University of São Paulo
Institute of Mathematics and Statistics
Bachelor of Computer Science

A Survey of Explainable AI (XAI) Methods for Convolutional Neural Networks

Antonio Fernando Silva e Cruz Filho
João Gabriel Andrade de Araujo Josephik

Final Essay

mac 499 — Capstone Project

Supervisor: Prof. Dr. Nina S. T. Hirata

São Paulo
2024

Agradecimentos

Do. Or do not. There is no try.

— Mestre Yoda

Texto texto. Texto opcional.

Resumo

Antonio Fernando Silva e Cruz Filho
João Gabriel Andrade de Araujo Josephik. **Title of the document: *a subtitle***. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2024.

Elemento obrigatório, constituído de uma sequência de frases concisas e objetivas, em forma de texto. Deve apresentar os objetivos, métodos empregados, resultados e conclusões. O resumo deve ser redigido em parágrafo único, conter no máximo 500 palavras e ser seguido dos termos representativos do conteúdo do trabalho (palavras-chave). Deve ser precedido da referência do documento. Texto texto. Texto texto.

Palavras-chave: Palavra-chave1. Palavra-chave2. Palavra-chave3.

Abstract

Antonio Fernando Silva e Cruz Filho
João Gabriel Andrade de Araujo Josephik. **A Survey of Explainable AI (XAI) Methods for Convolutional Neural Networks**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2024.

Elemento obrigatório, elaborado com as mesmas características do resumo em língua portuguesa. De acordo com o Regimento da Pós-Graduação da USP (Artigo 99), deve ser redigido em inglês para fins de divulgação. É uma boa ideia usar o sítio www.grammarly.com na preparação de textos em inglês. Text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text. Text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text.

Keywords: Keyword1. Keyword2. Keyword3.

List of Abbreviations

AI	Artificial Intelligence
ML	Machine Learning
XAI	Explainable AI
MLP	Multilayer Perceptron
CNN	Convolutional Neural Network
Conv	Convolution
IME	Institute of Mathematics and Statistics
USP	University of São Paulo

List of Figures

List of Tables

List of Programs

Contents

Introduction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Other Section Example

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Next Section example

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Chapter 1

Background

In this chapter, we will introduce important concepts required for the understanding of this study. We begin by introducing Explainable AI (XAI) and its principles. We also introduce Neural Networks and important concepts such as Gradient Descent and Back Propagation. Finally, we introduce Convolutional Neural Networks, the main focus of this study.

1.1 Explainable AI

With the rise of Machine Learning models in the last decade in the business and academic areas, Artificial Intelligence (AI) is becoming increasingly present in important decision-making tasks. However, as AI models have become more sophisticated, particularly with the advent of Deep Learning techniques, their internal workings have often remained opaque. Explainable AI (XAI) aims to make models and their decisions more transparent, interpretable and understandable to both experts and inexperienced users.

1.1.1 What is Explainable AI?

Defining a mathematical formalization to explainability of Machine Learning is a difficult task considering the subjective nature of what one may consider "explainable". In non-mathematical terms, Explainability in AI refers to the capacity to articulate or justify the behavior of a model, focusing on methods that explain a model's decisions after they are made.

Another important concept in the area is Interpretability, which can be defined as "the degree to which a human can understand the cause of a decision" by Miller (2017).[1] In this case, however, a model's decision is understandable entirely by its inherent transparency. In other terms, the model is simple enough to be interpretable by a human directly, without the use of external techniques.

[1] Miller, Tim. "Explanation in artificial intelligence: Insights from the social sciences." arXiv Preprint arXiv:1706.07269. (2017).

Models with low complexity whose decisions are understandable by humans are defined as *Interpretable Models*. Linear Regression, Logistic Regression and Decision Tree models are examples of models classified as *Interpretable Models*. Now, models with a level of complexity that prevents humans from directly understanding their decision-making processes are referred to as *Explainable Models*. Recently popular *Deep Learning Models* are one kind of *Explainable Models* and will be the main focus of this essay, especially *Deep Convolutional Neural Networks*, explored in section 1.3.

1.1.2 Why Explainable AI is Necessary

Creating explanations to a model's decisions can yield many advantages, including more ethical and fair decisions, correctly following regulatory compliances and easier model debugging.

To ensure ethical and fair decision-making, Machine Learning systems must provide justifiable decisions, as they often exploit discriminatory patterns to enhance accuracy, which can perpetuate harmful biases. For instance, the COMPAS algorithm, used in U.S. courts to assess recidivism risk, was analyzed by ProPublica[1] and found to exhibit significant bias against Black defendants, frequently overestimating their likelihood of reoffending compared to their actual risk.

Explainable AI (XAI) is sometimes a mandatory requirement, particularly under regulations like the United Kingdom's General Data Protection Regulation (GDPR). The GDPR mandates that organizations must provide clear and understandable explanations for decisions that significantly impact individuals, especially those made by automated systems commonly powered by Machine Learning algorithms. Without XAI, high-stakes decisions cannot leverage such models in the United Kingdom, highlighting the crucial role of explainability in enabling the broader adoption of Machine Learning for real-world applications while ensuring compliance and fairness.

When debugging Machine Learning models, their behavior can often be unpredictable, revealing biases that may not have been initially apparent to humans. These biases can result in high performance on training, validation, or even test datasets but lead to poor performance in real-world deployment. For instance, consider training an image classifier to differentiate between dog and cat images. The model may achieve impressive accuracy on images of dogs in green fields. However, upon examining the regions of the image the model relies on for its predictions, researchers might discover that it focuses on the background rather than the animals themselves. This happens because dog owners are more likely to photograph their pets outdoors, leading to an unintended association between dogs and green backgrounds. Techniques from Explainable AI, such as Grad-CAM (Selvaraju *et al.*, 2019) and Gradient Saliency methods, enable researchers to visualize these image regions, providing critical insights into model behavior and helping to address such biases.

[1] *"How We Analyzed the COMPAS Recidivism Algorithm", by ProPublica: https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm*

1.2 Gradient Descent

Let $f : A \to B$ where $A \subseteq \mathbb{R}^n$ for $n \in \mathbb{N}$ and $B \subseteq \mathbb{R}^+$. Suppose we want to find the solution to the optimization problem

$$\underset{x \in A}{\operatorname{argmin}} f(x) \tag{1.1}$$

when $\frac{\partial f}{\partial x}$ is known for any value of x. Considering that the vector $\frac{\partial f}{\partial x}$ points to the direction of the steepest ascent of the function, the vector $-\frac{\partial f}{\partial x}$ will point to the steepest descent from the given point x. Therefore, one can define an initial random value for x and update x using $-\frac{\partial f}{\partial x}$ and a scaling factor η in order to find a local minimum of f and an approximation to the solution of given optimization problem.

We can define such method using the following formula, where x_t represents the value of x at iteration t of the algorithm:

$$x_{t+1} = x_t - \eta \frac{\partial f(x_t)}{\partial x_t}. \tag{1.2}$$

The term η is often called the *learning rate* used in the Gradient Descent method and is often defined manually by the user.

The Gradient Descent method can be used to optimize a neural network's parameters to solve a given problem using a *loss function*.

1.3 Neural Networks

Neural Networks are proven to be universal approximators.[2] That means that Neural Networks are Machine Learning models capable of representing any continuous function, therefore making Neural networks adept at modeling a range of different complex problems. This class of models have seen a growing presence across both academic and industry landscapes. However, given the architecture of multiple hidden layers of Neural Networks creating complex internal patterns, such models are classified as Explainable Models.

In this section, the inner workings of Neural Networks will be explained, starting with the *Perceptron*, considered the fundamental building block of Neural Networks.

1.3.1 Perceptron

A Perceptron is a Machine Learning model inspired by how biological neurons work. It is a simple binary linear classifier that defines its parameters by linear combinations of points in the dataset. The Perceptron model can be described by Figure 1.1.

