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

Proxy Attention : Comparing and Combining Augmentation with Attention

Graduation Project
(Computational Intelligence and Robotics)

Subhaditya Mukherjee (s4747925)
April 26, 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

CHAPTER 1

INTRODUCTION

CHAPTER 2

BACKGROUND

2.1 Interpretability

- Need for Interpretability

2.2 Gradient Based Explanations

- Taxonomy

2.3 Augmentation

- Taxonomy

2.4 Datasets

To test Proxy Attention, the following datasets were used. Note: Images are resized to 224x224 pixels for consistency. These batch visualizations are generated by the author using the torchvision and matplotlib libraries.

2.4.1 CIFAR 100

The CIFAR 100 dataset, introduced by [1] is an image dataset with 60000 color images with dimensions 32x32 pixels. As the name suggests, the dataset has 100 unique classes. Each of these classes have 500 training images. Some of the classes are - airplane, bird, truck, ship, deer and dog. This dataset is used as a coarse grained classification dataset in this project.

2.4.2 Stanford dogs

2.4.3 Imagenette

2.4.4 ASL

2.4.5 Food-101

Plant Village

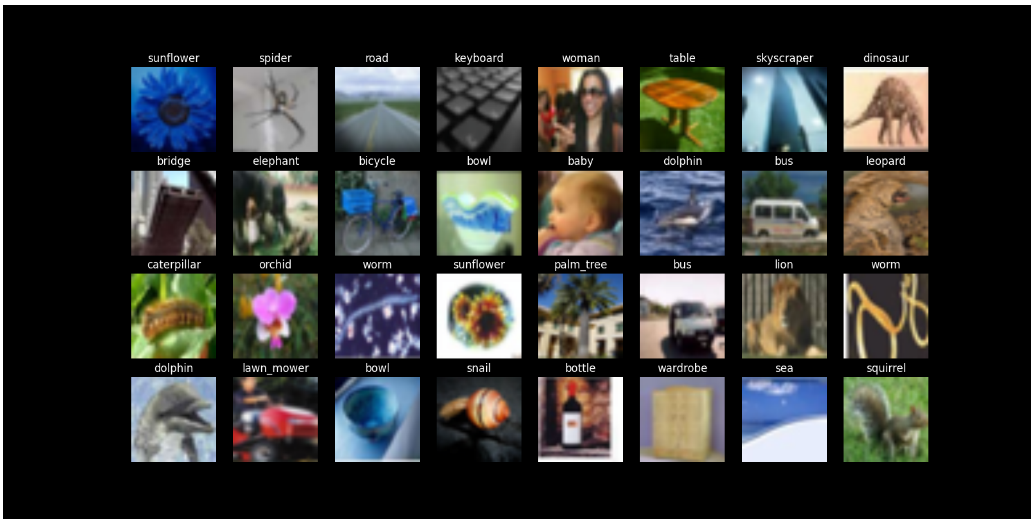

Figure 2.1: CIFAR100 Sample

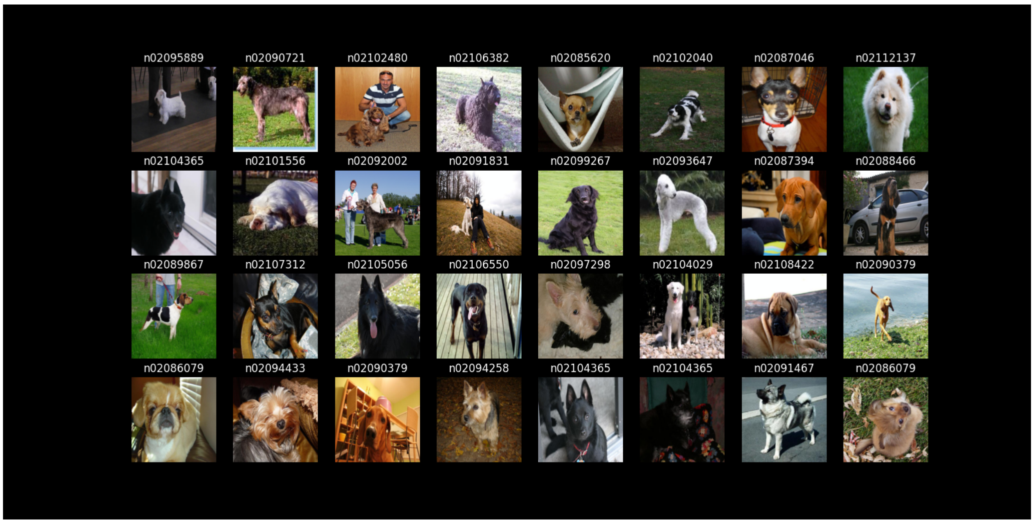

Figure 2.2: Stanford Dogs Sample

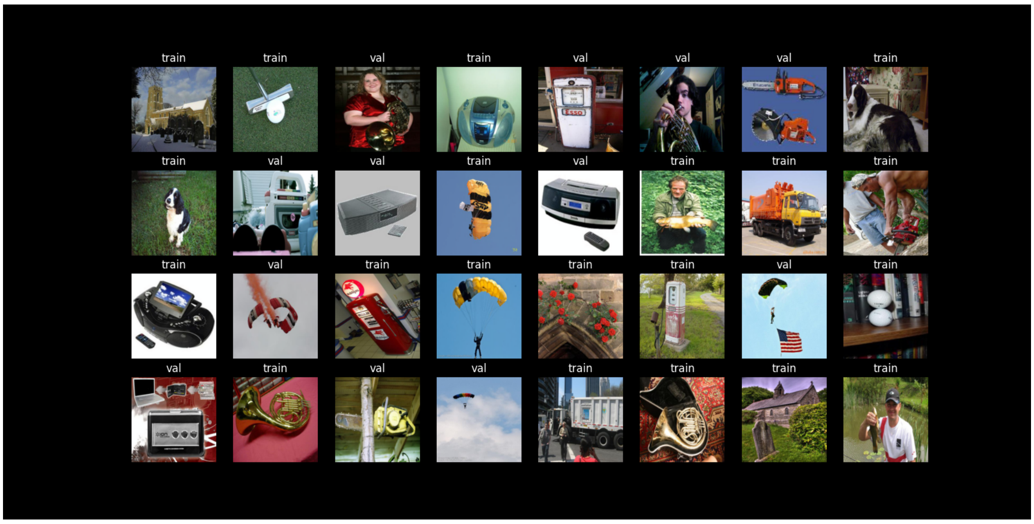

Figure 2.3: Imagenette Sample

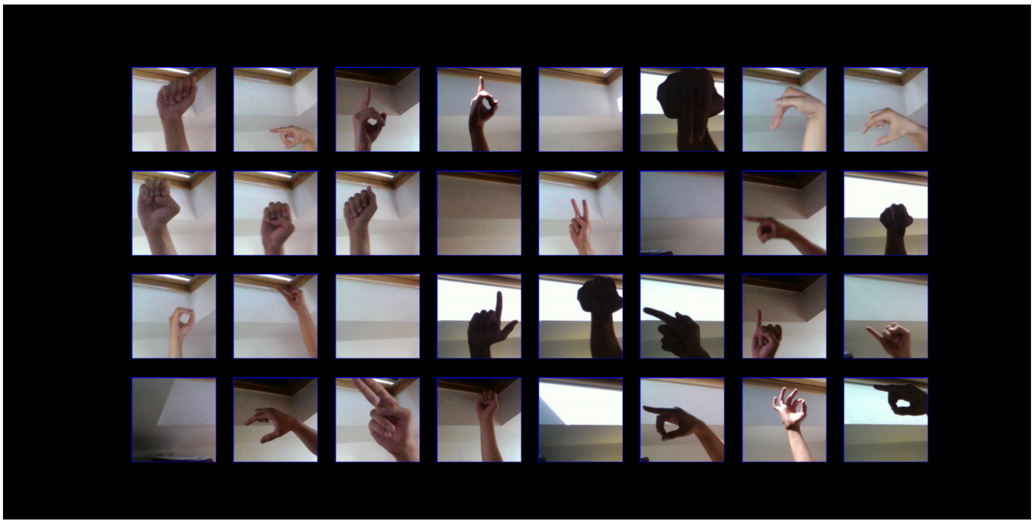

Figure 2.4: ASL Sample

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. [2] termed DeconvNet. DeconvNet works by inverting the operations that the network performs in the forward pass. After attaching the DeconvNet layers to the network, propagating through these layers gives a representation of features that the original CNN possessed. For a single class, the relevant reconstruction can be obtained 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. The Conv layer is replaced by a Deconv layer and the ReLU operation has the negative values clamped. While the pooling operation is not strictly invertible, the authors use switch variables that store the position of the maximum value for each pooling operation. While the DeconvNet works to a certain extent, the results are not as accurate as the ones obtained by other methods and are also biased towards the representations of the first layer.

