LONDON'S GLOBAL UNIVERSITY
Department of Computer Science

How much data is enough?

Identifying Causal Links in Semantic Learning

Ilai Bachrach

Supervisor: Benjamin Guedj

Faculty of Engineering

Department of Computer Science

University College London

A Project Report Presented in Partial Fulfillment of the Degree

MSc AI for Sustainable Development

September 2024

Abstract

To power systems that can organize data at the efficiency of the brain while being able to understand complex semantic relationships, requires a fundamental shift in how we design machine learning models. Existing Models that incorporate semantic relationships in data rely on 1) abundant data 2) dimensionality reduction methods or 3) augmentation strategies (the likes of mix-up, or re-mix). However, this is computationally expensive, does not provide the interpretability we hope to receive, and does not emulate the causal and dynamic nature of our brain. The causal paradigm reflects a fundamental shift in how we see and use causality to understand how models form semantic meaning and what data is needed to maintain meaning. In this MSc thesis, we introduce a novel causal model-agnostic interpretability method, that can be applied to knowledge graphs. Identifying causal links in data can help focus models on relevant parts of an input, allowing data to be slimmed out, and allowing models to be smaller while maintaining the same accuracy. With this method, we achieve state-of-the-art results on MNIST and CIFAR-10 with less than half of the data usually required. We also discuss the potential to extend this causal framework to multi-modal data as well as knowledge graphs especially relevant in the advent of LLMs. You can find initial visualizations of important pixels and distances in the shallow distance learning framework here. https://github.com/ilaiadaya/shadow

Keywords— causal learning - transfer learning - structural learning - knowledge graphs - shallow distance learning

You are smarter than your data. Data does not understand cause and effect; humans do.

— Judea Pearl

Acknowledgements

Special thank you to Benjamin Guedj and John Shaw Taylor for helping me when I needed it and inspiring me to go down paths I didn't know existed. Also Delmiro, you enabled me to switch to this topic when it was considered too late.

Declaration

I, Name, I declare that the thesis has been composed by myself and that the work has not be submitted for any other degree or professional qualification. I confirm that the work submitted is my own, except where work which has formed part of jointly-authored publications has been included and referenced. The report may be freely copied and distributed provided the source is explicitly acknowledged.

07/09/24

Signature *Date*

Table of Contents

List of Figures

List of Tables

List of Algorithms

1 | Introduction

How much data is enough? When humans experience the world this question does not commonly enter the conscious mind. However, the brain constantly considers the importance of the information it receives through all its senses. When a child experiences the world, it begins to notice what parts of the world are important, and which are less important to its survival and curiosity. But what mechanism does it use to determine whether to save a given experience, word, sound or smell to conscious memory? Where is the threshold between recalling a memory and not being able to do so? How fast does plasticity happen or how dynamic is our memory to new experiences? When a child sees examples of dogs and cats, how many examples does it need until it realizes how to distinguish the two, and does it use the dog as a reference when it distinguishes between dogs and tigers? To what extent is this ability a function of age and experience? [1]

When you see the following images, are you able to detect what digit it is?

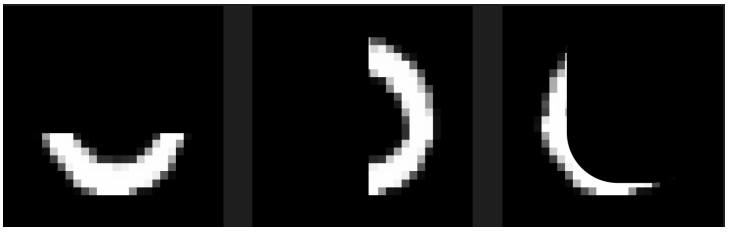


Figure 1.1: What is the minimum information we need to recognize this digit?

What about machines? Which models are able to capture this digit (a zero), and what pixels or part of the image may be important? Does a convolutional neural network perform better better at this than a simple Neural network? How does this differ amongst classes and as we scale the complexity of datasets? What procedure is most suitable for finding important pixels, is it more effective to pan around images with patches or evolve importances with a reinforcement learning approach? Does looking at differences in images from different classes help us identify important characteristics and establish

reference points for use in other domains [2]? How does this compare to language and can we extend this to a multi-modal case?

Figure 1.2: How many words or characters do you need to understand the sentence?

What if you do not know whether you are looking at a sentence or image of a digit, how does this change your ability to discriminate? Can you transfer your understanding of digits to other domains? Can we emulate the idea of spoken language by feeding snippets of images sequentially into a re-occurent neural network? How accurate do the model's predictions need to be for us to reliably consider what regions it marks as important [3]?

These questions laid the groundwork for what became a study of the minimum information needed for machines to be able to distinguish and learn differences in data. At the heart of this research are the topics of causality and frugality. Causality is the study of cause and effect (we provide a definition in the background and literature section), enabling us to formalize how changing one variable impacts another, which is frequently a concept that correlation-based algorithms do not incorporate. Frugality, on the other hand, asks us to question the use of heaps of data, recapturing that: 'we are drowning in information, while starving for wisdom' - E.O. Wilson. We explore how the questions we raise are answered by existing research in section 2 and how our unique research builds on ideas from existing methods aiming to answer the question: **To what extent can we use causality to reduce data used while maintaining accuracy of a model specifically for image classification?** We start with definitions for Causality, Frugality and transfer learning in the next section.

1.1 Background Understanding

In this section we define a few important concepts for our investigation. We use the mathematics behind causality for pixel perturbations. For the sake of completeness, we also provide the mathematics for related concepts for our investigation like transfer learning which we we do however not use in subsequent chapters.

Frugality.

Frugality in AI refers to the efficiency of a model in terms of computational resources and data required to achieve a certain performance level [4]. The goal here is: Using fewer data points m to achieve a comparable accuracy A.

Causality.

Causality in AI focuses on understanding and utilizing cause-and-effect relationships within data [5].

- Causal Relationship: A relationship where a change in one variable directly influences another variable, the cause and effect.
- Causal effect:

$$E[Y \mid do(T = 1)] - E[Y \mid do(T = 0)].$$

In the context of our study of pixel perturbations we ask: How much does Y change when we change T by one unit holding all else constant?

Transfer Learning.

Transfer learning involves leveraging knowledge from a given pre-trained model on one task to improve performance on a related task. This is given by a:

- Source Domain: $\mathcal{D}_S = \{(x_i^S, y_i^S)\}$.

- Target Domain: $\mathcal{D}_T = \{(x_i^T, y_i^T)\}$.
- with a mapping to: Find a function $f : \mathcal{X}_S \to \mathcal{Y}_T$ such that $f(x_i^T) \approx y_i^T$.

We aim to use the findings to improve the transferability of models by building databases of importances. This is discussed further in the conclusion.

Structural Learning.

Structural learning is the process of determining the structure of a probabilistic model, such as a Bayesian network, which can represent causal relationships between variables [6, 7]. For example for a graphical model: We want to learn the graph $G = (V, E)$ where V are nodes (variables) and E are edges (dependencies). We can interpret this as a causal relationship. This allows us in effect to better understand the relationships between the data and its classes.

Convolutional Neural Networks. Convolutional Neural Networks or short CNNs, work by applying "convolutional" filters to the input image. They do this to detect features like edges, textures and patterns, moving a given filter across an image. As the filter moves across the image a dot product between the filter and a given pixel value is computed producing a feature map. In the network a "node" corresponds to a filter and a given layer has multiple filters. The nodes in the next layer process the feature maps, combining information from previous layers. This results in more complex features as data progresses through layers. This also leads to increasingly complex features [8]. A typical CNN will also contain:

- Activation Functions: After a given convolution operation is applied, activation functions like ReLU add additional non-linearity.
- Pooling Layers: Operations such as max pooling reduce spatial dimensions of feature maps, and enable downsampling input as well as make the network less variant to small changes in the input.

- Flattening: feature maps are flattened into a 1d vector in effect preparing the data for a fully connected layer.
- Fully Connected Layers: to allow the network to learn high-level representations the flattened vector is passed through the fully connected layers where neurons are connected to neurons from previous layers

These features allow the CNN to learn complex features, with the downside that this generally requires at a minimum thousands of examples or features. In comparison, current state-of-the-art image classification models like Vision Transformers improve the quality of feature extraction by enabling models to focus more on relevant parts of an image. This comes at a high computational cost as attention is computed across all image patches leading to quadratic complexity in relation to the input size [9]. While this behaviour is useful and can help us understand what mechanisms work in identifying meaning, because of their computational complexity, we use CNNs for the remainder of this study. We synthesize this further in the literature review section.

Relating concepts.

In this study we use causality to effectively create a new frugal pipeline that includes a CNN. We use this pipeline to construct a resulting structural and Interpretable model that is in the future able to transfer the learning it does to different domains.

1.2 Research Objectives

The overall aim of this project is thus to explore and create a novel technique that enables models to capture more semantic meaning in data with a fraction of the data needed. For this we aim to understand existing structural learning approaches and methods. Along the way we would like to grasp how memory and the brain allow us to generalize from a handful of examples. We also aim to understand more closely how models behave

when we capture and focus on causal links in data. For this we designed and curated an algorithm for pixel importance that matches state-of-the-art results [10] on the MNIST dataset with a fraction of compute and contributes to Explainability of models in the cross-dimensional axis (not by example but by feature).

1.3 Motivation

With this research we hope to stimulate a fundamental shift in how machine learning is viewed, in a greater context of causality, and the development of semantic meaning. The brain processes around 1.3 MB of data per second running at the energy of a typical light bulb while developing profound pattern recognition abilities [11]. In contrast, a single modern GPU uses considerably more energy than a human in a typical day but demonstrates lower capability in semantic learning [12]. This research aims to delve into the intricate web of cause-and-effect connections that underlie the process of semantic learning. Our ultimate goal is to move to emulating more closely how we as humans perceive the world using machine learning. With our research we seek to improve the ability of machine learning models to capture causal relationships in data resulting in 1. equal accuracy with less data and 2) models that emulate the workings of the mind. Additionally, in the current combination of both abundant data, and niche use cases in AI, understanding the essential amount of information that is used versus needed for learning is essential, as it allows us to 1) understand how models behave and why and 2) build models for niche use cases that are able to transfer their learning, without the need for much data.

1.4 Thesis Structure

This investigation consists of 5 sections excluding this **introduction**:

Background and Literature Review. This is where we try to understand and syn-

thesize existing structural learning and interpretability approaches for images.

Methodology. This is where we explain our approach to finding and designing efficient learning algorithms as well as our main SHADOW method.

Results. This is where we report on our core experiments using our methodology and situate our research in broader topics.

Discussion. We further discuss the experimental results and provide limitations.

Conclusion. This is where we provide limitations and key takeaways.

1.5 Research Experiments

Our study revovles around the question: 'To what extent can we use causality to reduce data used while maintaining accuracy of a model specifically for image classification?'. To answer our main research question we designed research experiments that revolve around perturbing input data and as well as analyzing and using resulting predictions. We create a setup by which we are able to perturb parts of images of both training and test data and report resulting predictions. We use causal inference as a model-agnostic method to better understand the behaviour of models. As part of our initial question we also ask: Can we use important parts of images to construct new causal models that run at the efficiency of the human brain? We show that looking at data in terms of distances and determining convergence in knowledge graphs is very useful for domain-specific applications.

1.6 Scientific Contributions

We introduce Shallow Distance Learning (SHADOW), a new causal framework which identifies (1) important parts of images, (2) combines those parts into new patches and (3) computes visualizations of distances between patches of examples from different classes. SHADOW, Our novel model-agnostic interpretability method can be used to

understand and speed up existing model architectures that work with image and or text data. The first part enables the extraction of n number of important pixels or features in a multi-modal dataset by utilizing a patch and perturbation-based approach. The second component of the method enables training new model architectures with the found important features. The final component allows the computing of distances between important features of examples from different classes. This final component paves the way to more causal model architectures that utilize key features found from machine learning models, but can also be used without a model. The setup enables us to understand differences in the initial dataset, use those differences to supplement a found understanding of predictions from a machine learning model and use distances as a way to capture the information of our dataset, iteratively deciding how much data is enough by determining changes in distances as data is removed or added to a dataset. This is particularly useful in the context of knowledge graphs that explode in size in LLM applications as we explicate at the end of the next section (see Section 2.5).

2 | Literature Review

This project seeks to identify and characterize the causal factors influencing semantic learning outcomes. To pave the way to more frugal (see above 1.1 for a definition) AI systems with this research we designed a new learning algorithm that matches the state-of-the-art performance on MNIST, at a fraction of the data usually required, provides additional ways of interpretability and shows that by using a 'causal mindset', we can come up with additional ways to compress data in the context of knowledge graphs and distance learning. To achieve this we synthesized existing methods that capture and understand meaning in image data.

Existing research has typically focused on understanding semantic learning by adding an additional complex structure to the beginning of a model or augmenting data to a given dataset to see how a consequent model behaves, creating entirely new architectures that capture complex semantic meaning between data points, architecting a secondary layer after a given model, to add and capture additional semantic meaning. 'Identifying causal links in semantic learning' revolves most closely around 1), looking at the direct meaning of pixels and using this to design new causal architectures.

2.1 Identifying and Extrapolating Semantics in Data

We first look at research which looks at methods in 1). Researchers who built the AdaHGNN first created a hypergraph based on label embeddings and separated image features into vectors for each label. Hypergraph neural networks connect such vectors, and through this progression examine high-order semantic relationships [13]. The work effectively boosts multi-label classification performance. Since we are interested in the importance of individual and collections of pixels rather than the relationship between them, this study is different but uses an innovative concept to formalize relationships be-

tween vectors. This method is also not very computationally efficient as the complexity scales with sample size, which is also an issue for us, as we need to run through all the data. This begs the question if we can find a way to run through portions of data as a representative sample of the importance of pixels. Another method is a metric learning approach employed in [14] which 'calculates distances between pixels' looking at differences in samples and not features per se. Yet the concept of using distance as a measure of links in data is one that can be worth investigating. Rather than distances, researchers have also investigated the extraction of semantic meaning in a process called Semantic Explainable Artificial Intelligence (S-XAI) which uses row-centered PCA for feature extraction and semantic interpretation [15]. This method aims to pinpoint semantically relevant neurons and provide corresponding visualization to improve the 'interfaces' of the CNN. Significantly, identifying semantics in data can also help us make models more explainable. Unlike these approaches, [16] involves the use of mix-up, which interpolates between training samples to enhance generalization and robustness against adversarial attacks and corrupted labels. The approach of augmenting data rather than refining and reducing data is one that is frequently employed [17, 18, 19]. Several techniques exist that explore semantic meaning as representation and augmentation in the early stages of model development.

