university of groningen / faculty of mathematics and natural sciences / artificial intelligence

Proxy Attention : Approximating Attention in CNNs using XAI Techniques

Graduation Project

(Computational Intelligence and Robotics)

Subhaditya Mukherjee (s4747925)

May 4, 2023

Internal Supervisor: S.H. Mohades Kasaei, PhD

Second Internal Supervisor: Matias Valdenegro, PhD

(Artificial Intelligence, University of Groningen)

Artificial Intelligence

University of Groningen, The Netherlands

CONTENTS

LIST OF FIGURES

LIST OF TABLES

KEY

1. $\odot$ denotes element-wise multiplication
2. NN denotes Neural Network
3. CNN denotes Convolutional Neural Network
4. XAI denotes Explainable Artificial Intelligence
5. ViT denotes Vision Transformer

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

The problem statement of this study is creating a novel augmentation technique - **Proxy Attention**, that uses attention maps to improve the performance of any model by guiding its attention away from the regions that are not important for the classification task. In turn, this method should also improve the explainability of the model while maintaining the same architecture and hyperparameters.

1.2 Motivation

- Recent boom in using Transformers - Transformers are computationally expensive, harder to train, and require more data - The biggest advantage of Transformers is attention - CNNs are still extremely useful, but they lack the explainability of Transformers - The idea is to combine the best of both worlds - Not directly possible, so we use XAI techniques to approximate the effects of attention - Mistakes made by CNNs are often due to the model focusing on the wrong regions. While CNNs eventually learn to pick the right regions, this takes time and a lot of data. Proxy Attention aims to speed up this process, and make the model converge faster by guiding its attention away from the regions that are not important for the classification task. - Uses what the model already knows to help guide it by helping it better understand it's mistakes

1.3 Context and Novelty

The concept of Proxy Attention is relevant to any computer vision task, but we focus on image classification in this study. Using Proxy Attention, significant improvements in performance and explainability can be achieved without changing the architecture and with minimal changes to an existing code base.
The novelty of this study is the use of XAI techniques as an augmentation technique to approximate the effects of Attention in a CNN and guide the model's focus away from the regions that are preventing it from making the correct prediction. Proxy Attention was created in the hopes of showing that the explainability of CNNs can be improved by using the outputs from XAI techniques.
Combining these two concepts is a novel approach, and the author hopes that this study will inspire further research in this direction and motivate researchers to explore the possibilities of combining seemingly unrelated concepts to create novel solutions.

1.4 Challenges

The major challenges of this study were as follows:

- Creating a novel augmentation technique that uses attention maps to improve the model's performance.
- Testing the effect of Proxy attention on the explainability of the model.
- Comparing many hyperparameters and models with limited computational resources.
- Optimizing the usage of XAI techniques to improve the computational efficiency of Proxy Attention.

1.5 Research Questions

The main research questions that summarize the aims of this study are as follows.

1. Is it possible to create an augmentation technique that uses Attention maps?
2. Is it possible to approximate the effects of Attention from ViTs in a CNN without changing the architecture?
3. Is it possible to make a network converge faster and consequently require less data using the outputs from XAI techniques?
4. Does using Proxy Attention impact the explainability positively?

1.6 Thesis Outline

This thesis follows the following structure:

- **Chapter 2** provides the necessary background information about this study's relevant topics and datasets.
- **Chapter 3** provides a literature review of the relevant topics.
- **Chapter 4** describes the methodology used in this study and the implementation details.
- **Chapter 5** presents the results and answers the research questions.
- **Chapter 6** discusses the results in the context of the research questions.
- **Chapter 7** concludes the thesis and provides recommendations for future work.
- **Chapter 8** provides additional details and results.

CHAPTER 2

BACKGROUND

2.1 Interpretability

- Need for Interpretability

2.2 Gradient Based Explanations

- Taxonomy

2.3 Augmentation

- Taxonomy

CHAPTER 3

STATE OF THE ART

3.1 Gradient Based Explanations

One of the earlier approaches to Saliency maps for CNNs was proposed by Zeiler et al. [1] termed DeconvNet. DeconvNet works by inverting the network's operations in the forward pass. After attaching the DeconvNet layers to the network, propagating through these layers represents features that the original CNN possessed. The relevant reconstruction can be obtained for a single class by setting all the activations other than the one corresponding to the class to zero. The resulting image is then used to generate the saliency map. A Deconv layer replaces the Conv layer, and the ReLU operation has negative values clamped. While the pooling operation is not strictly invertible, the authors use switch variables that store the maximum value position for each pooling operation. While the DeconvNet works to a certain extent, the results are less accurate than the ones obtained by other methods and are also biased towards the representations of the first layer.

Building on the DeconvNet, Simonyan et al. [2] extrapolate the idea of class visualization to create one of the first approaches to Saliency maps. Their approach, also called Vanilla Gradient, ranks the pixels of an image I_0 by how important they are in the prediction of the Saliency score $S_c(I) \approx w^T I + b$. In this equation, w and b are the network weights and biases obtained by back-propagating wrt the image itself. The objective to be minimized thus is $\underset{I}{argmax} S_c(I) - \lambda ||I||_2^2$ where λ is used as a regularization parameter. Using these equations, a saliency map $A \in \mathbb{R}^{m \times n}$ ($m \times n$ stands for $height \times width$) can be computed. To find the map, we find the derivative of w, rearrange the elements and then process them according to the number of input channels. If the number of channels is greater than one, the maximum value over the channel is considered $A_{i,j} = \underset{ch}{max} |w_{h_{(i,j,ch)}}|$. Where ch is the colour channel of the pixel (i, j), $h(i, j, ch)$ is the index of the w corresponding to that pixel. The Vanilla Gradient method produces an approximate saliency map but has much noise. This leads to issues for more complex images. The methods proposed in the following papers have addressed many of the issues with Vanilla Gradients and DeconvNets [1].

In another paper, the authors propose a score-weighted approach (ScoreCAM) to create saliency maps [3]. Like many other methods, the images are first passed through the network, and the corresponding activations are obtained from the final convolutional layer. These activation maps are upsampled and normalized to $[0, 1]$. The highlighted activation map portions are then passed through a CNN with a SoftMax layer to obtain the score for each of the current classes. These scores are used to find the activation maps' relative importance. Finally, the sum of all these maps is computed using a linear combination with the corresponding target score and then passed through a ReLU operation. These operations can be mathematically represented as $L^c_{ScoreCAM} = ReLU(\sum_k w^c_k A^k)$, where k represents the index considered, c represents the current class and S_k represents the

outputs of the SoftMax as mentioned earlier layer. The authors find that the maps obtained using ScoreCAM are less noisy, and this method removes the dependency on unstable gradients compared to other methods.

A variant of GradCAM [4] was proposed by Selvaraju et al. [5] where, unlike GradCAM that finds the parts of the image that influence the model's decision, Guided GradCAM takes the positive gradients into account. These gradients are used to obtain an even more fine-grained representation of the outputs of the saliency map. While GradCAM backpropagates both positive and negative gradients, Guided Backprop only propagates the positive gradients and is defined as a pointwise multiplication of the results of GradCAM and Guided Backpropagation [6].

In combination with attribution methods, Noise Tunnel [7] is an algorithm that improves the accuracy of the masks obtained by these methods. Noise Tunnel was proposed to counter noisy and irrelevant attributions obtained by some gradient-based methods by adding a Gaussian Noise and then averaging the predictions over sampled attributions. Since all the samples are considered, this method has a significant computational overhead. For Smooth Grad [8], the new attribution is defined as $\hat{M}_c(x) = \frac{1}{n}\Sigma_1^n M_c(x + \mathcal{N}(0,\sigma^2))$. Where M_c is the attribution calculated by SmoothGrad, $\mathcal{N}(0, 0.01^2)$ is the Gaussian Noise with $\sigma = 0.01$ and n is the number of samples. Similarly for Smooth Grad Square, $\hat{M}_c(x) = \frac{1}{n}\Sigma_1^n \sqrt{M_c(x + \mathcal{N}(0,\sigma^2))}$. Noise Tunnel can also be used on Var Grad [9] with the equation $\hat{M}_c(x) = \frac{1}{n}\Sigma_{k=1}^n \{M_c(x + \mathcal{N}(0,\sigma^2))\}^2 - \{\hat{M}_c(x)\}^2$

For a model F, the attribution method Integrated Gradients [10] computes the contribution of each pixel in the image towards the final prediction. The model's output is used to calculate a pixel-wise partial derivative that is then integrated along a path starting from the baseline and ending at the input. Each step is scaled according to the partial derivative obtained in the previous step. For every step k with m total steps over the path, the IG equation is defined as $IntegratedGrads_i^{approx}(x) ::= (x_i - x_i') \times \Sigma_{k=1}^m \frac{\partial F(x' + \frac{k}{m} \times (x - x'))}{\partial x_i} \times \frac{1}{m}$. Where $(x_i - x_i')$ is the pixel-wise difference between the two images, $\frac{\partial F(x' + \frac{k}{m} \times (x - x'))}{\partial x_i}$ is the partial derivative of the model output F with respect to pixel i at the k-th step of the path, and $\frac{1}{m}$ is the scaling factor that ensures that each of the steps taken contributes equally to the final result.

Petsiuk et al. propose RISE [11], a saliency method that randomly alters the input images by applying random noise to each. After model predictions are obtained, the saliency map is generated by combining the partial maps over each modified image. RISE improves accuracy but needs a lot of computation time, considering that multiple models must be trained for each random noise sample.

Oyama et al. [12] found a strong correlation concerning the relationship between saliency maps and image classification accuracy. The authors found that the architecture and the initialization strategy influence the final saliency map. By analyzing the generated saliency maps, they find that if the model is randomly initialized and trained for image classification, having limited categories in the original dataset leads to overfitting. On the other hand, having many categories suppresses the overfitting of the objects present in the training dataset. On training their proposed network ReadoutNet on a fixation task (which requires the network to learn where to focus), they found that the accuracy of estimating the saliency map was linked to the image classification accuracy.

While a large amount of research focuses on interpreting the influence of a single image or neuron, Hohman et al. propose Summit, [13] a novel scalable summarization algorithm. Summit creates an attribution graph that distils the influence of neurons and substructures throughout the network used to make the final prediction. The attribution graph is created due to combining activation aggregation, a technique to find important neurons and neuron-influence aggregation, a technique to find relationships among the neurons identified in the previous step. After a forward pass through the network, the activation channels maximums are obtained to aggregate

the activations. These are then filtered by class and aggregated by taking the top k channels or the top k channels by weight. To quantify how much a layer influences the next, the authors aggregate the influences by creating a tensor I^l for all the network layers (l). How important channel i of the layer $l-1$ is determined by the aggregate tensor I^l_{cij} where j represents the output channel and c is the class of the image. Considering the j^{th} kernel of the layer $K^{(j)} \in \mathbb{R}^{H\times W\times C_{l-1}}$, a single channel Y can be represented using the 3D convolution operation by $Y_{:,:,j} = X * K^{(j)}$. This is equivalent to it's representation by the 2D convolution $Y_{:,:,j} = \Sigma_{i=1}^{C_{l-1}} X_{:,:,i} * K^{(j)}_{:,:,i}$. The value $X_{:,:,i} * K^{(j)}_{:,:,i}$ is the contribution of the current channel from the previous layer and the maximum of this value is used to generate the influence map.

