Bibliography cited

Автономная некоммерческая организация высшего образования
«Университет Иннополис»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)
по направлению подготовки
09.03.01 - «Информатика и вычислительная техника»

GRADUATION THESIS
(BACHELOR'S GRADUATION THESIS)
Field of Study
09.03.01 – «Computer Science»

Направленность (профиль) образовательной программы
«Информатика и вычислительная техника»
Area of Specialization / Academic Program Title:
«Computer Science»

Тема / Topic	**Интерпретируемая нейронная сеть с использованием признаков визуализации / Interpretable Neural Network using Feature Visualization**

Работу выполнил / Thesis is executed by	**Майер Маргарита Робертовна / Mayer Margarita Robertovna**	подпись / signature
Руководитель выпускной квалификационной работы / Supervisor of Graduation Thesis	**Мухаммад Фахим / Muhammad Fahim**	подпись / signature
Консультанты /		

Consultants		подпись / signature

Иннополис, Innopolis, 202_

Contents

List of Figures

Abstract

Understanding convolutional neural networks (CNNs) can be improved by identifying the features recognized by each neuron. Deep Visualization techniques, such as Activation Maximization, generate input images that maximize neuron activation, facilitating the identification of recognized features. However, these approaches assume neurons detect single features, whereas neurons may respond to multiple features. Consequently, the Multifaceted Feature Visualization (MFV) algorithm is employed to reveal the various aspects of each neuron by producing synthetic visualizations of images that activate them. Initially, the algorithm visualizes a special version of AlexNet neurons in a Caffe framework. Our objective is to implement the MFV algorithm on a PyTorch framework for AlexNet, VGG, GoogLeNet and SqueezeNet networks. We want to evaluate the effectiveness of the MFV method in visualizing and understanding modern CNNs, obtaining results that are subsequently analyzed and compared.

Chapter 1

Introduction

I Problem discussion

In the current era of rapid technological advancements, artificial intelligence (AI) and its subfields have emerged as dominant forces shaping our world. One of the most promising aspects of AI is the use of neural networks, which has significantly impacted various aspects of modern life. They have enabled computers to achieve human-like performance in tasks that were previously deemed impossible, such as image recognition [1], machine translation [2], and playing complex strategy games [3]. Neural networks have also brought forth numerous applications in healthcare, finance [4], agriculture [5], and many other sectors, leading to improvements in effi-

ciency, decision-making, and quality of life.

As the use of neural networks becomes increasingly widespread, it is crucial for researchers and practitioners to comprehend their intricacies. A deep understanding of these models facilitates the development of better and more efficient architectures, which in turn leads to further advancements in AI applications. Furthermore, understanding the mechanisms that govern neural networks is essential for addressing potential issues related to ethics, fairness, and safety, ensuring the responsible integration of AI into our society [6], [7].

Convolutional neural networks (CNNs) are often referred to as "black boxes" due to their complexity and lack of transparency. The sheer number of parameters, coupled with the nonlinear nature of the activation functions, makes it challenging to decipher the inner workings of these models. Traditional analytical methods are often insufficient in providing insight into the model's decision-making process, leading to difficulties in interpreting and validating CNNs [8]. This has prompted the development of techniques aimed at improving our understanding of these models. A comprehensive review of the primary methods employed for interpreting CNNs is presented

in the Literature Review section of this thesis.

A prominent technique for understanding and interpreting CNNs is the Multifaceted Feature Visualization (MFV) method [9]. This method is similar to the Activation Maximization method [10] because both methods aim to visualize the learned features of neurons in CNNs by generating images that maximally activate certain neurons. These visualization techniques help researchers understand the types of features a CNN has learned to recognize and provide insights into the inner workings of the network. However, they differ in their assumptions and approaches; while AM assumes that each neuron detects a single feature, MFV takes into account that neurons may respond to multiple features simultaneously. By providing a more comprehensive understanding of these networks, MFV holds the potential to enhance the efficiency, and reliability of applications that rely on CNNs.

However, it is important to note that the original MFV algorithm was developed several years ago, utilizing outdated frameworks and libraries, with the specific versions not documented in the original publication. In this thesis, our contribution involves several steps: first, we will run the authors' code and identify the exact versions

of libraries and Caffe framework used for MFV algorithm. Next, we will implement the MFV algorithm using PyTorch framework and libraries as needed. Finally, we will apply the updated MFV algorithm to state-of-the-art CNNs, including AlexNet, VGG, GoogLeNet and SqueezeNet, in order to demonstrate its applicability and relevance in the current landscape of deep learning research.

Therefore, the research questions are as follows:

- RQ1: What is the role of the MFV algorithm in enhancing our understanding of CNNs, particularly in terms of uncovering the diverse features learned by individual neurons?
- RQ2: How to understand contemporary CNN using the MFV algorithm?
- RQ3: What are the outcomes and insights revealed by the MFV algorithm when applied to contemporary CNNs?

II Thesis Outline

The rest of the work is organized as follows:

- Chapter 2 provides an overview of various approaches to understanding CNNs and emphasizes the role of the MFV in this

context.

- Chapters 3 and 4 describe the methodology and implementation of the experiments that are conducted to answer questions RQ2.

- In Chapter 5, the results of these experiments are discussed, with conclusions drawn in response to RQ3 presented in Sections 5.1 and 5.3, respectively.

Chapter 2

Literature Review

In this chapter, we delve into the relevant literature that provides a foundation for addressing the problem we have identified. Our primary objective is to implement the MFV algorithm within a PyTorch framework for contemporary neural networks. Consequently, we will examine the available methods and explore the limitations in machine learning (ML) to obtain a comprehensive understanding of the problem space.

I Available methods

The primary objective of this study is to implement the MFV algorithm within a PyTorch framework for contemporary neural net-

works, with the aim of assessing its effectiveness in visualizing and comprehending modern CNNs. To accomplish this, we first aim to investigate various interpretability techniques, in order to determine the position of the MFV algorithm within this field and compare its performance with alternative approaches.

The escalating complexity of CNNs has led to a growing demand for techniques that facilitate their interpretation and understanding. As a response to this challenge, various methods have been devised, which we extensively examine in this chapter. We categorize these approaches into several groups: Deep Visualization methods, Gradient-based methods, Propagation-based methods, Local interpretability methods, and Concept-based methods. Additionally, we compare the methods within each group to provide valuable insights into their respective strengths and weaknesses.

II Deep Visualization methods

Deep Visualization methods concentrate on the activations of neurons within the neural network to comprehend the features learned by the model. Their objective is to create input images or patterns that elicit the specific activation for neurons, offering insights

into the model's internal representations and the kinds of features it has learned to identify. In this section, we describe three methods that focus on visualizing neuron activations: Activation Maximization, Multifaceted Feature Visualization, and Deep Dream.

A. Activation Maximization

Activation Maximization (AM) [10] serves as a quintessential example of Deep Visualization techniques. The primary distinguishing characteristic of AM lies in its capacity to maximize a neuron's activation. As an optimization approach, AM is developed to expose the internal representations of deep neural networks (DNNs) by generating input images or patterns that elicit maximal activation for a specific neuron or output class (Fig. 1). Through the visualization of these images, researchers are able to acquire insights into the variety of features the network has learned to identify, consequently facilitating a more profound comprehension of the underlying mechanisms.