Building on the DeconvNet, Simonyan et al. [3] 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 prediction the Saliency score $S_c(I) \approx w^T I + b$. In this equation, w and b are the weights and biases of the network 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, rearrage 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 color 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 a lot of noise. This leads to issues for more complex images. Many of the issues with Vanilla Gradients and DeconvNets [2] have been addressed by the methods proposed in the following papers.

In another paper, the authors propose a score weighted approach (ScoreCAM) to create saliency maps [4]. 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 then upsampled and normalized to the range of $[0, 1]$. The portions of the activation maps that were highlighted 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 relative importance of all the activation maps. 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 mathmaticaly represented as $L^c_{ScoreCAM} = ReLU(\underset{k}{\Sigma} w^c_k A^k)$, where k represents the index considered, c represents the current class and S_k represents the outputs of the aforementioned SoftMax layer. The authors find that the maps obtained using ScoreCAM are less noisy and using this method removes dependancy on unstable gradients as compared to other methods.

A variant of GradCAM [5] was proposed by Selvaraju et al. [6] 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 [7].

In combination with attribution methods, Noise Tunnel [8] 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 of the 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 [9], 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 [10] 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 [11] computes the contribution of each pixel in the image towards the final prediction. The output of the model 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 of the steps are 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 pixelwise 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 contribute equally to the final result.

Petsiuk et al. propose RISE [12], a saliency method that randomly alters the input images by applying random noise to each of them. After model predictions are obtained, the saliency map is generated by a combination of the partial maps over each of the modified images. RISE improves accuracy but needs a lot of computation time considering that multiple models need to be trained for each of the random noise samples.

With regard to the relationship between saliency maps and image classification accuracy, Oyama et al. [13] found a strong correlation between the two. The authors found that both 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 a large number of categories supresses the overfitting for the objects present in the training dataset. On training their proposed network ReadoutNet on a fixation task (a 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 intepreting the influence of a single image or neuron, Hohman et al. propose Summit, [14] a novel scalable summarization algorithm. Summit creates an attribution graph that distills the influence of neurons and substructures throughout the network that are used to make the final prediction. The attribution graph is created as a result of 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. To aggregate the activations, after a forward pass through the network, the activation channels maximums are obtained. These are then filtered by class and aggregated by either 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 layers of the network (l). How important channel i of the layer $l-1$ is determined by the aggregate tensor I_{cij}^l 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 $Y_{:,:,j} = \Sigma_{i=1}^{C_{l-1}} X_{:,:,i} * K_{:,:,i}^{(j)}$. The value $X_{:,:,i} * K_{:,:,i}^{(j)}$ is the contribution of the current channel from the previous layer and the maximum of this value is used to generate the influence map.

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
Conductance
Deep Fool
Deep Lift
Generalizing Adversarial Exp With Gradcam
Shap
Smooth Grad
Smooth Grad Square
Lime
Sp Lime
Lrp
Var Grad
Visualizing 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 [15] first applies multiple transformations randomly and in parallel chains to each image. These transformations can include combinations of Translation, Rotation, Shearing etc. The outputs of these combinations are then mixed to form a new image, which is then further mixed with the original image to form the new image. This combination is done to improve 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 [16] to ensure that the images are stable across a range of 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 [17] propose an augmentation method they call Cutout. In this method, random sized square patches are removed from the images by replacing the corresponding pixels with a constant value (usually 0). Selecting the region involves picking a random pixel value and then creating a uniform sized square around the chosen pixel. The authors also find that Cutout performs better in combination with other methods rather than just being used by itself. Cutout can be expressed as an element-wise multiplication operation $x_{cutout} = x \odot M$, where 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 [17], where the chosen patch is replaced with zero pixels, in CutMix [18] 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 $\widetilde{x} = M \odot x_A + (1-M) \odot x_B$; $\widetilde{y} = \lambda y_A + (1-\lambda) y_B$. where x is an RGB image, y is the respective label, M is a binary mask of the patch of the image that will be dropped and $\odot$ represents element wise multiplication. The new training sample $\widetilde{x}, \widetilde{y}$ 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 [19] but differs in the sense that CutMix focuses on generating locally natural images. Building up on [18],

Walawalkar et al. propose an alternative method of replacing patches in an image they call Attentive CutMix [20]. In this method, instead of randomly pasting patches in the image, a pre-trained network is used to identify attentive regions from the image. Similar to the earlier approach, these patches are then mapped back to the original image. Doing so allows the network to select backround regions that are important for the task while also updating the label information.

