Improving CNN Robustness via CS Shapley Value-guided Augmentation

E4040.2024Fall.MYZH.report.mz3088
Mingyu Zhang mz3088
Columbia University

Abstract

This project aims to replicate the findings of the paper "Rethinking and Improving Robustness of Convolutional Neural Networks: a Shapley Value-based Approach in Frequency Domain" by Chen et al. (2022). The project largely achieved the original goal, including the construction of the ResNet18 model and the implementation of the Class-wise Shapley value-guided Augmentation (CSA) method, which uses frequency components with negative contributions from the same class for data augmentation to improve the robustness of CNNs. Notably, I achieved a higher accuracy than the original paper's results in the PDG-20 adversarial attack.

1. Introduction

As I read in the research by Chen Yiting, Ren Qibing, and Yan Junchi, despite the powerful capabilities of Convolutional Neural Networks (CNNs) in many computer vision tasks, CNNs are vulnerable to adversarial examples, which are small, imperceptible noise patterns that humans cannot detect but cause the model to fail. In the frequency domain, unlike many studies that suggest CNNs are more affected by high-frequency components (HFC), they chose to consider the contribution of each individual frequency component in both Low-Frequency Components (LFC) and High-Frequency Components (HFC) through Shapley values. Shapley values, originally from game theory, are a method for allocating contributions in cooperative games, used to measure each participant's contribution to the outcome. In machine learning, Shapley values are widely applied in feature importance evaluation and model interpretation, helping us understand the role each feature plays in the model's decision-making.

By identifying positive frequency components (PFC) and negative frequency components (NFC) that contribute to the model's predictions, and enhancing the data based on the NFC found in similar data samples, they achieved robustness. Unlike previous studies that focused on qualitative dataset-level analysis, they proposed a simple yet principled data augmentation approach. Using a ResNet-based network, they demonstrated significant robustness improvements on the CIFAR-10 dataset.

Technically, the goal of this project is to first build and fine-tune a CNN model to achieve high accuracy on a dataset. Since the calculation of Shapley values depends on a pre-trained model to compute the influence of a specific data point (in this case, a frequency) on the overall prediction result, the objective is to extract a sufficient number of images for each category and calculate the influence of each frequency on the prediction results in terms of Shapley values. This allows us to identify frequencies that have a positive impact on the prediction result (positive Shapley value), referred to as PFC (Positive Frequency Component), and frequencies that have a negative impact (negative Shapley value), referred to as NFC (Negative Frequency Component).

With these tools in hand, the ultimate goal is to build a network guided by Class-wise Shapley value augmentation (CSA) and test its performance under adversarial attacks and adversarial training conditions. The broader goal includes attempting to build a CNN model from scratch, experimenting with structures and parameters to achieve good predictive results, learning more about adversarial training and the frequency domain of images, and attempting to fully understand a more scientific approach—rather than an engineering perspective—to improve the strengths of deep learning models, such as robustness.

The main difficulties I encountered included building the model from scratch. Understanding the model structure and parameter passing was essential for me, as well as identifying mistakes in the model I was writing, whether structural or syntax errors. Additionally, the training duration and resources required were significant. To achieve a good accuracy with the ResNet model, I had to train it for many epochs, far more than any model I had worked on in assignments during the semester. Moreover, adapting ResNet18 to the CIFAR-10 dataset took time, especially with efforts to prevent overfitting, which involved many attempts. Understanding how images work in the frequency domain was a long learning process, especially since I lacked prior knowledge in this area. As a result, it was difficult to think of built-in TensorFlow functions related to frequency domain calculations, and the functions I tried to write myself inevitably contained errors. Overall, I believe patience and trying to understand the underlying logic are the best approaches to facing these challenges. Despite the difficulties, the satisfaction after solving a challenge is even greater.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

The original paper aimed to find a more reasonable method rather than just considering HFC and LFC. The frequency

part can be calculated by Discrete Fourier Transform (DFT): $\mathcal{F}(\cdot): R^{d_1 \times d_2} \to C^{d_1 \times d_2}$

$$\mathcal{F}(X)(u,v) = \sum_{m=0}^{d_1-1} \sum_{n=0}^{d_2-1} X(m,n) e^{-i2\pi\left(\frac{mu}{d_1}+\frac{nv}{d_2}\right)},$$

and Inverse Discrete Fourier Transform (IDFT) $\mathcal{F}^{-1}(\cdot): C^{d_1 \times d_2} \to R^{d_1 \times d_2}$

$$X(m,n) = \frac{1}{d_1 d_2} \sum_{u=0}^{d_1-1} \sum_{v=0}^{d_2-1} \mathcal{F}(X)(u,v) e^{i2\pi\left(\frac{mu}{d_1}+\frac{mv}{d_2}\right)},$$

through which they can achieve the transformation of images in the frequency domain.

Next, they considered Shapley value which considers a set of playersin a cooperative game, where each player has a contribution value, the Shapley value of a specific player represents their average contribution across all possible feature combinations.

$$\phi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!\,(n-|S|-1)!}{n!} [v(S \cup \{i\}) - v(S)]$$

This evaluates the average contribution of player by considering all possible orders in which features can be added and calculating the difference in value between adding feature i to a subset S and not adding it. Shapley value has many desirable properties such as efficiency, symmetry, and linearity.