2.2 Understanding Semantics and Explainability in existing architectures

We now examine model architectures that aim to improve semantic learning. A common case is the Gaussian Process Latent Variable Model [20]. These models use a Bayesian setup with variational compression of information for more efficient learning. The Gaussian setup enables the model and user to identify a latent structure for the data. Another niche technique is to use a simple neural network with output regulation with the help

of cluster-based soft targets [21]. This process groups the output predictions to come up with soft targets which in the training process help to increase accuracy and to reduce overfitting [21]. However, this method might still risk overfitting if the model becomes too tailored to the training data, and the concept of soft assignments may lack the complexity of Gaussian processes, which could be an avenue for investigation in determining the importance of pixels. Another promising approach includes a model with a hierarchical structure of spiking neural networks complemented with better feature learning. This addresses the problem of capturing temporal and spatial characteristics of images in an efficient manner, thus improving performance [22]. Similarly, [23] uses the wavelets process, allowing the model to learn about asymmetric and locally developed features of patterns. The authors also describe the idea of recursive partitioning for high dimensional data, which could be an important ingredient for working with large and dense images. In [24] a similar architecture is described with recursive refinement embedded into Faster-RCNN to learn features. However, this can quickly lead to an increase in complexity. Another way to reduce pixels when dealing with high dimensional images is explicated in [25], which looks at matching points from different samples - a similar process to [14]. The idea is that even in transformations we can find corresponding 'pixels' that are important but where the important pixel that corresponds between features is a different one but refers to the same information of the image. This works by using a sparse semantic learning network called SSL-Net utilizing region-to-whole learning to improve the accuracy of learned features. This enables semantic learning even from 'sparsified' representations, improving recognition of interrelated features. This is useful if an image has millions of pixels or if a dataset is noisy. However, it does not allow us to understand individual pixels importances. In [26] on the other hand, the proposed model architecture is an explainable neural network with local optimization metrics and structural gradients for neuron ranking. This approach incorporates the principles of XAI similar to [15] but into the network to enhance its efficiency and interpretability. The

idea of ranking neurons and not input data, reflects the classical paradigm of explainable AI, by turning one neuron off we can see how the model's accuracy changes. This is partially our goal, but with input pixelsm not neurons.

Many more systems like XAI exist. In this section, we explore a variety of tools for explaining machine learning models, highlighting key differences in our approach. One such tool is Layer-wise Relevance Propagation (LRP) which backpropagates relevance scores to the input pixels, first introduced by [27]. Additional gradient based methods which compute gradients of outputs with respect to an input image are sensitivity analysis systems like [28] and [29] which compute the average gradients from a baseline image to an input image. A method that similarly uses gradients of neurons in the network is saliency maps [30], and [31] which use graph based structures to represent image regions. Activation maps on the other hand provide a more granular approach visualizing activations in specific layers of a neural network. Class Activation Mapping (CAM) uses global average pooling identifying important regions for the model's prediction, and Grad-CAM extends this by including gradient-based information, for more accurate activation maps. In our work, we build on this idea, with the inclusion of a causal model agnostic methodology. Tools that use a similar perturbation based approach exist, specifically they focus on occlusion sensitivity which occludes different parts of an image and measures resulting changes in the model's prediction [3]. Another method that uses a similar approach on a specific sample is explicated in [32] called RISE (Randomized Input Sampling for Explanation), whereby the algorithm generates several occluded versions of a given image by applying random masks, and then calculates the average effect of each pixel's occlusion. This provides a probabilistic estimate of important regions. Similarly, [33] segments pixels into superpixels or groups and evaluates their corresponding performance. However, this does not use a modular patch-based approach and does not use both training and testing data as well as applying this causal approach to multi-modal data.

Another method is LIME (Local Interpretable Model-agnostic Explanations) which per-

turbs an input image and then builds a local surrogate model to explain the prediction [34]. However, LIME is in fact a new linear model that nests itself on top of the existing model. Generally, several methods like this exist that sit on top of the data rather than the model itself. Specifically ensemble methods like Random Forest [35] which use permutation feature importance, measuring the decrease in model performance when values of a specific feature are shuffled randomly. We propose a similar approach extending to multi-modal data in part 3 of our experiments. Unlike LIME and Random Forest, Shapley values [36] use cooperative game theory to find the importance of individual pixels, which is computationally heavy, but works well to come up with explanations for specific input pixels however does not scale for high dimensional image data.

Several of the existing tools leverage an internal methodology specific to neural networks and leverage the model's structure and gradients. The model-agnostic approaches are computationally intense and generally work with individual examples or features or do not provide extendability to multimodality. With our causal pixel perturbation approach which is model agnostic, we can work with training and testing data, employing varying patch sizes and sliding windows, leading to potentially a more robust understanding of the workings of a complex convolutional neural network. We also extend this work to multimodal data within knowledge graphs.

2.3 Architecting Semantic and explainable systems

There are several other approaches that aim to capture meaning as part of new model architectures. Unlike perturbation methods, Attention mechanisms assign different weights to pixels and regions in images based on their relevance in a task [37] and spatial attention specifically [38] focuses on important regions of an image to improve model performance in tasks like image captioning or object detection. However, the complexity becomes so large that the relation between what the model places attention to or what the model

sees as initially important regions does not necessarily correspond to what is important in later stages of the network. In [39] the authors propose a transformer-based neural network exploring similarities between samples. The model employs attention mechanisms to learn more complex dependencies in data. The model learns representations by understanding how different samples in a batch are related to each other rather than each sample being treated as an isolated case. Similarly, we could also use batches and compare within them rather than the whole dataset. More explicitly, the authors in [40] propose a model architecture which includes using CNNs with ACE for extracting the concepts of interest, where multi-resolution segmentation and clustering are used to rank relevant concepts. The authors chose to restrict their investigation to 100 classes and 50 images from each class, which could be a relevant approach for this research. However, it does make us question the validity of this approach as more images and categories would have added to the paper's credibility. Rather than capturing areas of interest to capture additional higher-level meaning, researchers have also looked at turning images into a kind of sequence 'voice' using a PixelRNN architecture, a two-dimensional LSTM layer, for detailed image generation [41]. It could be insightful to create a model that captures multimodal signals of data, hence capturing additional semantic meaning that is lost when feeding input in parallel. Being aware of the workings of how the human brain constructs semantic meaning is crucial, such is investigated in the architecture proposed by [42]. The model includes a generative episodic memory structure and mechanisms for semantic completion relevant to the hippocampus's function of recording and recalling incomplete memory fragments. The presented concept involves semantically integrating episodic and working memory networks to improve memory concretization and recall. This dynamic and temporal behavior is particularly useful to understand in the context of models that statically identify areas of importance. Finally, we look at a model that acts as a secondary layer on top of an existing architecture. The Semantic Learning Machine (SLM) neuroevolutionary algorithm is used to enhance and reform the seman-

tic learning process [41]. The SLM appears after a flattened CNN layer, determining whether a tumor is benign. The concept of adding additional semantic information or in this case a 'machine' after an initial model, is something we saw earlier in [21] with soft assignment from the output layer but has had limited success as it adds complexity to the architecture.

2.4 Key Takeaways from Literature

'Identifying Causal Links in Semantic Learning' seeks to identify early causal connections in data that a model can use to be faster and smaller. The studies we discussed navigate around this topic but do not explicitly investigate it. However, several key aspects can be learned from existing literature. First, we must consider the added complexity of algorithms for higher dimensional data (imagining millions of pixels), and should take inspiration from recursive assignment, or clustering methods employed by existing studies. We must also take inspiration from the architectures that embed causal methods to add additional semantic meaning to networks, perhaps experimenting with such makeups. We must also consider the temporal and dynamic nature of semantic learning which several algorithms fall short in [21, 25] and some mention and experiment with [22, 42]. As we saw, existing literature falls within one of three parts in the development of algorithms and many researchers tend to delve into additional complexity, rather than attempting to reduce complexity. Explainability also seems to be an important factor for many studies, which aim to use their methods to improve understanding of how models build up semantic meaning. The depth of research is large; however, we saw that simple methods like mix-up are popular among researchers. This begs the question if identifying simple causal links in data can ultimately enable us to reduce pixels, and consequently the layers of a network, reducing overall complexity while maintaining the same accuracy, and understanding the true nature of how algorithms and brains capture semantic meaning.

2.5 A note on LLM Applications

Understanding causality and the emergence of semantic meaning becomes particularly significant with the advent of applications that use Large Language Models (LLMs). In the field, LLM applications typically look somewhat like this:

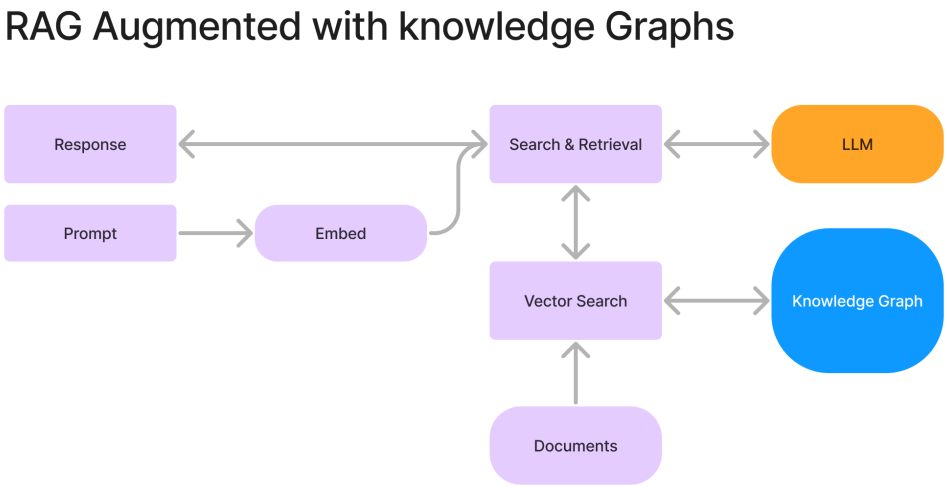


Figure 2.1: LLM Applications rely on well maintained knowledge Graphs

Applications have a knowledge graph that continuously grows in size as the user interacts with the system. For instance, consider a large language model (LLM) designed for mental health applications where user input continuously generates new data by interacting with the system. The LLM is made to generate data and we want to capture these interactions. With each user session, the knowledge graph will increase in size. The question then becomes how to store this information most efficiently in the graph. There are two classes of solutions. The first is that of entity resolution and de-duplication which

revolves around dropping similar data between nodes. We focus on the second technique which revolves around graph sparsification algorithms. The technique of spectral sparsification approximates a graph with a sparse graph that retains the most important spectral properties. Notably, Spielman and Srivastava developed nearly linear time algorithms for spectral sparsification [43, 44]. Another method is known as Graph Convolutional Networks (GCNs). Techniques like sparsified GCNs can prune unnecessary edges without significant information loss [45]. The other domain is random sampling which encompasses techniques of sampling edges where the sampled graph remains a good approximation. [46]. We take the approach of using a combination of machine learning (to identify important aspects of the graph) in a multimodal setting and causality to identify parts that can be removed.

The graph's purpose is to have information that can be quickly retrieved and accessed for providing more personal content to the user with information from potentially other users (which could be said to be other nodes in the network) all with multi-modalities (text, images, maybe voice and video). Using the causal perturbation-based methodology we outline in section 3, we show that we can delete information that is not relevant to this recommendation task without changing the ability to recommend. For the multi-modal case, we can also use this iterative approach inspired by the perturbations of earlier experiments in image data.

Note that recommendation is only one potential task, this is not limited to it but can also involve distinguishing between classes or between information within the user's realm.

3 | Methodology

This chapter introduces all components used in the experiments, including models, tasks, datasets, techniques and metrics. Our experiments use different architectures and algorithm variations. The experiments primarily revolve around MNIST. We report on accuracy as well as standard classification metrics for evaluating the models performances. We provide extensive details on the experimental setup and resulting pipeline which consist of three distinct components:

1. **Find pixel or patch importances, exploring different methods using testing and training data**
2. **Using the found importances to train and run new models**
3. **Use the found importances in combination with distances and knowledge graphs to reduce data used**

Below you can see an overview pipeline for our method:

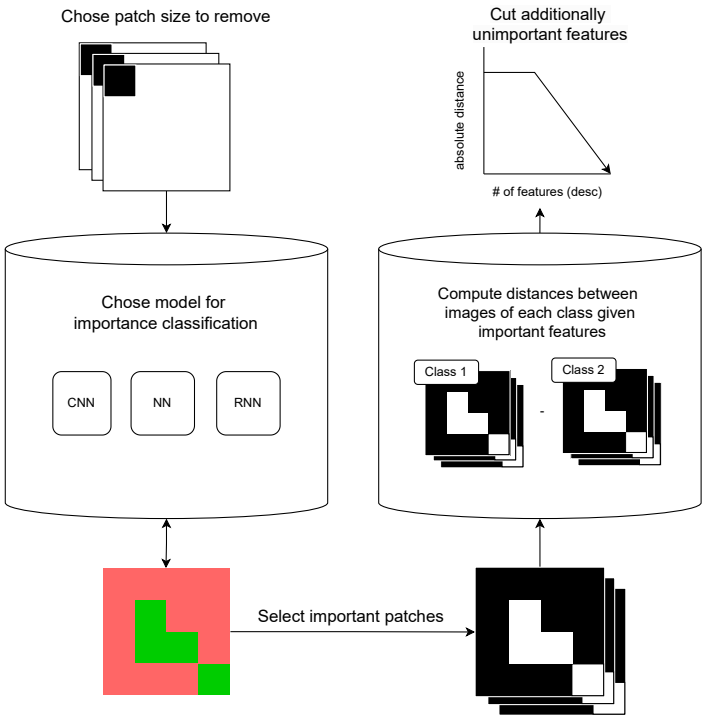


Figure 3.1: Overview of Pipeline

Understanding 'how much data is enough' in the context of our study requires thinking about what 'enough' means. For a given model we can measure this with accuracy, whereas for a graph structure we can look at distance measures. With our method we aim to understand the minimum information needed to maintain the accuracy of the model (part 1 and 2) and distance of a graph (part 3). To do this we need to understand what parts of the data are more important and why. The causal approach we chose is based on perturbations and as explicated before, has several advantages over other methods. Our method begins by choosing a patch to remove in a given image. We use the remaining image to train or test a model and return corresponding results. The way we chose patches and what models we use is explicated in 3.3.2. Once we have an idea of which parts of an image are more useful than others, we can use different methods to re-train a new model 3.3.3. We can also use those parts to determine differences between

images as explained in 3.3.4. This saves computational resources as the important parts contain the most relevant information. We can then look at changes in differences to see how many of the important features we need to retain without losing information. We can also think about other potential model outcomes by looking at differences between images, such as new causal models.