Building upon Integrated Gradients, Dhamdhere et al. propose [14] Conductance, a means to boost the attributions provided by IG to specific neurons in the hidden layer. This is done by decomposing the computation that IG performs. The authors apply this method to the Inception network [15] and can find the filters that influence the final predictions most. For a neuron y , the network can be represented as a function $F : R^n \rightarrow [0,1]$. Given an input $x \in R^n$ and a baseline input $x' \in R^n$, the IG for the i^{th} dimension at x is given by $IG_i(x) ::== (x_i - x'_i)\int_{\alpha=0}^{1} \frac{\partial F(x'+\alpha(x-x'))}{\partial x_i} d\alpha$. Considering $\frac{\partial F(x)}{\partial x_i}$ to be the gradient of F along i^{th} dimension at x, the Conductance for y can be defined as $Cond_i^y(x) ::== (x_i - x'_i)\int_{\alpha=0}^{1} \frac{\partial F(x'+\alpha(x-x'))}{\partial y} \cdot \frac{\partial y}{\partial x_i} d\alpha$. The authors also propose methods of evaluating Conductance by the assumption that an influential hidden network should be good at predicting the given input class. This assumption can be validated by two metrics : the $Gradient \times Activation : y \times \frac{\partial F(x'+\alpha\times(x-x'))}{\partial y} d\alpha$ and the Internal Influence : $IntInf^y(x) ::= \int_{\alpha=0}^{1} \frac{\partial F(x'+\alpha(x-x'))}{\partial y} d\alpha$.

Consider an image classification task where an input image x is classified as a single class from a set C. For every class $c \in C$, the output class is represented as $class(x) = argmax_{c\in C} S_c(x)$. Using this $class$, a sensitivity map $M_c(x)$ can be generated by differentiating with respect to x, $M_c(x) = \frac{\partial S_c}{\partial x}$. M_c, being a sensitivity map ([2]), thus representing the influential regions of the image used to make the prediction. Since these maps are noisy, Smilkov et al. propose SmoothGrad [8], a modification of the previous method where instead of using ∂S_c, a smoothing is applied using a Gaussian kernel to ∂S_c. The authors also find that it is impossible to directly compute the smoothing due to high dimensionality and thus approximate the calculation by averaging multiple maps computed in the neighbourhood of x using random sampling. The final SmoothGrad equation then becomes $\hat{M}_c(x) = \frac{1}{n}\Sigma_1^n M_c(x + \mathcal{N}(0,\sigma^2))$, where $\mathcal{N}(0,\sigma^2)$ is the Gaussian noise and σ is the standard deviation.

Beware Of Inmates

Interpretation Is Fragile

Sanity Checks

The Unreliability Of Saliency Methods

There And Back Again

Cam

Gradcam++

Guided Backprop

Salience Map

Sam Resnet

Deep Fool

Deep Lift

Generalizing Adversarial Exp With Gradcam

Shap
Smooth Grad Square
Lime
Sp Lime
Lrp
Var Grad
Visualizing the Impact Of Feature Attribution Baselines
Adaptive Whitening Saliency
Bayesian Rule List
Deep Visual Explanations
Dynamic Visual Attention
Embedding Knowledge Into Deep Attention Map
Graph-Based Visual Saliency

3.2 Augmentation

Another augmentation strategy proposed by [16] first applies multiple transformations randomly and in parallel chains to each image. These transformations can include combinations of Translation, Rotation, Shearing and others. The outputs of these combinations are then mixed to form a new image, which is further mixed with the original image to form the new image. This combination improves performance in cases where data shifts are encountered in production. Once the images are mixed, a skip connection is used to combine the results of the chains. AugMix also uses the Jensen-Shannon Divergence consistency loss [17] to ensure the images are stable across various inputs. Considering KL to be Kullback-Leibler Divergence, the Jensen-Shannon Divergence can be defined as $JS(p_{orig}; p_{augmix1}; p_{augmix2}) = \frac{1}{3}(KL[p_{orig}||M||] + KL[p_{augmix1}||M||] + KL[p_{augmix2}||M||])$, where M is the mean of the three distributions $p_{orig}, p_{augmix1}, p_{augmix2}$.

Devries et al., in their paper [18], propose an augmentation method they call Cutout. This method removes random-sized square patches from the images by replacing the corresponding pixels with a constant value (usually 0). Selecting the region involves picking a random pixel value and creating a uniform-sized square around the chosen pixel. The authors also find that Cutout performs better with other methods than just being used by itself. Cutout can be expressed as an element-wise multiplication operation $x_{cutout} = x \odot M$, x is the original image, M is a binary mask of the same size as x with randomly chosen coordinates of a square patch of pixels to be cut out, and $\odot$ denotes element-wise multiplication.

Unlike Cutout [18], where the chosen patch is replaced with zero pixels, in CutMix [19], the chosen patch is replaced with a randomly chosen patch from a different region of the same image. Yun et al. propose this approach as multiple class labels can be learned with a single image. CutMix can be defined by the following operations $\tilde{x} = M \odot x_A + (1 - M) \odot x_B$; $\tilde{y} = \lambda y_A + (1 - \lambda) y_B$. x is an RGB image, y is the respective label, M is a binary mask of the image patch that will be dropped, and $\odot$ represents element-wise multiplication. The new training sample $\tilde{x}, \tilde{y}$ is created by combining two other training samples x_A, y_A and x_B, y_B. To control the combination ratio λ, a sample from the $\beta(1,1)$ distribution is chosen. This combination is quite similar to [20] but differs in the sense that CutMix focuses on generating locally natural images. Building upon [19], Walawalkar et al. propose an alternative method of replacing patches in an image they call Attentive CutMix [21]. Instead of randomly pasting patches in the image, this method uses a pre-trained network to identify

attentive regions from the image. Similar to the earlier approach, these patches are mapped back to the original image. Doing so allows the network to select important background regions for the task while also updating the label information.

Many of the algorithms use rectangular or square-shaped masks. While effective, French et al. propose Cow Mask [22], a new masking method that uses irregularly shaped masks with a Gaussian filter to reduce noise. The authors also propose two mixing methods, one that builds up on Random Erasing [23], and another that uses Cut Mix [19]. A pixel-wise mixing threshold is also chosen, and either mixing or erasing is applied to the image based on this threshold. This augmentation technique is shown to be effective in semi-supervised learning.

proposed another approach involving a cut-paste methodology citedwibediCutPasteLearn2017. In their paper, the authors propose a new method of augmentation that extracts instances of objects from the images. Instead of pasting them on other images, they are pasted on randomly chosen backgrounds. This method leads to pixel artefacts in the images, as selecting the objects is a noisy process. To overcome the drop in performance, the authors apply a Gaussian blur and Poisson blending to the boundaries of the pasted objects. Further augmentation is applied before pasting the objects by rotation, occlusion and truncation. The authors also find that this approach makes the network more robust to image artefacts.

In their paper, Singh et al. [24] propose a data augmentation method that takes an image as an input and divides it into a grid. Each of the sub-grids is then turned off with a given probability. These sub-grids can be connected or independent of each other, and the turned-off grids are replaced by the average pixel value of all the images in the dataset.

One of the major drawbacks of algorithms that rely on modifying image patches (such as [24, 18, 23]) is that they sometimes delete parts of the image that might be useful to the network. To overcome this problem, Chen et al. propose a new method Grid Mask [25], that uses evenly spaced grids to find a balance between the amount of information that is deleted and stored. Using the number of grids and their respective sizes as a hyperparameter, the authors find that Grid Mask effectively preserves important parts of the image.

Zhang et al. propose another data augmentation method that uses a CAM [26] to identify the most important regions of an image. These parts are then thresholded, scaled, translated and pasted onto the target image. A similar process is also applied to the target image, and the attentive parts of the original image are used to replace the corresponding attentive parts of the target image. Similar to previous methods, the labels are also updated to reflect the changes in the image.

While Cutout augmentation [18] is applied to every image in the dataset, Zhong et al. propose a new method, Random Erasing, that takes a probability of being applied into account [23]. In Random Erasing, contiguous rectangular regions are selected and replaced randomly with random upper and lower limits chosen for both region area and aspect ratio. A region-aware detection algorithm is applied for object detection tasks to make the network more robust to occlusion. Note that Cutout removes square patches, while Random Erasing removes square or rectangular patches.

Many augmentation methods that rely on randomly choosing regions to cut and paste from sometimes fail to work well with regions that need more object information. ResizeMix [27] tackles this problem by replacing the patch with a proportionally resized version of the selected image. This method is similar to CutMix [19] but differs in the sense that ResizeMix uses a resized version of the entire image instead of a randomly chosen patch.

Another augmentation technique that applies random cropping and pasting is RICAP [28]. In this method, four regions are cropped from different images and pasted together to form a new image. The created image thus

has multiple mixed labels. A uniform distribution is used to determine the area of each cropped region in the final image. The authors propose multiple variants of RICAP that use different points of origin for cropping. The method works best when the cropped regions use the corners as the origin, allowing the network to see more of the image.

In their paper, Inoue et al. propose a method that merges images not by cut and paste but by averaging their pixel intensities. While algorithms like Mixup [20] modify the image's labels proportional to the amount of mixing between the original and the target images, Sample Pairing [29] maintains the same training labels. Sample Pairing follows an interval-based augmentation policy, where the network is trained for 100 epochs before being introduced to the mixed images. This process is also repeated cyclically with eight epochs of training with mixed images followed by 2 epochs of training with normal images.

With the success of mask-based approaches for data augmentation, there have been many papers that attempt to fix the flaws of previous research. One such method is SmoothMix [30], which builds up on both CutMix [19] and Cutout [18] but modifies the mask to have softer edges. The intensity of the masked edges gradually decreases and depends on the strength of the mask. The updated pixel values are thus obtained by mixing the mask with the original image according to the formula $\lambda = \frac{\Sigma_{i=1}^{W}\Sigma_{j=1}^{H} G_{ij}}{WH}$. Where G_{ij} is the pixel value of mask G and H, W are the height and width of the image, respectively. The new pixel values are then $(x_{new}, y_{new}) = (G.xa + (1-G).xb, \lambda.ya + (1-\lambda).yb)$

One of the older data augmentation methods is SMOTE [31]. This algorithm is not domain specific, but in the context of computer vision, it can be used to balance datasets that suffer from imbalanced labels. SMOTE generates new samples by combining the K-nearest neighbours of the minority class images to form new instances. Although many of the other methods discussed in this paper are more effective, SMOTE is still useful.