The AM algorithm comprises several essential steps. First, a target neuron or output class is chosen, followed by the initialization of an input image with random noise. Subsequently, the model's

(a) dalmatian (b) cup (c) bell pepper

Fig. 1. Computationally generated images learned by a ConvNet, visualizing neurons responsible for three classes: (a) dalmatian, (b) cup, and (c) bell pepper [11].

weights are fixed, a forward pass is executed, calculating the gradient of the target neuron's activation concerning the input image. The input image is then updated based on this gradient, refining the image to maximize the neuron's activation. This iterative process continues until convergence or a specified stopping criterion is met.

Despite its usefulness in providing insights into the features learned by DNNs, AM has some drawbacks. For instance, the resulting images may appear unrealistic or overly optimized, and the method may not be as effective for neurons that learn multiple features [9]. There are several variations of AM that address its challenges, such as generating high-frequency noise or unrealistic images: Gaussian blur [12], total variation, and jitter [13]. Nonethe-

less, AM remains a valuable tool for understanding and interpreting the inner workings of neural networks [14].

B. Multifaceted Feature Visualization

The Multifaceted Feature Visualization (MFV) method is an approach for exploring the multiple facets that each neuron in a DNN responds to [9]. The method is baced in the idea that each class consists of multiple intra-class clusters, which represent various facets of the class. In other words, MFV is capable of revealing that a DNN has acquired the ability to identify vastly distinct aspects of a class. For example, in the movie theater class, it can recognize the interior of the theater featuring rows of seats and a stage, as well as the external facade seen during daytime or nighttime (Fig.5).

In comparison with the AM method, MFV is quite similar, with the main difference being the initialization process of AM algorithms. The MFV method employs a multi-step process to obtain an initialization per facet, referred to as the "mean image". First, the method projects the training set images that maximally activate a neuron into a low-dimensional space, specifically a 2D space using t-SNE. Next, it employs k-means clustering to group the images.

Finally, the method averages the n closest images to each cluster centroid, resulting in the creation of the initial image.

The MFV method incorporates several key ideas. First, it acknowledges the presence of intra-class clusters that represent distinct facets of a class. Second, it addresses the challenge of the AM in the presence of multiple facets by specifying which facet of the class to reconstruct. This prevents different areas of the image from optimizing towards reconstructing different fragments of various facets. Third, by initializing the AM optimization with the "mean image" of a single facet, the likelihood of reconstructing an image of that specific type is increased.

The authors have demonstrated the effectiveness of the MFV method using the ImageNet dataset, although its applicability extends beyond computer vision to other domains such as speech recognition and machine translation.

C. Deep Dream

Deep Dream is an algorithm developed by Google that leverages neural networks to produce artistic images [13]. The process begins with selecting an input image and a target layer within a pre-trained

Fig. 2. MFV visualization: 4 facets per "movie theater" neuron. In each pair of images, the bottom is the facet visualization that represents a cluster of images from the training set, and the top is the closest image to the visualization from the same cluster [9].

neural network, such as the Inception model [15]. The algorithm then feeds the input image into the network and performs a forward pass until the target layer is reached. Subsequently, activations at the chosen layer are manipulated to amplify specific features identified in the image. A backward pass computes the gradient of the modified activations concerning the input image, which is used to update the image and enhance the chosen features [16].

Unlike AM and MFV methods, which focus on individual neurons and generate images to maximize the activation of this neuron, Deep Dream typically targets a specific layer within a pre-trained

neural network. The choice of the layer affects the features emphasized in the output. Iterated for a predefined number of iterations or until a stopping criterion is satisfied, the process culminates in a final Deep Dream output. Post-processing techniques, including image normalization, clipping, or rescaling, may be executed to generate the ultimate Deep Dream image.

III Gradient-based methods

Gradient-based methods involve computing the gradients of the output with respect to the input or intermediate layers to identify the importance of input features or neurons. These techniques help visualize the sensitivity of the output to changes in the input, highlighting the regions in the input data that contribute the most to the model's prediction or decision-making process. In this section, we discuss several methods, including Saliency Maps, Guided Backpropagation, and Grad-CAM, which contribute to the understanding and interpretation of DNNs.

A. Saliency Maps

Saliency Maps are a visualization technique that highlights the most important regions or pixels in an input image that contribute to the model's output [11]. The method computes the gradient of the output with respect to the input image, providing a measure of the sensitivity of the output to small changes in the input. Saliency Maps typically represent the importance of different input features for a specific output class, helping to understand which parts of the input image are the most critical for the model's decision.

B. Guided Backpropagation

Guided Backpropagation is a method that combines the ideas of backpropagation and deconvolutional networks to visualize the features learned by DNNs [17]. The technique backpropagates the gradients of the output with respect to the input through the network while preserving only the positive gradients at each neuron. This results in clearer and more focused visualizations of the features that contribute to the model's predictions, helping to understand the internal representations of the neural network.

C. Grad-CAM

Grad-CAM is a technique that generates visual explanations for decisions made by DNNs by highlighting the regions in the input image that are most relevant to the model's predictions [18]. Grad-CAM computes the gradients of the output class scores with respect to the feature maps of the last convolutional layer and combines these gradients with the feature maps to produce class activation maps. These maps provide a visual representation of the areas in the input image that contribute the most to the model's decision, offering insights into the model's behavior.

Saliency Maps, Guided Backpropagation, and Grad-CAM all seek to elucidate CNN decision-making processes by leveraging gradients. While all Gradient-based methods rely on gradients to identify influential features or regions in input images, their utilization and interpretation of gradients differ.

Saliency Maps identify crucial pixels using the gradient of the output with respect to the input, but can produce noisy visualizations. Guided Backpropagation refines this approach by imposing constraints during the backward pass, yielding clearer visualizations. Grad-CAM, in contrast, generates coarse-grained, spatially inter-

pretable explanations by computing gradients with respect to the final convolutional layer's activation maps, highlighting regions contributing most to the output for a specific class.

IV Propagation-based methods

Propagation-based methods strive to comprehend the contributions of individual neurons within the network by attributing the output to corresponding input features. These methods involve back-propagating the output contributions through the layers of the network. Propagation-based methods are generally more complex than Gradient-based methods as they take into account the relationships between neurons throughout the network. In this section, we examine Layer-wise Relevance Propagation and DeepLIFT, which provide a comprehensive understanding of the associations between neurons and their contributions to the model's output, offering valuable insights into the complex interrelationships within the network.

A. Layer-wise Relevance Propagation

Layer-wise Relevance Propagation (LRP) is an interpretability method for DNNs that explains a model's output by attributing rel-

evance scores to input features [19]. The algorithm initializes the output neuron's activation as its initial relevance score and iteratively propagates these scores backward through the network. During propagation, relevance scores are redistributed based on the neurons' contributions, following the relevance conservation principle to maintain the total relevance across layers. The process is repeated until the input layer is reached, and the resulting relevance scores reveal the importance of each input feature in the model's decision-making process. LRP provides insights into the model's inner workings and the key input features that contribute to the output.