[2] Hornik, K., Stinchcombe, M., White, H. Multilayer feedforward networks are universal approximators. https://doi.org/10.1016/0893-6080(89)90020-8

[3] https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

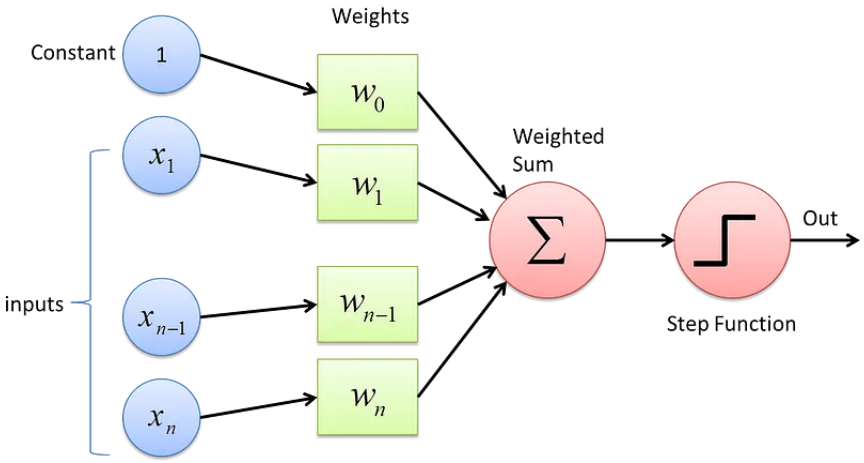


Figure 1.1: *Perceptron Architecture. Font:* Towards Data Science[3].

Where the weights w_i for $i \in \{0, 1, \cdots, n\}$ are trainable parameters and the step function can be defined as $\sigma : \mathbb{R} \to \{0, 1\}$ such that

$$\sigma(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0. \end{cases} \tag{1.3}$$

Therefore, the Perceptron model can be defined as the function[4] $f : \mathbb{R}^n \to \{0, 1\}$ where

$$f(x) = \sigma(w_0 + \sum_{i=1}^{n} w_i x_i). \tag{1.4}$$

The Perceptron model updates its parameters using each sample (x, y) of the dataset with the rule

$$w_i^{t+1} = w_i^t + \eta \, (y - f(x)) x_i \tag{1.5}$$

for $i \in \{1, \cdots, n\}$ and

$$w_0^{t+1} = w_0^t + \eta \, (y - f(x)), \tag{1.6}$$

where η is the *learning rate* hyperparameter and t is the update iteration number.

As a linear model, the Perceptron can only model linear problems, which only represent a small subset of real world problems. As a solution, researchers started combining Perceptrons in a layered structure, called Multilayer Perceptron, also famously known as *Neural Networks*.

[4] The independent term w_0 is usually called the *bias* of the Perceptron (or neuron)

1.3.2 Multilayer Perceptron (MLP)

By stacking multiple Perceptrons into multiple layers, one can build more complex decision boundaries and model more complex functions. For example, by using the Multilayer Perceptron, one can model a XOR function:

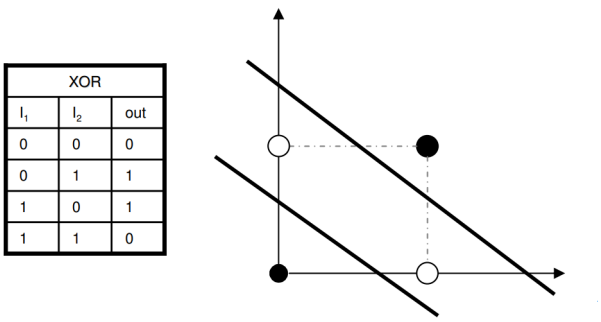


Figure 1.2: *The XOR Problem Font:* Kevin Swingler Lecture Notes.

The XOR problem can not be solved by using a single Perceptron, since it is not a linear problem. However, by using two Perceptrons (which corresponds to the two lines in the figure), one can model the non-linear XOR problem. In this specific example, one can define the uppermost line as the decision boundary of the Perceptron $f_1(x) = \sigma(-x_1 - x_2 + 1.5)$ and the lowermost line as the decision boundary of the Perceptron $f_2(x) = \sigma(x_1 + x_2 - 0.5)$. Considering the Perceptrons f_1 and f_2, we can create a new Perceptron that receives the outputs of those Perceptrons and returns the result of the XOR function. For example, we can define the Perceptron $g(x) = \sigma(x_1 + x_2 - 1.5)$, generating the following results:

x_1	x_2	$f_1(x)$	$f_2(x)$	$g(f_1(x), f_2(x))$
0	0	1	0	0
0	1	1	1	1
1	0	1	1	1
1	1	0	1	0

Table 1.1: *Perceptron output for binary combinations of x_1 and x_2.*

Showing that the non-linear problem can be successfully be solved by the Multilayer Perceptron.

Although the Multilayer Perceptron has the perk of being able to model complex functions, we are still limited by how to model is trained, since it cannot be trained by using the update rule of the traditional Perceptron.

By using a technique known as Backpropagation, the Multilayer Perceptron can be trained using gradient-based update rules, like the Gradient Descent.

1.3.3 Backpropagation

In order to find the weight's gradients of our MLP, one can use Backpropagation, a technique that involves computing those gradients by performing a *Forward Pass* and a *Backward Pass* on the Neural Network.

Forward Pass

The Forward Pass basically consists in computing the input through the network and comparing the output with the expected value using a loss function $\mathcal{L}$. During the forward pass, the output of each layer is stored in memory to be later used in the Backward Pass.

Backward Pass

In the Backward Pass, the gradients of the loss with respect to the weights are calculated to update the network. By using the chain rule, one can start off by calculating the gradients of the last layer of the network, and then use the result to calculate the gradients of the weights of the previous layer, *going on the opposite direction of the Forward Pass.*

Updating Weights

With the gradients of the loss with respect to the weights calculated, now the weights are updated in an iterative process, until a satisfiable loss/accuracy is achieved or new model/dataset tunning is necessary.

1.4 Convolutional Neural Networks

1.4.1 Convolutions

First, it is important to define what a convolution is. Given two discrete one-dimensional signals f and g, their convolution $f * g$ is defined as:

$$(f * g)[n] = \sum_{i=-\infty}^{+\infty} f[n]g[n-i]$$

Given two discrete two-dimensional signals f and g, their convolution $f * g$ is calculated as:

$$(f * g)[m][n] = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} f[m][n] \cdot g[m-i][n-j]$$

In practice, the signal g is represented by a window (or kernel), usually square and of odd size. Thus, we can abstract convolution as the multiplication of a sliding window. The picture below illustrates this process. It is important to note that the window needs to be flipped during the convolution, although this is not illustrated in the picture.

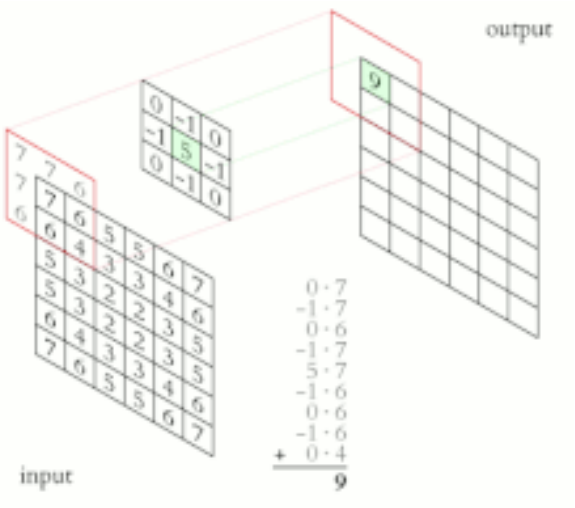


Figure 1.3: *2D Convolution*

This process can be used to apply different filters to images. For example: using a 3×3 window with all weights equal to $\frac{1}{9}$, we can generate a filter that blurs the image (moving average). Below is an example of applying this filter:

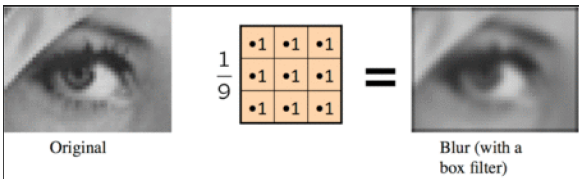


Figure 1.4: *Filtered Image (3x3 Mean Filter)*

It is also important to note that convolution is commonly implemented in machine learning contexts as "cross-correlation," which is a very similar operation but without the flipping of the window. Note that, since the weights are learned in our case, there is no difference. Therefore, in our context, convolution and cross-correlation are synonymous.

