Visualizing Convolutional Neural Networks Using Nudged Elastic Band Methods

Nion Schürmeyer

Supervisors: Luke Effenberger & Gordon Pipa

2020
September 05
Bachelor Thesis
Cognitive Science B.Sc
Institute of Cognitive Science

Visualizing Convolutional Neural Networks Using Nudged Elastic Band Methods

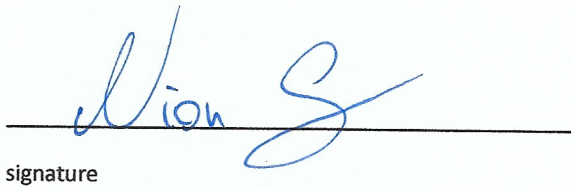
Nion Schürmeyer

Abstract

This thesis proposes a nudged elastic band algorithm applied to feature visualization, in order to produce short video clips, giving an overview over different facets of a feature. While deep learning approaches have remarkable success a in variety of applications, their major drawback is lacking interpretability, which poses problems for real world applications. A technique to gain insight on convolutional neural networks is feature visualization, where images are optimized towards activating a certain part of the network. These images only depict facets of the underlying feature and a diversity of visualizations is needed for a more accurate portrayal. The nudged elastic band algorithm takes multiple feature visualization as input and optimizes a path connecting them, towards maximum activation. The resulting path is rendered into a video, where each frame is optimized towards activating a certain feature map. Using the video clips and feature visualizations, I give an overview over the different features encoded in MobileNetV2, trained on the imagenet dataset. The videos are a convenient way of portraying features and future work can further improve image quality. Although feature visualization is a powerful tool for network interpretability, more research is needed for a better understanding of neural network behavior.

Declaration of Authorship

I hereby certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university.



signature

49076 Osnabrück 05.09.2020

city, date

Contents

Chapter 1

Introduction

Over the last decade machine learning models using deep neural networks won a wide range of contest and were able to perform tasks, for which no solution was found by traditional artificial intelligence yet. These achievements range over tasks in image processing, language processing and game playing among others [30, 32, 26]. The renewed interest in deep learning started when AlexNet won the imagenet challenge in 2012 [15]. Successful application was made possible by increased computational power, which allows for training deeper neural networks with larger datasets, although the ideas for deep learning date back to the 1940s. Since AlexNet, the field has expanded rapidly and a wide range of different architectures suited for processing different data structures have been developed [7, chapter 1]. One major drawback of deep neural networks is their black box nature. In comparison to traditional AI, where explicit rules are formulated by which the model arrives at its output, deep neural networks learn implicit rules, which are encoded in their parameters. This way of learning allows solving problems which are hard to near impossible using traditional AI, but comes at the cost of interpretability [7, chapter 1]. The processes by which the network arrives at its output are incomprehensible to us [23]. This is problematic if the model is applied to tasks where wrong decision are of serious consequence. For one it is important to know why a mistake happened in order to fix it. Further the question of responsibility rises. The developers of a model can do their best to avoid mistakes, but can they be held responsible for a mistake that nobody can understand? Thirdly being unable to comprehend the details of how a network processes information is unsatisfactory from a scientific viewpoint. For these reasons there has been a lot of research in understanding the inner workings of deep neural networks [18, 19, 20, 21, 22, 34] and to develop more interpretable architectures [16, 35].

This thesis focuses on the interpretability of convolutional neural networks (CNNs) trained to classify images. In particular on the field of feature visualization. CNNs learn and combine features to classify input images into categories. These features are encoded in the network parameters and can not simply be related to semantic ideas, which we understand. Feature visualization optimizes a random input image towards maximally activating a certain featuremap of the network. The resulting image can be interpreted as what that featuremap is looking for. Depending on the technique used, the images show different facets of the feature. My implementation is oriented after the Distill blog post by Olah, Mordvintsev, and Schubert [21], which provides an overview over various techniques. However the features do not only re-

late to one semantical idea. They can have a number of different facets and we might not even have a corresponding concept in our language for them. In order to provide an overview over these facets, I used a nudged elastic band (NEB) algorithm to produce short video clips, consisting of frames which are all optimized towards activating one featuremap. The NEB algorithm takes already optimized images as an input and optimizes a path between the images toward maximum activation [1][2]. I used the optimized images and videos to visualize the features of MobileNetV2, a lightweight state of the art network, trained on the imagenet dataset [24]. To summarize, the aims of this thesis are.

1. Implementing existing techniques of feature visualization.
2. Visualizing features using an NEB algorithm with feature visualizations as input.
3. Use 1 and 2 to provide on overview of the different features encoded in MobilenetV2.

[1] the code of my implementation can be found under https://github.com/Niioon/CNN_Visualizations

[2] the video results are hosted on github under https://niioon.github.io/CNN_Visualizations/

Chapter 2

Convolutional Neural Networks

This chapter provides a short introduction to the basics of convolutional neural networks on which later chapters build. Further the architecture of MobilenetV2 is discussed in more detail, as it is relevant for interpreting the features of the different network layers.

2.1 Introduction to CNNs

Convolutional neural networks have a structure specialized on processing images. The convolution operations allow to process images without flattening them to a vector, which is required for standard multi layer perceptrons (MLP) and results in the loss of the information encoded in the spatial position of the pixels. CNNs gained popularity in 2012 when AlexNet won the image net classification challenge [15]. Since then there has been a rapid development of new architectures. The idea is older though and a successful application was first published in 1998 by LeCun et al. [17], who applied CNNs to recognize handwritten numbers.

2.1.1 Convolution

The core technique employed by CNNs is convolution, where a filter kernel slides over the image and the dot product of the kernel with the underlying image patch is computed at each location. Strictly speaking the filter is reversed in the convolution operation and what is performed in CNNs is called cross correlation or the sliding dot product. In the following I will use the term convolution for this procedure as is usual in the deep learning literature. A gray scale image is a 2D matrix of gray values. Convolution for a 2D image I at position $I(i, j)$ is defined as follows [7, Chapter 9].

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) \cdot K(m,n) \tag{2.1}$$

K is a 2D filter kernel of size $m \times n$. The resulting convoluted image is also called a feature map. Different features can be detected depending on the design of the kernel. The convolution operation has properties which fit the requirements of machine learning systems well. Convolution implements sparse connections as the kernel is smaller than the input image. This allows for detecting small features in images independent of other image content. The same kernel is used for all positions

in the operation, making convolution translation equivariant. For example if the image is shifted to the side, the resulting featuremap will be shifted accordingly. Further it reduces the amount of parameters compared to a standard MLP, making the operation faster and more memory efficient [7, Chapter 9].

2.1.2 Convolutional Layers

In CNNs multiple convolutional filters are applied to the image in each layer. Usually a bias is added and a non-linear activation function is applied afterwards. Non linearity is a crucial part a network as it allows to approximate non-linear functions. Instead of predefining filters, the weights for the filters are initialized randomly and then learned in the training process. The CNN therefore learns features, which are helpful in classifying the training data [17]. Each input channel has a separate 2D kernel with individual weights, which are summed up over the channels, resulting in a 2D featuremap. A typical convolutional layer is defined by the number of filters, the kernel size, the stride and the padding mode. The Stride defines the step size with which the kernel is slided over the input [15]. Padding is used for handling the edges of images. Usually the image is padded with zeros when the kernel is not overlapping with the image. When no padding is used the resulting featuremap has smaller dimensions, like depicted in Figure 2.1.

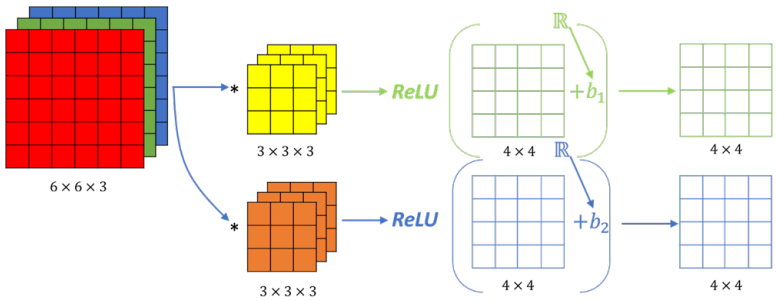


Figure 2.1: Example of a convolutional layer with two filters, no padding, a stride of one and Relu activation [2].

2.1.3 Pooling layers

Convolutional layers are often followed by pooling or subsampling layers which reduce the size of the featuremaps. In a pooling layer multiple values are collapsed into a single value by some function (Figure 2.2). Typical functions are the max or the mean of the values. The layer is further defined by the poolsize and the stride. Usually the poolsize is equal to the stride and the image is divided into separate sections, as depicted in Figure 2.2. If the poolsize is bigger than the stride the pools overlap [15]. Pooling has the role of making the featuremaps invariant to small translations. The assumption is that it does not matter exactly in which pixel position features are found, only that they are similarly spaced in relation to each other. Pooling can therefore be viewed as an infinitely strong prior, which enforces that the function learned by the layer must be invariant to small transformations. [7].

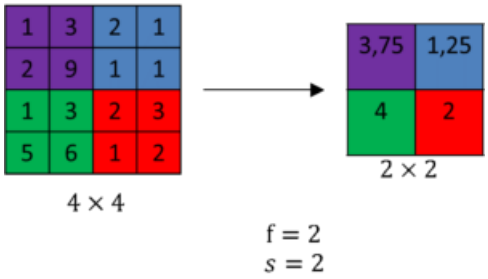


Figure 2.2: Example of mean pooling with a poolsize and a stride of two [3].

2.1.4 CNN Architecture

A classical CNN architecture is a feature extractor followed by a classifier. The feature extractor consists of convolutional and pooling layers and learns different features from the dataset. The first convolutional layer processes the input image. The following layers take the input of the previous layer and combine the earlier features to more complex ones. The output of the feature extractor is transformed into a vector using global average pooling, which collapses each feature map into a single value. The resulting vector is fed to the classifier. The classifier is a standard MLP with usually no or one hidden layer, which outputs a probability for each class using a softmax activation function [7].

2.2 MobileNetV2

MobileNet architectures are a series of CNNs, tailored for application on mobile devices and provide state of the art performance while focusing on low computational costs [10]. In particular I am going to visualize MobileNetV2 trained on the imagenet dataset. MobileNetV2 is a convenient choice for feature visualization as it is an architecture with good performance, while being computationally much more efficient than similar state of the art networks [24].

2.2.1 Depthwise Seperable Convolutions

MobileNetV2 employs depthwise separable convolutions in place of standard convolutions. A depth wise separable convolution splits the convolution process into two layers. The first performs a depthwise convolution, followed by a point wise convolution in the second layer. A depthwise convolution performs convolution with a single filter for each channel (Figure 2.3). The point wise convolution is responsible for combining the channels and building new features (Figure 2.4). Consider a standard convolutional layer of input size $h \times w \times d$ with d' convolution kernels of size $k \times k \times d$, where d' is the output depth. Depth wise convolution will convolute each channel of the input $h \times w \times d$ with a separate kernel of size $k \times k$, resulting in an output of size $h \times w \times d$. Point wise convolution then uses d' kernels of size $1 \times 1 \times d$ to produce an output of $h \times w \times d'$. For $k = 3$ depth wise separable convolution is around 8 to 9 times faster than standard convolution, with only a small reduction in accuracy [10].

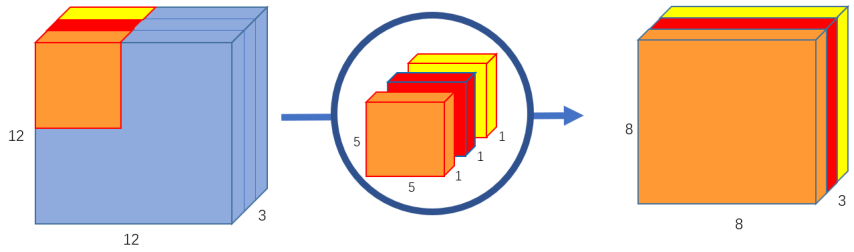


Figure 2.3: Example of depthwise convolution without padding [31].

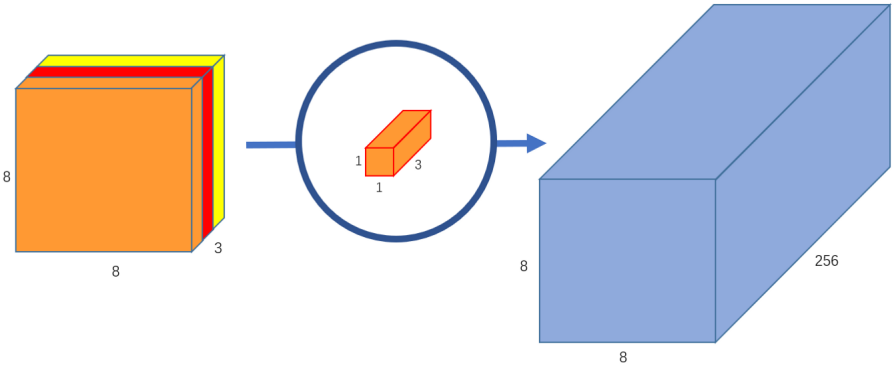


Figure 2.4: Example of pointwise convolution with 256 $1 \times 1 \times 3$ kernels. [31].

2.2.2 Inverted Residual Bottlenecks

The building blocks of MobileNetV2 are called inverted residual bottlenecks and use depthwise separable convolution in combination with residual bottlenecks. Residual connections were first introduced to deep learning in 2015 with the Res-Net architecture [12]. They are shortcut connections, which forward the output of a layer to a deeper layer without any transformations. Residual connections have a number of advantages over traditional architectures. They accelerate the training and allow for deeper networks in the first place. Furthermore they improve gradient flow, making vanishing gradients less of a problem [12]. Residual connections are usually used in bottleneck building blocks, where 1 x 1 convolutions are used to compress and expand the depth of the output and the residual connections allow direct information flow between the layers with expanded dimensionality. A featuremap can be seen as a manifold, which is assumed to be embedded in a subspace of lower dimensionality. As the information can be mapped to a low dimensional subspace the bottlenecks actually contain all the relevant information. Therefore the residual connections connect the bottlenecks in the MobileNetV2 architecture. The building blocks are therefore called Inverted Residual Bottlenecks (Figure 2.5). Sandler et al. [24] showed that residual connections between bottlenecks perform better than connections between expanded layers and are more memory efficient as well. Additionally no non-linearity is applied at the bottlenecks as they found that too much non-linearity can actually hurt the training process, by destroying information in the low dimensional subspaces. The non-linearity needed to approximate complex functions is applied in the expansion layers between the bottlenecks [24]. The detailed structure of the block is depicted in Figure 2.6.