B. Deep Learning Important FeaTures

Deep Learning Important FeaTures (DeepLIFT) is an interpretability method for explaining DNN predictions by attributing importance scores to input features [20] The technique involves selecting a reference input, computing the difference-from-reference for each input feature, and performing a forward pass through the network with both the actual and reference inputs. During a backward pass, contribution scores are computed for each neuron by comparing activation differences between the actual input and reference in-

put. These contribution scores are propagated from the output to the input layer, fairly distributing the total contribution among the neuron's incoming connections. The resulting importance scores at the input layer can be visualized as a heatmap or other visualization, facilitating model interpretation and identification of influential input features.

LRP and DeepLIFT are both attribution methods for DNNs that seek to understand the influence of input features on model predictions. The commonality between them lies in their goal of quantifying the importance of individual input features. However, LRP works by propagating relevance scores backward through the network in a layer-wise fashion, whereas DeepLIFT computes feature attributions by comparing the activations of neurons to a reference input. While both techniques provide valuable insights, their underlying methodologies and assumptions differ, which may lead to variations in the resulting attributions.

V Local interpretability methods

In this section, we discuss LIME, SHAP, and Counterfactual Explanations methods, which contribute to the understanding and

interpretation of DNNs. These methods aim to provide human-understandable explanations for individual predictions or instances made by model, regardless of the model's architecture or complexity. By offering local explanations, they help in understanding, validating, and improving model behavior for specific inputs, enhancing the interpretability and transparency of complex models in real-world applications.

A. Local Interpretable Model-agnostic Explanations

Local Interpretable Model-agnostic Explanations (LIME), proposed by Ribeiro et al. (2016) [21], is a model-agnostic interpretability method that provides local explanations for individual predictions by creating simplified approximations of the model's behavior around a specific input. The process begins by generating perturbations of the input data and predicting their outcomes using the complex model. Next, an interpretable model, such as linear regression or a decision tree, is trained on the perturbed instances, using the predictions from the complex model as labels. This interpretable model allows for the extraction of feature importance scores, which provide an explanation of the complex model's prediction for the in-

stance of interest. Finally, the explanation is presented in a human-understandable format, such as a bar chart displaying feature importance scores or a decision tree, offering insight into the model's decision-making process.

B. SHapley Additive exPlanations

SHapley Additive exPlanations (SHAP) is an interpretability method that quantifies each feature's contribution to a model's prediction for a specific instance by assigning a Shapley value [22], [23]. The process involves framing the feature attribution problem as a cooperative game, calculating the Shapley value for each feature by evaluating their average marginal contribution across all possible feature coalitions, and summing the Shapley values to obtain the SHAP value for a specific prediction. The SHAP values provide insights into the impact of each feature on the model's prediction, with higher absolute values indicating more significant influence. Visualization techniques such as force plots, bar plots, or summary plots can be employed to facilitate the interpretation of feature importance based on SHAP values.

C. Counterfactual Explanations

Counterfactual explanations, as discussed by Wachter et al. (2017) [24], provide insights into a model's decision-making process by identifying the minimal changes required to alter a model's prediction for a specific input. These explanations help users understand the model's behavior by presenting them with alternative, actionable scenarios that would have led to a different outcome. Counterfactual explanations can be generated using optimization techniques or by leveraging pre-trained generative models.

LIME, SHAP, and Counterfactual Explanations aim to provide local, instance-wise explanations, but differ in their approaches. LIME generates perturbations of input data, fits an interpretable model, and determines the significance of input features, while SHAP uses cooperative game theory to allocate the output prediction among input features according to their respective contributions. Counterfactual Explanations, on the other hand, focus on finding the smallest change in input features required to alter the model's prediction, providing insights into the model's decision boundary.

These methods represent some of the most well-known and widely-used techniques for interpreting CNNs, providing insights

into their internal representations, feature importance, and decision-making processes.

VI Limitations in ML implementations

In this section, we will focus on the limitations encountered in ML implementations within the scope of this thesis, specifically addressing the floating-point problem.

VII Floating point problem

The floating-point problem is a numerical issue that arises due to the finite precision of floating-point arithmetic in computers [25]. Representing real numbers with a finite number of bits inevitably leads to approximation errors when performing arithmetic operations, as some numbers cannot be exactly represented in binary [26]. These errors may accumulate and affect the overall accuracy of computations, especially in algorithms with iterative or recursive procedures.

Floating-point errors are likely to occur in situations where numbers of vastly different magnitudes are combined in calculations,

as the limited precision can cause a loss of significant digits. This is particularly problematic in scientific computing [27], ML [28], and numerical simulations, where high precision is often required for reliable results.

To mitigate the impact of floating-point errors, several approaches can be employed. One such approach is to use higher-precision arithmetic, such as double-precision or even quadruple-precision floating-point representations [29]. Another method involves utilizing numerical algorithms that are designed to be more robust to round-off errors, such as those based on interval arithmetic or error analysis. Additionally, careful scaling and preconditioning of problem variables can help to reduce the magnitude disparities that exacerbate floating-point issues [27].

However, in some cases, it might not be possible to entirely eliminate floating-point errors, particularly when dealing with ill-conditioned problems or when the computational cost of using higher-precision arithmetic is prohibitive [30]. In such situations, it is essential to understand the limitations of the chosen numerical methods and to assess the impact of floating-point errors on the obtained results, possibly by employing techniques such as error es-

timation, sensitivity analysis, or Monte Carlo simulations.

Chapter 3

Methodology

This chapter describes actions that were performed to conduct the research. Section I outlines steps of experiment organization. Section II introduce Multifaceted Feature Visualization (MFV) approach, describes how we create mean images. In Section III, we discuss the criteria used to evaluate the performance of the MFV algorithm on modern neural networks. Section IV describes the dataset used for creating visualizations. Section V introduce liabrary versions that we use to run Caffe model.

I Research Outline

In this thesis, we are interested in the MFV algorithm and its practical applications. Initially, the algorithm was implemented using outdated libraries and the Caffe framework [9]. As the Caffe framework is no longer in use, we decide to migrate the code to the PyTorch framework and employ modern libraries. Additionally, our goal is to apply the code to contemporary neural networks and compare the obtained results. For this purpose, we divide our task into several steps:

1. Create an environment for running authors' solution on Caffe, determine the necessary dependency versions, which were previously unknown.

2. Reproduce the results described in the paper. [9].

3. Transfer the pre-trained neural network model used by the authors to PyTorch.

4. Migrate the MFV algorithm to PyTorch.

5. Reproduce results from paper on PyTorch.

6. Apply the algorithm to other neural networks and compare the results using the criteria from the authors' paper.

II MFV

In this section, we will discuss the details of the MFV algorithm that we apply in our work.

The MFV algorithm is a technique designed to visualize the diverse facets or aspects that each neuron in a deep neural network (DNN) responds to. It is built upon the understanding that each class within a dataset exhibits multiple intra-class clusters, reflecting the various characteristics of the class.

The MFV algorithm works by modifying the initialization process of the Activation Maximization (AM) method. The MFV algorithm aims to provide a more focused visualization of the different facets within a class, rather than producing an averaged representation that may not capture the distinctions between the various aspects.

To better understand the MFV algorithm, let us first delve into the logic of the Activation Maximization (AM) algorithm, since MFV is based on the AM algorithm. In the subsection "AM algo-

rithm", we will examine the workings of this algorithm. In subsection "Mean image algorithm", we will describe how to create "mean image", which serves as the starting point for the algorithm's initialization.

