Stockholms universitet

CIFAR-10 Classification Task Based on ResNet-18 and D-RISE Saliency Map Analysis

CIFAR-10 Klassificeringsuppgift Baserat på ResNet-18 och D-RISE Saliency Map-analys

Xin Tang

Handledare: Chun-Biu Li

Examinator: Josefin Ahlkrona

Inlämningsdatum: 2024-05-20

Kandidatuppsats i datalogi
Bachelor Thesis in Computer Science

Abstract

Deep learning, especially Convolutional Neural Networks (CNNs), has made significant achievements in image recognition and classification tasks. Residual Network (ResNet), a revolutionary neural network architecture developed from CNNs, excels in visual tasks such as image recognition and object detection. Given the complexity of deep learning networks and their operation as "black-box", the explainability of models has become a key issue in the field of AI. By using saliency maps to reveal key features during the model's processing, we can enhance our understanding and trust in the model's classification decisions. This thesis employs the ResNet-18 architecture for image classification on the CIFAR-10 dataset, created by the Canadian Institute for Advanced Research. The study aims to enhance the models explainability by demonstrating the importance of variables through saliency maps. This thesis not only validates the generalization ability of the model but also successfully reproduces and optimizes the Detector Randomized Input Sampling for Explanation (D-RISE) algorithm applied to the ResNet-18 classification task, which visually displays the key features captured by the model and reveals the models decision logic. Lastly, in the discussion section of the thesis, we give different proposals for the similarity score calculation method in the D-RISE algorithm to enhance the mathematical explainability behind the algorithm.

Sammanfattning

Djupinlärning, särskilt konvolutionella neurala nätverk (CNN), har gjort betydande framsteg inom bildigenkänning och klassificeringsuppgifter. Residual Network (ResNet), en revolutionerande arkitektur för neurala nätverk utvecklad från CNN, utmärker sig i visuella uppgifter såsom bildigenkänning och objektdetektering. Med tanke på djupinlärningsnätverkens komplexitet och deras funktion som "svarta lådor" har förklaringsbarheten av modeller blivit en nyckelfråga inom AI-området. Genom att använda salienskartor för att avslöja huvuddrag under modellens bearbetning kan vi förbättra vår förståelse och vårt förtroende för modellens klassificeringsbeslut. Denna uppsats använder ResNet-18-arkitekturen för bildklassificering på CIFAR-10-datasetet och försöker förbättra modellens förklaringsbarhet genom att demonstrera variablernas betydelse med hjälp av salienskartor. Uppsatsen validerar inte bara modellens generaliseringsförmåga, utan reproducerar och optimerar också framgångsrikt D-RISE-algoritmen som tillämpas på ResNet-18-klassificeringsuppgiften, vilket visuellt visar de huvuddrag som modellen fångar under bearbetningen och avslöjar modellens beslutslogik. Slutligen föreslår vi i diskussionsavsnittet ändringar i beräkningsmetoden för likhetspoäng i D-RISE-algoritmen för att förbättra den matematiska förklaringsbarheten bakom algoritmen.

Acknowledgements

I would like to express my heartfelt thanks to my supervisors, *Chun-Biu Li*, at the Department of Mathematics, Stockholm University. His guidance and expertise have been invaluable throughout this project, providing me with the support and knowledge necessary to navigate this complex and challenging journey. His enthusiastic and exploratory attitude towards scientific research has also had a positive impact on my thesis work. Under his guidance, I have come to understand the learning process of researching a topic and the methodology of solving problems.

Contents

1 Introduction

Since the 1990s, image analysis has become an important area of research in artificial intelligence, and in the past 20 years, machine learning and deep learning have made rapid advancements in image processing. Starting in 2010, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC)[5] has become one of the main drivers of research in computer vision. This annual competition uses a large database of labeled images to assess algorithms for image classification and object recognition. In 2012, `AlexNet`[10] won the ILSVRC 2012 competition, marking the true beginning of the deep learning era. Its design introduced ReLU (*Rectified Linear Unit*) non-linear activation, dropout techniques, and the use of GPU in training, which greatly improved the speed and effectiveness of training deep networks. However, the deep learning community still faced issues such as vanishing and exploding gradients in training deep networks. As the network depth increased, earlier network structures like AlexNet showed performance degradation, meaning that greater network depth does not always lead to better performance.

In 2015, `ResNet`, proposed by Kaiming He and his team at Microsoft Research[3], achieved breakthrough results in that year's ImageNet competition. ResNet introduced the concept of "residual learning" to address these issues. In a ResNet, the input is not only passed through hidden layers but also establishes "identity connections" that skip one or more layers. This design allows the network to learn the residuals between the input and output, enabling the network to model nested function classes. This helps alleviate the problem of vanishing gradients in deep networks, allowing for successful training of networks with hundreds of layers without significant performance degradation. Although in recent years, Transformer-based models like `ViT` (Vision Transformer)[1] have gained attention for their potential ability to achieve superior performance in certain image tasks, residual connections remain a crucial part of the Transformer architecture. The simplicity of ResNets structure, which is easy to understand and implement, ensures that it remains widely used in many practical applications and research areas.

Although models in deep learning can achieve high performance, their decision-making process is often opaque and hard to explain, known as the "black-box" issue. People may lack trust in the decisions made by these black-box models, and if there are issues during the training and optimization process, it can be difficult for developers to pinpoint the problems. As a result, `explainable AI`[11] has become a topic of interest in both industry and academia. In the field of image analysis, `saliency maps`[13] are one powerful tool to address this issue. Saliency maps were originally developed to mimic the attention mechanisms in the human visual system, helping to identify the most eye-catching areas in images. This concept was later introduced to computer vision, especially with the development of deep learning technology, making it even more important. Through saliency maps, researchers and developers can identify the areas that the model focuses on when processing specific images, which helps them better understand the model's behavior. If the model makes an incorrect decision, saliency maps can also help identify the image areas that led to the error, providing direct clues for improving the model. Thereby enhancing the model's explainability.

This thesis, serving as an undergraduate degree project, was inspired in the context of advancements in ResNet and model's interpretability mentioned above. On one hand, this thesis aims

to build and train the classic ResNet-18 [3] model from scratch for a classification task on the CIFAR-10 dataset[4], demonstrating the model's excellent performance in image classification. On the other hand, the thesis continues focusing on the interpretability of the model by using saliency maps to explain the model's decision-making process. Notably, the saliency maps in this study are based on the D-RISE (*Detector Randomized Input Sampling for Explanation*) algorithm, introduced by `Petsiuk et al.`[8] in 2021, originally applied to the YoLo (*You Only Look Once*) detection model[9]. This thesis adapts and reproduces the D-RISE algorithm, enabling its application on the trained ResNet-18 model instead.

In this thesis, *Chapter 2, 3* and *4* introduce the fundamentals of deep learning, the architecture of convolutional and residual networks, and the basic principles of saliency maps and the D-RISE algorithm. *Chapter 5* details the dataset preprocessing and the initial setup of the ResNet-18 model and D-RISE algorithm parameters. *Chapter 6* summarizes the experimental results, including training curves, accuracy rates, and examples of saliency maps. Finally, *Chapter 7* discusses the issues about mathematical interpretability behind the D-RISE algorithm and explores solutions to address relevant issues. *Chapter 8* serves as the appendix, containing figures relevant to this thesis.

Additionally, the experimental section of this thesis is implemented in Python, primarily using the `TensorFlow` framework for deep learning[15]. `CUDA`[6] and `cuDNN`[7] are utilized to enable GPU computation, which significantly accelerates the training processes. You can find the relevant codes and programming environment information at my GitHub repository[14].

2 Theoretical Framework of Deep Learning

In this chapter, we will introduce the background knowledge related to deep learning frameworks and theories. The content is based on Chapter 4 of the book by Goodfellow et al. [2].

2.1 Foundations of Neural Networks

Neural networks, as the name implies, are computational systems inspired by the biological neural networks of human brains, composed of interconnected units, or neurons. The architecture of neural networks is generally deep and complex, thus enabling the network to perform complex computations and learn from data with complex patterns. The ResNet-18 model that this thesis focuses on are a variant of the neural network architectures. Therefore, before diving into the main topic, this section will first introduce several important concepts of artificial neural networks.

2.1.1 Neuron and Neural Network Architecture

An activation function is a mathematical function that receives linearly transformed input and, through a nonlinear computation, produces output as the input to the next layer. The main purpose of an activation function is to introduce nonlinearity into the model. In a general way, a neuron receives inputs, each multiplied by a weight, and the total sum of these weighted inputs is then modified by a bias term:

$$z = WX + b. \tag{2.1}$$

Here, X represents the input vector and W represents the weight matrix, The output of the neuron, y, is then determined by applying an *activation function* f to z:

$$y = f(z). \tag{2.2}$$

Neural networks are structured in layers: the *input layer*, which receives the input data; one or more *hidden layers*, which process the inputs; and the *output layer*, which produces the final output. The *depth* of the network refers to the number of layers, and the *width* refers to the number of neurons in a single layer that can vary from layer to layer.

So, we can represent the input and output of a specific layer in a neural network as follows:

$$h^i = f^i(z^i) = f^i(W^i h^{i-1} + b^i), \tag{2.3}$$

where h^i denotes the output of the i-th hidden layer, f^i is the activation function applied in the i-th layer, and z^i represents the weighted sum of outputs from the previous layer (h^{i-1}), which

weighted by weight matrix of the i-th hidden layer W^i and shifted by bias vector of i-th hidden layer b^i. Each layer's output becomes the next layer's input, propagated from the input layer to the output layer in a process called *forward propagation*.

2.1.2 Activation Functions and ReLU

Activation functions determine the output of a neuron given a set of inputs and are essential for introducing non-linearity into the network, enabling it to learn complex patterns. Common activation functions with input z, including but not limited to, are:

$$\text{Sigmoid} : \sigma(z) = \frac{1}{1 + e^{-z}}, \tag{2.4}$$

$$\text{ReLU} : f(z) = \max(0, z), \tag{2.5}$$

$$\text{Tanh} : \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}. \tag{2.6}$$

ReLU is advantageous because it is easy to implement, and it also avoids the vanishing gradient problem because it allows positive values to pass through unchanged while only blocking negative values. In contrast, other activation functions, such as sigmoid, compress input to a range of [0, 1], which can lead to gradients gradually diminishing or even disappearing during backpropagation (*chapter 2.2.1*). Since ReLU maintains a constant gradient of 1 for positive values, it ensures that gradients do not shrink rapidly during backpropagation, enabling effective weight updates. For more information about the vanishing gradient issue refers to *Chapter 3.3*.

Although ReLU has the advantages mentioned above, its gradient being zero for negative inputs can lead to the "dead neuron" problem. To address this, we can use *Leaky ReLU*, which assigns a small slope to negative inputs. However, since this is outside the scope of this thesis, we won't discuss it further here.

2.1.3 Softmax Function in Multi-Class Classification

When handling problems that involve classifying into many categories, the softmax function is a standard approach for setting up the model's output layer. This is because the softmax function outputs a probability distribution over all possible classes, assigning a probability value to each category. The class with the highest probability is then selected as the predicted output, which makes softmax a common solution for multiclass classification problems. In this thesis, we have used the softmax in the output layer in the ResNet-18 model for the ten-category classification task based on CIFAR-10.

Given a set of classes, the softmax function takes the log of unnormalized probabilities from the fully connected layer, where z, W, b, X are the output vector, the weight matrix, the bias vector and the input vector of the fully connected layer: $z = WX + b$.

We denote that $z_i = \log \tilde{P}(y = i \mid x)$, where z_i is the i-th element in the vector z, representing the unnormalized log probability of category i given the input x, and $\tilde{P}$ represents the unnormalized probability. The softmax function transforms these unnormalized probabilities into a valid probability distribution. The probability for each class is computed as the exponentiated

log probability of the corresponding class divided by the sum of exponentiated log probabilities for all classes, that is:

$$P(y = i|x) = \text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}, \tag{2.7}$$

where K represents the number of classes. *Eq 2.7* ensures that each output element of the softmax layer is between 0 and 1, and the sum of the entire vector equals 1.

2.1.4 Loss Function and Cost Function

Through forward propagation, inputs pass through various hidden layers ending to the output layer which gives the outputs of the model that, when compared with the true labels, generate a loss. In this subsection, we will dive into the concept of loss function and cost function.

