CS-6304 Assignment 1

Abdulhaseeb Khan : 25100077, Aazen Saleem : 25100031, Ahsan Mir : 25100325

September 30, 2024

Introduction

In the field of computer vision, understanding the underlying biases and generalization capabilities of machine learning models is paramount. This project explores the performance of three distinct models: two discriminative models, VGG-19 and Vision Transformer (ViT-16), alongside one contrastive model, CLIP. Each model was fine-tuned on the CIFAR-10 dataset, which serves as a benchmark for image classification tasks.

To evaluate the models' capabilities, we examine variations of the CIFAR-10 dataset to identify different locality and semantic biases. Following this, we assessed their performance on an independent and identically distributed (IID) dataset. The project further investigates domain generalization by fine-tuning the models on the more complex CIFAR-100 dataset and the PACS dataset, providing insights into how well these models adapt to new data distributions and maintain performance across varying domains. Through this comprehensive analysis, we aim to contribute valuable knowledge towards the understanding of model biases and their implications in real-world applications.

The datasets used in this project are as follows:

- **CIFAR-10**: https://www.kaggle.com/c/cifar-10/
- **CIFAR-100**: https://www.kaggle.com/datasets/fedesoriano/cifar100
- **PACS**: https://www.kaggle.com/datasets/nickfratto/pacs-dataset
- **MNIST**: https://www.kaggle.com/datasets/hojjatk/mnist-dataset
- **Colourized MNIST**: https://github.com/jayaneetha/colorized-MNIST.git

Additionally, several other variations of CIFAR-10 were used throughout the analysis.
task 1 methodology:

Methodology

Text-to-Image Generative Model as Zero-Shot Classifier

In this task, we implemented a zero-shot image classification algorithm using a pre-trained text-to-image generative model based on latent diffusion, specifically Stable Diffusion v1-4. The goal was to leverage the generative capabilities of the diffusion model to perform classification without any additional training, by assessing how well the model could reconstruct an input image when conditioned on different class prompts. The system predicted the class of an input image by comparing reconstruction losses across various class-conditioned generations.

The core components of the system were derived from the Stable Diffusion v1-4 model, which consisted of the following modules:

- **Variational Autoencoder (VAE):** Encoded input images into a latent space and decoded latent representations back to image space.

- **UNet Model:** Acted as the denoising network in the diffusion process, predicting the noise residual at each timestep.
- **Text Encoder (CLIP Text Encoder):** Converted textual prompts into embeddings to condition the UNet during the reverse diffusion process.
- **Scheduler:** Managed the forward (noising) and reverse (denoising) diffusion processes.

All components were initialized with pretrained weights from Stable Diffusion v1-4. The model operated in half-precision (`float16`) to optimize memory usage and computational efficiency.

The system performed zero-shot classification on input images by following these key steps:

1. **Data Preprocessing:** Input images were resized to 512×512 pixels and normalized to the $[-1, 1]$ range to match the VAE's expected input format.
2. **Class Prompts:** A list of class names was converted into textual prompts in the form of 'A photo of a {class}", aligning with the training data format of the Stable Diffusion model.
3. **Assumptions:**
   - The input images were sufficiently represented within the training distribution of the Stable Diffusion model.
   - The textual prompts effectively captured the semantic meaning of each class.
   - The choice of prompt format 'A photo of a {class}" was assumed to be effective in eliciting the correct conditional generations from the model.
   - The latent space representations produced by the VAE were assumed to be compatible with the diffusion model's reverse process and could generalize to reconstruct input images when conditioned on different unseen class prompts.

The classification process involved simulating the forward and reverse diffusion processes for each class prompt and computing reconstruction losses to determine the most probable class:

1. **Encoding the Input Image**
   - The preprocessed input image $\mathbf{x}$ was encoded into the latent space using the VAE encoder:
   $$\mathbf{z} = \text{VAEEncoder}(\mathbf{x})$$
   - The latent representation was scaled using the VAE's scaling factor γ:
   $$\mathbf{z} \leftarrow \gamma \mathbf{z}$$
2. **Generating Text Embeddings**
   - Each class prompt was tokenized using the tokenizer associated with the CLIP text encoder.
   - The tokenized prompts were converted into text embeddings $\{\phi(y_i)\}$ using the CLIP text encoder:
   $$\phi(y_i) = \text{TextEncoder}(\text{Tokenizer}(\text{'A photo of a } y_i\text{"}))$$
3. **Forward Diffusion (Noising Process)**
   - A random timestep t was sampled uniformly from $\{0, 1, \ldots, T-1\}$, where T was the total number of diffusion steps.
   - Noise was added to the latent representation to obtain the noised latent $\mathbf{z}_t$:
   $$\mathbf{z}_t = \sqrt{\bar{\alpha}_t}\mathbf{z} + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$
   where $\bar{\alpha}_t$ was the cumulative product of α_t from the scheduler, and $\boldsymbol{\epsilon}$ was standard Gaussian noise.

4. **Reverse Diffusion (Denoising Process) and Reconstruction**

   - For each class y_i, the UNet model predicted the noise residual conditioned on the text embedding $\phi(y_i)$:
   $$\hat{\boldsymbol{\epsilon}}_\theta = \text{UNet}(\mathbf{z}_t, t, \phi(y_i))$$
   - The latent representation was reconstructed:
   $$\tilde{\mathbf{z}}0 = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(\mathbf{z}_t - \sqrt{1-\bar{\alpha}_t}\hat{\boldsymbol{\epsilon}}\theta\right)$$

5. **Computing the Weighted Reconstruction Loss**

   - The reconstruction loss for each class was computed as the mean squared error (MSE) between the original latent $\mathbf{z}$ and the reconstructed latent $\tilde{\mathbf{z}}_0$:
   $$L_i = \|\mathbf{z} - \tilde{\mathbf{z}}_0\|_2^2$$
   - A weighting function w_t was applied to emphasize certain timesteps:
   $$w_t = \exp\left(-7\frac{t}{T}\right)$$
   - The weighted loss was calculated:
   $$\tilde{L}_i = w_t L_i$$

6. **Iterative Scoring and Class Pruning**

   - The above steps were repeated for a specified number of iterations or until only one class remained after pruning.
   - After each iteration, classes were evaluated based on their accumulated weighted losses.
   - A paired t-test was performed between the class with the lowest mean loss and the other classes to determine if any classes could be pruned:
   $$p\text{-value} = \text{ttest}(\{\tilde{L}\text{best}\}, \{\tilde{L}_i\})$$
   - Classes with a p-value below a predefined threshold (e.g., 0.05) were pruned from consideration.