A. AM

In a more informal sense, AM is an optimization algorithm that adjusts the pixel colors in the image to achieve maximum activation for a specific neuron. This is done by calculating the derivative of the target neuron's activation concerning each pixel, which outlines the changes in pixel color needed to enhance that neuron's activation.

Formally, we may pose the AM problem for a unit with index j on a layer l of a network Φ as finding an image x^* where:

$$x^* = argmax(\Phi_{lj}(x) - R_{\theta}(x)) \tag{3.1}$$

In this context, $R_{\theta}(x)$ represents a parameterized regularization function that may incorporate multiple regularizers, each serving to penalize the search differently, ultimately enhancing the overall image quality.

The MFV algorithm can be further divided into the following

steps:

1. Mean images generation (to be described in an upcoming section). These mean images serve as initializations for the next step.

2. Objective function definition: Define an objective function for the activation maximization, which aims to maximize the activation of a specific neuron in response to the input image.

3. Optimization Algorithm Selection: Choose an optimization algorithm, such as gradient ascent, that adjusts the input image to maximize the objective function.

4. Activation Maximization: Employ the selected optimization algorithm to adjust the input image, initializing the process with the mean image of one facet. Iterate the optimization process until a specified criterion is met or a predefined number of iterations is reached.

5. Post-processing: Apply optional post-processing techniques, such as regularization or smoothing, to the optimized images to improve their visual quality.

6. Visualization: Visualize the synthesized images to reveal different facets of the neuron's activation, enabling a better understanding of the DNN's decision-making process.

By following these steps, the MFV algorithm provides insights into the multiple aspects that each neuron responds to within a DNN, allowing for a more in-depth analysis and interpretation of the model's behavior.

B. Mean image algorithm

In this section, we discuss the algorithm for creating mean images, which is key to understanding the distinctive features of the MFV method.

The development of this algorithm is based on several ideas. The authors of the paper [9] rely on the concept that each neuron is responsible for understanding a specific concept or class, which often includes several narrower concepts or subclasses. For instance, within the bell pepper class, one can observe bell peppers of diverse colors, individually or in groups, cut open or whole, and so on (Fig. 4). In this case, it is challenging for the AM algorithm to visualize multiple concepts in a single image, as different parts of the visual-

ization represent distinct ideas, resulting in a blurry and unclear visualization. To enhance the quality of visualizations, it was proposed to initiate the optimization not with a random image, as implemented in the AM method, but with a particular mean image representing a specific facet. As a result, the visualization starting from the mean image depicts one of the subclasses the neuron is responsible for understanding. This visualization is clearer and more comprehensible due to the fact that the algorithm visualizes a less complex idea, i.e., a pepper of a specific color in a specific position, rather than all peppers at once. Thus, the MFV algorithm separates the neuron's representation into multiple distinct visualizations, each illustrating a facet of that neuron, leading to a more effective visualization compared to the AM algorithm.

The process of creating mean images is described in the algorithm. (Fig. 3).

Initially, we embed all images of a class into a two-dimensional space using PCA [31], [32] and t-SNE [33]. Next, we apply k-means clustering to identify k distinct types of images (Fig. 4). It should be noted that we visualize 10 facets per neuron in this case, but by altering k, it is possible to visualize fewer or more facets. A mean image

is generated by averaging m = 15 images. As a result, we procure a mean images for each neuron, which serve as the initializing images for the visualization of the facets pertaining to this neuron (Fig 5).

Input: a set of images U and a number of facets k
1. for each image in U, **compute** high-level (here fc7) hidden code Φ_i
2. **Reduce** the dimensionality of each code Φ_i from 4096 to 50 via PCA.
3. **Run** t-SNE visualization on the entire set of codes Φ_i to produce a 2-D embedding (examples in Fig. 4).
4. **Locate** k clusters in the embedding via k-means.
for each cluster
   5. **Compute** a mean image $\mathbf{x_0}$ by averaging the 15 images nearest to the cluster centroid.

Fig. 3. Mean images algorithm [9].

III Perfomance measure

In this section, we will discuss the criteria used to evaluate visualizations obtained for various modern neural networks. The visualizations are images that represent specific facets of neurons in neural networks. To evaluate them, criteria taken from an article describing the MFV algorithm are used. There are five criteria in total.

In total, there are five criteria:

1. **Colors appear natural.** The object we aim to visualize should

Fig. 4. Visualizing various facets of a neuron detecting bell peppers. In the center, bell pepper class training images are projected into two dimensions using t-SNE and clustered by k-means. On the sides, synthetic images are generated by the MFV for the "bell pepper" class neuron, corresponding to each of the 10 numbered facets [9]. Best viewed in electronic format, using color and zoom capabilities.

have colors that it would typically display in reality. For instance, if we aim to visualize the 'bell pepper' facet, it should display the colors that a real bell pepper would typically have.

2. **Single central object.** Most AM algorithms generate visualizations that encompass multiple objects in a single image, which often results in these objects appearing small and un-

Fig. 5. MFV visualization: 4 facets per "theater" neuron. In each pair of images, the bottom is the facet visualization that represents a cluster of images from the training set, and the top is the closest image to the visualization from the same cluster [9].

clear. The MFV algorithm, however, aims to visualize a single, central object as it anticipates that this approach will produce a more detailed and comprehensible visualization.

3. **Global structure.** We strive to create a visualization of an object with a relatively defined structure, in order to facilitate better understanding of the visualization. This implies that the object we are visualizing should have defined boundaries that distinguish it from the background.

4. **Clarity of details.** The visualization should possess sufficient

details, making it easier to recognize and understand the object.

5. **Clear context.** This criterion is based on the idea that the clearer the surrounding environment of the object is defined, the more understandable the visualization will be. For instance, if we are visualizing a facet of a mountain, which often has the sky as its background, we anticipate that the background of such a facet would be fairly homogeneous and of a blue color.

IV Dataset

In this section, we will discuss the data utilized for generating the mean images.

In order to generate mean images, we utilized the ImageNet dataset [34], just as in the original paper. The dataset contains 1,300 images for each class, each with its unique dimensions. We resized the images to 227 by 227 pixels since our ultimate goal is to produce a mean image of this size, mirroring the approach used in the original paper.

V Liabrary versions

To execute the original version of the MFV algorithm on a personal computer, it became necessary to determine the versions of the utilized libraries. Unfortunately, these specifics were not provided in the authors' original publication.

