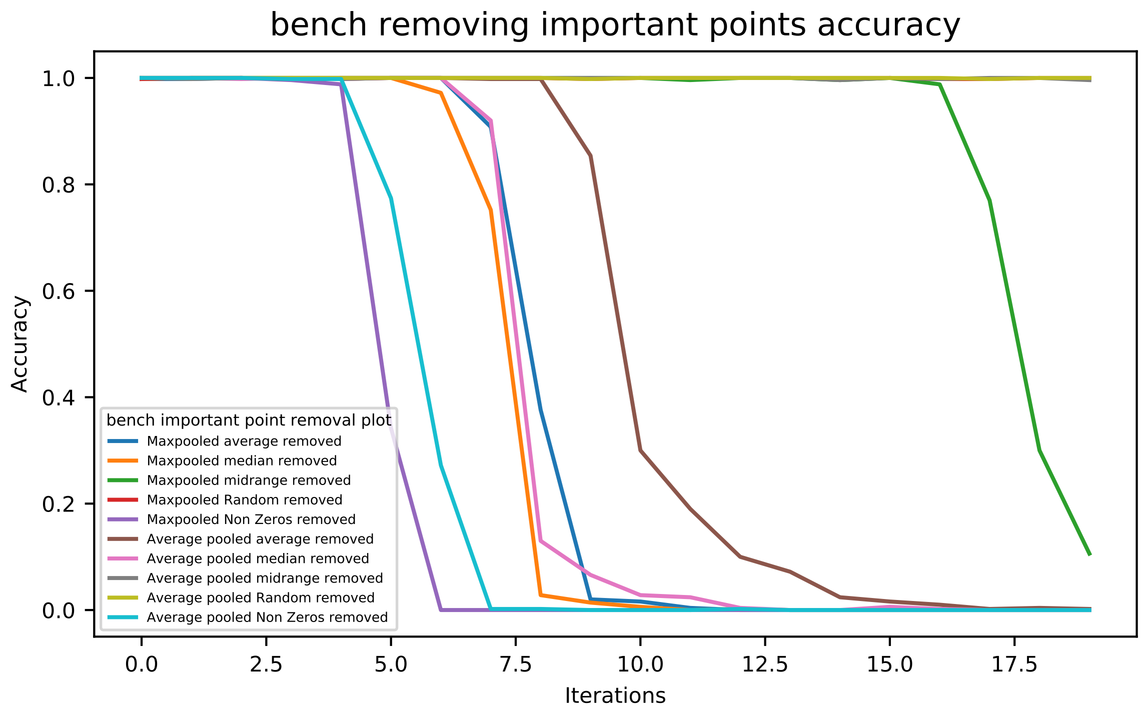


Figure 6.5: Bench accuracy plot for the removal of important points for each pooling variant and each threshold method with 500 batches per iteration.

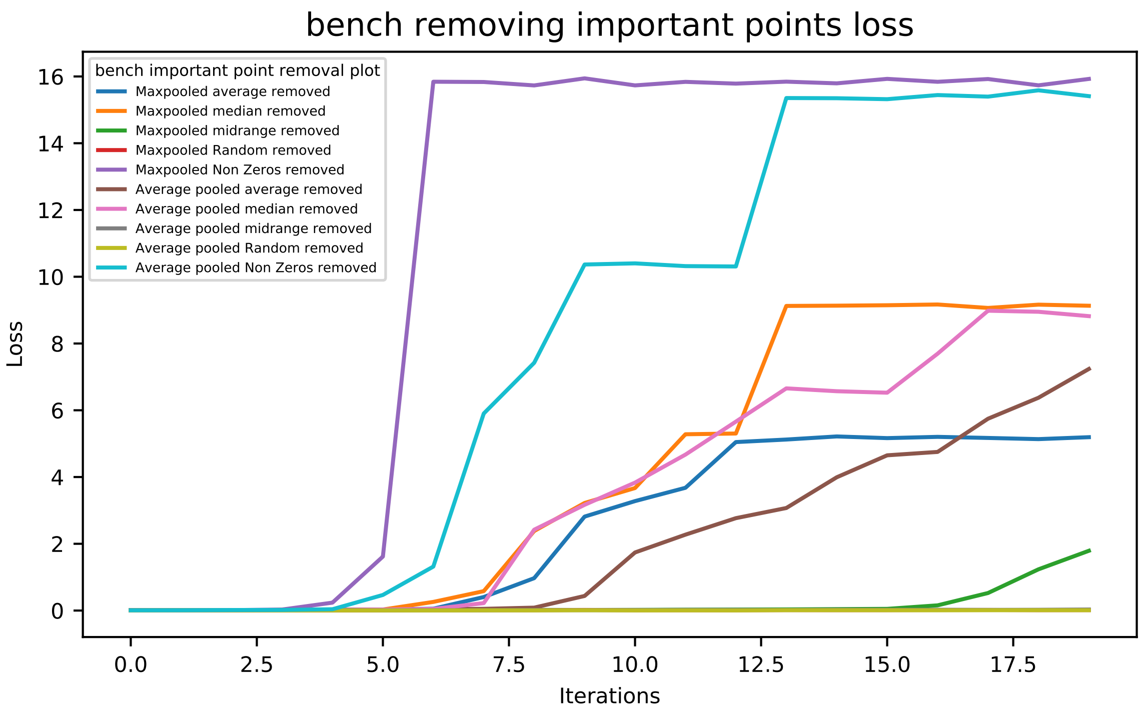


Figure 6.6: Bench loss plot for the removal of important points for each pooling variant and each threshold method with 500 batches used for every iteration.

6.2.2 Removing unimportant points

In the following test sets only the unimportant points will be removed. In order to determine which points are unimportant, the p-grad-CAM algorithm is applied once again. All point scores $T(p, w^c)$ that are below a predefined threshold are considered unimportant and will be dropped. To calculate the thresholds we use similar equal-zero, random, average, median and mid range methods as in the previous section as well as average and maxpooling, to compute the accuracy and loss.
The resulting residual point clouds are then fed back into the network, to be consecutively evaluated for their accuracy, loss and prediction changes. Since only low scoring unimportant points are being removed, this time no major drop of accuracy and loss respectively would be expected. This should be true because unimportant points should not drive the network decision making at all. However this would only be the case if our algorithm was indeed able to correctly identify the unimportant points and not falsely delete important points as well in the process. Therefor incorrect methods, deleting false positive points, would be expected to see an accuracy drop and respectively a raise of loss over the course of the evaluation.
For a first test the airplane, bed and bench point cloud objects were chosen again in order to have a comparison between the previous tests, that were removing important points. So far they indeed confirm the good results shown on the previous section.

The current result plots of the accuracy and loss for the airplane, bed and bench object can be seen on Figure 6.7,Figure 6.8,Figure 6.9,Figure 6.10,Figure 6.11 and Figure 6.12 on page 49. It shows some very interesting results, as the majority of threshold approaches drop almost immediately in accuracy down to zero. This means that these approaches have deleted not only unimportant points but a fair number of important points as well. Only the maxpooled equal-zero weight removal shows absolutely no drop in accuracy and therefor no raise in loss at all, the same equal-zero but average pooled method however does drop in accuracy. This means as a consequence that removing those points with weights of zero using the maxpooling approach for the gradient calculation, yields the highest precision on removing the unimportant points, while leaving the important points untouched. This also confirms the in the previous section observed better performance of maxpooling versus average pooling. Even though this was not a given result, it is not that much of a surprise considering that the original network used maxpooling too. This reinforces the maxpooled equal-zero weight removal approach as a prime candidate to be chosen for the final algorithm. However in order to finalize this choice, a greater set of measurements and a comparison to a state-of-the-art method is required to get a statistically more meaningful decision base, which is done in Section 6.3 on page 50.

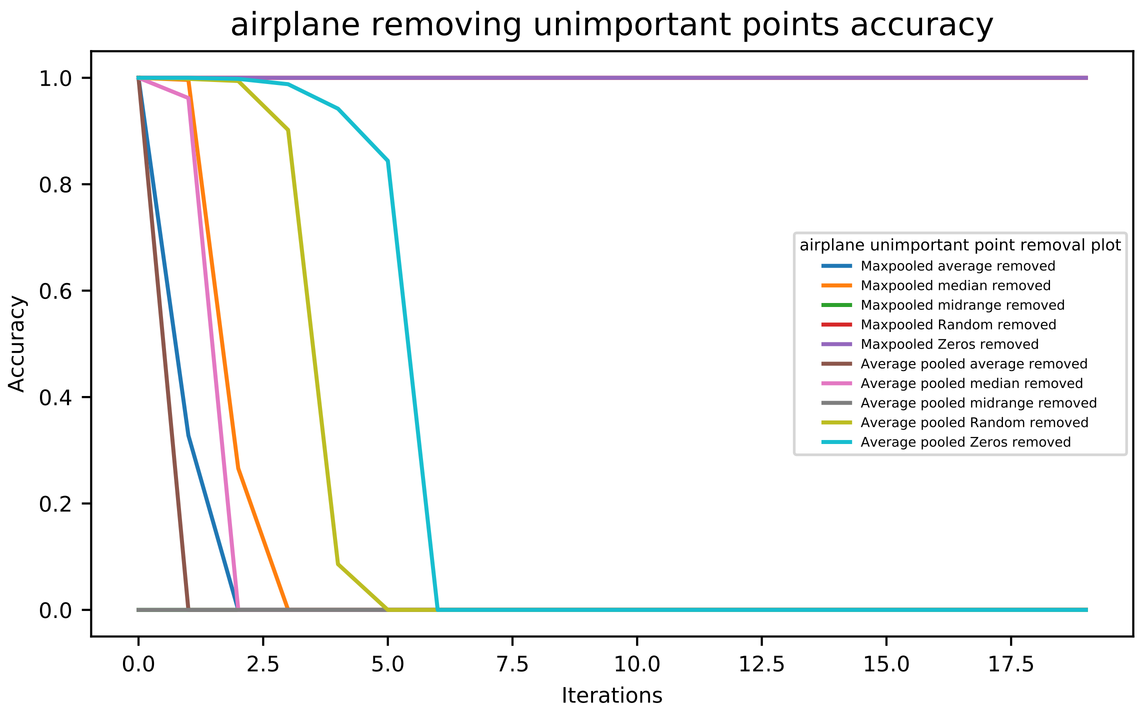


Figure 6.7: Airplane accuracy plot for the removal of unimportant points for each pooling variant (maxpooled vs average-pooled) and each threshold method (random, zero, arithmetic mean (average), median and mid range). Each iteration is the average value calculated over 500 batches of the same point cloud object. Once eligible points were removed the respective result point cloud is fed back into the next iteration.

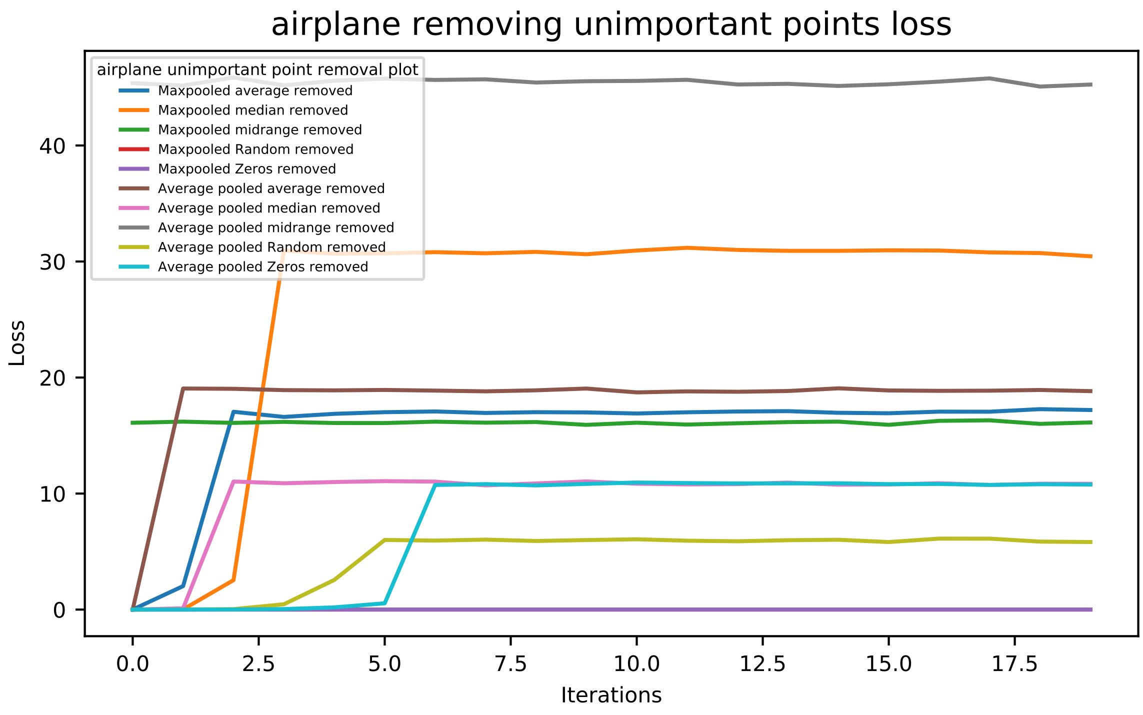


Figure 6.8: Airplane loss plot for the removal of unimportant points for each pooling variant and each threshold method. Each iteration is the mean of 500 batches.