7. **Final Prediction**

   - The class with the lowest mean weighted reconstruction loss after all iterations was selected as the predicted category:
   $$\hat{y} = \arg\min_{y_i} \mathbb{E}[\tilde{L}_i]$$

Task 2: Model Evaluation on CIFAR-10

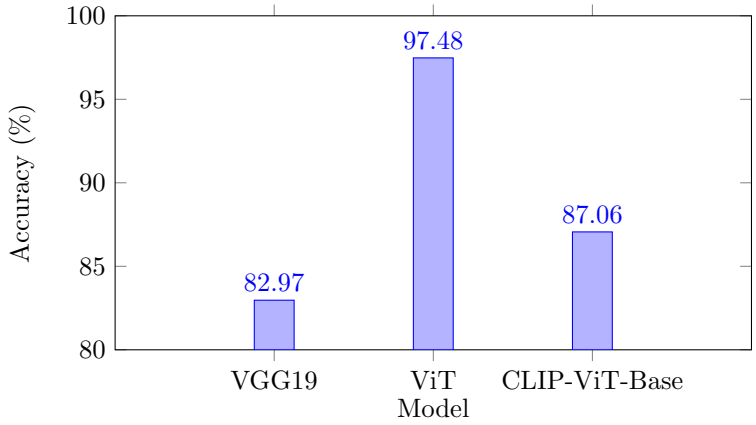
In Task 2, we focused on evaluating our models on a standard, general-purpose dataset, CIFAR-10. The classification accuracy obtained from this evaluation serves as a benchmark for comparison in future tasks. This task allows us to assess the performance of the pretrained models we are using—VGG19, ViT, and CLIP-ViT-Base—under familiar conditions.

In terms of methodology and implementation, the discriminative models followed a similar approach. First, we defined the model and applied necessary modifications before fine-tuning it on CIFAR-10. Specifically, we froze the feature extractor (the backbone) and replaced the classifier head with a new one suited to the number of classes in the dataset (in this case, 10). We then loaded the CIFAR-10 training set directly from `torchvision.datasets.CIFAR10` and fine-tuned the discriminative models for 3 epochs. After

fine-tuning, we evaluated the models on the CIFAR-10 test set to assess their classification accuracy.

By contrast, the CLIP-ViT-Base model, which uses a contrastive learning approach, required a different methodology. We employed it for zero-shot image classification, meaning there was no need for a train-test split or model fine-tuning. First, we loaded the test set using `torchvision.datasets.CIFAR10`. Then, we listed all class names from the CIFAR-10 dataset and used them to define text inputs. In zero-shot classification, visual features are mapped into a semantic space defined by word embeddings, so we measured cosine similarity between the image and text features. The predicted label is the one with the highest cosine similarity. This process iterates over the entire dataset, predicting labels for each image and reporting accuracy figures.

Overall, all three models achieved strong performance on the CIFAR-10 dataset. The ViT model achieved an accuracy of 97.48%, while the VGG19 model reached 82.97%, and the contrastive model, CLIP-ViT-Base, achieved an accuracy of 87.06%.



Task 3: Out-of-Domain Model Evaluation

Task 3 focused on evaluating how well our models perform on out-of-domain data. For this evaluation, we utilized two distinct datasets: the PACS dataset and the CIFAR-100 dataset. PACS represents a covariate shift in the data, while CIFAR-100 introduces a semantic shift.

First, let's explore the PACS dataset. As with previous tasks, for the discriminative models, we froze the feature extractor backbone and replaced the classifier head with one suitable for the dataset, which in this case has 7 classes. To streamline the process, we used the DeepLake library along with a custom wrapper to load the PACS dataset without downloading it locally. After loading the training dataset, we trained our models—VGG19 and ViT—for 3 epochs, following a similar approach to Task 2. It's important to note that training was only applied to the discriminative models.

For the contrastive model, CLIP-ViT-Base, no fine-tuning was performed, nor did we replace the classifier head or modify the model. Instead, we directly loaded the CLIP-ViT-Base model, loaded the test dataset, and applied zero-shot image classification to obtain accuracy results.

The resulting accuracies on the PACS dataset were as follows:

Next, we evaluated the models on the CIFAR-100 dataset. The methodology for the discriminative models was similar: we froze the feature extractor backbone and replaced the classifier head with one tailored for CIFAR-100, which contains 100 classes. We loaded the training set using `torchvision.datasets.CIFAR100` and fine-tuned the models for 3 epochs. Afterward, we evaluated the models on the CIFAR-100 test set to assess their performance on out-of-domain data, specifically in the context of semantic shift.

Model	Accuracy (%)
CLIP-ViT-Base	81.34
VGG19	78.01
ViT	91.03

Table 1: Accuracy results on PACS dataset

For the contrastive model, CLIP-ViT-Base, we followed the same zero-shot classification approach as with PACS, where no fine-tuning was required and the test set was used directly for evaluation.

The resulting accuracies on the CIFAR-100 dataset were as follows:

Model	Accuracy (%)
CLIP-ViT-Base	59.40
VGG19	60.01
ViT	86.85

Table 2: Accuracy results on CIFAR-100 dataset

Task 4: Semantic Biases of Models

This task explores the semantic biases of our models, specifically focusing on shape, texture, and color biases. We aim to investigate how each model may exhibit an over-reliance on certain visual features over others. This analysis was conducted on all three models: CLIP-ViT-Base, VGG19, and ViT.