3.1 Models and dataset

In this study, we utilize a CNN among other architectures like simple neural networks (NN) and re-occurent neural networks (RNNs). We show below the architecture for the CNN, NN and RNN used in our experiments. We use 5 epochs for all models respectively. Note that we do experiment with other variations of these baseline architectures, however we rank algorithms by time first and then take the best performing variants for each model architecture, with the following being ranked first.

3.1.1 Convolutional Neural Network (CNN)

The following the describes the network architecture of the CNN used in this study.

Layer Name	Configuration
Input	Image of shape (28, 28, 1)
Convolutional	`Conv2D(32, (3, 3), activation='relu')`
Pooling	`MaxPooling2D((2, 2))`
Convolutional	`Conv2D(64, (3, 3), activation='relu')`
Pooling	`MaxPooling2D((2, 2))`
Flattening	`Flatten()`
Fully Connected	`Dense(128, activation='relu')`
Fully Connected	`Dense(10, activation='softmax')`

Table 3.1: Convolutional Neural Network (CNN) Architecture

Note that for CIFAR-10 we use a different CNN, we explicate this further in the next

part of the methodology.

3.1.2 Neural Network (NN)

The following shows the network architecture for the NN.

Layer Name	Configuration
Input	Flattened image of shape (784,)
Fully Connected	`Dense(800, activation='relu', input_dim=784)`
Fully Connected	`Dense(10, activation='softmax')`

Table 3.2: Neural Network (NN) Architecture

We use this NN to show differences in the important pixels used by this model in comparison to the CNN.

3.1.3 Recurrent Neural Network (RNN)

We use an out of the box Simple RNN from tensorflow.

Layer Name	Configuration
Input	Flattened image of shape (784,)
Reshaping	`Reshape((784, 1), input_shape=(784,))`
Reccurrent	`SimpleRNN(32)`
Fully Connected	`Dense(10, activation='softmax')`

Table 3.3: Recurrent Neural Network Architecture

Note that we use the RNN exclusively in experimentation with MNIST.

3.1.4 Model loss

In all models, we utilize categorical cross-entropy loss (CCE), also known as log-loss.

$$\mathcal{L} = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})$$

where:

- N is the number of samples.
- C is the number of classes.
- $y_{i,c}$ is a binary indicator (0 or 1) if class label c is the correct classification for sample i.
- $\hat{y}_{i,c}$ is the predicted probability of sample i being in class c.

A perfect model has a cross-entropy loss of 0. The CCE gives a high penalty to values that are confident and wrong, helping the model distinguish effectively between false and correct assumptions.

3.1.5 Datasets

For the experiments in this research, we use MNIST, which consists of 60,000 training images and 10,000 testing images, each of which is a 28x28 grayscale image containing a digit between 0 and 9. For the following experiments we only use 10% of training data.

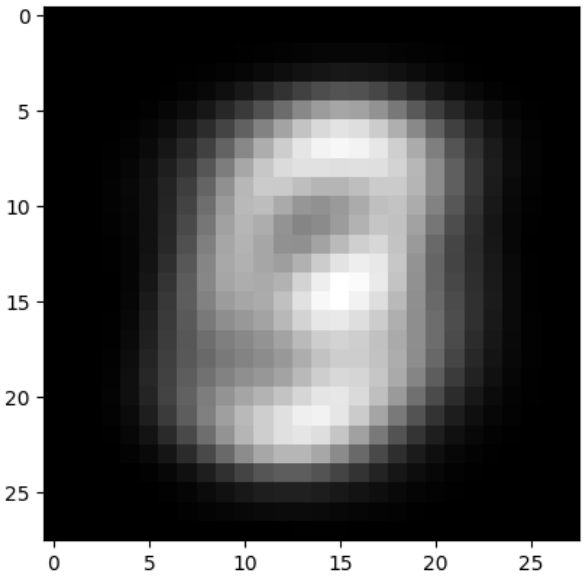


Figure 3.2: Average Pixel Value MNIST

Above you can see the average pixel value for examples from MNIST, which are centrally located in the image.

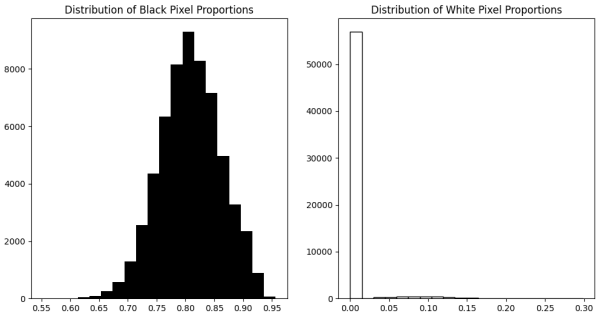


Figure 3.3: Average Pixel Value MNIST

Above you can see the intensity of white and black distributions in the dataset. This can help us see how changing pixel values to either extreme may impede the models ability to distinguish and recognize important pixels. The dataset contains many black pixels with high intensity and only a few very white pixels.

To verify our investigation, we also make use of CIFAR-10. The CIFAR 10 dataset includes 60000 color images sized 32 by 32 pixels categorized into 10 classes with 6000 images in each class for training purposes and another batch of test images consisting of 50000 training images and 10000 test images, in total. The categories from 0 to 9 are: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, Truck. With CIFAR-10 we have the added complexity of dealing with colors. As we will see later, we adjust our methodology to work for this use case. We can see the average distribution of pixel intensities for all colors below.

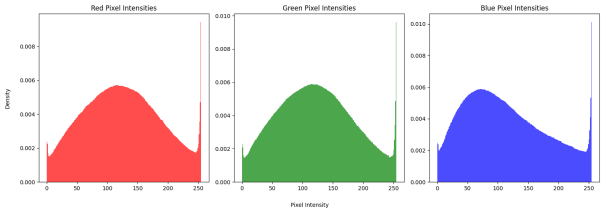


Figure 3.4: Average Pixel Value CIFAR-10

We can see that the distribution for blue is slightly skewed to the left but all images seem to contain all three colors. This means we do not need to make special considerations on this matter. Unless explicitly stated, we will continue to use MNIST for this investigation.

3.2 Metrics

We use the following metrics for our study. We focus on accuracy as it is particularly useful since the classes of the dataset are balanced and it provides a intuitive reflection of how frequently the models predictions are correct.

3.2.1 Metrics of Accuracy and Euclidean Distance

Accuracy:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

.

Pixel Importance:

$$\text{Importance} = 1 - \text{Accuracy}$$

.

Euclidean Distance:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

.

- $d(x, y)$: The Euclidean distance between points x and y.
- x_i: The i-th component of the vector x.
- y_i: The i-th component of the vector y.
- n: The number of dimensions.

3.2.2 Distance Convergence

Distance convergence refers to the stabilization of distances between data points as more features are considered. Convergence Criterion: $\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} d(x_i, y_i) = \text{constant}$.

3.2.3 Knowledge Graph Active Learning

The aim of this is to determine if additional data is required by looking at changes in the Euclidean distance between nodes in the graph as new information is added.

We define the change in Euclidean distance as follows:

$$\Delta d = \mathbf{v}_{new} - \mathbf{v}_{old} \ .$$

Where $\mathbf{v}_{new}$ and $\mathbf{v}_{old}$ represent the vectors of node features before and after adding new

information, respectively. We also define a **Decision Criterion:** If $\Delta d \approx 0$, then this additional data is not needed as the additional information does not significantly alter the graph structure. Alternatively, we can also look at perturbing examples (selectively removing parts of sentences or words (for example in a multi-modal graph setup) and figure out if this leads to a change in distance.

3.3 Experimental Setup

Our main experimental architecture has three distinct parts. These sections are described below. Note that the compute for part 1 consists of 4 vCPUs and 16 GB memory. For part 2 we use a 13 GB RAM, 110 GB Disk Space, Python 3 Google Compute Engine.

3.3.1 Preliminary Experiments

Initially, we experiment with random and Gaussian sampling. We ask the question: does the model work well with random and Gaussian sampling? We expect that for Gaussian sampling in particular, this will work well but may not extrapolate to more complex datasets where the content of images is not in the center.

3.3.2 Part 1: Importance of Pixels or Patches in an Image

In this part, we aim to find the importance of pixels or patches in an image. For this, we use 10% of the training data.

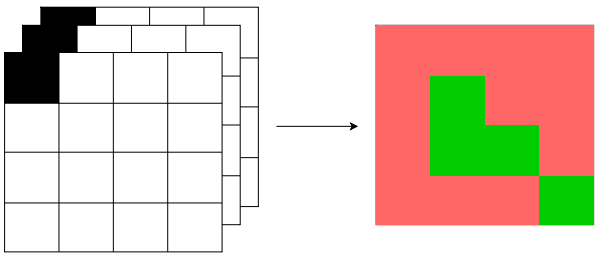


Figure 3.5: Overview of Part 1

Our goal is to go from analyzing individual pixels or patches to having a heatmap representation of what areas in images and in images of a given class are more important than others. In the context of causality, we ask what the effect of changing or in our case turning off one part of an image is on the accuracy of the model.

We approach this in two ways:

1. **Sliding Window Method:**
    - Slide through the relevant sections of the image.
    - Turn each section off by setting it to zero.
    - Train a CNN with examples from the MNIST dataset where the selected section is set to zero.
    - Report the importance of each section based on its impact on performance.
2. **We can do the same approach with the testing data, without the need to re-run a new model every time. This considerably reduces the computational and time complexity.**

The pseudo algorithm for the overarching approach (EvaluatePatchImportance) is detailed in the algorithms section. The complexity of this algorithm is attributed to the number of pixels or patches the algorithm has to go through, which is given by the following: **Patch Size Calculation:**

- Given an image of size $n \times n$ and patch size $p \times p$:
- Number of patches and complexity:

$$\left(\frac{n}{p}\right)^2 \approx O\left(p\right)$$

We repeat each method 3 times to receive an average importance value for each patch.

We then use these in part 2.

3.3.3 Part 2: Re-train with important pixels or patches

In this section, we go from having found important parts of images to using those important parts to feed into a new model.

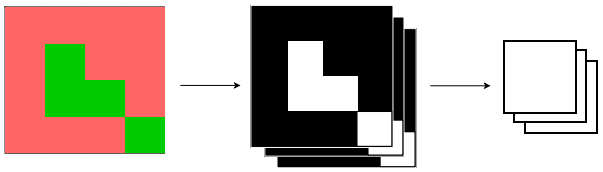


Figure 3.6: Overview of Part 2

We use only patch size = 1, to receive maximal granularity and only 10% of the training data. This section details two components:

1. **SmartPatchV1**

    - This method takes in a given number of important pixels and
    - runs a model for this number of pixels with all other pixels are either (a) in White, (b) in Black, (c) sampled randomly.
    - Repeats this process for different pixel amounts: 700, 600, 500, 300, 200, 100, 50, 20
    - comparing outcomes for selecting the (a) most important, (b) least important (c) random selection of pixels.

    Note that we have two versions for this (a) a loop version - which looped through all examples and (b) a no loop version which auto-sets the relevant pixels to true or false.

2. **SmartPatchV2**

- This method takes in a given number of important pixels and
- Calculates the shape of a new patch that only involves those pixels and
- runs a model with only those pixels as a new patch
- comparing results between (a) the most important pixels selected, (b) the least important, (c) the same random selection of pixels, (d) a different random selection for each sample.