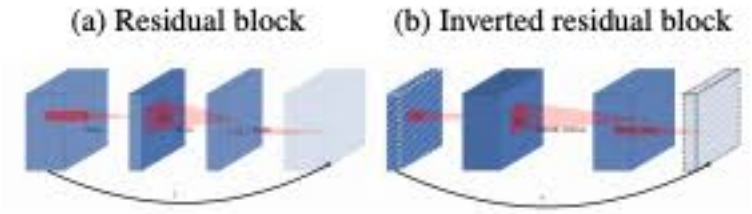


Figure 2.5: Residual Blocks compared Inverted Residual Blocks, diagonally hatched blocks do not use non-linearity, block thickness indicates the number of filters. Figure from Sandler et al. [24]

Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Figure 2.6: Inverted residual block transforming from k to k' channels with stride s and expansion factor t. Figure from Sandler et al. [24]

2.2.3 Model Architecture

The architecture of MobileNetV2 is shown in Figure 2.7. It contains a convolutional layer with 32 filters, followed by 19 residual blocks. Notably no pooling layers are included in the architecture. Instead a few convolutional layers with a stride bigger than two are used, which was found to perform equally well as traditional architectures employing pooling operations [28]. Relu6 is used as activation function. The output of the feature extractor is transformed using average pooling.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure 2.7: Mobile Net V2 Architecture. Each line describes a sequence of 1 or more identical layers, repeated n times. All layers in the same sequence have c output channels. The first layer of each sequence has stride s, while the other use stride 1. Spatial convolution uses 3×3 kernels. t is the expansion factor used in Figure 2.6. Figure from Sandler et al. [24]

Chapter 3

Visualizing Convolutional Neural Networks

3.1 Network Interpretability

The major problem of deep learning compared to traditional approaches of AI is that a neural network is a black box. While it is possible to read out the hidden representations of an input in a network, they are not interpretable to us. Therefore the decision process of a network is incomprehensible. This is problematic from a scientific viewpoint as well as for real world applications, where wrong decisions may have dire consequences. Think for example of self driving cars. If a stop sign is mistaken for a go sign an accident can be the consequence and it is important to figure out, where the mistake is and how to fix it. Furthermore there is the question of blame. Can you just entrust a black box with important decisions without knowing how it decides and then accept possible mistakes? Consider the example of a network which looks for cancer in MRI scans. It might perform better than a doctor, missing less cancers but when an error occurs you do not know why and have no one to take responsibility. We do not have to rely on hypothetical examples to understand the importance of network interpretability. In 2019 the apple credit card made news with a controversy about gender discrimination [33]. The credit limit was computed by a machine learning algorithm, which was trained with examples of granted credit dependent on a number of factors like annual income. The couple Jamie and David Hansson applied for a credit line and even though they share tax returns and bank accounts, David was granted 20 times more credit[8]. After posting the issue in social media plenty of other people reported similar stories. The power of neural networks is their ability to learn patterns from a data set and generalize these to unseen data. However there might be correlations in the training data, which the network was not intended to use for its decision process. It seems like the data used for training was biased and the algorithm learned that gender is a relevant factor for predicting credit. When David Hansson complained at Apple, the customer service was unable to explain the given credits. For an application like this, it is crucially important that decision processes are transparent. This story demonstrates how problematic employing AI system without transparency and proper checking for biases can be.

In the case of CNNs the network is able to learn meaningful features from a pixel based input and combine them to classify high level concepts like a cat or a certain

number [17]. If the data set provides a wide range of cat images, the network is able to classify images, which were not part of the training set, correctly because the learned features generalize to unseen cats. No explicit definition of what makes a cat a cat needs to be provided. We consider the feature of pointy ears an important part of a cat. The network learns its own features and how to combine them from the data set, in order to correctly classify objects in images. These features are implicit in the network parameters and we are unable to translate from network parameters to meaningful features which we could use to comprehend the network decision process. Feature visualization uses the information encoded in the network to produce images, which help us understand the nature of the features. We are still not able to translate from network parameters to high level concepts as the features might correspond to multiple ideas or no corresponding concept exists at all. However we can get an idea of the features by relating them to concepts we know. The task of illuminating the black box of a CNN consists of two sub tasks. The first is to understand what the features are. The second issue and arguably the harder problem is to understand how these features are combined to arrive at the output [21]. I will focus on the first of the two problems here. There are plenty of different techniques to make sense of the processes in a CNN, but providing a complete overview would exceed the scope of this thesis. I will discuss the approach of dataset examples and the phenomenon of adversarial examples. Dataset examples illustrate some difficulties in interpreting CNNs. Adversarial examples are closely related to and can be seen as a different facet of Feature Visualization. Then I will present results of my own implementation of feature visualization, replicating some of the techniques presented in the Distill blog post by Olah, Mordvintsev, and Schubert [21]. Further I will also use the lucid library developed by the authors of the blog post to illustrate methods which I did not implement myself.

3.2 Dataset Examples

A rather simple approach to gain insight on the learned features, is to feed an image through the network and extract the activation for each featuremap. When a featuremap has a high activation for an image, the image contains something the featuremap is trained to recognize. It can be interpreted as the featuremap looking for something in the image. However dataset examples do not tell us which features of the image actually activate the featuremap. Therefore dataset examples are best used in combination with other visualization techniques. In Figure 3.1 dataset examples can be seen in combination with activation patterns which show what part of the image caused the activation. These patterns were reconstructed from the images using a Deconvnet, a network which performs the operations of a CNN in reverse [34]. Dataset examples alongside feature visualizations are shown in Figure 3.2. The features in networks are not as clearly describable by high level concepts as we would like. There seem to be mixed features and the activation is not always caused by what we would assume from the dataset examples. For example in Figure 3.1 on the top right the activation patterns reveal that dogs alongside surrounding grass activate the featuremap. The visualizations in Figure 3.2 also reveal ambiguities which were not obvious from the dataset examples.

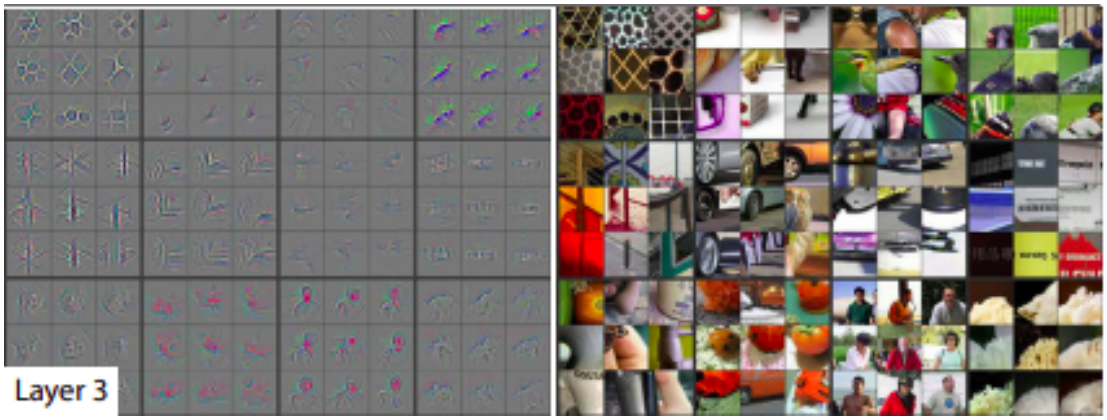


Figure 3.1: Examples from the validation set on the right, with the corresponding patters of activation on the left. Image from Zeiler and Fergus [34].

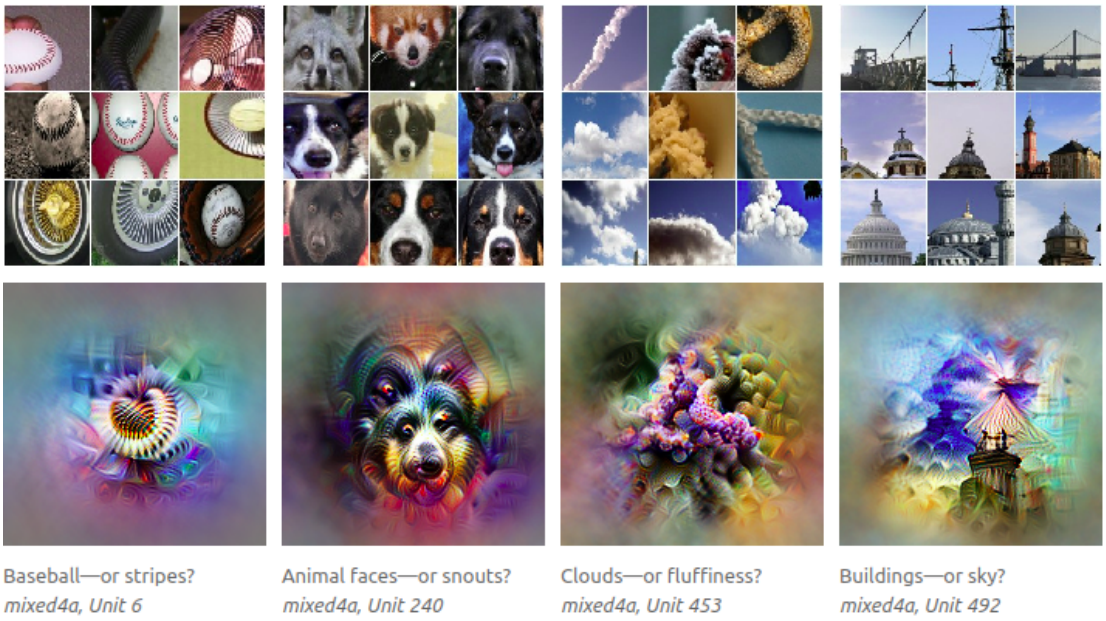


Figure 3.2: Dataset examples with corresponding feature visualization below. Image from Olah, Mordvintsev, and Schubert [21].

3.3 Adversarial Examples

Adversarial examples are images, which fool neural networks into false high confidence predictions. Although they do not depict interpretable features, they show important facets of featuremaps. An example created by Goodfellow, Shlens, and Szegedy [6] is depicted in Figure 3.3. Adding seemingly random noise to the picture of a panda causes the network to predict a wrong class with a high confidence, while for human perception the two images seem identical. The noise is optimized in a similar way, images are optimized for feature visualization. That tells us that CNNs seem to respond to high frequency patterns, which are not perceivable to humans. Adversarial examples are a general problem for CNNs and can be found over a variety of architectures [29].

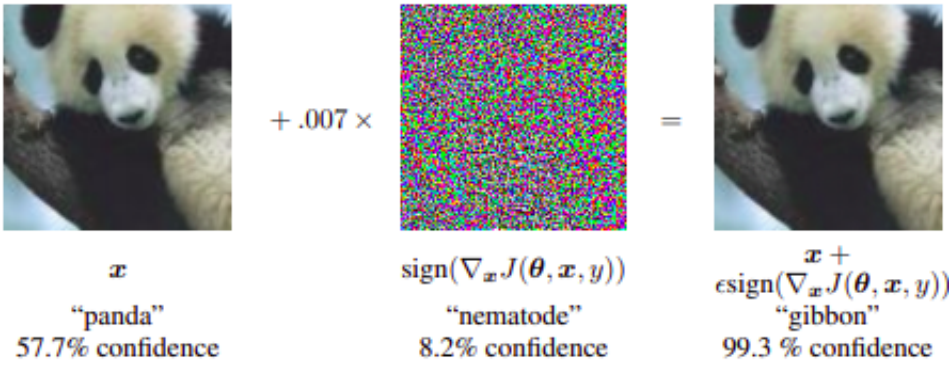


Figure 3.3: adversarial example

3.4 Feature Visualization

3.4.1 General Approach

Instead of using gradient descent on the network parameters to minimize a loss function, feature visualization applies the gradients to the input image. Starting with a randomly initialized image to avoid any bias, the image is optimized towards maximally activating a specified featuremap. As the goal is to maximise the average activation of the featuremap, the loss function to be minimized is the negative average activation (Equation 3.1). $F(\theta)^{M \times N}$ is the activation of the featuremap F with network input θ.

$$L(\theta) = -A(\theta) = -\frac{1}{MN} \sum_{n=0}^{N} \sum_{m=0}^{M} F(\theta)[m, n] \tag{3.1}$$

The hyperparameters required for the optimization algorithm are the learning rate γ and the number of optimization steps T. The average activation does seem to stabilize at around 500 optimization steps at the latest. I used $T = 512$ as a default. For the learning rate I found 0.05 to produce good results. As an optimizer I used Adam with default settings, also no clear advantage in image quality or optimization behavior was apparent compared to standard gradient descent. In the following I will present a number of different techniques which modify this general approach to achieve better results. These approaches are taken from the Distill Article on Feature Visualization Olah, Mordvintsev, and Schubert [21]. The techniques will be illustrated by examples of two different featuremaps from MobileNetV2. This does not show the performance of the techniques over a variety of featuremaps but makes the effects comparable to each other. A wider variety of featuremaps will be visualized in chapter 5.

3.4.2 Naive Optimization

Performing standard gradient ascent on a random input image does produce very noisy images. A high activation is achieved by exploiting high frequency patterns, which activate the featuremap. The results are very similar to the earlier discussed adversarial examples. Although these visualizations are not very helpful in understanding how the network classifies images, they tells us something about the nature of features learned in neural networks. The activations for these nonsensical images

are higher than the ones which show to us comprehensible features and should not be neglected. They show a different facet, may it be undesirable, of the learned feature.

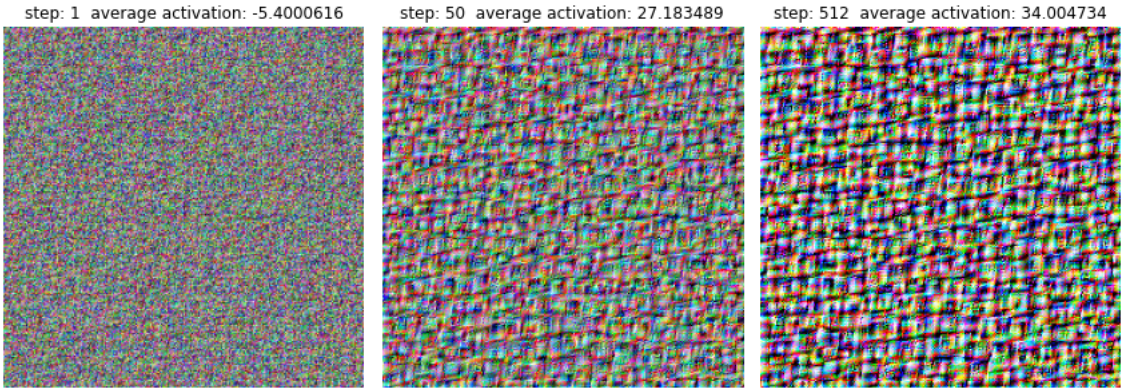


Figure 3.4: Block_4_add, unit 6, Image after step 1, 50 and 512

In Figure 3.4 results for an early layer are depicted. The image is very noisy although some structure becomes apparent. The featuremap seems to be activated by a rectangle like texture. The visualization for Block 12 is even more noisy (Figure 3.5). There does not seem to be an interpretable structure to the image.

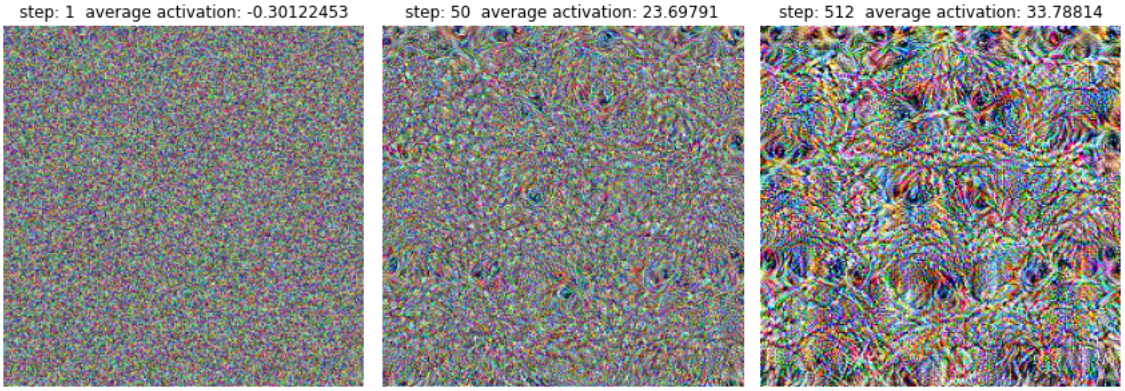


Figure 3.5: Block_12_add, unit 1, Image after step 1, 50 and 512

3.4.3 Regularization Penalties

In order to produce more interpretable feature visualizations, regularization techniques are needed to avoid converging on local minima, which are dominated by high frequency patterns. One way of suppressing noise in the optimization process is to introduce penalties to the loss function. Instead of just optimizing towards the maximum average activation, some penalty for noise is added to the loss function (3.2).

$$L(\theta) = -A(\theta) + \lambda \cdot P(\theta) \tag{3.2}$$

$A(\theta)$ is the average activation of the image θ and $P(\theta)$ the penalty. This penalty is scaled by a factor λ. If λ is too small the penalty has no meaningful impact. If λ

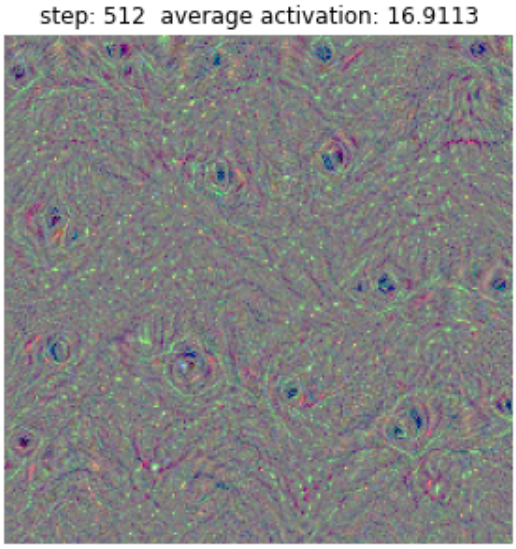
is too large the image is mostly optimized towards minimizing the penalty and not maximizing the average activation. In the following I will discuss l1 regularization and total variation as penalties.