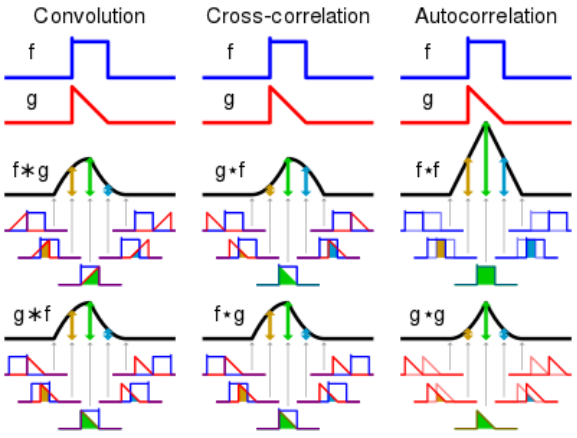


Figure 1.5: *Comparison with Cross-Correlation*

A pertinent question that can be asked is what happens at the edges of the image.

When the window is sliding over them, what happens to the missing pixels? The process of filling in these pixels is called padding. Padding can be done with zeros, the nearest pixel, or not be done at all. Note that when there is no padding, the image decreases in size after convolution.

Another important hyperparameter that can be adjusted is the "stride." This defines how many positions the window is moved at a time. That is: a stride value different from 1 also implies a decrease in image size after convolution.

1.4.2 Convolutional Layer

In order to understand the need for convolutional networks, we have to understand why it's impractical to use fully-connected networks to process images. Let's walk though an example to see that.

Consider a color image with dimensions 512x512. If we were to process this image with a conventional neural network, the input layer would have $3 \cdot 512 \cdot 512 = 786432$ dimensions. Assume a hidden layer with only 128 neurons (which is relatively small). Just between these two layers, there would be 100663296 parameters! This is highly inefficient.

The solution to this problem is to extract features from the image, which will serve as input to the network. These features could include various aspects such as symmetry, black levels, contrast, presence or absence of patterns, etc. All of these features will serve as input to the network. As a result, we can reduce the input layer's dimensionality from several hundred thousand to just a few dozen.

However, a challenge still remains: how do we select these features? We can apply convolutions to the image to calculate interesting features, and these convolutions can be learned alongside the rest of the network! It is important to understand some essential details about these networks before proceeding. Each convolutional layer has three dimensions: height, width, and the number of channels. The input layer typically has one channel for black and white images, or three channels for color images.

Each channel in each convolutional layer combines all the channels from the previous layer. In other words, the "windows" used have weights for all the channels. These windows slide over the data from the previous layer to generate **one channel** in the next layer.

Let us consider an example. Suppose we have a network that processes color images of size 128×128 pixels. This network has 3 convolutional layers with 16, 32, and 64 channels per layer, respectively. Assume a window size of 3 for all layers. In this case, we have:

- First layer: window size is $3 \times 3 \times 3$. With 16 output channels, we will have

$$16 \cdot 3 \cdot 3 \cdot 3 = 432 \text{ parameters.}$$

- Second layer: window size is $3 \times 3 \times 16$. With 32 output channels, we will have

$$32 \cdot 3 \cdot 3 \cdot 16 = 4608 \text{ parameters.}$$

- Third layer: window size is $3 \times 3 \times 32$. With 64 output channels, we will have

$$64 \cdot 3 \cdot 3 \cdot 32 = 18432 \text{ parameters.}$$

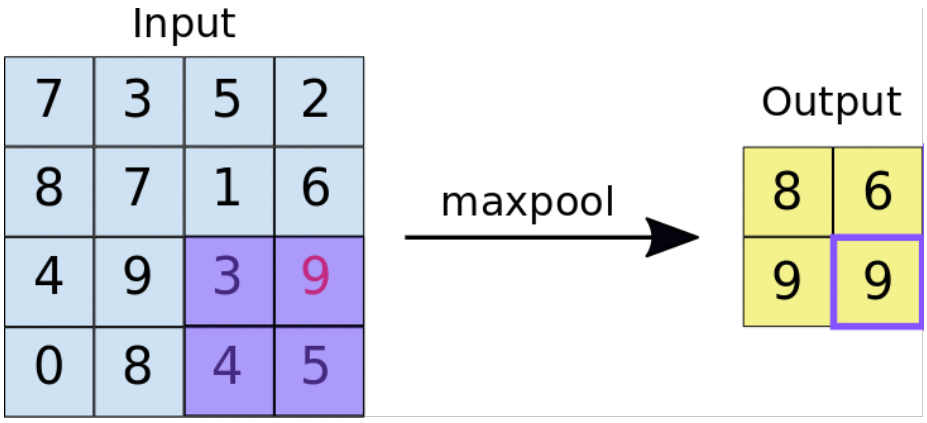
Another important detail is the output dimension of each layer. This depends on whether or not **padding** is used. **Padding** refers to how the layer behaves at the image edges. We can complete the image with zeros, the nearest pixel value, or the pixel value from the opposite edge of the image. If padding is not used, the output dimensions will decrease by $2\lfloor \frac{W}{2} \rfloor$, where W is the window size. For instance, with a window size of 3, each layer will reduce the image size by 2 pixels. If the input is 128×128, the output of the first layer will be 126×126, the second layer will output 124×124, and so on.

1.4.3 Pooling

Remember, each convolution extracts a feature from the image. Therefore, when we perform another convolution using the outputs from the previous layer, we are combining features extracted from the image to compute new features. As a result, deeper layers extract more complex features from the image. For instance, the first layer may extract features like the presence of vertical straight lines, while the tenth layer may extract features like "presence of dog snouts."

Thus, the features involved gradually become less localized and more global (pertaining to the entire image). This is why it is useful to summarize information into smaller dimensions as the network deepens.

To accomplish this, we use "pooling" layers. These layers work similarly to convolutions: windows slide over the data and compute an output based on nearby pixels. However, this time, a function is used to aggregate these data. Common functions include "max" (maximum value) and "avg" (average value).

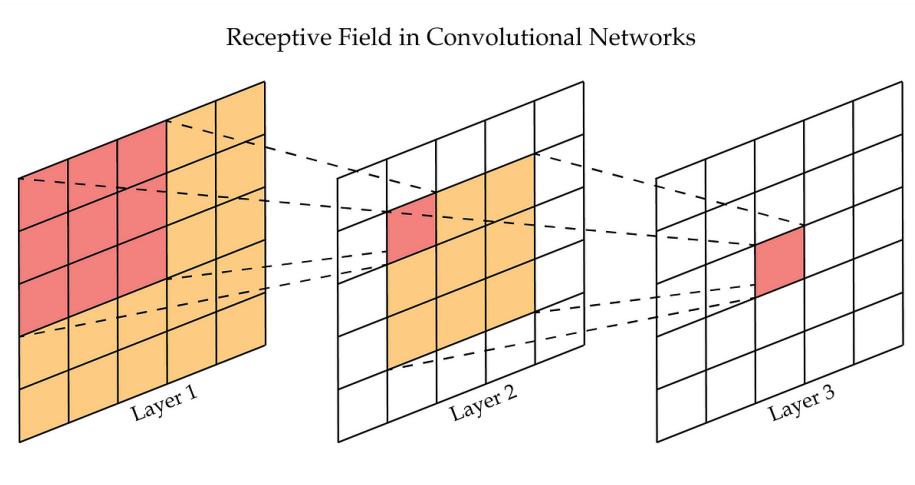


1.4.4 Receptive Field

An important concept for understanding the power of **deep convolutional networks** is the **receptive field**. This concept relates to the power that chained convolutions have.

Consider an input image. Apply a 3×3 convolution to it. Now, apply another 3×3

convolution to the output of the first convolution. Observe this output image. How much information does each pixel contain about its neighbors?



The answer is that each pixel contains information from a region of size 5×5 around it! This is the receptive field of these neurons.

A common misconception is that, since the receptive field of two 3×3 convolutions is 5×5, two 3×3 convolutions have the same **expressive power** as a 5×5 convolution. This **is not** true. A 5×5 convolution has 25 parameters, while two chained 3×3 convolutions only have 18 parameters.

Chapter 2

GradCAM

One of the most proeminent model-specific methods to acquire explanations for classification with CNNs is GradCAM. The intuition for the method is simple. At the last conovlutional layers, we have several channels that represent each a different feature. Those features are used by the next part of the network to produce the final output. If we want to know which parts of the image are being more useful to the network, we can look at the feature maps and observe which parts of the image are generating the signal used by the rest of the network.