Consider frequency components in each image as the player, and they cooperate with each other and lead to the prediction results of the model. To apply the Shapley value to the frequency components, define the set of players for data sample (X, y) as:

$$\mathcal{N}_{\mathbf{X}} = \{\mathcal{F}(\mathbf{X})(0,1), \cdots, \mathcal{F}(\mathbf{X})(d_1 - 1, d_2 - 1)\}$$

For a model $f(\cdot)$, they defined the output of the model on $S \subseteq \mathcal{N}$ as:

$$f(S) = f\left(\mathcal{F}^{-1}(\sigma_S \odot \mathcal{F}(\mathbf{X}))\right)_{\mathbf{y}}$$

where $f(\cdot)_{\mathbf{y}}$ denotes the y -th output of the model and $\odot$ denotes Hadamard product. $\sigma_S \in [0,1]^{d_1 \times d_2}$ is the mask generated according to S satisfying:

$$\sigma_S(u,v) = \mathbb{1}[\mathcal{F}(\mathbf{X})(u,v) \in S]$$

where $\mathbb{1}(\cdot)$ is the indicator function.

They proposed to augment data with features composed by NFC of the same class, named Class-wise Shapley value-guided Augmentation (CSA). With a CNN model f, the CSA p_X^f is to result of masking the PFC of image X

$$p_X^f(u,v) = \mathbb{1}\left[\phi_{u,v}^{X,f} < 0\right] \cdot \mathcal{F}(X)(u,v)$$

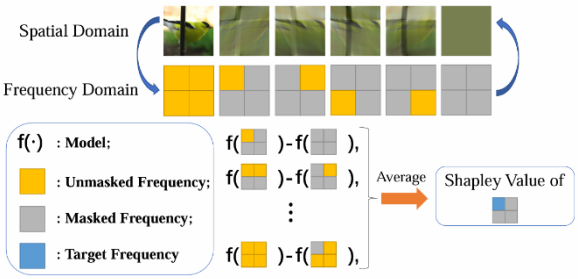
For each class c, they randomly took data samples (X_1, c), (X_2, c), ..., (X_n, c) and generate a set of CSAs with a pretrained model $\hat{f}$, where $\mathcal{P}_c^{\hat{f}} = \left\{p_{X_1}^{\hat{f}}, p_{X_2}^{\hat{f}}, \cdots, p_{X_n}^{\hat{f}}\right\}$ and trained their model with

$$\theta = \arg\min_{\theta} \mathbb{E}\left[\max_{\delta<\|\epsilon\|_p} \ell\left(f\left[\mathcal{F}^{-1}\left(\mathcal{F}(X) + \alpha * p^{\hat{f}}\right) + \delta\right], y\right)\right], p^{\hat{f}} \in \mathcal{P}_y^{\hat{f}}$$

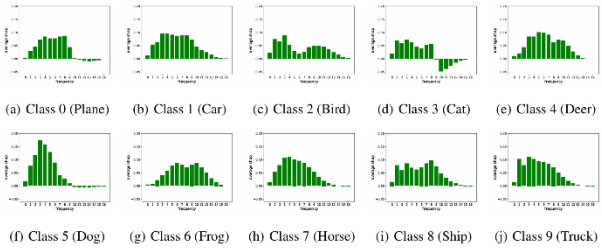
where the α is a coefficient of the CSA $p^{\hat{f}}$.

2.2 Key Results of the Original Paper

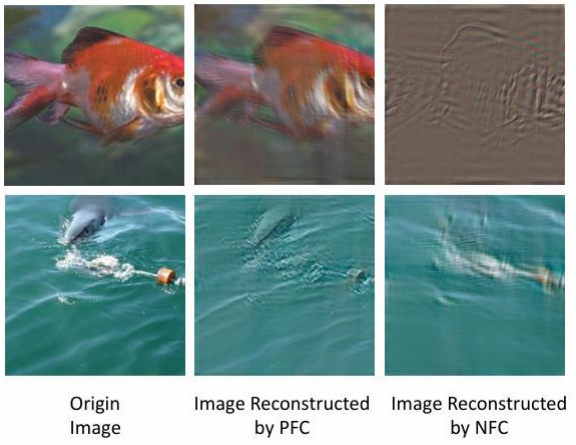
To help understander computations of Shapley valyes under frequency bias , they illustrated the 2 ×2 demo as below.



They showed that the frequency bias varies at the instance-level.:



To gain more understanding of NFC and PFC, they showed below origin image, image reconstructed with PFC, images reconstructed with NFC, and the heat map of Shapley value in frequency domain.



They used ResNet18 as backbone model and performed adversial training and adversarial attack on the models with and without CSA. This is their results:

Methods	Clean Acc	Acc under PGD-20	Acc under Auto attack
PGD-AT	**84.49**	46.38	44.06
PGD-AT+CSA	82.91	**49.42**	**46.56**
TRADES	**81.71**	51.08	47.74
TRADES+ CSA	81.62	**52.06**	**49.16**

3. Methodology of the Students' Project

First, I will attempt to build the baseline model ResNet-18, and all subsequent calculations will be based on this classic model. After obtaining a reasonably accurate ResNet-18 model trained on the CIFAR-10 dataset, I will use this model to obtain the NFC and PFC for each class of images. Based on the NFC, I will build a CSANet model with ResNet-18 as the backbone, and test its performance with and without adversarial attacks and adversarial training, comparing the results.