Shape Bias

To evaluate shape bias, we utilized the CIFAR-10G dataset, a modified version of the CIFAR-10 dataset where color and texture information were removed, leaving only shape-based features. The models' performances were measured on this dataset, and the shape bias was calculated using the following formula:

$$\text{Shape Bias} = \frac{\text{Shape Accuracy}}{\text{Total Accuracy}}$$

Here, the total accuracy refers to the model's accuracy on the original CIFAR-10 dataset, while the shape accuracy is the accuracy achieved on CIFAR-10G. The results for shape bias across the models were as follows:

- CLIP-ViT-Base: 0.47
- VGG19: 0.33
- ViT: 0.18

Texture Bias

For texture bias, we created a modified version of CIFAR-10G by applying elastic transformations, which distort shapes and enhance texture features while removing color. This allowed us to assess how each model responds to texture-dominant inputs. The texture bias was calculated as:

$$\text{Texture Bias} = \frac{\text{Texture Accuracy}}{\text{Total Accuracy}}$$

Again, total accuracy refers to performance on the original CIFAR-10 dataset, while texture accuracy is derived from the model's performance on the transformed dataset. The results for texture bias were:

- CLIP-ViT-Base: 0.42
- VGG19: 0.45
- ViT: 0.17

Color Bias

To assess color bias, we utilized two separate datasets: the MNIST dataset (grayscale) and the Colorized MNIST dataset. The color bias was computed as follows:

$$\text{Color Bias} = \frac{\text{Color Accuracy}}{\text{Total Accuracy}}$$

We first evaluated the models on the MNIST dataset to obtain the total accuracy, followed by an evaluation on the Colorized MNIST dataset to measure color accuracy. The resulting color biases were:

- CLIP-ViT-Base: 0.79
- VGG19: 0.41
- ViT: 0.59

These analyses provide a comprehensive understanding of the shape, texture, and color biases present in each model, offering insights into their reliance on specific visual cues.

Bias	**CLIP-ViT-Base**	**VGG19**	**ViT**
Shape Bias	0.47	0.33	0.18
Texture Bias	0.42	0.45	0.17
Color Bias	0.79	0.41	0.59

Table 3: Bias values for each model

Task 5: Inductive Biases of Models: Locality Biases

In our exploration of locality biases in neural networks, we first conducted an experiment on the CIFAR-10 dataset to evaluate how localized noise impacts model performance. By injecting Gaussian noise into an 8x8 patch of each image, while maintaining consistency across all images, we aimed to assess the robustness of our fine-tuned models. The results indicated a significant drop in accuracy, with the Vision Transformer (ViT) model experiencing a 79.7% decrease, the VGG model a 16.5% drop, and CLIP showing a drop in accuracy from 87.06% to 73.51%. This highlights VGG's better resistance to localized noise compared to the other models.

In another analysis, we introduced 4x4 patch scrambling to the CIFAR-10 test set to disrupt the global structure of the images. This manipulation caused a considerable performance decline, with ViT showing an 83.96% drop in accuracy, VGG a 62.42% drop, and CLIP falling from 87.06% to 18.85%.

Lastly, we applied the artistic style of Van Gogh's *Starry Night* to a random selection of 100 CIFAR-10 images, utilizing the VGG19 model for feature extraction and optimization. This global style transfer also resulted in accuracy drops, with VGG decreasing by 63.97% and ViT only by 22.45%, suggesting that ViT performed better under conditions of global style change.

Task 6: Combining Convolution and Self-attention

In this task, we explored various operations that combine convolution and attention mechanisms to enhance feature extraction in deep learning models. We implemented six different operations using a randomly sampled 32×32 image of a horse from the CIFAR-10 dataset. Each operation was designed as a single-layer model to isolate and understand its effect on feature extraction. Below, we detail the implementation of each operation, including the mathematical formulations and the specific steps taken.

1. **Depthwise Convolution**

   Depthwise convolution operates on each input channel independently, applying a separate convolutional filter to each channel without mixing information across channels. The mathematical formulation for the depthwise convolution at position i is:

   $$y_i = \sum_{j \in \mathcal{L}(i)} W_{i-j} \odot X_j$$

   - y_i: Output feature at position i.
   - X_j: Input feature at position j.
   - W_{i-j}: Convolutional kernel weight corresponding to the relative position $i - j$.
   - $\mathcal{L}(i)$: Local neighborhood around position i (defined by the kernel size).
   - $\odot$: Element-wise multiplication.

   Implementation Steps:

   (a) Identified the local neighborhood $\mathcal{L}(i)$ for each position i based on the kernel size.

   (b) Applied a separate convolutional kernel to each channel independently.

   (c) For each position i, performed element-wise multiplication of W_{i-j} with X_j within the local neighborhood.

   (d) Summed the results over the local neighborhood to obtain y_i.

   (e) Assumed zero-padding to handle border effects.

2. **Self-Attention**

   Self-attention captures global dependencies by computing interactions between all pairs of positions in the input. Assuming $Q = K = V = X$, with no learnable parameters, the operation is defined as:

   $$y_i = \sum_{j \in \mathcal{G}} \frac{\exp(x_i^\top x_j)}{\sum_{k \in \mathcal{G}} \exp(x_i^\top x_k)} x_j$$

   - y_i: Output feature at position i.
   - x_i, x_j: Input feature vectors at positions i and j.
   - $\mathcal{G}$: Set of all positions in the input (global context).

   Implementation Steps:

   (a) Flattened the spatial dimensions to compute dot products between all pairs (i, j).

   (b) Computed similarity scores $x_i^\top x_j$ for all positions.

   (c) Exponentiated the similarity scores to obtain unnormalized attention weights.

   (d) Normalized the attention weights using softmax over all positions.

   (e) Multiplied the attention weights by the corresponding x_j and summed over all j to get y_i.

3. **Post-Normalization Combination**

   This operation integrates convolutional weights after the attention normalization step, enhancing local features while capturing global dependencies. It is formulated as:

$$y_i^{\text{post}} = \sum_{j \in \mathcal{G}} \left(\frac{\exp(x_i^\top x_j)}{\sum_{k \in \mathcal{G}} \exp(x_i^\top x_k)} + W_{i-j} \right) x_j$$

   Implementation Steps:

   (a) Computed attention weights over the global context using self-attention.

   (b) Averaged convolutional weights W_{i-j} over channels to obtain scalar values.

   (c) Added W_{i-j} to the normalized attention weights.

   (d) Multiplied the combined weights by x_j and summed over all j to obtain y_i^{post}.