The problem with this approach is that the features have informations about all the output classes. How we know what features are more important to the decision? The idea behind GradCAM is to **average the feature maps weighted by the gradient** of each channel with respect to a specific class.

However, this will still highlight the regions that have a negative influence to the decision. To filter out those regions, the result is passed through ReLU. The result is a coarse heatmap of the image highlighting important regions.

The formula for the heatmap with regard to the class c is:

$$H = \text{ReLU}(\sum_k \alpha_k^c A^k)$$

Where α_k^c, the weight of the k-th feature map for the class c, is defined as:

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_i^k j}$$

Where A^k is the k-th feature map, y^c is the output for the c class, and Z is the number of neurons in each map.

2.1 Guided Backpropagation

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer quis eros ultrices libero accumsan tempus eu in lacus. Cras magna ex, congue vitae fermentum nec, interdum ac eros. Praesent egestas risus eget turpis malesuada molestie. Suspendisse commodo sollicitudin nulla eu consectetur. Etiam bibendum, enim ut pretium elementum, lorem nisl imperdiet eros, non rhoncus enim eros eget nunc. Aliquam congue nisi commodo dui imperdiet, facilisis placerat nunc venenatis. Pellentesque pharetra leo at libero suscipit vehicula. Donec pellentesque at metus vel mattis. Donec id ex eget libero efficitur lobortis non quis neque. Aliquam malesuada urna neque. Donec ut pharetra turpis, fermentum dignissim ligula. In aliquam nisi sit amet lectus posuere volutpat. Nunc in suscipit erat. Ut et pulvinar ante, id viverra orci. Suspendisse augue justo, commodo convallis magna vitae, condimentum vestibulum nisl. Vivamus justo urna, posuere vitae nibh sit amet, egestas efficitur orci.

2.2 Guided GradCAM

The output of GradCAM has the dimensions of the last convolutional layer of the network, and has to be upsampled to be overlayed on top of the input image. This results in a very coarse heatmap, with rough borders and lost details. To solve this, we can multiply (pixel by pixel) the heatmap with other simpler method, as guided-backpropagation.

Chapter 3

Feature Visualization

3.1 The Optimization Problem

Feature Visualization is a technique that involves maximizing values of neurons, sets of neurons or even layers of a Neural Network in order to understand concepts learned by the model. By maximizing a neuron's value, we can better understand what set of features each part of our network is learning to capture and verify if the network is aligned with human judgement. In simple terms, Feature Visualization can be described by the following optimization problem:

$$\text{img}^* = \underset{\text{img}}{\text{argmax}}\, h_{n,x,y,z}(\text{img}) \tag{3.1}$$

where h represents the activation of a neuron, img is the input of the network, x and y represent the spatial position of the neuron, n is the layer of the network and z is the channel index. This expressions represents the problem of maximizing a value of a single neuron. For a set of neurons, the problem can be described as the formula:

$$\text{img}^* = \underset{\text{img}}{\text{argmax}} \sum_{(n,x,y,z) \in A} h_{n,x,y,z}(\text{img}) \tag{3.2}$$

where $A \subseteq \mathbb{N}^4$ is a set of combinations of network layer, channel index and spatial position vectors.

In order to find a solution to the optimization problem, we can use the Gradient Descent technique presented in Chapter 1. However, instead of minimizing an optimization problem, we are looking for a solution that maximizes Equation 3.1 or Equation 3.2. Therefore, instead of subtracting the partial derivative term of Equation 1.2 we just need to add the partial derivative multiplied by the *learning rate* (We will call the technique *Gradient Ascent*). Thus, the following formula is derived for Feature Visualization:

$$\text{img}_{t+1} = \text{img}_t + \eta \sum_{(n,x,y,z) \in A} \frac{\partial h_{n,x,y,z}(\text{img}_t)}{\partial \text{img}_t} \tag{3.3}$$

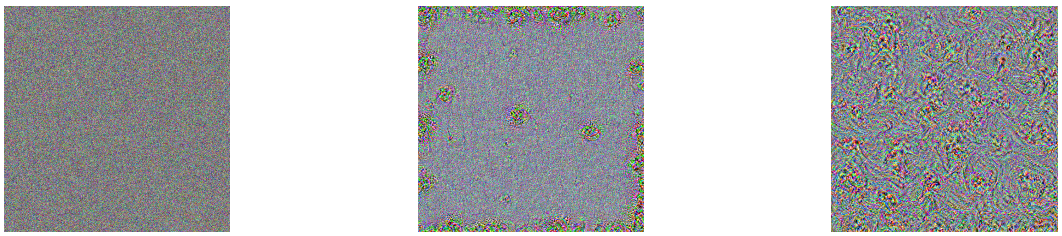
where t is the iteration t of the algorithm.

The initial image img_0 can be defined in two ways:

- A completely random initialization, with $\text{img}_0(c, x, y) = U[0, 1]$ for each channel c and image spatial positions x and y.
- A user defined image, normally a real world image.

The choice between these two approaches depends on the researcher's ultimate objective when using the algorithm. Providing an initial non-random image allows researchers to focus on specific features within the image and examine their effects on a particular set of neurons. In contrast, using a random image may be better suited for discovering unknown features.

Like most optimization problems, the Feature Visualization problem doesn't strictly have a single optimal solution, but rather multiple viable solutions that maximize the given set of neurons. For example, a set of seemingly random images may activate a neuron fully while at the same time an image of a cute dog may also activate this same neuron to its maximum value. Directly applying *Gradient Ascent* to random images without additional techniques often produces images that poorly align with human perception. This outcome occurs not because human-aligned images fail to maximize the neuron's value, but because solutions comprising seemingly random pixels are closer to the image's initial state.



(a) *Original Image*

(b) *Feature Visualization of Convolution layer 10 of VGG16*

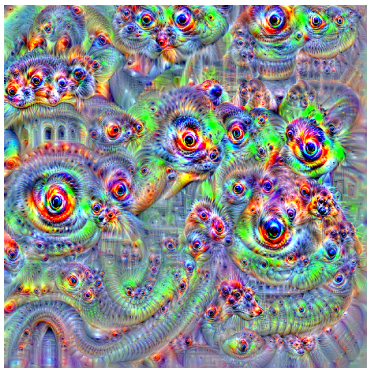
(c) *Feature Visualization of Pug (ImageNet index 254) neuron of VGG16*

Figure 3.1: *Feature Visualization images using solely Gradient Ascent. Little human-recognizable features are present in the resulting images*

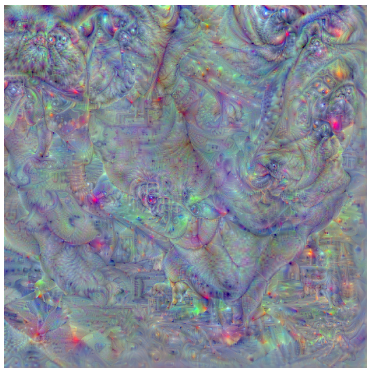
To identify human-aligned features in our models, we need effective techniques for generating images that closely correspond to those features. A closer examination reveals that Feature Visualization using solely Gradient Ascent typically produces features that occupy only small portions of the final image.

To expand these features across a larger area and create more intricate forms, we can downscale the image and apply the Gradient Ascent step. Once the downscaled image has been refined, it can be upscaled by a specific factor, and Gradient Ascent can be reapplied.