Huang et al. propose SnapMix [32], where choosing the patch size to be cut is determined from the beta distributions of both the original and target images. The extracted patches are then merged with random image regions, each of which is different in size. Labels are also updated by taking the composition of the images into account. Cao et al. address the problem of class imbalance by performing data augmentation on images that are part of a minority class. From the labels of the images that were mixed, the final label is chosen as the label of the image with the least representation in the dataset. The authors call this method ReMix [33]. Dvornik et al. propose Visual Context Augmentation [34] that uses a NN to understand the context of objects in the image before pasting them in the target image. The authors generate training data by first generating pairs of context images with the objects masked out. These images are then fed into the NN to learn the difference between objects and backgrounds given the masked pixels. Once the model has learnt this information, instances of the objects are placed into the masked regions of the target image.

While many techniques are based on Mixup [20], they are mostly focused on generating new samples of images from the existing data. Doing so is useful but sometimes leads to generating examples that confuse the network and do not represent the data. To tackle this issue, Kim et al. [35] propose Pizzle Mix, an algorithm that learns to copy patches of images between each other while taking saliency into account. Puzzle Mix learns to minimize the equation $h(x_0, x_1) = (1-z) \odot \Pi_0^T x_0 + z \odot \Pi_1^T x_1$ where x_0, x_1 are the two images, z_i is a binary mask, $\lambda = \frac{1}{n}\Sigma_i z_i$ is the mixing ratio and Π_0, Π_1 represent $n \times n$ grids that denote the amount of mass that is transported during transport of the image patch to another location.

Attributemix Augmentation with curriculum learning Co mixup Image Mixing and deletion

Keep augment Latent space interpo Randaugment Random distortion Saliencymix

Spec augment

3.2.1 Limitations

While each of these papers has its strengths, a few limitations were identified. These limitations do not affect the methods themselves but rather how they are used in the project context.

- Most of the algorithms are used as a final post-processing of the outputs to find the inherent biases present in the network. While this is the most common use case, it does not influence the network to learn from its mistakes and improve its performance. The XAI methods generally focus on explaining the network's decisions rather than improving them. This research proposes performing the latter.
- Contextual awareness in image classification is difficult to achieve without special networks or longer training times. While object detection tasks require this knowledge, networks trained purely for classification can do without it. That being the case, most of the research surveyed tackles this challenge in ways that are not generalizable to other networks easily. Proxy Attention, conversely, is independent of the network and can be used with any model and dataset.
- Combining the fields of XAI and data augmentation to improve network performance is a rare practice. This research is performed to bridge the gap between the two fields and to show that they can be used together to improve not only the performance of the network but also the explainability of the network's decisions simultaneously.

CHAPTER 4

IMPLEMENTATION

4.1 Overview

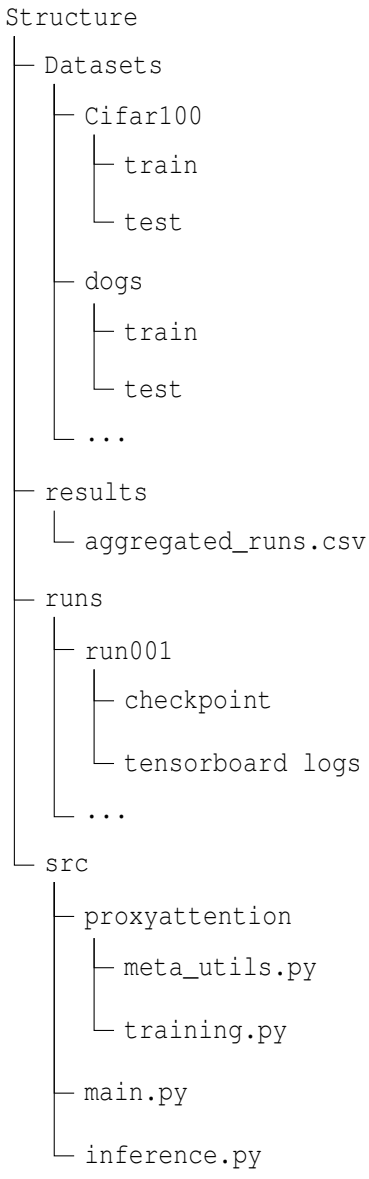


Figure 4.1: Code Directory Structure

4.2 Datasets

To test Proxy Attention, a variety of datasets were chosen. The datasets were chosen to be of varying difficulty, and to have varying number of classes. The datasets used are **CIFAR100, Stanford Dogs, ASL Alphabet and Plant Disease dataset**. The datasets are described in detail in the following sections.

The images provided by the datasets are of varying sizes, but are resized to a similar size for consistency. These visualizations were generated by the author, and are not fully representative of the original dataset but are provided for reference. Due to space constraints, not all classes are shown in the visualizations. The complete list of classes and examples can be found in the links provided.

4.2.1 CIFAR 100

The CIFAR 100 dataset, introduced by [36], is an image dataset with 60000 colour images with dimensions 32x32 pixels. As the name suggests, the dataset has 100 unique classes. Each of these classes has 500 training images. Some classes are - **airplane, bird, truck, ship, deer and dog**. This dataset is used as a coarse-grained classification dataset in this project.
The dataset and complete class information can be found here.

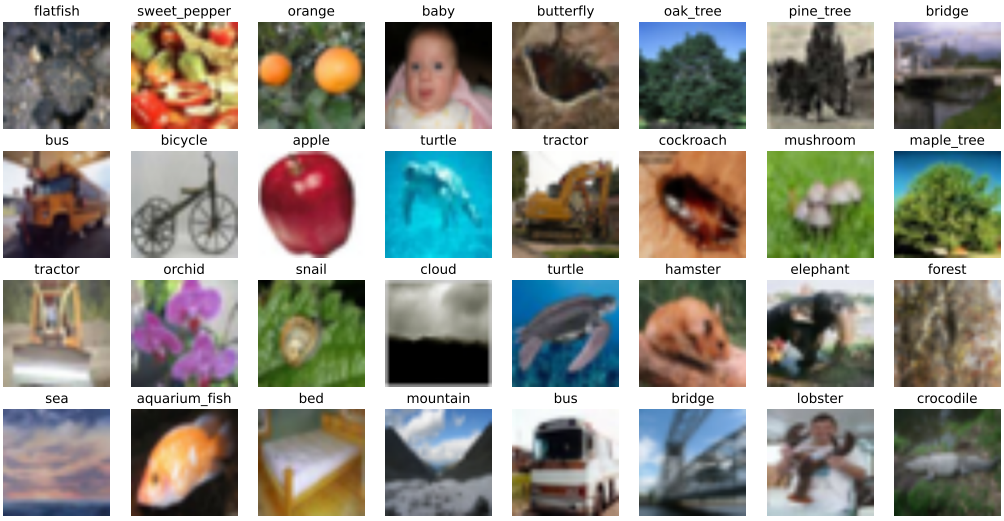


Figure 4.2: CIFAR100 Sample

4.2.2 Stanford dogs

The Stanford Dogs dataset [37] is a popular fine-grained image classification dataset. There are more than 20k images in this dataset categorized into 120 classes of dog breeds like the **Afghan Hound , Appenzeller** etc. Being a fine-grained dataset, the images are very similar to each other and the classification task is much harder. This dataset was chosen to further evaluate the explainability of Proxy Attention.
The dataset and complete class information can be found here.

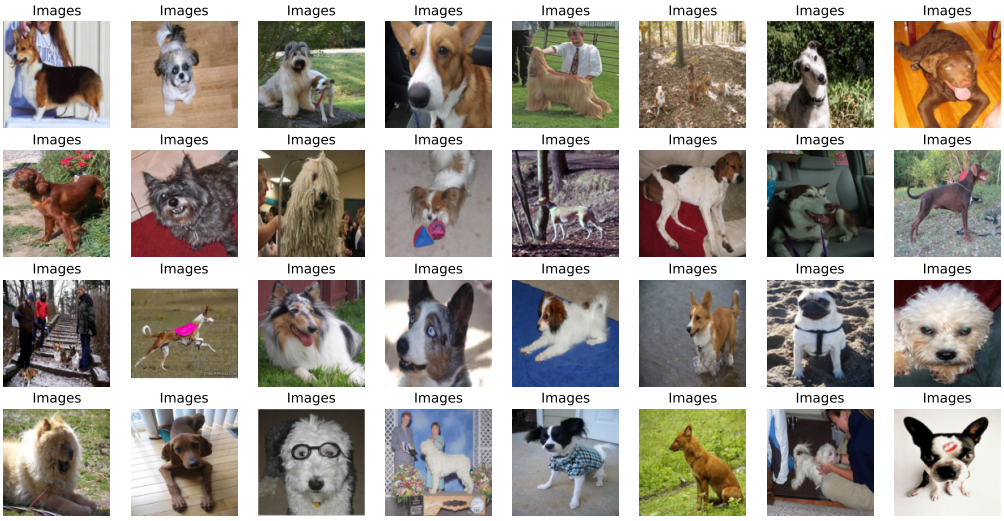


Figure 4.3: Stanford Dogs Sample

4.2.3 ASL Alphabet

The ASL dataset is a collection of hand pose images from the American Sign Language. There is no pose information with this dataset, but the images can be used for classification using the provided class labels. The dataset chosen to evaluate Proxy Attention is the ASL Alphabet dataset, a more specific subset that has all the letters of the English alphabet along with the special characters **del, space and nothing**. The background is mostly the same, with minor changes. The data is also recorded from people with a similar skin tone, which makes the task easier.

This is an easy to classify dataset and was used as an initial test of the Proxy Attention mechanism. The results of the same are left in for future reference.

The dataset and complete class information can be found here.

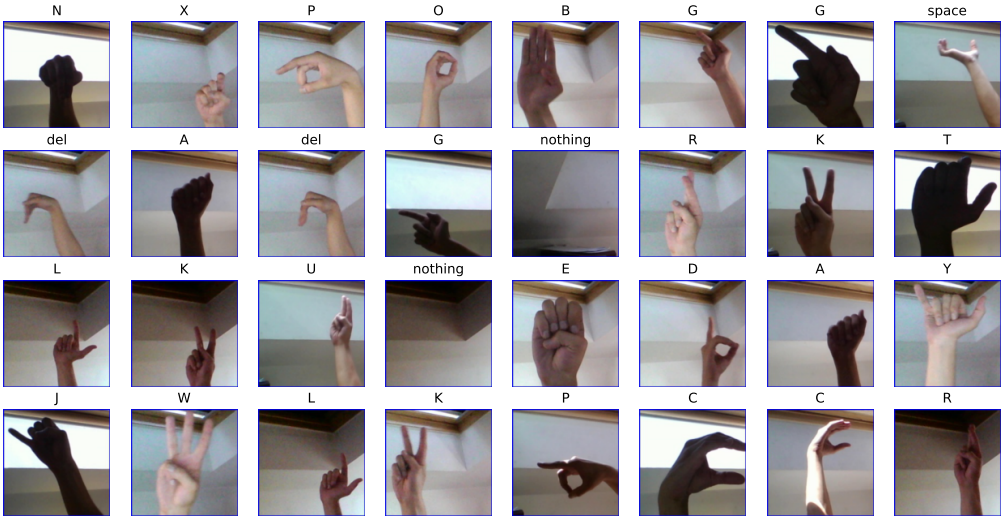


Figure 4.4: ASL Sample

4.2.4 Plant Disease

This dataset consists of images of plant diseases across a variety of plants. The dataset is also a fine-grained classification dataset with 39 classes. Other than a few diseases, most of them are quite similar to each other, making the classification task harder. Some examples of the classes are **apple scab, blueberry healthy, cherry powdery mildew** etc.
The dataset and complete class information can be found here.