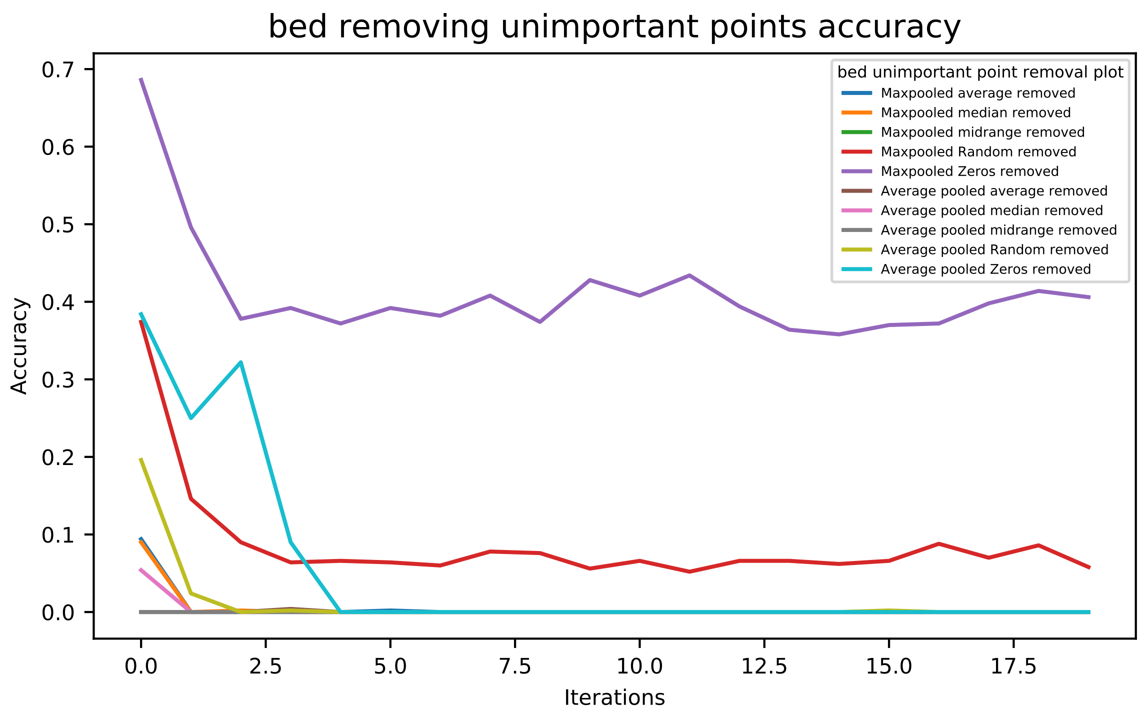


Figure 6.9: Bed accuracy plot for the removal of unimportant points for each pooling variant and each threshold method with 500 batches.

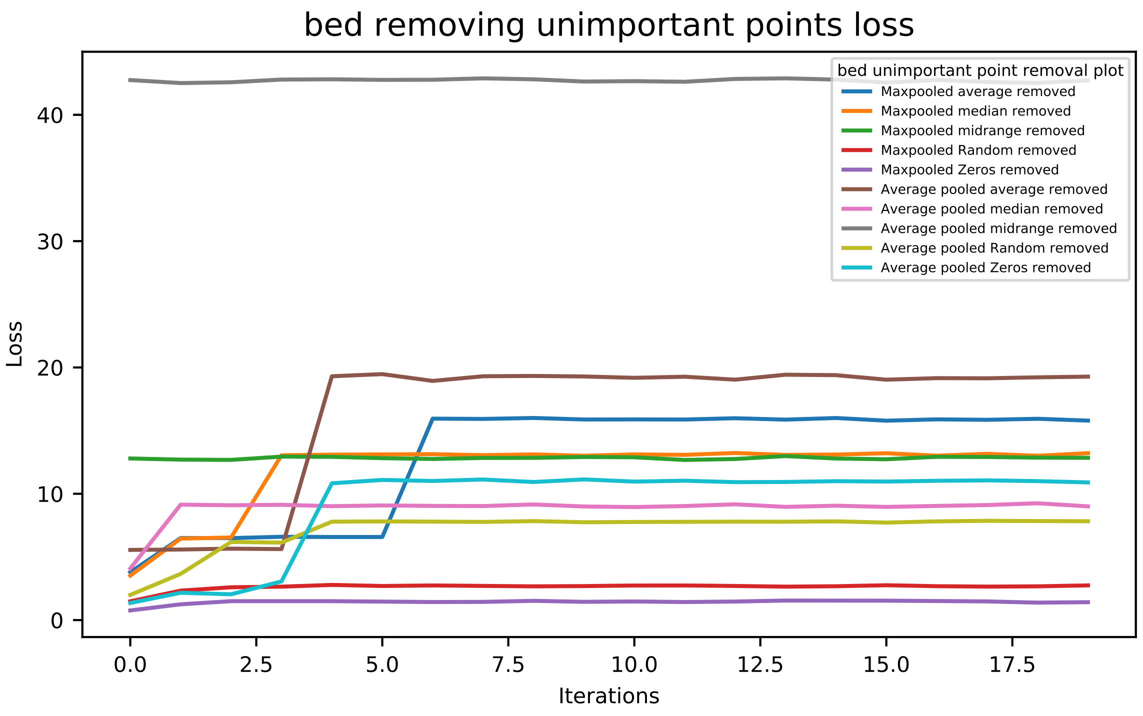


Figure 6.10: Bed loss plot for the removal of unimportant points for each pooling variant and each threshold method with 500 batches.

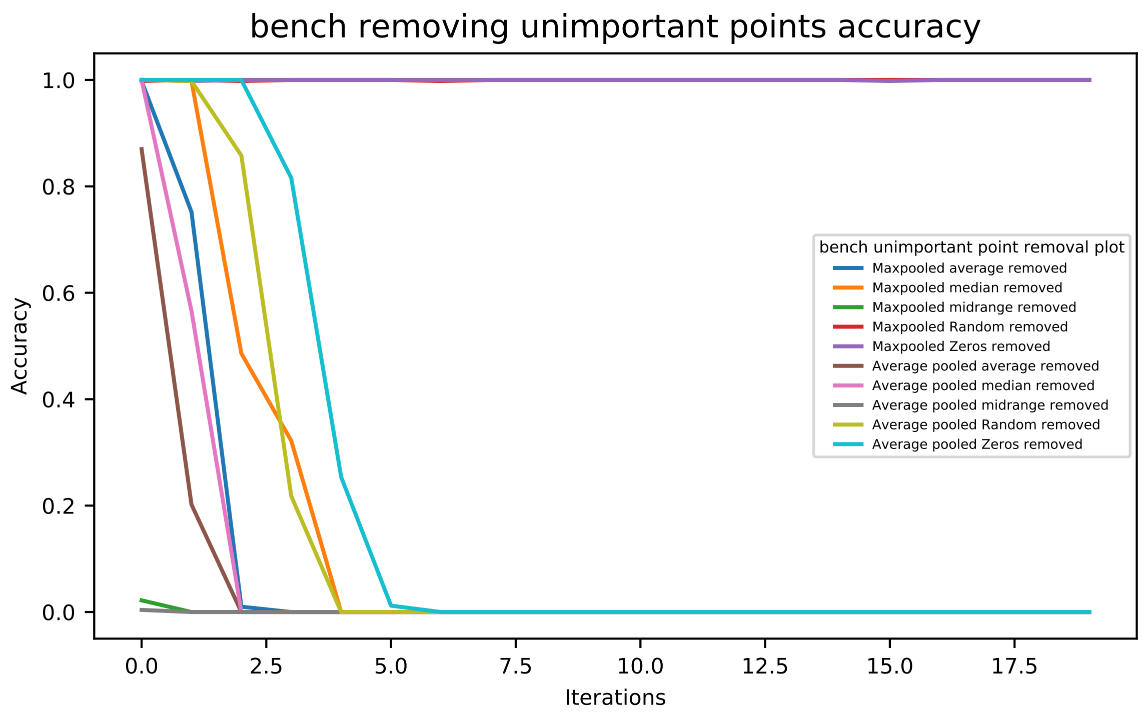


Figure 6.11: Bench accuracy plot for the removal of unimportant points for each pooling variant and each threshold method with 500 batches.

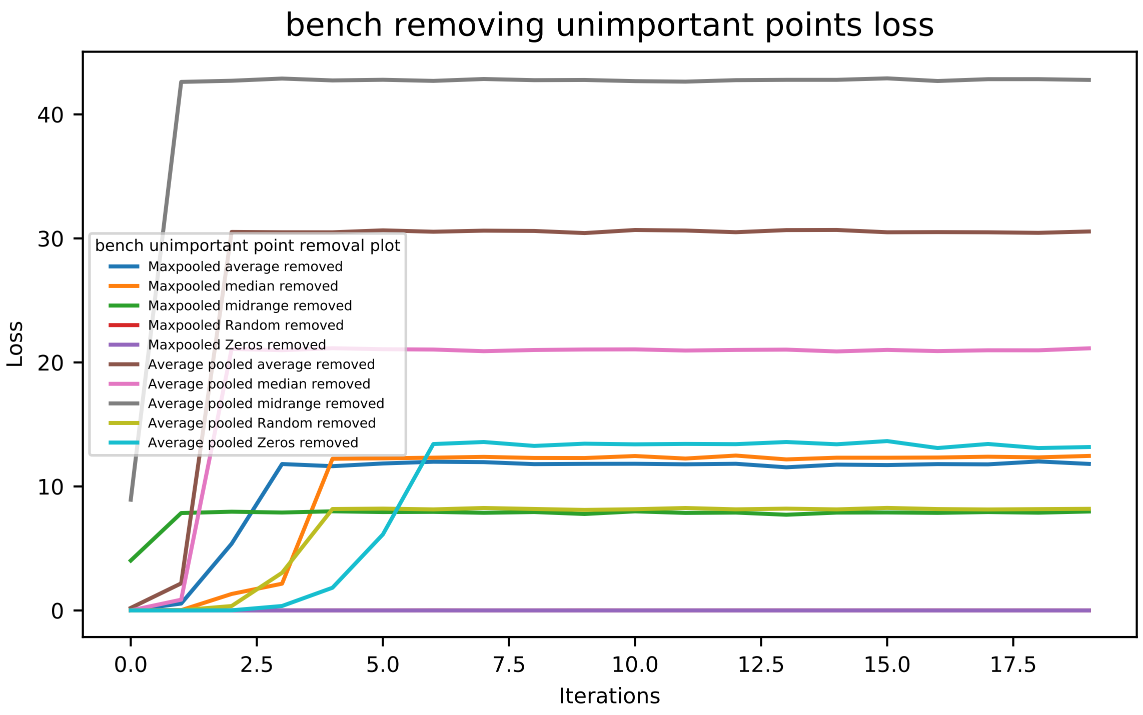


Figure 6.12: Bench loss plot for the removal of unimportant points for each pooling variant and each threshold method with 500 batches.

6.3 Comparing p-grad-CAM to Saliency Maps

As a next step we need to compare the resulting visualisations of our p-grad-CAM algorithm with the visual representations of the Saliency Maps [Zheng et al., 2018]. To gain a quality metric we let both algorithms run until the prediction accuracy plummets below 0.5 and starts deviating from the ground truth. The resulting visualization shows the points that had to be dropped until the prediction deviated, with each point colored by their respective score weight, which can be observed on Figure 6.13.

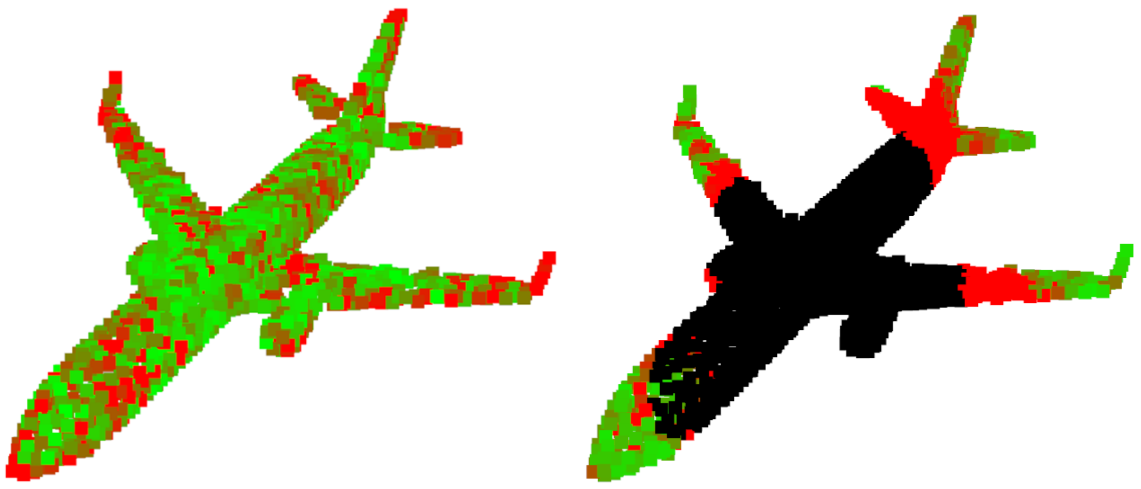


Figure 6.13: Comparison of our p-grad-CAM heat map (left image) plot with the same airplane point cloud object but with the saliency map algorithm applied to it (right image).