3.1. Objectives and Technical Challenges

The original paper includes a lot of content, much of which I omitted, such as their exploration of adversarial attacks focused on LFC and the analysis of fairness in adversarial training within the frequency domain. I primarily focused on building the model and improving its robustness, especially in relation to the results listed in the tables.

My initial goal was to establish and fine-tune a ResNet-18 model, trained on the CIFAR-10 dataset, and achieve high accuracy by adjusting the model's parameters and results. Since the computation of Shapley values relies on this model, it's critical that the model achieves good accuracy. A model with poor accuracy could easily misjudge PFC and NFC, directly affecting the final results of data augmentation with Shapley values. The first issue I encountered was severe overfitting in the ResNet model. As is commonly known, the first convolution layer of ResNet-18 is a 7×7 convolution (stride=2) followed by 3×3 max pooling (stride=2). However, during training, this structure led to serious overfitting. I tried increasing data augmentation and adding dropout, but the results were only average. Additionally, I found that learning rate schedulers and warm-up techniques, which I learned about in other sources, didn't have much of an effect.

After reading other papers on training for CIFAR-10 specifically, I realized that the CIFAR-10 images are too small for this structure. So, I modified the model to use smaller 3x3 convolutions with smaller strides. Also, because the images are small, the max-pooling layer was ineffective, and I had to pay more attention to the inputs and outputs of the fully connected layers. Once I addressed these issues, I focused on keeping the data augmentation moderate to prevent overfitting, while continually monitoring the training loss to ensure no obvious overfitting.

The training process was quite long. My objective was to achieve a well-trained model with high accuracy, so I spent a lot of time fine-tuning the ResNet model. The biggest challenge was the large number of images and the depth of the ResNet-18 model, which resulted in long training times and required many epochs. Even with large CPU virtual machines on GCP, the training process frequently crashed. In such cases, I made sure to record checkpoints during training. I also noticed that if I set a seed to reproduce results, the overfitting phenomenon became more pronounced. On the other hand, when I didn't set the seed, the testing accuracy dropped slightly when resuming from saved checkpoints. To save time, I applied for more quota on GCP and ran multiple virtual machines simultaneously. Each time I did this, I received results much faster.

Calculating Shapley values in the frequency domain required more understanding on my part. Initially, I naively tried to compute the values based on the formulas from the paper, but the results were poor and inaccurate. During my research, I learned that TensorFlow has pre-built functions for these calculations, which made the process much easier. However, it also required careful attention to the format conversion of image data and maintaining consistency in data types.

Additionally, understanding the concept of "class-wise" was a learning process. My first approach was to divide the frequencies of the image into components, such as a, b, and

c. I then considered training the model on 100 images using only frequency a, another set of 100 images using frequencies a and b, and subtracting their differences, and so on, creating many combinations. However, intuitively, I realized that this approach would be difficult to implement due to the computational load. The number of combinations increases with the number of classes, and each combination would require retraining the model. In reality, the calculation works differently. Instead, the paper calculates Shapley values for each image and its individual frequencies by feeding the image with certain frequencies into the trained model and computing prediction accuracy. Even so, during actual implementation, I found that the calculated Shapley values were unstable and the number of calculations was too large to process. I even tried disabling the GPU, but the process still crashed. I believe that with more knowledge in algorithm acceleration, I could achieve better results. To get better results in the subsequent CSA process, I decided to use the values provided in the paper for further training.

The process of applying theoretical knowledge in practice and correcting my understanding was very interesting. I also gained more insights into adversarial training and the frequency domain of images.

Finally, I built the CSANet model, which took the most time. The main reason was that I didn't have any clear reference or discussion regarding its structure. When setting up the trainer, I wasn't sure if I was doing it right, such as calculating and passing the adversarial loss. It wasn't until the model was completely built that I could verify the results, but I often had to adjust the entire structure along the way. Learning how to implement adversarial training and attacks required me to start over, but compared to other tasks, it wasn't too complicated, though it involved many parameters. CSANet uses ResNet as the backbone, which made passing the model very challenging. The part that took the most time was dealing with gradient calculations. I kept receiving warnings that some of my gradient variables were zero. Eventually, I found that adding a single line of code addressed this issue. I believe the CSANet model works well and largely achieved the objectives I set for the project.