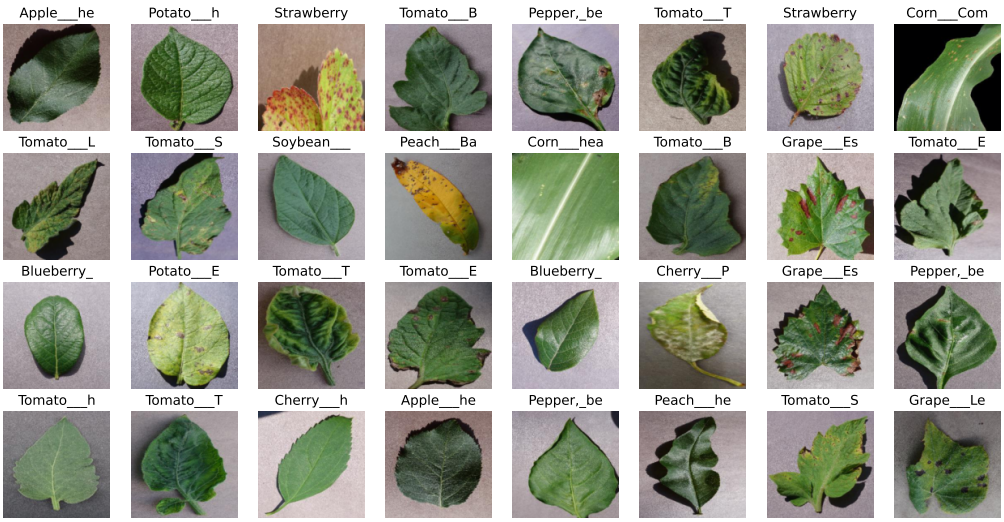


Figure 4.5: Plant Disease

4.2.5 Caltech101

The Caltech101 [38] dataset was created to tackle the absence of a uniform baseline comparison for vision classification tasks. The dataset has 101 categories of images, with a total of 9146 images. A background category is also included, which has images that do not belong to any of the 101 categories. An advantage of this dataset is that the images are of uniform size and have low clutter and occlusion, making it easier to classify. The caveat is that some categories have fewer samples than others.
The dataset and complete class information can be found here.

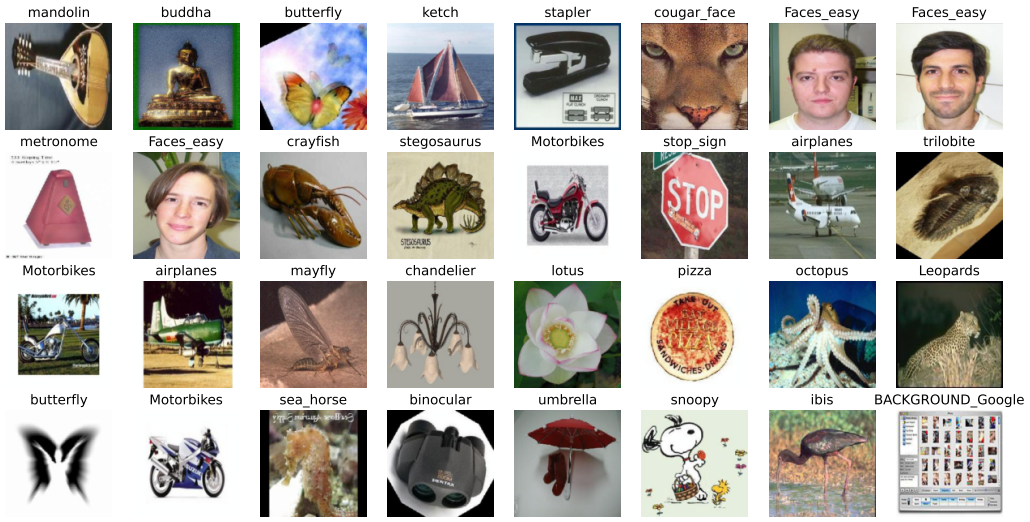


Figure 4.6: Caltech101

4.3 Proxy Attention

Algorithm 1 Single Batch Proxy Attention

Require: $input\_wrong$
Require: CAM
Require: $proxy\_threshold$
Require: $proxy\_image\_weight$
$grads \leftarrow CAM(input\_wrong)$
$inversed\_normalized\_inputs \leftarrow inverse\_normalize(input\_wrong)$
$output \leftarrow REPLACE(grads \geq proxy\_threshold, ((1 - proxy\_image\_weight) * grads) * inversed\_normalized\_inputs, inversed\_normalized\_inputs)$

4.4 Challenges and Potential Solutions

Being a novel method, many challenges were faced while implementing Proxy Attention. While solving all the issues faced due to time constraints was not feasible, the author tried to tackle as many as possible. Many of these issues were posed as optimization problems and were considered hyperparameters that could be tuned to

Algorithm 2 Batch Proxy Attention

Require: $input\_wrong$
Require: $label\_wrong$
Require: $subset\_chosen$
  $chosen\_inds = CEIL(subset\_chosen * LENGTH(input\_wrong))$
  $input\_wrong\_subset = input\_wrong[: chosen_inds]$
  $label\_wrong\_subset = label\_wrong[: chosen_inds]$
  $processed\_labels \leftarrow [], processed\_thresholds \leftarrow []$
  for $i \leftarrow 0$ **to** $LENGTH(label\_wrong\_subset)$ **do**
    pass #TODO
  end for

improve performance. This section discusses the possible solutions that were tested. Further details about each parameter can be found in Section 4.4.3.

4.4.1 Proxy Method

The Proxy Attention step involves replacing the pixels in the original images based on the attention maps obtained from a trained model. There are many different ways in which this can be done, some that were explored in the literature, some that were implemented and others that were left for future research. The following are the different methods that were considered:

Image Statistics Based Replacement

These methods use local or global statistical information from the images for replacement. All these methods can be computed per image, batch, or entire dataset.

1. **Average Pixel Value**: The average pixel value of the original image is used for replacement.
2. **Max Pixel Value**: The maximum pixel value of the original image is used for replacement.
3. **Min Pixel Value**: The minimum pixel value of the original image is used for replacement.
4. **0/255 Pixel Value**: The pixel value of 0 or 255 is used for replacement, where 0 refers to black and 255 refers to white.

These methods are simple but naive, leading to significant information loss. In many cases, if many images have their values replaced with these values, the model might become biased towards predicting a specific class when an image contains many pixels with these values. Due to this reason, these methods were not considered for the final implementation. A visualization of these methods can be found in Figure 4.7.

Data Augmentation Based Replacement

Data Augmentation techniques involve computing some transformation over images. Many of these methods were covered in the literature survey (Section 3.2), some of which replaced the pixels with random values, pixels sampled from either the current image or another image in the dataset, or even deleted the pixels. Most of these methods do not consider the model itself, but some, such as Saliency Mix [39] use saliency measures to find patches from other images in the dataset that are used to replace the chosen pixels. These methods inspired

Proxy Attention, but instead of replacing image patches or deleting pixels, it uses a gradient-based method to down-weight the pixels that might have led to the wrong prediction. This method moves away from using naive statistical information but enables the model to learn from its mistakes eventually.

GAN Based Replacement

Modifying the Weights

Instead of replacing the pixels, another possible method would be to modify the network weights directly. While many research papers elaborate on methods to perform this procedure, this domain still needs to be researched enough to be used easily. Research on this domain has been done from the early 90s [40], but practical implementation of such a network that learns to modify its weight while training has not been extremely successful [41]. That being the case, implementing such a method is left to future research.

add some papers

Multiply with Attention Map

The method chosen for this research does not directly replace the image's pixels but weights them using the attention map generated by passing the image through the trained model. The obtained attention map is thus multiplied with the original image. In line with the principles of Proxy Attention, this allows the network to understand that the parts of the image it initially focused on did not lead to the correct result. Note that doing so is only possible if the network has seen this image. Because the images are slightly modified after the Proxy Attention step, if the network still needs to learn what the original image looks like, it might make more mistakes in the future by learning the wrong set of features. A caveat of this method is that, after successfully applying the Proxy step to an image, the number of weighted pixels increases and, over time, might lead to the image not having any useful features left. This loss of information is tackled by clearing the proxy images every couple of steps. A visualization of this method compared to others can be found in Figure 4.7.

Figure 4.7: Methods

4.4.2 Training Biases

Gradient-based XAI methods are not perfect, and in many cases, they are unable to provide accurate explanations for the predictions made by the model. Since Proxy Attention relies on the outputs of these methods, this might lead to the model learning biased representations of the data. This section discusses the different biases that might be introduced by using these methods in combination with Proxy Attention and how they can potentially be mitigated.

Method Bias

Not all explainability methods perform equally. Some methods are shown to have better masks generated, while other methods are more computationally expensive. Since Proxy Attention heavily depends on these methods, using them may lead to additional artefacts in the generated images. Some methods lead to better results while being used alongside Proxy Attention. To test the effects of this, multiple gradient-based methods are used to compare the performance of the networks.

Mask Bias

Proxy Attention uses the attention maps produced by gradient-based methods and multiplies them on the original image as a mask. While this works well, the masks themselves have edge artefacts that may lead to corrupting some regions of the image. These artefacts are further amplified for smaller image sizes and might impact performance in the long run. Potential solutions include:

1. Smoothing the masks before applying them to the image using techniques such as Eigen Smoothing. This could help in reducing the edge artefacts.
2. Ensuring that only a certain percentage of the image is replaced by the Proxy Attention step. Doing so would preserve more information.

Learning Bias

1. Testing multiple schedules of when to apply the Proxy Attention step. This would help in understanding which part of the training process would benefit from the Proxy Attention step the most, reducing the computational overhead in the long run.
2. Not reusing previously masked images for the Proxy Step. Doing so ensures that the artefacts are not propagated further into the training process.