This result is indeed peculiar as it shows that the non zero threshold approach removes more then the minimal amount of points in order to change the prediction of the network. This means that even though the non zero approach is the fastest to drop in accuracy, it is also the method, that yields the least accurate heat map. Even though it does capture the critical points that drive the decision making, it also removes too many other less important points, which is unfortunate and rules this method out for the final algorithm. This requires us to find a new thresholding candidate, so instead of removing every point with a weight greater then zero, the point drop strategy is changed to only remove few points with high scores, to see if the algorithm provides a more accurate result. The best threshold method for this is surprisingly the mid range method, which showed the latest occurring accuracy drop in our measurements. This means that the p-grad-CAM algorithm will now also require more iterations before it concludes but now hopefully with a more accurate visualization of the important regions compared to the non zero threshold. Reason for this is that a lower number of points in each iteration is selected and dropped. Therefor it takes more iterations and in consequence more runtime until all important points were removed. This new approach also takes better account of the observation, that points supersede each other in their importance scoring for the network. Meaning that once the high score ranking points were removed, the previously low scoring points suddenly became more important for the network. This way the algorithm slowly works through all the critical points without dropping too many less important points, which should give a more precise visualization of the salient regions. The results of this new visualization using the mid range threshold are displayed on Figure 6.14, Figure 6.16, Figure 6.18 and Figure 6.20 over the following pages. For better comparability between the two algorithms, the accuracy with respect to the remaining points has been plotted, see Figure 6.15, Figure 6.17,Figure 6.19 and Figure 6.21 on page 55.

These plots show the main difference between the two algorithms, being that our algorithm removes a dynamic amount of points and thus needs less iterations to finish but seemingly also removes slightly more points then the Saliency Maps method in the process. The Saliency Maps algorithm works by always removing a pre-defined constant amount of points per iteration and the numbers of iterations itself. This requires hand tweaking the values for the iterations and points removed per iteration. In this example we tuned the saliency map to remove one point per iteration and then recompute the gradient based on the changed point cloud. The number of necessary iterations was then found by trying an arbitrary start value and then slowly adjust it until the prediction deviated. Other configurations would also be possible like removing 20 points per iteration. However the latter yields a lower accuracy on the results because the gradient is updated only once per iteration. This means that 20 points are dropped without taking the impact of the removal of each point into account. This yields a significant difference in the outcome of the algorithm, see Figure 6.22 on page 55.

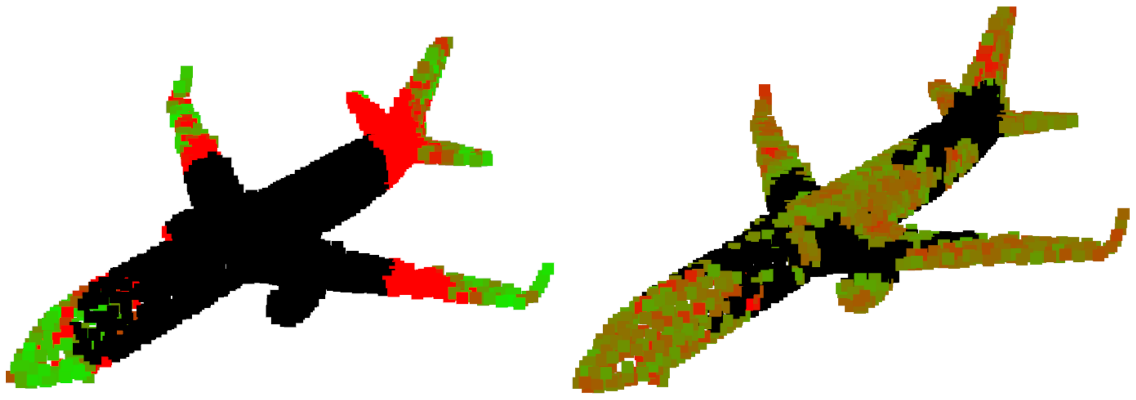


Figure 6.14: Airplane saliency map comparison between p-grad-CAM (left image) and Saliency Maps (right image). Here p-grad-CAM dropped 1250 out of 2048 while Saliency Maps dropped 690 points, before the prediction deviated.

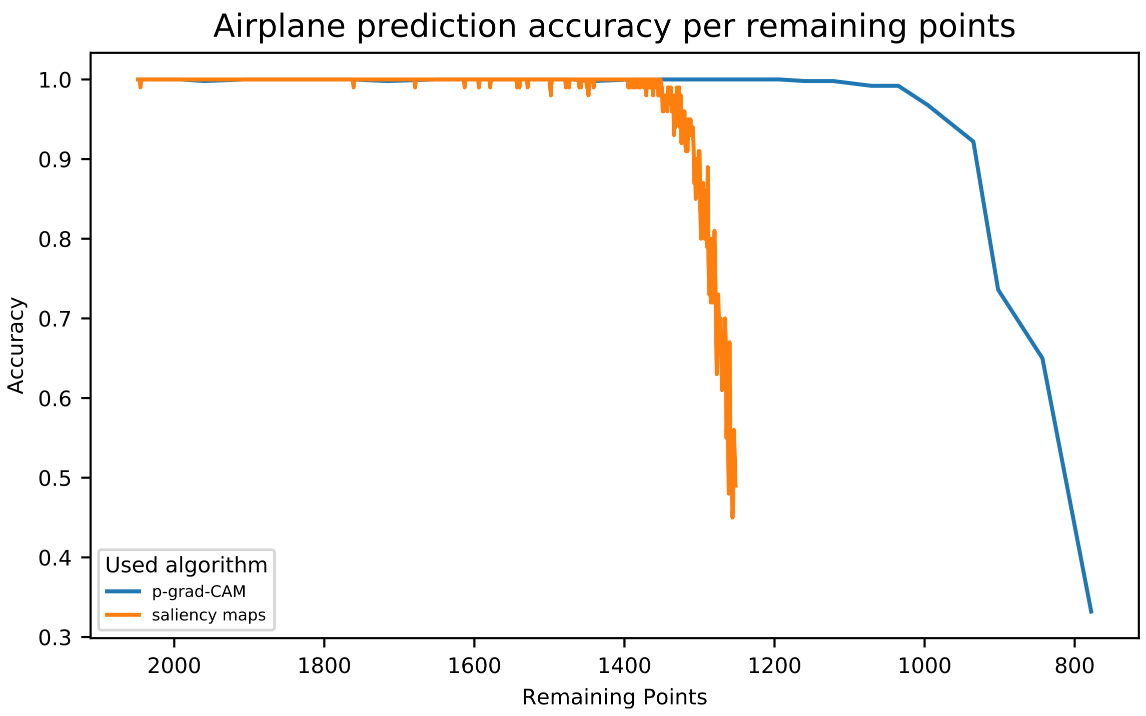


Figure 6.15: Prediction accuracy of the airplane point cloud object with high scoring points consecutively dropped until the prediction deviated.

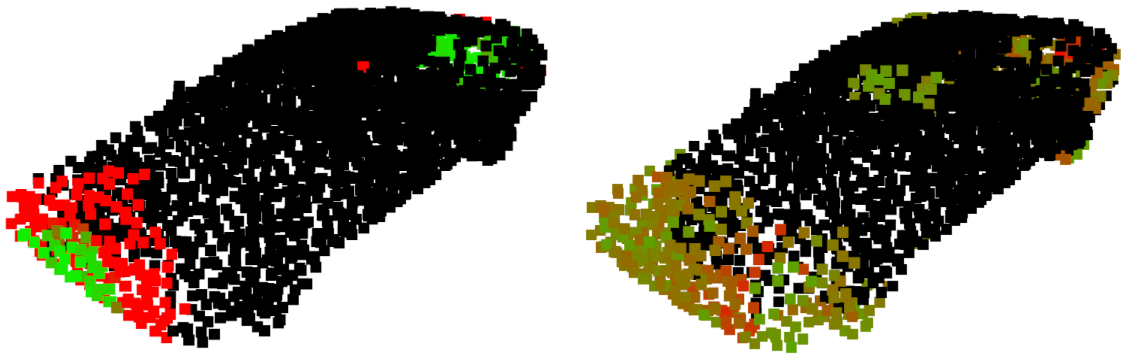


Figure 6.16: Car object saliency map comparison between p-grad-CAM (left image) and Saliency Maps [Zheng et al., 2018] (right image). Our approach dropped 671 out of 2048 points before the prediction deviated from the ground truth. The saliency approach needed roughly the half with 345 points before their prediction did not match the ground truth anymore.

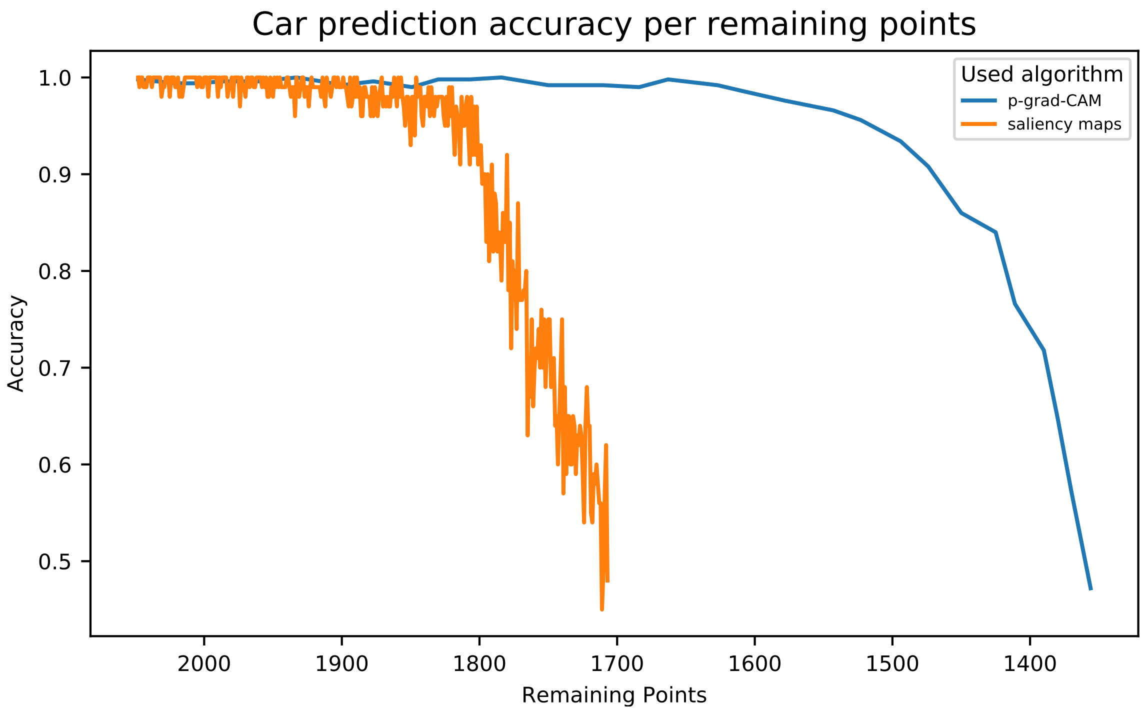


Figure 6.17: Prediction accuracy of the car point cloud object with respect to the remaining points after each iteration for each used algorithm.

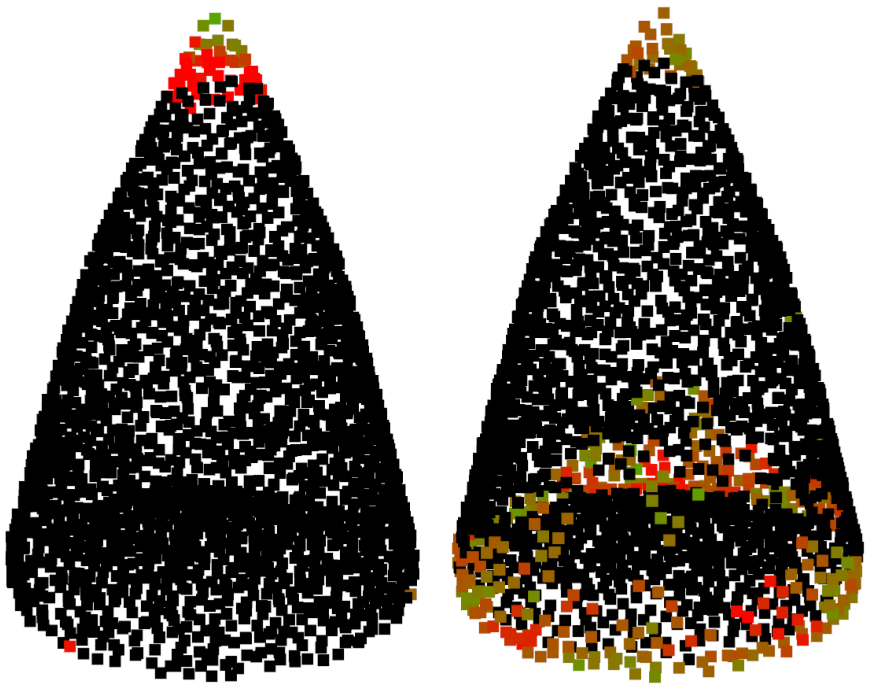


Figure 6.18: Cone saliency map comparison between p-grad-CAM (left image) and Saliency Maps [Zheng et al., 2018] (right image). Our approach dropped 326 out of 2048 points before the prediction deviated from the ground truth. The saliency only needed 59 points before their prediction deviated from the ground truth.