3.2.Problem Formulation and Design Description

Consider a data set of $(\boldsymbol{X}, \boldsymbol{y})$ where $X \in R^{d_1 \times d_2}$ is the input image and $y \in \{1, \dots, C\}$ is the label (C classes of images). Consider a neural network $f(\cdot, \theta): R^{d_1 \times d_2} \to R^C$. Discrete Fourier Transfom (DFT) $\mathcal{F}(\cdot)$ map the input image (finit signal) of $R^{d_1 \times d_2}$ to the frequency domain of $C^{d_1 \times d_2}$. $\mathcal{F}(\cdot): R^{d_1 \times d_2} \to C^{d_1 \times d_2}$ can be computed as:

$$\mathcal{F}(X)(u,v) = \sum_{m=0}^{d_1-1} \sum_{n=0}^{d_2-1} X(m,n) e^{-i2\pi\left(\frac{mu}{d_1}+\frac{nv}{d_2}\right)}.$$

Inverse Discrete Fourier Transform (IDFT) is therefore $\mathcal{F}^{-1}(\cdot): C^{d_1 \times d_2} \to R^{d_1 \times d_2}$ and can be computed as:

$$X(m,n) = \frac{1}{d_1 d_2} \sum_{u=0}^{d_1-1} \sum_{v=0}^{d_2-1} \mathcal{F}(X)(u,v) e^{i2\pi\left(\frac{mu}{d_1}+\frac{mv}{d_2}\right)}.$$

Adversarial attack refers to the process of introducing a small perturbation δ, bounded in l_P space, that causes the model to make incorrect predictions:

$$\delta = \arg\max_{\delta \mathrm{l}} (f(X+\delta), y), \quad s.t. \quad |\delta|_p \le \epsilon.$$

Adversarial training is a very effective defense strategies, typically involving the minimization of the loss function on adversarially perturbed training data.

Shapley value considers a set of players $N = \{1,2, \dots, n\}$ in a cooperative game, where each player has a contribution value, the Shapley value of a specific player i (i.e., a feature, a pixel or a certain frequancy) represents their average contribution across all possible feature combinations. Let $v(S)$ be the value function for a subset $S \subseteq N$, representing the total reward (or prediction result in machine learning) obtained by the cooperation of the players in S. The Shapley value $\phi_i(v)$ of player i is given by:

$$\phi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!\,(n-|S|-1)!}{n!} [v(S \cup \{i\}) - v(S)]$$

where $S \subseteq N \setminus \{i\}$ is any subset of players excluding player i, $v(S)$ is the value function of the subset S, representing the contribution of players in S, and $|S|$ is the size of the subset S, and n is the total number of players.

This evaluates the average contribution of player i by considering all possible orders in which features can be added and calculating the difference in value between adding feature i to a subset S and not adding it. Shapley value has many desirable properties such as efficiency, symmetry, and linearity.

Consider frequency components in each image as the player, and they cooperate with each other and lead to the prediction results of the model. To apply the Shapley value to the frequency components, define the set of players for data sample (X, y) as:

$$\mathcal{N}_{\mathbf{X}} = \{\mathcal{F}(\mathbf{X})(0,1), \cdots, \mathcal{F}(\mathbf{X})(d_1-1, d_2-1)\}$$

For a model $f(\cdot)$, they defined the output of the model on $S \subseteq \mathcal{N}$ as:

$$f(S) = f\left(\mathcal{F}^{-1}(\sigma_S \odot \mathcal{F}(\mathbf{X}))\right)_{\mathbf{y}}$$

where $f(\cdot)_{\mathbf{y}}$ denotes the y -th output of the model and $\odot$ denotes Hadamard product. $\sigma_S \in [0,1]^{d_1 \times d_2}$ is the mask generated according to S satisfying

$$\sigma_S(u,v) = \mathbb{1}[\mathcal{F}(\mathbf{X})(u,v) \in S]$$

where $\mathbb{1}(\cdot)$ is the indicator function.

As by definition, frequency components with negative Shapley value harms the prediction accuracy of the model. By proactively correcting the misalignment of features of the model, I can improve the adversarial robustness for future attacks. The paper proposed to augment data with features composed by NFC of the same class, named Class-wise Shapley value-guided Augmentation (CSA). With a CNN model f, the CSA p_X^f is to result of masking the PFC of image X

$$p_X^f(u,v) = \mathbb{1}\left[\phi_{u,v}^{X,f} < 0\right] \cdot \mathcal{F}(X)(u,v).$$

For each class c, they randomly took data samples (X_1, c), (X_2, c), ..., (X_n, c) and generate a set of CSAs with a pretrained model $\hat{f}$, where $\mathcal{P}_c^{\hat{f}} = \left\{p_{X_1}^{\hat{f}}, p_{X_2}^{\hat{f}}, \cdots, p_{X_n}^{\hat{f}}\right\}$ and trained their model with

$$\theta = \arg\min_{\theta} \mathbb{E}\left[\max_{\delta < \|\epsilon\|_p} \ell\left(f\left[\mathcal{F}^{-1}\left(\mathcal{F}(X) + \alpha * p^{\hat{f}}\right) + \delta\right], y\right)\right], p^{\hat{f}} \in \mathcal{P}_y^{\hat{f}}$$

where the α is a coefficient of the CSA $p^{\hat{f}}$

4. Implementation

In this section, I will provide detailed discussion about my implementation.