L1 Regularization

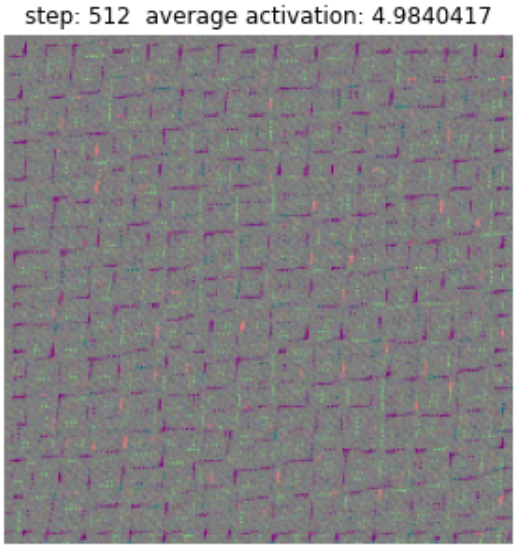
L1 Regularization is defined as the sum of absolute values of the image, resulting in the following loss function.

$$L(\theta) = -A(\theta) + \lambda \cdot \sum_{i,j} |p_{i,j}| \tag{3.3}$$

This loss function punishes large absolute pixel values, resulting in less noisy but very grayish image, as the image values are centered around zero. An example is depicted in Figure 3.6. The visualizations are still not a particular informative but they tell us that there are certain structures that take a larger part in the activation of the featuremap. As the penalty restricts the use of large pixel values in the image, most of the image is optimized towards a grayish muddy color. However the parts which have other colors seem to have a bigger impact on the activation, as they have bigger pixel values despite the penalty. For the featuremap in layer 4 vertical and horizontal edges seem to be important (Figure 3.6b). The activations of the featuremaps are notably lower. Therefore these images are not as much what the featuremap is looking for, compared to the unregulated examples, but the parts which are not gray show us what has the biggest impact on the activation. These images have some resemblance to the activations patterns produced by the earlier discussed Deconvnet approach (Figure 3.1). They are not produced for a specific input image, but can be interpreted in a similar way.



(a) Block 12 add, Unit 1



(b) Block 4 add, Unit 6

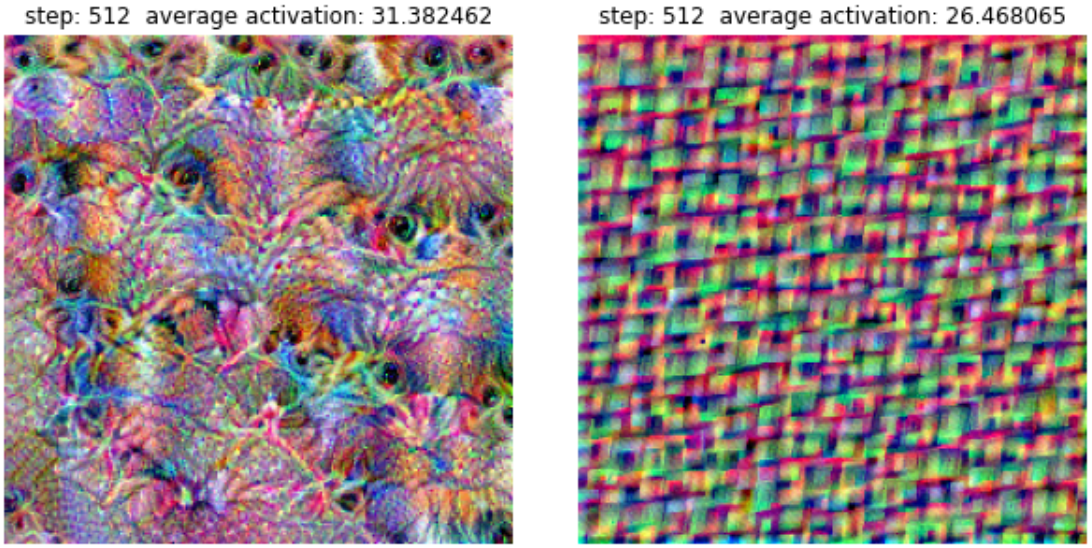
Figure 3.6: Example of L1 regularization with $\lambda = 0.0035$

Total Variation

Another regularization, which penalizes noise on a local level, is the total variation (TV) of the image. It is defined as the sum of the absolute differences for neighboring pixel-values in the image. The resulting loss function is

$$L(\theta) = -A(\theta) + \lambda \cdot \sum_{i,j} |p_{i+1,j} - p_{i,j}| + |p_{i,j+1} - p_{i,j}| \tag{3.4}$$

Using total variation blurs the images as it discourages large differences in value between neighboring pixels. Figure 3.7b shows a texture of what looks liked stacked rectangles. The edges are blurry, which is an undesirable side effect of the total variation penalty. The visualization depicted in figure 3.7a is still rather noisy but some clearer cut sections could be interpreted as eyes. This coincides with the image 3.6a, which revealed some not recognizable structures. Fine tuning λ can produce better results depending on the featuremap. When for example a featuremap is activated by sharp edges, increasing λ only worsens the result. For cases where homogeneous areas lead to high activations, a higher penalty can produce good results.



(a) Block 12 add, Unit 1 (b) Block 4 add, Unit 6

Figure 3.7: Example of a total variation penalty with $\lambda = 0.0003$

3.4.4 Blurring

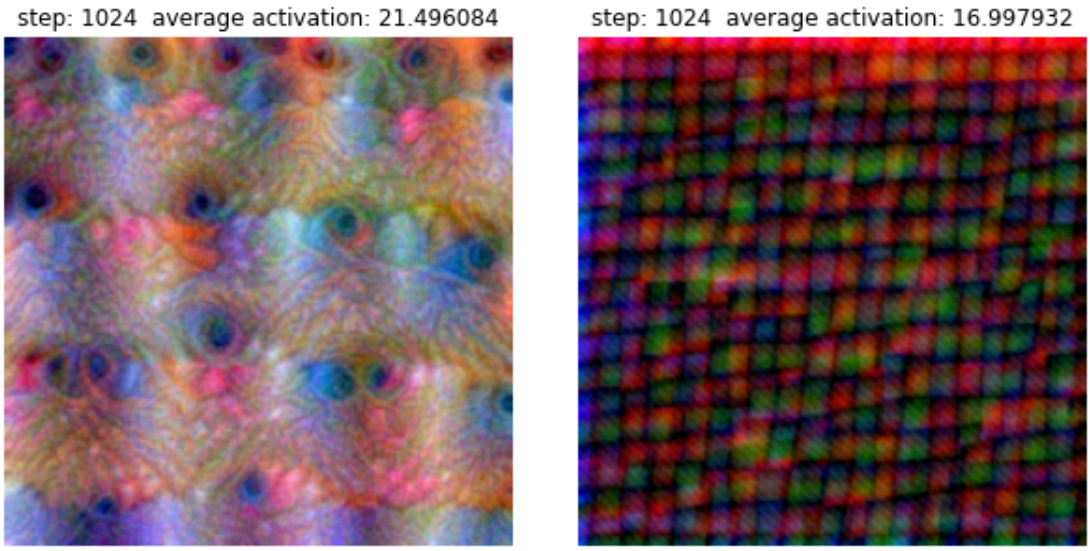
Another possibility of suppressing noise is to apply blurring filters to the input image [1]. Using filters slows down the convergence of the activation. The images were therefore optimized for 1024 steps instead.

Gaussian Filters

The standard blurring filter is a gaussian kernel, which weights pixels depending on their distance to the kernel center. Applying a gaussian kernel each optimization

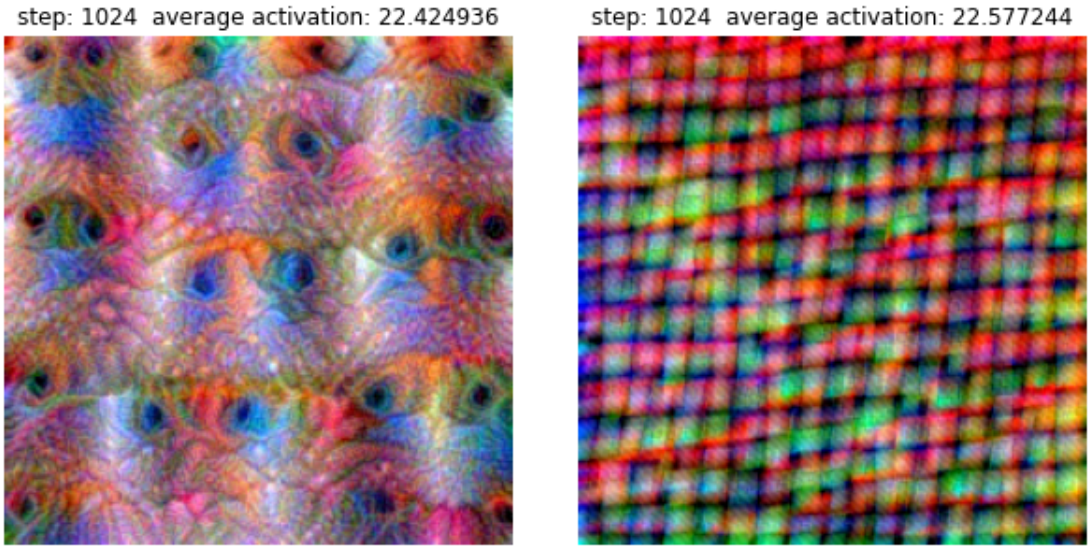
[1] I used the cv2 implementations of the gaussian and bilateral filter

step reduces noise but has the side effect of blurring edges (Figure 3.8).



(a) Block 12 add, Unit 1 (b) Block 4 add, Unit 6

Figure 3.8: Gaussian filter applied every step with kernel size 7, $\sigma = 0.7$



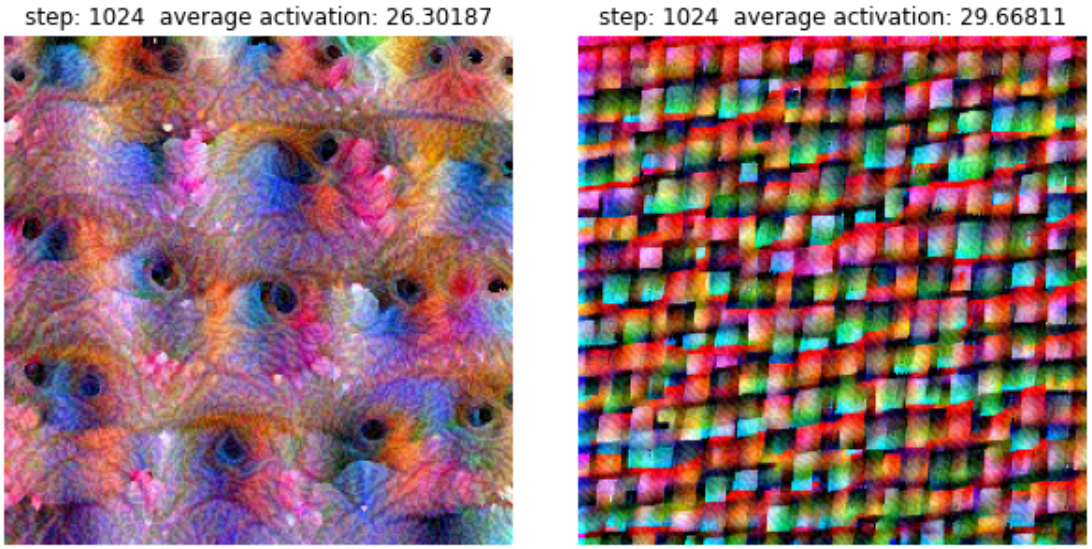
(a) Block 12 add, Unit 1 (b) Block 4 add, Unit 6

Figure 3.9: Gaussian filter applied every step with kernel size 7 and σ_t

A strategy to allow for clearer contours is to gradually decrease the σ of the kernel. This should in theory force the creation of bigger, vaguely defined patterns, which are then refined later in the optimization process. I used a schedule which distributes σ between 1.5 and 0.5 (Appendix .2). The results have clearer edges but are not dramatically better (Figure 3.9). More experimentation with the schedule could lead to better results.

Bilateral Filters

Bilateral Filters remove noise while maintaining relatively sharp edges. I found that blurring the image every second step produced the best results. The choice of the standard deviations is more tricky than for the gaussian filters. I did not find a schedule for σ which improved the results. The visualizations are a more clear cut than for the gaussian kernel (Figure 3.10). However bilateral filters bias by enforcing regions of homogeneous color.



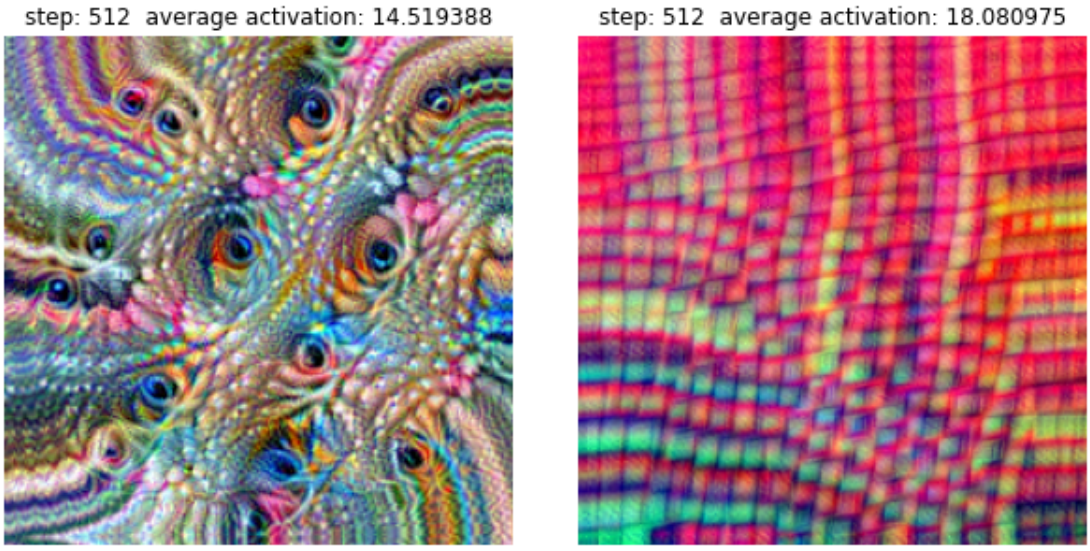
(a) Block 12 add, Unit 1

(b) Block 4 add, Unit 6

Figure 3.10: Bilateral filter applied every second step with kernel size 7, $\sigma_{color} = 0.8$, $\sigma_{spatial} = 1$

3.4.5 Transformation Robustness

The second family of regularization discussed by Olah, Mordvintsev, and Schubert [21] is transformation robustness. Transformation robustness means applying small stochastic transformation to the image before each optimization step, with the goal of finding an image which activates the featuremap even if it is slightly transformed. The transformations I use are jittering, scaling and rotating. The schedule for these transformations is oriented after the one used by Olah, Mordvintsev, and Schubert [21], but with overall smaller transformations (Appendix .1). I found that especially scaling disrupts the optimization process to much, when overused. The featuremaps seem to have a preferred size of the objects they recognize. Consider the examples of eyes. When zooming in on a part of the image, meaning a scaling with factor bigger than one, an eye might take a bigger part of the picture and be optimized in more detail, which produces a detailed intermediate image. Over time however, the eye gets optimized away and replaced by smaller eyes of the preferred size, partly resetting the optimization process. Visualizations using transformation robustness are depicted in Figure 3.11. In general the results are significantly better. In the case of Figure 3.11b the results seem to be a little worse compared to previous approaches. This is probably due to the rotation, which counteracts the preferred vertical and horizontal edges.