4. **Pre-Normalization Combination**

   This operation integrates convolutional weights before the attention normalization step, influencing the attention distribution. The formulation is:

$$y_i^{\text{pre}} = \sum_{j \in \mathcal{G}} \frac{\exp(x_i^\top x_j + W_{i-j})}{\sum_{k \in \mathcal{G}} \exp(x_i^\top x_k + W_{i-k})} x_j$$

   Implementation Steps:

   (a) Computed attention scores $s_{ij} = x_i^\top x_j + W_{i-j}$ over the global context.

   (b) Applied softmax normalization to obtain attention weights a_{ij}.

   (c) Multiplied the attention weights by x_j and summed over all j to get y_i^{pre}.

5. **Attention Modulated Convolution**

   This operation combines convolution and attention within a local neighborhood defined by the convolutional kernel. The operation is defined as:

$$Y(i) = \sum_{j \in \mathcal{L}(i)} A(i,j) \odot X(j) \odot W(i-j)$$

   - $A(i,j)$: Attention weight computed as:

$$A(i,j) = \frac{\exp(Q(i) \cdot K(j))}{\sum_{k \in \mathcal{L}(i)} \exp(Q(i) \cdot K(k))}$$

   - $Q(i)$, $K(j)$: Query and key vectors derived from $X(i)$ and $X(j)$, respectively.

   Implementation Steps:

   (a) Applied learnable linear transformations to compute $Q(i)$ and $K(j)$.

   (b) For each position i, identified the local neighborhood $\mathcal{L}(i)$.

   (c) Computed attention scores within the local neighborhood.

   (d) Applied softmax normalization over $\mathcal{L}(i)$ to get $A(i,j)$.

   (e) Performed element-wise multiplication of $X(j)$ and $W(i-j)$.

   (f) Multiplied the result by $A(i,j)$ and summed over j to obtain $Y(i)$.

6. **Convolution Modulated Attention**

   This operation modulates the attention mechanism using convolutional outputs before computing the attention weights. It is formulated as:

$$\text{Attention}(X) = \text{Softmax}(\text{DWConv}_{F\times F}(X, W)) \odot V$$

   - $\text{DWConv}_{F\times F}(X, W)$: Depth-wise convolution of X with kernel W.
   - V: Value tensor, possibly a transformation of X.

   Implementation Steps:

   (a) Applied depth-wise convolution to X to compute attention scores.
   (b) Applied softmax over spatial dimensions to obtain attention weights.
   (c) Computed value tensor V using a learnable transformation.
   (d) Multiplied attention weights by V to obtain the output.

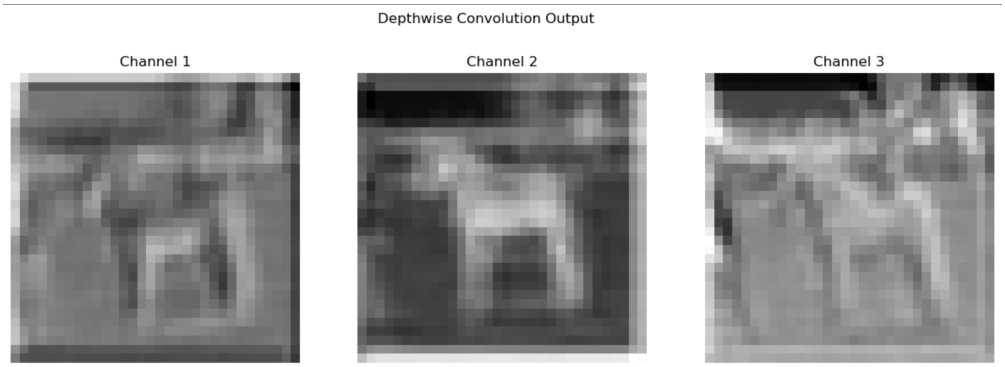
Results

For each operation, we processed the sample image and visualized the resulting feature maps and attention maps to analyze the effects on feature extraction. The original image sampled from CIFAR-10 is shown below.

1. **Depthwise Convolution**

   The feature maps showed that each channel emphasized different local patterns, such as edges or textures, due to the independent convolutional kernels. Information was propagated locally within each channel, with each output pixel depending on its local neighborhood.

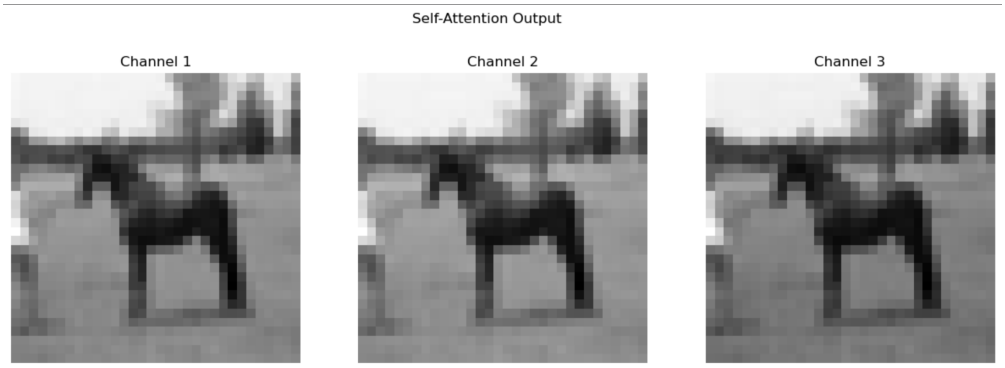
   Feature Maps:



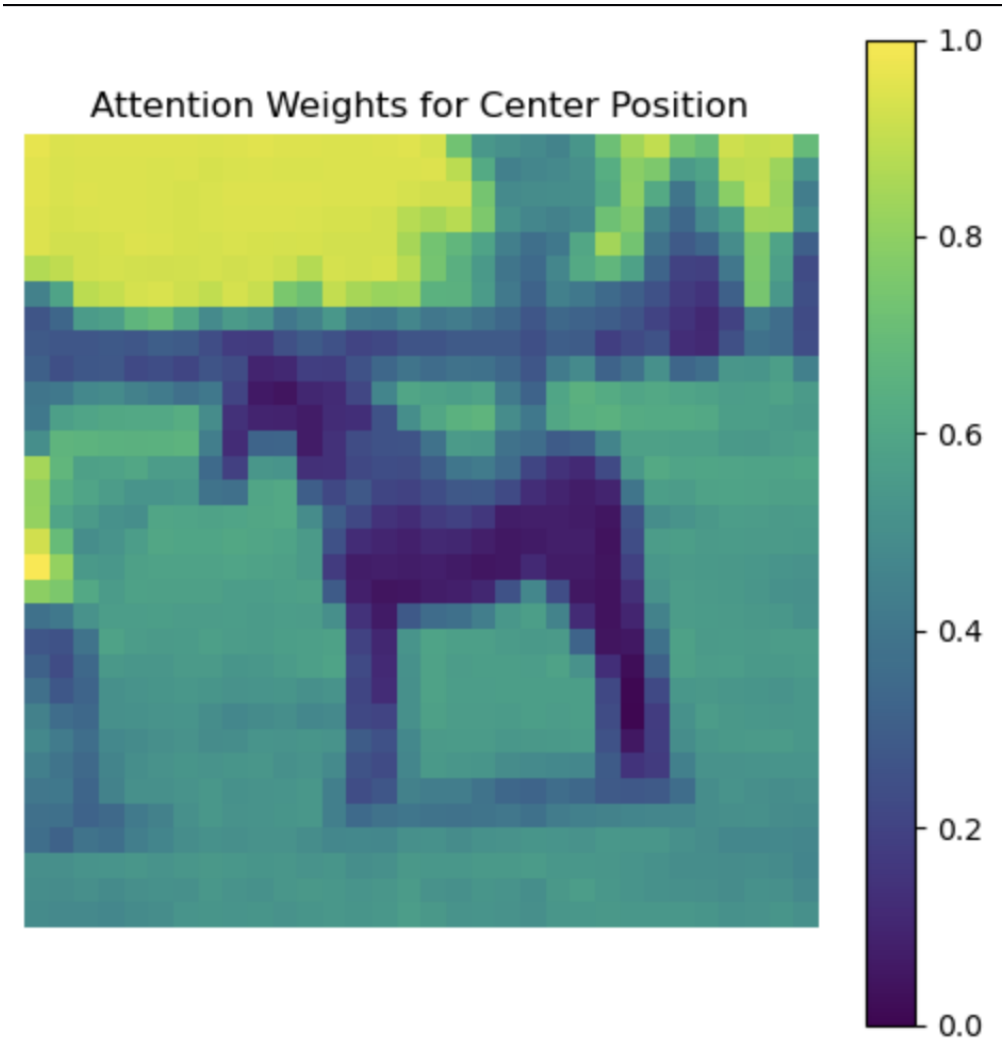
2. **Self-Attention**

   The self-attention mechanism captured global dependencies, allowing each output position to attend to all input positions. The attention maps indicated how different regions influenced each other based on feature similarity.

   Feature Maps:



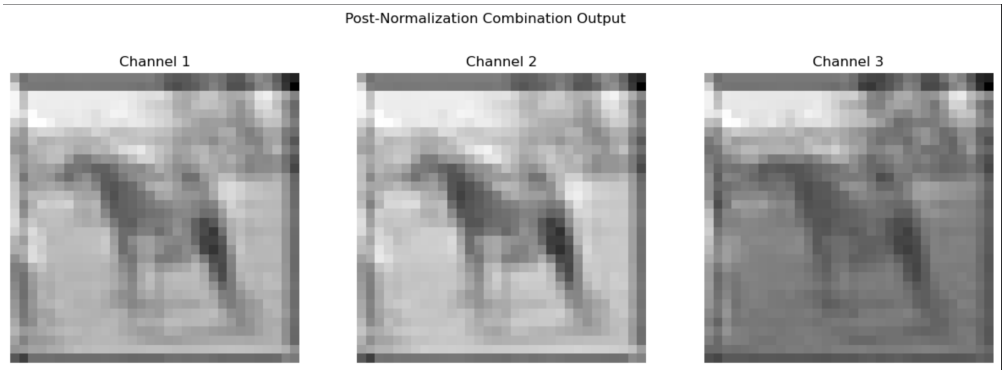
   Attention Map for Center Position:



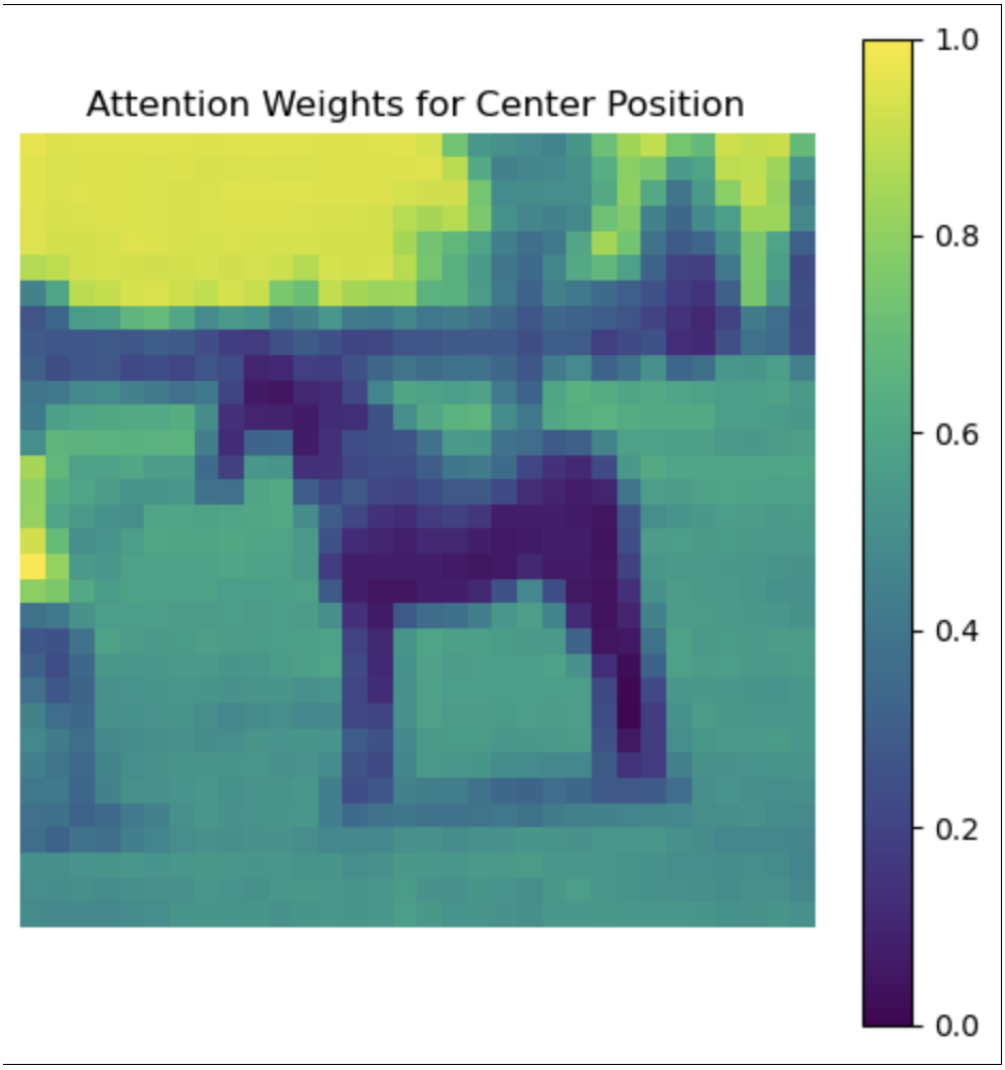
3. **Post-Normalization Combination**

   The feature maps demonstrated a balance between global context and enhanced local features. The attention weights, combined with convolutional weights after normalization, allowed for reinforced local patterns while maintaining global awareness.

   Feature Maps:



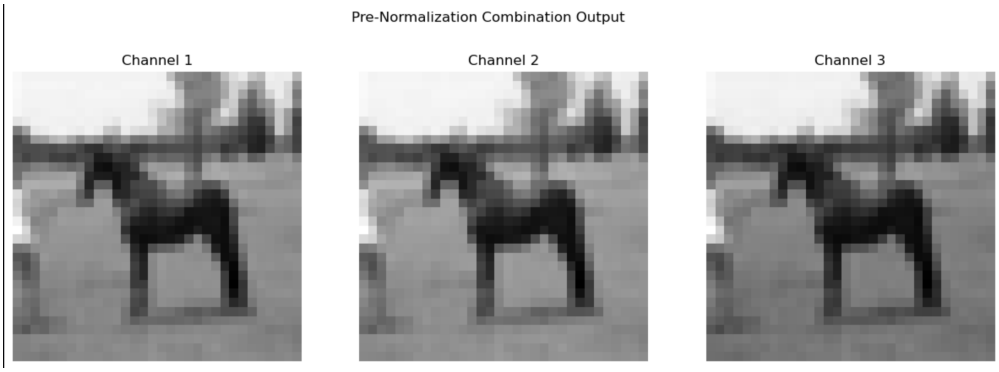
   Attention Map for Center Position:



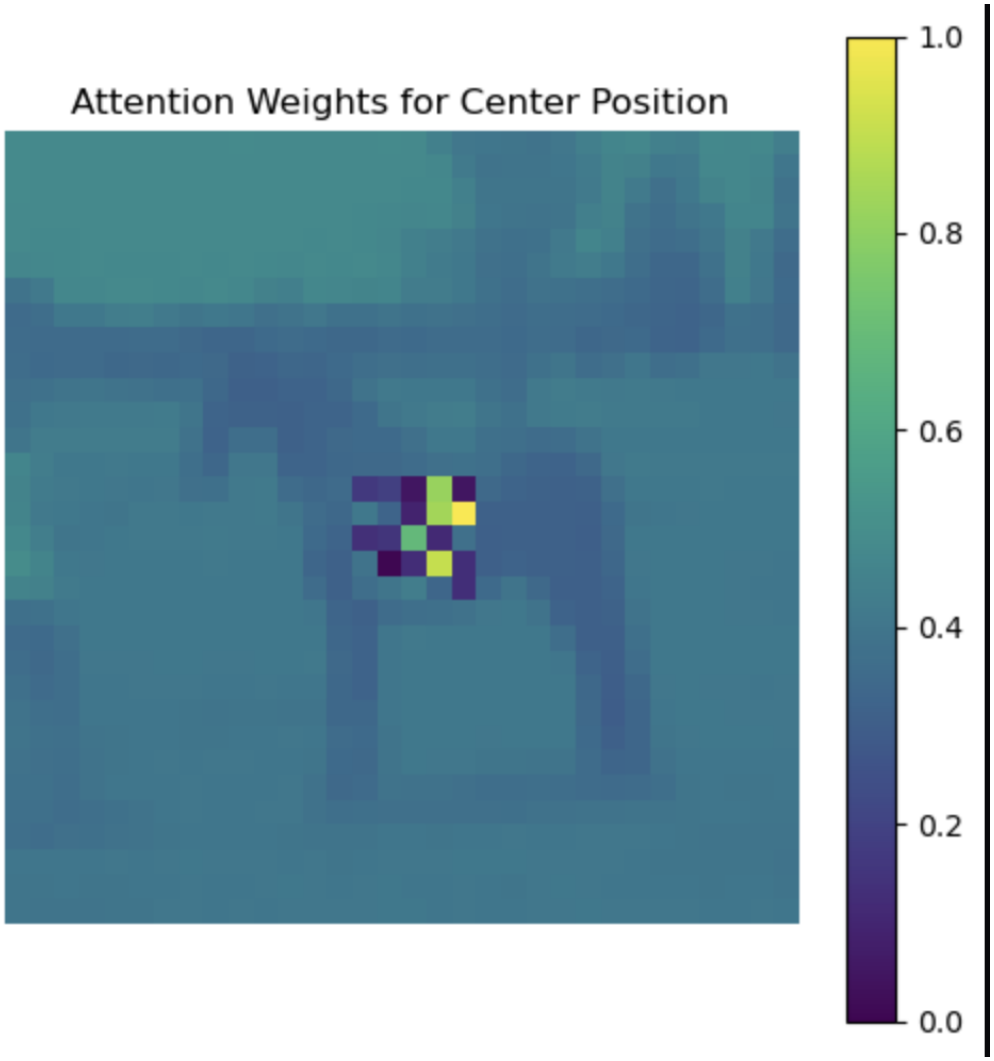
4. **Pre-Normalization Combination**

   The feature maps exhibited sharper local details, as convolutional weights influenced the attention distribution before normalization. The attention maps showed a stronger focus on specific parts of the image, highlighting important features more prominently.