In the context of machine learning and neural networks, a loss function and a cost function both quantify the difference between the predicted outputs and the true labels. They are often used interchangeably in many context, but they have subtle differences. A *loss function* measures the error between the predicted outputs and the true labels for a single data point. In contrast, a *cost function* is the average or total loss over all data points in a dataset, assessing the performance of the model across the entire dataset. A general formula for a loss function is typically expressed as:

$$L(y, \hat{y}) = f(y, \hat{y}) \tag{2.8}$$

Here, y represents the true label, $\hat{y}$ represents the predictions, and f is a function used to calculate the error between these two. For regression tasks, the function f is typically the Mean Square Error (MSE), and for classification tasks, it is cross-entropy, which will be discussed shortly. The cost function can be expressed as:

$$J = \frac{1}{M} \sum_{i=1}^{M} L(y^{(i)}, \hat{y}^{(i)}) \tag{2.9}$$

In this formula, M is the total number of data points, $y^{(i)}$ and $\hat{y}^{(i)}$ are the true label and the model's prediction for the i-th data point, respectively. This formula shows that the cost function is an average of the individual loss values, evaluating the model's performance across the whole dataset.

Given the widespread use of 'loss' in machine learning literature and practice, and its intuitive representation of prediction errors, this thesis will uniformly use the term 'loss' to describe the error functions for both mini-batches (*Chapter 2.2.2*) and the overall dataset.

2.1.5 Cross-Entropy

The softmax function is commonly used for classification problems to transform model outputs into probabilities. Cross-entropy loss is a widely used loss function for classification tasks. Cross-entropy is derived from information theory and is closely related to Kullback-Leibler (KL) divergence, which measures how one probability distribution diverges from a second, expected distribution. The softmax function is suitable for multi-class classification as it converts model outputs into probabilities that sum to one, which aligns well with the one-hot encoded true labels. By using cross-entropy loss, we aim at minimizing the KL divergence between the predicted probability distribution and the true label distribution, justifying its use as an appropriate loss function in classification tasks.

One-hot encoding is a method used to convert categorical data into a binary representation. Each category is represented by a vector that has one element set to 1 and all other elements set to 0. The position of the 1 indicates the class.

Mathematically, the cross-entropy loss for a single data point can be expressed as:

$$L(y, \hat{y}) = -\sum_{i=1}^{K} y_i \log(\hat{y}_i) = -\sum_{i=1}^{K} y_i \log(P(y = i \mid x)) \tag{2.10}$$

Here, y represents the true distribution of the labels for the current data point (one-hot encoded), $\hat{y}$ represents the predicted probabilities resulting from the softmax function and K is the number of classes. The summation runs over all the classes, and the loss becomes the negative log-probability of the true class's predicted probability. By minimizing this loss across all data points in the training set, the model is trained to increase the probability of the correct class, which means the accuracy for the classification task is improved.

2.2 Model Training and Optimization

In machine learning, training a model is an iterative process. The goal of model training is to optimize parameters to minimize the loss function. The training process is also about finding the balance point between bias and variance of the loss to avoid overfitting and underfitting. In the following subsections, we will introduce the main theories related to model training and parameter optimization.

2.2.1 Back-Propagation

Back-Propagation is a fundamental approach for training neural networks. It efficiently computes gradients of a loss function with respect to the parameters of the network. Back-propagation uses the chain rule in reverse, starting from the network's output layer and calculating the gradient of the loss function layer by layer toward the input layer. This process accurately determines the direction in which the loss function decreases the fastest, known as the negative gradient direction. At the end of a back-propagation evaluation, the gradient descent algorithm, for example, SGD (*Stochastic Gradient Descent*, *Chapter 2.2.2*) updates the network parameters according to the gradient direction and the learning rate. The learning rate sets the size of the step in each update, serving as a crucial hyperparameter that controls the amount of parameter adjustments. The network gradually updates the parameters with each iteration using this procedure until

it reaches the endpoint of training epochs or the learning curve has become overfitting(*Chapter 2.2.4*).

The chain rule in the context of back-propagation is exemplified by the following equations, based on a 3 layer model, by calculating the gradient of the network's output z with respect to its parameters w, where f is the activation function applied at each layer, the derivative chain rule can be expressed as:

$$\frac{\partial z}{\partial w} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w} \tag{2.11}$$

$$= f'(y)f'(x)f'(w), \tag{2.12}$$

Here, $f'(y)$, $f'(x)$, and $f'(w)$ represent the derivatives of the activation function at the first, second, and third layers, respectively. The product of these derivatives gives the final gradient in this simple example.

2.2.2 SGD Algorithm

In deep learning, *Stochastic Gradient Descent (SGD)* is a common optimization method. The main idea is to randomly pick a small subset of data, often called a minibatch, to estimate the gradient of the loss function with respect to the parameters. This set of data is usually drawn from the dataset in a way that each sample is independent and has the same chance of being picked as any other. The SGD method updates the parameters over many iterations, moving them in the gradient descent direction, aiming to minimize the loss function. The update rule for SGD is defined as follows:

$$\theta_{t+1} = \theta_t - \epsilon_t \nabla_\theta J(\theta_t; X^{(i:i+m)}, y^{(i:i+m)}). \tag{2.13}$$

Here, θ_t represents the parameters of the model at the iterative step t, J is the loss function, $X^{(i:i+m)}$ and $y^{(i:i+m)}$ are the features and labels of the minibatch from i-th to $i+m$-th datapoint, ϵ_t is the learning rate at step t and $\nabla_\theta J$ represents the stochastic gradient of loss function with respect to parameters based on the data in minibatch.

When minibatch sizes are large, the efficiency of training tends to be higher and the gradient estimation more stable. However, these benefits come at the cost of increased computation cost, and there is a risk that the optimization could converge to local minima, potentially harming the model's performance. With small minibatch sizes, the model's parameters are updated more frequently. The minibatches have different patterns since they are drawn randomly, which leads to that the parameters will be updated from different initials in different gradient descent directions. This encourages a better exploration of the parameter space, which can help the model avoid local minima. But in this case of small minibatch sizes, the variance in gradient estimation for each minibatch is relatively high, which may make the training process less stable, and it may needs more iterations to achieve convergence.

Compared to traditional batch gradient descent, the main advantage of SGD is its efficiency for large datasets, as it does not needs to go through the entire dataset to compute the gradient. Moreover, the inherent noise from sampling minibatches randomly can help the model escape local minima, potentially enhancing exploration in the parameter space.

Another key characteristic of SGD is the use of an adaptive learning rate, rather than a fixed one. Using adaptive learning rates allows dynamic adjustment of the learning rate, speeding up model convergence to optimal parameters. In particular, at the start of training, the model parameters are usually far from optimal. Here, a larger learning rate can help the model quickly descend to approach the potential optimal area. As the model parameters begin to converge toward the optimal solution, a large learning rate may cause oscillations near the minimum, making it hard to finely adjust to the optima. The adaptive learning rate will gradually decrease, allowing the model to adjust with finer steps, thus more precisely approaching the optimal solution and reducing the risk of *overshooting*, which refers to a situation where the step size of an update is too large, causing the model's parameters to bypass the optimal point of the loss function.

Furthermore, this thesis utilizes the cosine decay learning rate strategy, a method of dynamically adjusting the learning rate during the model training. Unlike the traditional linear decay of learning rates, the cosine decay strategy adjusts the learning rate periodically in accordance with the shape of the cosine function. The formula for the cosine decay learning rate is as follows:

$$\epsilon(t) = \epsilon_{min} + 0.5 \times (\epsilon_{max} - \epsilon_{min}) \times \left(1 + \cos\left(\frac{t\pi}{T}\right)\right) \tag{2.14}$$

Here, $\epsilon(t)$ is the learning rate at step t, ϵ_{max} and ϵ_{min} are the maximum and minimum values of the learning rate, respectively, and T is the decay period, meaning that after T steps, the learning rate completes a cycle from ϵ_{max} to ϵ_{min}. In this cosine decay period, a larger learning rate aims in rapid convergence and exploration, while the smaller learning rate prevents the overshooting problem when the loss is close to its optima. In the other cycle when t increases until $2T$, the learning rate increases again from ϵ_{min} to ϵ_{max}, which may help the model escape local minima, preventing premature convergence to suboptimal solutions.

2.2.3 Momentum Algorithm

Additionally, in this thesis, we employ the Stochastic Gradient Descent (SGD) algorithm with momentum. The *momentum* algorithm is an optimization method used to accelerate the gradient descent algorithm. It introduces a "velocity" variable $\mathbf{v}$ in the parameter space, which tracks the cumulative movement of gradients and guides the parameter updates. The update rules for the momentum algorithm can be expressed as:

$$\mathbf{v} \leftarrow \alpha\mathbf{v} - \epsilon\nabla_\theta \left(\frac{1}{M}\sum_{i=1}^{M} L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})\right), \tag{2.15}$$

$$\theta \leftarrow \theta + \mathbf{v}. \tag{2.16}$$

Here, α is the momentum hyperparameter, and ϵ is the learning rate. ∇_θ represents the gradient of the cost function with respect to the parameters θ. $\mathbf{x}^{(i)}$ and $\mathbf{y}^{(i)}$ represent the i-th sample and its label respectively, and M represents the number of datapoints.

SGD with momentum not only considers the current gradient direction but also accumulates effects from previous gradients. When the Hessian matrix, a second-order derivative matrix describing the loss function's curvature, has a poor condition number, the gradient of the loss can

be very large in some directions, while in other directions, it can be very small. We can image a extreme case where the loss function's shape can be steeper in directions that do not lead to the optimal solution and relatively flatter towards the optimal solution. In such scenarios, standard gradient descent struggles to converge to the optimal solution. Momentum modifies parameter updates by incorporating a portion of the previous step's update, namely the accumulated momentum effect. This assists in maintaining a higher parameter update step size across flat regions while reducing the oscillation of parameters' update in steep areas. Consequently, this method accelerates the learning process, enhancing both the speed and stability of convergence.

2.2.4 Trading off Bias and Variance

In machine learning, the bias-variance tradeoff is a core concept that describes the relationship between model complexity and the model's ability to generalize on unseen data. In machine learning models, we typically use the output loss (such as Mean Squared Error, MSE) to describe this tradeoff. Bias represents the differences between the true labels and the model's predictions, while variance indicates the variation in the model's predictions across different training datasets. Specifically, simpler models tend to have higher bias and lower variance, while more complex models exhibit the opposite, having lower bias and higher variance.

We can also use the Mean Squared Error (MSE) of parameters to demonstrate this tradeoff, which is a perspective from the theory of inference. The MSE of parameters is the average squared difference between the estimated parameters $\hat{\theta}$ and their true values θ, which can be decomposed into the square of bias and variance, which respectively represent the model's systematic error and the model's sensitivity to training data:

$$\begin{aligned}
\text{MSE} &= \mathbb{E}[(\hat{\theta} - \theta)^2] && (2.17)\\
&= \mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}] + \mathbb{E}[\hat{\theta}] - \theta)^2] && (2.18)\\
&= \mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2] + \mathbb{E}[(\mathbb{E}[\hat{\theta}] - \theta)^2] + 2\mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}])(\mathbb{E}[\hat{\theta}] - \theta)] && (2.19)\\
&= \text{Var}(\hat{\theta}) + \text{Bias}(\hat{\theta})^2, && (2.20)
\end{aligned}$$

where $\mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}])(\mathbb{E}[\hat{\theta}] - \theta)] = 0$.

The model's systematic error refers to the square of the difference between the expected prediction of model parameters and the true parameter values. A high value of square of bias indicates that the model may be too simple (**underfitting**) and unable to capture all relevant features of the data. The model's sensitivity to training data refers to the variability of model predictions of parameters across different training datasets. A model with high variance shows large differences in predictions across different data samples, indicating that the model is highly sensitive to random fluctuations in the training data, which is typically due to the model being overly complex (**overfitting**).

There is an optimal point in model training where both bias and variance are relatively low, indicating a balance between simplicity and complexity. This balance is crucial in machine learning, and we often use learning curves to monitor changes in model performance, identifying a U-turn point between underfitting and overfitting. In *Figure 8.1* in the *Chapter 8 Appendix*, we illustrate this with a learning curve that shows the relationship between model loss and the number of training epochs. Here, the loss on both the training and validation sets for each epoch is observed, allowing us to apply the bias and variance tradeoff theory to determine the optimal number of epochs.