For both versions, we take the average across 5 trials and report the standard deviation. Note that we also compare initial results to selecting the most important pixels for each class for both versions. To illustrate the previous explanation let's visualize what we mean:

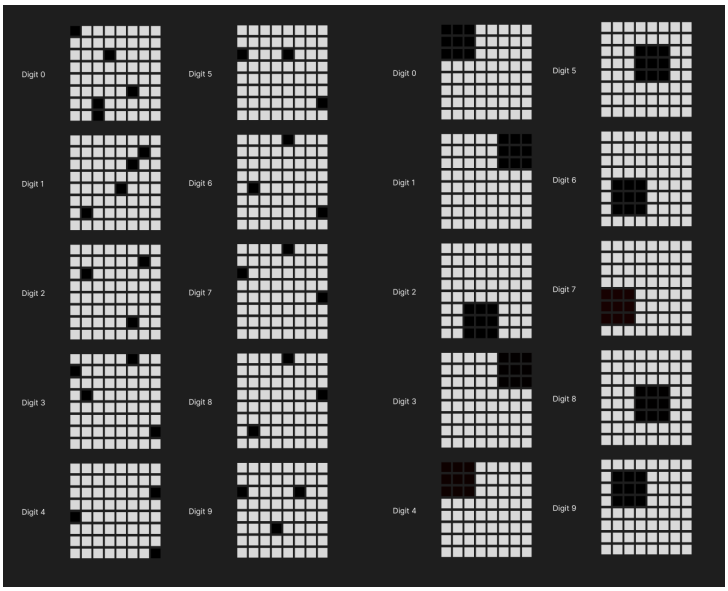


Figure 3.7: example of important pixels per class

As we can see Part 1 of our experimental setup returns a list of important pixels or patches per class. We can then combine these in different ways to train a new algorithm. See below:

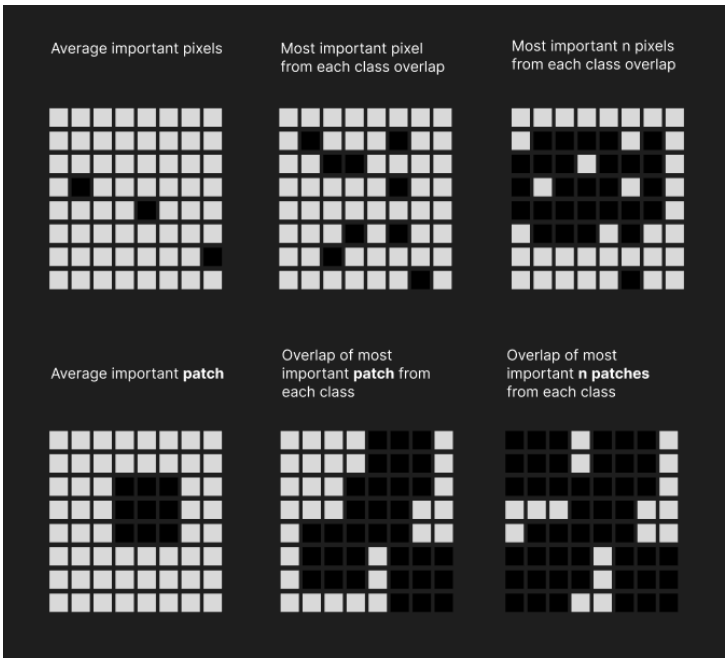


Figure 3.8: Options

We can take the average of important pixels, the most important pixel or pixel(s) from each class, and the same works for patches. We can then decide to mask each image with this option or have separate masks for images of each class. In our case, we look at the average important pixels and keep the same mask across cases. Whereas we do experiment with changing n important pixels per class, and random selection.

3.3.4 Part 3: SHADOW: Distance Learning Methods (Shallow Distance Learning)

Finally, we use the found importances to calculate distances between important pixels of the given dataset, by subtracting important pixels of images from different classes.

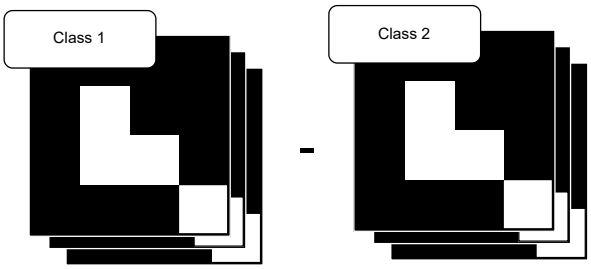


Figure 3.9: Overview of Part 3

Other than re-training an existing machine learning model on important pixels we can use those important pixels to compute distances between classes, effectively having a way to compare classes in a time efficient way. This experimental section contains 2 parts:
MNIST. We first look at MNIST and compute the Euclidean distance between important pixels of examples from each class. We use this to construct a sort of distance graph. We vary the number of important pixels used to see if "convergence" happens, as described in section 3.2.2. For additional details refer to the algorithm in 3.4.3. The complexity for the graph algorithm is given below.

CreateDistanceGraph Complexity:

- Total complexity: $O(C^2)$, where C is the number of classes.

MULTIMODAL MNIST. Here we look at the same approach as before but with the added modality of sentences that are connected to examples in the dataset. We start with all data used (in this case we assume a node not to be a class but a collection of text and image for a given user) and reduce the number of features used, investigating if we can reduce the number of features without hurting the total distance.

4 | Experiments and Results

In this chapter we analyze the results of our experiments, starting with 1. Reporting and comparing results for pixel importance methods. 2. Reporting results for V1 and V2 of SmartPatch and 3. Looking at results for initial multi-modal graph-based experiments. In our pursuit to understand the minimum information needed, we looked at many scenarios and cases, here we boil it down to key findings to stimulate discussions on causality and new frugal machine learning systems.

4.1 Preliminary Experiments

To understand the minimum amount of information needed we started by experimenting with using random and Gaussian sampling to sample pixels from a given image, and use those samples to compute accuracies. We saw quickly that this works well in both cases. See below for a visualization.

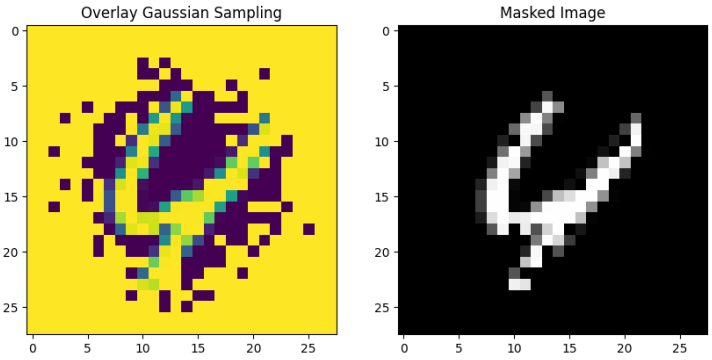


Figure 4.1: Performance with Gaussian noise

However, in the Gaussian case, we realized that this approach would be limited to MNIST as we know that images are centrally located, which is not the case with other more complex datasets. We proceeded to implement random sampling in our main experimental setup.

4.2 Evaluation of importances

To extract pixel importances we used 10% of training data, for further details on experimental setup see methodology section part 2. The following shows the results of pixel importances for corresponding patch sizes where training took place as outlined in part 2 of the methodology section. We train the corresponding patch number of CNNs turning off the relevant pixel, and report the importance score. We show below the result of 3 repeated experiments on the training set for patch sizes (= 1, 2, 7, 14):

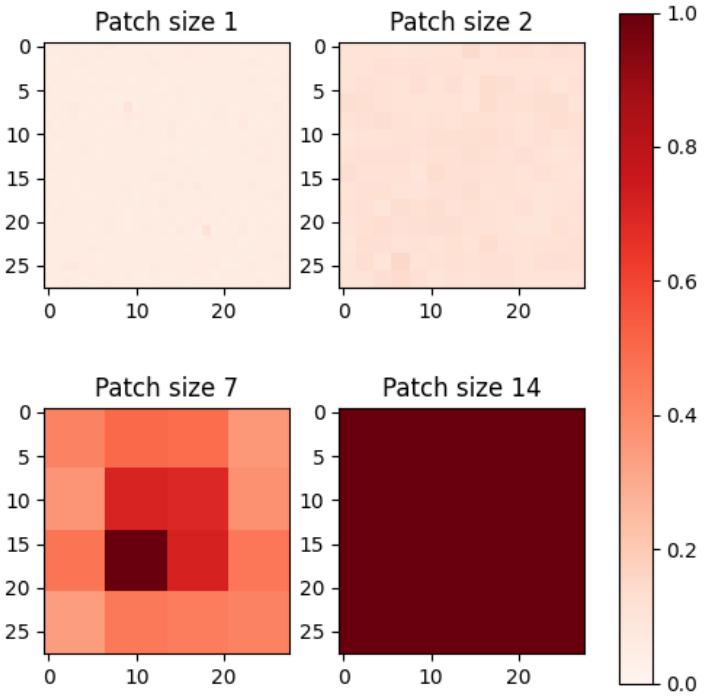


Figure 4.2: Overall Important Patches on training data from CNN

We can see that as we reduce granularity of the patches, they become more important, i.e. as the patch size increases, removing a given patch reduces the accuracy more. The most 'explanatory' patch size is in this case 7. As explicated before, we then look at perturbing examples in the test set, we can do so for the entire test set, turning pixels

off one by one, and reporting the corresponding importance score.

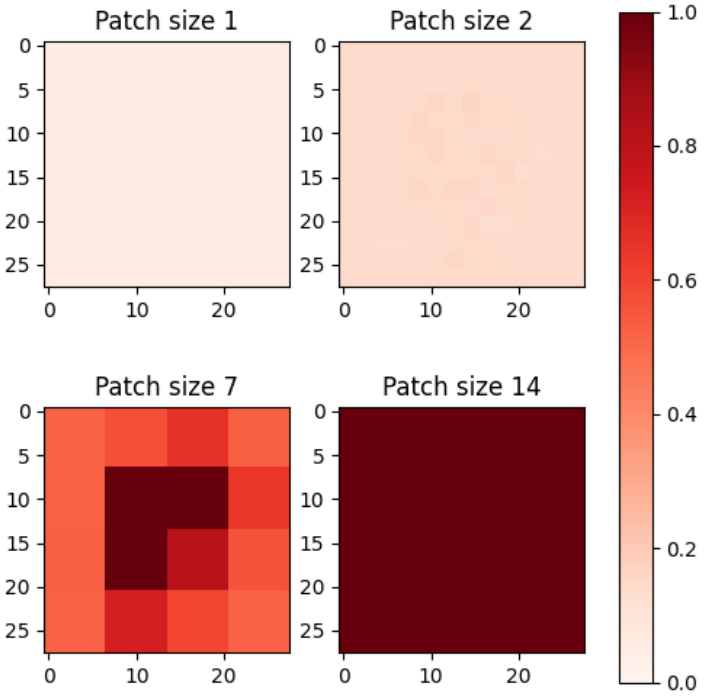
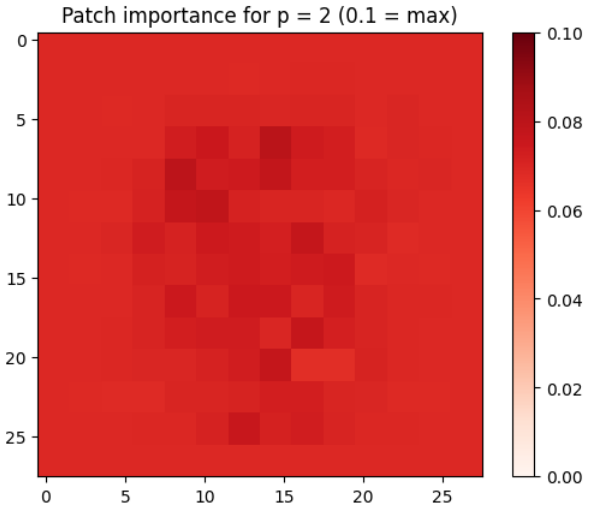


Figure 4.3: Overall Important Patches on evaluation data from CNN

As seen, Patch size 7 gives the most evident result showing the importance of the central region. The other patch size sub-graphs do not show much difference on this scale. As we increase granularity it becomes harder for the model to see differences in important pixels in the test set rather than training. Especially for patch size = 1 and 2, we can see fewer differences than in the results using the training data setup. However, if we experiment with the maximum scale, i.e. reduce the maximum we can start to see the granularity on a finer level, see below.



Albeit this is more informative than before and an important additional mechanism for Explainability in our toolkit running faster than the training set (we do not need to re-train numerous CNNs), because the training method seems to show slightly more robust results, we continue with this method for subsequent experiments.

As can be seen, we can also look at differences in important pixels and least important pixels both overall and per class, we found that using this method leads to either over-fitting using V1 or random results using the V2 algorithm of only patches. Below are the 10 Most and Least Important Pixels per Class from the Patch=1 CNN model.

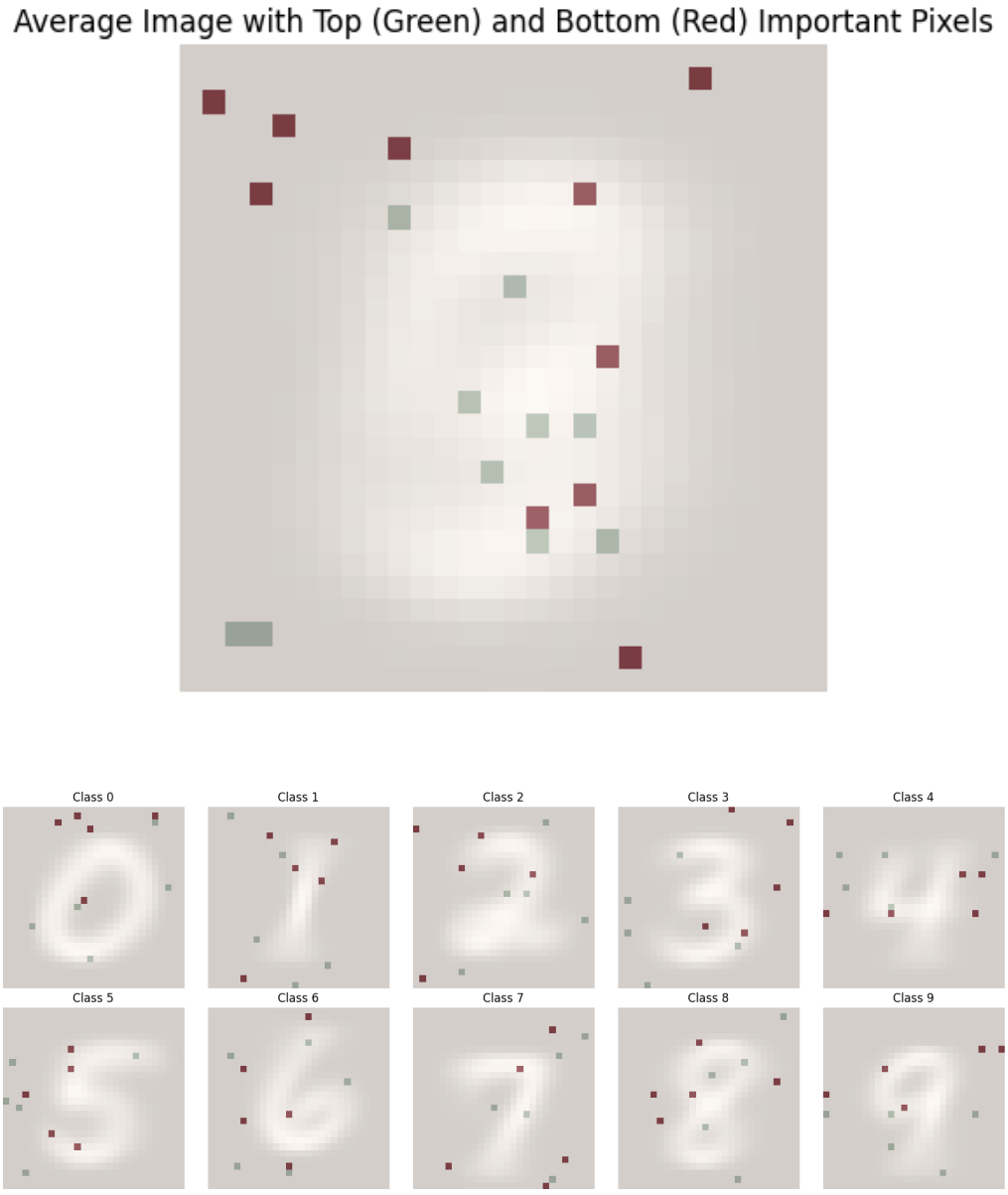


Figure 4.4: Average importance pixels per class

As can be seen, the model frequently shows pixels in key locations as important which have higher average pixel values, but also highlights potential regions where it recognizes the lack of high pixel values to make a correct classification. The model also seems to have unimportant pixels in 'surprisingly important spots', illustrating the potential

complexity of the convolution, to be analyzed further in the discussion section. Note that for more than 10 most important pixels please explore our interactive pixel importances website detailed in Appendix 1.