   Feature Maps:



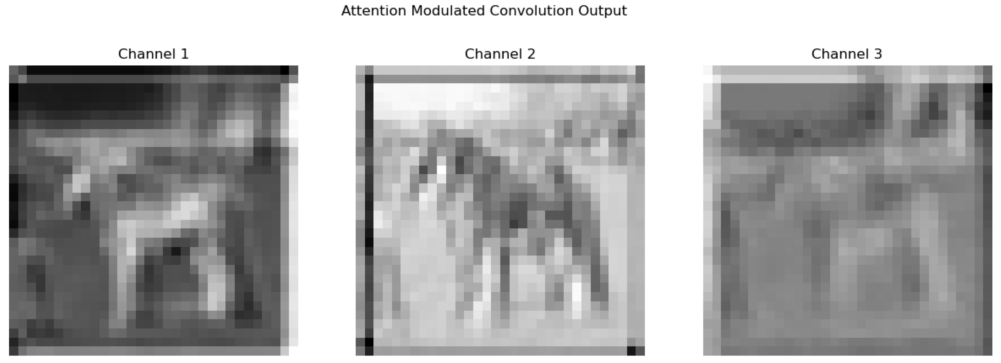
   Attention Map for Center Position:



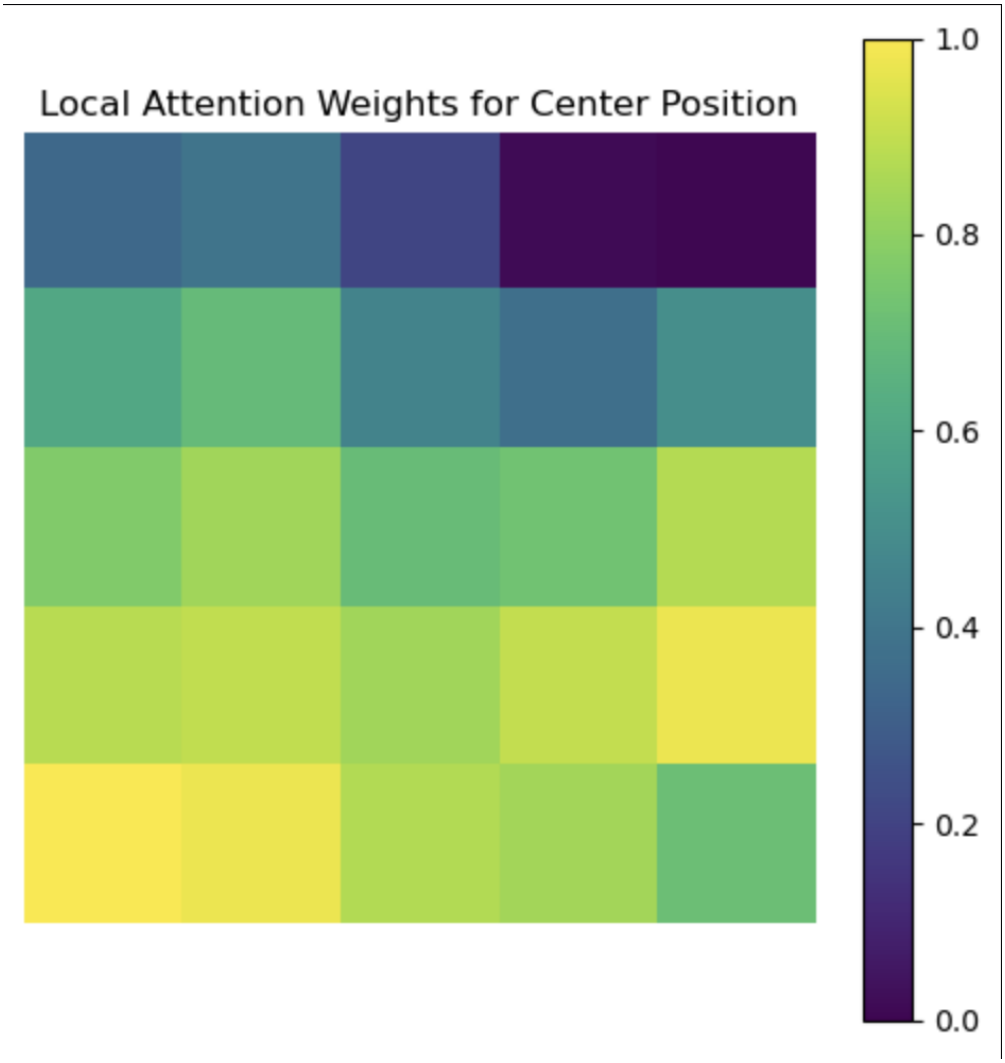
5. **Attention Modulated Convolution**

   The operation emphasized important features within the local neighborhood, as shown in the feature maps. The attention mechanism allowed for dynamic weighting of local features, enhancing relevant patterns.