2.2.5 Data Augmentation

Data augmentation is a key strategy for improving the generalization ability of neural network models and preventing overfitting. The generalization ability of a model refers to its capacity to make accurate predictions on new, unseen data. By applying various transformations to the training dataset, we artificially expand it, enabling the model to learn from a broader range of examples without requiring new data. Importantly, the labels of the transformed images remain unchanged. This technique is especially useful when dealing with limited datasets. The main techniques in data augmentation include:

1. **Rotation**: This technique involves rotating the image by a certain degree to create a new training example. This helps the model handle objects at different orientations.
2. **Flipping**: Images can be flipped horizontally or vertically. This is especially useful for training on objects where orientation can vary, such as in natural scenes.
3. **Scaling**: This technique involves resizing images either by zooming in or zooming out. It helps the model learn to recognize objects at different scales.
4. **Color augmentation**: Adjusting the color settings of images, like changing the brightness, contrast, saturation, or hue, can help the model become less sensitive to color variations.
5. **Cropping**: Randomly cropping sections of the image can help the model focus on different parts of the object, which is useful for learning to recognize objects no matter which part of the object is shown in the image.
6. **Noise injection**: Adding random noise to images can help models become more robust to variations and imperfections in input data.

While data augmentation techniques can effectively improve the generalization ability of models, they also have some drawbacks. First, by introducing variations to increase the diversity of the training data, unnecessary noise might also be introduced. This noise can make it difficult for the model to recognize the true features of the data, thereby reducing its performance. Additionally, data augmentation increases the size of the training dataset, which means the model needs to process more data during training. This indicates that more computational resources and time required to train the model, especially when complex augmentation techniques are used, making the impact even more significant.

2.2.6 Batch Normalization

Batch Normalization is a widely used technique in deep learning to reduce internal covariate shift and improve model stability during training. Internal Covariate Shift is an issue that can arise during the training of deep learning models, referring to the changes in the distribution of input data across the layers of the network during training. This phenomenon is primarily caused by the updates of parameters in earlier layers, which require subsequent layers to continuously adapt to these changes.

Because the distribution of input data changes across different layers, a significant issue can arise with activation functions such as ReLU. In the worst case, if a shift moves the input distribution primarily into negative values, these inputs will be completely nullified by ReLU, as it outputs zero for any negative input. This negatively affects the training process. It can slow down training, make convergence more difficult, and potentially degrade the overall performance of the model. By using batch normalization, we can maintain a relatively stable input distribution

for each layer, thereby stabilizing the learning process across layers. It normalizes the inputs to the batch normalization layer, which performs batch normalization, ensuring they have a mean of 0 and a variance of 1. The mathematical formulas are as follows:

Given a batch of data $\{x_1, ..., x_M\}$, its mean μ_B and variance σ_B^2 are computed as:

$$\mu_B = \frac{1}{M} \sum_{i=1}^{M} x_i, \tag{2.21}$$

$$\sigma_B^2 = \frac{1}{M} \sum_{i=1}^{M} (x_i - \mu_B)^2. \tag{2.22}$$

Then, each input x_i is normalized to:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \quad i = 1, 2, , ..., M, \tag{2.23}$$

where ϵ is a small number to prevent division by zero. Finally, the normalized $\hat{x}_i$ is scaled and shifted by learnable parameters γ and β to produce the final output y_i:

$$y_i = \gamma \hat{x}_i + \beta. \tag{2.24}$$

The reason for adding this step after batch normalization is that although batch normalization helps address the issue of internal covariate shift, it may also lead to the loss of some useful distribution properties that the original layers learned during training, referring to over-standardizing. To restore these potentially useful features, we perform scaling and shifting of the data after batch normalization.

3 Convolutional Networks and Residual Networks

Based on Chapter 9 of the work by `Goodfellow et al.`[2], Convolutional Neural Networks (CNNs) are specialized types of neural networks that are particularly effective in analyzing visual data, primarily through the use of convolutional layers that can identify patterns and features in images. A key difference between CNNs and fully connected neural networks, such as the Multilayer Perceptron (MLP) which consists of multiple layers of neurons with non-linear activation functions, is that CNNs incorporate a mathematical process called convolution operation in at least one of their layers. These layers where convolution operation is employed are known as convolutional layers. In the following subsection, we will discuss in more detail about the principle of convolution operation, the concept of the convolution kernel, and explain why CNNs are particularly suited for image classification tasks, compared with fully connected neural networks. Actually, CNNs is not only limited to process image data, they are also applicable to other types of data, such as time series data. While this thesis focuses on image classification tasks, readers interested in other applications can refer to the book by `Goodfellow et al.`[2].

3.1 The Convolution Operation

A convolution kernel, also known as a convolution filter, is essentially a matrix, and it is a critical component in convolution operation. In the context of two-dimensional image classification task, the convolution operation can be defined as:

$$S(i,j) = (I * K)(i,j) = \sum_{m=0}^{T-1} \sum_{n=0}^{T-1} I(i+m, j+n) K(m,n), \tag{3.1}$$

where $S(i,j)$ is the pixel value of the image after convolution operation at position (i,j), I is the input image before convolution operation, K is the convolution kernel, (m,n) represents the indices of the convolution kernel, and the size of kernel is $T \times T$.

As we see, the convolution kernel K performs element-wise multiplication with each local area of the input image I and sums all the products together. In this way, convolution operations can be seen as an approach for feature extraction. This is because different convolution kernels, which contain different element values, referred to as different weights, respond differently to specific features. So actually, various convolution kernels with different weights can capture different features or patterns in the image, such as edges, lines, corners, or textures. Additionally, if we want to extract detailed features in the image, we can begin with a relative small-sized kernel, compared with the size of the input image, it is because the small-sized kernel will calculate the convolution based on a small area in the image, which captured the local and detailed features.

During the convolution operation, the kernel does not only stay at the initial position, but it slides across the entire input image, performing convolution calculations with the same weights

of the kernel at each new area of the image, thus generating a new image called a feature map, which is a collection of features extracted from input data through convolution operations in a neural network, particularly in CNNs, reflecting spatial patterns after convolution operations of specific filters. This approach that different parts of an image undergo convolution calculation with same kernel allows the convolutional layer to detect or capture the similar features at different positions regardless of their specific locations in the image. Based on the description above, we can introduce two important concepts in convolution operation:

1. **Local receptive fields**: In convolutional layers, each neuron connects only to a small area of the input data, known as the local receptive field. For example, when processing an image, a kernel might only be applied on a small 3×3 or 5×5 area of the input image, i.e. T in *Eq 3.1* equals to 3 or 5. This means that the elements in feature maps come from a small area with the same size, rather than the entire image.
2. **Weight sharing**: In convolutional layers, the same weights of a convolution kernel are applied to different areas of the input data to detect similar features at different locations, which achieves weight sharing, instead of assign a weight to each pixel. This can significantly reduce the number of parameters and complexity of the model, as well as the computational cost.

Furthermore, stride is also an essential setting for performing convolution operations in conjunction with a convolution kernel. The stride is the number of pixels the convolution kernel skips across the input image each time. For example, a stride of 1 means the kernel moves one pixel at a time, while a stride of 2 means it moves two pixels. The kernel slides across the image according to the set stride until the entire image is covered and processed. The size of the stride directly affects the number of convolution calculations performed. A larger stride can speed up the convolution operation by reducing the number of times the kernel needs to slide across the image, which in turn decreases the number of convolution calculations and the size of the generated feature map.

This process of reducing the spatial dimensions of the feature maps is known as *downsampling*. Downsampling is the process of reducing the sampling rate or the resolution of a signal, image, or dataset. In the context of CNNs, downsampling typically refers to reducing the spatial dimensions of the feature maps, often achieved using techniques such as pooling (*Chapter 3.2*) or using convolutional layers with strides greater than one. The purpose of downsampling is to decrease the computational load, reduce the number of parameters, and achieve translation invariance while preserving the most important features of the input data.

In convolutional models, small kernels and strides can be used in the initial convolutional layers, as this captures more image details, as mentioned above. As the model depth increases, larger size of kernels or larger strides can be used to extract more abstract features. This shows actually a process about how the receptive field expands within a convolutional network. Let us illustrate with an example. Assume the first convolutional layer uses a 3×3 convolution kernel with a stride of 1. Each kernel covers a 3×3 area of the input image and moves one pixel at a time. Therefore, each element of the feature map has a 3×3 local receptive field in the input image. Assume the second layer uses a 5×5 convolution kernel, covering a 5×5 area of the feature map output from the first layer, the receptive field of the second layer is then expanded to 7×7. This means that as the depth of the CNN increases, the receptive field also increases, which implies that the feature maps in the deeper layers contain more abstract feature information. Through this process, the CNN achieves a gradual extraction of features from details to abstraction of the image.

In summary, convolution operations effectively capture similar features within each local receptive field by sliding convolution kernels over the image. It also reduces the number of model parameters and its complexity through a weight-sharing mechanism, thus helps prevent overfitting. Additionally, by adjusting the stride and increasing the number of convolutional layers, we can use downsampling to match the decrease in dimensionality of the representation for layers closer to the output. The advantages above make convolution operation well-suited for processing image data.

3.2 Pooling

Besides that, the pooling layer is also a crucial component of CNNs and is typically positioned after one or more convolutional layers. For instance, in the model discussed in this thesis, after several convolutional layers have captured the abstract features of the input data, a pooling layer is used, aiming at picking up summary statistics from the feature maps, and it also can be used for preparing the data for entry into a fully connected output layer to yield classification predictions. In the CNNs, *Summary statistics* refer to concise representations of the information contained in the feature maps. These statistics capture the most important information, summarized through pooling techniques. The common methods used in pooling layer includes Max Pooling, Average Pooling and Stochastic Pooling.

In the model presented in this thesis, we utilize average pooling, which computes the average of all the values from a feature map within a defined pooling window and assigns it as the single value for that window in the pooled feature map. Mathematically, if we have a pooling window of size f, the average pooling operation can be expressed as:

$$P(i,j) = \frac{1}{f^2} \sum_{m=0}^{f-1} \sum_{n=0}^{f-1} I(i+m, j+n). \tag{3.2}$$

Here, $P(i,j)$ represents the value after pooling at location (i,j), $I(i,j)$ represents the original value at location (i,j), m,n are indices of the pooling window, and $f \times f$ is the size of the pooling window. Therefore, we actually use an average value to represent the element values of feature maps from the previous convolutional layer before the average pooling layer, meaning that average pooling represents the general level of activity within the pooling window. In this way, the average pooling layer aids the model in abstracting and compressing the features of the image, which helps to reduce the dimensionality of the data and retain essential features while discarding less important details.

Specifically, the ResNet-18 model in the thesis uses a 4×4 window for average pooling. That means the average pooling layer will calculate the average of the values within each 4×4 block and outputs a single value. In this way, the feature map from the last convolutional layer is abstracted by average pooling layer from size of 4×4 into 1×1, The output of the pooling layer is prepared for the output layer which is a fully connected layer followed by a softmax activation.

3.3 Residual Networks (ResNet)

In the field of deep learning, Residual Networks (ResNets) introduced an innovative neural network structure to address the primary challenge faced by traditional convolutional networks

(CNNs) when training very deep networks: *vanishing gradients*. This issue makes the network difficult to train, especially as the depth of the network increases, leading to a phenomenon known as the degradation problem, where performance degrades with increasing network depth. The vanishing gradients issue occurs when the gradient becomes very small during backpropagation, almost approaching zero. When the gradient is too small, it hardly adjusts the weights in the network, preventing effective learning and causing extremely slow parameter updates, particularly in the parts of the network close to the input layer.

ResNet effectively addresses the issue of vanishing gradients, thereby helping to avoid the problem of network degradation in deep networks. This improvement over traditional CNN architecture is achieved by introducing residual blocks and skip connections, which will be discussed in detail in the next subsection. However, another issue related to the degradation problem is exploding gradients, which is a problem that recurrent neural networks (RNNs) can face. This issue is not the main focus of this thesis. Readers interested in learning more about exploding gradients can refer to the work by `Zhang et al.`[16], on which the content of this chapter is based.

3.3.1 Residual Blocks and Skip Connection

Residual blocks are the basic units that make up a ResNet. A residual block in this thesis has two convolutional layers and each convolutional layer is usually followed by batch normalization (*Chapter 2.2.6*) and a non-linear activation function, like ReLU (*Chapter 2.1.2*).