List of versions:

- backports.functools-lru-cache==1.6.4
- backports.functools-lru-cache==1.6.4
- backports.shutil-get-terminal-size==1.0.0
- cycler==0.10.0
- Cython==0.19.2
- decorator==4.2.1
- enum34==1.1.10
- ipython==5.6.0
- ipython-genutils==0.2.0
- kiwisolver==1.0.1

- matplotlib==2.2.2
- networkx==2.1
- numpy==1.11.0
- pathlib2==2.3.7.post1
- pexpect==4.8.0
- pickleshare==0.7.5
- Pillow==5.1.0
- prompt-toolkit==1.0.18
- protobuf==3.5.2.post1
- ptyprocess==0.7.0
- Pygments==2.2.0
- pyparsing==2.2.0
- python-dateutil==2.8.2
- pytz==2023.3
- PyWavelets==0.5.2

- scandir==1.10.0
- scikit-image==0.13.1
- scipy==0.17.0
- simplegeneric==0.8.1
- six==1.16.0
- subprocess32==3.5.4
- traitlets==4.3.2
- typing==3.10.0.0
- wcwidth==0.2.6

Chapter 4

Implementation

In this chapter, we delve into the practical execution of the steps detailed in the Methodology chapter, specifically within the Research Outline section. In Section I, we touch upon the use of the Caffe and Pytorch frameworks in our research. Section II tackles our implementation of the MFV code in our environment, including the determination of appropriate library and framework versions. We also detail our process of replicating the results presented in the paper [9]. Section III discusses the translation of the Caffe algorithm to Pytorch, the challenges encountered, and the methods used to overcome them. Section IV elucidates the process of mean image creation. In Section V, we describe the application of the algorithm to other neural networks.

I Frameworks overview

In our research, we run the MFV algorithm on two frameworks: Caffe and Pytorch. Our goal is to achieve the same high-quality visualization on both of these frameworks as demonstrated in the research paper. To accomplish this, our first step is to conduct an overview of the frameworks that are integral to our implementation.

A. Caffe

Caffe (Convolutional Architecture for Fast Feature Embedding) is an open-source deep learning framework developed by the Berkeley Vision and Learning Center (BVLC) that focuses on CNNs and supports image classification, segmentation, and other computer vision tasks [35]. Caffe is designed for computational efficiency, allowing for fast training and deployment of DNNs on both CPUs and GPUs. It offers a modular architecture, which enables users to easily define, train, and test different neural network architectures using a simple text-based configuration file. Moreover, Caffe provides pre-trained models and an extensive library of layer types, facilitating the rapid prototyping and experimentation of deep learning appli-

cations. Caffe exhibits the versatility to operate with a variety of backends, encompassing both Atlac and OpenBLAS. This versatility contributes to its robust functionality, allowing it to handle diverse computational tasks and scenarios.

B. Pytorch

PyTorch is an open-source ML framework developed by Facebook's AI Research lab that primarily focuses on deep learning applications, including natural language processing, computer vision, and reinforcement learning [36]. It is characterized by its dynamic computation graph (DCG) approach, which provides the flexibility to create and modify computation graphs during runtime, making it especially suitable for complex and dynamic model architectures. PyTorch's intuitive interface and "eager execution" mode allow for easy debugging and prototyping, making it a popular choice among researchers and developers.

II Caffe model

In our study, we examine the functioning of the MFV algorithm through multiple methods. Primarily, we implement the algorithm

as originally designed by the authors, with the aim of verifying its performance as described in the article. This section is specifically dedicated to this subtask.

Our initial approach was to execute the authors' Caffe-based code using Google Colaboratory. Google Colaboratory, often referred to as Google Colab, is a browser-based platform developed by Google that allows the writing and execution of Python code. This cloud-based tool is particularly beneficial for data science and machine learning tasks, providing free access to computational resources, including Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs). Google Colab facilitates collaboration on computational projects, eliminating the need for intricate setup procedures or resource provisioning. Furthermore, it integrates effortlessly with Google Drive, promoting easy sharing, editing, and management of notebooks and scripts. Google Colab also supports a broad array of machine learning libraries, positioning it as an invaluable tool in the realm of artificial intelligence research and development.

Executing the code within Google Colaboratory was not feasible due to the absence of the required version of Caffe and other

libraries. For instance, the code utilized Python version 2, which is not supported by Google Colaboratory. Consequently, we opted to run the code locally on a personal computer.

The primary challenge in executing the code stemmed from its inception several years ago, necessitating specific versions of libraries and the Caffe framework. The authors did not specify which exact versions were required, thus it was our task to determine these versions.

In order to discern the specific version of the Caffe framework employed in the code, we meticulously analyzed the documentation across multiple versions of this framework.To determine the versions of the libraries, we tested the code operation across different versions and responded to the emerging errors in the following manner. The errors pointed to missing methods in the library; we then reviewed the documentation and identified the version of the library that included the specified method. Ultimately, through a process of trial and error, we determined the required versions of the libraries.

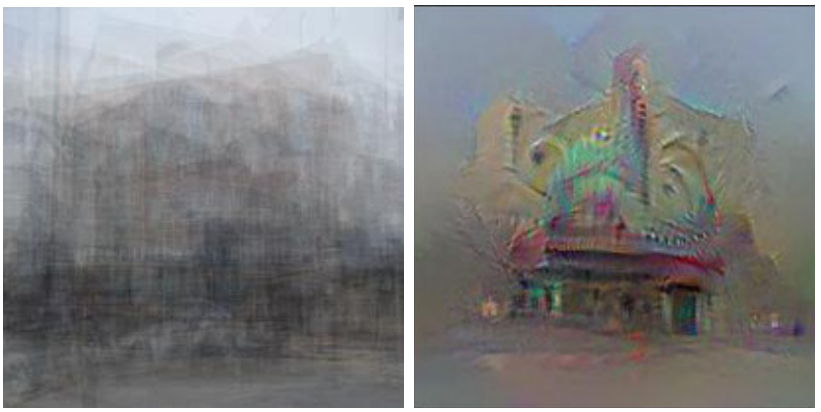
To ensure the preservation of the library versions and guarantee the ability for future system operations, we decided to encapsulate the entire working environment within a Docker container. Docker

is an open-source platform that automates the deployment, scaling, and management of applications within software containers. These containers provide a lightweight, standalone package of a piece of software, including everything it needs to run: code, runtime, system tools, libraries, and settings. The key advantage of Docker is its ability to deliver software consistently across environments, thus improving development productivity and facilitating application deployment.

Ultimately, we were successful in executing the code within our environment and achieved an outcome identical to the one depicted in the original paper. For instance, we initiated our algorithm with a starting image, the mean image of a theater, and managed to produce a visualization of the theater facet (Fig. 6). Hence, the code accompanying the article operates correctly and visualizes the neuron facets as delineated in the paper. As a result, we can utilize this code in future work and rewrite it in PyTorch.

As an outcome, we formulated a Dockerfile that constructs the necessary environment for launching the program.

In the present chapter, we delineated our process of executing the original Caffe code in a local environment and determining the



(a) mean image (b) visualization

Fig. 6. Mean image and subsequent visualization derived from it

requisite versions of the libraries and Caffe framework. By running the code locally, we were able to verify that the authors' implementation indeed yields the results delineated in the article. Thus, we are now confident that upon fully translating the code from Caffe to PyTorch, we should attain results equivalent to those reported in the article.

III Pytorch model

In this section, we discuss the fundamental concepts that guided our translation of the neural network from Caffe to PyTorch. We also detail the challenges we encountered during this process and the solutions we implemented in our program.

A. AlexNet neural network

The MFV algorithm was devised to visualize the facets of neurons specific to a special version of the AlexNet neural network. This distinct AlexNet version incorporates several layers of Local Response Normalization (LRN), which execute local response normalization over an input signal constituted of multiple input planes, with channels occupying the second dimension and applying normalization across these channels. To replicate the results as delineated in the original paper, we undertook the task of recreating this special version of the AlexNet neural network.