   Feature Maps:



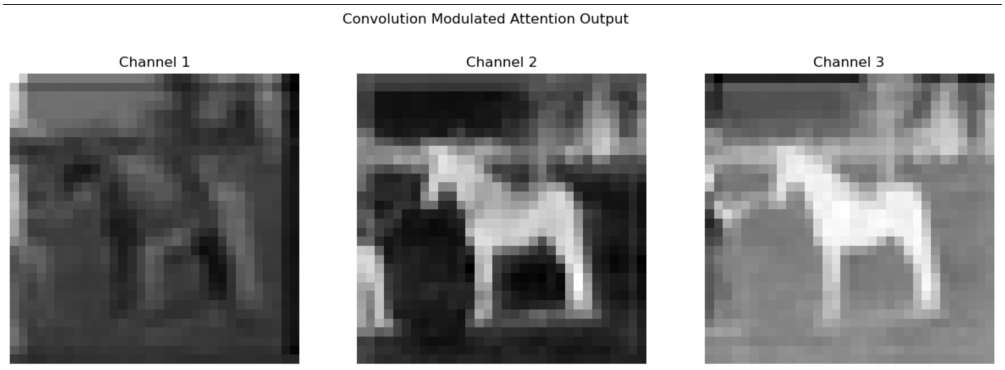
   Attention Map for Center Position:



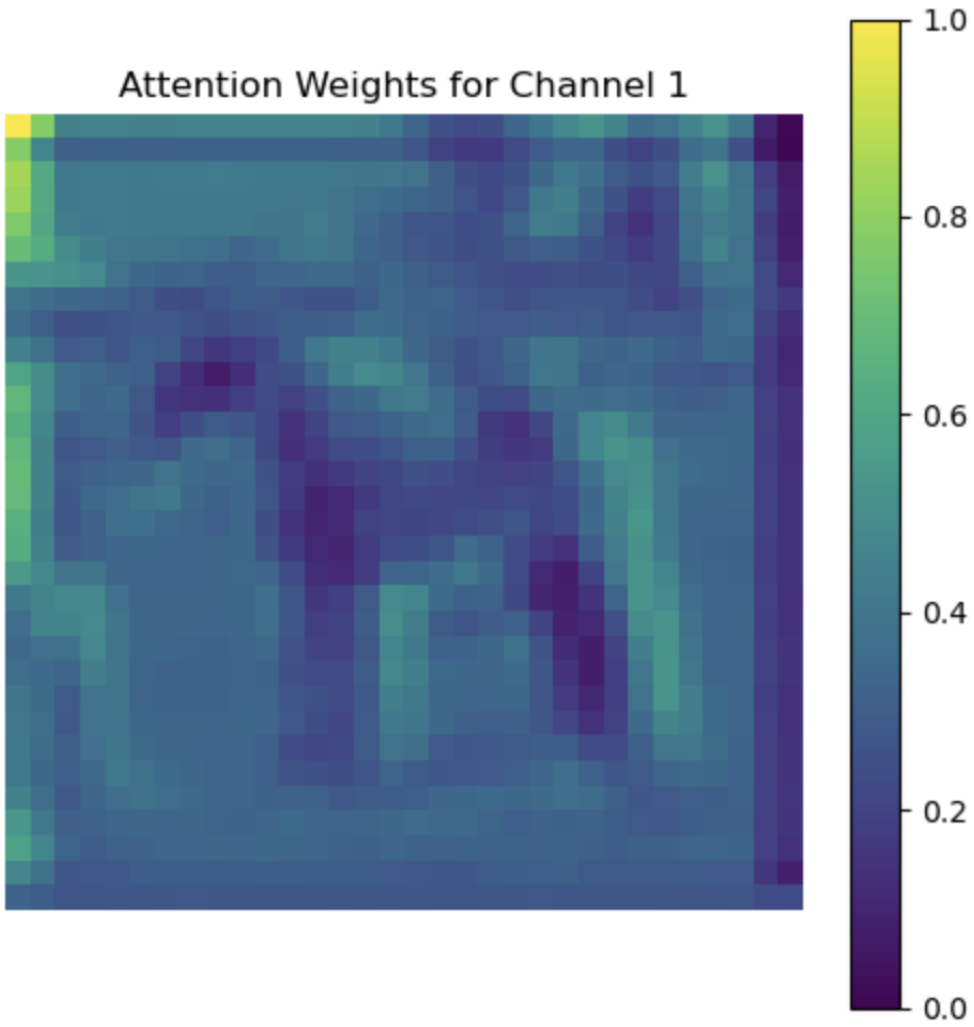
6. **Convolution Modulated Attention**

   The feature maps highlighted how attention weights modulated the input features, enhancing significant regions. The attention maps revealed spatial locations with higher importance within each channel.

   Feature Maps:



   Attention Map for Center Position:



Discussion

By integrating convolution and attention mechanisms, we enhanced feature extraction capabilities, balancing local detail preservation with global context understanding. Operations that combined global attention with local convolutional features, such as the post-normalization and pre-normalization combinations, demonstrated improved feature representation. The attention modulated convolution and convolution modulated attention highlighted the importance of focusing on significant features within local neighborhoods. Moreover, experimenting with different operations demonstrated how different combinations of convolution and attention mechanisms affect feature extraction, information propagation, and receptive fields. We outline these differences observed as follows,

Information Propagation:

- **Depthwise Convolution**: Information propagated locally within each channel, capturing spatial features without cross-channel interaction.
- **Self-Attention**: Enabled global information propagation, with each position attending to all others based on feature similarity.
- **Post-Normalization Combination**: Combined global attention with enhanced local features by adding convolutional weights after normalization.

- **Pre-Normalization Combination**: Influenced attention distribution towards local features by adding convolutional weights before normalization.
- **Attention Modulated Convolution**: Propagated information within the local neighborhood, focusing on important local features through attention modulation.
- **Convolution Modulated Attention**: Emphasized local feature extraction while highlighting significant spatial locations via attention weights.

Receptive Field:

- **Depthwise Convolution**: Limited to the kernel size, affecting local spatial patterns.
- **Self-Attention**: Global receptive field, capturing long-range dependencies.
- **Post-Normalization Combination**: Maintained a global receptive field with added local emphasis.
- **Pre-Normalization Combination**: Global receptive field, but attention focused more on local regions due to convolutional influence.
- **Attention Modulated Convolution**: Receptive field defined by the kernel size, focusing on local neighborhoods.
- **Convolution Modulated Attention**: Receptive field determined by the depth-wise convolution kernel, capturing local features.

Team Member	Contributions
Haseeb	Task 2, 3, 4, 5 on CLIP; Task 2, 3 on VGG
Aazen	Task 2, 3, 4, 5 on ViT; Task 4, 5 on VGG; Created global style change dataset for Task 5
Ahsan	Task 1, Task 6; Created dataset with noised and scrambled sets for Task 5; Sourced datasets for Task 4

Table 4: Contributions of each team member to the project.