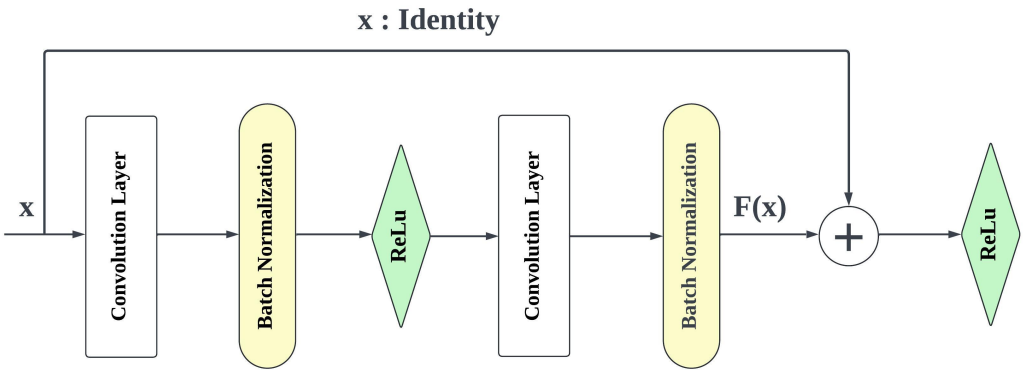


Figure 3.1: **A Residual block with skip connection in ResNet-18**: A residual block in ResNet-18 model includes two convolutional layers followed by batch normalization and ReLU activation function. The input of residual block x is connected directly to the output of the second batch normalization $F(x)$. This direct path is called skip connection. ReLU is applied on the sum $x + F(x)$ in the last step. This design allows the model to learn nonlinear features from the input through both the direct path and the layers within the residual block.

In ResNet, each residual block internally includes a skip connection that allows the input to bypass the convolutional layers within the block, connecting directly to the output end of the block. This skip connection performs an *identical mapping*, which means a function that maps every element to itself. This structure ensures that the output of the residual block is a direct

sum of the input and the output of the block. Therefore, the output of a residual block can be seen as a superset of its input because an unchanged part is always transmitted to the output. Additionally, each added residual block can be seen as a fine-tuning of the existing model structure. The change induced by the residual block is called the residual. By adding this residual to the input of the direct path in a residual block, we construct *nested functions*, where the function here refers to the models function expression. Stacking several residual blocks constructs a nested space of model functions. Because the skip connections adopt identical mapping, this ensures that there is always a part of the nested structure that retains the shallow layer's function expressions. Even if some intermediate layers in the residual blocks do not learn useful information or if the gradients passing through some block layers are very small, they will not significantly impact the overall performance of the network. This allows deeper networks to gradually learn complex representations by stacking residual blocks, thereby approximating the optimal function expression of the model without encountering training issues such as vanishing gradients. Thus, this nested structure ensures that increasing the model's complexity does not degrade its performance in a deep network.

In this thesis, the ResNet-18 network, which will be discussed in detail in the next subsection, consists of 8 residual blocks, each containing 2 convolutional layers, as illustrated in *Figure 3.1*. After each convolutional layer, batch normalization follows immediately. The purpose of this is to standardize the output of each convolutional layer and to improve the stability of model training. After the first batch normalization, a ReLU activation function is used to introduce non-linearity. And then the output of ReLU layer will be processed by the second convolutional layer, as we have mentiond in *Chapter 3.1*, the receptive field is expanded here and more abstract features are captured. The second convolutional layer is also followed by a batch normalization, after which we introduce a skip connection directly from the block's input. Then, the sum is processed again through a ReLU activation function. The reason for this is that the ReLU activation function provides nonlinearity, which is essential for deep learning models to learn the nonliear features. By adding the input from the skip connection before applying the ReLU, it ensures that the nonlinearity of the input features through the direct path is preserved, even after propagation through multiple residual blocks.

3.3.2 ResNet-18

ResNet-18, as an 18-layer model in the residual networks, is a fundamental architecture in the field of deep learning for image classification tasks. The structure of ResNet-18, as shown in *Figure 3.2*, mainly includes:

Initial Convolutional Layer: In the standard ResNet-18 model, a 7×7 convolution kernel with a stride of 2 is used. However, for our adaptation to the CIFAR-10 dataset, which consists of small-scale images of size 32×32, we utilize a 3×3 convolution kernel with a stride of 1 instead. This change aims to capture more detailed features that might be lost with larger kernels and strides. Using a 7×7 kernel with a stride of 2 results in a feature map with elements each having a receptive field of 7×7 from the model input, and effectively reduces the feature map size by half. This can lead to an overly abstract representation of features in the initial stages, whereas a 3×3 kernel with a stride of 1 helps in preserving the local details crucial for small-scaled images in CIFAR-10.

Residual Blocks: The model contains 4 groups of residual blocks, constituting layers 2-17, each group containing two residual blocks, with each block, as shown in *Figure 3.1*, composed of two layers of 3×3 convolutional layers, following with batch normalization and ReLU activation

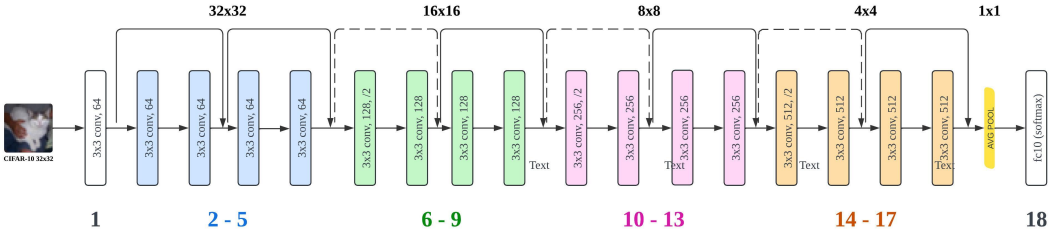


Figure 3.2: **ResNet-18 architecture:** The ResNet-18 model discussed in this thesis is trained on the CIFAR-10 dataset. It contains an initial convolutional layer followed by eight residual blocks, each containing two convolutional layers, the residual block's structure as illustrated in *Figure 3.1.* Details on kernel sizes, channel counts, stride and feature map dimensions are provided in this figure. For example, the configuration 3×3 *conv, 128, /2* indicates a kernel size of 3×3 with a stride of 2 and 128 channels; unless specified otherwise, the stride defaults to 1. The dimensions of the feature maps, such as 32×32, are indicated above the arrows in the diagram. The arrow in the diagram represents a skip connection, which also delineates the start and end points of a residual block. The solid lines indicate cases where the feature map size does not change, while the dashed lines indicate cases where the feature map size is halved. Additionally, the numbers 1-18 at the bottom of the diagram represent the layer numbers of the model. The term `AVG POOL` refers to the average pooling layer, and `fcn10(softmax)` denotes the fully connected output layer, which utilizes a softmax activation function.

layers. In these four groups of residual blocks, the number of channels in each group is 64, 128, 256, and 512, respectively, and the size of the feature maps decreases from the input layer to the output layer. Specifically:

- **First Group of Residual Blocks (Layers 2-5)**: This group contains two residual blocks, each consisting of two convolutional layers. Each convolutional layer uses a 3×3 kernel with a stride of 1 and maintains 64 channels. The feature map size remains unchanged at 32×32. The term "channels" refers to the number of the feature maps, and this number also matches the count of kernels, indicating that there are 64 kernels. In the context of processing a colored image with RGB (Red, Green, Blue) channels, each color channel can be visualized as a 32×32 matrix with pixel values ranging from 0 to 255. In the first convolutional layer, each color channel undergoes convolution operation with all 64 kernels, consequently generating 64 feature maps for each color.

- **Second Group of Residual Blocks (Layers 6-9)**: In this group, the first convolutional layer has a stride of 2, halving the size of the feature map to 16×16, with the number of channels increased to 128. The reduction of size of the feature map helps in decreasing the computational cost, and it also indicates that the receptive field of feature map has been expanded and more abstract features are captured. Additionally, increasing the number of channels, which means increasing the number of convolution kernels, allows the model to capture more complex and various features. The other convolutional layers in this group still has stride of 1, which means the size of feature map remains size of 16×16 to the last layer in this group.

- **Third Group of Residual Blocks (Layers 10-13)**: Similar to the second group, but the number of channels increases to 256, and the first convolutional layer in this group halves the size of the feature map again to 8×8 by employing the kernel with size of 3×3 and the stride of 2.

- **Fourth Group of Residual Blocks (Layers 14-17)**: In the final group, the number of channels increases to 512, and the size of the feature map reduces to 4×4 by employing the stride of 2 in the first convolutional layer.

Average pooling layer and Output layer: After four groups of residual blocks, the size of the feature maps gradually decreases, and the number of channels gradually increases, thereby achieving a more abstract and complex image features. Subsequently, the model further reduces the dimensionality of the feature map through a average pooling layer to a size of 1×1, and outputs the final classification results through a fully connected layer with a softmax activation function.

The main design principle of the ResNet-18 model centers around using residual blocks and skip connections to prevent performance degradation, such as gradient vanishing issue in deep networks. ResNet-18 also leverages the benefits of convolution operations, such as weight sharing and receptive field expansion, which reduce computational costs and model complexity, and help to prevent overfitting, while preserving abstract and complex features from images.

4 Saliency Map and D-RISE

In this chapter, we will introduce the background knowledge of saliency maps and an innovative algorithm called D-RISE used to generate saliency maps.

4.1 Saliency Map

Saliency Map is a visualization technique used to reveal features or regions of images that deep learning models focus on when performing classification tasks. By highlighting parts of the image that contribute most to the model's decision-making, saliency maps help researchers and developers understand the model's behavior. This technology plays a key role in improving model's interpretability and there by enhancing users' trust in the model decisions.

Methods for generating saliency maps can be broadly divided into two categories: gradient-based white-box algorithms and black-box algorithms. Gradient-based white-box algorithms use information about the model's internal structure to find out the impact of certain features in the feature maps on the model's output prediction by calculating gradient information. A typical algorithm, Grad-CAM (*Gradient-weighted Class Activation Mapping*) [12], generates heatmaps by analyzing gradient information in convolutional layers, showing which parts of the image are crucial for the model's decisions. On the other hand, black-box algorithms like RISE (*Randomized Input Sampling for Explanation*) and D-RISE(*Detector Randomized Input Sampling for Explanation*) do not require internal structure information of the model but infer salient regions by modifying the original input image and observing the corresponding effects and changes caused by the modification on the output preidictions.

In the following sub-section, we briefly introduce a classic white-box algorithm for saliency maps, Grad-CAM. Since it is not the main focus of this thesis, we will only cover its basic principles to provide readers with an intuitive understanding.

4.2 Grad-CAM Algorithm

Grad-CAM works based on the gradient information in the model. By calculating the gradient information of the model's output predictions with respect to the feature maps of a specific convolutional layer, it highlights the important features in the image that contribute to predict classification.

In practical implementations of the Grad-CAM algorithm, we often compute the gradient information with respect to the output feature maps of the last convolutional layer. This is because the feature maps from the last layer are best at capturing the high-level semantic information of the input image, or in other words, the most abstract features, rather than focusing on specific detailed features like edges and textures. These abstract features are considered more crucial to classification prediction. But theoretically, we can also apply the Grad-CAM algorithm to other

convolutional layers within the network. Doing so allows us to present salient visual explanations based on feature maps from different layers, since each layer captures different levels of abstract features and details. For example, the feature maps from shallower layers might focus more on low-level features such as textures and edges, while the feature maps from deeper layers focus on more abstract features which may be more related to the classification prediction. So if a feature map has a high gradient, this means that even small changes in the pixels of this feature map can cause significant changes in the model's output (such as the probability of the predicted class). In other words, high gradient values indicate that the pixels in these feature maps have a major impact on the model's decisions.

As we have discussed above, when using the Grad-CAM algorithm, we need to understand the model's structure in order to perform gradient calculations on the feature maps of a certain convolutional layer, but usually on the last convolutional layer. This is what is referred to as a white-box algorithm.

4.3 D-RISE Algorithm

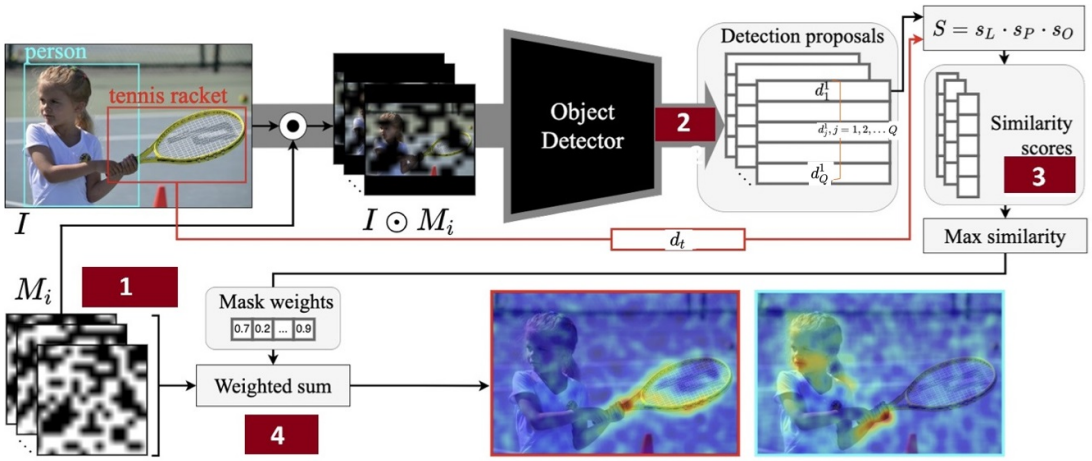


Figure 4.1: **The D-RISE algorithm mainly consists of the following four steps:** 1. Generation of binary masks, 2. Proposal generation, 3. Similarity score generation, 4. Generation of saliency maps using a weighted sum. You can find detailed information on each step in the subsequent subchapters 4.3.1 to 4.3.4. In this figure, I represents a certain original image, M_i represents the i-th binary mask, $\odot$ represents the masking operation, $d_1^1, ..., d_Q^1$ represents a series of proposals generated by the 1st mask, the number of proposals is Q, and S represents the similarity score which refers to *Eq 4.3 - 4.6*.