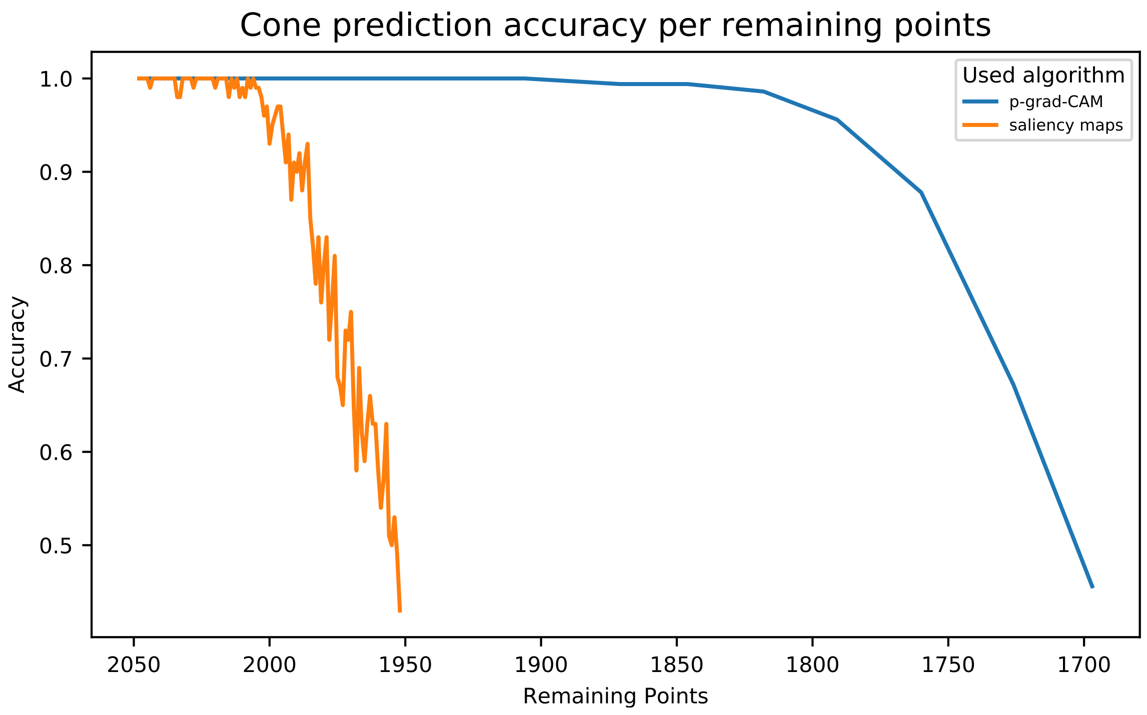


Figure 6.19: Prediction accuracy plot for the cone object, which shows the amount of remaining points, after the alorithms have terminated because the prediction deviated from the ground truth.

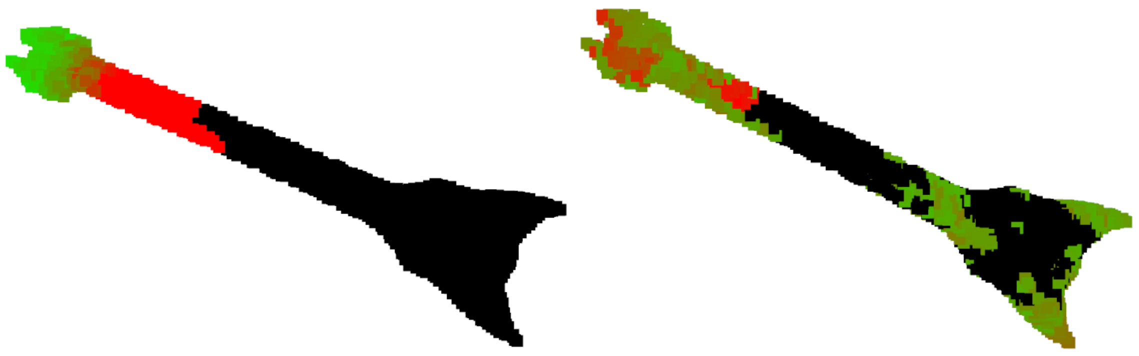


Figure 6.20: Guitar object saliency map comparison between p-grad-CAM (left image) and Saliency Maps [Zheng et al., 2018] (right image). Our approach dropped 810 out of 2048 points before the prediction deviated from the ground truth. The saliency approach again needed less with only 440 points before their prediction deviated too.

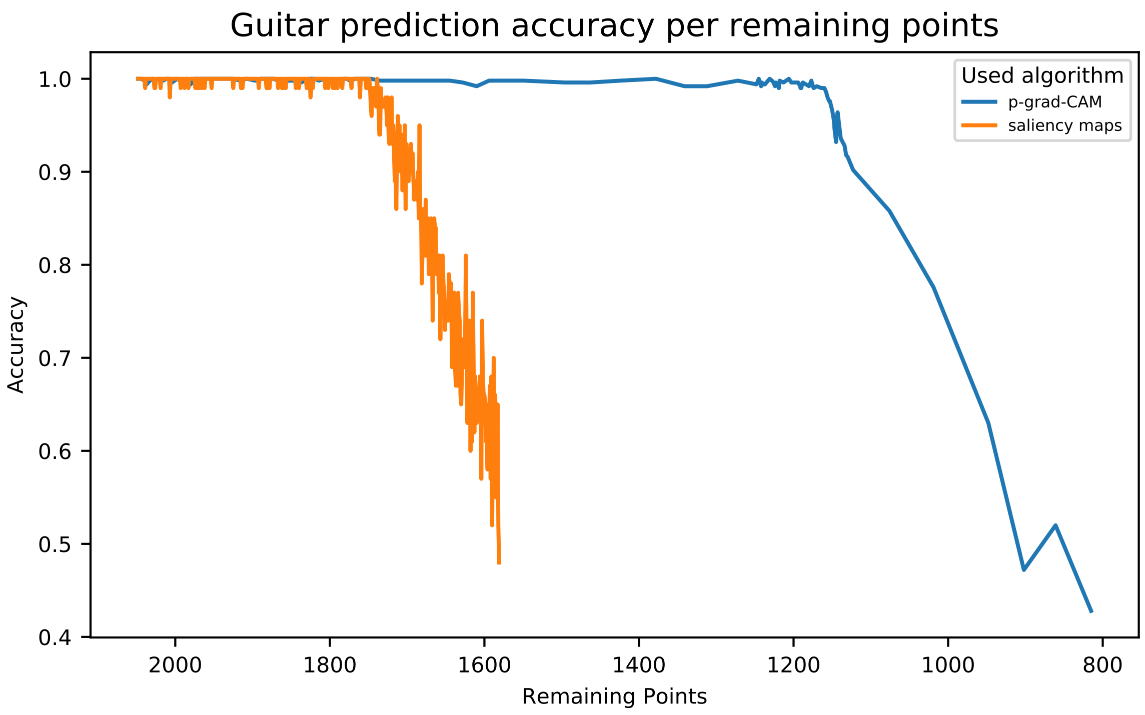


Figure 6.21: Prediction accuracy for the guitar point cloud object with respect to the remaining points after each iteration of the algorithms.

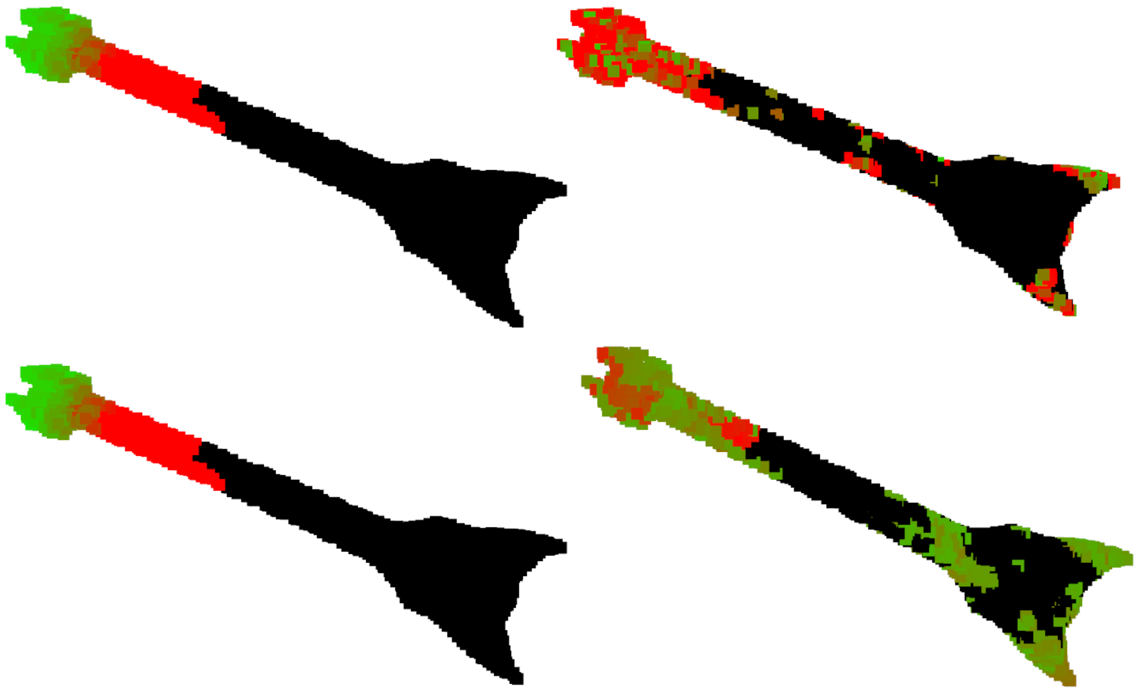


Figure 6.22: Guitar object comparison of the saliency algorithm removing 1 point per iteration (top left image) to 20 points per iteration (top right image). Both algorithms remove 440 points in total. Notice how the 20 points per iteration looks very similar to the result of our p-grad-CAM algorithm (bottom right)

6.4 Performance analysis

The comparisons, shown in the previous Section, however have a major issue. The Saliency Maps algorithm requires the amount of points, that should be dropped and the amount of necessary iterations, to be set a-priori by the user before starting the algorithm. Therefor Saliency Maps, in its current form, does not qualify for any bulk analysis of the entire stock of 40 available shapes from the `modelnet40_ply_hdf5_2048` database. The previous figures were all hand-crafted samples with a manually tuned Saliency Maps algorithm, via trial and error, to produce the best visualization under the minimal necessary amount of points to be removed.

However to still gain a meaningful comparison, it would be necessary to run both algorithms on all 40 shapes available from the database, without having to tweak values manually for each shape. This is only possible if the Saliency Maps approach would work similar to our p-grad-CAM algorithm, self evaluating the removal of points with respect to the impact on prediction accuracy. To achieve this we have slightly changed the Saliency Maps algorithm code so that it removes one point per iteration and then evaluates the changed point cloud for prediction accuracy drop. This enables the Saliency Maps algorithm to be used without any manual tweaking of variables, since the number of points removed is constant 1 and the number of iterations depends on the deviation of the prediction. For better clarity we named this modified algorithm Adaptive Saliency Maps (ASM). By using this altered algorithm, unsupervised automated bulk processing of all 40 shapes available from the `modelnet40_ply_hdf5_2048` database becomes possible. In the following we collected the amount of points removed and the required total algorithm run times for both the p-grad-CAM and ASM algorithm over all 40 available shapes. The results can be observed on Figure 6.23, Figure 6.24, Figure 6.25, Figure 6.26, Figure 6.27, Figure 6.28, Figure 6.29 and Table 6.3 on page 64. It is particularly interesting to see that the ASM test results further confirm the performance observations of the hand-crafted samples using the original Saliency Maps algorithm. It also shows that the ASM algorithm achieves a similar accuracy performance as the original, which required tweaking values manually in a trial-and-error fashion, yet now being much less time intensive. The results from Table 6.3 on page 64 also show how the p-grad-CAM algorithm roughly requires in average about $\frac{638}{364} \approx 1.75$ times more points to be removed before the prediction changes compared to ASM. This observation is congruent with the observations, where p-grad-CAM colors noticeable more points as important then ASM does. This means that, by inversion of the premise, our approach is about $\frac{638-364}{638} \cdot 100 \approx 43\%$ less accurate then the ASM approach, in regard of finding the minimum amount of points necessary for a prediction change. It has to be mentioned though, that the p-grad-CAM approach still contains the majority of the critical points that ASM had as well, plus some additional less important points that did neither appear in the ASM nor the original Saliency Maps approach either. One result which is also striking the eye is that the `flower_pot`,`night_stand`,`person`,`stairs`,`stool` and `wardrobe` had zero points removed for both algorithms. This indicates the there was not even a single iteration executed because the prediction was never correct in the first place. This was confirmed when we looked at the offending shapes using just the original Pointnet and the prediction was still wrong. Pinpointing the exact reason for this could be subject for future research.