Repeating this process until the image reaches its original size enables the generation of more detailed and compelling results, as demonstrated below:



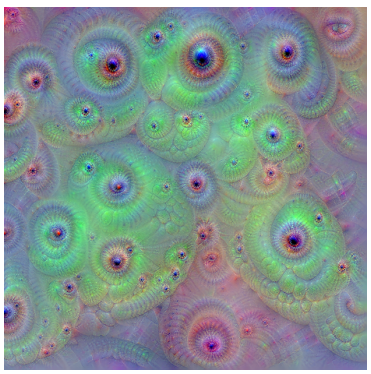
(a) *Feature Visualization of Convolution layer 10 of VGG16*



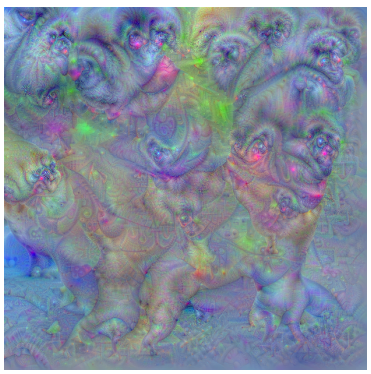
(b) *Feature Visualization of Pug (ImageNet index 254) neuron of VGG16*

Figure 3.2: *By employing Gradient Ascent across multiple image scales, we achieve results that align more closely with human perception. In Subfigure 3.2a, eye-like structures frequently emerge in the generated images, while in Subfigure 3.2b, fur-like textures (top-left portion of image) reminiscent of a Pug's facial coat are present.*

Some features aligned with human perception can now be visible by applying this method on random images. However, high frequency noise is still very present in the generated images, a feature not very common in real life pictures. In order to mitigate the high frequency noise, we may apply regularization techniques to the random image on every iteration of the Feature Visualization algorithm. For example, by applying Gaussian Blur, high frequency regions are dimished because Gaussian blur is a low-pass filter.



(a) *Feature Visualization of Convolution layer 10 of VGG16*



(b) *Feature Visualization of Pug (ImageNet index 254) neuron of VGG16*

Figure 3.3: *Feature Visualization images generated by applying Gaussian Blur besides multi-scale Gradient Ascent. This technique applied on Layer 10 predictably yields an image with much lower frequencies than 3.2a. Also, patterns like snouts or eyes are still present with round and circular features. As for Subfigure 3.3b, pug-like faces are very present in the generated image, unlike what is present in Subfigure 3.2b.*

By incorporating regularization, as shown in Figure 3.3, we generated more human-aligned images with clearly recognizable features. Using the techniques found in this section, an implementation will be proposed in the next section.

3.2 Implementation

Following the theory explored in the last section, an implementation in Python using the library Pytorch will be proposed in this section.

For our experiments, we will use a VGG16 CNN architecture trained on ImageNet. The model is made available by the library Torchvision. We can import the model with pretrained weights with the code bellow:

Program 3.1 Loading pretrained VGG16 model

```
1  from torchvision.models import vgg16, VGG16_Weights
2
3  model = vgg16(weights=VGG16_Weights.IMAGENET1K_V1)
```

Now, we need to define a way to track activations of certain layers or neurons of the network in order optimize the Feature Visualization images. In Pytorch, we can define *Hooks*, which will update every time a forward pass is executed in the network. In our implementation, we defined the following Hook class to track activations:

Program 3.2 Hook Class

```
1  class Hook:
2      def __init__(self, model_layer: Sequential):
3          self.hook = model_layer.register_forward_hook(self.hook_fn)
4
5      def hook_fn(self, _, input: Tensor, output: Tensor):
6          self.input = input
7          self.output = output
8
9      def close(self):
10         self.hook.remove()
```

Where activations can be retrieved after each forward pass by checking the hook's output:

Program 3.3 Hook Usage

```
1  model = vgg16(weights=VGG16_Weights.IMAGENET1K_V1)
2  hook = Hook(model.features[22])
3  model(image) # compute forward pass in model
4  activations = hook.output # retrieve model.features[22] activations
```

Using hooks, the Gradient Ascent step can now be implemented:

Program 3.4 Gradient Ascent Step with Normalization

```
1   def gradient_ascent_step(
2       image: torch.Tensor,
3       model: torch.nn.Module,
4       hook: Hook,
5       learning_rate: float
6       ) -> torch.Tensor:
7
8       blur = GaussianBlur(kernel_size=7, sigma=0.9)
9       image = blur(image)
10
11      image.requires_grad_()
12
13      model(image)
14      activations = hook.output
15
16      loss = (activations**2).sum()
17
18      loss.backward()
19      normalized_grad = (image.grad - image.grad.mean()) / image.grad.std()
20
21      image.grad.zero_()
22      image = image.detach()
23      image += learning_rate * normalized_grad
24
25      return image
```

The algorithm begins by applying a blur to the input image in lines 8 and 9 to regularize the results, producing a smoother and less noisy output, as discussed in the previous section. Next, gradient computation is enabled for the image in line 11, a necessary step for handling PyTorch tensors. Following this, the activations in the layer are calculated and aggregated into a single value in lines 13 to 16, which is then used to compute the gradient. Finally, the gradient is computed in line 18, normalized in line 19, and the image is updated in line 23, completing one step of Gradient Ascent.

Now, we can compute Gradient Ascent through mulitple iterations with multiple image scales, until we are satisfied with the results.

Program 3.5 Feature Visualization

```
1   def feature_visualization(
2       image: torch.Tensor,
3       model: torch.nn.Module,
4       model_layer: torch.nn.Module,
5       pyramid_levels: int,
6       growth_rate: float,
7       steps: int,
8       learning_rate: float
9   ) -> torch.Tensor:
10
11      resizer = Resizer(pyramid_levels, image)
12
13      hook = Hook(model_layer, backward=False)
14
15      for pyramid_level in range(pyramid_levels):
16          image = resizer.resize(image, pyramid_level)
17
18          for _ in range(steps):
19              image = gradient_ascent_step(image, model, hook, learning_rate)
20
21      image = image.clamp(0, 1)
22      return image
```

In simple terms, we create an *Resizer* object responsible to rescale the image on every step of the outer for loop (line 16). Then, we bind the hook to the desired model layer in line 13 and process the image for each image size computed in line 16 and for the desired amount of Gradient Ascent steps for each iteration. After the process, the image is clamped back to the interval $[0, 1]$, since the Gradient Ascent steps can bring the generated image pixels out of the interval.

3.3 Experiments

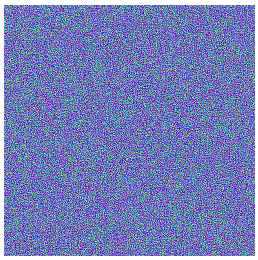
In this section, we present a series of experiments conducted to evaluate and demonstrate the effectiveness of feature visualization techniques. These experiments are designed to highlight the interpretability of the learned features in deep neural networks, analyze their behavior across different layers and architectures, and explore their potential applications. Through these experiments, we aim to provide both qualitative and quantitative insights into the representational power and limitations of feature visualization.

The experiments in this section are performed using the VGG16 convolutional neural network (CNN) architecture, pre-trained on the ImageNet dataset. The hyperparameters will be carefully optimized to produce results that closely align with human interpretability and understanding.

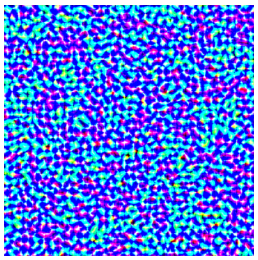
3.3.1 Layer-Wise Visualization

To create feature visualization images that capture the most significant features of an entire layer, we compute the gradient of the sum of all neuron activations within that layer. This approach highlights the collective importance of features across the layer. As an initial example, feature visualization images were generated for Layer 10 of the VGG16 architecture, as shown in Figures 3.2a and 3.3a. Next, we will extend this exploration to various layers of the network to analyze how features evolve throughout the architecture.