At this point, we can also ask, whether another CNN architecture or network performs better or differently. As we see in [10], many deeper CNN architectures exist, and deep neural networks that provide valuable performance. In the NN B for example, the heatmaps seem slightly clearer than in the convolution. Whereas in the RNN, the heatmaps are not very indicative of important parts of images, but rather show how the model works giving more importance to patches that are fed into the model later.

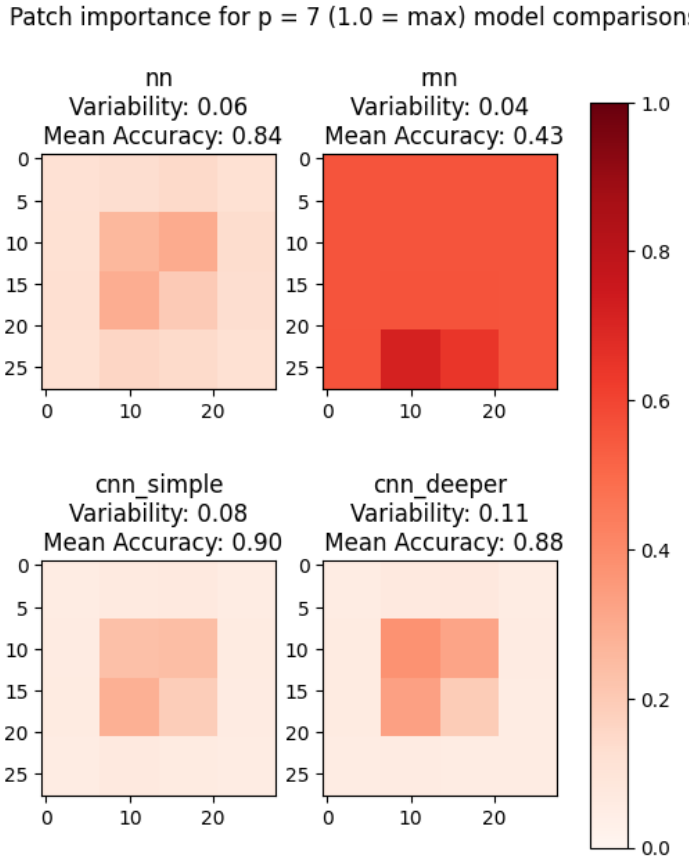


Figure 4.5: Overview of important patches for different model architectures

We can see here that the NN gives more value to patches in the top and bottom (which tend to have zero values in the dataset), whereas the CNNs both exclusively focus on the central region. Note that the deeper CNN has an additional Convolutional layer of size 128. We can see that even adding another layer increases the distinguishability of patches, as the top left patch in the deeper CNN is more important now than it was before. Additionally, the accuracy of the NN and the CNNs is above .97 respectively, whereas the RNN has an accuracy of 0.54. To what extent the accuracy of the model impact the heatmaps and when the given accuracy is high enough to allow for distinguishable heatmaps is a open question. Clearly, the RNNs accuracy is not high enough to provide results that match our expectations. On the other hand, the NN, CNN and deeper CNN all have a high accuracy which is not indicative of which model is "better" at conveying importances of pixels. It is an open question thus, which accuracy is needed to allow for trustworthy results. To help with this we can calculate what we call Variability which is the average difference between patch values, and the mean accuracy, which is the mean accuracy of all patches. The Variability can also be seen as a standard deviation and allows us to see if the model attributes high importance to some patches. For example, the deeper CNN has a patch that is darker than the others. The mean accuracy can help us gauge how "robust" the model is, giving an indication of how much the model relies on specific parts of the image when they are removed. For example, the simple CNN (architecture is outlined in 3.1), has slightly higher mean accuracy. While these metrics are based on our patch-based approach, we urge users of our library to use them with caution.

Finally, we draw attention back to the ideal patch size. We can see that with a patch size of 7 like above, we receive decent detail over what patches the model considers important. However, when we look at patch sizes 1 or 2 the model provides unexpected results. Only the per-class breakdown provides interpretable results. Patch size 4 B on the other hand, provides an interplay between granularity and explainability. However, in the case of

patch size 4, the model attains a high value to the central bottom right region, whereas in patch size 7 the region in the top left seems more important. How well the found importances thus translate between patch sizes is an open question.

4.3 SmartPatch Experiments

As explicated in the methodology, we can use the results from importances in the SmartPatch experiments, looking at several setups to identify the robustness of using important pixels in training new models. We use patch size = 1, equal to 1 patch per pixel for maximal granularity. See below for results using SmartPatchV1 which selects either important, unimportant or randomly sampled pixels and turns all other pixels into either (0, 255, or random values). The following 9 experiments reflect these combinations while selecting different pixel amounts (20, 50, 100, 200, 300, 500, 600, 700) across 5 trials reporting on accuracy. We use the CNN architecture as explicated in methodology section 3.1.

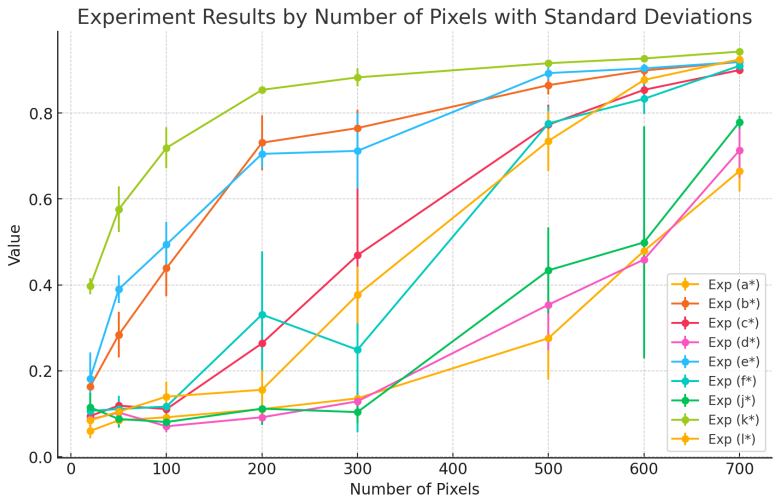


Figure 4.6: SmartPatchV1 Results

Following this order.[1]

We can see that in all configurations when random or white is turned on, the model evidently has difficulty distinguishing between important pixels. However, in experimental setup (b*) we can see that the model does well, even for the unimportant and random pixels the model does well when the other pixels are set to black (e*) and (k*) respectively. Even when selecting only half of pixels we receive accuracies above 90 percent. Yet, the random case outperforms the important and unimportant cases, indicative that this method adds some form of variability that helps the model generalize. To investigate this behaviour further, we look further at Smartpatch v2, where we train with only the selected pixels. We also report here that for training with class-based important pixels we receive close to random results (0.1) in all trials, and thus decided to keep this out of further investigation. Perhaps, the neural networks are not suitable for using data that is too variable (different pixels are selected for examples from each class), thus making it difficult for the convolutions to learn connections, which we discuss further in section 5. We saw already in 4.4 that different classes have different corresponding important pixels.

Below are the results for SmartPatchV2.

[1] a* most important, rest in White, b* most important, rest in Black, c* most important, rest in random, d* least important, rest in White, e* least important, rest in Black, f* least important, rest in random, g* random selection, rest in White, h* random selection, rest in Black, i* random selection, rest in Random.

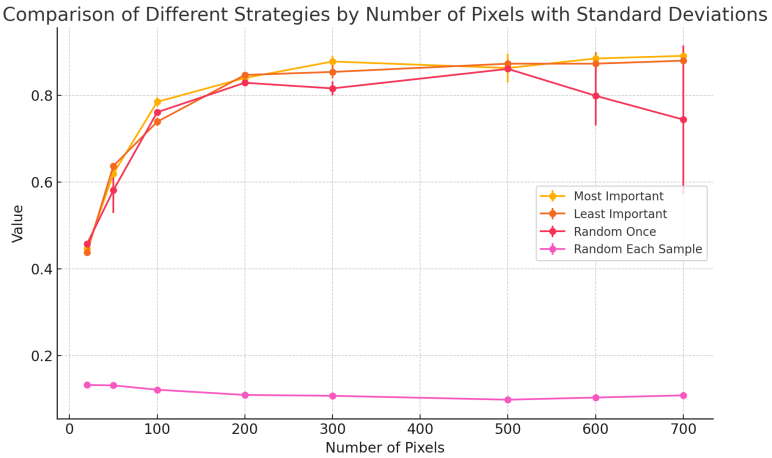


Figure 4.7: SmartPatchV2 Results

As can be seen, feeding the most important pixels into the CNN as a patch, provides high accuracy, with the added benefit of computational efficiency, where a model with 84% accuracy runs in 6.2s using only 200 pixels. The random for each sample setting here (as described before) unlike in 4.6, selects a different corresponding random number of pixels for each sample. The random case also consistently does worse than both the most and least important pixel configuration. To validate our results we now look at CIFAR-10.

4.4 CIFAR-10 and Time Complexity

With this dataset, we quickly realized that using the methodology to re-train a new CNN for each patch turned off will take days to train with our compute resources. Given the larger architecture for the CNN C.1, which boasts more layers, this would not be sensible. Due to this time issue, we decided to use the second option in the methodology we outlined, which is to use the test data instead of the training data, turning off a given patch in the test data and logging the resulting importance as defined in the methodology. As we also saw earlier, a patch size of around 7 provides a balance between interpretable and useful results. Since CIFAR consists of 32 by 32 images, we chose 8,

the closest divisible number. Note that the analysis below is no longer sorrounding how much information needs to be fed into the network initially for it to perform well, but rather, what information is important to the existing model to chose what information to keep in a downstream task.

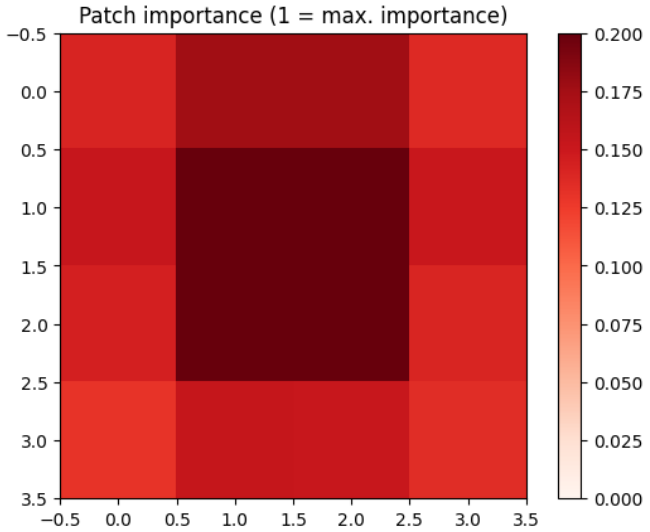


Figure 4.8: Patch = 8 - CIFAR evaluation data

We can see that like in MNIST, the most important patches seem to be concentrated centrally in the image. While there are important patches sorrounding the centre, clearly the most important ones are in the centre. With the size of the model, and the compute available, even calculating this takes considerable time. We can of course also look at a more granular level like a patch size of 1. Like before, this granularity while useful for the methodology, does not provide much visual insight. If we look at the per class breakdown of important patches for patch size 1, which you can find here C, we can see that some classes give high important too many pixels, whereas other classes have certain regions that seem to be important for the model. For example, class 5 has important patches located lower down in the image, whereas class 3 has important pixels further up. We can see here that using the test data is a real alternative and provides reasonably

fast and useful results. Finding the right granularity of patch size is then a question of purpose. If we use this patch importance method for interpretability, using higher patch values proves more useful, however, when we want to use this method for using important pixels in downstream tasks like in our SHADOW methodology, we should likely focus on taking the lowest possible patch values. Another question that we can ask is how well the granularity translates across patch sizes. For example, even though the patch size = 8 importances show high values in the central region, if we look at higher granularity like patch size = 1, we will not necessarily see the most important pixels in the centre, but perhaps scattered like with MNIST. In those cases, using the training data seems more robust.