(a) Block 12 add, Unit 1 (b) Block 4 add, Unit 6

Figure 3.11: Optimized by transforming the image each step

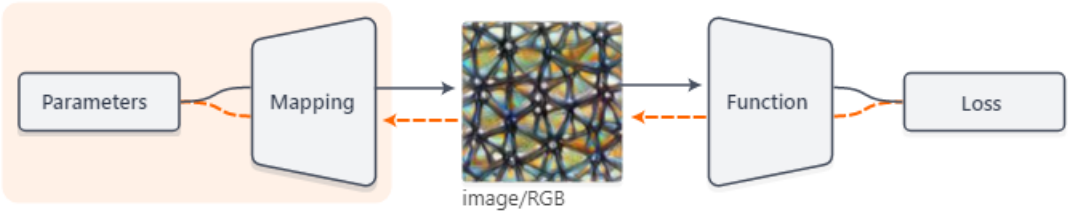
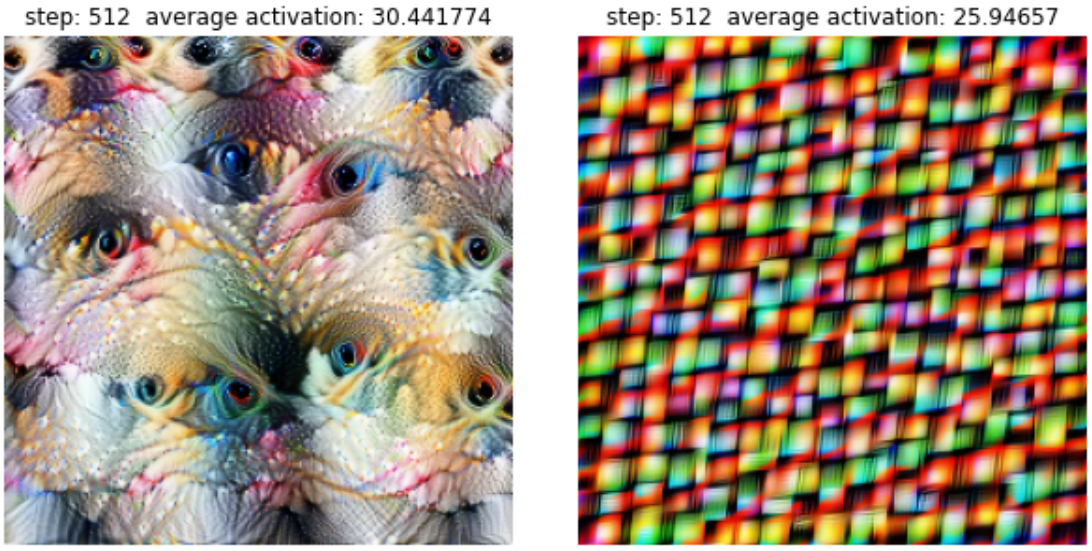


Figure 3.12: Images can be represented in a different parameterization. As long as the mapping to RGB parameterization is differentiable, it is possible to backpropagate through it. Image from Mordvintsev et al. [19]

3.4.6 Differentiable Parameterization

The third family of regularizations presented by Olah, Mordvintsev, and Schubert [21] is differentiable image parameterization (Figure 3.12). Changing the parameterization can make the optimization process easier and change the basins of attractions of the local minima. The minima themselves do not change but the right parameterization can enlarge the basins of desirable minima [19]. Olah, Mordvintsev, and Schubert [21] use a parameterization in Fourier space, which results in spatially decorrelated pixels. Further they measured the correlation of colors in the imagenet training set and decorrelated them using a Cholesky decomposition. For computing images using these techniques I used the lucid tensorflow library. In Figure 3.13 the visualizations with decorrelated colors and image parameterization in Fourier space are depicted. Lucid also uses transformation robustness as default (Appendix .1). The images are a lot less noisy and have clearer colors. In order to make the results comparable, visualizations using lucid with standard parameterization are shown in Figure 3.14. The results are roughly similar to mine, although lucid does a better job of avoiding edge artifacts due to transformation robustness. Further the images converge on a higher activation.

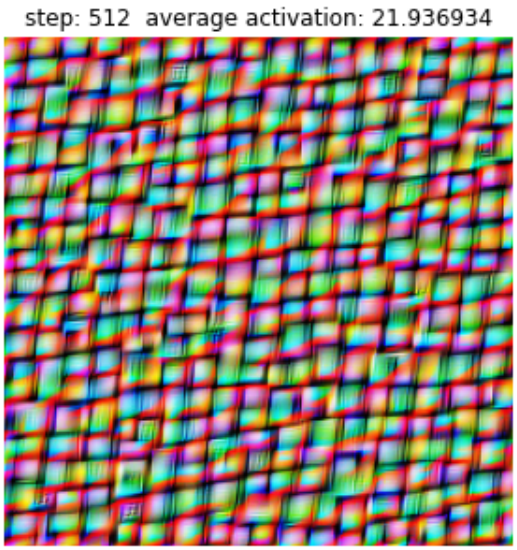


(a) Block 12 add, Unit 1 (b) Block 4 add, Unit 6

Figure 3.13: Optimized using lucid with default hyperparameters

In Figure 3.15 an example is shown, where differentiable image parameterization reveals facets, which my visualizations did not reveal. This happened especially for later layers, where visualizations are prone to get more noisy. The lucid image reveals dog faces, especially dog snouts. In Figure 3.15b some structures are visible, which kind of look like dog heads but it is only obvious when you know what you are looking for. The image is dominated by the background pattern, which is only slightly visible in the lucid visualization. Overall the lucid image converges on a higher activation while being less noisy.

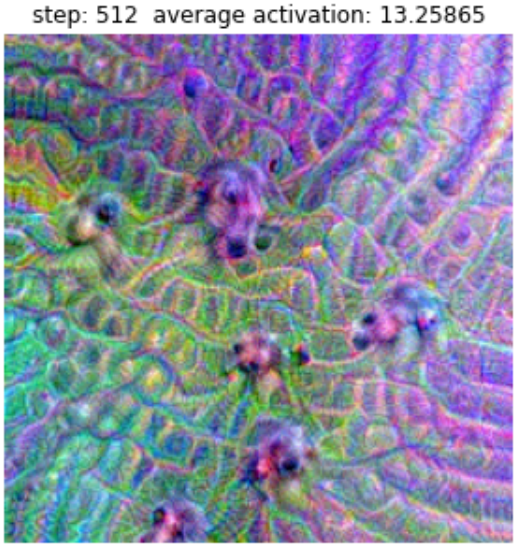
(a) Block 12 add, Unit 1



(b) Block 4 add, Unit 6

Figure 3.14: Optimized using lucid with default hyperparameters, without image parameterization in Fourier space and decorrelated colors

(a) lucid with differentiable parameterization



(b) my implementation

Figure 3.15: A visualization using lucid in comparison to my visualization using transformation robustness for Block 12 add, Unit 29

Chapter 4

Nudged Elastic Band Methods for Feature Visualization

The idea of using nudged elastic band (NEB) algorithms to visualize neural networks, originates from the paper "Essentially no Barriers in Neural Network Energy Landscape" by Draxler et al. [4], where a NEB algorithm is used to find minimum energy pathways in the loss surface of neural network parameter settings. I will first present the algorithm like it is used in the paper and then transfer it to feature visualization.

4.1 Minimum Energy Paths

A loss function can be considered a mapping of a multi-dimensional point, with each network parameter representing a coordinate, to a loss value. The resulting distribution of losses over the network parameter space is often referred to as the loss surface. When performing gradient descent to train a network, we navigate the parameter space towards a local minimum in the loss surface. Figure 4.1 shows an example for a 2D parameter space. The saddle point describes the maximum loss, which connects the two minima.

Draxler et al. [4] used a NEB algorithm to compute a continuous minimum energy path through the parameter space, which connects one local minimum to another. Each local minimum corresponds to a network trained to convergence from distinct, unrelated parameter initialisations. They showed, that unlike depicted in Figure 4.1, minima are not distinct valleys, which are connected by saddle points of high losses. Instead minima are connected to each other by pathways over which the loss remains essentially flat. Therefore minima should rather be considered points on a single connected manifold of low loss.

4.2 Nudged Elastic Band Algorithm

The goal of the NEB algorithm is to find a continuous path between two minima of the loss function, with the lowest maximum loss. The loss function is dependent on the training set, the network architecture and the network parameters θ. Keeping the training set and network architecture constant, the loss function can be written as $L(\theta)$. Each minimum is a parameter setting of a neural network trained to

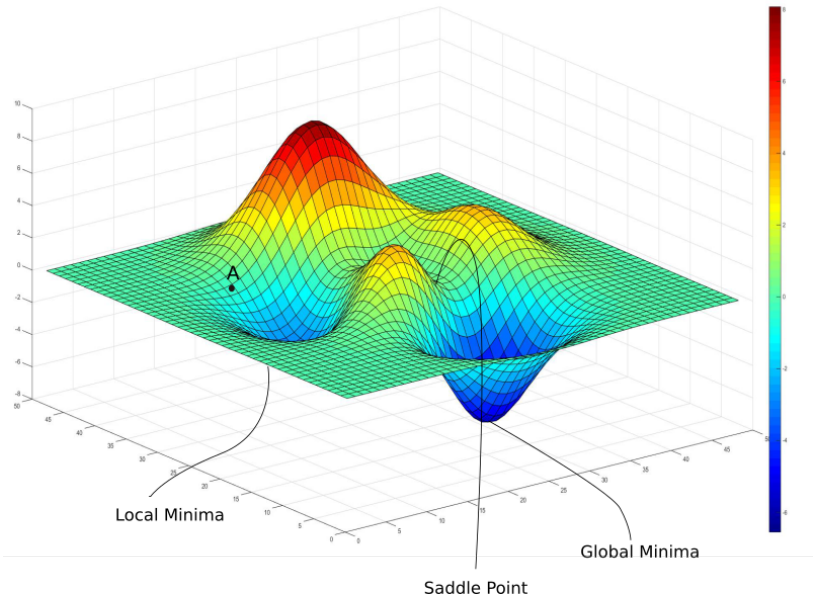


Figure 4.1: Example loss surface [13]

convergence. Formally the goal is to find a continuous path p^* which connects the minima θ_1 and θ_2 with the lowest maximum loss.

$$p(\theta_1, \theta_2)^* = \underset{p\ from\ \theta_1 to\ \theta_1}{\operatorname{argmin}} \left\{ \max_{\theta \in p} L(\theta) \right\} \tag{4.1}$$

A path between two minima is represented as a chain of pivots, which are connected by straight line segments. The NEB algorithm applies gradient forces perpendicular to each pivot, moving it in the direction of minimum loss. This is done until convergence. The point on the resulting path with the highest loss is called a saddle point as it is a saddle point of the loss function.

4.2.1 Mechanical Model

The NEB algorithm is inspired by a mechanical model where a number of $N + 2$ pivots are connected with each other by springs. The first and final pivots are fixed to the respective minima. The other pivots are optimized using gradient descent, minimizing the following energy function.

$$E(p) = \sum_{i=1}^{N} L(p_i) + \sum_{i=0}^{N} \frac{1}{2} k \|p_{i+1} - p_i\|^2 \tag{4.2}$$

Iteratively the sums of the loss and spring force are applied to the trainable pivots. The loss force points in the direction of the gradient, while the spring force pulls the pivots together and assures sufficient sampling density in high loss regions. The impact of the spring force is regulated by the spring stiffness constant k. If k is chosen too small the pivots would slide of the saddle point as the loss force pushes them in the direction of steepest loss decrease. As a result the pivots accumulate in areas of low loss, resulting in a path, which is densely sampled in areas of low loss but sparsely sampled in areas of high loss. As the goal of the algorithm is minimize the maximum loss, dense sampling in high loss regions is needed. If on the other

hand the spring constant is chosen too large, shorter paths with higher losses might be preferred, as the spring force increases quadratically with respect to the total path length.

4.2.2 Nudged Elastic Band

The nudged elastic band algorithm, derived from the above model, was originally presented by Jonsson, Mills, and Jacobsen [11]. I directly present the improved version by Henkelman and Jonsson [9] here. It improves on the mechanical model by nudging the forces so that the loss force only acts perpendicular and the spring force parallel to the path (Figure 4.2). The below equation describes the resulting force F_i for each pivot p_i.

$$F_i^{NEB} = F_i^L \mid_\perp + F_i^S \mid_\parallel \tag{4.3}$$

This way the spring force redistributes the pivots along the path, instead of generally shortening it and the loss force is kept from pushing multiple pivots together. The direction of the path at pivot p_i is defined by the local tangent $\hat{\tau}_i$. Consequently the forces are:

$$F_i^L \mid_\perp = -(\Delta L(p_i) - (\Delta L(p_i) \cdot \hat{\tau}_i)\hat{\tau}_i)$$
$$F_i^S \mid_\parallel = (F_i^S \cdot \hat{\tau}_i)\hat{\tau}_i$$

where the spring force is the difference in distance between the pivot and its neighbors, scaled by the spring constant k.

$$F_i^S = -k(\|p_i - p_{i-1}\| - \|p_{i+1} - p_i\|)$$

This formulation of the spring force enforces equal distances between pivots.

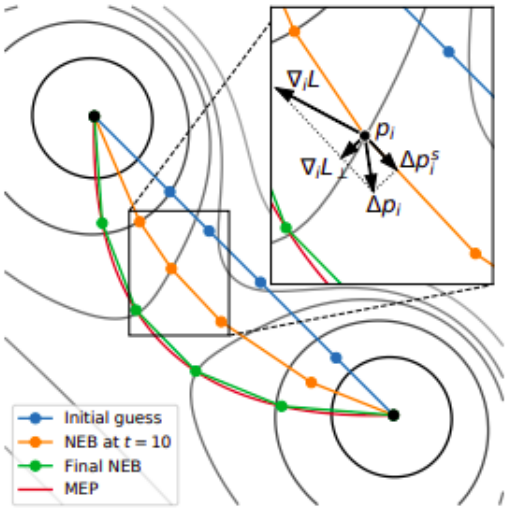


Figure 4.2: 2-dimensional loss surface with a minimum energy path and the NEB path at iteration 0, 10 and convergence. Δp_i is the update being applied to pivot p_i. The local tangent points to the neighboring pivot with higher energy. $\Delta_i L_\perp$ is the part of the loss force $\Delta_i L$ that acts perpendicular to the tangent. Δp_i^s is the spring force acting parallel to the tangent [4]

The local tangent is chosen to point in the direction of the neighboring pivot with the highest energy. N normalizes the length to one.

$$\hat{\tau}_i = N \begin{cases} p_{i+1} - p_i & if L(p_{i+1}) > L(p_{i-1}) \\ p_i - p_{i-1} & else \end{cases}$$

This particular choice of $\hat{\tau}$ prevents kinks in the path and ensures a good approximation near the saddle point [9].

The algorithm requires two hyper parameters, the spring stiffness k and the number of pivots N. The choice of k can be problematic. A too large k will lead to unstable optimization because the pivots are moved large distances each step, while a too small k requires a lot of iterations in order to redistribute the pivots. Draxler et al. [4] had trouble finding a k that works well for different loss surfaces and numbers of pivots and therefore resorted to so called string methods. Instead of applying the spring force, the pivots are redistributed evenly over the path each optimization step. The loss force still acts perpendicular to the path. Sheppard, Terrell, and Henkelman [25] found no significant differences in the final path and the convergence behavior between string methods and NEB methods using a spring force.

In Figure 4.3 the algorithm used by Draxler et al. is depicted. In addition to the number of pivots, it introduces the optimization steps T and the learning rate γ as hyperparameters. These are part of using gradient descent for optimization. In theory any other gradient optimization technique can be used.