Another remarkable result is the required run time of the p-grad-CAM approach compared to the Adaptive Saliency Maps algorithm, which roughly yields a speedup of $\frac{263s}{115s} \approx 2.287$, see Table 6.3 on page 64 for details. So even though the ASM approach yields a more accurate result, p-grad-CAM requires less time to finish, due to our thresholding system which removes more points per iteration compared to ASM, see Figure 6.23,Figure 6.24,Figure 6.25,Figure 6.26,Figure 6.27 and Figure 6.23. This is not surprising since removing more points results in less iterations to be required before the algorithm could terminate. With the high computational cost of the evaluation, since the network has to be queried a hundred times per iteration for a reliable averaged prediction result, less iterations directly translate into less run time. Therefor in conclusion the p-grad-CAM approach, in its current form, provides a good compromise between accuracy and run time, proving that the grad-CAM [R. Selvaraju et al., 2017] concept is also applicable in the realm of 3D point clouds.

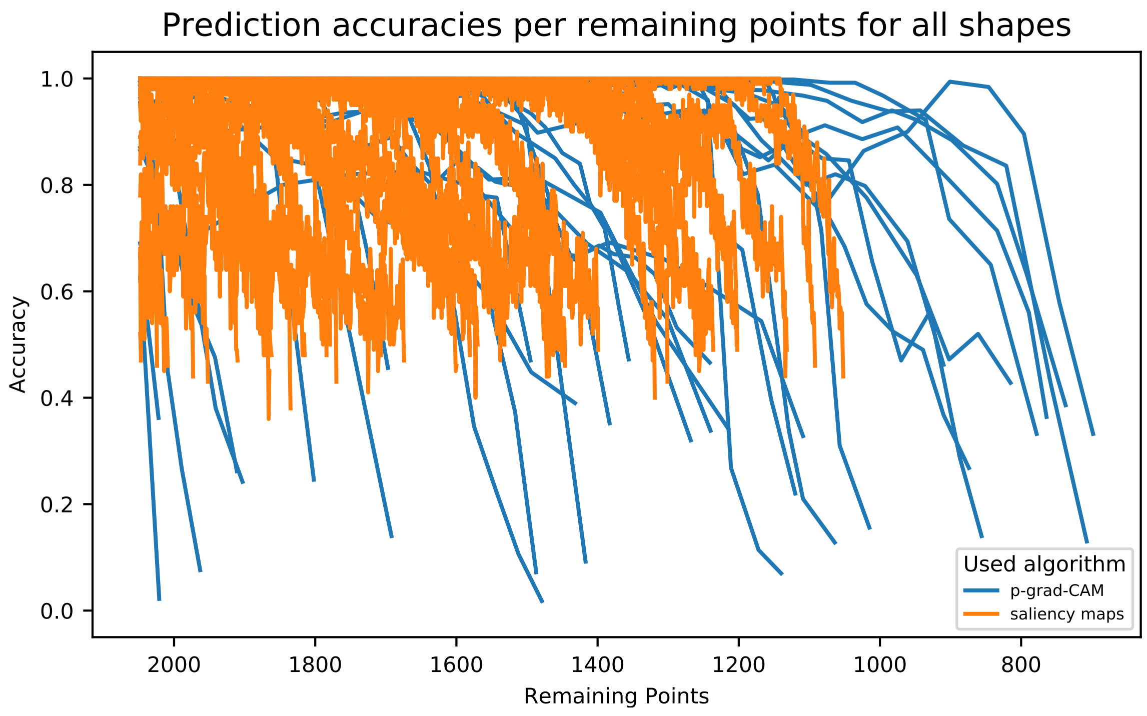


Figure 6.23: Change on the total remaining amount of points removed for each iteration for each of the 40 available point cloud objects, respectively for both the p-grad-CAM and ASM algorithm. All objects start with 2048 points and successively have some points removed until the prediction changes. For better readability the shapes have been split into sets of 5 objects for each algorithm which are shown on the following pages.

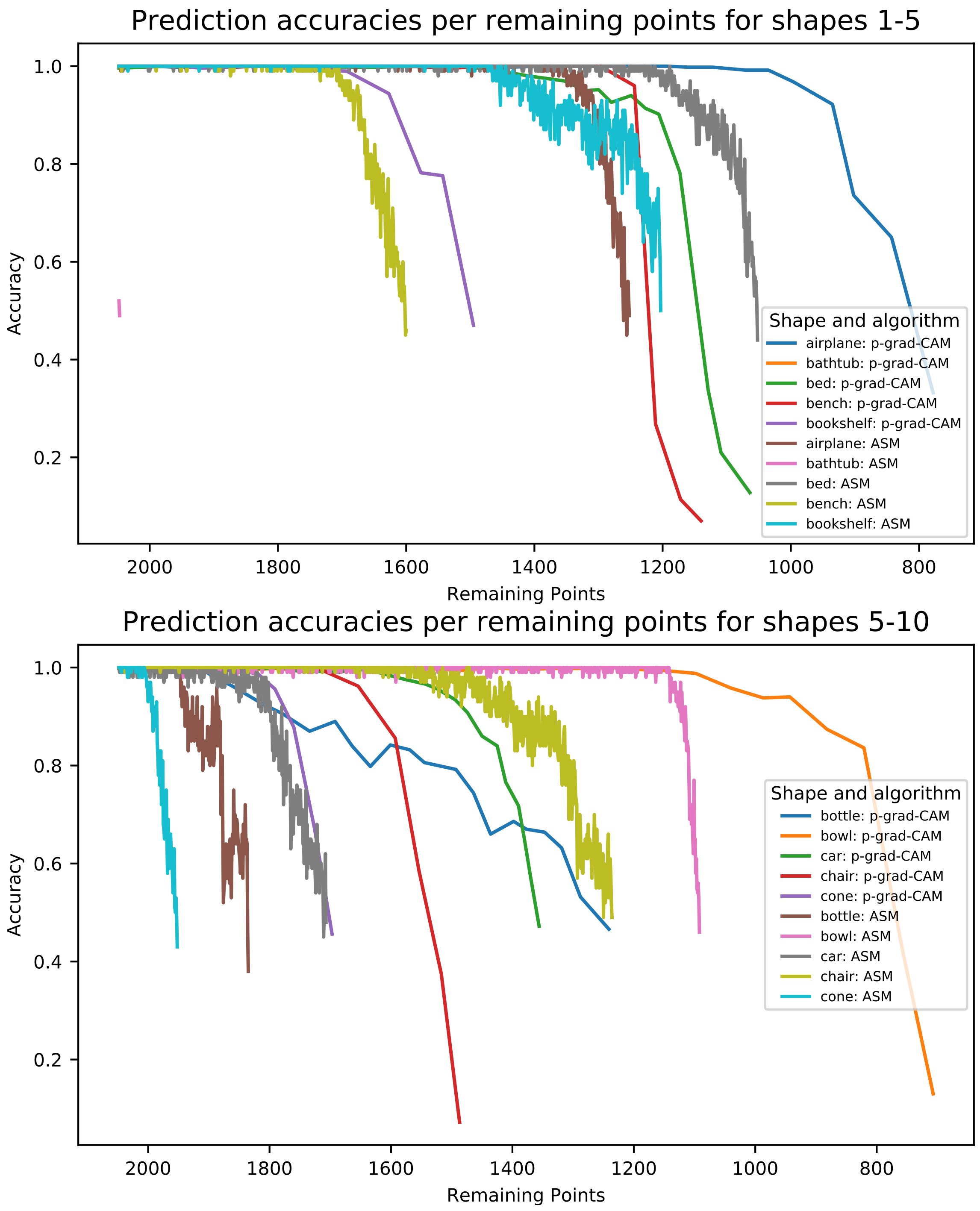


Figure 6.24: Change on the total remaining amount of points for each iteration of p-grad-CAM, with shapes 1-10 of all 40 point cloud object highlighted via colorization. All objects start with 2048 points and have p-grad-CAM remove points until the prediction changes.

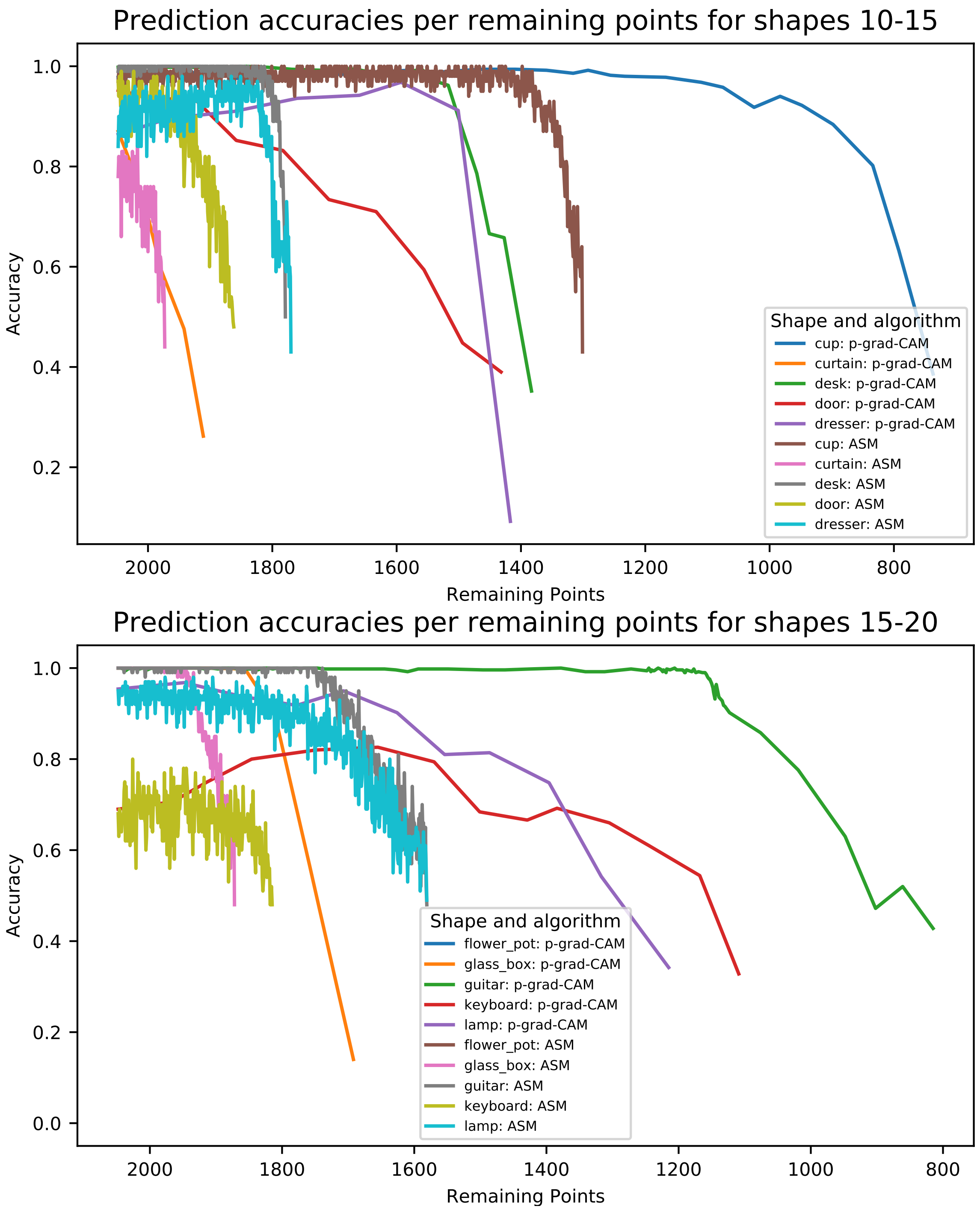
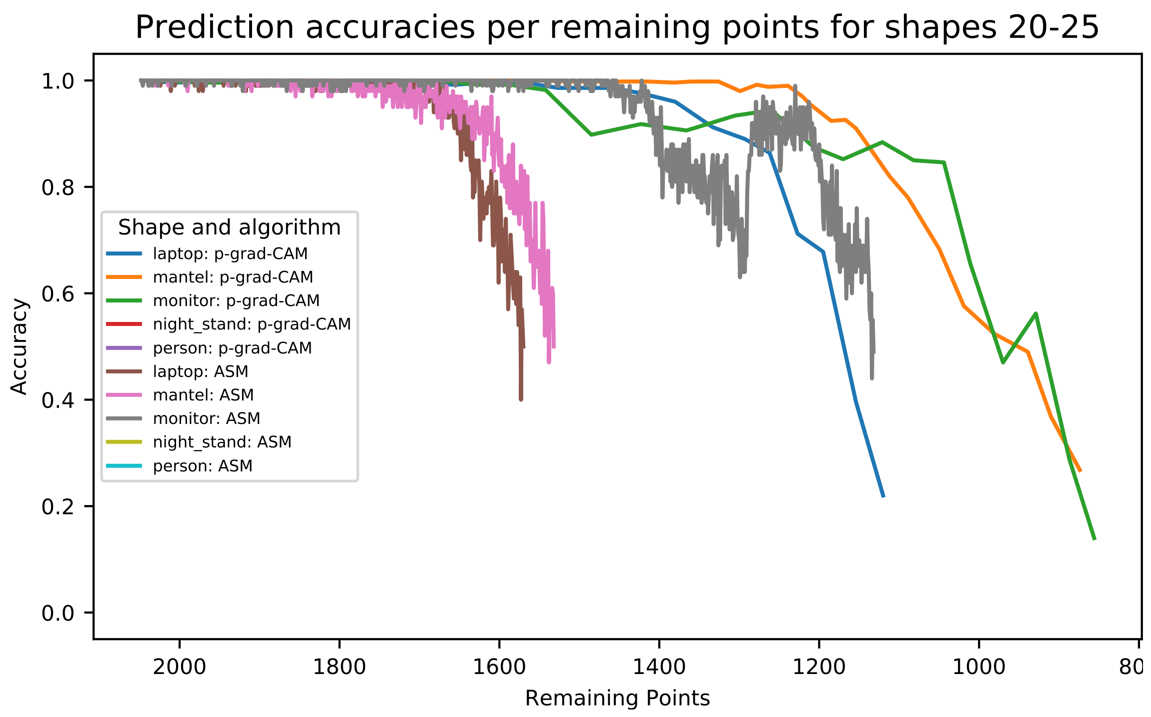