Dataset Bias

4.4.3 Hyper Parameters

Proxy Attention is a novel method, and so there is no previous research on the best hyperparameters to use. The objective in choosing them is to find a balance between performance, computational overhead, and memory usage. The following section discusses the different hyperparameters that were tested and the reasoning behind their selection. add values for all hyperparameters

Gradient Method

Many gradient-based methods are available for generating attention maps from trained networks. While many of these methods were mentioned in the survey, it was impossible to test them all. Since Proxy Attention's effectiveness depends quite a bit on the gradient method used, it was important to test them.
The important factor considered while choosing these methods was the difference in complexity and the power of explanation they provide. While algorithms like GradCAM++ [42] provide more nuanced and better explanations of the image, older algorithms like Vanilla Gradients [1] are not so accurate. The objective here was to

understand if using a more powerful method would improve performance with respect to classification accuracy when used with Proxy Attention. If this is the case, then it is possible to use more powerful methods to further improve performance in the future. The gradient methods that were tested are as follows:

- **GradCAM++** [42].
- **GradCAM** [4]

Gradient Threshold

Every gradient method considered generates a heatmap where the higher the activation, the more important the pixel is. The activations are mapped to a $[0, 1]$ range with higher values in the heatmap indicating higher activation values. Since using Proxy Attention would mean that the pixels with the chosen activation values would be down-weighted, choosing a threshold value would result in the best classification accuracy was important. This is a balancing act as choosing too small of a threshold would result in larger parts of the image being down-weighted, while choosing too large of a threshold would result in the image being down-weighted too little and hence being too close to the original image to make any difference.
A visualization of the different thresholds tested is shown in Figure 4.8.

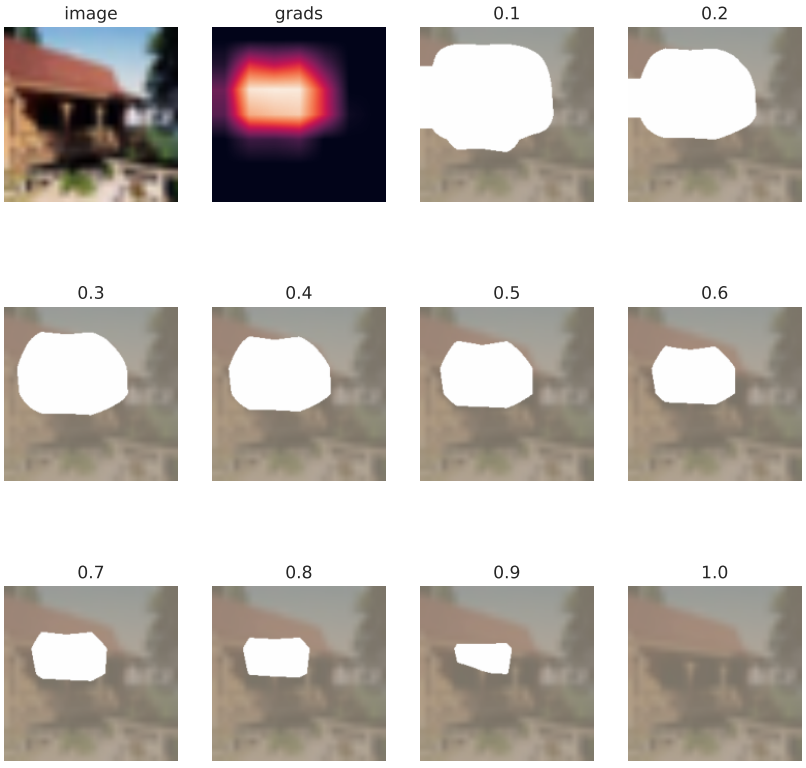


Figure 4.8: Gradient Thresholds

Multiply Weight

The Multiply Weight hyperparameter is used to control how strongly the attention map is applied to the image. The values are in the range $[0, 1]$. A higher value would mean that the image is more strongly affected by the attention map, while a lower value would mean that the image is less affected. This is a balancing act as choosing too high of a value would mean that the image is affected too much, and important features might be lost from the image. Choosing too low of a value would mean that the image is not affected enough, rendering the Proxy step useless. The optimal value of this hyperparameter is found based on the results of the experiments conducted.
generate and add figure here

Proxy Step Schedule

Proxy Attention is a novel method, which means that there was no previous literature on how often to apply the Proxy step. To test this, multiple schedules were tested to understand how the network performs when the Proxy step is applied at different times.
The challenge faced while testing for this was that if the Proxy step was applied too many times, it might lead to overfitting, while if it was applied too few times, it might not have any effect. One might consider applying the Proxy step for every step, but this would be too computationally expensive. Since Proxy Attention also relies on the understanding of the model, applying the Proxy step too many times initially, when the network is not trained yet, might degrade performance as well.
These issues indicate a need for a schedule for the Proxy step as well. It is manually scheduled as of now, except when using the schedule generator (which is also a naive method).
Future work might include generating a schedule with respect to the validation accuracy. This might be a good idea as, if the network is not learning well, the Proxy step could be applied more often. But if the performance is already sufficient, then there remains no need to apply the Proxy step as frequently and potentially degrade performance.

Subset Of Wrongly Classified

This hyperparameter was chosen to understand if increasing the number of images that are passed to the Proxy step would help in improving performance. While providing more images might lead to better performance, the more images that are passed to the Proxy step, the more computationally expensive it becomes. To test this, both ends of the spectrum were tested, with a small fraction and a large fraction of the images being passed through the Proxy step.
As of now, the number of images passed to the Proxy step is taken as a fraction of the total number of images in the dataset. Future work could include a schedule for this as well, where the number of images passed to the Proxy step decreases over time as the network learns more and does not need as much help in improving performance.

4.5 Data Loading and Pre-Processing

Since many datasets were used in the project, it was important to ensure that the data was consistent across all the experiments. Efficient use of memory and network performance were the main priorities while designing the

data loading and pre-processing steps. This section details all the tweaks, custom loaders, and pre-processing steps that were used to ensure the same.

4.5.1 Data Directory structure

The data was stored in a specific directory structure similar to the ImageNet [43] dataset to maintain consistency across the different experiments. Every dataset is divided into training and validation folders. Most of the datasets used in the project come with this split, but a validation split is created manually for those that do not. (Note that the test split is created from the training split while training the model and is not hardcoded.) For every class in the dataset, a subfolder within the parent folder is created with the name of the class. All the datasets used are stored in the same folder on an SSD for ease of access and performance. The directory structure is shown in Figure 4.9.

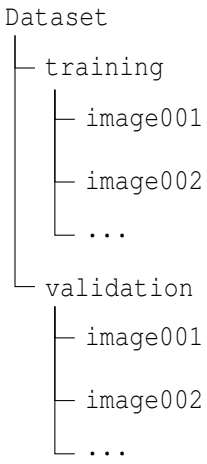


Figure 4.9: Dataset Directory Structure

4.5.2 Custom Data Loading

A custom data loading logic was implementd for this project. The steps are as follows:

1. First, the previously generated proxy images (if they exist) are cleared from the data folder. This is done to ensure that the proxy images are not loaded by mistake.

2. For all the remaining images, the file paths are listed and shuffled.

3. If the current step is a Proxy step, then for every proxy image loaded, the corresponding original image is not given to the data loader. This is done so that the number of images that the networks trained with and without Proxy Attention are equal.

4. If the subset parameter is set to a positive value, then only a subset of the data is obtained. If not, the entire dataset is used.

5. Using these file paths, a Pandas DataFrame is created with the file path and label. The label is generated using a label function (Ref 4.5.3) based on the file path.

6. The labels are encoded and transformed into numerical values using the LabelEncoder and LabelBinarizer classes from the sklearn library. A label map and reverse label map are created and stored in memory, a step that is useful for inference.

7. To ensure an equal percentage of samples per class (some datasets used have unequal distributions), a Stratified K-Fold oversampling is applied.

8. Before loading the images, an additional check is performed to ensure that the images have 3 channels. If they do not, then they are converted to RGB. This check is performed as some images in the datasets could be transparent, have 4 channels, or accidentally be grayscale. Not handling these images lead to errors while training and hence they are preemptively converted to RGB.

4.5.3 Label function

To ensure easy compatibility with new datasets, label functions are used to obtain labels given a file path instead of hardcoding them. This enables a uniform API that can be extended to any dataset by writing a simple lambda function.
In the previous step, a Pandas DataFrame with the file paths of all the images was created. To generate labels, the label function is mapped across the entire column of file paths. This function is also used to create the label map and reverse label map, that is useful for inference.
For example, consider the ASL dataset. The file path for a single training image is of the form

```
/media/subhaditya/datasets/ASL/asl_alphabet_train/asl_alphabet_train/A/A1.jpg
```

To obtain the label, a lambda function **lambda** x: x. split ("/")[−2] is used. The label function splits the string into a list by the Unix path label separator "/" and returns the second last element in the list. Thus, the label for this image becomes **A**.

4.5.4 Clearing proxy images

The images are saved locally for every iteration of the Proxy Attention step. That being the case, using these generated images over further iterations of the Proxy Attention step is possible. Since these images replace the original image from the data set, it is possible to use them as a direct substitute for the original images in the data set. Note that doing so would give the network more images when using Proxy Attention during training, which is potentially an unfair comparison. Only a single image is chosen during the data loading process to avoid this issue. Thus, this becomes a hyperparameter where the options are either to store the last generated proxy images across iterations and use those images as direct replacements for the original images or not perform the step.
In the long run, the option to persist the images across iterations could lead to the network learning artefacts introduced in prior iterations. To make sure that the networks that train with Proxy Attention are fairly compared with the ones that do not, the data loader is only passed either the original image or its substitute but not both.

4.5.5 Augmentations

Augmentations are a useful step in training neural networks and increasing robustness to new data. Since this project is a test of Proxy Attention and not of improving performance of specific architectures, a minimal set of

augmentations are used. All the transforms are applied using **torchvision**.
For both training and validation, the images are normalized using the ImageNet statistics. This is done to maintain a standard, and also since the pretrained models used have been trained on ImageNet [43]. The mean and standard deviation used are

```
mean = [0.485, 0.456, 0.406]
std = [0.229, 0.224, 0.225]
```

For training, the images are resized to 224×224. Random horizontal flips are also applied with a probability of 0.5 along with random rotations with a maximum angle of 10 degrees and a similar probability. The images are then converted to Tensors.
For validation, the images are resized to 224×224 and then converted to Tensors. No further augmentations are applied.

4.6 Architectures

This section discusses the architectures used and the library used to implement them.

4.6.1 TIMM

The library used to load the models is called **TIMM** [44]. It is a PyTorch library that provides a vast number of models with the option of loading pretrained versions of the same. The library also provides an easy way to customize the loading options for transfer learning, including the ability to choose the number of classes, the number of layers to freeze, Global Pooling options, etc.
The following models were used in this project:

4.6.2 ResNet - 18,50

4.6.3 VGG16

4.6.4 EfficientNetB0

4.6.5 ViT Base Patch 16×224

4.7 Grid Search

A grid search was performed to test the effectiveness of Proxy Attention and to find the best combination of hyperparameters. The grid search was performed on a single machine with a single GPU. An analysis script was written to determine what trials to run instead of using a separate optimization framework (Ref 4.15). Due to limited resources, an initial sweep over the hyperparameters was performed using a low memory network (ResNet18 [45]), a subset of the Dogs dataset ([37]), a simple gradient method (GradCAM [4]) and a small number of epochs. A separate process was started for each trial in the grid search, and the memory was cleared after each trial. This process was repeated until the best combination of hyperparameters was found. Once the worst-performing parameters were eliminated, the rest of the trials were run for the other networks, datasets and methods. Although it was possible to use a separate optimization framework and an algorithm like Bayesian

Optimization to find the best combination of hyperparameters, the parameters were semi-automatically chosen instead due to a lack of resources and time.