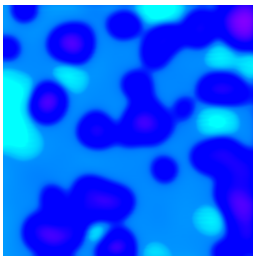
Initial Layers (1-5)



(a) *No Multiscaling, No Blurring, learning_rate* = 0.4

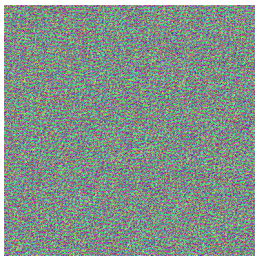


(b) *4 Layer Multiscaling, No Blurring, learning_rate* = 0.04

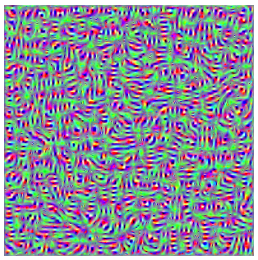


(c) *4 Layer Multiscaling, Blurring, learning_rate* = 0.04

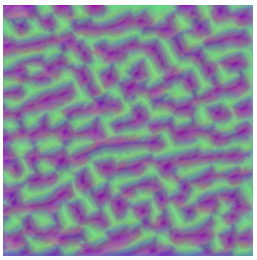
Figure 3.4: *Layer 1*



(a) *No Multiscaling, No Blurring, learning_rate* = 0.4

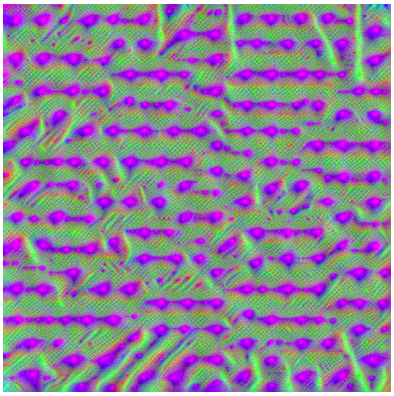


(b) *4 Layer Multiscaling, No Blurring, learning_rate* = 0.04

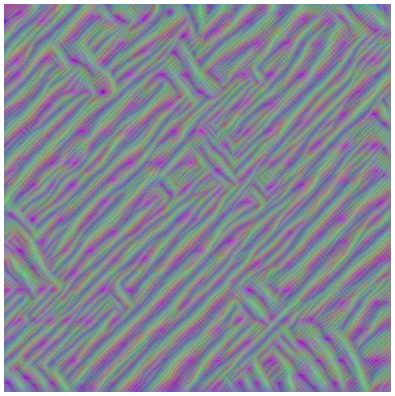


(c) *4 Layer Multiscaling, Blurring, learning_rate* = 0.04

Figure 3.5: *Layer 3*



(a) *4 Layer Multiscaling, Blurring, learning_rate* = 0.22

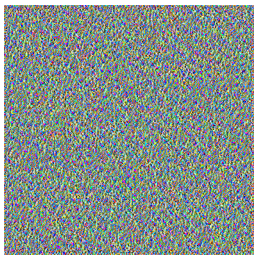


(b) *4 Layer Multiscaling, Blurring, learning_rate* = 0.08

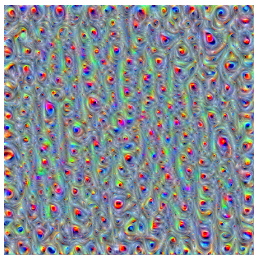
Figure 3.6: *Layer 5*

Intuitively, the initial layers of a CNN capture more simple features from images, like edges and simple formats. By analyzing the generated images, one can clearly notice the complexity enhancement throughout the layers of the network. For layer 1 (Figure 3.4), the generated images mainly focus on maximizing a color value similar to blue, probably correlated to the color distribution of the images on ImageNet. However, in layers 3 (Figure 3.5) and 5 (Figure 3.6) some patterns are already noticeable in the figures, with patterns similar to lines and curves being created.

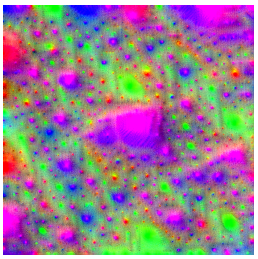
Intermediary Layers (6-9)



(a) *No Multiscaling, No Blurring, learning_rate* = 0.4

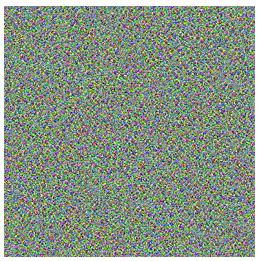


(b) *4 Layer Multiscaling, No Blurring, learning_rate* = 0.04

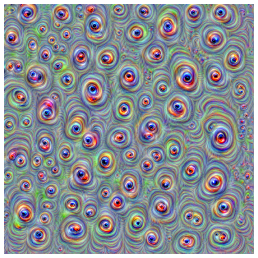


(c) *6 Layer Multiscaling, Blurring, learning_rate* = 0.22

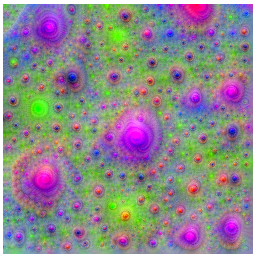
Figure 3.7: *Layer 6*



(a) *No Multiscaling, No Blurring, learning_rate* = 0.4

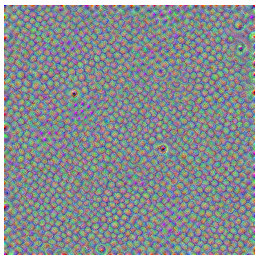


(b) *4 Layer Multiscaling, No Blurring, learning_rate* = 0.04

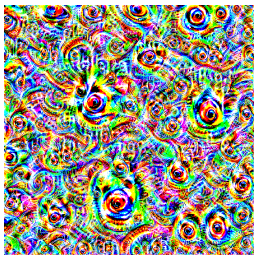


(c) *6 Layer Multiscaling, Blurring, learning_rate* = 0.22

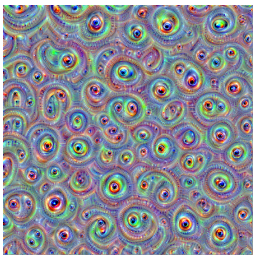
Figure 3.8: *Layer 8*



(a) *No Multiscaling, Blurring, learning_rate* = 0.4



(b) *4 Layer Multiscaling, No Blurring, learning_rate* = 0.4

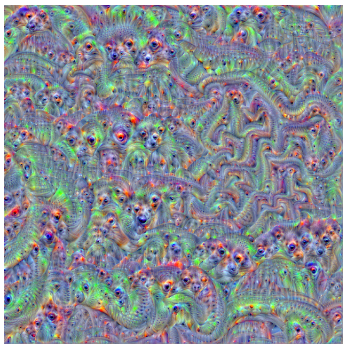


(c) *4 Layer Multiscaling, No Blurring, learning_rate* = 0.04

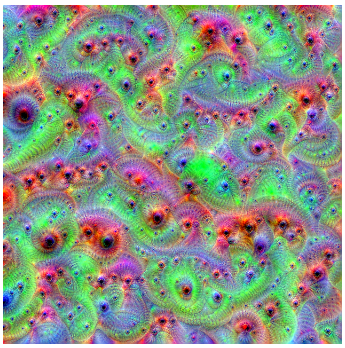
Figure 3.9: *Layer 9*

From our experiments, we can observe that complex features emerge from intermediary layers, with eye-like patterns appearing in Figures 3.8 and 3.9, probably related to the huge amount of animal pictures in the training dataset. Also, curves and circles are way more present in this layer range, showing that the network learned throughout its layers to maximize its activation to more *curvy patterns*, since real life pictures tend to have less linear line segments.

Final Layers (10-13)



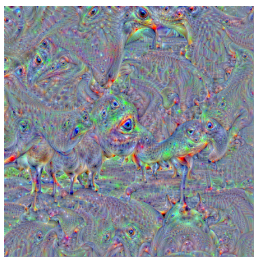
(a) *4 Layer Multiscaling, No Blurring, learning_rate* = 0.04



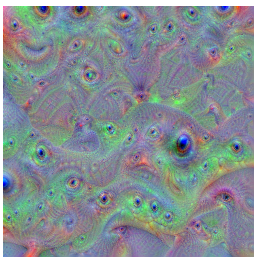
(b) *4 Layer Multiscaling, Blurring, learning_rate* = 0.4

Figure 3.10: *Layer 10*

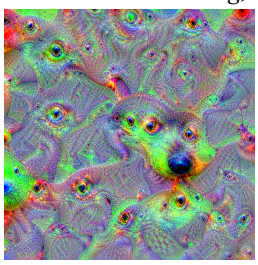
(a) *4 Layer Multiscaling, No Blurring, learning_rate* = 0.4



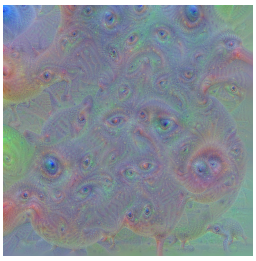
(b) *4 Layer Multiscaling, No Blurring, learning_rate* = 0.04