Figure 6.25: Change on the total remaining amount of points for each iteration of p-grad-CAM, with shapes 10-20 of all 40 point cloud object highlighted via colorization. All objects start with 2048 points and have p-grad-CAM remove points until the prediction changes.



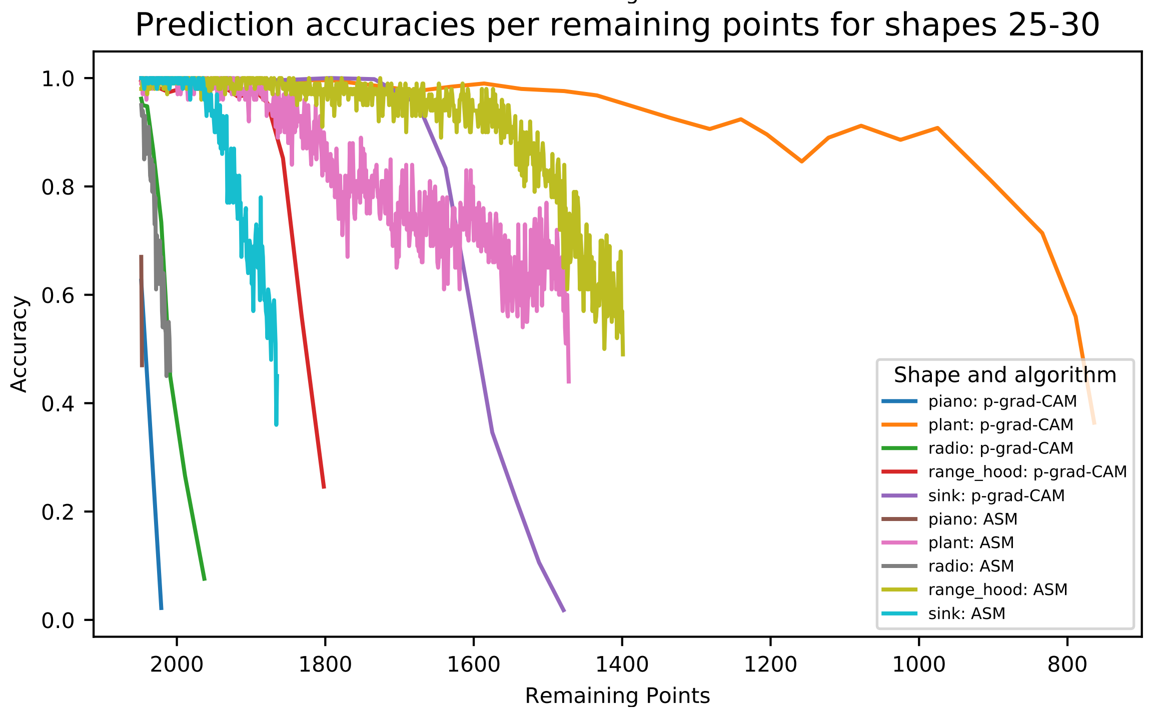
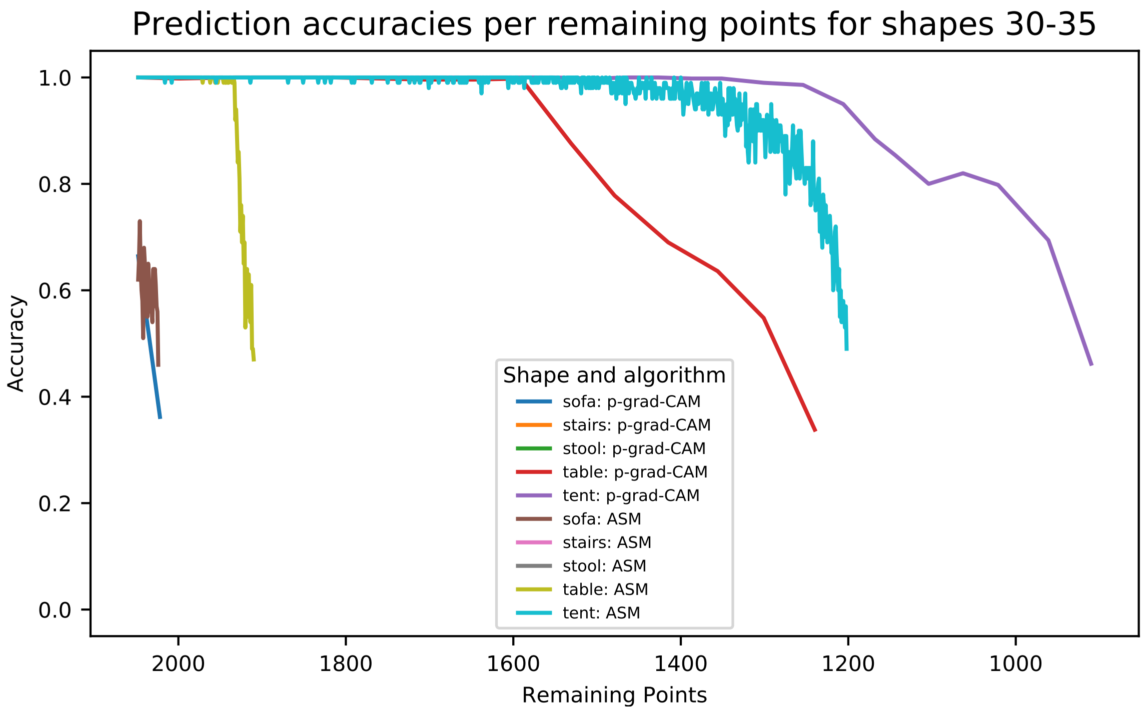


Figure 6.26: Change on the total remaining amount of points for each iteration of p-grad-CAM, with shapes 20-30 of all 40 point cloud object highlighted via colorization. All objects start with 2048 points and have p-grad-CAM remove points until the prediction changes.



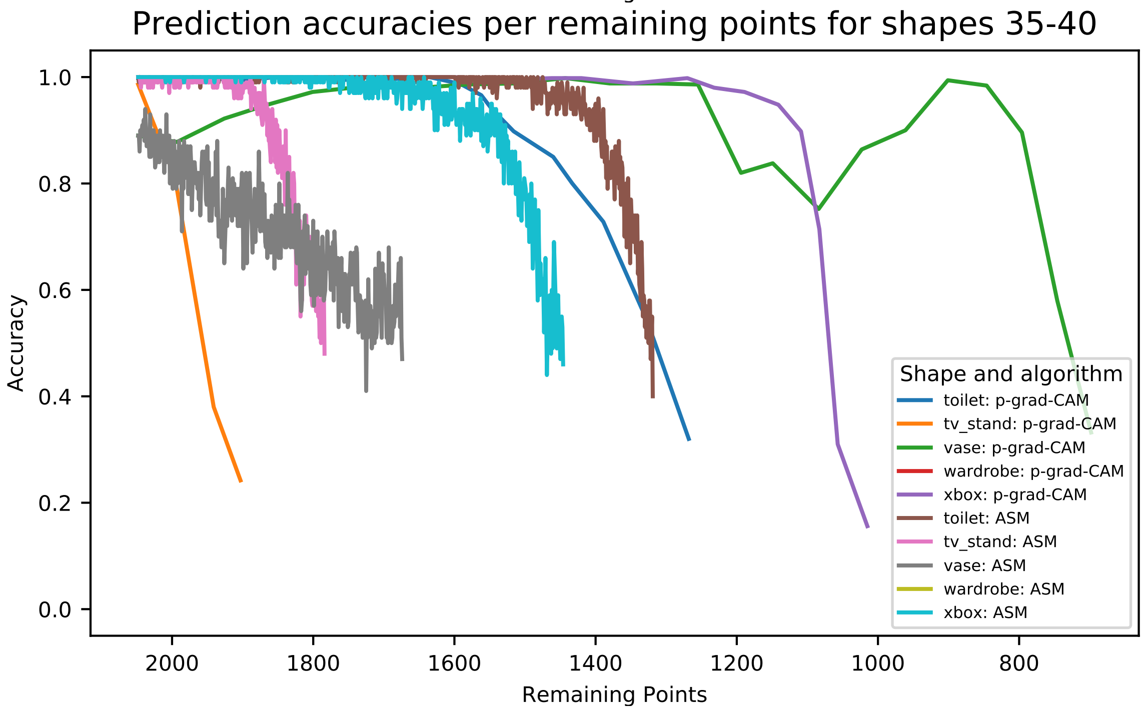


Figure 6.27: Change on the total remaining amount of points for each iteration of p-grad-CAM, with shapes 30-40 of all 40 point cloud object highlighted via colorization. All objects start with 2048 points and have p-grad-CAM remove points until the prediction changes.

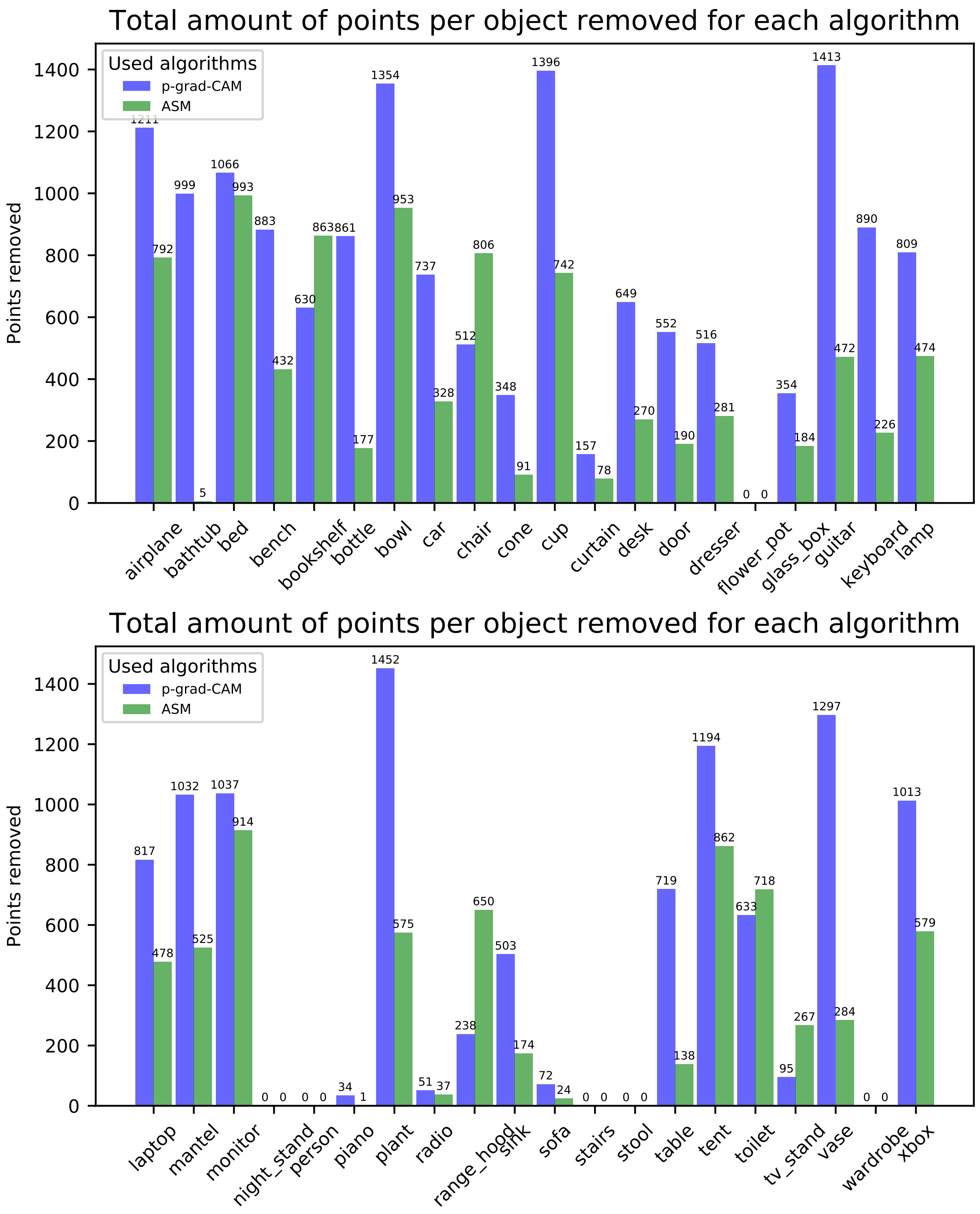


Figure 6.28: Total amount of removed points to make the prediction deviate from the ground truth for each of the 40 shapes from the **modelnet40_ply_hdf5_2048** database for both the p-grad-CAM and the ASM algorithm. All shapes initially contained 2048 points.