4.8 Training Resumption

Training resumption was an important part of this project. Since Proxy Attention is applied between training runs, it was important to be able to resume training from the last checkpoint. Furthermore, since a single machine was used, resuming broken trials easily was a useful feature to have. This section discusses the challenges faced in creating this feature and the solutions implemented to overcome them.

4.8.1 Checkpoints

While checkpoints are almost always a good idea, they were especially important in this project. The Proxy Attention step is applied between training runs, and to preserve memory, it unloads the existing models and DataLoaders from the GPU. This means that when continuing training, the models and DataLoaders need to be reloaded before the next training run. Doing so would effectively reset the training process, so it was important to have checkpoints to resume training. As part of the final analysis, the author also iterated over the trained models and compared the explainability of models trained with or without Proxy Attention. Having saved checkpoints made this process much easier.

4.8.2 Broken Trials

A challenge of training on a single machine was that the training process could be interrupted at any time. Since this project required several experiments to find the best combination of hyperparameters, it was important to be able to resume training in case of any interruptions. The author initially tried using libraries such as Optuna and Ray Tune, but these added too much of an overhead for this project. (Ref 4.8.3) A custom solution was implemented instead.
The objective of the solution was to be able to reload the last configuration and continue the training from there. In the implementation, the trials were generated as a list of possible configurations, and the program iterated over the list to run them. If the trial broke, the list of configurations and position of the current trial in the list was saved as a pickled dictionary. Using this saved object, the script could easily reload the last configuration and continue training without having to start from the beginning.
An additional useful feature of this solution was that it allowed the author to quickly patch any minor bugs without having to reiterate over the entire list of trials. This was especially useful when the author was testing the code for the first time and had to fix several bugs in the code.

4.8.3 Challenges with External Libraries

Some of the challenges that were faced while using external libraries are as follows:

1. **GPU cache**: While Ray Tune and Optuna manage resources efficiently, they did not clear the GPU cache effectively. PyTorch, by default, holds on to the GPU cache and does not release it until the program is closed for efficiency. This would not be a problem for a single training run, but if many trials were being run, the cache would quickly fill up and cause the training to crash. This does not imply that using Proxy Attention makes it impossible to use such libraries but that it was easier to implement a custom solution.

2. **Cluster** : Both libraries were written to enable running large-scale experiments over multiple machines. While this would be useful for a large-scale project, it added unnecessary complexity to this project as all the experiments were run on a single machine.

3. **Grid Search** : Both libraries mentioned above were designed for hyperparameter tuning and to implement multiple grid search variants. While this would be useful, it would stop many trials that would eventually be useful to analyze. In this project, it was important to have results for each of the trials, and since the author could not find a way to disable the default Early Stopping behaviour as part of the grid search, a simple trial generation algorithm was created instead.

4.9 Optimizations

The following section discusses the optimizations that were implemented to improve the performance of Proxy Attention, training the networks and reducing memory usage.

4.9.1 Proxy Step specific

- Major bottleneck in the training process is the Proxy Attention step, as it applies an XAI algorithm to each of the images in the batch of wrongly classified images - To reduce - Proxy step used as a callback - CPU is used to store because GPU memory is limited - wrong images and labels are stored during an epoch on the CPU - Gradient computation is also disabled for these images as it is not required and unnecessarily increases memory usage - these are then batched and passed to the Proxy step - Proxy step is applied to the batch of images and the gradients are computed - All computations are done on the GPU using PyTorch tensors unlike many libraries that use numpy arrays - Replacing the pixels in the image is done using torch.where which has been shown to be faster than other methods ref? - The gradients are deleted from the GPU after the step is completed to reduce memory usage - All preprocessing steps, label changes etc are done in batches to reduce memory usage and CPU calls - It is a known issue that saving images as png files is slow using Pillow and thus the images are saved as jpeg files instead ref?

4.9.2 Workers

By default, PyTorch uses a single worker to load data from the SSD. This is not ideal, as the resources must be fully utilized. The author used eight workers to load data from the SSD, which improved the training process's performance. Increasing the number of workers beyond a certain point does not necessarily improve performance due to the overhead of transferring data between the CPU and GPU and might lead to detrimental effects.

4.9.3 Mixed Precision

Mixed Precision Training [46] involves computing most of the operations in the network in half-precision (16-bit) and only using full precision (32-bit) for important operations such as the loss function. This allows for much larger batch sizes, faster training, and reduced memory usage. Micikevicius et al. also find that using Mixed Precision training does not significantly affect the model's accuracy. With all these benefits, using Mixed Precision training was a no-brainer for this project.

The only caveat is that only some operations are stable in half precision. Operations like Batch Normalization tend to break when using Mixed Precision training, and unless managed, the model fails to converge. PyTorch supports automatic casting to and from half-precision and this API was used for this project. It is also a registered issue that Transformer models sometimes fail to converge with Mixed Precision due to the way that Attention is calculated (Ref PyTorch Issue #40497), and so for the Vision Transformer [47] model, the author had to use full precision.

4.9.4 No grad

Since a single machine was used for training, it was important to reduce memory usage wherever possible. Since is not necessary to store the gradients for the validation step as they are not used for anything, one of the ways to reduce memory consumption is to disable gradient computation for the validation step. This is done by using the `torch.no_grad()` context manager. The optimizer's `zero_grad()` method is used to clear the gradients (using `set_to_none=True`). The additional parameter `set_to_none` is shown to have better performance (Refer to the Official PyTorch tuning guide for more information.) as it involves fewer operations.

4.9.5 Pillow SIMD

One of the bottlenecks in the training process is loading images from the SSD. A few years ago the image processing library Pillow, used to be one of the majorly used libraries for loading images. In recent times, it has been superceded by the Pillow SIMD library, which uses SIMD instructions to improve performance. It is a drop-in replacement for Pillow and requires no changes to the code but can increase image I/O speeds by a significant margin (Benchmark comparison between Pillow and Pillow SIMD). Add more info about SIMD

4.10 Tensorboard

Tensorboard is a utility for managing and visualizing training logs. In this project, it is used to store the training configurations, metrics, images and other information that is generated during training. Since Tensorboard uses a custom file format to store this information, it can be used to store any information. Unlike many other logging utilities, Tensorboard stores all its logs locally. While storing them online might be useful in some cases, it is more difficult to manage and quite unnecessary for this project. Another useful feature of Tensorboard is the ability to see live updates while training is in progress. This is useful for debugging and ensuring the training is progressing as expected.

4.11 Optimizer

- AdamW is used as the optimizer - lr is set to 1e-3 - weight decay is set to 1e-5 - ref?

4.12 LR scheduler

- One Cycle LR is used as the learning rate scheduler - max lr is set to 2e-3 - ref?

4.13 Loss function

- Cross Entropy Loss is used as the loss function - ref?

4.14 Batch Size Finder

To maximize training performance, a batch size finder 3 is used to find the optimal batch size for each model.

The batch size finder algorithm is rather simple. It starts by testing for a small batch size of 2. This batch size is then successively, either incremented or decremented, based on the current GPU configuration's ability to support that batch of data. A random batch of data with the size that is to be tested is generated and passed through the required model. If the GPU fails to accommodate the current batch of data, the loop terminates, and the required batch size is obtained. The rest of the steps required to train a network are also performed on this randomly generated data. This algorithm remains the same for any model, data type, or other further optimizations applied (such as mixed precision training [46]) and is robust to multiple GPUs being used for training.

Algorithm 3 Batch Size Finder Algorithm

Require: $dataset\_size$
Require: max_batch_size
  $batch\_size \leftarrow 2$
  while TRUE **do**
    if max_batch_size is not $None$ & $batch\_size \geq max\_batch\_size$ **then**
      $batch\_size \leftarrow max\_batch\_size$
    end if
    if $batch\_size \geq dataset\_size$ **then**
      $batch\_size \leftarrow batch\_size//2$
    end if
    if $failed$ is $False$ **then**
      loop
        $inputs \leftarrow random((batch\_size, input\_shape))$
        $targets \leftarrow random((batch\_size, output\_shape))$
        $outputs \leftarrow model(inputs)$
        $loss \leftarrow MSE(outputs, targets)$
        $loss.backward()$
        $optimizer.step()$
        $optimizer.zero\_grad()$
        $failed \leftarrow True$
        $batch\_size \leftarrow batch\_size * 2$
      end loop
    else if $failed$ is $True$ **then**
      $failed \leftarrow False$
      $batch\_size \leftarrow batch\_size//2$
    end if
  end while

4.15 Result Aggregation

The biggest caveat of using Tensorboard is that the logs it generates cannot be directly queried in the interface itself. To overcome this, a custom script was written to query the logs and generate a DataFrame that combines all the logs into a single pandas DataFrame. This makes it possible to not only query the logs but also to perform any kind of analysis on them. This script can easily answer specific queries such as "What is the best accuracy across all the networks for 'gradcam++', 'dogs dataset' and 'proxy_threshold = 0.5'?". This makes it possible to easily compare the performance of different models and configurations.
Since the script for aggregating logs is rather useful, it was made publicly available as a Github Gist.

4.16 Inference

Inference refers to using a trained model to make predictions on new data. In this project, a large number of models were trained. A separate script was created to use any of the previously trained models for inference.
This script follows from the result aggregation and can use queries over the dataframe generated in the previous step. Since the generated dataframe also contains the path to the saved model, this script can use that information along with the names of the architecture, dataset and other hyper-parameters to load the required models easily.
The inference script also contains functions for comparing both the accuracies and the explainability of two pre-trained models given a validation dataset or a list of images.
For a batch of images and given a set of hyperparameters, the script loads two models - one trained with Proxy Attention and one trained without. The same dataloader is passed through both models to obtain predictions. Only EigenGradCAM is used for this evaluation phase to ensure a fair comparison, and since GradCAM++ was used for training, it would not be fair to use it for evaluation as well. modify this a bit

CHAPTER 5

RESULTS

In line with the research questions, the evaluation section aims to quantify the performance gains obtained by using the Proxy Attention method. The section will compare the performance of networks that were trained with, and without Proxy Attention on the basis of classification metrics, and explainability improvements.
Note that complete performance logs can be found in the appendix.