Algorithm 1 NEB

Input: initial path $p^{(0)}$ with $N + 2$ pivots,
$p_0^{(0)} = \theta_1$ and $p_{N+1}^{(0)} = \theta_2$.
for t = $1, \ldots, T$ **do**
  Redistribute pivots on path $p^{(t-1)}$ and store as p.
  for i = $1, \ldots, N$ **do**
    Compute projected loss force $F_i = F_i^L\big|_\perp$.
    Store pivot $p_i^{(t)} = p_i + \gamma F_i$.
  end for
end for
return final path $p^{(T)}$

Figure 4.3: Pseudo code of the NEB algorithm used by Draxler et al. [4]

4.3 NEB for Feature Visualization

The NEB algorithm can be transferred to feature visualization in the same way optimizing network parameters can be transferred to optimizing input images. Instead of computing a path between two network parameter settings, a path between two optimized images can be computed. While the algorithm can be applied to the images in a similar way, the goals are a different. Draxler et al. [4] found a dense path of largest minimum activation from one minimum to another to proof that minima in the loss surface are connected manifolds and not distinct valleys. Here the goal is to create a sequence of images, which can be played as a video and provides an overview of what activates the featuremap.

The problem, that arises when using the NEB algorithm to compute pathways between optimized images, is that these images are only maxima of the NEB loss surface [1] if the same loss function was used for their optimization. This conflicts with connecting images which were optimized with different regularizations, in order to provide a wide overview over the featuremap's facets. It is still possible to apply NEB but the images are optimized to the loss function used in the algorithm. With respect to the starting images the pivots will therefore be over-optimized. Just applying gradient ascent to input images leads to results comparable to the adversarial examples, as shown earlier. Therefore the intermediate images will be optimized towards high frequency patterns if standard gradient ascent is used. As techniques in the optimization process of the starting images were applied to suppress these high frequency patterns, this is not desirable. Similarly a method for suppressing high frequency patterns in the NEB algorithm needs to be introduced. The most successful technique was transformation robustness, where small spatial transformations were applied to the image each step. However transformation robustness can not be included in the NEB algorithm as the chain of pivots is completely destroyed when the pivots are rotated or otherwise transformed. Although the perceived distance might be small after a transformation, the same does not hold for the euclidean distance, which defines the position of the image in parameter space afterwards. Therefore only regularization methods which change the gradient can be used in the NEB algorithm. I found the best performing methods of these to be the total variation penalty. Using a penalty does not solve the problem of optimizing on a different loss surface but it reduces the impact of the over-optimization on image quality. This is of course just a trade off between high frequency patterns on the one hand and blurring on the other. However for perceived image quality, blurring is the better choice. I also experimented with blurring the gradient before making it perpendicular to the local tangent. The results were noisier compared to the total variation penalty and the optimization process was slower. Using gradient blurring on top of the total variation penalty did not improve the results.

Sampling density in this application has the role of being dense enough that the transition from one image to another looks fluid. Draxler et al. [4] used a wrapper algorithm called AutoNEB which inserts extra pivots when the sampling density is too low. For guaranteeing a visually smooth path, I used a wrapper algorithm, which inserts additional pivots after one NEB run.

4.3.1 Formalization

The parameters of the two images, for which we compute the path, are named θ_1 and θ_2. Instead of finding a continuous path p^* which connects the minima θ_1 and θ_2 with the lowest maximum loss, the goal is to find a perceived continuous path which connects the maxima θ_1 and θ_2 with the highest minimum objective value $O(p_i)$.

$$p(\theta_1, \theta_2)^* = \underset{p\ from\ \theta_1 to\ \theta_1}{\operatorname{argmax}} \left\{ \min_{\theta \in p} O(\theta) \right\} \tag{4.4}$$

The objective function is the average activation $A(\theta)$ of the image θ minus a

[1] Even though the NEB algorithm for feature visualization technically maximizes an objective function, I will continue using the term loss surface

total variation penalty $TV(\theta)$ scaled by the factor λ.

$$O(\theta) = A(\theta) - \lambda \cdot TV(\theta)$$

Consequently the loss force now is the gradient force maximizing the objective function.

$$F_i^O \mid_\perp = \Delta O(p_i) - (\Delta O(p_i) \cdot \hat{\tau}_i)\hat{\tau}_i$$

As the pivots will be redistributed after every optimization step, the spring force is irrelevant. The tangent is accordingly chosen to point in the direction of the neighbor with the lowest objective value.

$$\hat{\tau}_i = N \begin{cases} p_{i+1} - p_i & if O(p_{i+1}) < O(p_{i-1}) \\ p_i - p_{i-1} & else \end{cases}$$

As over-optimization is an issue, running the algorithm to convergence results in undesirable images. Instead of performing T gradient descent steps, the algorithm is run to a maximum of T steps or until the pivot with the lowest activation has an activation as least as high as one of the two starting images. The other hyperparameters remain the same with N as the number of pivots and γ as the learning rate. Additionally a scaling factor λ for the total variation penalty is introduced. Algorithm 1 shows the modified NEB algorithm.

Algorithm 1: NEB for Feature Visualization

input : initial path $p^{(0)}$ with $N + 2$ pivots
$p_0^{(0)} = \theta_1$ and $p_{N+1}^{(0)} = \theta_2$;
$t = 0$;
while $t < T and \min_{\theta \in p} A(\theta) < (A(\theta_1)\ and\ A(\theta_2))$ **do**
  for $i = 1, ..., N$ **do**
    compute projected gradient force $F_i = F_i^O \mid_\perp$;
    update pivots $p_i^{(t)} = p_i + \gamma F_i$;
  end
  Redistribute pivots on path p^t;
end
return: final path p^t

The wrapper algorithm takes a list of optimized images as an input and computes one path connecting them according to the lists order (Algorithm 2). Producing a path between multiple images, which can be played as video with a frame rate of 30, requires a lot of pivots and is therefore computationally very expensive. In order to save computational resources the NEB algorithm is run with a small number of pivots first. Afterwards a number of linear interpolations are computed between neighboring pivots. If one of the interpolations has an activation lower than one of the neighboring original pivots, NEB is run again using the interpolations as additional pivots for a maximum number of steps T'. Otherwise the interpolations have a high enough activation and can be used as frames for the video. The algorithm introduces the additional hyperparameters I as the number of interpolations performed between neighboring pivots and T' as the maximum number of steps for the

NEB iteration with the interpolations as path. I used $T' = 100$ in order to avoid long run times when the algorithm runs with the additional pivots and the other termination condition is not reached.

Algorithm 2: NEB Wrapper

input : list of M starting images θ_1,...,θ_n
  $p^* = []$;
for *m=1,...,M* **do**
  p_m^0 = initialize path(θ_n, θ_{m+1}, N+2);
  p_m = NEB(p_m^0, T);
  p_m = interpolate pivots(p_m, I);
  if *activations of at least one interpolation smaller than activations of neighboring pivots* **then**
    p_m = NEB(p_m, T');
  end
  p^*.append(p_m);
end
return: final path p^*

In order to combine feature visualization with the NEB algorithm, I implemented another wrapper, which optimizes a number of images according to different regularization techniques and then produces a video with the computed NEB path and a graph of the image activations as frames.

4.4 Results

In the following I will present results and discuss problems for the NEB algorithm and the wrapper function.

4.4.1 NEB

The following examples were created using a learning rate of 0.002 and 28 intermediate images. In Figure 4.4 the activations of the pivots before optimization can be seen. Naturally the activations near to the starting images are high and the lowest activation is found somewhere in the middle. The algorithm runs until the activation of the pivot with the lowest activation is at least as high as the one of the staring images activations. This leads to an over-optimization of the pivots near the starting images, resulting in noisy images. A total variation penalty does not avoid the problem but restricts the over-optimization to a certain extent (Figure 4.5a). While the activations are only a little lower, the images are less noisy (Figure 4.5c). The other pivot which is prone to develop noise is the one with minimum activation. In Figure 4.5b this pivot is depicted for the different total variation penalties. Here the penalty also successfully reduces noise.

In Figure 4.6 another example is depicted. The first starting image was optimized without any regularization, the second with a TV penalty. It shows a similar pattern than the previous example. Further it showcases the negative side effects of the TV regularization when used with starting images which are more noisy to begin with. In Figure 4.6b the blurring seems excessive for the two higher penalties.

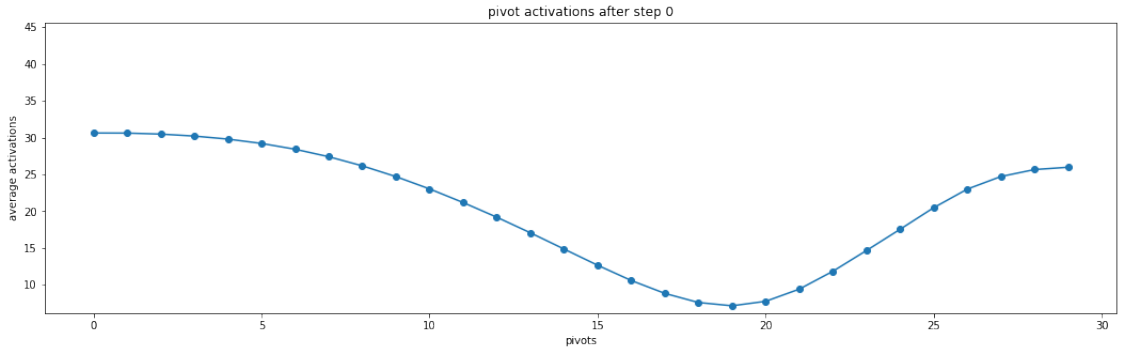
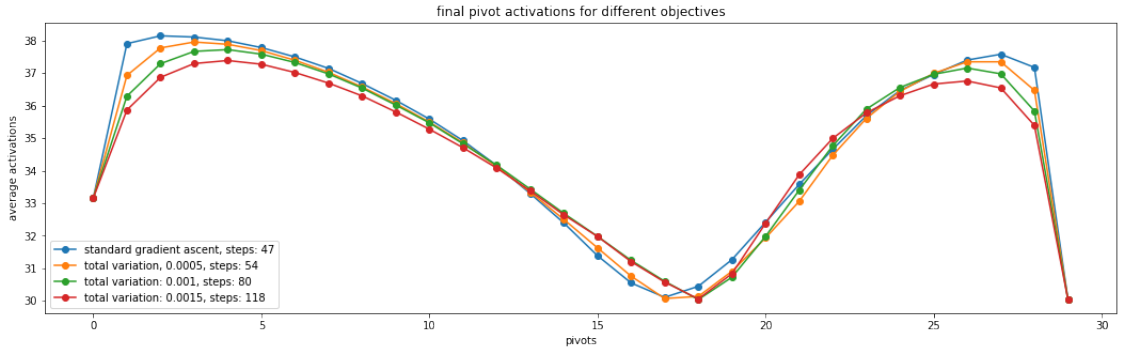
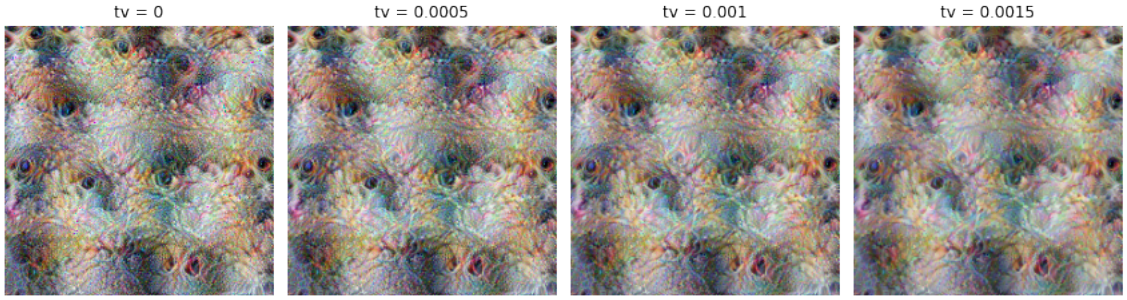


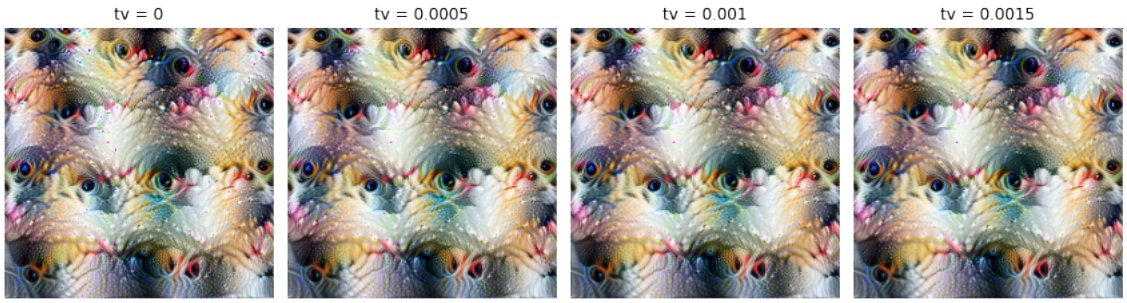
Figure 4.4: step 0



(a) Final activations for different TV penalties, the number of steps required rises with the scaling of the penalty

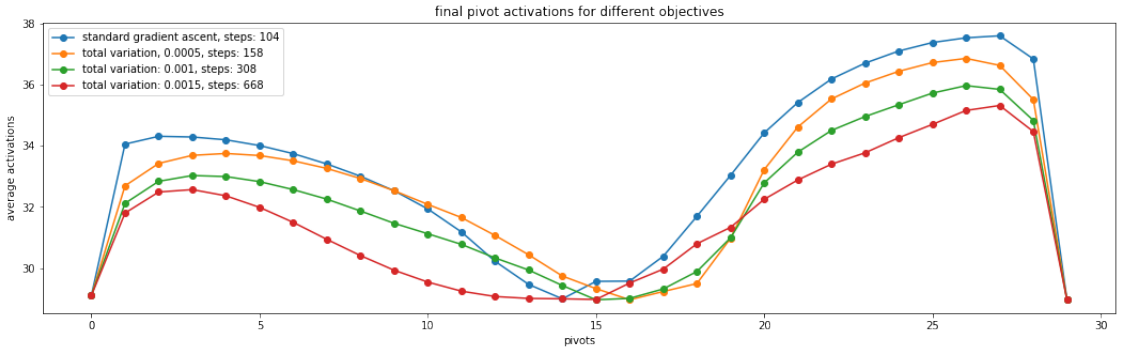


(b) The pivot with the minimum activation for the different penalties

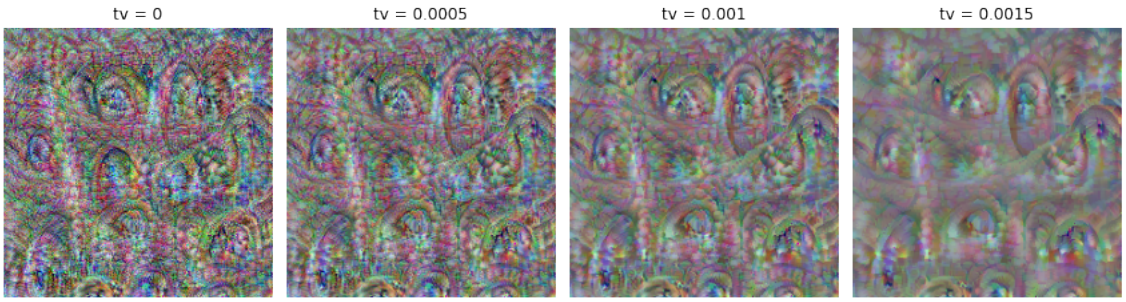


(c) The second to last pivot for the different penalties

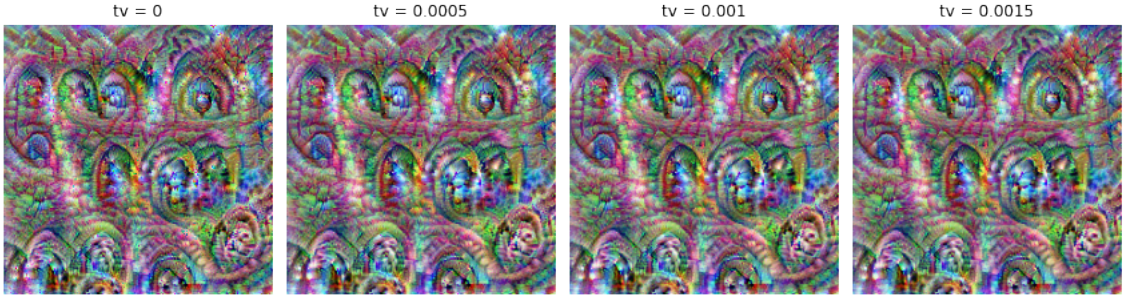
Figure 4.5: Example optimized towards activating layer 12 add unit 1 for different total variation penalties. The first starting images was optimized using no regularization techniques. The second starting image was computed using the lucid library with default parameters. Full video for $TV = 0.001$ can be found here