Based on work by `Petsiuk et al.`[8], the D-RISE algorithm is presented, which inherits the black-box advantage by generating a large number of random masks applied to the original image and then observing how these modifications affect the model's predictions without any knowledge about the model's structure. The D-RISE algorithm is originally designed for image detection tasks, applied to image detection models like Yolo (*You Only Look Once*), which is a real-time object detection system that divides an image into a grid and predicts the bounding boxes and

class probabilities for each grid cell, enabling efficient object recognition and localization. The original D-RISE algorithm is shown in *Figure 4.1*.

First, we provide an overview of how the D-RISE algorithm works based on the original work by `Petsiuk et al.`[8]. In the following subsections, we will explain it step by step in detail. The core principle of D-RISE involves generating large amounts of random binary mask matrices to mask various parts of the image. All masked images are processed through the model to obtain their classification predictions. Comparing the predictions of these masked images with the true labels of the original image generates similarity scores. These scores, used as weights combined with the binary masks, can generate a weighted sum mask matrix. Finally, this matrix is used to generate a saliency map to indicate the crucial areas of the original image for the model's prediction.

4.3.1 Mask Generation

The construction of saliency maps begins with the random mask generation, which proceeds as follows:

1. Utilizing the Bernoulli distribution, we independently generate elements in N binary mask matrices. The dimensions of each mask are set to $h \times w$, ensuring that they are less than the size of the original image $H \times W$. We assign a probability p for each pixel to be 1 and probability $1 - p$ to be 0, thus creating a sequence of binary masks. Here, p and N are hyperparameters.
2. Subsequently, we upsample these masks to a new size of $(h + 1)C_H \times (w + 1)C_W$ using bilinear interpolation. In this equation, $C_H = \frac{H}{h}$ and $C_W = \frac{W}{w}$ represent the scale factor of each cell in the original image dimensions, indicating how much each cell in the mask corresponds to the dimensions of the original image.
3. In the final step, we randomly crop sections of size $H \times W$ from the upscaled masks. The start offsets of cropping are chosen from a uniform distribution ranging between $(0, 0)$ and (C_H, C_W).

The reasons for generating binary masks through upsampling and cropping are as follows: Firstly, generating smaller mask matrices and then upsampling them can save computational resources by reducing the amount of data processed during the initial generation. Secondly, the upsampled masks, achieved through bilinear interpolation, will have a certain smoothing effect. This helps to produce masks that are more natural and continuous, avoiding excessive discreteness and abrupt changes in the image, which means the randomness in the masks retains some spatial structural features, which better simulates the locality and relationships between adjacent pixels in the image. Thirdly, by randomly cropping from the larger upsampled masks, each cropped mask can maintain randomness and diversity.

After generating the binary masks denoted as $M_i, i = 1, 2, ..., N$ in *Figure 4.1*, a masking operation is performed where an input image I is element-wise multiplied by a binary mask M_i, represented as $I \odot M_i$. This operation selectively retains parts of I where the element values in M_i is 1 and masks the rest part of I where the element values in M_i is 0, producing a masked image. In the same way, we can generate N masked images.

4.3.2 Proposal Generation

When using the YOLO model for image detection, for each masked image the D-RISE algorithm generates multiple proposals, each corresponding to a bounding box. This is because one image can contain several objects to be detected and the bounding box is a frame used to specifically mark the location and size of one certain object within an image. So the number of bounding boxes indicates the number of objects to be detected, and every bounding box corresponds to a D-RISE proposal. We denote each proposal as follows:

$$d_j^{(l)} = [L_j^{(l)}, O_j^{(l)}, P_j^{(l)}] \tag{4.1}$$

$$= [(x_1^{j,(l)}, y_1^{j,(l)}, x_2^{j,(l)}, y_2^{j,(l)}), O_j^{(l)}, (p_1^{j,(l)}, \ldots, p_K^{j,(l)})]. \tag{4.2}$$

Here, we denote that vector $d_j^{(l)}$ is the j-th proposal generated by the l-th random binary mask on a certain image, containing:

- Location information $L_j^{(l)}$, defined by the coordinates of the corners of the bounding box along its diagonal $(x_1^{j,(l)}, y_1^{j,(l)})$ and $(x_2^{j,(l)}, y_2^{j,(l)})$.
- Objectness score $O_j^{(l)} \in [0, 1]$, representing the probability that the bounding box contains any object of a certain category.
- Classification information $P_j^{(l)}$, representing a probability vector $(p_1^{j,(l)}, \ldots, p_K^{j,(l)})$ indicating the probability that the detected object belongs to each of K categories.

Through this step, we can generate several proposal vectors for each masked image, which is generated by masking an original image with a random binary mask matrix. These vectors include the location information of the bounding boxes, the probability of an object being detected, and the predicted classification probability of the detected object in the bounding boxes.

4.3.3 Similarity Metrics

To measure the similarity between a masked image and the corresponding original image, the concept of similarity score is introduced. The formula is as follows:

$$s(d_t, d_j^{(l)}) = s_L(d_t, d_j^{(l)}) \cdot s_P(d_t, d_j^{(l)}) \cdot s_O(d_t, d_j^{(l)}), \tag{4.3}$$

where

$$s_L(d_t, d_j^{(l)}) = IoU(L_t, L_j^{(l)}), \tag{4.4}$$

$$s_P(d_t, d_j^{(l)}) = \frac{P_t \cdot P_j^{(l)}}{\|P_t\| \|P_j^{(l)}\|}, \tag{4.5}$$

$$s_O(d_t, d_j^{(l)}) = O_j^{(l)}. \tag{4.6}$$

In *Eq 4.3 - 4.6*, $s(d_t, d_j^{(l)})$ is the similarity score between the target vector from the original image d_t and the j-th proposal vector $d_j^{(l)}$ from a masked image, referring to the l-th binary mask. Here, a target vector refers to a vector in the original image that includes three components: location coordinates L_t, objectness score, and classification information P_t. The location information

consists of the coordinates of a bounding box to capture a specific object marked in the original image. The objectness score is always 1, since the presence of this object at a specific location in the original image is already known and given, and the classification information is represented in a one-hot encoding format to denote the actual category of the object.

The similarity score is calculated as a product of three parts, where $s_L(d_t, d_j^{(l)})$ is the spatial proximity score using Intersection over Union (IoU) for bounding boxes of d_t and $d_j^{(l)}$, and L_t and $L_j^{(l)}$ are the coordinates information of the target and proposal bounding boxes in the original image and the masked image, respectively. $s_P(d_t, d_j^{(l)})$ is the classification similarity between d_t and $d_j^{(l)}$, defined as the normalized dot product of the probability vectors, where P_t, as mentioned, is the one hot representation of the target object's true label and $P_j^{(l)}$ is the predicted classification probability vector of the proposal $d_j^{(l)}$, and $s_O(d_t, d_j^{(l)})$ is the objectness score similarity, relevant only if the model provides an objectness score for the proposal vector, since the objectness score of the target vector is always 1, $O_j^{(l)}$ which represents the objectness score of the proposal vector $d_j^{(l)}$ will equal to the objectness similarity $s_O(d_t, d_j^{(l)})$, measuring how likely it is that a given bounding box contains an object in the masked image.

Through this step, each proposal and its corresponding target vector can generate a similarity score, which reflects the degree of similarity between the two. This score can be intuitively understood as follows: if there are two object of the same classification are detected at the same location in both the original and masked images, then the similarity score between the proposal and the target vector is 1, which is the maximum value.

4.3.4 Saliency Map Generation

Through the above steps, each original image will be randomly masked N times to generate N masked images. Each masked image will produce several proposals, which in turn generates several similarity scores, and then we choose the maximum of the similarity scores among them as the weight to the corresponding binary mask to calculate a weighted sum mask matrix:

$$H_m = \sum_{l=1}^{N} w_{l,m} M_{l,m}. \tag{4.7}$$

In this *Eq 4.7*, $w_{l,m}$ represents the max similarity score generated by the l-th binary mask of the m-th original image, while $M_{l,m}$ denotes the corresponding binary mask. H_m is a weighted sum of the N binary mask matrices corresponding to the m-th original image, is used to generate a heatmap, where larger numbers in H_m indicate that the corresponding pixels are more important, represented in red on the heatmap. Conversely, smaller numbers in the weighted sum matrix suggest that the corresponding pixels are not prominent and are shown in blue on the heatmap. Finally, overlaying this heatmap onto the original image, we obtain the saliency map based on the D-RISE algorithm.

The main idea of this step (*Eq* 4.7) is that, for each binary mask, we consider only the proposal most similar to the original image affected by its masking. This indicates that the binary mask is more effective at capturing the corresponding object and making the correct classification prediction on the proposal which has the highest similarity score. The max similarity score is used as a weight to calculate the weighted sum of the binary masks. This weighted sum reflects

the overall contribution of all binary masks in enabling the model to correctly detect objects in the masked images. This weighted sum matrix serves as the basis for generating a saliency map that visualizes the importance of features in the original images.

4.4 D-RISE Algorithm on ResNet-18

In this thesis, we have simplified the D-RISE algorithm (as illustrated in *Figure 4.2*) to adapt to the ResNet-18 model, which is not a detector model but a classification model. Since each image in the CIFAR-10 dataset contains only one object to be detected, so it is enough with one bounding box covering the entire image, sized at 32×32, which is the same size of images in CIFAR-10 dataset. From this perspective we can treat the classification task based on CIFAR-10 as a detection task with a single object within the bounding box covering the whole image. The specific steps of the adapted D-RISE algorithm to the ResNet-18 model based on the CIFAR-10 dataset are described as following:

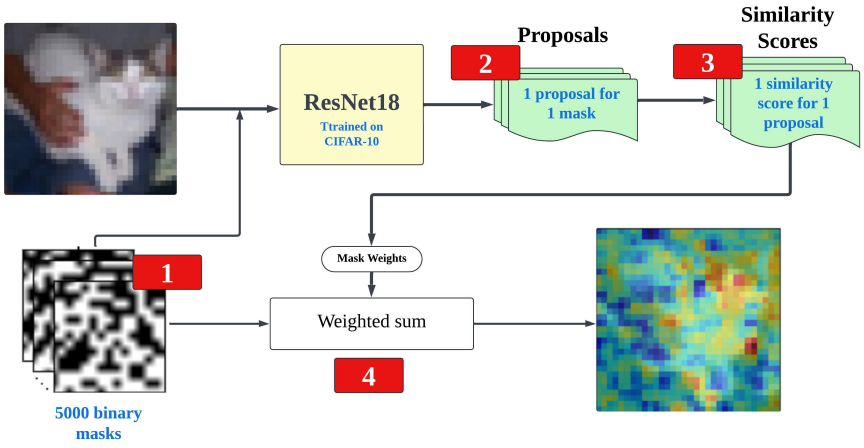


Figure 4.2: **The adapted D-RISE algorithm based on ResNet-18 and CIFAR-10:** The numbers in the red frames represent the steps in the adjusted algorithm which you can find detailed information in the following paragraphs. The main differences of the D-RISE in ResNet-18 model is that each image only has one object to be detected, meaning there is only one proposal for each masked image, therefore the only one proposal generates the only one similarity score, which will be used as the weights to calculate the weighted sum of mask matrices and the saliency map is generated based on it.

1. **Binary Mask Generation**: This step follows the procedures outlined in *sub-section 4.3.1*. Specifically, binary masks are generated randomly with a masking probability $P(0) = P(1) = 0.5$, meaning that the probabilities of masking (value 0) and revealing (value 1) are equal. During mask generation, we upsample masks from a smaller size of 17×17 to 34×34 and then crop them to 32×32 to match the size of images in the CIFAR-10 dataset. We generate 5000 binary masks in this step. The rationale behind choosing this specific masking probability and the number of masks will be discussed in detail in Chapter 5, which mainly focuses on the setup in the experiment.