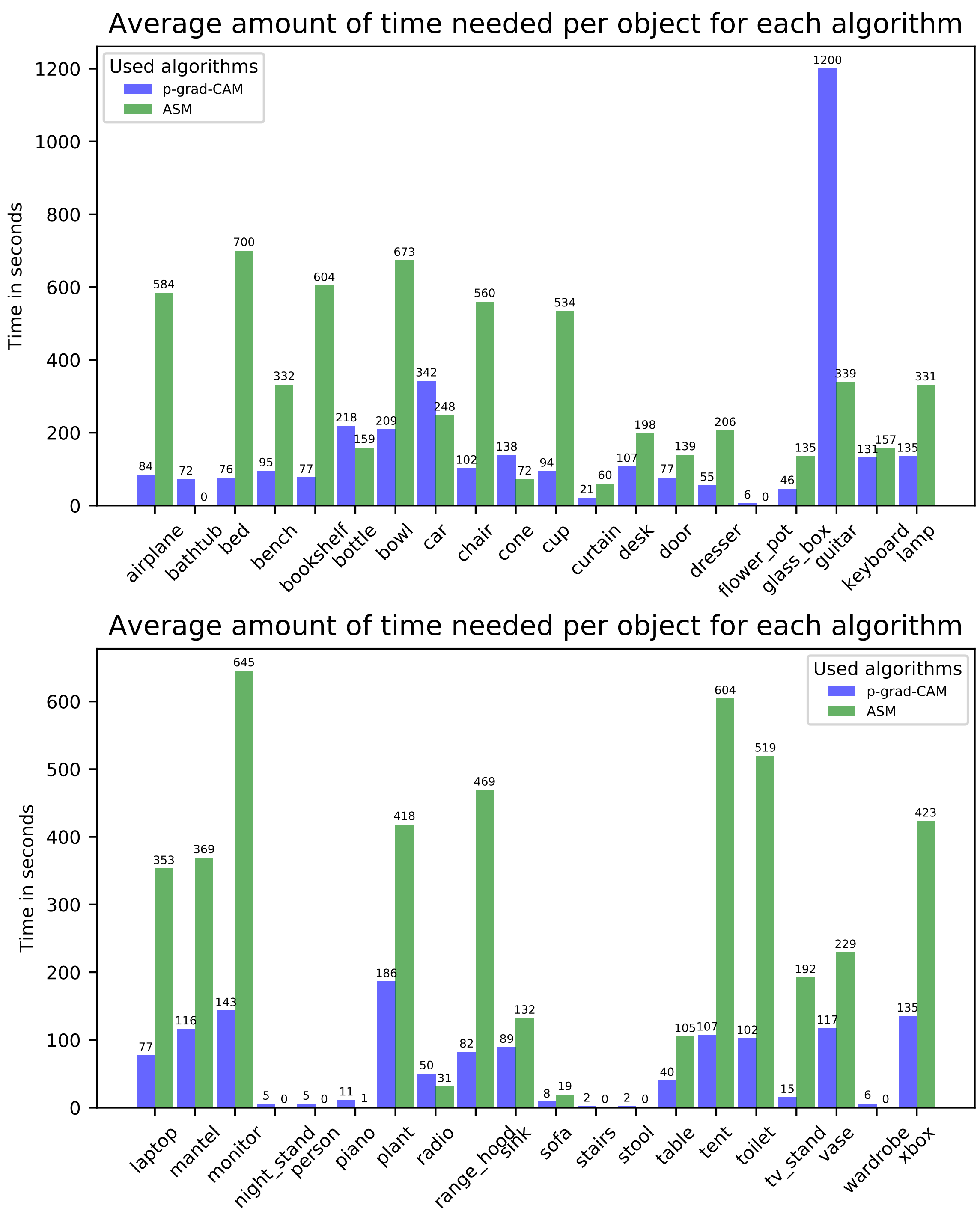


Figure 6.29: Total amount of algorithm run time for all of the 40 shapes from the **modelnet40_ply_hdf5_2048** database, plotted for the p-grad-CAM and ASM algorithm.

Measured average algorithm performance		
Used algorithms	Average points removed	Average Runtime
p-grad-CAM	638	115s
ASM	364	263s

Table 6.3: Average amount of points removed and average run time over all of the 40 shapes from the **modelnet40_ply_hdf5_2048** database for both the p-grad-CAM and the ASM algorithm. In conclusion p-grad-CAM is $\frac{638-364}{638} \cdot 100 \approx$ 43% less accurate but also $\frac{263s}{115s} \approx 2.287$ times as fast as ASM on average.

7 Summary

In the previous chapters the base idea of the Point cloud gradient Class Activation Mapping (p-grad-CAM) was introduced. The mathematical foundations of the underlying operations to retrieve the importance score weights $T(p, w^c)$ were discussed in a detailed design choice study. The initial approach also included room for uncertainty on which pooling or thresholding method to chose. These issues were examined in an ablation study, to determine which candidates ultimately qualified to be adapted for the final algorithm.
A possible weakness of Pointnet was then exposed when it became evident that Pointnet rotated point clouds only around the Y-Axis to augment the data set. This observation was backed by accuracy measurements of XYZ-Axis rotated point cloud objects which showed a significant drop in accuracy. This problem was fixed by our custom written rotation function. This required us to retrain the network to address this change in the database augmentation which showed significant improvements on accuracy. The results were then used to test the correctness of Point cloud gradient Class Activation Mapping (p-grad-CAM) against test sets of randomly removed points, to see if our approach actually yields better-then-random results and can actually compete with a similar approach called Adaptive Saliency Maps, which is a variation of Saliency Maps. In the very first tests of Point cloud gradient Class Activation Mapping (p-grad-CAM) the points with a greater-than-zero score removal approach looked the most promising regarding the speed upon the p-grad-CAM algorithm terminated. However at this point there was no significant quality metric for the algorithm available. As our method wasn't compared to another approach so far.
This changed once our p-grad-CAM algorithm was actually compared to the ASM method which differed drastically from our initial results, where the initially chosen greater-than-zero threshold method was used. After testing more threshold options it became evident that the mid range threshold yielded the best results. This was not actually surprising as it favors the most the removal of few important points, over many unimportant points of all tested threshold methods and thus gives a more accurate result upon which points are important for the network and which are not. Having all the test results available, it was now possible to assemble the final version of the p-grad-CAM algorithm, for the final bulk performance testing. The conducted tests then showed that the p-grad-CAM algorithm removes more points compared to the ASM approach but also needs less time to terminate. Whereas ASM is more accurate by having to remove less points before the prediction deviated but also required more time to finish. The original Saliency Maps ap-

proach had a very good performance runtime wise but required fine tuning the amount of points that should be removed a-priori. This makes trial-and-error procedures necessary to find the minimum amount of points that need to be removed in order to change the network prediction. This in turn offset by the initial gain in runtime because a lot of time is lost finding the best parameter configuration manually. Both the p-grad-CAM and the ASM approach have the advantage that they self-evaluate the changed point cloud for accuracy in each iteration and terminate once the prediction has deviated and the accuracy is below 0.5 in value.

7.1 Results

The p-grad-CAM algorithm results, shown in the previous chapters, yield a very close similarity over the areas of importance chosen by ASM. Still the ASM approach requires less points than the p-grad-CAM to be removed to provoke a false classification result of the network. This could not even be remedied in our algorithm by changing the point selection strategy to removing single points instead of multiple points, chosen by a threshold. In conclusion this means that p-grad-CAM yields a less accurate mapping of the networks attention and the critical points that actually drive the decision process, compared to the ASM algorithm. With p-grad-CAM it appears there exists more of a subset relation to ASM, where the minimum amount of critical points found by p-grad-CAM, is a subset of the amount of points detected by ASM. Another remarkable observation is that points "obscure" the importance of other points. This means that points with a previously lower importance score, suddenly had much higher scores values, once some points with high scores were removed. Another surprising result is that the mid range approach yielded the most accurate result here with the closest matching visual representation compared to ASM. Even though it was not the favourite pick from our first tests, where other methods initially appeared more promising but turned out to remove too many points in later tests. This is not very surprising as the mid range threshold favors removing few points with high point weight scores, which is pretty similar to what ASM does. Another issue that appeared in our tests was that, even if the original Saliency Maps approach required less points to change the prediction from correct to wrong, it had the downside of requiring the amount of points, that should be removed and the total amount of iterations themselves, to be set a-priori by the user. This inevitably requires trial-and-error methods to find the minimum amount of points and iterations to make the prediction change and therefor having captured the points of importance. This rendered the Saliency Maps approach to be unsuited for unsupervised bulk processing of all shapes, however such mass analysis of shapes was required to gain meaningful results. These issues were fixed by a slight rewrite of the Saliency Maps algorithm, to allow for automatic evaluation. This new variant was named Adaptive Saliency Maps (ASM) to address its difference compared to the original algorithm.

7.2 Discussion

The p-grad-CAM algorithm works by removing N points with score values higher then a pre-defined threshold and then evaluates the resulting residual point cloud for its accuracy and prediction in every iteration until the prediction of the network deviates from the ground truth. This yields the advantage that no major intervention of the user is required, to find a Class Activation Mapping (CAM) that captures the salient regions for a given class label. This is an improvement over the original Saliency Maps which required user intervention to manually tweak the parameters, that determined the amount of points removed per iteration and the total number of iterations themselves. However due to the nature of using a threshold, instead of computing the importance for all points and then removing only the highest scoring point, means that there is always the potential of more points being removed then the bare minimum. The reason for this is that each change of the point cloud may have an impact on the score values of the remaining points. Points that had one iteration before high values could suddenly have either lower scores are get dominated by another point with a much higher value making the point suddenly fall below the threshold. In our tests it turned out that using the threshold method did indeed yield a 33% worse accuracy compared to computing the scores individually for each point. However this also came at the cost of a 46.79 times worse runtime. So the threshold design choice is ultimately a compromise between runtime performance and accuracy. The airplane mid-range thresholded p-grad-CAM algorithm for example needed approximately 1 minute to finish its calculations and display the results to screen. Ideally our approach would remove the highest ranking point and recompute the weights for all points to take a possible weight change of the removed point into account. However this ideal algorithm often didn't even conclude after more then 15 minutes of runtime in our tests. The culprit of this bad performance can be traced back to the Tensorflow `sess.run()` command. This effectively makes the evaluation phase of our algorithm, where the changed point cloud is fed back to the network in order to retrieve the accuracy, the main performance sink of p-grad-CAM. These fairly long runtimes for both algorithms also makes them both unsuited for any kind of real time application. However this is not that much of an issue considering that both p-grad-CAM and ASM are intended for the purpose of "debugging" a network to visualize the responsible parts of a point cloud shape for the current classification result of the network. So if average runtime is the main concern for the application, then the p-grad-CAM would be a good choice. The ASM algorithm would be better suited where the average accuracy of the visualization is the preferred attribute over runtime performance.