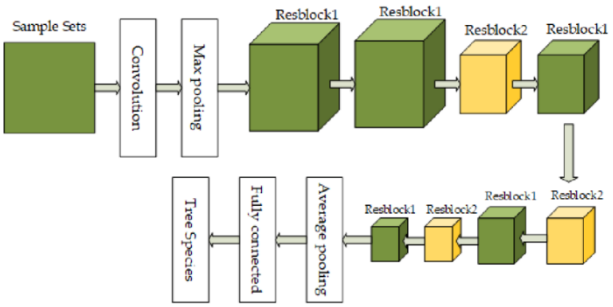
4.1 Data

The CIFAR-10 dataset is a widely used benchmark dataset for image classification tasks, created by Alex Krizhevsky and Geoffrey Hinton at the University of Toronto. It consists of 60,000 images, each of which is a 32x32 pixel color image spanning three channels (RGB). These images are evenly distributed across 10 distinct classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each class contains 6,000 images, with 50,000 designated for training and the remaining 10,000 reserved for testing purposes. The dataset is of low resolution and balanced class distribution. CIFAR-10 images show diversity in terms of perspectives, lighting conditions, and backgrounds. We can also easily access CIFAR-10 through TensorFlow frameworks with built-in functions. I will use this data to train my ResNet model, sample their Shapley values, and test on my CSANet model.

4.2 Deep Learning Network

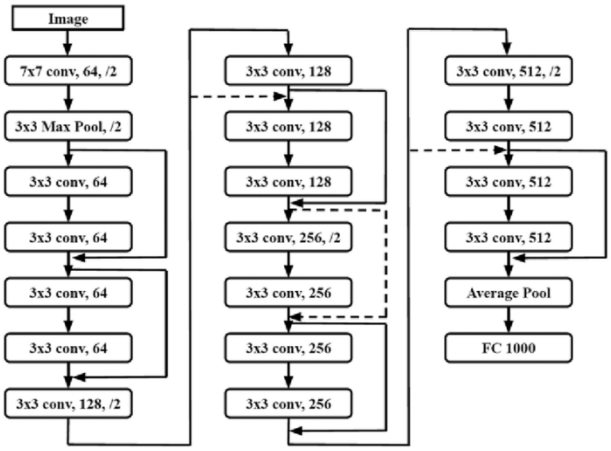
In this project, I primarily use the ResNet18 model. ResNet (Residual Network) is a deep convolutional neural network architecture proposed by He et al. in 2015. Due to its outstanding performance and innovative design, it has had a profound impact on the field of computer vision. The core idea of ResNet is to address the common issues of vanishing gradients and network degradation in deep networks by introducing residual blocks and skip connections. In traditional deep neural networks, as the number of layers increases, the model's performance often decreases due to optimization difficulties. However, ResNet's residual blocks allow the network to more easily learn identity mappings by learning the residual between the input and output, which significantly reduces the difficulty of optimization.

Residual blocks are the fundamental components of ResNet. There are two main types of residual blocks: the Basic Block, used in shallower networks like ResNet-18, and the Bottleneck Block, used in deeper networks. The Basic Block typically consists of two 3×3 convolutional layers, while the Bottleneck Block uses 1×1 convolutions for channel compression and expansion, combined with 3×3 convolutions to reduce computational complexity. Both types of residual blocks incorporate skip connections that directly add the input to the output, creating the "residual" structure.



Taking ResNet-18 as an example, its network structure includes an initial 7×7 convolutional layer followed by a 3×3 max pooling layer. It then consists of four stages made up of multiple bottleneck residual blocks, where the number of convolutional filters increases progressively in each stage. Finally, the network performs the classification task through global average pooling and a fully connected layer.

This modular design allows ResNet to support depths of hundreds or even thousands of layers, enabling it to achieve high accuracy on large datasets like CIFAR-10 while also reducing the model's computational complexity.



In the actual model construction, applying ResNet18 directly to the CIFAR-10 dataset resulted in severe overfitting. Recall that the first convolutional layer in ResNet18 is a 7×7 convolutional layer (stride=2) followed by max pooling (3×3, stride=2). However, the CIFAR-10 images are too small for this structure. To address this, I changed the 7×7 convolutional layer to 3×3 and reduced the stride as well. Additionally, since the images are small, the max pooling layer was also not effective, so I replaced it with Conv2D.

To prevent overfitting, I used BatchNormalization, dropout, and regularization, with BatchNormalization and dropout providing the best results. Regarding the learning rate, I tried using a learning rate warmer, but it didn't show significant effects. The learning rate scheduler also didn't have a very pronounced impact. I decided to use SGD combined with Momentum to alleviate gradient oscillations during optimization and accelerate convergence. Since CIFAR-10 images are small and the training iterations are numerous, momentum significantly improved efficiency.

Once overfitting was addressed, I didn't apply strong data augmentation—just enough to ensure that the training loss was closely monitored, and no significant overfitting occurred. I used padding and random horizontal flips. For batch size and epoch selection, a batch size of 128 worked well, with each epoch taking about one minute to train. Using 60 epochs, I achieved around 90% accuracy, and with 140 epochs, I reached about 92% accuracy.