2. **Proposal generating**: As we have mentioned above, each image in the CIFAR-10 dataset contains only one object to be detected, so the bounding box is set to cover the whole image and one masked image generates only one single proposal, which means the index of the proposal j in the *Eq 4.2* will always be 1. To simplify the notation, we denote the proposal vector as $d^{(l)}$ instead to represent the only one proposal generated by the l-th mask on a certain image, and *Eq 4.2* can be rewritten as:

$$d^{(l)} = [L^{(l)}, O^{(l)}, P^{(l)}] \tag{4.8}$$

$$= [(0, 0, 31, 31), 1, (p_1^{(l)}, \ldots, p_{10}^{(l)})], \tag{4.9}$$

where the superscript *(l)* denotes the l-th mask applied to a particular image, where $l = 1, 2, ..., N$, and N is the number of binary masks. The location information $L^{(l)}$ of the proposal vector $d^{(l)}$ contains the corners of the bounding box along the diagonal set to (0,0) and (31,31), which covers the whole image, respectively. The objectness score $O^{(l)}$ is set to 1, because we already know a certain image in CIFAR-10 dataset contains one object. The probability vector $(p_1^{(l)}, \ldots, p_{10}^{(l)})$ is taken from the ResNet-18 model's predicted classification probabilities of the image masked by the l-th binary mask, and the number of classes in the CIFAR-10 dataset is 10, that is why the probability vector has 10 elements.

3. **Similarity Metrics**: Since we assume that the bounding boxes completely overlap in both the masked image and the original image which cover the whole 32×32 images, which means the location information similarity will be 1, so we can rewrite *Eq 4.4*:

$$s_L(d_t, d^{(l)}) = IoU(L_t, L^{(l)}) = IoU(((0, 0), (31, 31)), ((0, 0), (31, 31))) = 1, \tag{4.10}$$

and according to the original paper by `Petsiuk et al.`[8], the objectness similarity in *Eq 4.6* is relevant here only if the model can provide a objectness score for the proposal vector in the masked image. Because the ResNet-18 model does not offer this information like a detector to present the probability of a certain object to be detected in a masked image, we can easily set the objectness similarity to 1, which means that it will not affect the calculation result of similarity score:

$$s_O(d_t, d^{(l)}) = O^{(l)} = 1. \tag{4.11}$$

So the *Eq 4.3* can be simplified as follows:

$$s(d_t, d^{(l)}) = s_P(d_t, d^{(l)}) = \frac{P_t \cdot P^{(l)}}{\|P_t\| \|P^{(l)}\|}. \tag{4.12}$$

Here, $s_P(d_t, d^{(l)})$ is the dot product of the one-hot representation of the true category label for a certain original image P_t, and the predicted classification probability of the l-th masked image $P^{(l)}$. $s(d_t, d^{(l)})$ represents the similarity score between the target vector d_t and the proposal vector $d^{(l)}$ generated by the l-th binary mask.

Furthurmore, one masked image has only one proposal in the context of ResNet18 and CIFAR-10, that is why the similarity score $s(d_t, d^{(l)})$ in *Eq 4.12* is also the maxium similarity score.

4. **Saliency map generation**: This step aligns with the methods outlined in *sub-section 4.3.4*, focusing on calculating the weighted sum of binary masks and visualizing the importance of variables by using saliency maps.

Through the aforementioned four steps, we have simplified and adjusted the D-RISE saliency map generation algorithm to suit the ResNet-18 model based on the CIFAR-10 dataset. This modification enables the D-RISE algorithm to be applicable to a model for classification tasks.

5 Setup of The Experiment

In this chapter, we will introduce the data preparation and model parameter settings involved in the experiments of this thesis. This includes data augmentation based on CIFAR-10 dataset, the initial settings of the ResNet-18 model, and the hyperparameter settings of the D-RISE algorithm.

5.1 Preparation and Data Augmentation for CIFAR-10 Dataset

This thesis conducts classification tasks on the CIFAR-10 dataset, which contains 10 categories: cats, dogs, airplanes, cars, birds, deer, frogs, horses, ships, trucks, and consists of 50,000 training images and 10,000 test images. In each category, CIFAR-10 has the same number of images, i.e. each category has 5000 training images and 1000 test images. Each image in CIFAR-10 is of 32×32 resolution.

In our thesis, We randomly divided the 50,000 training images into a training set containing 40,000 images and a validation set containing 10,000 images. We trained our model on the training set first and treated the validation set as unseen data. This approach allows us to assess the model's generalization ability using the validation dataset during the training stage, while keeping the test dataset completely unseen.

As mentioned in *Chapter 2.2.5*, data augmentation applies various transformations to the training dataset to artificially expand it. In our study, we implemented data augmentation techniques including `padding, random cropping,` and `horizontal flipping` on every input image in the training dataset, effectively doubling the size of the training set to 80,000. Specifically, we first pad images to increase the border space, then perform random cropping to simulate different positions of the subject within the frame, and finally apply random horizontal flips to simulate different horizontal orientations of the images. These steps enable the model to adapt to changes in the input data, allowing it to learn from a broader range of images, which helps prevent overfitting and improves the generalization ability and performance of neural network models on unknown data. Additionally, we have verified that data augmentation is essential in this study, since omitting it has led to significant degradation in test accuracy in the experiments, and that is also why data augmentation on CIFAR-10 dataset is implemented.

5.2 Setup of ResNet-18 Model

This section introduces the initial settings of ResNet-18 used in this thesis. The theoretical background and relevant discussions have been informed in detail in *Chapter 2*, where readers can find more information. Here, we will summarize the reasons behind the choices of these initial settings without diving into very detailed explanations to avoid redundancy in content.

1. **Mini-Batch Size (128)**: As mentioned in *Chapter 2.2.2*, selecting a larger mini batch size may make the training process more stable, but the model may have risk to stuck in the

local minima, which may affect the performance negatively of the model, such as causing underfitting or overfitting problem. Conversely, using a too small batch size can help the model escaping local optima, but it may make the training process unstable.

In the experiment, we have attempted with mini-batch sizes of 8, 32, `128`, 256. The smaller mini-batch sizes of 8 and 32, especially the mini-batch size of 8, make the learning curve significantly unstable in the initial learning stages. It is because when the batch size is too small, the data in each batch is insufficient to represent the true distribution of the entire dataset. As a result, the gradients calculated from these batches contain a high level of randomness, which can be seen as "noise". This noise can make the updates to the model parameters during the training process unstable, sometimes even leading to degradation of model performance or oscillations during training. The large mini-batch size of 256 has a stable learning process but the test accuracy gets worse, which may be caused by the model getting stuck in local minima. A larger batch size means that more samples is considered in each update step, which reduces the randomness in parameter updates. Although this reduction in randomness can make the training process smoother, it also diminishes the model's ability to explore new areas of the parameter space, especially including those paths that might lead to better solutions or the optima.

During our attempts, the mini-batch size of `128` has both a stable learning process and good test accuracy, which made it the choice in this experiment.

2. **SGD with Momentum and Learning Rate (0.1)**: In this thesis, we employed the SGD algorithm combined with momentum, as mentioned in *Chapters 2.2.2* and *2.2.3*. The SGD algorithm helps the model stably converge to the optimal solution by selecting an appropriate mini-batch size and aids the model in escaping local minima. Additionally, the momentum method accumulates past gradient direction information, which helps the model quickly converge to the optimal solution.

   The learning rate is set at 0.1, which is a typical choice for SGD optimization strategies, which offers a rapid convergence during the early stages of training because of the step size of learning is relative big. We also incorporate a cosine decay strategy to dynamically adjust the learning rate, which periodically changes the step size. This adjustment is important for fine-tuning of the models parameters when it approaches the optimal solution. In our experiments, we also tested the initial learning rates of 0.01 and 0.001. These settings resulted in a slower convergence process compared to a learning rate of 0.1. This suggested that a learning rate of 0.1 is an effective choice for our model.

   We chose the SGD algorithm combined with momentum over the Adam optimizer for our model because the former allows manually adjustable parameters. SGD with momentum permits developers to flexibly adjust the learning rate and momentum, providing additional manual control when training a specific model. While Adam also offers an adaptive learning rate, it limits further manual tuning possibilities. In this thesis, we also experimented with the Adam optimizer using various initial learning rate settings, but the training outcomes were not as satisfactory as our current setup, which utilizes SGD with momentum, a learning rate of 0.1 combined with cosine decay.

3. **Training Epochs (100)**: After monitoring and observing the learning curve of the training model, we settled on 100 epochs. The number of epochs is set based on the learning curve having already reached a stable convergence and avoiding issues related to both underfitting and overfitting. Because the setting of this hyperparameter is closely related to the training process, more details will be discussed in *Chapter 6.1*.

5.3 Setup of hyperparameters in D-RISE

This thesis primarily adopts the settings from the D-RISE thesis by `Petsiuk et al.`[8], which includes

1. **The number of binary masks** $N = 5000$: The purpose of randomly generating 5000 binary masks is to ensure that the randomness of the binary masks can create many enough masked images, making the masked images cover or reveal different details in the original image, which is important for later decisions on which parts of the image pixels are more important than others. Although a larger N value allows us to generate more random binary masks, which aids in producing more masked images, the increase in computational load (as each image undergoes N masking operations to generate saliency maps) is substantial. Therefore, we ultimately decided to follow the setup recommendations provided in the original study by `Petsiuk et al.`[8].

2. **The probability of masking** $p = 0.5$: This setting indicates generating random 1s and 0s with the same chance in the binary masks, is also recommended by `Petsiuk et al.`[8]. Besides the intuition that this probability will make a fair masking, we have also verified the effectiveness of this setting by changing this probability value to other values, such as 0.6, 0.7, 0.8, 0,9, which shows the similar results no matter visually or mathematically as the recommended setting, which will be discussed more in detail in *Chapter 6.3* and *Chapter 7*.

6 Results

In this section, we present the experimental results of the thesis, including the training and testing outcomes, and the performance evaluation of the ResNet-18 model on the CIFAR-10 dataset. Additionally, we analyze the saliency maps generated by the D-RISE algorithm to interpret the model's behavior in the classification task.

6.1 Classification Results of ResNet-18 Model

Firstly, we will introduce the training and testing results of the ResNet-18 model based on CIFAR-10 dataset and evaluate the performance of the model through the learning curve and confusion matrix.

6.1.1 The Learning Curve and Test Accuracy

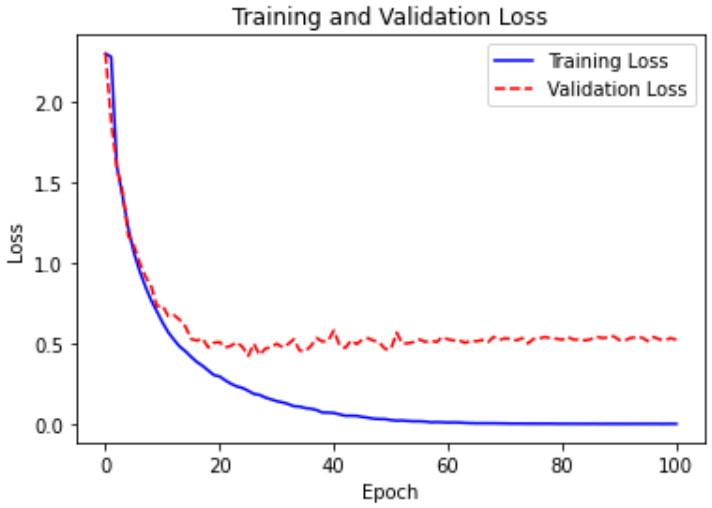


Figure 6.1: **The learning curve for ResNet-18 based on CIFAR-10**: The blue curve represents the training set, and the red curve represents the validation set. The x-axis denotes the number of epochs, while the y-axis represents the loss. Through this figure, we can observe how the loss changes as the number of epochs increases.

We used the ResNet-18 model to train on the CIFAR-10 classification task. The learning curve is shown in *Figure 6.1*. We can see that when the epoch is less than 20, the loss function decreases rapidly for both training and validation dataset, and then the learning curve gradually flattens

for validation dataset but it is till unstable, at the same time, the training learning curve continue going down. After reaching 60 epochs, the loss function has visually stably converged for both training and validation datasets. The training model is set to stop after 100 epochs, at which point the test dataset yields a classification accuracy:

$$\textbf{Test Accuracy} = 91.97\%, \tag{6.1}$$

which indicates that the generalization ability of the trained model is good.