(a) Final activations for different TV penalties



(b) The pivot with the minimum activation (14) for the different penalties



(c) The second to last pivot for the different penalties

Figure 4.6: Example optimized towards activating layer 12 add unit 32, for different total variation penalties. The first starting image was optimized using no regularization techniques. The second starting image was computed with a TV penalty of 0.0003. Full video for $TV = 0.001$ can be found here

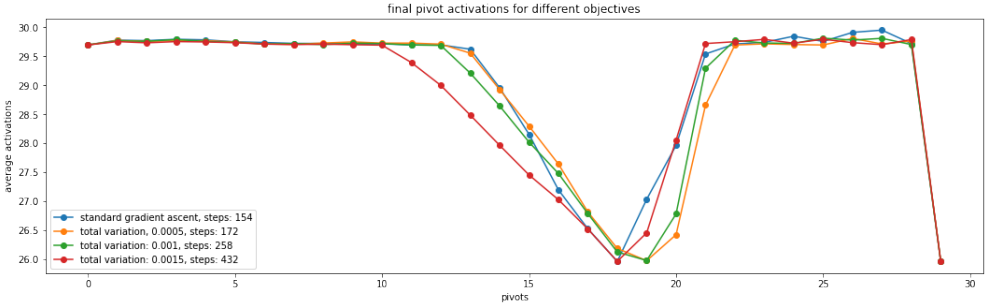


Figure 4.7: Final activations for different TV penalties with freezing pivots enabled

Freezing Pivots

One straight forward approach to avoid over-optimization would be to freeze the pivots, which have an activation higher than the ones of both starting images. However it turns out that freezing pivots seems to restrict the optimization of the other pivots on the chain. While it works fine in a lot of cases, in others the unfrozen pivots tend to exploit high frequency patterns in order to achieve a higher activation. In some other cases the restrictions seem to completely prevent the pivot with the lowest activation from reaching the threshold. In Figure 4.7 an example of final activations can be found. As the threshold for freezing is the maximum activation of the two starting images, over-optimization is only avoided for the pivots near the starting image with the higher activation. I chose not to include freezing pivots in the default setting of the algorithm as the drawbacks seemed to outweigh the benefits in more cases. The method could work better when having different freezing thresholds for the pivots depending on their position on the curve or using adaptive learning rates, which grow smaller the higher the activation gets instead.

4.4.2 NEB Wrapper

For the following examples I used the hyper parameter setting $\gamma = 0.002$, $T = 500$, $T' = 100$, $N = 18$ and $I = 3$, resulting in 83 intermediate images, between each starting image. Six starting images were optimized with the hyperparameters presented in chapter 3, resulting in a total of 420 frames played at a frame rate of 30. Example starting images are depicted in Figure 4.8. Final activations of the computed images are shown in Figure 4.9 and the resulting video is of rather good quality (Video).

In other cases a total variation penalty is counterproductive and the average activation of the intermediate images is actually optimized downwards (Figure 4.10), resulting in video full of artifacts (Video). For this featuremap the goal of minimizing the total variation conflicts with the one of maximizing the average activation. Partly this is due to featuremap being activated by images with a rather high total variation. The second issue is the overall small activation of the featuremap (Figure 4.10), making the regularization part of the objective function more impactful. Even though the total variation was scaled with a rather small λ of 0.0005, the penalty completely dominated the objective function. Using gradient descent without a total variation regularization did not produce good results either (Video). I found similar behavior for a few featuremaps in block 1, where the visualizations are full of high frequency edges. In other cases some intermediate images fail to reach the activation of the neighboring starting images and are heavily blurred. I failed to find a scaling for the total variation penalty, which produces good results over all featuremaps. Dynamically setting λ in relation to the starting images activation should improve the results. However this does not change the fact that a total variation penalty is an inadequate choice for certain featuremaps.

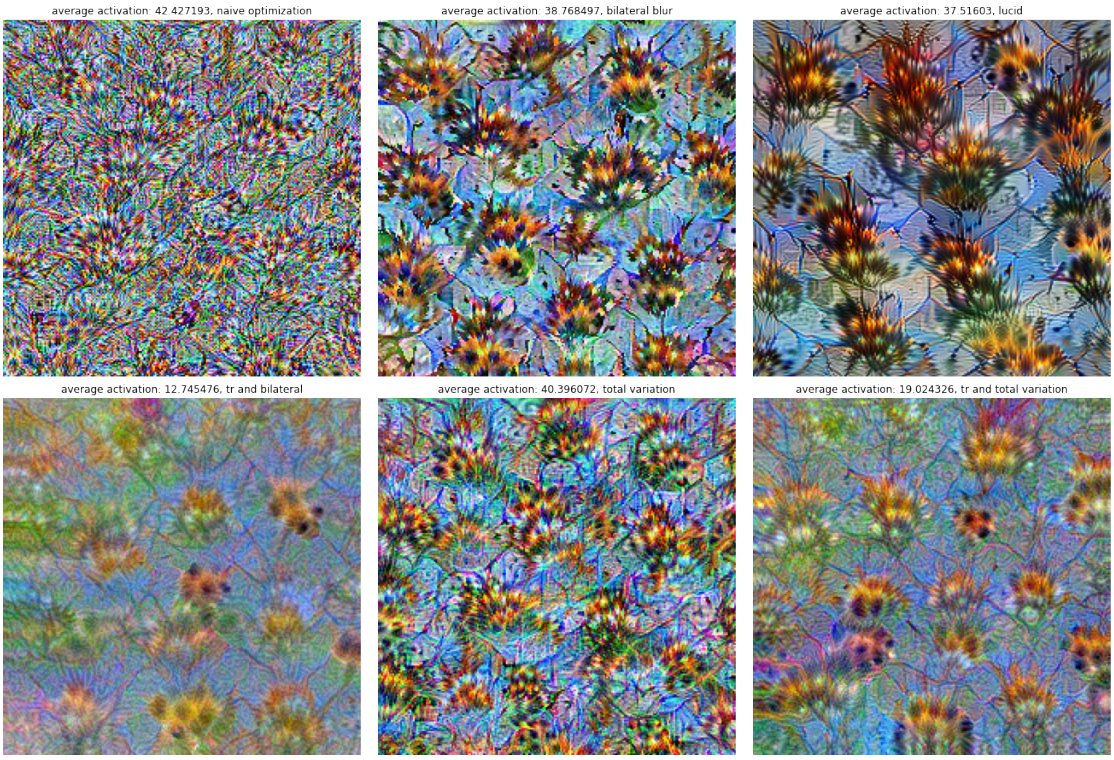


Figure 4.8: Starting images of the NEB path for layer 12 add, unit 33. The video can be found here

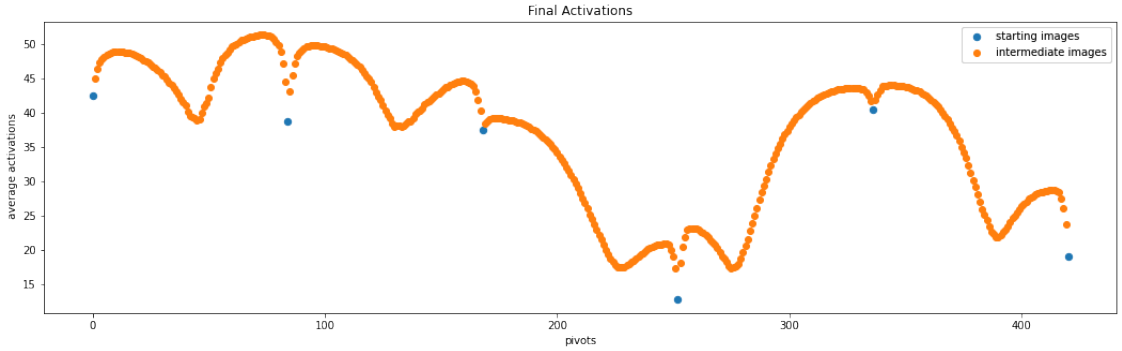


Figure 4.9: Layer 12 add, unit 33, activations of the final images

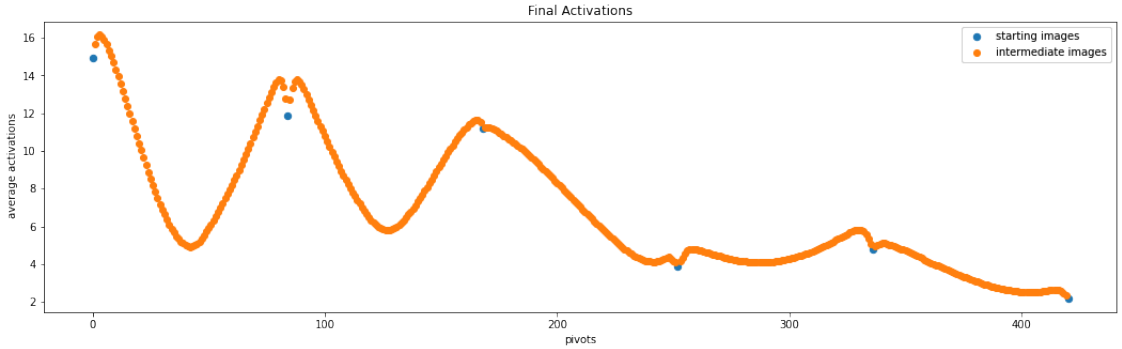


Figure 4.10: Layer 2 add, unit 6, activations of the final images

Chapter 5

Visualizing MobileNetV2

5.1 Different Layers of the Building Block

Like previously discussed the building blocks of MobileNetV2 consist of an expansion layer, a depthwise layer and a projection layer. The projection layer is the bottleneck with the lowest dimensionality. Sandler et al. [24] hypothesized that the meaningful information is encoded in the bottleneck. Therefore it is unsurprising that the features of the projection layers are the clearest and most informative. The depthwise layers just filter information from the expansion layer and do not combine information from different channels. Their visualizations do not reveal interesting features and range from pretty noisy to completely unintelligible. The expansion layers project the information from the bottleneck into a high dimensional space and their visualizations are of better quality compared to the depth wise layers. However the information they show is mostly redundant and can be found in similar or compressed forms in the projection layers. Features in the following projections layers always seem to combine a few different features from the expansion layers, which makes sense intuitively, considering the purpose of bottlenecks. Investigating the expansion layers more closely could be helpful in understanding how features evolve over the layers. This would however exceed the scope of this thesis and I will focus on just visualizing the features. Consequently the following visualizations will be limited to the projection layers.

5.2 Residual Connections

The MobilenetV2 architecture contains multiple identical blocks in a row, each with the same number of featuremaps (Figure 2.7). The bottlenecks of these blocks are connected by residual connections and therefore one would expect theses features to be somewhat similar or a combination of the two different input layers. In Figure 5.1 an example of the visualizations of the same unit over multiple layers is shown. There seems to be a common theme of fur, eyes and dog snouts as well as a preferred mix of colors. The visualization of block 7 add shows how the two previous features are merged into one. For block 9 add the image seems to to be influenced mostly by block 8 add. A similar behavior over other units and blocks was found, although seemingly unrelated themes can get mixed together. In Figure 5.2 there seems to be a theme of arcs into which some cat features get mixed in. This showcases that

visualizations just depict a facet of the underlying feature, especially in the add layers.

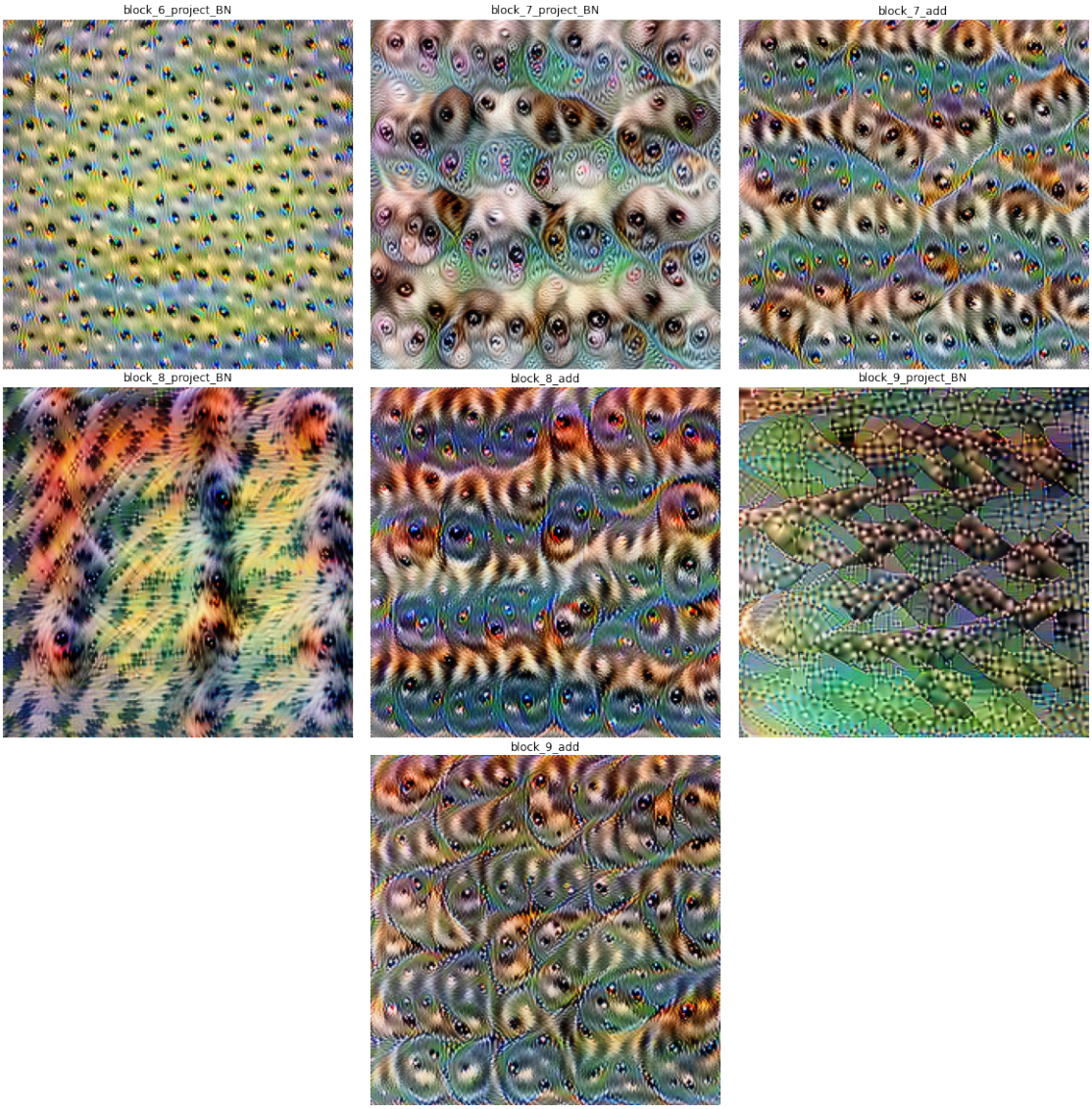


Figure 5.1: Visualizations for unit 31 over the different projection and add layers from block 6 to 9. The add layers take the preceding projection layer and the last layer of the previous block as input

5.3 Different Types of Features

A common theme of feature visualizations is that simple features like edges are detected early in the network and more complex features are found in later layers. The more complex features also tend to be more multi faceted and sometimes mix seemingly unrelated ideas together. Another stable is that in later layers the features get more noisy and are generally harder to visualize until they become unintelligible towards the end of the feature extractor. This is caused by the usual problems with gradient flow in very deep neural networks [21]. In the following I will give an overview of this progression of features in MobilenetV2. Here only the visualizations