To initialize the model in Caffe, we utilized the files "deploy_alexnet_updated.prototxt.txt" and "bvlc_reference_caffenet.caffemodel". These files are responsible for defining the model's architecture and weights, respectively. As a result, we obtained a neural network with the necessary modules and weights, as seen in the original network. To validate our efforts, we examined the network's output when presented with an image. Both networks were expected to classify the image using ImageNet, i.e., provide probabilities reflecting how strongly the image is associated with each class. Ultimately, the networks

yielded identical probabilities up to six decimal places.

We began to ascertain whether the observed discrepancy was significant. A decision was made to investigate the intermediate computational steps. During this examination, we detected that a minor deviation appears after the first convolutional layer. Subsequently, we extracted the weights and manually calculated the expected outputs after the convolutional layer using the numpy library, which is a powerful open-source Python library that provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. This process resulted in a third output, marginally differing from the other two outputs (computed using Caffe and Pytorch), suggesting the presence of a floating-point problem, as discussed in the literature review. We also experimented with different backends for Caffe, namely Atlas and OpenBLAS, both of which yielded results with minor discrepancies. Therefore, we concluded that the tests conducted indicate a floating-point problem and the accuracy of the new neural network would be sufficient to reproduce results closely aligned with those described in the paper.

B. MFV algorithm

In this section, we will delineate our process of translating the algorithm from Caffe to PyTorch, the challenges we encountered, and the methodologies employed to resolve these issues.

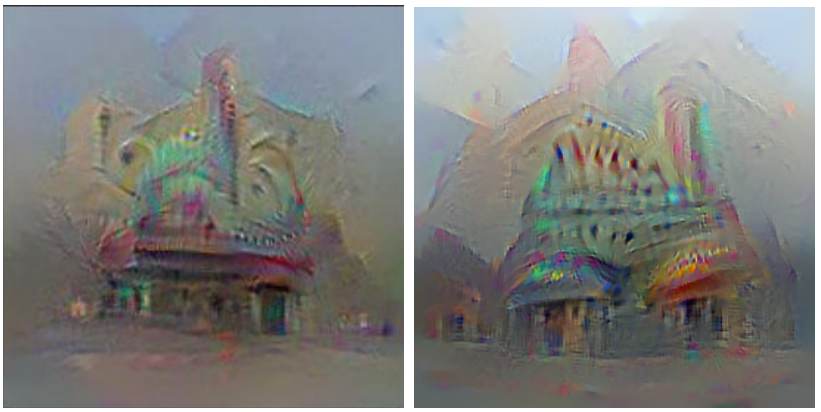
In our code, we implemented neural network logic using the PyTorch framework. The training process of the neural network is segmented into several stages. The authors have consolidated the hyperparameters of training at each stage, referring to these as 'octaves'.

After thoroughly examining the authors' code, we replicated the logic within our PyTorch-based code and encountered several challenges:

- the approach to error computation underwent a significant alteration. In the authors' work, the computation of the last layer's error utilized a manual method of gradient calculation. This method involved the authors independently computing and setting the gradients for all neurons in the final layer, before initiating the classical backpropagation algorithm. Contemporary frameworks view this approach as outdated and in-

stead prefer to employ automatic Loss functions to calculate gradients based on the network output and the desired result (target). Functions such as the L1 loss function, Mean Squared Error loss function, Negative Log-Likelihood loss function, among others, are often utilized to address similar tasks in modern frameworks. None of PyTorch's loss functions directly parallel the initial gradient computation method employed by the authors. Given that the PyTorch framework lacks an analogous loss function to that of the authors, we also used a manual method for gradient computation in our work to maximize alignment with the authors' results.

- During the training process, the authors employed a denoising method based on the Bregman iteration and the zoom function from the scipy library to enhance the sharpness and quality of the image after each iteration. This algorithm has not found its application in modern neural networks and hence lacks an implementation in the current PyTorch framework. To replicate the authors' results, it was necessary in our program to employ auxiliary libraries to apply this algorithm after each iteration. This requirement introduced the necessity for image conver-



(a) (a) (b) (b)

Fig. 7. Visualization on Caffe (a) and PyTorch frameworks (b)

sion to numpy format and back, creating an additional layer of complexity in our implementation.

We applied the MFV algorithm to a particular version of the AlexNet neural network, utilizing a mean image of one of the 'theatre' facets (Fig. 6). Ultimately, we obtained a visualization of the theatre that closely resembled the depiction in the original paper, albeit without a 100% match (Fig. 7). We firmly believe that the non-identical nature of the visualizations can be attributed to the floating-point problem, as elaborated upon earlier in the text.

IV Mean Images

The calculation of mean images is essential for initiating the process of visualizing neuronal facets. The MFV algorithm initiates

visualization not with a noisy image, but with a mean image, thereby visualizing only one of several facets of a neuron.

The algorithm for calculating the mean image is detailed in the Methodology chapter; in this section, we will describe the main characteristics of the algorithm's implementation.

For each neuron, we created 10 mean images, following the same approach as in the original paper. Initially, we selected approximately 1300 images from the training dataset of a specific class and applied preprocessing, namely center cropping of the specified image. Subsequently, we executed all steps outlined in the algorithm (Fig. 3).

As a result, we obtain the mean images, which will serve as the initial starting point for the MFV algorithm (Fig. 8). To verify the accuracy of the algorithm, we visualized images belonging to a single cluster 10). Moreover, for further verification of the algorithm's efficacy, the training images were projected into a two-dimensional space utilizing t-SNE and then segregated into clusters using the k-means method 9.

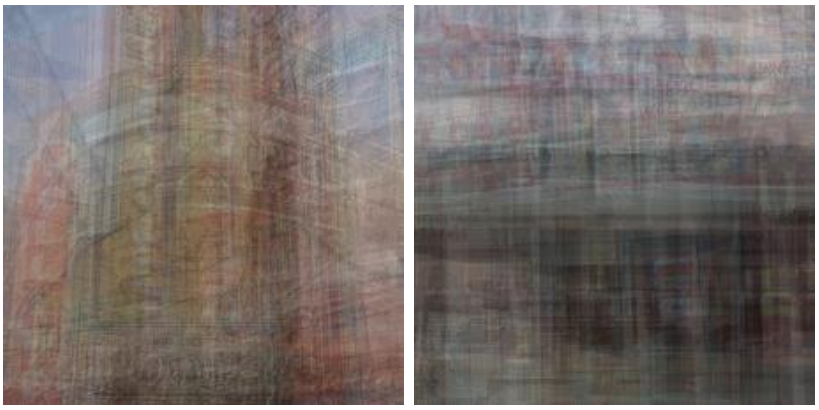


Fig. 8. Examples of mean images for "theater" neuron

V Visualization modern neural networks

In this section, we will detail how we applied the MFV algorithm to modern neural networks and the challenges we encountered.

We implemented the MFV algorithm on the following neural networks: VGG16, the original version of AlexNet, GoogleNet, and SqueezeNet. Consistent with the algorithm outlined in the original paper, we visualized the final layer of these networks, thereby generating the visual representations of neuron facets. The outcomes of these neural network operations can be found in the Results chapter.