7.3 Conclusion

In this paper a 3D point cloud based version of the original pixel image gradient-based visual saliency region localisation grad-CAM [R. Selvaraju et al., 2017] is implemented

to gain a better insight upon the critical points of importance for the reasoning of the decision making process of Pointnet. To verify the correctness of the visualisation generated by the p-grad-CAM algorithm, it was compared to the performance of the Adaptive Saliency Maps (ASM) method, which is a variant of the Saliency Maps [R. Qi et al., 2017] approach, measured in the number of points removed and runtime per algorithm. Through ablative testing it was also possible to show the significance of our approach compared to just randomly removing points from the point cloud.
Various possible design options were tested against each other to justify the choices made in this paper that ultimately lead to the final implementation of the p-grad-CAM algorithm. By finally setting the threshold to the mid range method, it was possible to get the best accuracy versus performance compromise that made grad-CAM both terminate in reasonable time and yield a result that is visually very similar to the results produced by the ASM method. This has shown that the grad-CAM base idea is also applicable to the realm of 3D point clouds applications and produces comparable results to a state-of-the-art method such as Saliency Maps.

7.4 Future work

The Point cloud gradient Class Activation Mapping (p-grad-CAM) algorithm in its current form however still shows room for improvements. Especially the performance is still lacking when compared to the ASM algorithm. It has a much better run time per iteration than p-grad-CAM, which could be a possible opportunity to investigate the exact reason behind this phenomenon. The visualisation could as well be subject for improvement because during all conducted tests, p-grad-CAM always required more points to be dropped before the prediction changed, compared to ASM. This issue was mainly attributed to the way we chose the point drop candidates by applying a threshold instead of evaluating each point individually. However in our tests it turned out, that even if each point was evaluated individually without applying any kind of thresholding by choosing only the highest scoring point like ASM does, then the p-grad-CAM algorithm still produced an almost identical visualization comparable to the results that were using thresholds. This change of p-grad-CAM also came with a dramatically increased run time which were orders of magnitude worse then the variant using thresholds to chose the highest scoring points, Figure 7.1 on the next page shows the results of our findings. Via profiling the involved code it was possible to trace the reason for this bad performance back to the `sess.run()` command where the changed point cloud is evaluated for both the accuracy and to retrieve the feature vector in order to compute the weights for each point. It would be interesting to know if performance improvements are possible by streamlining that function or using newer versions of Tensorflow.

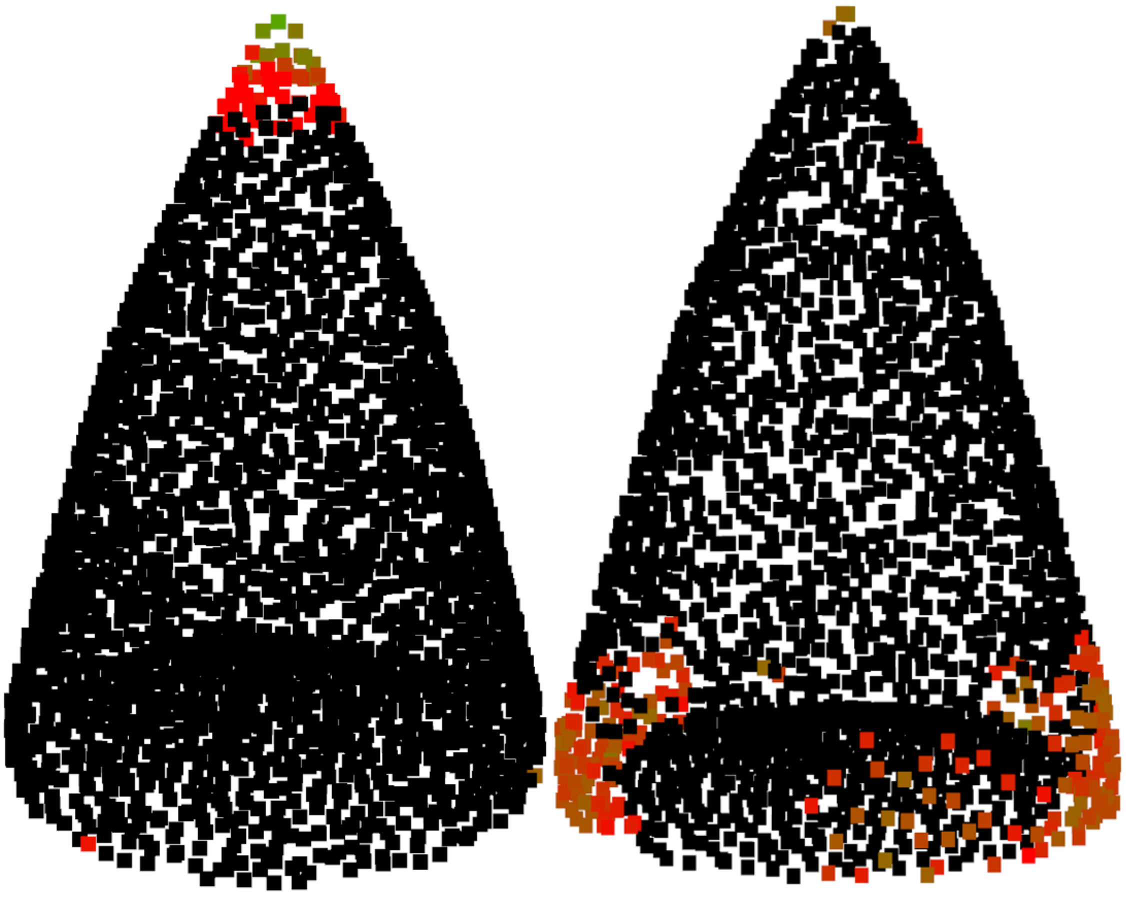


Figure 7.1: Comparison of the cone point cloud object removing one point per iteration for both the ASM (left image) and our p-grad-CAM algorithm (right image). The ideal p-grad-CAM algorithm, computing scores for each point individually, removed 219 points and terminated after 1411.393 seconds which is about 23.52 minutes. The ASM approach removed 91 points and terminated after 69.184 seconds. When using the p-grad-CAM with the mid range threshold our algorithm removes 326 points and terminates after 24.798 seconds. So by being $\frac{326-219}{326} \cdot 100 \approx 33\%$ less accurate with the mid range threshold than the ideal algorithm, we achieve a speedup of $S_{time} = \frac{1411.393s}{24.798s} \approx 46.79$ for the runtime performance.

Bibliography

Junhyuk Hyun, Hongje Seong, and Euntai Kim. Universal pooling - A new pooling method for convolutional neural networks. *CoRR*, abs/1907.11440, 2019. URL `http://arxiv.org/abs/1907.11440`.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016. URL `http://arxiv.org/abs/1609.02907`.

Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

Chen-Yu Lee, Patrick W. Gallagher, and Zhuowen Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In Arthur Gretton and Christian C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 464–472, Cadiz, Spain, 09–11 May 2016. PMLR. URL `http://proceedings.mlr.press/v51/lee16a.html`.

Phillip E. Pope, Soheil Kolouri, Mohammad Rostami, Charles E. Martin, and Heiko Hoffmann. Explainability methods for graph convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5099–5108. Curran Associates, Inc., 2017.

Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv:1612.00593v2 [cs.CV]*, page 19, 2017. URL `https://arxiv.org/pdf/1612.00593.pdf`.

Ramprasaath R. Selvaraju, Michael Cogwell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *arXiv:1610.02391v3 [cs.CV]*, page 24, 2017. URL `https://arxiv.org/pdf/1610.02391.pdf`.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualisingimage classification models and saliency maps. *arXiv*, page 8, 2014. URL `https://arxiv.org/pdf/1312.6034.pdf`.

Yan Wang, Chao Wei-Lun, Divyansh Garg, Bharath Hariharan, Mark Campbell Campbell, and Kilian Q. Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. *arXiv:1812.07179v4 [cs.CV] 25 Apr 2019*, page 16, 2019. URL `https://arxiv.org/pdf/1812.07179.pdf`.

Yulong Wang, Hang Su, Bo Zhang, and Xiaolin Hu. Interpret neural networks by identifying critical data routing paths. *Computer Vision Foundation*, page 9, 2018. URL `http://openaccess.thecvf.com/content_cvpr_2018/papers/Wang_Interpret_Neural_Networks_CVPR_2018_paper.pdf`.

Jianxin Wu. Introduction to convolutional neural networks. *semanticscholar*, page 8, 2017. URL `https://pdfs.semanticscholar.org/450c/a19932fcef1ca6d0442cbf52fec38fb9d1e5.pdf`.

Jianming Zhang, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. *CoRR*, abs/1608.00507, 2016. URL `http://arxiv.org/abs/1608.00507`.

Tianhang Zheng, Changyou Chen, Junsong Yuan, and Kui Ren. Learning saliency maps for adversarial point-cloud generation. *CoRR*, abs/1812.01687, 2018. URL `http://arxiv.org/abs/1812.01687`.

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. *In CVPR*, page 9, 2016. URL `http://cnnlocalization.csail.mit.edu/Zhou_Learning_Deep_Features_CVPR_2016_paper.pdf`.

Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Appendice

Code listings

```python
import numpy as np
def rotate_point_cloud(batch_data):
    """ Randomly rotate the point clouds to augument the dataset
        rotation is per shape based along up direction
        Input:
          BxNx3 array, original batch of point clouds
        Return:
          BxNx3 array, rotated batch of point clouds
    """
    rotated_data = np.zeros(batch_data.shape, dtype=np.float32)
    for k in range(batch_data.shape[0]):
        rotation_angle = np.random.uniform() * 2 * np.pi
        cosval = np.cos(rotation_angle)
        sinval = np.sin(rotation_angle)
        rotation_matrix = np.array([[cosval, 0, sinval],
                                    [0, 1, 0],
                                    [-sinval, 0, cosval]])
        shape_pc = batch_data[k, ...]
        rotated_data[k, ...] = np.dot(shape_pc.reshape((-1, 3)),
                                      rotation_matrix)
    return rotated_data
```

Listing 7.1: Pointnet data augmentation via random rotation around the Y-Axis.

```python
import numpy as np
def rotate_point_cloud_by_angle(batch_data, rotation_angle):
    """ Rotate the point cloud along up direction with certain angle.
        Input:
          BxNx3 array, original batch of point clouds
        Return:
          BxNx3 array, rotated batch of point clouds
    """
    rotated_data = np.zeros(batch_data.shape, dtype=np.float32)
    for k in range(batch_data.shape[0]):
        #rotation_angle = np.random.uniform() * 2 * np.pi
        cosval = np.cos(rotation_angle)
        sinval = np.sin(rotation_angle)
        rotation_matrix = np.array([[cosval, 0, sinval],
                                    [0, 1, 0],
                                    [-sinval, 0, cosval]])
        shape_pc = batch_data[k, ...]
        rotated_data[k, ...] = np.dot(shape_pc.reshape((-1, 3)),
                                      rotation_matrix)
    return rotated_data
```

Listing 7.2: Pointnet data augmentation via a rotation of `rotation_angle` around the Y-Axis.

```
import numpy as np
def rotate_point_cloud_XYZ(batch_data):
    rotated_data = np.zeros(batch_data.shape, dtype=np.float32)
    for k in range(batch_data.shape[0]):
        rotation_angle_X = np.random.uniform() * 2 * np.pi
        rotation_angle_Y = np.random.uniform() * 2 * np.pi
        rotation_angle_Z = np.random.uniform() * 2 * np.pi
        cosval_X = np.cos(rotation_angle_X)
        sinval_X = np.sin(rotation_angle_X)
        cosval_Y = np.cos(rotation_angle_Y)
        sinval_Y = np.sin(rotation_angle_Y)
        cosval_Z = np.cos(rotation_angle_Z)
        sinval_Z = np.sin(rotation_angle_Z)
        #--Rotate around the X-Axis
        rotation_matrix_X = np.array([[1, 0, 0],
                                      [0, cosval_X, -sinval_X],
                                      [0, sinval_X, cosval_X]])
        #--Rotate around the Y-Axis
        rotation_matrix_Y = np.array([[cosval_Y, 0, sinval_Y],
                                      [0, 1, 0],
                                      [-sinval_Y, 0, cosval_Y]])
        #--Rotate around the Z-Axis
        rotation_matrix_Z = np.array([[cosval_Z, -sinval_Z, 0],
                                      [sinval_Z, cosval_Z, 0],
                                      [0, 0, 1]])
        rotated_data[k, ...] = np.dot(batch_data[k, ...].reshape(
                                      (-1, 3)), rotation_matrix_X)
        rotated_data[k, ...] = np.dot(rotated_data[k, ...].reshape(
                                      (-1, 3)), rotation_matrix_Y)
        rotated_data[k, ...] = np.dot(rotated_data[k, ...].reshape(
                                      (-1, 3)), rotation_matrix_Z)
    return rotated_data
```

Listing 7.3: Improved data augmentation via rotation around the XYZ-Axis.

Declaration on Oath

I confirm that I wrote the thesis on my own, without using any other than the declared sources, references and tools and did not use any impermissible tools. All passages included from other works, whether verbatim or in content, have been identified as such. The content of the presented thesis has not been used as a whole for another scientific work or publication so far. If own publications have been included partially, they have been identified as such.
I agree to make my thesis accessible to the public by having it added to the library of the Computer Science Department.

Hamburg, the __________ Signature: _________________________