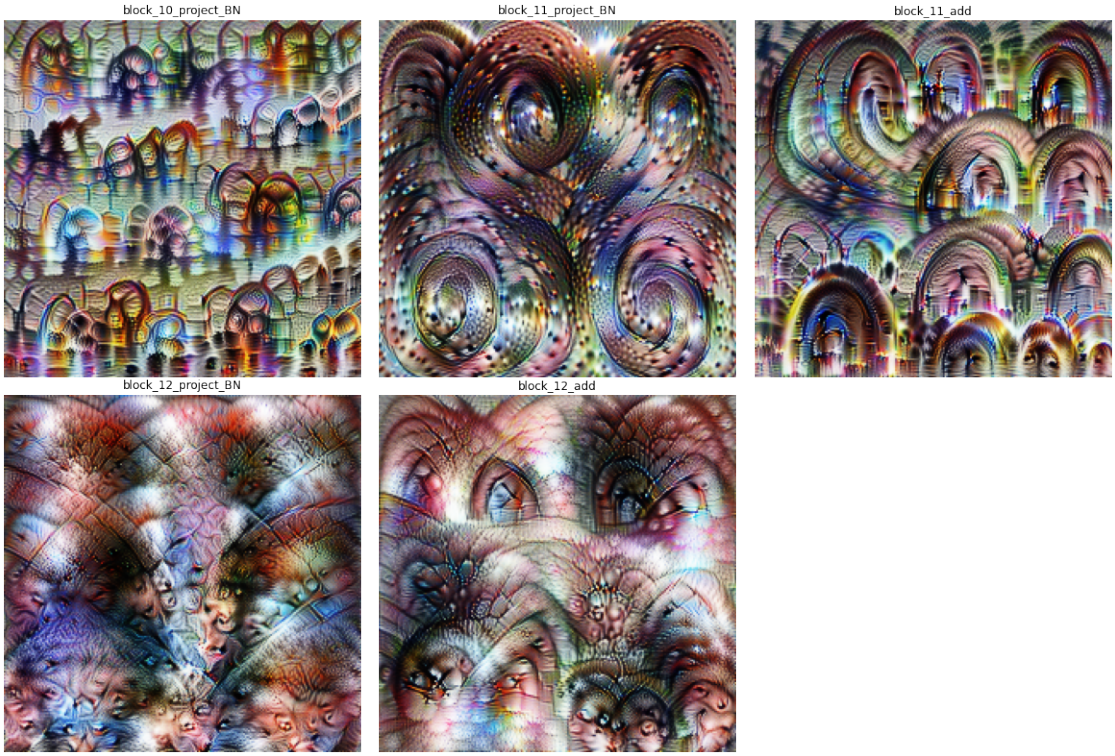


Figure 5.2: Visualizations for unit 31 over the different projection and add layers from block 10 to 12

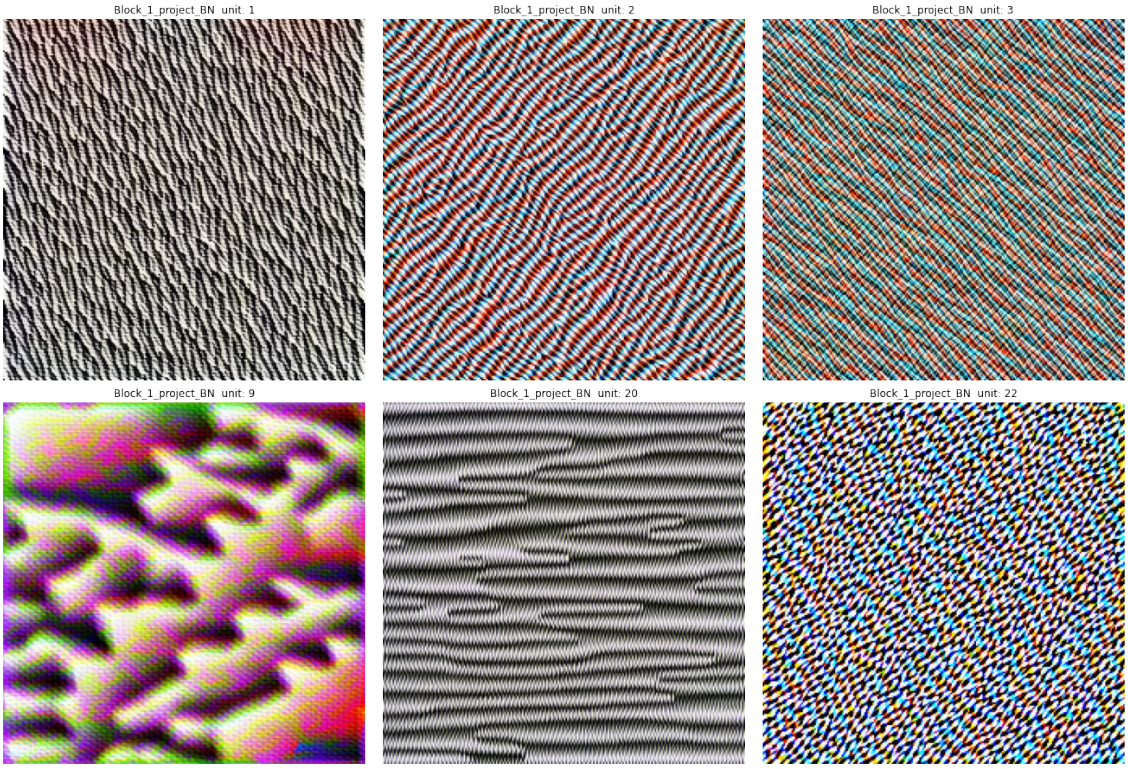


Figure 5.3: Edges. NEB visualizations can be found here

using lucid will be depicted together with links to videos produced by the NEB algorithm, which showcase a wider variety of visualizations.

In Figure 5.3 features from block 1 are depicted. The feature maps seem to be activated by differently angled edges of different sizes. The visualizations of block 3 to 5 show textures (Figure 5.4). Blocks 6 to 9 show patterns, which already contain more complex objects like eyes (Figure 5.5). In Figure 5.6 parts of objects can be seen. The themes of features get really distinct and it is easy to see to what kind of images they respond. Around block 12 the features correspond best to our high level concepts features like flowers or birds. Still most visualizations only show parts of these objects. More complete objects can be guessed at in the following layers (Figure 5.7). The visualizations are however much more noisy and different concepts get mixed together more. Here the videos become a lot more noisy as only the starting images optimized by lucid are of relatively good quality. From block 15 onward most visualizations are incomprehensible. The progression of the features is of course fluid and the categories are just for the purpose of structuring the results. Features of one layer, especially later ones, are not completely assignable to one category. Overall the features of MobileNetV2 are similar to the one of GoogLeNet by Olah, Mordvintsev, and Schubert [21] [1].

The most high level concepts are found visualizations of the different output classes. As the last layer, they are the most difficult to visualize and the images tend to be really noisy. For optimization class logits, the absolute class values before applying softmax, are used. The softmax probabilities can be maximized by minimizing the probabilities of other classes. Therefore the softmax layer does not necessarily produce visualizations that depict class related features. Visualizations of a few classes are depicted in Figure 5.8. It is possible to see a few elements of the class in the images but the overall quality is pretty bad. For the classes I also handpicked the best examples to show somewhat informative images. The other visualizations are also handpicked but the overall quality of the images over all the different channels is more consistent [2]. Classes can be more successfully visualized using example images as priors. These visualizations are biased by the choice of the prior but still can reveal different facets of the features by using a variety of priors and regularization techniques [18, 20].

[1] see https://distill.pub/2017/feature-visualization/appendix/ for GoogLeNet visualizations

[2] Visualizations of all the units of the projection layers can be found under https://github.com/Niioon/CNN_Visualizations/tree/master/mobilenet_v2_visualizations/images