4.3 Software Design

The link to my codes is https://github.com/ecbme4040/e4040-2024fall-project-MYZH-mz3088

First, to sample the data (combinations of players), I generate a mask. The pixels or frequencies that are masked will not be considered during the input to the model and the prediction process.

```
DEF sample_mask(img_w, img_h, mask_w, mask_h, static_center=False):
    # Step 1: Initialize mask and permutation order
    SET length TO mask_w * mask_h + 1
    SET order TO RANDOM_PERMUTATION(RANGE(0, mask_w * mask_h))

    # Step 2: Create a mask of ones
    SET mask TO ONES(length, 3, mask_w, mask_h)
    mask = RESHAPE(mask, (length, 3, -1))  # Flatten the last two dimensions

    # Step 3: Apply the permutation to mask
    FOR j FROM 1 TO length - 1:
        mask[j:, :, order[j - 1]] = 0

    # Step 4: Reshape mask back to original dimensions
    mask = RESHAPE(mask, (length, 3, mask_w, mask_h))

    # Step 5: Optionally set the center pixel of the mask to 1
    IF static_center IS TRUE:
        mask[:, :, mask_w // 2, mask_h // 2] = 1

    # Step 6: Resize the mask to match the input image size
    mask = RESIZE(mask, [img_w, img_h], METHOD="NEAREST")

    RETURN mask, order
```

Next, I calculate the Shapley values based on frequency bias. First, I apply the Discrete Fourier Transform (DFT) to convert the images into the frequency domain. Then, I use the previously generated mask to select which specific frequencies should be included in the reconstruction. After that, I apply the Inverse Discrete Fourier Transform (IDFT) to convert the images back. Finally, I calculate the Shapley values, which can be understood as the expected accuracy based on these frequency components.

```
DEF Shapley_calculator(img, label, model, sample_times, mask_size, k=0):
    # Step 1: Get the dimensions of the image
    SET b, c, w, h TO img.shape

    # Step 2: Initialize Shapley value array and generate distribution list
    SET shap_value TO ZEROS(mask_size ** 2,)
    SET dis_dict, dis TO gen_dis_list(mask_size, mask_size)
```

```
    # Step 3: Loop through the sample times to calculate Shapley values
    FOR i FROM 0 TO sample_times - 1:
        # Sample a mask and its order
        SET mask, order TO sample_mask_dict(w, h, mask_size, mask_size, dis_dict, dis)

        # Transform the image to frequency domain
        SET base TO transform_fft(img[k])

        # Apply the mask in the frequency domain
        SET masked_img TO base * mask

        # Transform back to spatial domain
        SET masked_img TO transform_ifft(masked_img)

        # Get model's output
        SET output TO model(masked_img)

        # Get the output corresponding to the specific label
        SET y TO output[:, label[k]]
        SET dy TO y[:-1]- y[1:]

        # Step 4: Calculate Shapley value for each feature
        FOR i FROM 0 TO LENGTH(dy) - 1:
            SET key TO f"{dis[order[i]]:.2f}"
            shap_value[dis_dict[key]] += dy[i]

    # Step 5: Normalize the Shapley values by the number of samples
    shap_value /= sample_times

    # Return the calculated Shapley value
    RETURN shap_value
```

After obtaining the Shapley values, I then reconstruct the images by dividing them into NFC and PFC.

```
FOR class_name IN classes:
    # Step 1: Process each file in the directory
    FOR file IN files:
        SET freq TO fft2d(img)

        # Step 2: Reshape Shapley values
        SET mask_size TO freq_shap.size
        SET freq_shap TO (freq_shap, [1, mask_size, mask_size])

        # Step 3: Construct positive and negative masks
        SET mask_pos TO tf.cast(freq_shap > 0, tf.complex64)
        SET mask_neg TO tf.cast(freq_shap < 0, tf.complex64)

        # Step 4: Remove center frequency
        SET mask_pos TO (mask_pos, [[mask_pos.shape[1] // 2, mask_pos.shape[2] // 2]], [0]
        )
        SET mask_neg TO tf.tensor_scatter_nd_update(
            mask_neg, [[mask_neg.shape[1] // 2, mask_neg.shape[2] // 2]], [0]
        )

        # Step 5: Generate positive and negative frequency images
        SET pos_freq TO freq * mask_pos
        SET neg_freq TO freq * mask_neg

        SET pos_img TO tf.math.real(ifft2d(pos_freq))
        SET neg_img TO tf.math.real(ifft2d(neg_freq))

        # Step 6: Denormalize the images
        FOR c FROM 0 TO 2:
            pos_img[..., c] = pos_img[..., c] * std[c] + mean[c]
            neg_img[..., c] = neg_img[..., c] * std[c] + mean[c]

        # Step 7: Save the positive and negative frequency images
```