Many of the algorithms use rectangular or square shaped masks. While they are effective, French et al. propose Cow Mask [21], a new method of masking that uses irregularly shaped masks with a Gaussian filter to reduce noise. The authors also propose two methods of mixing, one that builds up on Random Erasing [22], and another that uses Cut Mix [18]. 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.

Another approach involving a cut-paste methodology was proposed by [23]. In their paper, the authors propose a new method of augmentation that extracts instances of objects from the images and instead of pasting them on other images, they are pasted on randomly chosen backgrounds. This method leads to pixel artifacts in the images as selecting the objects is a noisy process. To overcome the drop in performance as a result of this, the authors apply a Gaussian blur and poisson blending to the boundaries of the pasted objects. Further augmentaion is applied before pasting the objects by rotation, occlusion and truncation. The authors also find that this approach makes the network more robust to artifacts in the images.

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 are then turned off with a given probability. These sub-grids can be connected or independant 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, 17, 22]) 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 is effective in preserving important parts of the image.

Zhang et al. propose another method of data augmentation 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 [17] 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 [22]. In Random Erasing, contiguous rectangular regions are selected and replaced at random with random upper and lower limits chosen for both region area and aspect ratio. For object detection tasks, a region aware detection algorithm is applied to make the network more robust to occlusion. Note that Cutout removes square patches, while Random Erasing either removes square or rectangular patches.

Many of the augmentation methods that rely on randomly choosing regions to cut and paste from sometimes fail to work well with regions that lack object information. ResizeMix [27] tackles this problem by replacing the patch with a proportional resized version of the selected image. This method is similar to CutMix [18] 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 then 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 multple variants of RICAP that use different points of origin for cropping. They find that the method works best when the cropped regions use the corners as the origin as it allows the network to see more of the image.

While algorithms like Mixup [19] modify the labels of the image proportional to the amount of mixing between the original and the target images, Sample Pairing [29] maintains the same training labels. In their paper, Inoue et al. propose a method that merges images not by cut and paste but by averaging their pixel intensites.

Sample Pairing follows an interval based augmentation policy, where the network is first trained for a 100 epochs normally before being introduced to the mixed images. This process is also repeated cyclicaly with eight epochs of training with mixed images followed by 2 epochs of training with normal images only.
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 [18] and Cutout [17] 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 methods of data augmentation 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 neighbors 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 a useful tool to have.
Huang et al. propose SnapMix [32], where choosing the size of the patch 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 are of different sizes. 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 bacgrounds 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 there are many techniques based on Mixup [19], they are mostly focused on generating new samples of images from the existing data. Doing so is useful, but sometimes leads to the generation of examples that confuse the network and are not representative of the actual 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 Augmentaiton with curriculum leanring Co mixup Image Mixing and deletion
Keep augment Latent space interpo Randaugment Random distortion Saliencymix
Spec augment

3.2.1 Summary

3.2.2 Limitations

- Context
- Does not make use of what the network knows
- Does not help the network learn from its mistakes

3.3 Architectures

Resnet 18, 50
VGG
Vision Transformer

CHAPTER 4

IMPLEMENTATION

4.1 Overview

4.2 Design Decisions

4.3 Proxy Attention

Algorithm 1 Single Batch Proxy Attention

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

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

4.4 Challenges and their Potential Solutions

Being a novel method, there were many challenges that were faced while implementing Proxy Attention. While it was not possible to solve all of the issues faced, the author tried to tackle as many as possible. Many of these issues were posed as optimization problems and were taken into account as hyperparameters that could be tuned to improve performance. This section discusses the possible solutions that were tested. Further details about each parameter can be found in 4.4.3, while the results of the experiments are discussed in **??**.