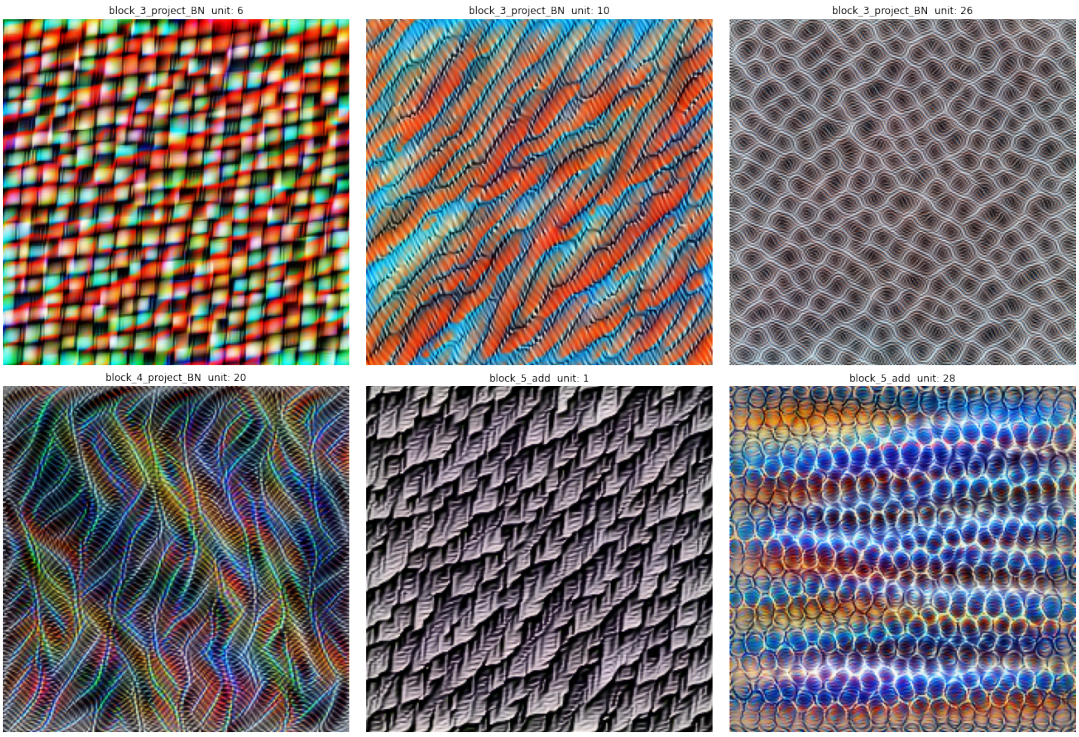


Figure 5.4: Textures. NEB visualizations can be found here

Figure 5.5: Patterns. NEB visualizations can be found here

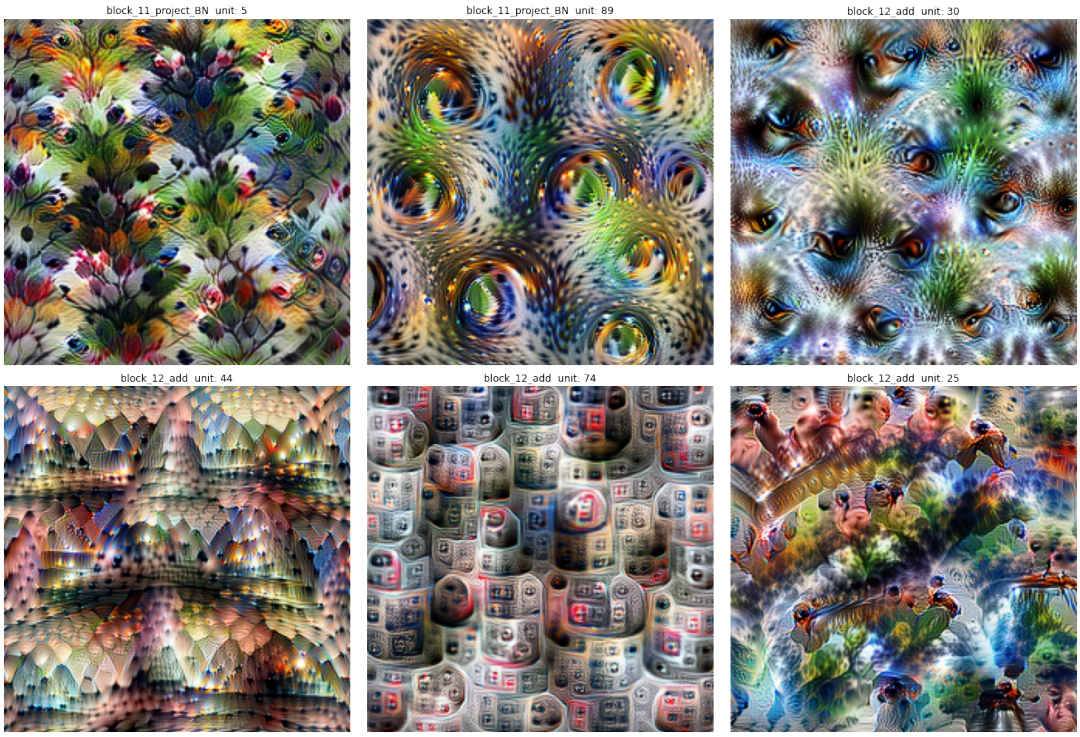


Figure 5.6: Parts. NEB visualizations can be found here

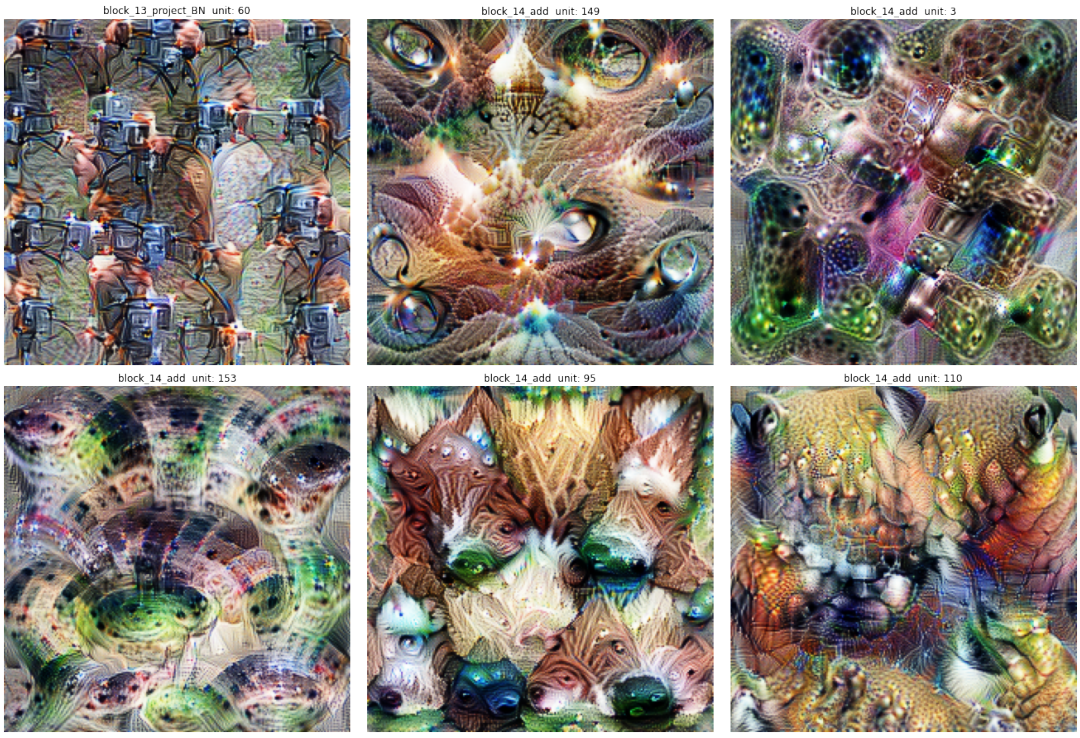


Figure 5.7: Objects. NEB visualizations can be found here

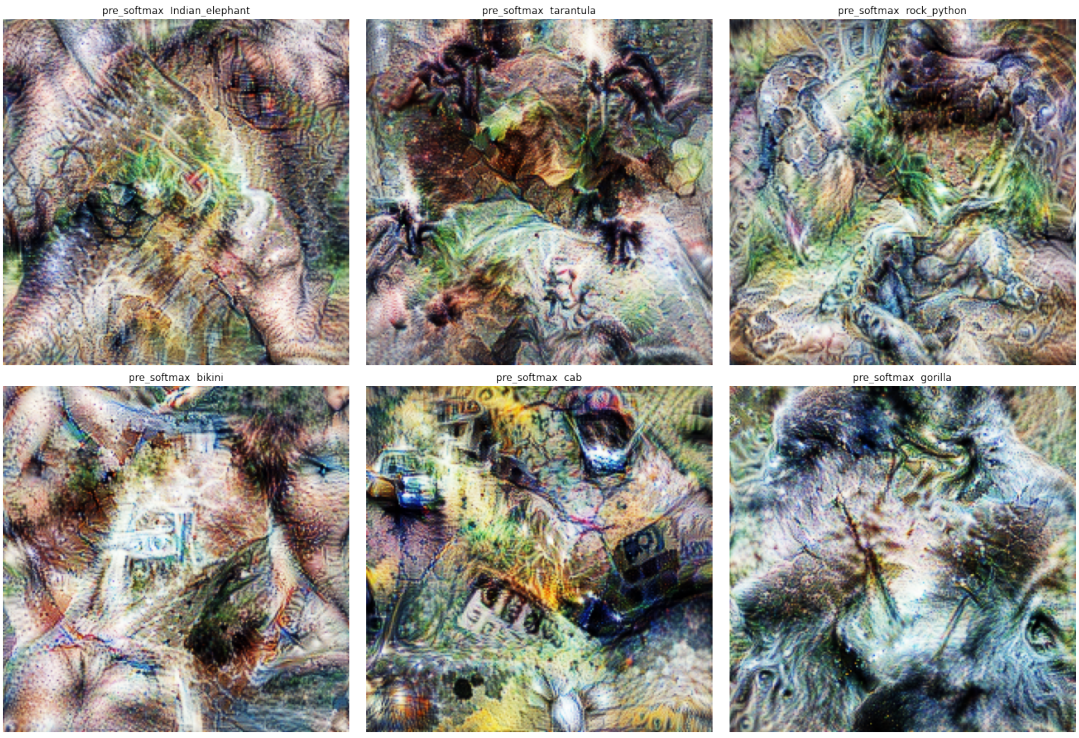


Figure 5.8: Classes. NEB visualizations can be found here

Chapter 6

Discussion and Conclusion

I implemented an optimization algorithm for feature visualization with different regularization techniques and discussed their respective strengths and drawbacks. Further I used the lucid library to provide visualizations using differentiable parameterization. I adapted a NEB algorithm and used it to optimize towards paths of maximum activation between different feature visualizations. The resulting path was used to produce short video clips, which are a convenient and visually appealing way to show an overview over the facets of a feature and their respective activations. Using both techniques I visualized the different layers of MobilenetV2, showcasing the different types of features.

6.1 Limitations and Future Work

The videos produced by the NEB algorithm still have limited diversity. They showcase a range of facets in the sense that adversarial example like images are also part of what activates a feature map. In other cases they also reveal different semantic ideas. This is not given however and sometimes the videos just show a diversity of image quality over the same semantic idea. Olah, Mordvintsev, and Schubert [21] discuss a method which can achieve variety while optimizing multiple images with the same regularization technique. They optimize a batch of images at once and add a diversity term to the objective function which forces the images to be different from one another. This method can also produce artifacts but helps revealing different facets of feature. The fact that the other regularizations are identical would help with the problem of using maxima from different loss surfaces for the NEB algorithm. Transformation robustness is still not an option for the NEB algorithm but a consistent set of regularization techniques makes choosing the hyperparameters for the objective function of NEB easier. Further experimentation with hyperparameters and regularizations techniques only affecting the gradient, might also yield settings which produce high quality visualizations. This would allow to apply NEB to actual maxima of its loss surface and therefore avoid the problem of over optimization.

Despite the total variation penalty the images in the videos are still optimized towards noisy maxima. The optimization process could be further improved by using differentiable parameterization in the NEB algorithm. Parameterizing the image in a spacial and color decorrelated space leads to better image quality as was discussed earlier. Right now when using starting images, which were optimized with differentiable parameterization, the pivots in between are optimized using standard

RGB parameterization. Parameterizing all starting images and the pivots in Fourier space should increase image quality significantly.

When watching the videos the movement seems to slow down around the starting images, even though the euclidean distance of all images between two starting images is nearly even. However perceived distance between images does not correspond to euclidean distance well [36]. Ideally the images would be spaced equally according to a similarity measure which better reflects perceived distance. Trying to space the pivots according to a different similarity measure would disrupt the path to much, similar to applying transformation robustness. Differentiable parameterization could also be the solution to this problem. If the images can be parameterized in a space, where perceived distance corresponds to euclidean distance, the NEB algorithm would enforce equal perceived distance. There are similarity measures which correspond better to perceived distance, like the structure similarity index, but it turns out that the hidden representations in CNNs trained on image classification can be used to compute similarity measures which outperform pixel based metrics [36]. CNNs trained on image classification learn these representations as a side effect. Deep learning can also be used to learn distance measures specifically. These networks are trained to embed images into a space where euclidean distance is small for similar images and high for unsimilar images [14]. In order to apply differentiable parameterization a mapping from our embedding back to RGB space is required. The embedding learned by CNNs are approximately invertible [5] and can be transformed into autoencoders to make relatively good reconstructions of images from hidden representations [37]. However a more promising approach are invertible neural networks, which are designed to learn bijective functions. MintNet and i-ResNet are invertible convolutional architectures which allow reconstruction of the input images from the network output with very little loss [27, 1]. These networks were not trained on full resolution imagenet yet but future research will surely improve existing and develop new architectures. Training such a network on an invertible embedding where euclidean distance better represents perceived distance would allow to evenly space the pivots in the NEB algorithm according to perceived distance. This might even enable the use of transformation robustness, as small shifts and rotations should only slightly move the image in the embedding space. Learning such embeddings could prove very useful for feature visualization using differentiable parameterization in general. Depending on what properties the learned embedding has the optimization process can be influenced in different ways.

6.2 Interfaces for Network Interpretability

Feature Visualization is a powerful tool in making CNNs more interpretable and just visualizing the different channels is already very informative. However the picture of network behavior is still far from complete. Dataset examples are a good addition to feature visualizations and help getting a more thorough understanding of the features, especially in later layers where features seem to correspond to more semantical ideas. This leaves the second more challenging task of network interpretability, understanding how features concretely interact to classify images. Olah et al. [22] introduce interfaces which combine feature visualization with attribution methods and show a more detailed picture of network behavior. Saliency maps reveal how much impact different parts of the input image have on the output.

Channel attribution similarly investigates which feature detectors are responsible for the classification to what extend. Feature visualization and attribution methods can not only target the different channels of layers but arbitrary slices of the hidden representation. Targeting neurons and spatial positions with these methods, reveals which part of the image or feature map is actually responsible for the activation (Figure 6.1).

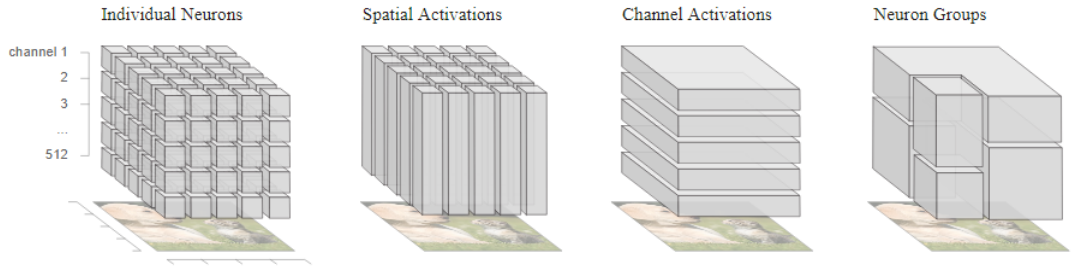


Figure 6.1: The cube of activations convolutional neural networks compute at each layer. Different slices allow targeting activations of neurons, spatial positions, channels and groups of neurons [22]

There are thousands of different channels in a network, which already can be an overwhelming amount of information and much more individual neurons. For interfaces to be useful they need to display a manageable amount of information in an understandable way. Dividing the cube of activations into more arbitrary groups helps compressing the information. However, in order for these groups to be meaningful they need to be computed individually for each input image. These interfaces already are a big step towards a more complete understanding of network interactions. Still they only show specific aspects of model behavior. Further research in this direction seems promising for a broader overview of model behavior [22].

6.3 Conclusion

Feature visualization allows for a meaningful interpretation of complex features encoded in unintelligible network parameters. Although the visualizations can never contain all the information of a feature, it is surprising how good many of them relate to semantic ideas. Just visualizing the different channels of a network tells a lot about how the images are processed. The videos produced by the NEB algorithm are a convenient way to visualize the multi-faceted nature of features and future work on the methods can further improve image quality. Still neural networks mostly remain black boxes and a lot of research in the field of network interpretability remains, especially considering the rapid development of new architectures and applications. Interfaces, which connect different techniques to paint a clearer and more detailed picture of model behavior, are already an important step towards a better understanding of deep neural networks. In order to make models easier to understand, interpretability should also be a focus in the design of new architectures. Keeping up with developing techniques to understand the models we are building is essential for the future development of the field. For a future with more complex and autonomous AI systems, human oversight and insight are key aspects.

References

[1] Behrmann, Jens et al. Invertible Residual Networks (2018). arXiv: 1811.00995.

[2] datahacker.rs. CNN One Layer of A ConvNet. [Online; accessed September 02, 2020]. 2018. URL: http://datahacker.rs/one-layer-covolutional-neural-network/.

[3] datahacker.rs. CNN Pooling Layers. [Online; accessed September 02, 2020]. 2018. URL: http://datahacker.rs/pooling-layers-cnn/.

[4] Draxler, Felix et al. Essentially No Barriers in Neural Network Energy Landscape (2018). arXiv: 1803.00885.

[5] Gilbert, Anna C. et al. Towards Understanding the Invertibility of Convolutional Neural Networks (2017). arXiv: 1705.08664.

[6] Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *Proceedings of the International Conference on Learning Representations (ICLR)*. 2015.

[7] Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. Deep Learning. http://www.deeplearningbook.org. MIT Press, 2016.

[8] Hansson, Jamie Heinemeier. About the Apple Card. Nov. 11, 2019. URL: https://dhh.dk/2019/about-the-apple-card.html.

[9] Henkelman, Graeme and Jonsson, Hannes. Improved tangent estimate in the nudged elastic band method for finding minimum energy paths and saddle points. *J. Chem. Phys.* 113 (2000), p. 9978.

[10] Howard, Andrew G. et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017. arXiv: 1704.04861.

[11] Jonsson, Hannes, Mills, Greg, and Jacobsen, Karsten W. Nudged Elastic Band Method for Finding Minimum Energy Paths of Transitions. *Classical and Quantum Dynamics in Condensed Phase Simulations* (1998), pp. 385–404.

[12] Kaiming, He et al. Deep residual learning for image recognition. *CoRR* (2015). abs/1512.03385.

[13] Kathuria, Ayoosh. Intro to optimization in deep learning: Gradient Descent. [Online; accessed September 02, 2020]. 2018. URL: https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/.

[14] Kaya, M and Bilge H, S. Deep Metric Learning: A Survey. *Symmetry* 11(9):1066 (2019).

[15] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[16] Kuo, C. -C. Jay et al. Interpretable Convolutional Neural Networks via Feedforward Design. 2018. arXiv: 1810.02786.

[17] LeCun, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. Vol. 86(11). 1998, pp. 2278–2324.

[18] Mahendran, Aravindh and Vedaldi, Andrea. Visualizing Deep Convolutional Neural Networks Using Natural Pre-Images (2015). DOI: 10.1007/s11263-016-0911-8. arXiv: 1512.02017.

[19] Mordvintsev, Alexander et al. Differentiable Image Parameterizations. *Distill* (2018). DOI: 10.23915/distill.00012.

[20] Nguyen, Anh, Yosinski, Jason, and Clune, Jeff. Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks. 2016. arXiv: 1602.03616.

[21] Olah, Chris, Mordvintsev, Alexander, and Schubert, Ludwig. Feature Visualization. *Distill* (2017). DOI: 10.23915/distill.00007.

[22] Olah, Chris et al. The Building Blocks of Interpretability. *Distill* (2018). DOI: 10.23915/distill.00010.

[23] Q., Zhang and S.-C., Zhu. Visual interpretability for deep learning: a survey. *Frontiers of Information Technology & Electronic Engineering* 19(1) (2018), pp. 27–39.

[24] Sandler M.and Howard, Andrew G. et al. Mobilenetv2: Inverted residuals and linear bottlenecks. *CVPR* (2018).

[25] Sheppard, Daniel, Terrell, Rye, and Henkelman, Graeme. Optimization methods for finding minimum energy paths. *J. Chem. Phys.* 128 (2008), p. 134106.

[26] Silver, D. and Huang A.and Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. *Nature* 529 (2016), pp. 484–489. DOI: doi.org/10.1038/nature16961,.

[27] Song, Yang, Meng, Chenlin, and Ermon, Stefano. MintNet: Building Invertible Neural Networks with Masked Convolutions. 2019. arXiv: 1907.07945.

[28] Springenberg, Jost Tobias et al. Striving for Simplicity: The All Convolutional Net. 2014. arXiv: 1412.6806.

[29] Szegedy, Christian et al. Intriguing properties of neural networks. 2013. arXiv: 1312.6199.

[30] Taigman, Yaniv et al. Deepface: Closing the gap to human-level performance in face verification. *CVPR* (2014), pp. 1701–1708.

[31] towardsdatascience. A Basic Introduction to Separable Convolutions. [Online; accessed September 02, 2020]. 2018. URL: https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728.

[32] Vaswani, Ashish et al. Attention Is All You Need. 2017. arXiv: 1706.03762.

[33] Vigdor, Neil. Apple Card Investigated After Gender Discrimination Complaints. *The New York Times* (Nov. 10, 2019).

[34] Zeiler, M. and Fergus, R. Visualizing and Understanding Convolutional Networks. *Computer Vision - ECCV* (2014), pp. 818–833.

[35] Zhang, Quanshi, Wu, Ying Nian, and Zhu, Song-Chun. Interpretable Convolutional Neural Networks. 2017. arXiv: 1710.00935.

[36] Zhang, Richard et al. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. 2018. arXiv: 1907.07945.

[37] Zhang, Yuting, Lee, Kibok, and Lee, Honglak. Augmenting Supervised Neural Networks with Unsupervised Objectives for Large-scale Image Classification (2016). arXiv: 1606.06582.

Appendices

.1 Transformation Robustness Schedules

The schedule of my implementation.

- Pad the input by 9 pixels
- Jittering with 6 pixels
- Scaling by a factor randomly selected from [0.975, 0.99, 1.01, 1.025, 1, 1, 1, 1]
- Rotating by an angle in degrees randomly selected from [-3, -2, -1, 0, 1, 2, 3, 0, 0, 0, 0, 0]
- Jittering with 3 pixels
- Cropping the padding

The schedule used by in the Distill Blog post by Olah, Mordvintsev, and Schubert [21]

- Padding the input by 16 pixels
- Jittering by up to 16 pixels
- Scaling by a factor randomly selected from [1, 0.975, 1.025, 0.95, 1.05]
- Rotating by an angle in degrees randomly selected from [-5, -4, -3, -2, -1, 0 , 1, 2, 3, 4, 5, 0, 0, 0, 0, 0]
- Jittering by up to 8 pixels
- Cropping the padding

The default schedule of the lucid version 0.3.9-alpha

- Pad the input by 12 pixels
- Jittering with 8 pixels
- Scaling by a factor randomly selected from [0.9, 0.92, 0.94, 0.96, 0.98, 1.0, 1.02, 1.04, 1.06, 1.08, 1.1]
- Rotating by an angle in degrees randomly selected from [-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 0, 0, 0, 0]
- Jittering with 4 pixels
- Cropping the padding

.2 Blurring Schedule

I used the following schedule for blurring with the gaussian filter

$$\sigma_t = 1.5 - sigmoid((\frac{t}{T} - 0.5)8)$$

t is the current optimization step and T the number of steps. Consequently σ is distributed between 1.5 and 0.5.