To see how well our methodology performs we now consider the same experimental setup as SmartPatchV2 for MNIST. Like before we combine n pixels into a new larger patch that we feed into a model. We use the n most important pixels which we call experiment 0 (found from patch size = 1 using the evaluation data now). We compare this to selecting the n least important pixels (experiment 1) and n randomly selected pixels (experiment 2). We also look at the time complexity for all three experiments. Similar to before we report the average and standard deviation for 5 trials. Note that even in the best case, the accuracy is quite low as CIFAR-10 contains considerably more complex images than MNIST. We opted for this as we want to see if our method is stable with lower accuracies as well. We also note that the total accuracy is not as relevant as the relative accuracy between removing pixels and between experiments.

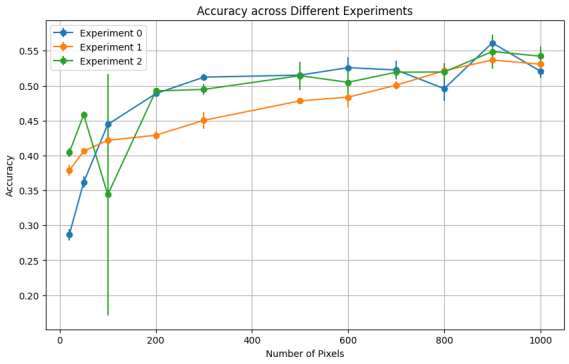


Figure 4.9: CIFAR-10 SmartPatchV2

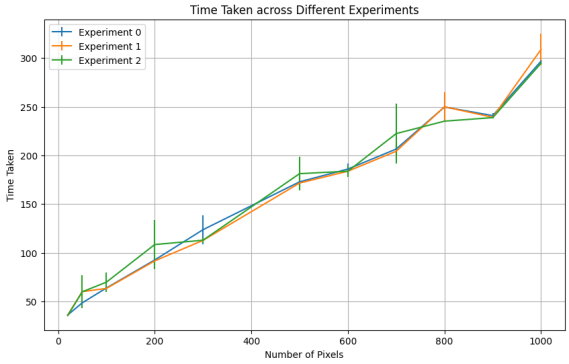


Figure 4.10: CIFAR-10 time taken

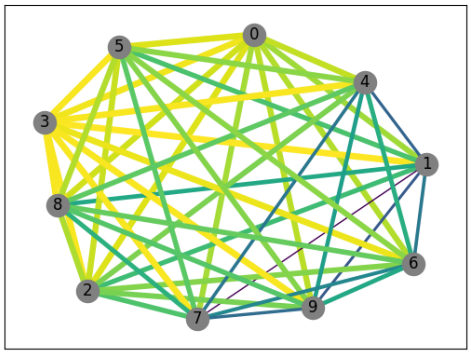
As can be seen, using the most important pixels is stable throughout and provides the highest results, outperforming both random and least important pixel configurations. While the properties of randomly selecting pixels seem desirable, they make the model less stable generally providing more variability. When the number of pixels used is high, the results converge, whereas when we reduce the number of pixels slightly, the results seem to be almost higher. This is possibly explained by the fact that if all pixels are used there is no selection happening, whereas if the number of pixels is high but not all, the model takes in a selection of the most important pixels. Along the way, we can see that taking the most important pixels outperforms taking the least important pixels, and is generally more stable than selecting pixels at random. In the case of selecting importances from evaluations based on the test data, however, we must consider not to overfit our model on the test data. An alternative would be to use the training data in the same way, and then use the test data in the final component. However, as we saw earlier, we would expect the results to be similar, albeit a surprise is possible. Additionally, we can see that the accuracy using the most important pixels decreases rapidly in the low amounts, whereas that for the least important pixels seems slightly more stable. Considering the values on either end of the spectrum should be done with caution as the model is likely not stable enough to make accurate predictions on the left side, and on the right side the value of the individual experiments diminishes. With respect to the

time complexity, we can see that this grows relatively linearly with the number of pixels being used. Overall we can say that using 500 to 400 pixels (roughly half of the original amount) results in less than half the time taken to fit the given CNN, with minimally diminished accuracy. Despite the limitations mentioned, this method has the potential to be used in production environments.

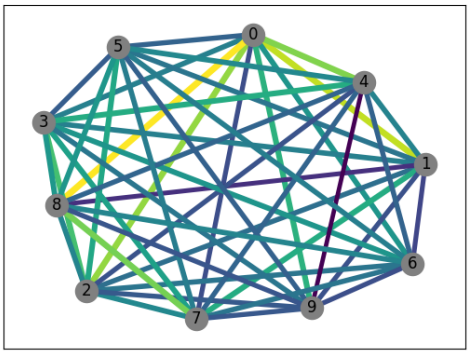
In effect, we used the idea of causality to turn given pixels on and off, defined our own importance criteria and then asked the question, can we use the found importances to train a new model that works faster. We have now shown that this is possible and we can reduce training speed by 50% by using the 300 most important pixels in the dataset. However, we can also ask if we can use the found importances to train a new causal type architecture, paving the way to the creation of new causal models. One such avenue is that of using the found important pixels to determine distances between important pixels of different classes in a time-efficient manner. Given our discussion on different pixel importances per class, we decided to focus on this more closely, determining whether we can use important pixels to determine overall class differences. We go beyond this to see how we can potentially use these differences to further determine how much data is really needed to maintain differences in classes. We look at this now.

4.5 Graphs and distances

As explicated, in the 3rd step of our methodology we can use important pixel configurations to look at class differences, computing the Euclidean distance between the important pixels of examples from each class. In the below graphics, the darkness corresponds to the closeness of classes, i.e. the darker a given line the closer two classes are.

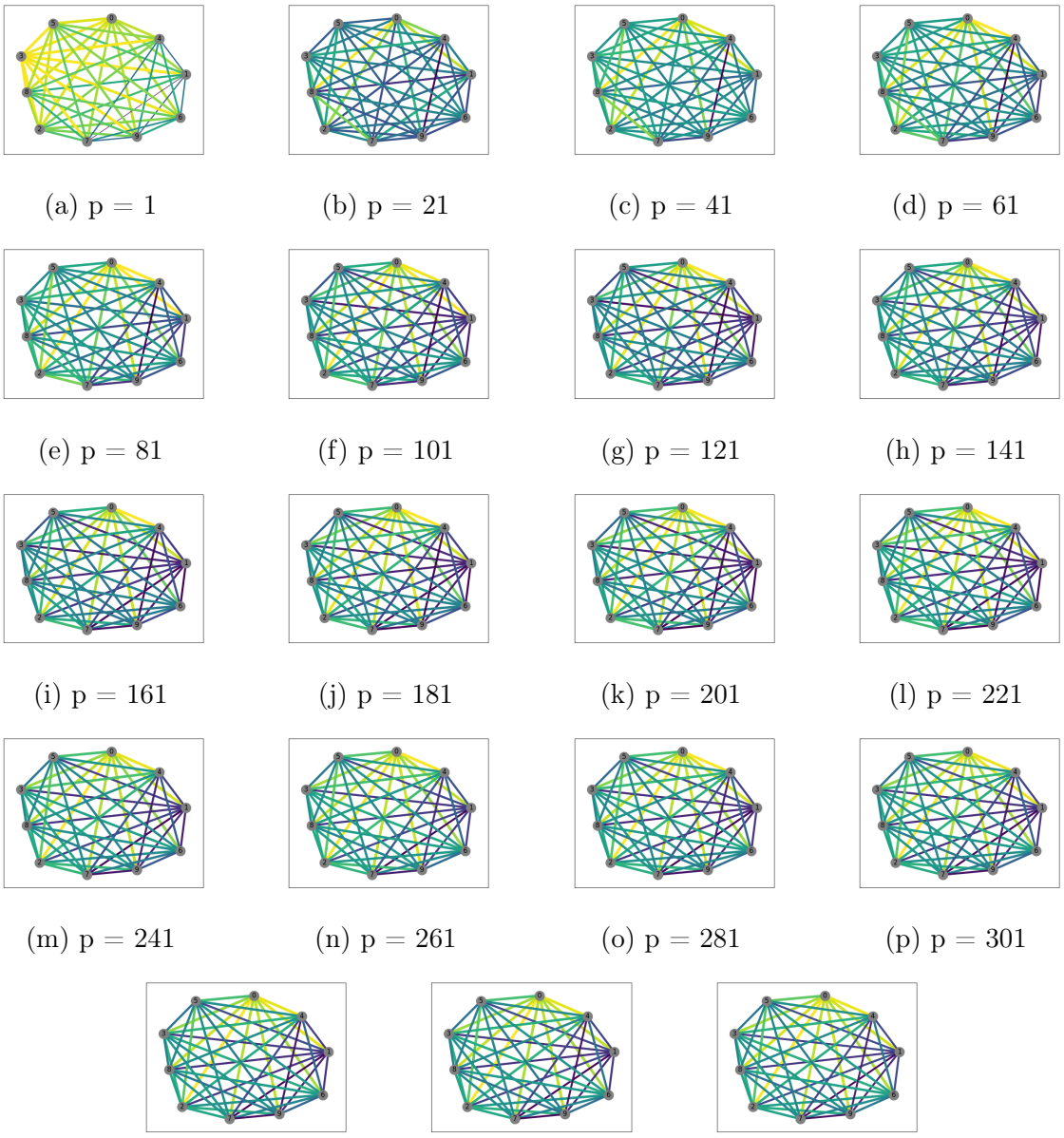


(a) p = 1



(b) p = 21

We can see that as we increase the number of important pixels for which we are computing distances, the graph changes color. We can do this up to a certain number of pixels, to see how these distances converge. Note that the equation for Euclidean distance is given in section 3.2.2.. Next we can see how the graph visually converges for different important pixel amounts, taken from the CNN patch = 1 configuration.



(a) p = 1 (b) p = 21 (c) p = 41 (d) p = 61

(e) p = 81 (f) p = 101 (g) p = 121 (h) p = 141

(i) p = 161 (j) p = 181 (k) p = 201 (l) p = 221

(m) p = 241 (n) p = 261 (o) p = 281 (p) p = 301

(q) p = 321 (r) p = 341 (s) p = 361

Figure 4.12: Graphs of distance between classes for varying important pixel (p) amounts where the distance between classes is represented by the color of the connection.

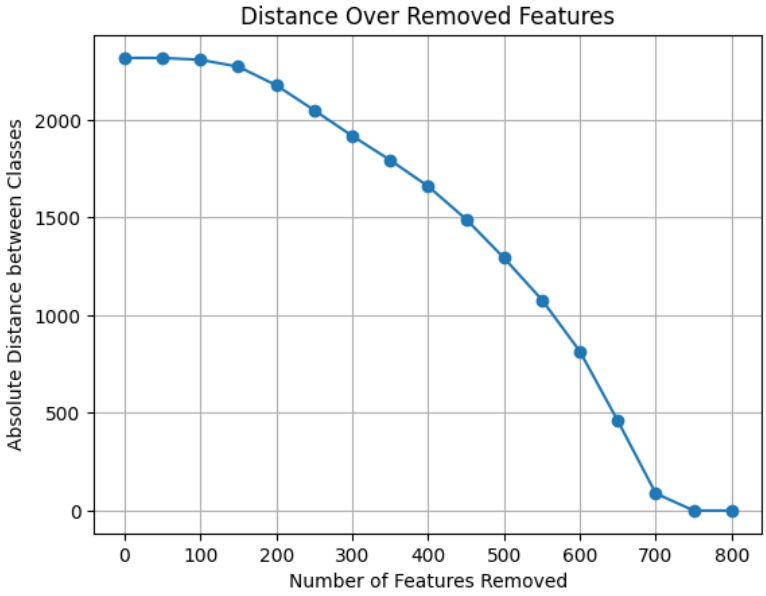
Again, the brighter the color of a connection the further apart is a given pair of classes.

We can see a convergence after around 200 pixels and can verify this mathematically

by looking at the average change in distances. We used this concept to create causal algorithms, some of which have existed in the past. These models provide accuracies of around 80% but provide hope that this direction is an avenue for further exploration. For the purpose of this paper, however, we explore the concept of distances in a multi-modal setting first in the next section.

4.5.1 Multi-modal distances

We can extrapolate the case of distances for multi-modal data. As explicated we generated text data corresponding to the image data. Here are the examples we match with the relevant images from each class: "The digit 0 can be found in the center of the image.", "In this example, the digit 1 is clearly written by hand.", "You can see the digit 2 at the top right corner of the image.", "Notice how the digit 3 stands out against the background.", "This image showcases the digit 4 in a bold font.", "The digit 5 appears slightly tilted in the middle.", "Here, the digit 6 is drawn in a lighter shade.", "Look at the bottom left to find the digit 7.", "The digit 8 is prominently displayed in the upper section.", "At the bottom right, you'll see the digit 9.". We use a total of 1000 examples for illustrative purposes. We encode both the image and text data into a space with 800 features. We then randomly remove a set of features computing the Euclidean distance before every removal.



We start with a baseline distance and can see that this distance stays constant for removing up to 100 features. We can thus reduce the size of the network without removing information on the closeness of classes. Noted, this technique can be applied with removing unimportant rather than random features, and for non-shared embeddings, which is further explicated in the next section.

5 | Conclusion

We introduced a novel causal model-agnostic technique to learn important parts of images and multi-modal data. This method can be used as an Interpretability measure but also to "compress" the data used in a given model and knowledge graph. We saw that we can reduce the data used by up to a half with minimal loss in accuracy and a significant reduction in time for both MNIST and CIFAR-10. We also unveil the idea of shallow distance learning, which can aid the identification of potentially complex causal relationships between classes in critical situations. We can use this method to further determine how many pixels we "really" need to maintain the distances of the graph. In the case of multi-modal MNIST, we can remove up to 100 features without substantially changing distances between classes. With this, we envision a future where no device is too small to make inferences at lightning speed and where models are deeply interpretable by design moving towards emulating the efficiency of the human mind.