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 either local or global stastistical information from the images for replacement. All these methods can be computed either per image, per batch or over the 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 in the sense that they lead to significant information loss. In many cases, if a large number of images have their values replaced with these values, then the model might become biased towards predicting a specific class when an image contains a large number of pixels with these values. Due to this reason, these methods were not considered for the final implementation.

Data Augmentation Based Replacement

Data Augmentation techniques essentially involve computing some transformation over images. Many of these methods were covered in the literature survey 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 [36] use measures of saliency to find patches from other images in the dataset that are used to replace the chosen pixels. Proxy Attention was inspired by these methods but instead of replacing patches of the image or deleting pixels, it uses a graident based method to downweight the pixels that might have led to the wrong prediction. This method moves away from using naive statistical information but enables the model to eventually learn from the mistakes that it made.

GAN Based Replacement

Modifying the Weights

Instead of replacing the pixels themselves, another possible method would be to modify the weights of the network directly. While there are many research papers that elaborate on methods to perform this procedure, this domain is not researched enough yet to be used easily. Research on this domain has been done from the early 90s [37], but practical implementation of such a network that learns to modify its own weight while training have not been extremely successful [38]. That being the case, implementing such a method is left to future research.
add some papers

Multiply with Attention Map

The method that was chosen for this research does not replace the pixels of the image directly, but weights them using the attention map generated by passing the image through the trained model. The obtained attention map is thus multiplied onto the original image. In line with the principles of proxy attention, this is done to allow the network to understand that the parts of the image that 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 before. Because, after the Proxy Attention step, the images are slightly modified, if the network has not learnt 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 a image, the number of pixels that are weighted 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.

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 might then lead to additional artefacts in the generated images. It might also be the case that 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 seems to work well, the masks themselves have edge artefacts that might lead to corrupting some regions of the image. These artefacts are further amplified for smaller image sizes and might impact performance in the log run. Potential solutions include:

1. Smoothing the masks before applying them to the image using techniques such as Eigen Smoothing. This would potentially 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

Gradient Method

There are many gradient based methods that are available for generating attention maps from trained networks. While many of these methods were mentioned in the survey, it was not possible to test all of them. Since the effectiveness of Proxy Attention depends quite a bit on the gradient method used, it was important to test them.

The important factor that was considered while choosing these methods was the difference in complexity and the power of explanation that they provide. While algorithms like GradCAM++ [39] provide more nuanced and better explanations of the image, older algorithms like Vanilla Gradients [2] 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 indeed is the case, then it would be possible to use more powerful methods to further improve performance in the future. The gradient methods that were tested are as follows:

- **GradCAM++** [39].
- **GradCAM** [5]

Gradient Threshold Considered

Every gradient method considered results in the generation of a heatmap where the higher the activation, the more important the pixel is. The activations are mapped to a range of $[0, 1]$ 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 downweighted, it was important to choose a threshold value that would result in the best classification accuracy.
This is a balancing act as choosing too small of a threshold would result in larger parts of the image being downweighted, while choosing too large of a threshold would result in the image being downweighted too little and hence being too close to the original image to make any difference.
The threshold values that were considered are as follows:

Multiply Weight

Proxy Step Schedule

Subset Of Wrongly Classified

4.5 Data Loading and Pre Processing

4.5.1 Directory structure

4.5.2 Label function

4.5.3 Workers

4.5.4 Clearing proxy images

For every iteration of the proxy attention step, the images are saved locally. That being the case, it is possible to use these generated images over further iterations of the proxy attention step. Since these images are replacement over the original image from the data set, it is possible to use these images as a direct substitute for the original images in the data set. Note that doing so would lead to the network being given more images in the case of using Proxy Attention during training, which is potentially an unfair comparison. To avoid this issue, only a single image is chosen during the data loading process. This thus becomes a hyper parameter 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 to not perform the step.
In the long run, the option to persist the images across iterations might potentially lead to the network learning artefacts that were 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 it's substitute but not both.

4.5.5 Splits

4.5.6 Augmentations

Imagenet Normalize Tensor

4.6 Architectures

TIMM

4.7 Grid Search

To test the effectiveness of Proxy Attention and to find the best combination of hyper parameters, a grid search was performed. The grid search was performed on a single machine with a single GPU and 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 intial sweep over the hyperparameters was performed using a low memory network (ResNet18 [40]), a subset of the Dogs dataset ([41]), a simple graident method (GradCAM [5]) 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 hyper parameters 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 hyper parameters, due to lack of resources and time, the parameters were semi-manually chosen instead.