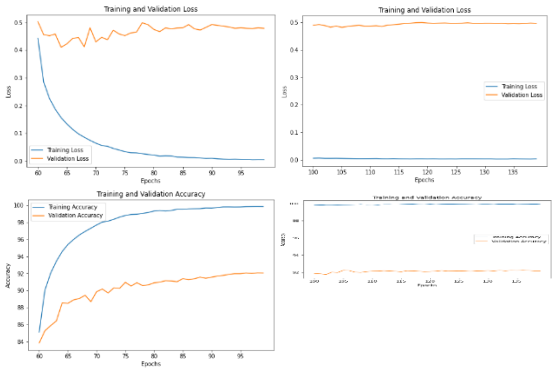
Then calculate class wise NFC：

```
# Step 1: Determine the minimum number of samples per class
FOR i FROM 0 TO num_classes - 1:

    # Step 2: Count the number of files containing "neg " in their name
    FOR f IN files:
        IF "neg " IN f:
            INCREMENT num by 1

    # Step 3: Update conf_per_class to the minimum number of "neg " files found
    IF num < conf_per_class:
        SET conf_per_class TO num

    # Step 4: process each file
    FOR f IN files:
        IF "neg" IN f AND num < conf_per_class:

            # Step 5: Append the image to the corresponding class in conf_set
            APPEND image TO conf_set[i]
            INCREMENT num by 1

# Step 6: Concatenate all class tensors into a single tensor
SET conf_set TO (conf_set, axis=0)
```

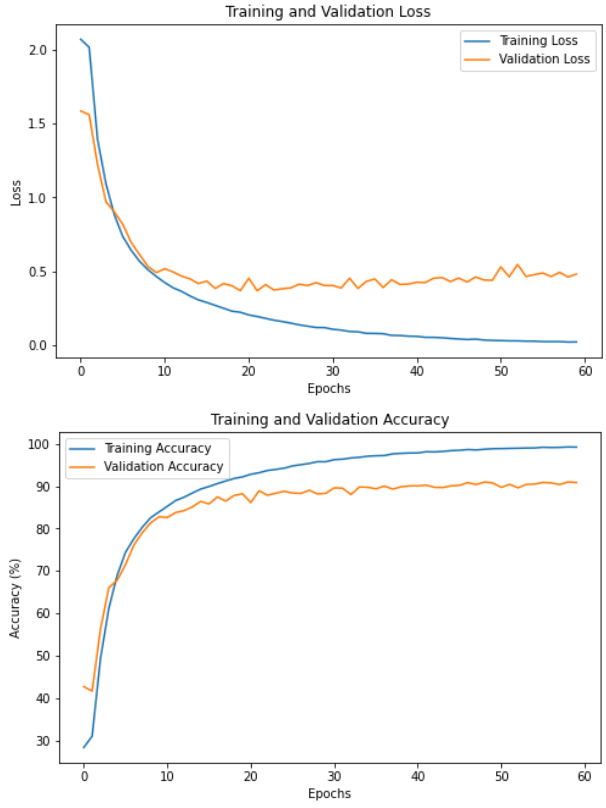
5. Results

5.1 Project Results

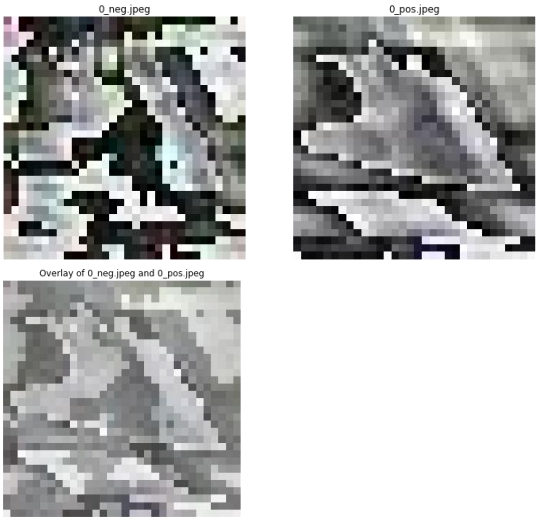
Here is the training result of ResNet18. I achieved a final accuracy of 92.18% using 140 epochs and approximately 140 minutes of training. Below are two graphs showing the loss and two graphs showing the accuracy, with the first 60 epochs missing due to a kernel crash.



Another complete and relatively accurate training session was conducted using 60 epochs, achieving a maximum accuracy of 91.0%.



Below I present the reconstructed images in NFC (Negative Frequency Components) and PFC (Positive Frequency Components), as well as the results after merging the reconstructed components.



Using PGD Adv Training for all models, I have shown the performance of model of purely ResNet18 and ResNet + CSA (called CSANet) with/without adv attack

Methods	**Clean Acc**	**Acc under PGD-10**	**Acc under PGD-20**
PGD-AT	**82.23**	68.62	66.80
PGD-AT+CSA	80.41	**70.21**	**69.48**

5.2 Comparison of the Results Between the Original Paper and Students' Project

First, I present a comparison of the final accuracy. It can be observed that my model performs better under the PGD-20 attack compared to the results reported in the paper, which is a positive outcome.

Methods		**Clean Acc**	**Acc under PGD-20**
PGD-AT	Orignal	**84.49**	46.38
	Student	82.23	**66.80**
PGD-AT+CSA	Orignal	**82.91**	49.42
	Student	80.41	**69.48**

Regarding training length/time, training/verification error, and test error, these details were not provided in the original paper, so I was unable to make direct comparisons.

Notably, despite using parameters roughly equivalent to those in the original paper, I trained for only 10 epochs, whereas the original work used 100 epochs.