5.1 Future Directions

We suggest building on the findings of this research to add additional experiments with combinations of different pixels. We also suggest experimenting with and determining the 'sweet spot' between the dynamic and stationary behaviour of a model, that focuses on different aspects for each class but is still able to compare these parts without over-fitting to training data. This does not seem plausible with systems like CNNs. An interplay between a probabilistic model, a causal model and a graphical model seems like the best setup to distinguish between data in a time-efficient and computationally simple way. Looking more closely at the interplay between important pixels for different samples can also give a more real indication of what components of an image matter for classification. For example, an important pixel in one sample might not be an important pixel in another sample. By using the same approach we used, one can figure out importances per sample

and then compare and contrast. Along with this idea we also believe that building up a database of important pixels for transfer learning and building up points of reference as mentioned in the introduction, can enable the foundation for more frugal AI systems in all domains. Doing so could also involve experiments with people to see what parts of images humans deem more important, some of which have already been done.

We would also suggest experimenting with models that capture multiple modalities in data combining RNNs with other methods like CNNs. We also believe that increasing cross domain generalisability by collecting important pixel values per class starting with MNIST is a valid avenue for further development. This is why we made the interactive website in the first place https://github.com/ilaiadaya/shadow.

We also suggest further experimenting with iteratively removing or altering parts of the nodes' data: For each part of the nodes' data (e.g., a feature, a set of pixels, a time window in a time series, etc.), remove or alter that part and recompute the pairwise distances between nodes.

Finally, we suggest running our experiments with datasets that have an increased number of pixels and an increased number of classes (like CIFAR-100) to see how the findings of our study extrapolate across domains. We suggest combining this with experimenting with other architectures like visual-attention models to see which one acts as the best data "compressor" and identifies pixels in a way that maintains the highest accuracy.

5.2 Takeaways

With our investigation, we aim to stimulate a discussion on causality and our work to enable a deeper notion of Interpretability. We also hope that, unlike previous Interpretability methods, this work is also seen as an efficiency improvement for image classification. We also saw that the ML paradigm does not seem to work with different important pixels per class, which hints at looking for causal methodologies that are able

to utilize different important pixels per class. We also stipulate the importance of using a method like SHADOW to look at the relative distances between data points and classes. It seems that as we move from uni-modal to multi-modal data causality becomes even more important, and can provide insights into the relativity of data. Above all, we hope this research has enabled a renewed sense of curiosity into the realms of causality and the human mind, and how machines and humans develop semantic understandings of the world.

References

[1] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Advances in neural information processing systems*, pp. 3630–3638, 2016.

[2] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[3] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, pp. 818–833, Springer, 2014.

[4] M. Evchenko, J. Vanschoren, H. H. Hoos, M. Schoenauer, and M. Sebag, "Frugal machine learning," *CoRR*, vol. abs/2111.03731, 2021.

[5] K. Xia, K.-Z. Lee, Y. Bengio, and E. Bareinboim, "The causal-neural connection: Expressiveness, learnability, and inference," 2022.

[6] J. Pearl, *Causality: Models, Reasoning and Inference*. Cambridge University Press, 2009.

[7] P. Spirtes, C. Glymour, and R. Scheines, *Causation, Prediction, and Search*. MIT Press, 2000.

[8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[9] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations (ICLR)*, 2021.

[10] Y. LeCun, C. Cortes, and C. J. C. Burges, "The MNIST database of handwritten digits." http://yann.lecun.com/exdb/mnist/, 1998.

[11] C. Koch, *The Quest for Consciousness: A Neurobiological Approach*. Roberts and Company Publishers, 2004.

[12] J. Markoff, "Brainlike computers, learning from experience," *The New York Times*, Nov 2011.

[13] X. Wu, Q. Chen, W. Li, Y. Xiao, and B. Hu, "Adahgnn: Adaptive hypergraph neural networks for multi-label image classification," in *Proceedings of the 28th ACM International Conference on Multimedia*, pp. 284–293, 2020.

[14] C. Wang, W. Zheng, J. Li, J. Zhou, and J. Lu, "Deep factorized metric learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7672–7682, 2023.

[15] H. Xu, Y. Chen, and D. Zhang, "Semantic interpretation for convolutional neural networks: What makes a cat a cat?," *Advanced Science*, vol. 9, no. 35, 2022.

[16] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "Mixup: Beyond empirical risk minimization," in *International Conference on Learning Representations*, 2018.

[17] V. Verma, A. Lamb, C. Beckham, A. Najafi, I. Mitliagkas, D. Lopez-Paz, and Y. Bengio, "Manifold mixup: Better representations by interpolating hidden states," in *Proceedings of the 36th International Conference on Machine Learning*, vol. 97 of *PMLR*, pp. 6438–6447, 2019.

[18] H. Guo, Y. Mao, and R. Zhang, "Mixup as locally linear out-of-manifold regularization," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 3714–3722, 2019.

[19] S. Thulasidasan, G. Chennupati, J. A. Bilmes, T. Bhattacharya, and S. Michalak, "On mixup training: Improved calibration and predictive uncertainty for deep neural networks," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[20] A. Damianou, N. D. Lawrence, and C. H. Ek, "Multi-view learning as a nonparametric nonlinear inter-battery factor analysis," *Journal of Machine Learning Research*, vol. 22, no. 86, pp. 1–51, 2021.

[21] D. Chen, "Output regularization with cluster-based soft targets," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 4, pp. 1378–1389, 2023.

[22] Q. Xu, Y. Li, J. Shen, P. Zhang, J. K. Liu, H. Tang, and G. Pan, "Hierarchical spiking-based model for efficient image classification with enhanced feature extraction and encoding," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 4, pp. 1024–1036, 2022.

[23] M. Li and L. Ma, "Learning asymmetric and local features in multi-dimensional data through wavelets with recursive partitioning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, pp. 7674–7687, Nov 2022.

[24] J. Wang, O. Russakovsky, and D. Ramanan, "The more you look, the more you see: Towards general object understanding through recursive refinement," in *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2018.

[25] S. Chen, G. Xiao, Z. Shi, J. Guo, and J. Ma, "Ssl-net: Sparse semantic learning for identifying reliable correspondences," *Pattern Recognition*, vol. 146, p. Article 110039, 2024.

[26] S.-Y. Kung and Z. Hou, "Augment deep bp-parameter learning with local xai-structural learning," *Journal of Communications in Information and Systems*, vol. 20, no. 3, pp. 319–352, 2020.

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

[28] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen, and K.-R. Müller, "How to explain individual classification decisions," *Journal of Machine Learning Research*, vol. 11, pp. 1803–1831, 2010.

[29] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3319–3328, 2017.

[30] L. Itti, C. Koch, and E. Niebur, "A model of saliency-based visual attention for rapid scene analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 11, pp. 1254–1259, 1998.

[31] J. Harel, C. Koch, and P. Perona, "Graph-based visual saliency," in *Advances in Neural Information Processing Systems*, pp. 545–552, 2006.

[32] V. Petsiuk, A. Das, and K. Saenko, "Rise: Randomized input sampling for explanation of black-box models," in *Proceedings of the British Machine Vision Conference (BMVC)*, 2018.

[33] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "Slic superpixels compared to state-of-the-art superpixel methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2274–2282, 2012.

[34] 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*, pp. 1135–1144, 2016.

[35] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[36] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Å. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

[38] K. Xu, J. L. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International Conference on Machine Learning*, pp. 2048–2057, 2015.

[39] Z. Hou, B. Yu, and D. Tao, "Batchformer: Learning to explore sample relationships for robust representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7256–7266, 2022.

[40] A. Ghorbani, J. Wexler, J. Zou, and B. Kim, "Towards automatic concept-based explanations," *arXiv preprint arXiv:1902.03129*, 2019.

[41] P. Lapa, I. Gonzalves, L. Rundo, and M. Castelli, "Semantic learning machine improves the cnn-based detection of prostate cancer in non-contrast-enhanced mri," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1837–1845, 2019.

[42] Z. Fayyaz, A. Altamimi, C. Zoellner, N. Klein, and O. T. Wolf, "A model of semantic completion in generative episodic memory," *Neural Computation*, vol. 34, pp. 1841–1870, Sep 2022.

[43] D. A. Spielman and N. Srivastava, "Graph sparsification by effective resistances," *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1913–1926, 2004.

[44] Y. Chen, H. Ye, S. Vedula, A. Bronstein, R. Dreslinski, T. Mudge, and N. Talati, "Demystifying graph sparsification algorithms in graph properties preservation," *arXiv preprint arXiv:2311.12314*, 2023.

[45] L. Liu, W. Chen, and H. Tong, "Graph sparsification with graph convolutional networks," *International Journal of Data Science and Analytics*, vol. 12, no. 3, pp. 243–256, 2021.

[46] A. Benczúr and D. R. Karger, "Spectral sparsification of graphs," *SIAM Journal on Computing*, vol. 44, no. 2, pp. 319–348, 2015.

A | Source Code

Source code for all of the methods implemented in Chap. 3 for the project can be found in the GitHub repository:
https://github.com/ilaiadaya/shadow.

Code for the interactive website can be found here:
https://github.com/ilaiadaya/pixel_interactive.

The interactive website can be found here:
https://pixel-interactive.streamlit.app/.

B | Supplementary Experimental Results

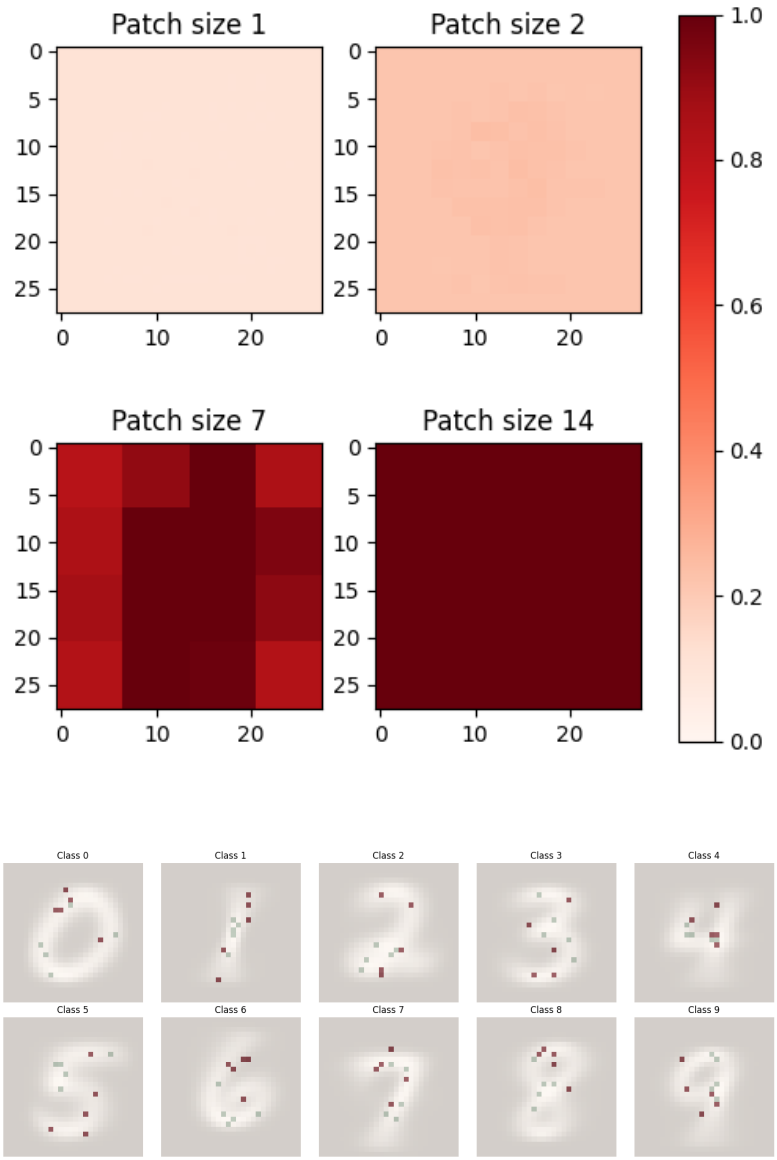
Table B.1: SmartPatchV1 Results

Exp	Number of Pixels							
	20	50	100	200	300	500	600	700
(a*)	0.060 ± 0.017 (37.7s)	0.085 ± 0.010 (27.8s)	0.092 ± 0.009 (36.7s)	0.111 ± 0.028 (39.9s)	0.136 ± 0.047 (36.0s)	0.276 ± 0.096 (34.0s)	0.479 ± 0.045 (40.3s)	0.665 ± 0.047 (38.0s)
(b*)	0.163 ± 0.017 (33.9s)	0.284 ± 0.053 (33.8s)	0.439 ± 0.065 (35.1s)	0.731 ± 0.064 (32.8s)	0.765 ± 0.043 (34.7s)	0.865 ± 0.022 (33.7s)	0.899 ± 0.015 (40.7s)	0.919 ± 0.003 (32.9s)
(c*)	0.095 ± 0.027 (38.6s)	0.119 ± 0.005 (42.2s)	0.111 ± 0.014 (35.9s)	0.264 ± 0.073 (36.7s)	0.469 ± 0.180 (37.4s)	0.773 ± 0.047 (34.9s)	0.854 ± 0.025 (34.7s)	0.900 ± 0.010 (33.3s)
(d*)	0.087 ± 0.016 (30.2s)	0.103 ± 0.010 (34.8s)	0.071 ± 0.014 (43.2s)	0.092 ± 0.008 (38.9s)	0.129 ± 0.019 (42.6s)	0.354 ± 0.106 (38.3s)	0.459 ± 0.079 (39.9s)	0.713 ± 0.063 (35.0s)
(e*)	0.182 ± 0.061 (34.7s)	0.390 ± 0.032 (27.7s)	0.494 ± 0.053 (42.6s)	0.705 ± 0.033 (35.1s)	0.712 ± 0.088 (35.2s)	0.893 ± 0.016 (38.7s)	0.904 ± 0.010 (33.8s)	0.919 ± 0.005 (43.5s)
(f*)	0.107 ± 0.019 (32.0s)	0.111 ± 0.031 (35.1s)	0.117 ± 0.010 (39.7s)	0.331 ± 0.147 (42.6s)	0.249 ± 0.192 (36.1s)	0.776 ± 0.038 (32.3s)	0.833 ± 0.035 (35.0s)	0.910 ± 0.008 (38.1s)
(j*)	0.115 ± 0.033 (40.2s)	0.088 ± 0.020 (37.9s)	0.081 ± 0.020 (37.0s)	0.112 ± 0.038 (37.5s)	0.104 ± 0.025 (42.5s)	0.434 ± 0.100 (36.2s)	0.499 ± 0.270 (38.2s)	0.779 ± 0.015 (38.7s)
(k*)	0.397 ± 0.019 (35.3s)	0.576 ± 0.053 (29.8s)	0.719 ± 0.048 (37.5s)	0.854 ± 0.005 (34.3s)	0.883 ± 0.021 (42.6s)	0.916 ± 0.005 (30.9s)	0.927 ± 0.002 (30.4s)	0.943 ± 0.002 (38.0s)
(l*)	0.085 ± 0.024 (38.7s)	0.105 ± 0.006 (42.4s)	0.140 ± 0.035 (42.5s)	0.156 ± 0.045 (35.1s)	0.377 ± 0.066 (39.7s)	0.735 ± 0.070 (36.1s)	0.877 ± 0.008 (37.5s)	0.925 ± 0.006 (38.4s)