4.8 Training Resumption

This project required quite a few experiments to be performed to find the best combination of hyper parameters. Due to limitations in the amount of time and resources available, it was important to be able to resume training in case of any interruptions. The author intially tried using libraries such as Optuna and Ray Tune but these did not play well on a single machine. (Ref 4.8.3) Considering the scope of this project, a custom solution was implemented instead.

4.8.1 Checkpoints

While checkpoints are almost always a good idea to have, they were especially important in this project. The Proxy Attention step is applied in 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 and so it was important to have checkpoints to resume training from. 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

Another challenge of training on a single machine was that the training process could be interrupted at any time. Since multiple trials were being run, it was important to be able to reload the last configuration and continue training from there. The trials were generated as a list of possible configurations and the author iterated over the list to run the trials. If the trail 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 author could reload the last configuration and continue training easily.

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 do 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 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 of these 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 for this project as all the experiments were run on a single machine.

3. **Grid Search** : Both of the libraries mentioned above were designed to be used for hyper parameter tuning and they implement multiple variants of grid search. While this would be useful, it would stop a lot of trials that would have eventually been useful to analyze. In this project, it was important to be able to have results for each of the trials and the author could not find a way to disable the default Early Stopping behavior as part of the grid search.

4.9 Optimizations

4.9.1 Mixed Precision

Mixed Precision training [42] 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 overall reduced memory usage. Micikevicius et al. also find that using Mixed Precision training does not significantly affect the accuracy of the model. With all of these benefits, using Mixed Precision training was a no brainer for this project.

The only caveat is that not all operations are yet 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 [43] model, the author had to use full precision.

4.9.2 No grad

4.9.3 Batched Proxy step

4.10 Tensorboard

Tensorboard is a utility for managing and visualizating 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 kind of information. Unlike a lot of other logging utilities, Tensorboard stores all it's 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 making sure that the training is progressing as expected.

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. Specific queries such as "What is the best accuracy across all the networks for 'gradcam++', 'dogs dataset' and 'proxy_threshold = 0.5'?" can be easily answered using this script. This makes it possible to easily compare the performance of different models and different configurations.

Since the script for aggregating logs is rather useful, it was made publicly available as a Github Gist.

4.11 Optimizer

4.12 LR scheduler

4.13 Loss function

4.14 Batch Size Finder

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

The batch size finder algorithm is rather simple. It is starts off by testing for a small batch size of 2. This batch size is then successively, either incremented, or decremented based on the ability of the current GPU configuration to be able to support that batch of data. A random batch of data with the size that is to be tested is generated and is passed through the required model. The rest of the steps required to train a network are also performed on this randomly generated data. If the GPU fails to accommodate the current batch of data, the loop terminates and the required batch size is obtained. This algorithm remains the same for any model, any data type, any other further optimisations applied (such as mixed precision training [42]), 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

4.16 Inference

CHAPTER 5

EVALUATION

5.1 Metric Based Analysis

5.2 Visual Based Analysis

5.3 Summary

CHAPTER 6

CONCLUSION

6.1 Contributions

6.2 Lessons Learned

6.3 Future Work

CHAPTER 7

APPENDIX

BIBLIOGRAPHY

[1] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: ().

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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).

[11] 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.

[12] 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.

[13] 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).

[14] 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.

[15] 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.

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

[17] 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.

[18] 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).

[19] 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.

[20] 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.

[21] 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.

[22] 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).

[23] Debidatta Dwibedi, Ishan Misra, and Martial Hebert. "Cut, Paste and Learn: Surprisingly Easy Synthesis for Instance Detection". In: Proceedings of the IEEE International Conference on Computer Vision. 2017, pp. 1301–1310. URL: https://openaccess.thecvf.com/content_iccv_2017/html/Dwibedi_Cut_Paste_and_ICCV_2017_paper.html (visited on 03/31/2023).

[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] 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.

[37] 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).

[38] 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).

[39] 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).

[40] 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.

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

[42] Paulius Micikevicius et al. "Mixed Precision Training". 2017. arXiv: 1710.03740.

[43] 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.