A fundamental issue with the MFV algorithm was that it do not consider the standard deviation (std) value for neural networks. The algorithm only accounted for the mean value, which influenced the results during both pre-processing and post-processing, as well

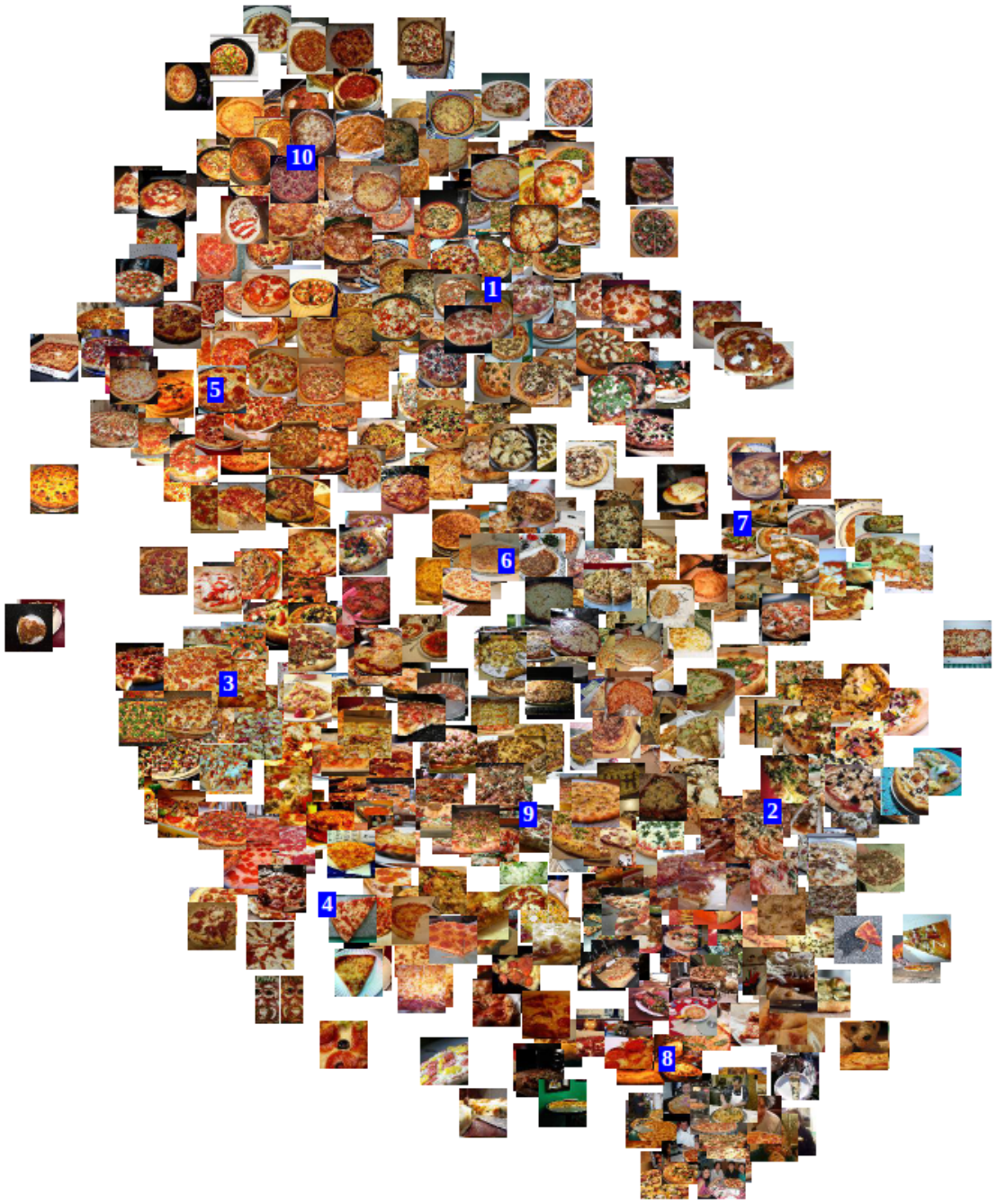


Fig. 9. Pizza class training images are projected into two dimensions using t-SNE and clustered by k-means. In varying clusters, diverse presentations of pizza are observed, encompassing a slice of pizza, a complete pizza, as well as a close-up depiction of pizza. Best viewed in electronic format, using color and zoom capabilities.

Fig. 10. Images from the "theater" cluster. These images are taken during the day, outdoors, and the theaters have signs. The MFV algorithm will visualize these patterns, creating a facet for the "theater" neuron.

as within the algorithm when a step was made to alter the image. Consequently, we incorporated the effect of the std value into the algorithm alongside the mean image value.

In our implementation, similar to the original authors, we utilized the concept of octaves, which dictate the behavior of the algorithm via a set of various hyperparameters. These parameters include the number of iterations for modifying the input image and the zoom parameter intended to generate a visualization of the object in focus.

Chapter 5

Results and Evaluation

I Migrating the MFV algorithm from Caffe to PyTorch

II Applying the algorithm to the modern neural networks

A. Colors appear natural

In the process of visualizing an object, it is crucial to represent its characteristic chromatic features faithfully. For instance, in the visualization of the ’bell pepper’ facet, it is essential to exhibit the range of hues that a real bell pepper would conventionally possess.

B. Single central object

Predominantly, Activation Maximization (AM) algorithms generate visualizations that encompass an assortment of objects within a singular image, leading to the depicted objects often appearing diminutive and indistinct. In contrast, the MFV algorithm is designed to focus on the visualization of a singular, central object, with the anticipation that such an approach would culminate in a visualization that is markedly more detailed and readily comprehensible.

C.

D.

E.

F.

G. Evaluation metrics

H. Results

1) AlexNet:

2) VGG:

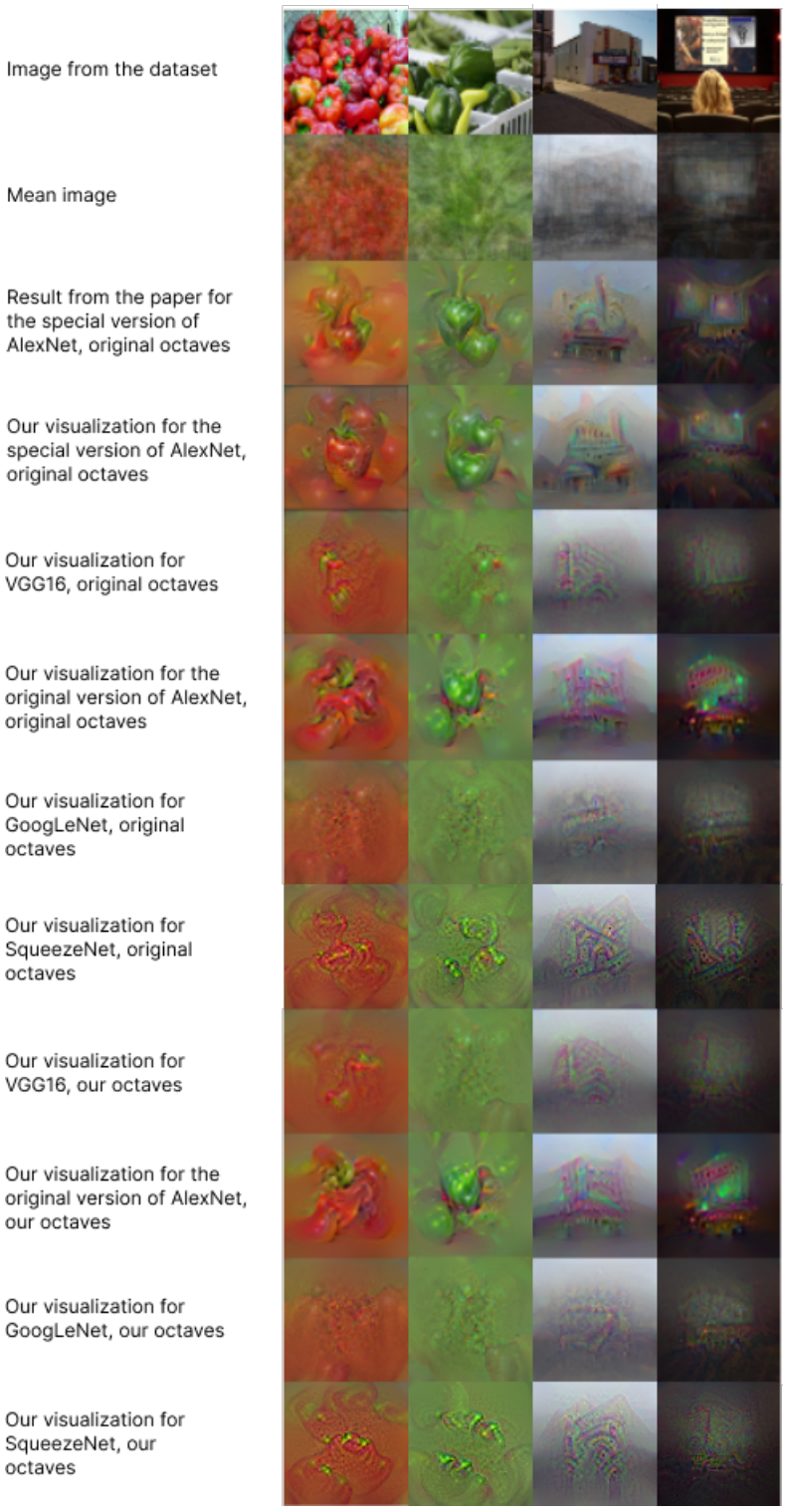


Fig. 11. Images from the "theater" cluster. These images are taken during the day, outdoors, and the theaters have signs. The MFV algorithm will visualize these patterns, creating a facet for the "theater" neuron.

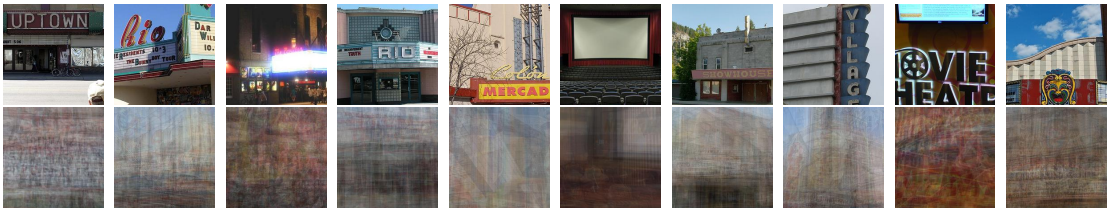


Fig. 12. 10 mean images for "theater" neuron. In each pair of images, the bottom is the mean image that represents a starting point for facet visualization, and the top is the example of image from each cluster

Chapter 6

Conclusion

describe conclusion 1Lorem 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.

Future works

Bregman iteration, nd zoom pytorch . .. PyTorch .

Bibliography cited

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[2] Y. Wu, M. Schuster, Z. Chen, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[3] D. Silver, J. Schrittwieser, K. Simonyan, *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[4] J. Sirignano and R. Cont, "Universal features of price formation in financial markets: Perspectives from deep learning," *Quantitative Finance*, vol. 19, no. 9, pp. 1449–1459, 2019.

[5] A. Kamilaris and F. X. Prenafeta-Boldú, "Deep learning in agriculture: A survey," *Computers and electronics in agriculture*, vol. 147, pp. 70–90, 2018.

[6] A. Jobin, M. Ienca, and E. Vayena, "The global landscape of ai ethics guidelines," *Nature Machine Intelligence*, vol. 1, no. 9, pp. 389–399, 2019.

[7] A. B. Arrieta, N. Dıaz-Rodrıguez, J. Del Ser, *et al.*, "Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai," *Information fusion*, vol. 58, pp. 82–115, 2020.

[8] G. Montavon, W. Samek, and K.-R. Müller, "Methods for interpreting and understanding deep neural networks," *Digital signal processing*, vol. 73, pp. 1–15, 2018.

[9] A. Nguyen, J. Yosinski, and J. Clune, "Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks," *arXiv preprint arXiv:1602.03616*, 2016.

[10] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing higher-layer features of a deep network," *University of Montreal*, vol. 1341, no. 3, p. 1, 2009.

[11] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[12] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," *arXiv preprint arXiv:1506.06579*, 2015.

[13] A. Mordvintsev, C. Olah, and M. Tyka, "Inceptionism: Going deeper into neural networks," 2015.

[14] C. Olah, A. Mordvintsev, and L. Schubert, "Feature visualization," *Distill*, vol. 2, no. 11, e7, 2017.

[15] C. Szegedy, W. Liu, Y. Jia, *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[16] A. Mahendran and A. Vedaldi, "Visualizing deep convolutional neural networks using natural pre-images," *Interna-*

tional Journal of Computer Vision, vol. 120, no. 3, pp. 233–255, 2016.

[17] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806*, 2014.

[18] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.

[19] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PloS one*, vol. 10, no. 7, e0130140, 2015.

[20] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *International conference on machine learning*, PMLR, 2017, pp. 3145–3153.

[21] M. T. Ribeiro, S. Singh, and C. Guestrin, “" why should i trust you?" explaining the predictions of any classifier,” in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[22] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” *Advances in neural information processing systems*, vol. 30, 2017.

[23] L. S. Shapley *et al.*, “A value for n-person games,” 1953.

[24] S. Wachter, B. Mittelstadt, and C. Russell, “Counterfactual explanations without opening the black box: Automated decisions and the gdpr,” *Harv. JL & Tech.*, vol. 31, p. 841, 2017.

[25] D. Goldberg, “What every computer scientist should know about floating-point arithmetic,” *ACM computing surveys (CSUR)*, vol. 23, no. 1, pp. 5–48, 1991.

[26] T. J. Dekker, “A floating-point technique for extending the available precision,” *Numerische Mathematik*, vol. 18, no. 3, pp. 224–242, 1971.

[27] N. J. Higham, *Accuracy and stability of numerical algorithms*. SIAM, 2002.

[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[29] J. W. Demmel, *Applied numerical linear algebra*. SIAM, 1997.

[30] J. H. Wilkinson, *Rounding errors in algebraic processes*. Courier Corporation, 1994.

[31] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, vol. 2, no. 11, pp. 559–572, 1901.

[32] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[33] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, vol. 9, no. 11, 2008.

[34] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in

2009 IEEE conference on computer vision and pattern recognition, Ieee, 2009, pp. 248–255.

[35] Y. Jia, E. Shelhamer, J. Donahue, *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 675–678.

[36] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.