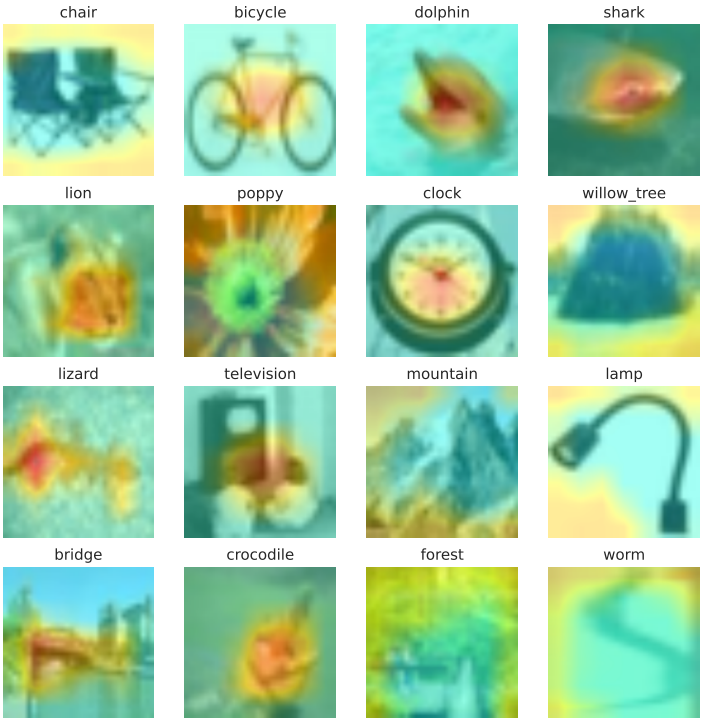
5.1 Accuracy

This section explores the validation accuracy obtained by the models for different hyperparameters and datasets. Since the task at hand is a classification task, this measure is a direct comparison of the performance of the models.

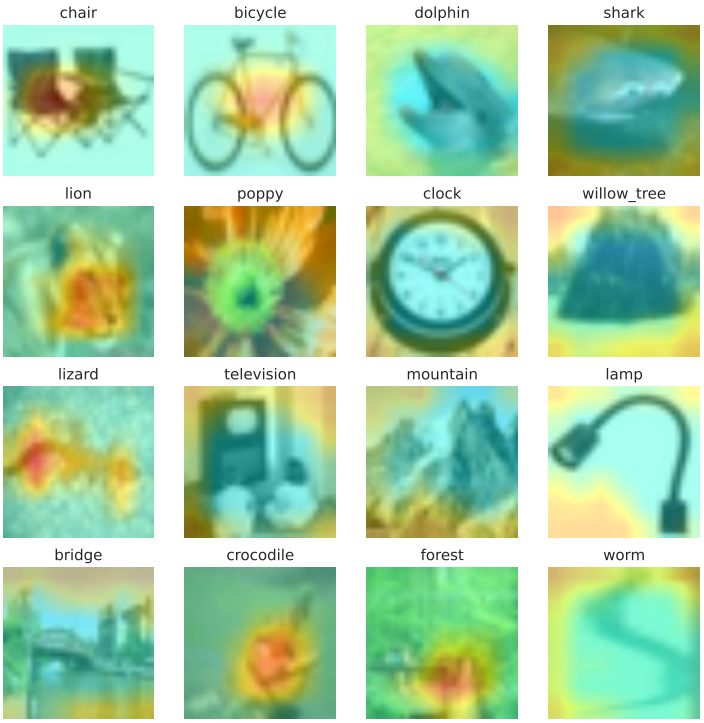
5.2 Explanability

This section explores the explainability of the models for different hyperparameters and datasets by using a trained model to generate attention maps for a given input image. The attention maps are compared between the same network (with the same hyperparameters) trained with and without Proxy Attention.

5.2.1 CIFAR 100

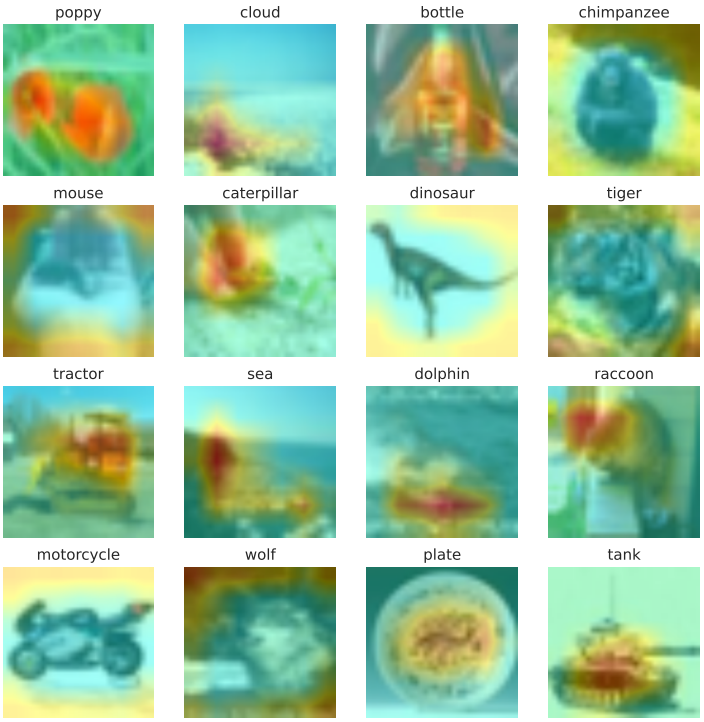


(a) With Proxy Attention

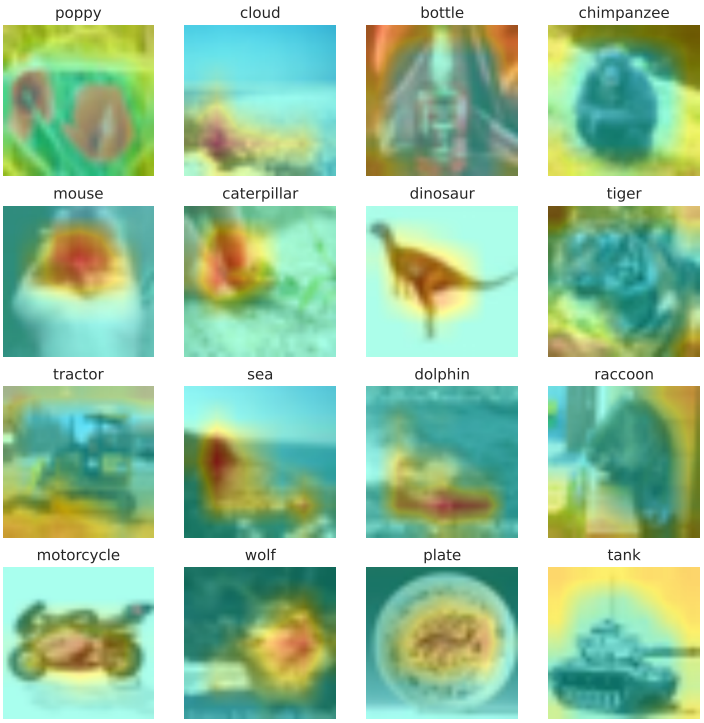


(b) Without Proxy Attention

Figure 5.1: Comparison of attention maps generated by models trained with and without Proxy Attention



(a) With Proxy Attention



(b) Without Proxy Attention

Figure 5.2: Comparison of attention maps generated by models trained with and without Proxy Attention

(a) With Proxy Attention

(b) Without Proxy Attention

Figure 5.3: Comparison of attention maps generated by models trained with and without Proxy Attention

CHAPTER 6

DISCUSSION

6.1 Research Questions

1. Is it possible to create an augmentation technique that uses Attention maps?
2. Is it possible to approximate the effects of Attention from ViTs in a CNN without changing the architecture?
3. Is it possible to make a network converge faster and consequently require less data using the outputs from XAI techniques?
4. Does using Proxy Attention impact the explainability positively?

6.2 General Discussion

- optimizations , memory leaks - managing computational resources and multiple experiments - Improvements with better XAI techniques, better CNNs

CHAPTER 7

CONCLUSION

7.1 Contributions

- Novel augmentation technique that uses attention maps - Better explainability , better performance - Faster convergence, less data - Easy to implement, no change in architecture - Experiments with a large number of hyperparameters and models to test the robustness of the method and find the best configuration - Open source callback code that can be used to easily add Proxy Attention to any existing code base - Script to easily parse tensorboard logs to a unified DataFrame for easy analysis - Scripts to reproduce all the results in this thesis along with training logs

7.2 Lessons Learned

- Combining research from different domains to create a novel method - Importance of hyperparameter tuning - Understading memory leaks and debugging them, how to optimize memory usage - Importance of writing functional (instead of class oriented) code that can be easily reused and modified - Better understanding of Augmentation, XAI techniques - Better understanding of configuring the training loop

7.3 Future Work

- Schedule both the number of proxy steps, number of images for Proxy step based on validation performance - Experiment with more XAI methods - Experiment with smoothing the attention maps before using them for the Proxy step

CHAPTER 8

APPENDIX

BIBLIOGRAPHY

[1] Matthew D. Zeiler and Rob Fergus. *Visualizing and Understanding Convolutional Networks*. Nov. 28, 2013. DOI: 10.48550/arXiv.1311.2901. arXiv: 1311.2901 [cs]. URL: http://arxiv.org/abs/1311.2901 (visited on 11/28/2022). preprint.

[2] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. *Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps*. Apr. 19, 2014. arXiv: 1312.6034 [cs]. URL: http://arxiv.org/abs/1312.6034 (visited on 11/18/2022). preprint.

[3] Haofan Wang et al. *Score-CAM: Score-Weighted Visual Explanations for Convolutional Neural Networks*. Version 2. Apr. 13, 2020. arXiv: 1910.01279 [cs]. URL: http://arxiv.org/abs/1910.01279 (visited on 02/16/2023). preprint.

[4] Ramprasaath R Selvaraju et al. "Grad-CAM: Visual Explanations From Deep Networks via Gradient-Based Localization". In: (), p. 9.

[5] Ramprasaath R. Selvaraju et al. *Grad-CAM: Why Did You Say That?* Jan. 25, 2017. arXiv: 1611.07450 [cs, stat]. URL: http://arxiv.org/abs/1611.07450 (visited on 02/20/2023). preprint.

[6] Jost Tobias Springenberg et al. *Striving for Simplicity: The All Convolutional Net*. Apr. 13, 2015. DOI: 10.48550/arXiv.1412.6806. arXiv: 1412.6806 [cs]. URL: http://arxiv.org/abs/1412.6806 (visited on 11/18/2022). preprint.

[7] Narine Kokhlikyan et al. *Captum: A Unified and Generic Model Interpretability Library for PyTorch*. Sept. 16, 2020. DOI: 10.48550/arXiv.2009.07896. arXiv: 2009.07896 [cs, stat]. URL: http://arxiv.org/abs/2009.07896 (visited on 04/04/2023). preprint.

[8] Daniel Smilkov et al. *SmoothGrad: Removing Noise by Adding Noise*. June 12, 2017. DOI: 10.48550/arXiv.1706.03825. arXiv: 1706.03825 [cs, stat]. URL: http://arxiv.org/abs/1706.03825 (visited on 11/28/2022). preprint.