(c) *4 Layer Multiscaling, Blurring, learning_rate* = 0.22

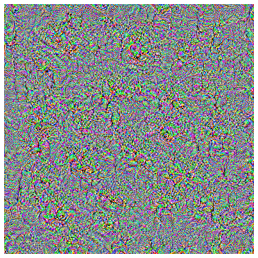


(d) *4 Layer Multiscaling, Blurring, learning_rate* = 0.4

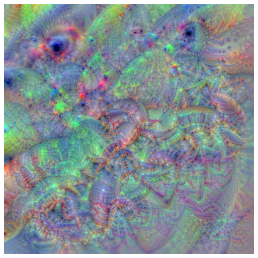


(e) *6 Layer Multiscaling, Blurring, learning_rate* = 0.1

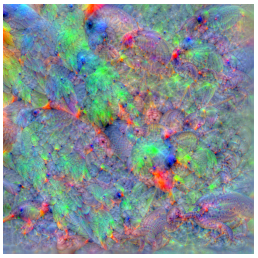
Figure 3.11: *Layer 12*



(a) *No Multiscaling, No Blurring, learning_rate* = 0.4



(b) *6 Layer Multiscaling, Blurring, learning_rate* = 0.08



(c) *6 Layer Multiscaling, Blurring, learning_rate* = 0.1

Figure 3.12: *Layer 13*

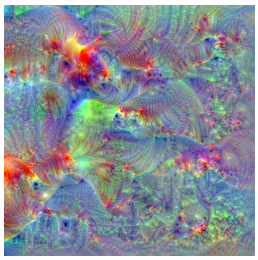
On the final layers of the network, it is possible to see that not only complex shapes are being generated but complete forms that appear like animals are also present. For example, in Figure 3.11, in image (d), a pattern very close to a dog's muzzle is present close to the center of the image. Also, for image (c) in the same layer, patterns like birds faces are also recognizable in the picture.

3.3.2 Class Visualization

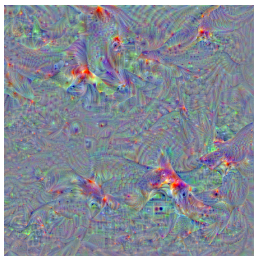
By maximizing the activation of a neuron corresponding to a specific class, we can generate images that visually represent the model's understanding of each class. This approach helps assess whether the model accurately captures the defining characteristics of each class, rather than relying on unrelated noisy biases potentially present in the dataset.

An initial example of this technique is illustrated in Figures 3.2b and 3.3b. Next, we will examine various classes using different hyperparameter configurations to explore the method further.

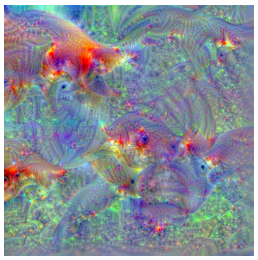
Class 1 - Goldfish



(a) *6 Layer Multiscaling, Blurring, learning_rate* = 0.1



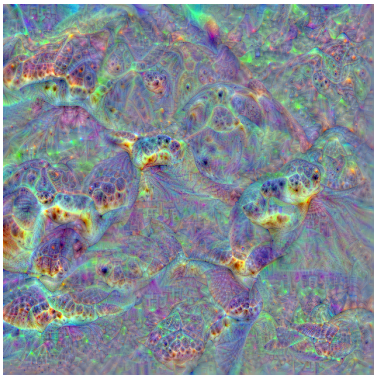
(b) *4 Layer Multiscaling, No Blurring, learning_rate* = 0.04



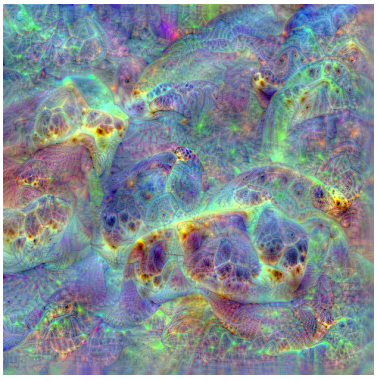
(c) *6 Layer Multiscaling, Blurring, learning_rate* = 0.09

Figure 3.13: *Goldfish Class Feature Visualization for VGG16*

Class 33 - Turtle



(a) *4 Layer Multiscaling, Blurring, learning_rate* = 0.1



(b) *6 Layer Multiscaling, Blurring, learning_rate* = 0.1

Figure 3.14: *Turtle Class Feature Visualization for VGG16*

Class 77 - Spider

Class 207 - Golder Retriever

Class 254 - Pug

Class 761 - Remote Control

Class 771 - Safe

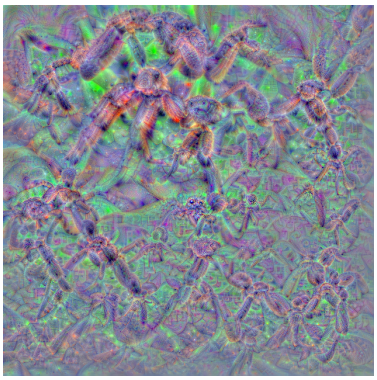
Class 783 - Screw

Class 817 - Sports Car

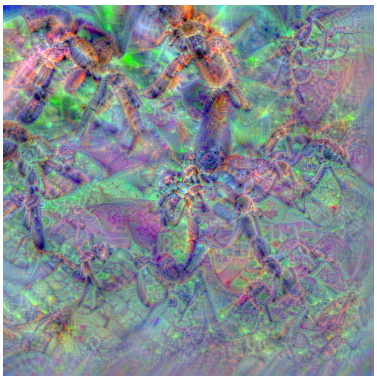
Class 883 - Vase

Class 963 - Pizza

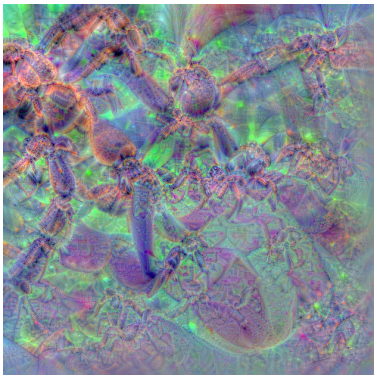
3.3.3 Feature Visualization in Non-Random Initial Images



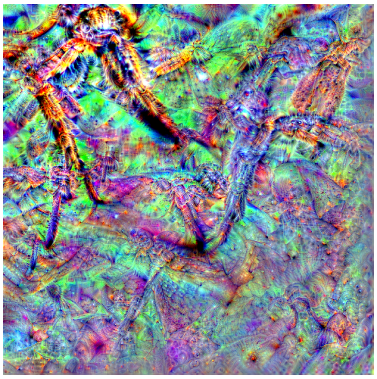
(a) *4 Layer Multiscaling, Blurring, learning_rate* $= 0.1$



(b) *6 Layer Multiscaling, Blurring, learning_rate* $= 0.1$

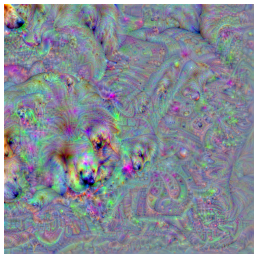


(c) *6 Layer Multiscaling, Blurring, learning_rate* $= 0.09$

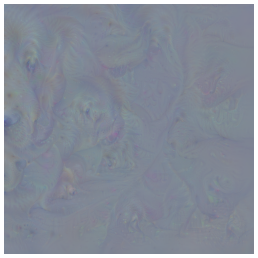


(d) *6 Layer Multiscaling, No Blurring, learning_rate* $= 0.09$

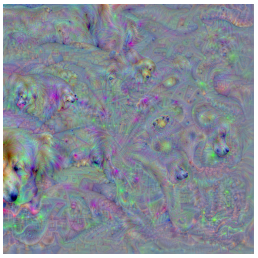
Figure 3.15: *Spider Class Feature Visualization for VGG16*



(a) *4 Layer Multiscaling, Blurring, learning_rate* $= 0.1$

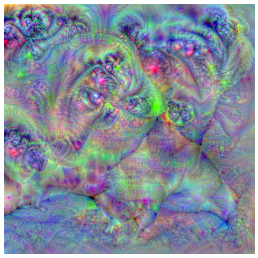


(b) *6 Layer Multiscaling, Blurring, learning_rate* $= 0.01$

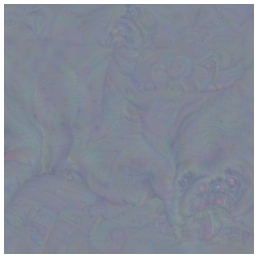


(c) *4 Layer Multiscaling, Blurring, learning_rate* $= 0.07$

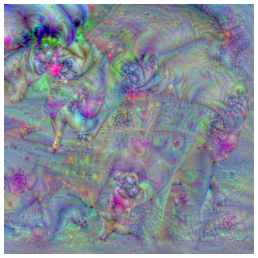
Figure 3.16: *Golden Retriever Class Feature Visualization for VGG16*