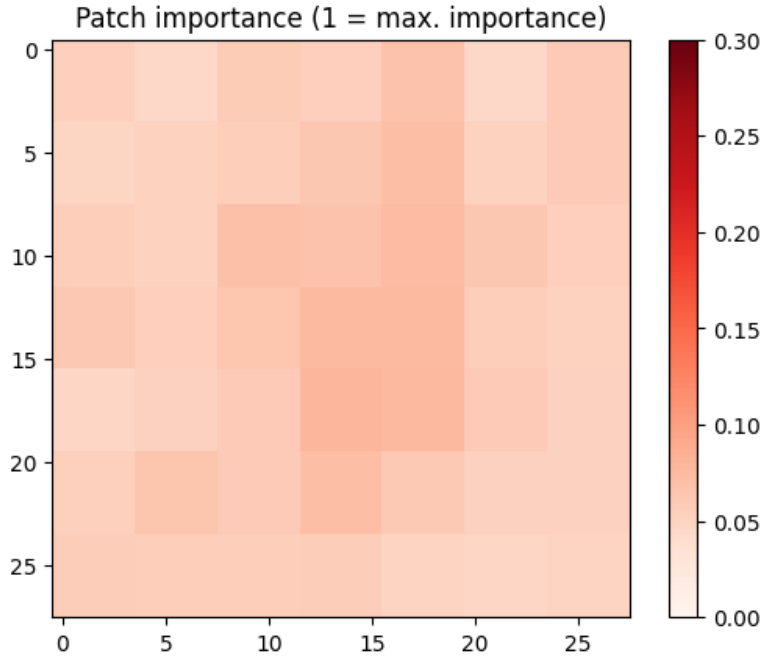
Table B.2: SmartPatchV2 Results

Pixels	Most Important	Least Important	Random Once	Random Each Sample
20	0.447 ± 0.007 (2.3s)	0.438 ± 0.004 (2.4s)	0.457 ± 0.006 (2.9s)	0.132 ± 0.002 (2.2s)
50	0.619 ± 0.007 (3.8s)	0.637 ± 0.003 (4.0s)	0.581 ± 0.052 (3.6s)	0.131 ± 0.001 (3.4s)
100	0.785 ± 0.010 (6.0s)	0.739 ± 0.009 (5.0s)	0.761 ± 0.005 (6.0s)	0.121 ± 0.005 (5.1s)
200	0.840 ± 0.007 (6.2s)	0.847 ± 0.007 (8.3s)	0.829 ± 0.007 (9.3s)	0.109 ± 0.003 (7.8s)
300	0.878 ± 0.013 (11.0s)	0.854 ± 0.015 (9.9s)	0.816 ± 0.016 (8.9s)	0.107 ± 0.003 (10.4s)
500	0.863 ± 0.034 (17.0s)	0.873 ± 0.008 (15.9s)	0.861 ± 0.004 (17.8s)	0.098 ± 0.001 (19.6s)
600	0.885 ± 0.010 (18.5s)	0.873 ± 0.027 (21.6s)	0.799 ± 0.068 (18.3s)	0.103 ± 0.008 (21.5s)
700	0.891 ± 0.019 (28.8s)	0.880 ± 0.003 (18.7s)	0.744 ± 0.171 (20.0s)	0.108 ± 0.008 (21.7s)

Results for Neural Network with varrying patch sizes and per class for patch size = 1.



Results for CNN patch size = 4


Patch importance (1 = max. importance)
0
5
10
15
20
25
0
5
10
15
20
25
0.30
0.25
0.20
0.15
0.10
0.05
0.00

C | CIFAR-10

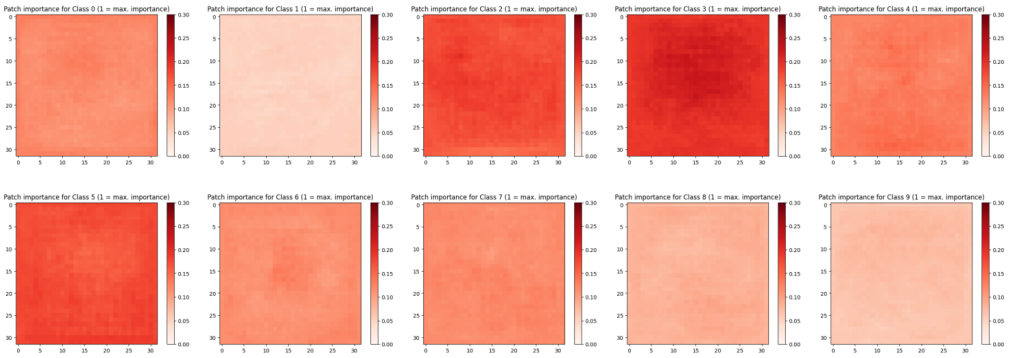


Figure C.1: CIFAR-10 per-class results

C.1 CNN architecture used with CIFAR-10

Layer Name	Configuration
Input	Image of shape `(dims[0], dims[1], CHANNELS)`
Convolutional	`Conv2D(96, (3, 3), activation='relu', padding='same')`
Dropout	`Dropout(0.2)`
Convolutional	`Conv2D(96, (3, 3), activation='relu', padding='same')`
Convolutional	`Conv2D(96, (3, 3), activation='relu', padding='same', strides=2)`
Dropout	`Dropout(0.5)`
Convolutional	`Conv2D(192, (3, 3), activation='relu', padding='same')`
Convolutional	`Conv2D(192, (3, 3), activation='relu', padding='same')`
Convolutional	`Conv2D(192, (3, 3), activation='relu', padding='same', strides=2)`
Dropout	`Dropout(0.5)`
Convolutional	`Conv2D(192, (3, 3), padding='same')`
Activation	`Activation('relu')`
Convolutional	`Conv2D(192, (1, 1), padding='valid')`
Activation	`Activation('relu')`
Convolutional	`Conv2D(10, (1, 1), padding='valid')`
Global Pooling	`GlobalAveragePooling2D()`
Activation	`Activation('softmax')`

D | Algorithms

D.1 Evaluation of Patch Importance

Algorithm 1 EvaluatePatchImportance

function EVALUATEPATCHIMPORTANCE(X_{train_subset}, y_{train_subset}, X_{test}, y_{test}, *model_builder*, *patch_size*, *name*)
    $pixel\_accuracy \leftarrow$ zero vector of size equal to the number of features in X_{train_subset}
    $listn \leftarrow$ list of zero vectors, one for each class
    $patch\_importance \leftarrow$ empty dictionary
    $class\_accuracies, class\_accuracies\_by\_pixel \leftarrow$ empty dictionaries
    $sample\_width \leftarrow \sqrt{\text{number of features in } X_{train\_subset}}$
    $no\_patches \leftarrow$ total number of features$/patch_size^2$
    $patches\_per\_row \leftarrow sample\_width/patch\_size$
    for $patch\_index \leftarrow 0$ to $no\_patches$ **do**
        $X_{train\_modified} \leftarrow$ copy of X_{train_subset}
        $patch\_row, patch\_col \leftarrow$ calculated based on $patch\_index$
        $patch\_indicies \leftarrow$ calculated based on $patch\_row$ and $patch\_col$
        $X_{train\_modified}[patch\_indicies] \leftarrow 0$
        $results \leftarrow$ Train and evaluate the model using $X_{train\_modified}, y_{train\_subset}, X_{test}, y_{test}$, and
        $model\_builder$
        $y_{pred\_classes}, y_{test\_classes} \leftarrow$ predicted and true classes from $results$
        $patch\_importance[patch\_row, patch\_col] \leftarrow 1 -$ accuracy of model
        $pixel\_accuracy[patch\_indicies] \leftarrow$ accuracy of the model
        for each $class\_label$ **do**
            $class\_accuracy \leftarrow$ accuracy of the model for the current class
            $class\_accuracies[class\_label, patch\_index] \leftarrow 1 - class\_accuracy$
            $listn[class\_label][patch\_indicies] \leftarrow 1 - class\_accuracy$
        end for
    end for
    return $patch\_importance$, $class\_accuracies$, $pixel\_accuracy$, $listn$
end function

D.2 Run model with selected pixels only

Algorithm 2 RunModelWithSelectedPixelsOnlyV2

function RunModelWithSelectedPixelsOnlyV2(X_{train2}, X_{test2}, y_{train2}, y_{test2}, $X_{train\_subset2}$, $y_{train\_subset2}$, k, $pixels$, val, $model\_builder$, $name$, $random\_selection$, $rando\_each\_sample$)
    if $random\_selection$ **is** True **then**
        print "running randomly"
        $all\_pixels \leftarrow$ list of all pixel indices in $X_{train\_subset2}$
        $pixels\_to\_turn\_on \leftarrow$ randomly select k pixels from all_pixels
    else
        print "running with important pixels"
        $pixels\_to\_turn\_on \leftarrow$ select top k pixels based on importance
        $pixels\_to\_turn\_on \leftarrow$ sort($pixels\_to\_turn\_on$)
    end if
    if $rando\_each\_sample$ **is** False **then**
        $pixels\_to\_turn\_on \leftarrow$ convert to numpy array
        $X_{train\_modified} \leftarrow X_{train\_subset2}[:, pixels\_to\_turn\_on]$
        $X_{test\_modified} \leftarrow X_{test2}[:, pixels\_to\_turn\_on]$
        print "X_train_modified shape: ", $X_{train\_modified}.shape$
    else
        if $k > X_{train\_subset2}.shape[1]$ **then**
            raise ValueError("k (k) cannot be larger than the number of features ($X_{train\_subset2}.shape[1]$)")
        end if
        $X_{train\_modified} \leftarrow$ randomly select k pixels from each sample in $X_{train\_subset2}$
        $X_{test\_modified} \leftarrow$ randomly select k pixels from each sample inX_{test2}
    end if
    $dims \leftarrow$ CalculateRectangleDimensions(k)
    if $dims$ **is** None **then**
        print "Cannot reshape " k "pixels into a rectangle"
        return
    end if
    $reshaped\_x\_train\_array \leftarrow X_{train\_modified}.reshape(-1, dims[0], dims[1], 1)$
    $reshaped\_x\_test\_array \leftarrow X_{test\_modified}.reshape(-1, dims[0], dims[1], 1)$
    $results \leftarrow$ TrainAndEvaluateModelSelectedPixelsOnlyV2($reshaped\_x\_train\_array$,
    $y_{train\_subset2}, reshaped\_x\_test\_array, y_{test2}, model\_builder, name$)
end function

D.3 Create distance graphs for classes

Algorithm 3 CreateDistanceGraph

function CreateDistanceGraph(X_{train}, y_{train}, $pixel\_importance\_df$, $output\_gif$, fps, $step$)
    $avg\_distance\_over\_time \leftarrow$ empty list
    $images \leftarrow$ empty list
    $G \leftarrow$ new graph
    for $i \leftarrow 0$ to 9 **do**
        $G.add\_node(i)$
    end for
    $pos \leftarrow$ spring layout of G with seed 42
    for $n \leftarrow 1$ to $X_{train}.shape[1]$ **step** $step$ **do**
        $top\_n\_pixels \leftarrow pixel\_importance\_df$.sort_values(by='importance', ascending=False)
        $.head(n)['pixel\_id'].values$
        $top\_n\_pixels \leftarrow X_{train}.columns[top\_n\_pixels]$
        $X_{train\_subset} \leftarrow X_{train}.loc[:, top\_n\_pixels]$
        $avg\_distances \leftarrow$ zeros matrix of size $(10, 10)$
        $start \leftarrow$ current time
        for $i \leftarrow 0$ to 9 **do**
            for $j \leftarrow i + 1$ to 9 **do**
                $Xi_{train} \leftarrow X_{train\_subset}[y_{train} == i]$
                $Xj_{train} \leftarrow X_{train\_subset}[y_{train} == j]$
                $distances \leftarrow$ pairwise Manhattan distances between Xi_{train} and Xj_{train}
                $avg\_distances[i, j] \leftarrow$ round(mean of $distances, 0$)
                $avg\_distances[j, i] \leftarrow avg\_distances[i, j]$
            end for
        end for
        for $i \leftarrow 0$ to 9 **do**
            for $j \leftarrow i + 1$ to 9 **do**
                if $avg\_distances[i, j] > 0$ **then**
                    $G.add\_edge(i, j, weight = avg\_distances[i, j])$
                end if
            end for
        end for
        $weights \leftarrow$ get edge attributes of G for 'weight'
        $weights\_normalized \leftarrow$ [float(i)/max($weights.values()$) for i in $weights.values()$]
        draw network edges with G, pos, edge_color=$weights\_normalized$,
        $width =$ [w*5 for w in weights_normalized]
        draw network nodes with G, pos, node_color='grey'
        draw network labels with G, pos
    end for
end function