[9] Lorenz Richter et al. "VarGrad: A Low-Variance Gradient Estimator for Variational Inference". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 13481–13492. URL: https://proceedings.neurips.cc/paper/2020/hash/9c22c0b51b3202246463e986c7e205df-Abstract.html (visited on 02/20/2023).

[10] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. *Axiomatic Attribution for Deep Networks*. June 12, 2017. DOI: 10.48550/arXiv.1703.01365. arXiv: 1703.01365 [cs]. URL: http://arxiv.org/abs/1703.01365 (visited on 03/24/2023). preprint.

[11] Vitali Petsiuk, Abir Das, and Kate Saenko. *RISE: Randomized Input Sampling for Explanation of Black-box Models*. Sept. 25, 2018. arXiv: 1806.07421 [cs]. URL: http://arxiv.org/abs/1806.07421 (visited on 02/20/2023). preprint.

[12] Taiki Oyama and Takao Yamanaka. "Influence of Image Classification Accuracy on Saliency Map Estimation". In: *CAAI Transactions on Intelligence Technology* 3.3 (2018), pp. 140–152. ISSN: 2468-2322. DOI: 10.1049/trit.2018.1012. URL: https://onlinelibrary.wiley.com/doi/abs/10.1049/trit.2018.1012 (visited on 10/03/2022).

[13] Fred Hohman et al. *Summit: Scaling Deep Learning Interpretability by Visualizing Activation and Attribution Summarizations*. Sept. 2, 2019. arXiv: 1904.02323 [cs]. URL: http://arxiv.org/abs/1904.02323 (visited on 02/20/2023). preprint.

[14] Kedar Dhamdhere, Mukund Sundararajan, and Qiqi Yan. *How Important Is a Neuron?* May 30, 2018. DOI: 10.48550/arXiv.1805.12233. arXiv: 1805.12233 [cs, stat]. URL: http://arxiv.org/abs/1805.12233 (visited on 11/28/2022). preprint.

[15] Christian Szegedy et al. *Going Deeper with Convolutions*. Sept. 16, 2014. DOI: 10.48550/arXiv.1409.4842. arXiv: 1409.4842 [cs]. URL: http://arxiv.org/abs/1409.4842 (visited on 04/29/2023). preprint.

[16] Dan Hendrycks et al. *AugMix: A Simple Data Processing Method to Improve Robustness and Uncertainty*. Feb. 17, 2020. arXiv: 1912.02781 [cs, stat]. URL: http://arxiv.org/abs/1912.02781 (visited on 01/16/2023). preprint.

[17] Jianhua Lin. "Divergence Measures Based on the Shannon Entropy". In: ().

[18] Terrance DeVries and Graham W. Taylor. *Improved Regularization of Convolutional Neural Networks with Cutout*. Nov. 29, 2017. DOI: 10.48550/arXiv.1708.04552. arXiv: 1708.04552 [cs]. URL: http://arxiv.org/abs/1708.04552 (visited on 03/27/2023). preprint.

[19] Sangdoo Yun et al. "CutMix: Regularization Strategy to Train Strong Classifiers With Localizable Features". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019 IEEE/CVF International Conference on Computer Vision (ICCV). Seoul, Korea (South): IEEE, Oct. 2019, pp. 6022–6031. ISBN: 978-1-72814-803-8. DOI: 10.1109/ICCV.2019.00612. URL: https://ieeexplore.ieee.org/document/9008296/ (visited on 02/20/2023).

[20] Hongyi Zhang et al. *Mixup: Beyond Empirical Risk Minimization*. Apr. 27, 2018. DOI: 10.48550/arXiv.1710.09412. arXiv: 1710.09412 [cs, stat]. URL: http://arxiv.org/abs/1710.09412 (visited on 03/27/2023). preprint.

[21] Devesh Walawalkar et al. *Attentive CutMix: An Enhanced Data Augmentation Approach for Deep Learning Based Image Classification*. Apr. 5, 2020. DOI: 10.48550/arXiv.2003.13048. arXiv: 2003.13048 [cs]. URL: http://arxiv.org/abs/2003.13048 (visited on 03/29/2023). preprint.

[22] Geoff French, Avital Oliver, and Tim Salimans. *Milking CowMask for Semi-Supervised Image Classification*. June 5, 2020. DOI: 10.48550/arXiv.2003.12022. arXiv: 2003.12022 [cs]. URL: http://arxiv.org/abs/2003.12022 (visited on 03/31/2023). preprint.

[23] Zhun Zhong et al. “Random Erasing Data Augmentation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.07 (Apr. 3, 2020), pp. 13001–13008. ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v34i07.7000. URL: https://aaai.org/ojs/index.php/AAAI/article/view/7000 (visited on 10/21/2022).

[24] Krishna Kumar Singh et al. *Hide-and-Seek: A Data Augmentation Technique for Weakly-Supervised Localization and Beyond*. Nov. 6, 2018. DOI: 10.48550/arXiv.1811.02545. arXiv: 1811.02545 [cs]. URL: http://arxiv.org/abs/1811.02545 (visited on 03/27/2023). preprint.

[25] Pengguang Chen et al. *GridMask Data Augmentation*. Jan. 13, 2020. DOI: 10.48550/arXiv.2001.04086. arXiv: 2001.04086 [cs]. URL: http://arxiv.org/abs/2001.04086 (visited on 03/31/2023). preprint.

[26] Bolei Zhou et al. “Learning Deep Features for Discriminative Localization”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV, USA: IEEE, June 2016, pp. 2921–2929. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.319. URL: http://ieeexplore.ieee.org/document/7780688/ (visited on 02/20/2023).

[27] Jie Qin et al. *ResizeMix: Mixing Data with Preserved Object Information and True Labels*. Dec. 20, 2020. arXiv: 2012.11101 [cs]. URL: http://arxiv.org/abs/2012.11101 (visited on 03/29/2023). preprint.

[28] Ryo Takahashi, Takashi Matsubara, and Kuniaki Uehara. “Data Augmentation Using Random Image Cropping and Patching for Deep CNNs”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 30.9 (Sept. 2020), pp. 2917–2931. ISSN: 1051-8215, 1558-2205. DOI: 10.1109/TCSVT.2019.2935128. arXiv: 1811.09030 [cs]. URL: http://arxiv.org/abs/1811.09030 (visited on 03/30/2023).

[29] Hiroshi Inoue. *Data Augmentation by Pairing Samples for Images Classification*. Apr. 11, 2018. arXiv: 1801.02929 [cs, stat]. URL: http://arxiv.org/abs/1801.02929 (visited on 03/30/2023). preprint.

[30] Jin-Ha Lee et al. “SmoothMix: A Simple Yet Effective Data Augmentation to Train Robust Classifiers”. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops. 2020, pp. 756–757. URL: https://openaccess.thecvf.com/content_CVPRW_2020/html/w45/Lee_SmoothMix_A_Simple_Yet_Effective_Data_Augmentation_to_Train_Robust_CVPRW_2020_paper.html (visited on 03/29/2023).

[31] *SMOTE: Synthetic Minority Over-sampling Technique | Journal of Artificial Intelligence Research*. URL: https://www.jair.org/index.php/jair/article/view/10302 (visited on 03/31/2023).

[32] Shaoli Huang, Xinchao Wang, and Dacheng Tao. “SnapMix: Semantically Proportional Mixing for Augmenting Fine-grained Data”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.2 (2 May 18, 2021), pp. 1628–1636. ISSN: 2374-3468. DOI: 10.1609/aaai.v35i2.16255. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16255 (visited on 03/31/2023).

[33] Jie Cao et al. “ReMix: Towards Image-to-Image Translation With Limited Data”. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021, pp. 15018–15027. URL: https://openaccess.thecvf.com/content/CVPR2021/html/Cao_ReMix_Towards_Image-to-Image_Translation_With_Limited_Data_CVPR_2021_paper.html (visited on 03/31/2023).

[34] Nikita Dvornik, Julien Mairal, and Cordelia Schmid. “Modeling Visual Context Is Key to Augmenting Object Detection Datasets”. In: Proceedings of the European Conference on Computer Vision (ECCV). 2018, pp. 364–380. URL: https://openaccess.thecvf.com/content_ECCV_2018/html/NIKITA_DVORNIK_Modeling_Visual_Context_ECCV_2018_paper.html (visited on 10/21/2022).

[35] Jang-Hyun Kim, Wonho Choo, and Hyun Oh Song. “Puzzle Mix: Exploiting Saliency and Local Statistics for Optimal Mixup”. In: *Proceedings of the 37th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, Nov. 21, 2020, pp. 5275–5285. URL: https://proceedings.mlr.press/v119/kim20b.html (visited on 04/04/2023).

[36] Alex Krizhevsky. “Learning Multiple Layers of Features from Tiny Images”. In: ().

[37] Aditya Khosla et al. “Novel Dataset for Fine-Grained Image Categorization: Stanford Dogs”. In: ().

[38] Fei-Fei Li et al. *Caltech 101*. 2022. DOI: 10.22002/D1.20086.

[39] A. F. M. Shahab Uddin et al. *SaliencyMix: A Saliency Guided Data Augmentation Strategy for Better Regularization*. July 27, 2021. arXiv: 2006.01791 [cs, stat]. URL: http://arxiv.org/abs/2006.01791 (visited on 04/11/2023). preprint.

[40] J. Schmidhuber. “A ‘Self-Referential’ Weight Matrix”. In: *ICANN ’93*. Ed. by Stan Gielen and Bert Kappen. London: Springer London, 1993, pp. 446–450. ISBN: 978-3-540-19839-0 978-1-4471-2063-6. DOI: 10.1007/978-1-4471-2063-6_107. URL: http://link.springer.com/10.1007/978-1-4471-2063-6_107 (visited on 04/26/2023).

[41] Kazuki Irie et al. “A Modern Self-Referential Weight Matrix That Learns to Modify Itself”. In: *Proceedings of the 39th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, June 28, 2022, pp. 9660–9677. URL: https://proceedings.mlr.press/v162/irie22b.html (visited on 04/26/2023).

[42] Aditya Chattopadhay et al. “Grad-CAM++: Generalized Gradient-Based Visual Explanations for Deep Convolutional Networks”. In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2018 IEEE Winter Conference on Applications of Computer Vision (WACV). Lake Tahoe, NV: IEEE, Mar. 2018, pp. 839–847. ISBN: 978-1-5386-4886-5. DOI: 10.1109/WACV.2018.00097. URL: https://ieeexplore.ieee.org/document/8354201/ (visited on 02/20/2023).

[43] J. Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *CVPR09*. 2009.

[44] Ross Wightman. *PyTorch Image Models*. https://github.com/rwightman/pytorch-image-models. 2019. DOI: 10.5281/zenodo.4414861.

[45] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[46] Paulius Micikevicius et al. “Mixed Precision Training”. 2017. arXiv: 1710.03740.

[47] Alexey Dosovitskiy et al. *An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale*. June 3, 2021. DOI: 10.48550/arXiv.2010.11929. arXiv: 2010.11929 [cs]. URL: http://arxiv.org/abs/2010.11929 (visited on 10/21/2022). preprint.