According to the bias-variance tradeoff, finding the U-turn point is somewhat challenging in this case, since it is difficult to visually identify a clear turning point where we would expect to see the validation loss begins to increase siginificantly. Early stopping can be useful for selecting the appropriate epoch hyperparameter for the model. However, this technique was not employed in this thesis because it is not particularly suitable for the scenario shown in *Figure 6.1*.

Early stopping is a technique to avoid overfitting, which automatically halts training when the performance on the validation set begins to decline, specifically if the loss does not decrease after a pre-set number of epochs. The validity of early stopping relies heavily on significant changes in the validation set loss, particularly when the validation loss starts to increase, which typically indicates the signal of model overfitting. As observed in the learning curve, the validation loss fluctuates between epochs 20 to 60, not yet converging to a minimum value, and early stopping could mistakenly interpret these fluctuations as signs of overfitting when an increase of loss is detected. After epoch 60, both training and validation losses become very stable, which is crucial for early stopping. Without a clear increase in loss, early stopping may fail to capture the signal to cease training, leading to an inaccurate decision of when to stop the model training.

Due to the reasons mentioned above, this thesis adopts the approach of training the model multiple times, averaging the results of multiple runs to determine the optimal number of epochs. This is because the SGD optimizer is stochastic, due to the randomness in the generation of mini-batches, and thus the exploration of the parameter space for optimal solutions may vary, which causes differences in the convergence speed of the learning curves with each run. By repeatedly training and monitoring the changes in validation loss, identifying the epochs where the curves stabilize each time, and calculating the average number of epochs, the reliability of choice of this hyperparameter(Epochs) is enhanced. After conducting over 20 training experiments, it was observed that around epoch 100, the validation loss reaches or gets very close to its minimum value. Therefore, the epoch hyperparameter for this thesis is set at 100.

6.1.2 The Confusion Matrix

To gain better insights into the classification results on the test set, we can use a confusion matrix. This matrix shows how many images from each category are correctly classified and how many are incorrectly classified as other categories. As shown in *Figure 6.2*, we can observe that there are more cases of misclassification between animals than between animals and non-animals. Cats and dogs are the most representative examples in misclassification between animals, where 843 images labeled as cats were correctly classified by the ResNet-18 model, while 63 images labeled as cats were incorrectly classified as dogs. Similarly, 855 images labeled as dogs were correctly classified, and 92 images labeled as dogs were incorrectly classified as cats. This situation matches our intuition since animals tend to have more similar physical features to each other. Especially for cats and dogs, in our perception, they share similarities in color, body size, the appearance of their limbs, and even some facial features, which can confuse the model to predict right

classification. In comparison, images labeled as animals were less frequently misclassified as non-animal images. For example, only 18 images labeled as birds were misclassified as airplanes, and 10 images labeled as airplanes were misclassified as birds. This is likely because the physical characteristics between animals and non-animals are more distinct.

Figure 6.2: **Confusion Matrix for classification task on CIFAR-10:** The horizontal axis of the matrix represents the labels predicted by the model, and the vertical axis represents the true labels. Both axes have 10 labels representing the 10 categories in the CIFAR-10 dataset, and each cell's value indicates the number of classification results for that particular true label row and predicted label column. In understanding this matrix, the numbers on the diagonal represent the count of correct predictions by the model. The higher the numbers on the diagonal, the better the accuracy of the model's predictions for that category. The non-diagonal values in the matrix indicate misclassifications. For example, in the airplane row, the model incorrectly predicted 5 airplane samples as automobiles. The matrix also uses shades of color to visually display the magnitude of the numbers, where darker colors indicate higher values, allowing for quick identification of categories where the model performs well and those that the model tends to confuse.

6.2 Salinecy Maps

Through the confusion matrix, we can gain an overall understanding of correct and incorrect classifications. However, the confusion matrix does not allow us to interpret which specific features in the images lead to correct classification decisions, nor can it reveal which factors mislead the model. Instead, we need to use saliency maps to analyze the decision-making logic of the model. Saliency maps can visually show which features and details in the original images

are crucial for the model to determine the classification. We will analyze the saliency maps from three perspectives: saliency maps for correctly classified images, saliency maps for incorrectly classified images, and saliency maps that consider context features for classification.

In this chapter, we primarily selected images labeled as cats, dogs, birds, and airplanes for our case studies. The reasons for specifically choosing these pictures are as follows: On one hand, through the confusion matrix, we can see that cats and dogs are two groups of images most frequently misclassified, possibly due to the high similarity in their biological features. By using saliency maps, we can try to understand which features the model captures correctly and which features lead to incorrect classifications. On the other hand, we also observed through the confusion matrix that misclassifications between animal and non-animal categories are relatively rare. Among these, the misrecognition of birds and airplanes is quite representative. By analyzing the saliency maps, we can explore which features cause the model to confuse animals with non-animal categories.

It is worth mentioning that the algorithm used to generate the saliency maps in this thesis is an adapted version of the D-RISE algorithm. Since the original D-RISE is based on the YOLO image detector, this thesis adjusted the algorithm so that it can be applied to the ResNet-18 classification model. For specific technical details, please refer to *Chapter 4.4* of this thesis.

Moreover, in *Figures 6.3, 6.4*, and *6.5*, you will see a scale next to each saliency map. This scale reflects the values of the weighted sum of the binary mask matrices (*Eq 4.7*) to each original image, which is used to generate the saliency maps. We observe significant variations in the scales across different images, which we will explore further in the discussion part in *Chapter* 7.

6.2.1 Saliency Maps for Correctly Classified Images

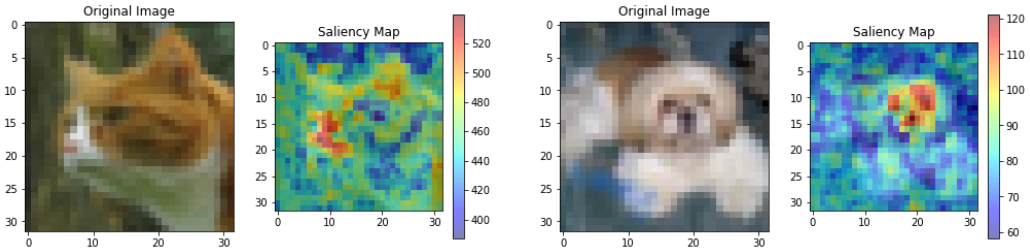
In *Figure 6.3*, we can see the saliency maps for individual images labeled as cats, dogs, birds, and airplanes that were correctly classified. The features captured by the saliency maps align with our visual perceptions. For example, for images labeled as cats and dogs, the saliency maps captured features of the head or face, which means, in some cases, the facial features are distinct enough between cats and dogs for the model to distinguish cats and dogs.

Additionally, we see that for images labeled as birds, the features captured by the saliency map are focused on the forehead and beak. For images labeled as airplanes, the features are focused on the white tips of the wings and the tail section. In this case, it is not difficult for the model to distinguish the images with physical features which themselves have significant differences. The areas highlighted in red by the saliency map can help us understand how the model correctly distinguishes these images.

6.2.2 Saliency Maps for Incorrectly Classified Images

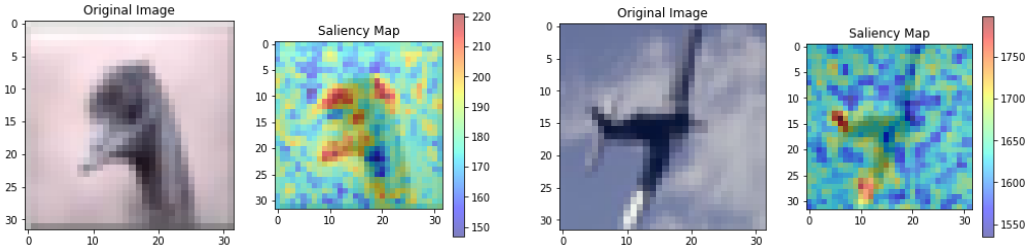
Next, let us analyze the images that were incorrectly classified. As shown in *Figure 6.4*, we see that in cases where cats are misclassified as dogs, and dogs as cats, the saliency maps still focus on the head and face areas. This shows that in some situations, the model still can not perfectly distinguish between the facial features of cats and dogs. In other words, the D-RISE algorithm might be misled by similar facial features between cats and dogs.

At the same time, we notice that in cases where birds and airplanes are misclassified as each other, the saliency map captures important features at the positions of wings and wingtips for



(a) Cat classified as Cat

(b) Dog classified as Dog



(c) Bird classified as Bird

(d) Airplane classified as Airplane

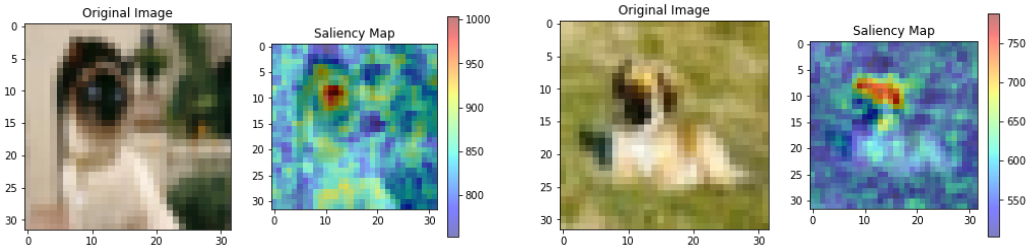
Figure 6.3: **Saliency maps for correct classifications:** This set of images includes four subfigures, and only correctly classified images are displayed. Each subfigure contains two parts: the left side shows the original image, and the right side shows a DRISE saliency map. The saliency map is used to indicate which areas most significantly influence the model's classification decisions. The color changes within the map highlight the importance of specific features in the image for the model to accurately classify them. The redder the area, the more important the feature. The color scale next to each saliency map indicates the range of saliency values, representing the range of values in the weighted sum of mask matrices (*Eq 4.7*).

both birds and airplane images. This indicates that the model struggles to classify correctly when the wings of birds and wingtips of airplanes are somewhat similar.

6.2.3 Context Detecting in Saliency Maps

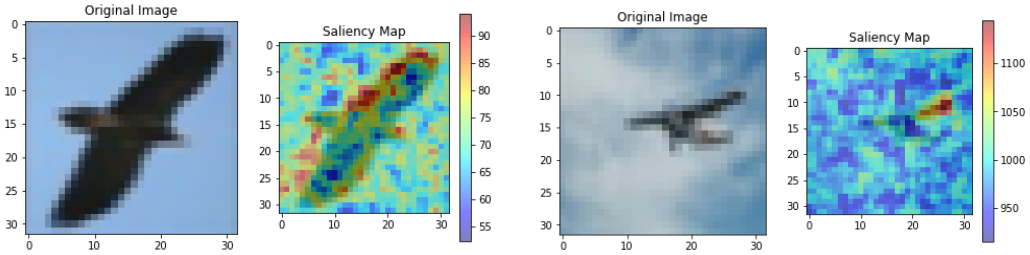
In some situations, we find that the saliency maps capture more than just the features of the target object itself. They can also highlight important information in the background. For example, as shown in *Figure 6.5*, in images correctly classified as airplanes, some saliency maps prominently show the clouds in the sky. At the same time, we also see that for pictures classified as birds, the highlighted areas on the saliency map include not only the bird's head and beak but also the branch on which the bird is standing. This indicates that the model, when classifying, relies not only on the features of the target itself but also uses background information as a basis for classification.

In summary, the saliency maps generated by using the D-RISE algorithm visually highlight the important parts of the images for classification, which means the saliency maps have passed visual validation. To further verify the validity of the saliency maps based on D-RISE, we also need to conduct further mathematical quantitative analysis, which is discussed in detail in the next chapter.



(a) Cat classified as Dog

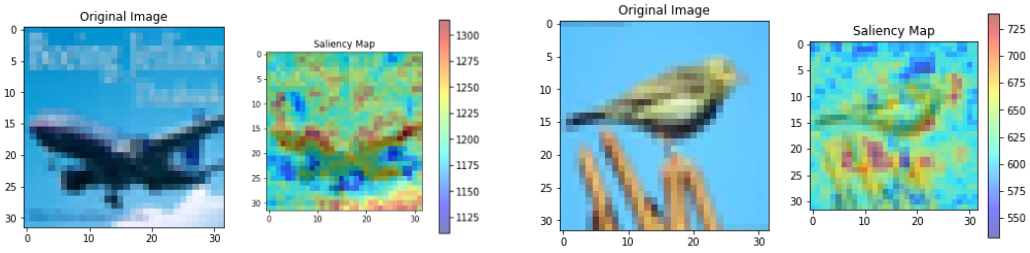
(b) Dog classified as Cat



(c) Bird classified as Airplane

(d) Airplane classified as Bird

Figure 6.4: **Saliency maps for incorrect classifications:** This set of images includes four subfigures, showing the incorrectly classified images with corresponding DRISE saliency maps.