(a) *6 Layer Multiscaling, Blurring, learning_rate* $= 0.1$



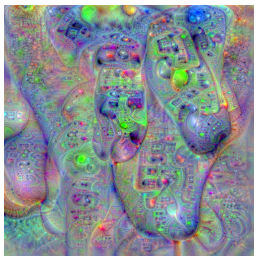
(b) *4 Layer Multiscaling, Blurring, learning_rate* $= 0.01$



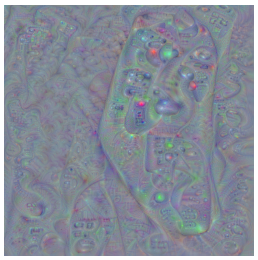
(c) *6 Layer Multiscaling, Blurring, learning_rate* $= 0.08$

Figure 3.17: *Pug Class Feature Visualization for VGG16*

(a) *4 Layer Multiscaling, No Blurring, learning_rate* $= 0.1$

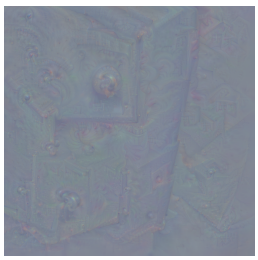


(b) *6 Layer Multiscaling, Blurring, learning_rate* $= 0.1$

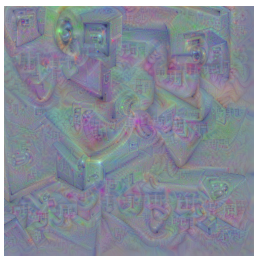


(c) *4 Layer Multiscaling, Blurring, learning_rate* $= 0.04$

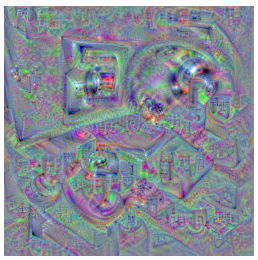
Figure 3.18: *Remote Controller Class Feature Visualization for VGG16*



(a) *6 Layer Multiscaling, Blurring, learning_rate* $= 0.01$

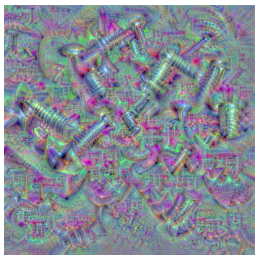


(b) *6 Layer Multiscaling, Blurring, learning_rate* $= 0.04$

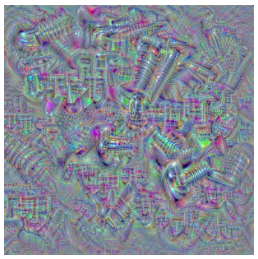


(c) *4 Layer Multiscaling, Blurring, learning_rate* $= 0.09$

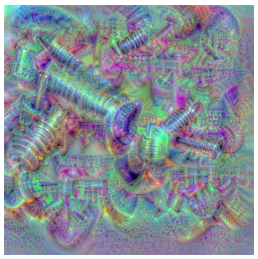
Figure 3.19: *Safe Class Feature Visualization for VGG16*



(a) *4 Layer Multiscaling, Blurring, learning_rate* $= 0.1$

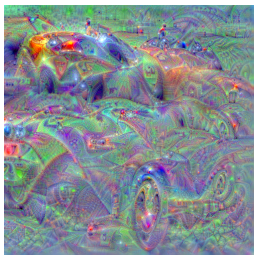


(b) *4 Layer Multiscaling, No Blurring, learning_rate* $= 0.04$



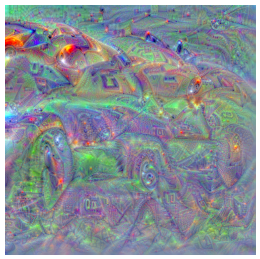
(c) *6 Layer Multiscaling, Blurring, learning_rate* $= 0.09$

Figure 3.20: *Screw Class Feature Visualization for VGG16*



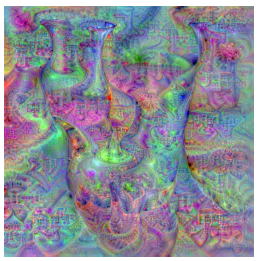
(a) *6 Layer Multiscaling, Blurring, learning_rate* = 0.1

(b) *6 Layer Multiscaling, Blurring, learning_rate* = 0.08



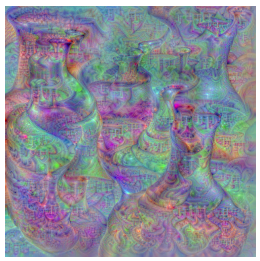
(c) *6 Layer Multiscaling, Blurring, learning_rate* = 0.09

Figure 3.21: *Sports Car Class Feature Visualization for VGG16*



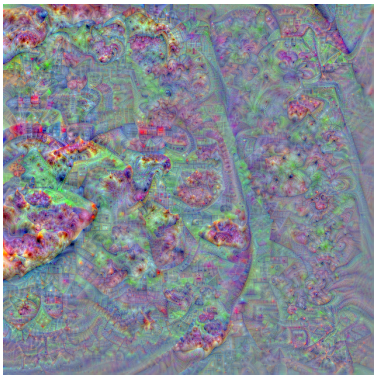
(a) *6 Layer Multiscaling, Blurring, learning_rate* = 0.1

(b) *6 Layer Multiscaling, Blurring, learning_rate* = 0.04

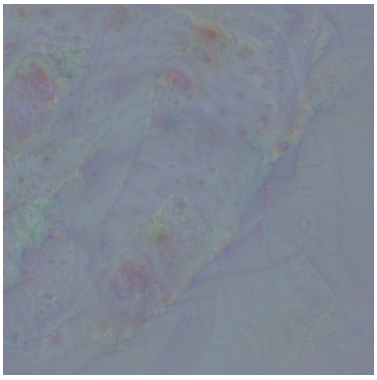


(c) *6 Layer Multiscaling, Blurring, learning_rate* = 0.08

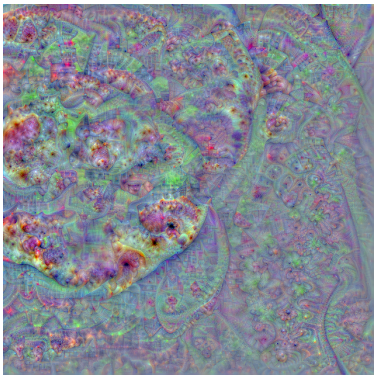
Figure 3.22: *Vase Class Feature Visualization for VGG16*



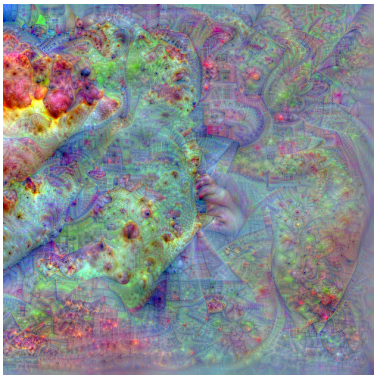
(a) *4 Layer Multiscaling, Blurring, learning_rate* $= 0.1$



(b) *6 Layer Multiscaling, Blurring, learning_rate* $= 0.01$



(c) *4 Layer Multiscaling, Blurring, learning_rate* $= 0.09$



(d) *6 Layer Multiscaling, Blurring, learning_rate* $= 0.09$

Figure 3.23: *Pizza Class Feature Visualization for VGG16*

Chapter 4

Local Interpretable Model-agnostic Explanations (LIME)

References

[Selvaraju *et al.* 2019] Ramprasaath R. Selvaraju *et al.* "Grad-cam: visual explanations from deep networks via gradient-based localization". *International Journal of Computer Vision* 128.2 (Oct. 2019), pp. 336–359. ISSN: 1573-1405. DOI: 10.1007/s11263-019-01228-7. URL: http://dx.doi.org/10.1007/s11263-019-01228-7 (cit. on p. 4).