5.3 Discussion / Insights Gained

I used the same dataset, the same Shapley values, and largely similar hyperparameters. The main difference lies in the number of training epochs. In the original paper, ResNet18 was trained for 160 epochs, while my maximum attempt was only 140 epochs. However, the original paper did not provide the accuracy of their trained model, so I assume it was quite high. As for CSANet, due to the extensive time required for adversarial training and attacks, I only trained for 10 epochs, which took three hours, whereas the original paper used 100 epochs. The paper only mentioned the time required to compute Shapley values, ranging from a few minutes to an hour, but in my case, the computation was so intensive that it caused kernel crashes. This makes me curious about how they managed 100 epochs of training time.

Additionally, I suspect that there might be differences in the specific PGD attack and training setup I used compared to theirs. While my accuracy during PGD adversarial training was similar to theirs, my results under the PGD-20 attack outperformed the original paper. I am very curious about the reasons behind this.

I gained a tremendous amount of insight into replicating someone else's work. This was actually my first attempt at replication, and I am relatively satisfied with my results. The most important lesson I learned is to avoid directly looking at someone else's code, as it can create preconceived notions and hinder your own thinking. Secondly, during replication, it is essential to repeatedly revisit the original paper. I often discovered details I had previously overlooked and quickly identified when I was heading in the wrong direction.

After reproducing their results, I found that I gained a deeper understanding of the overall structure of the paper. I believe this experience was incredibly meaningful.

6. Future Work

First, as discussed with the TA in my proposal, initially I tried to experiment with higher-resolution datasets. However, the computation required to generate stable Shapley values for such datasets grows exponentially, making it infeasible to attempt this idea.

Additionally, the original reason I considered this paper was my interest in developing a more rigorous Class-wise Shapley Value method[5]. However, due to limited computational resources, I was unable to implement this as well.

Third, the paper mentioned adversarial attacks specifically targeting low-frequency components (LFC). Although I reviewed the relevant materials, implementing this in practice requires substantial additional debugging, so I have postponed it for future plans.

Fourth, the original paper suggested extending this approach to other backbones, such as Transformers and Graph Neural Networks (GNNs). I am particularly curious about the performance on Transformers because, to my understanding, their connection to frequency components is limited. Additionally, such experiments would require significant computational resources. These are all directions I hope to discuss further or leave for future work. Finally, I believe my higher accuracy under adversarial attacks compared to the original work, despite fewer epochs, suggests potential for further improvements. I am curious if making my computational process more efficient and training for more epochs could lead to even better results.

7. Conclusion

This project successfully replicates most of the core findings of the paper *Rethinking and Improving Robustness of Convolutional Neural Networks: A Shapley Value-based Approach in the Frequency Domain.* The project essentially achieves most of the main original goals outlined in the proposal, including building the ResNet18 model and implementing the augmentation method of CSA. This method leverages frequency components with negative contributions from the same class for data augmentation to enhance the robustness of CNNs. Notably, under the PGD-20 adversarial attack, this project achieves higher accuracy than reported in the original paper. A straightforward direction for future work is to explore the effectiveness of CSA on Graph Neural Networks (GNNs). I am willing to consider this as a topic for future research. An obvious direction for future improvement is to make the model more computationally efficient and to train it for more epochs under adversarial attacks.

6. Acknowledgement

First, I would like to thank all the TAs of 4040, especially the TA who always responded to my questions on Ed Discussion incredibly quickly and the TA who helped me during the Friday night office hours! You have helped me so much!! I also want to thank the TA who held recitation before exams and two classmates shared of their cats. So sweet. Honestly, as a student from the statistics department, most of my programming experience has been self-reliant debugging. Having someone to answer my questions is truly amazing.

I must thank the developed online communities, especially CSDN, GitHub, and all the Google links that pointed me toward solutions for my debugging issues.

Of course, I have to express my gratitude to the professor of this course for the excellent teaching and course design. I am genuinely impressed by how the professor created

assignments that allowed me to quickly get hands-on with deep learning. I also want to thank the slip days and the extra credits for assignments. This course has been an incredible experience.

Also, thanks to my computer for not crashing.

7. References

[1]ecbme4040/e4040-2024fall-project-MYZH-mz3088: e4040-fall2024-e4040-2024fall-project-e4040-2024Fall-project created by GitHub Classroom
[2] Chen, Y., Ren, Q. and Yan, J., 2022. Rethinking and improving robustness of convolutional neural networks: a shapley value-based approach in frequency domain. Advances in neural information processing systems, 35, pp.324-337.
[3] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
[4] https://paperswithcode.com/dataset/cifar-10
[5] Schoch, S., Xu, H. and Ji, Y., 2022. CS-Shapley: class-wise Shapley values for data valuation in classification. Advances in Neural Information Processing Systems, 35, pp.34574-34585.
[6] Madry, A., 2017. Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083.

8. Appendix

8.1 Individual Student Contributions in Fractions

	mz3088
Last Name	Zhang
Fraction of (useful) total contribution	1
What I did 1	Built and Trained ResNet18, achieving high acc rate
What I did 2	Computed Shapley values and reconstructed the images
What I did 3	Built and Trained CSANet, and tested it with/without adversarial training/attack