(a) Bird classified as Bird

(b) Airplane classified as Airplane

Figure 6.5: **Saliency maps with context detection:** This set of images includes two subfigures, showing the cases where saliency maps highlight the context information in the cases of correct classification.

7 Discussion and Summary

As we seen in *Figure 6.3*, *Figure 6.4* and *Figure 6.5*, the features captured by the saliency maps matched our intuitive feelings. However, to ensure the validity of the D-RISE saliency maps, we also need to verify this method mathematically. Here we noticed some issues, which may suggest limitations of the D-RISE algorithm when applied to the ResNet-18 classification model. To simplify, from this point forward, the weighted sum of the binary masks (as defined in *Eq 4.7*) will be referred to as the D-RISE matrix.

7.1 Scale Problem in D-RISE Matrix

We have noticed that the scale of element values in the D-RISE matrix vary significantly among the saliency maps, in some of them showing an extremely low scale. For example, a saliency map for correct classifications shows a scale of 60-120 (see *Figure 6.3b*), while another saliency map for incorrect classifications has a scale of 55-90 (see *Figure 6.4c*). To determine whether these scales are normal, we need to quantify reasonable values for the saliency maps.

We know there are 1024 (32×32) values in the weighted sum matrix. The upper bound of the expectation value of each element in D-RISE matrix is:

$$\text{Upper bound} = 1 \cdot 5000 \cdot 0.5 = 2500. \tag{7.1}$$

Here means, every element in a binary mask has expectation value $P(1) \cdot 1 + P(0) \cdot 0 = 0.5$, where $P(0) = 0.5$ is the masking probability. Here, for each mask of the 5000 binary masks, the corresponding similarity score is taken maximum 1, which means every mask leads to a totally correct classification. On the other hand, if each masked image is a uniform guess with a similarity score of 0.1, the minimum expected value of each element in D-RISE matrix would be:

$$\text{Lower bound} = 0.1 \cdot 5000 \cdot 0.5 = 250. \tag{7.2}$$

The situation, that elements values in the D-RISE matrix below the bound in *Eq.7.2*, could theoretically occur, but it is hard to interpret in practice. For example, if every random mask just leads to a random guess in predictions, then we can not trust on the saliency map based on the D-RISE matrix, because the captured features do not contribute something more than a random guess. This means that the features highlighted in the saliency maps are not necessarily more important than those that could lead to a correct prediction through random guessing.

7.2 Refining Similarity Metrics in D-RISE

To improve the mathematical validity of the D-RISE method, we tried different approaches to redefine the similarity score. This idea is based on the fact that a saliency map is generated by a

D-RISE matrix, whose values originate from the weighted sum of binary masks, referring to *Eq 4.7*. The generation of binary masks is primarily controlled by the masking probability, which did not address the scale problem after experimenting with other different masking probability values of $0.6, 0.7, 0.8$ and 0.9. Therefore, we decided to modify the weights(*similarity score*) by employing different mathematical quantities instead.

The primary idea involves using cross-entropy instead of the dot product to calculate the similarity score (*Eq 4.5*). Cross-entropy is chosen because it naturally measures the difference between the true label vector and the predicted probabilities, and it is also the lost function of the model to evaluate the performance of the classification task. According to the definition of cross-entropy, the formula is as follows:

$$H(p, q) = -\sum_{x} p \log q, \tag{7.3}$$

where p is the true label of the original image and q is the predicted probability to the correct category for the masked image. We know that when the prediction perfectly matches the true label, the cross-entropy H_{min} is 0, which will give the lower limit of element value S_{min} in D-RISE matrix:

$$H_{min} = -1 \cdot \log_2(1) = 0 \text{ bit}, \tag{7.4}$$

$$S_{min} = 0 \times 5000 \times 0.5 = 0. \tag{7.5}$$

When the prediction is a random guess with the correct category probability at 0.1, the maximum value of cross-entropy is:

$$H_{max} = -1 \cdot \log_2(0.1) \approx 3.3 \text{ bits}. \tag{7.6}$$

At this point, our expected upper limit for element value in D-RISE matrix is:

$$S_{max} = 3.3 \times 5000 \times 0.5 = 8250, \tag{7.7}$$

It is important to note that the lower limit of the similarity score corresponds to the best case scenario, where all masks have perfect predictions, whereas the upper limit represents the worst expectations, indicating that no mask performs better than random guessing. As shown in *Figure 8.2* in appendix, using cross-entropy as a similarity score does not solve the problem of extreme scale values.

In the similar way, we also experimented with other methods to calculate the similarity score, including using the formulas: $e^{\|v_1 - v_2\|^2}$, $(1 + \|v_1 - v_2\|^2)^{-1}$, etc, where v_1 represents the one hot encoded vector for true label and v_2 represents the prediction of the masked image. These mathematical methods can also be used to represent the similarity or difference between two vectors. However, these methods for calculating similarity scores encounter the same issues with extreme values in the D-RISE matrix.

7.3 Summary

In summary, we successfully trained the ResNet-18 model for classification tasks using the CIFAR-10 dataset. By monitoring the learning curve, we identified suitable hyperparameter settings, and we also confirmed the model's good generalization ability, achieving 91.97% classification accuracy on the test set. We also explored which features in the image captured by

the model for make classification decisions by employing saliency maps based on the D-RISE algorithm. These saliency maps visually matched our conceptual expectations and successfully captured important features in the images.

However, there are still potential issues with the mathematical validity of the saliency maps when the elements in the D-RISE matrix exceed the highest or lowest expected values. Preliminary attempts to redefine the similarity score from the original thesis did not yield significant improvements. Based on this, we believe that using the D-RISE algorithm from the literature for intuitive display of saliency maps on the ResNet-18 model may be feasible, but we need to take into account the scale of the element values in the D-RISE matrix. If it shows a mathematically extreme case in scale, we must be cautious when interpreting or trusting the corresponding saliency map. To evaluate the validity of the D-RISE method when applied to the ResNet-18 model based on CIFAR-10 dataset, it may be necessary to summarize the proportion of images where a scale problem arises. This analysis will reveal whether the method consistently causes a scale issue, thereby providing insights into the algorithm's robustness and practical utility. Given the significant computational demand involved in this process (We have 10,000 images in the test set, with each undergoing 5,000 mask calculations to generate a saliency map), this could be an area for further investigation. Additionally, exploring other mathematical methods to calculate the similarity score can also be considered as future work. It is also worth noting that the modification of the objectness score (*Eq. 4.6*) can be a direction for further research. This is because ResNet-18 does not provide this metric, and according to `Petsiuk et al.`[8], the objectness score is always set to 1. However, this approach gives every mask the same chance to contribute to the D-RISE matrix, including masks that may negatively affect the detection. Therefore, the objectness score value of 1, as suggested by `Petsiuk et al.`[8], may have limitations when applied to models that do not generate an objectness score. In such cases, the objectness score may need to be redefined.

8 Appendix

8.1 An Example of Learning Curve

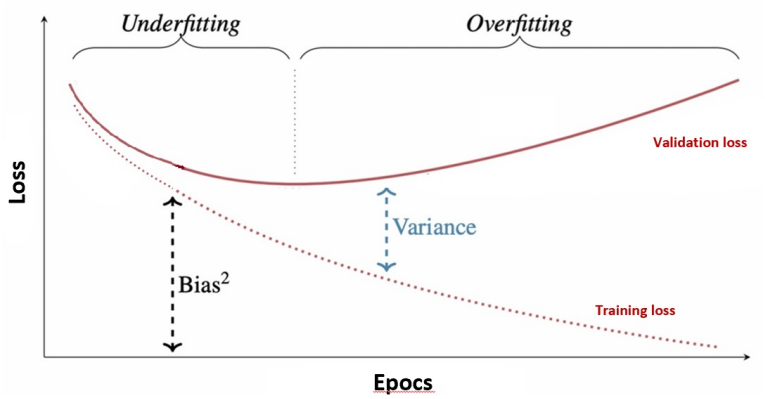
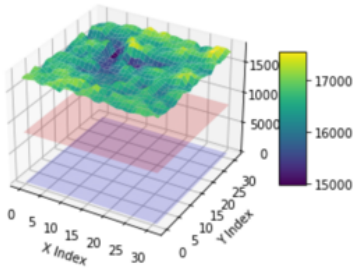
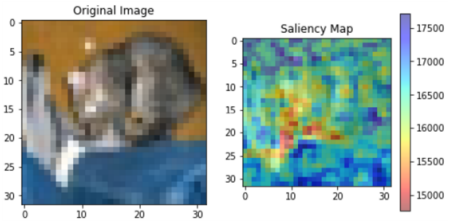


Figure 8.1: **Learning curve**: A curve illustrates the training process of the model. The horizontal axis represents the training epochs, and the vertical axis shows the loss. On the left side, labeled "*Underfitting*," both the training loss (dotted line) and the validation loss (solid line) converge but remain high, indicating the model is too simple to capture the underlying pattern in the data, a condition known as high bias. As the epochs increase, moving towards the right side labeled "*Overfitting*," the training loss continues to decrease, eventually approaching zero. This suggests that the model fits the training data extremely well. However, the validation loss begins to diverge and rise after a certain point, indicating the model has started to fit the noise in the training data rather than generalizing from it, leading to high variance. The curve demonstrates the tradeoff between bias and variance, a fundamental challenge in machine learning, emphasizing the importance of finding the optimal point where both training and validation losses are minimized for effective model training.

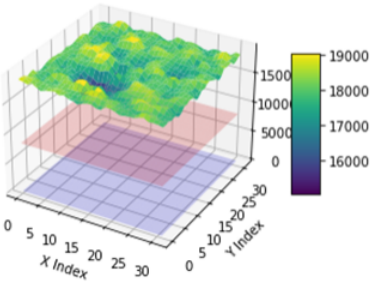
8.2 Figure for Scale problem in D-RISE



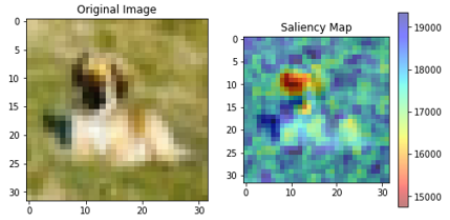
(a) D-RISE matrix values scale for b



(b) Cat classified as Cat



(c) D-RISE matrix values scale for d



(d) Dog classified as Cat

Figure 8.2: **D-RISE matrix values scale vs. Saliency map for cross-entropy as similarity score:** The figure contains four subfigures. On the left, there is a 3D plot used to show whether the elements of the D-RISE matrix fall within a reasonable range. The purple plane at the bottom represents the lower bound (*Eq 7.5*), while the pink plane in the middle represents the upper bound (*Eq 7.7*). Together, they determine a reasonable range of scale values. The green plane at the top represents the actual element values of the D-RISE matrix, which significantly exceed the upper bound. The subfigures on the right include the original images along with the corresponding saliency maps. Here, although using cross-entropy as a similarity score successfully highlights reasonable important features for classification in saliency maps, it is not mathematically reliable.

Bibliography

[1] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[6] NVIDIA Corporation. *CUDA Toolkit Documentation*. NVIDIA Corporation, 2024. Available at: `https://docs.nvidia.com/cuda/doc/index.html`, Accessed: 2024-05-17.

[7] NVIDIA Corporation. *cuDNN Documentation*. NVIDIA Corporation, 2024. Available at: `https://docs.nvidia.com/deeplearning/cudnn/`, Accessed: 2024-05-17.

[8] Vitali Petsiuk, Rajiv Jain, Varun Manjunatha, Vlad I Morariu, Ashutosh Mehra, Vicente Ordonez, and Kate Saenko. Black-box explanation of object detectors via saliency maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11443–11452, 2021.

[9] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[10] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[11] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*, 2017.

[12] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[13] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Visualising image classification models and saliency maps. *Deep Inside Convolutional Networks*, 2:2, 2014.

[14] Xin Tang. Relevant codes for degree project in computer science for spring semester 2024 (su), 2024. Available at: `https://github.com/bupttaxi/Degree-Project-in-CS-SU-`, Accessed: 2024-05-19.

[15] TensorFlow Development Team. *TensorFlow: Large-scale machine learning on heterogeneous systems*. Google Brain, 2024. Available at: `https://www.tensorflow.org/`,Accessed: 2024-05-17.

[16] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

Datalogi
www.math.su.se

Beräkningsmatematik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm