Faculté des Sciences et Ingénierie - Sorbonne université

Master Informatique parcours - Données Apprentissage et Connaissances

Reconnaissance Des Formes pour l'analyse et l'interprétation d'images

Project report

Section 2
Deep Learning Applications

Done by:
Samy NEHLIL - Allaa BOUTALEB
Supervised by:
Nicolas Thome

December 2023

Contents

List of Figures

Chapter 1

Transfer Learning through feature extraction from a CNN

1.1 Introduction

Transfer Learning, a transformative approach in machine learning, leverages the concept $\mathcal{T}_s \rightarrow \mathcal{T}_t$, where knowledge from a source task $\mathcal{T}_s$ is applied to a target task $\mathcal{T}_t$. This technique is pivotal in overcoming data scarcity and reducing computational costs, as it repurposes pre-trained models for new tasks. By transferring learned features from extensive datasets, it enhances model performance in specialized areas.

1.2 Context and Problem Statement

Conventional machine learning relies heavily on large, domain-specific datasets, a requirement often impractical due to data limitations and high computational demands. Transfer Learning addresses this by adopting the principle $\mathcal{M}_s \rightarrow \mathcal{M}_t$, where a model $\mathcal{M}_s$ trained on a source task is adapted to a model $\mathcal{M}_t$ for a target task. The challenge lies in balancing knowledge retention and adaptation, encapsulated in the formulation $\mathcal{L}(\mathcal{M}_s, \mathcal{M}_t)$, where $\mathcal{L}$ represents the learning function. This balance is crucial to prevent negative transfer and optimize feature transferability. Transfer Learning thus not only streamlines the learning process but also extends the capabilities of AI in diverse applications.

1.3 Roadmap

This practical is organized into three distinct sections:

1. **Exploring VGG-16 Architecture:** We begin our journey with an in-depth exploration of the VGG-16 architecture. This includes a list of detailed answers to elucidate the nuances of VGG-16, followed by practical applications on various images from the CIFAR-10 dataset. Additionally, we enhance our understanding by visualizing the activation maps within the network, providing insights into its internal workings.

2. **Transfer Learning with VGG-16 on the 15 Scene Dataset:** The second section delves into the application of Transfer Learning using VGG-16 on the 15 Scene dataset. Here, we not only apply the principles of Transfer Learning but also intertwine it with a standalone classifier like SVMs, accompanied by a discussion on the synergy between these methodologies. This section also includes addressing pertinent questions related to Transfer Learning.

3. **Going Further:** In our final section, we conduct a deeper analysis. We study the impacts of different cutoff levels in the VGG-16 network when integrated with SVMs and fine-tune the SVM's C parameter for optimal performance. The exploration extends to examining the effects of data augmentation on model performance and analyzing the implications of freezing versus not freezing VGG-16 layers. Concluding the section, we broaden our scope by comparing the performance of VGG-16 with other pre-trained models such as ResNet, AlexNet, and Inception (GoogLeNet), offering a comprehensive view of the landscape of pre-trained models in Transfer Learning.

1.4 Section 1: VGG16 Architecture

1.4.1 Q1. Knowing that the fully-connected layers account for the majority of the parameters in a model, give an estimate on the number of parameters of VGG16.

For VGG16:
1. The first fully connected layer has an input feature map of **7x7x512** and outputs to **4096 neurons.**
2. The second fully connected layer takes these **4096 features** and outputs to **1000 classes**.
Therefore, we calculate the parameters as follows:
1. Parameters for the first fully connected layer = **(7 * 7 * 512) * 4096 + 4096 (bias terms).**
2. Parameters for the second fully connected layer = **4096 * 1000 + 1000 (bias terms).**
The correct estimate for the number of parameters in the two fully connected layers of the VGG16 model is approximately **106.86 million**, which includes both the weights and the biases for each layer.

1.4.2 Q2. What is the output size of the last layer of VGG16? What does it correspond to?

The output size of the last layer of VGG16 is **1x1000**. This corresponds to the number of classes in the ImageNet dataset, which VGG16 is commonly trained on. Each of the 1000 units in the output layer corresponds to a specific class in the ImageNet dataset, representing the model's predicted probability that the input image belongs to each of those classes. This output is often passed through a softmax function to convert the logits to normalized probabilities.

1.4.3 Q3. Bonus : Apply the network on other images

In this section, we evaluated the performance of the VGG-16 network, originally designed for ImageNet, on the CIFAR-10 dataset. CIFAR-10 is a well-known dataset in the field of machine learning and computer vision, consisting of 60,000 32x32 color images in 10 different classes, with 6,000 images per class. It is widely used for benchmarking image recognition algorithms.

Figure 1.1: CIFAR-10 dataset

- **Loading the CIFAR-10 dataset**: For our experiment, we adapted the images from CIFAR-10 to be compatible with VGG-16. This involved resizing the images to 224x224 pixels, which is the input size expected by VGG-16. Additionally, the images were normalized using mean values $\mu = (0.4914, 0.4822, 0.4465)$ and standard deviation values $\sigma = (0.2023, 0.1994, 0.2010)$. These values are

specific to the CIFAR-10 dataset and differ from the normalization parameters used for ImageNet. This normalization helps in aligning the data distribution of CIFAR-10 with the expectations of the VGG-16 model, which was originally trained on ImageNet.

- **Modifiying VGG-16**: VGG-16 was originally designed to classify images into one of 1,000 classes for the ImageNet challenge. However, since CIFAR-10 only contains 10 classes, we modified the network by replacing the final classification layer. The new layer in VGG-16 outputs a 10-dimensional vector to match the 10 categories of CIFAR-10, allowing the network to classify the CIFAR-10 images effectively.

- **Wrapping everything up with lightning**: To streamline the experimentation process, we utilized PyTorch Lightning, a wrapper for PyTorch that simplifies code and enhances readability. PyTorch Lightning allowed us to focus on the model and the logic, abstracting away much of the boilerplate code typically associated with deep learning experiments.

- **Training VGG-16**: For training VGG-16 on the CIFAR-10 dataset, we set the batch size to 40 and used a learning rate of 0.001. We also incorporated weight decay of 5^{-4} in the optimization process. Weight decay is a form of regularization that helps prevent overfitting by penalizing large weights in the model. This is especially crucial for a dataset like CIFAR-10, which, despite its popularity, is relatively small compared to ImageNet. The weight decay ensures that the model does not become too specialized to the training data, enabling it to generalize better to new, unseen data.

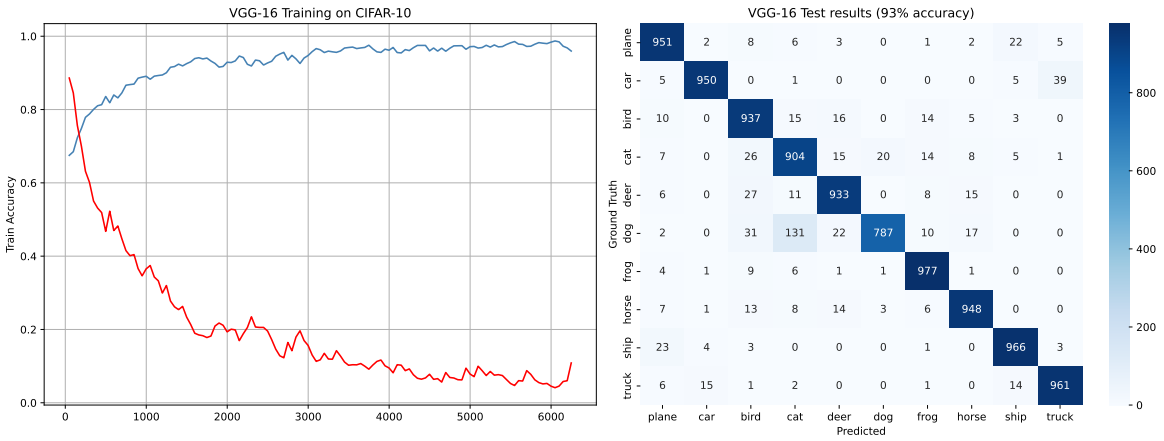


Figure 1.2: VGG-16 on CIFAR-10 dataset

1.4.4 Q4. Bonus : Visualizing activation maps

To interpret the results from the activation maps obtained after passing an image through the first convolutional layer of VGG-16, we need to consider what the first layer of a convolutional neural network (CNN) typically learns and how it reacts to input images.

Understanding Convolutional Filters:

- Each filter in a CNN's convolutional layer is designed to respond to a specific type of feature in the input image, such as edges, textures, or color gradients.

- When an image is passed through the CNN, these filters produce activation maps that highlight the areas of the image where the features they're designed to detect are most present.

- Bright areas in an activation map indicate high activation, meaning the filter detected the feature it's looking for in that region of the image.
- Dark areas indicate low activation, where the feature of interest is absent or not prominent.

First Layer Activation Maps: The first layer is usually responsible for detecting low-level features. In the case of a cat or a dog image, these might include:

- **Edges:** Filters may detect the boundaries between the cat and the background.
- **Textures:** Some filters might be sensitive to the fur texture of the cat.
- **Colors:** Filters could highlight regions with specific color patterns that are characteristic of the cat.

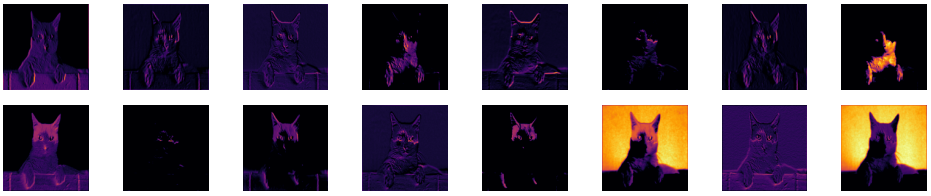


Figure 1.3: Activation maps of a cat image

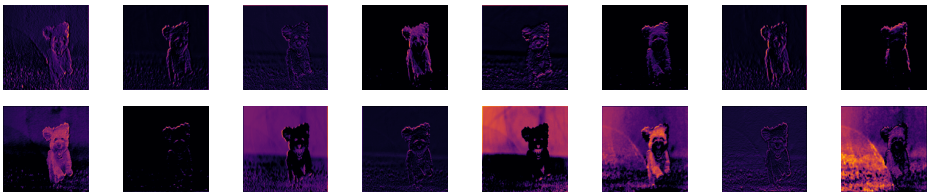


Figure 1.4: Activation maps of a dog image

1.5 Section 2: Transfer Learning with VGG16 on 15 Scene

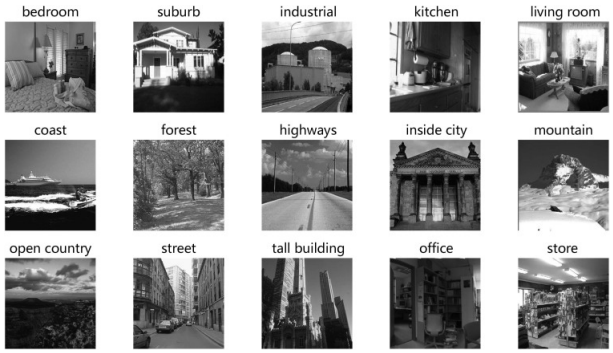


Figure 1.5: 15-Scene dataset

1.5.1 Approach

1.5.1.1 Q5. Why not directly train VGG16 on 15 Scene?

VGG16 is a deep convolutional network with 138 million parameters, designed for large-scale image recognition tasks. The 15 Scene dataset, on the other hand, is relatively small and consists of only 15 categories with a few thousand images. Directly training such a large model on a small dataset is impractical for several reasons:

- **Data Scarcity:** VGG16's architecture is data-hungry due to its depth. The 15 Scene dataset may not provide enough examples to optimize all the parameters without overfitting.
- **Transferability of Features:** VGG16 has been shown to learn hierarchical features that are transferable across different domains. The lower and middle layers capture generic features (edges, textures, etc.), which are relevant across different tasks. It is more efficient to utilize these pre-learned features rather than learning from scratch.
- **Resource Intensity:** Training from scratch requires significant computational resources and time. Transfer learning allows for leveraging pre-trained weights, reducing both the computational cost and time.

1.5.1.2 Q6. How can pre-training on ImageNet help classification for 15 Scene?

ImageNet is a large-scale, diverse dataset with over a million images across 1000 categories. Pre-training on such a dataset ensures that the network learns a robust, generalized representation of visual features. This pre-training serves two main functions:

- **Feature Generalization:** It ensures that the network captures a broad range of features, which can be beneficial for the 15 Scene dataset, which consists of various natural and urban scenes.
- **Learning Efficiency:** When fine-tuning on the 15 Scene dataset, the model can adjust the weights of these pre-trained features to better suit the new task, leading to potentially better performance with significantly less data.

1.5.1.3 Q7. What are the limits of feature extraction?

While feature extraction from a pre-trained model is a powerful technique, it has its limitations:

- **Domain Specificity:** VGG-16 is trained on the ImageNet dataset, which is quite diverse. However, the features learned by the network may be too general for the specific context of a smaller or more specialized dataset like the 15 Scene. This can limit the effectiveness of transferred features.
- **Layer Selection:** Choosing which layer to extract features from is critical. In VGG-16, deeper layers may encode features too specific to ImageNet classes, which may not align well with the target dataset. The relu7 layer is close to the output and may carry this risk.
- **Feature Redundancy:** Not all features from the relu7 layer may be relevant for the new task. This can introduce redundancy and unnecessary complexity into the subsequent classifier, like an SVM.

1.5.2 Feature Extraction with VGG16

1.5.2.1 Q8. What is the impact of the layer at which the features are extracted?

The layer at which features are extracted in VGG-16 determines the level of abstraction and detail of the features. Early layers capture basic details like edges and textures, while deeper layers encapsulate higher-level features that are more abstract and specific to the training data. Choosing the right layer is crucial because it affects the transferability of features to new tasks and can influence the performance of the downstream classifier.

1.5.2.2 Q9. The images from 15 Scene are black and white, but VGG16 requires RGB images. How can we get around this problem?

To use black and white images with VGG16, which is designed for RGB images, we can simply replicate the single grayscale channel across the three color channels expected by the network. This allows the network to process grayscale images as if they were color images without requiring any change in the input layer of the model.

1.5.3 Training SVM classifiers

1.5.3.1 Q10. Rather than training an independent classifier, is it possible to just use the neural network?

Yes, it is possible to use the VGG-16 neural network directly on the 15 Scene dataset, rather than training an independent classifier like SVM. This approach involves adapting VGG-16, which was originally trained on ImageNet, to the specific requirements of the 15 Scene dataset through a process called fine-tuning. Here's how we can do it:

- **Adapting the Output Layer:** VGG-16 is designed to classify 1000 different ImageNet classes. For the 15 Scene dataset, we need to classify 15 different scenes. Therefore, we must replace the last fully connected layer with a new one that outputs 15 classes instead of 1000.
- **Fine-Tuning:** Instead of training the entire network from scratch, we use the pre-trained weights as a starting point. We can either train the entire network on the new dataset or freeze the earlier layers and only train the modified final layers. Fine-tuning only the last few layers is faster and requires less data.

1.6 Section 3: Going Further

In this section, we categorize our experiments into two distinct groups based on the methodology and objectives:

- The first category encompasses experiments that combine the VGG-16 architecture with an SVM classifier. Here, we primarily focused on modifying VGG-16 by experimenting with various cut-off points and delving into the implications of the C parameter within the SVM's performance. These initial experiments were crucial in evaluating the synergistic effectiveness of VGG-16 when integrated with an SVM classifier.
- The second category of our experiments marks a shift in our approach, particularly evident in our analysis of data augmentation effects, the impact of freezing layers in the VGG model during learning, and comparative studies with other models. In these latter experiments, we diverged from using the SVM classifier. Instead, we adopted a strategy of modifying the neural network models directly. This modification, referred to as 'decapitation,' involved removing the original final layers of the models, including VGG-16, and replacing them with a new classification layer tailored for 15 output categories. This approach enabled a more straightforward and direct assessment of the intrinsic performance capabilities of these altered neural network architectures in various classification tasks.

1.6.1 Experimentation with Varying Cutoffs in VGG-16 Architecture

Context: The VGG-16 architecture is renowned for its simplicity, primarily utilizing 3x3 convolutional layers, progressively stacked in deeper arrangements. Volume reduction within the network is managed through max pooling. The architecture culminates in two densely connected layers, each comprising 4096 nodes, and concludes with a softmax classification layer. This overview, while comprehensive, only scratches the surface of the architecture's complexity.

Figure 1.6: VGG-16 Architecture

In implementations such as PyTorch's rendition of VGG-16, the model includes an intricate arrangement of Conv2D layers interspersed with ReLU activation functions. These layers are systematically organized into five distinct blocks, where each block comprises a varying count of convolutional layers, succeeded by a max-pooling layer.

Experiment: In our experiment, we aimed to assess the classification efficacy of the Scene 15 dataset utilizing a Support Vector Machine (SVM) linked to the VGG-16 model. The crux of this experiment was to explore various cutoff points within the convolutional (feature extraction) and classifier segments of the VGG-16 framework. Our objective was to identify the optimal cutoff combination that maximizes classification accuracy.

The experimental procedure is delineated as follows:

1. **Feature Extraction via VGG-16:** Leveraging the convolutional layers of VGG-16 for feature extraction, we explored assorted cutoff points within these layers to evaluate the impact of extraction depth on classification accuracy. Concurrently, various cutoff points in the classifier segment of VGG-16 were examined to discern their influence on accuracy.
2. **Classification Using SVM:** The processed output from VGG-16 was then channeled into an SVM classifier.

We tested multiple configurations of cutoffs in both the convolutional and classifier layers, aiming to discover the most effective setup for the Scene 15 dataset classification.

To effectively analyze and present our findings, we utilized a heatmap for visualization. In this heatmap:

- The x-axis represents different cutoffs in the classifier part of VGG-16.
- The y-axis shows various cutoffs in the convolutional (feature extractor) part.
- The values in the heatmap are the SVM test accuracies obtained for each combination of cutoffs.

This heatmap provided us with a clear visual representation of how different configurations impact the classification accuracy, enabling us to pinpoint the most effective combination for the Scene 15 dataset classification.

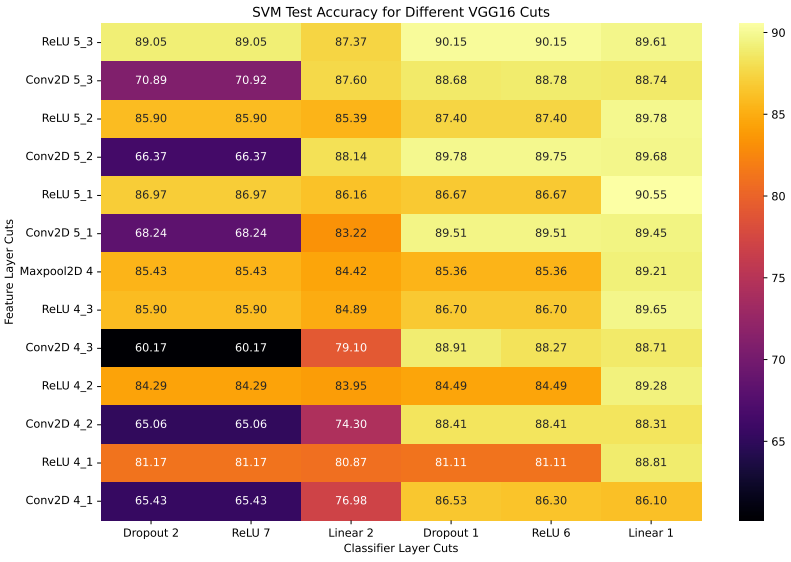


Figure 1.7: SVM Test accuracy based on different cuts in VGG's **conv** and **classifier** layers

Observations: The analysis of the heatmap generated from our experiment led to some intriguing observations:

- The most effective configuration for classification accuracy was identified when the cutoff was applied at the third-to-last ReLU activation layer (specifically, ReLU 5_1) within the convolutional section, coupled with a cutoff at the initial fully connected (FC) layer (Linear 1) in the classifier segment. This particular combination suggests a synergy between deeper feature extraction in the convolutional layers and early-stage classification in the linear layers.
- It was observed that earlier cutoffs in the classifier segment tended to produce less than optimal results, particularly when the convolutional section of VGG-16 was not fully utilized. On the other hand, implementing cutoffs at later stages within the classifier part appeared to compensate for earlier cutoffs in the convolutional section. This finding indicates a trade-off between depth of feature extraction and the complexity of the classifier part, suggesting that deeper convolutional processing can be somewhat mitigated by more comprehensive linear classification layers.

Conclusion: These observations highlight the intricate balance between feature extraction and classification layers in deep learning models. The depth of feature extraction plays a crucial role in the model's ability to discern and differentiate complex patterns in the data. However, this needs to be carefully balanced with the capacity and complexity of the classifier part, which is responsible for making sense of these features and making accurate predictions.

1.6.2 Analyzing the Impact of the C Parameter on SVM Performance

Context: The C parameter in Support Vector Machines (SVM) serves as a crucial hyperparameter, primarily functioning as a regularization term. It essentially controls the trade-off between achieving a low training error and maintaining a low model complexity. In simpler terms, a higher value of C implies less regularization, allowing the model to focus more on fitting the training data as accurately as possible, potentially at the risk of overfitting. Conversely, a lower C value increases regularization, which may lead to a simpler decision boundary but potentially underfit the data.

Experiment: Building upon our previous experiment with the VGG-16 model, we conducted a further analysis by tuning the C parameter of the SVM. We retained the optimal combination of cutoffs from the previous experiment for consistency. The values of C tested were:

$$0.0001, 0.001, 0.01, 0.1, 1$$

To ensure a thorough evaluation, we performed a 5-fold cross-validation for each candidate C value, culminating in a total of 25 fits. This method allowed us to robustly assess the performance of each C value under different subsets of the data.
For each value of C, we meticulously recorded two key metrics:

- **Mean Cross-Validation (CV) Score:** This metric provided an average of the model's performance scores over the 5 folds of cross-validation. It served as an indicator of how well the model, with a specific C value, generalized to unseen data.
- **Mean Fit Time (in seconds):** This metric represented the average time taken for the model to fit the data in each fold of the cross-validation. It helped us understand the computational efficiency associated with different C values.

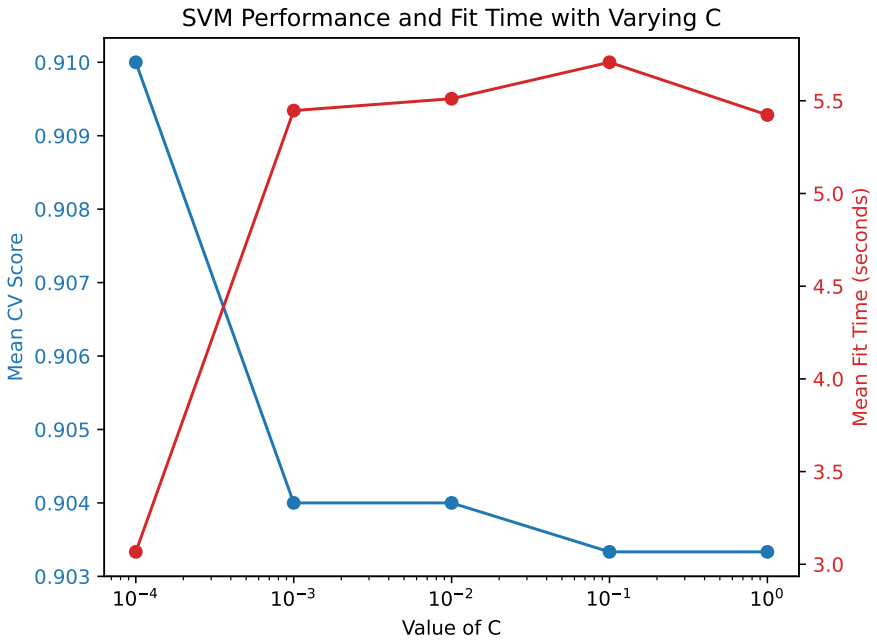


Figure 1.8: SVM fit time and performance with varying C parameter

Observations: From the experiment focusing on the C parameter in our SVM model, the following insights were drawn:

- The C value of 0.0001 delivered the most favorable results. This setting not only achieved the highest Cross-Validation (CV) Score but also boasted the shortest fit time, clocking in at approximately 3 seconds.
- Increasing the C value leads the model towards tolerating fewer classification errors, which in turn results in a more intricate decision boundary. While the C values of 0.001, 0.01, 0.1, and 1 still produced commendable scores (maintaining over 90% accuracy), they required slightly more time to fit and demonstrated marginally lower performance compared to a C value of 0.0001 (which achieved 91% accuracy in just 3 seconds).

Conclusion: The experiment indicates that lower values of C are preferable in our SVM model when coupled with VGG-16 on a simple dataset like Scene 15. Our findings advocate for a more regularized approach in SVM. The efficacy of lower C values can be attributed to the need for balancing the model's complexity with the simplicity of the dataset. This balance is crucial for preventing overfitting.

1.6.3 Exploring the Impact of Data Augmentation on Model Performance

Context: Data augmentation is a technique used to enhance the diversity and quantity of training data by applying various transformations to existing data samples. Its primary purpose is to introduce variability and simulate different perspectives, thus helping the model to generalize better and reduce the risk of overfitting. This is particularly crucial in deep learning, where models often require large amounts of diverse data to perform optimally.

Experiment: In our study, we applied data augmentation to the 15 Scene dataset, enhancing it with additional transformations beyond basic resizing and normalization. The applied transformations included:

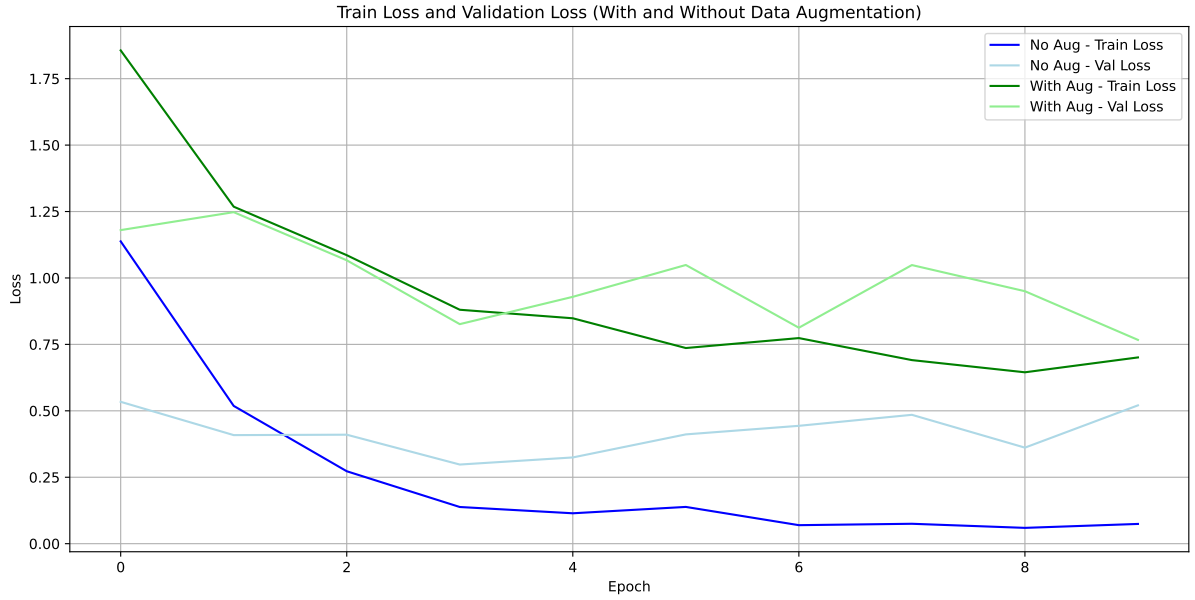
- `transforms.RandomHorizontalFlip(p=0.5)`,
- `transforms.RandomRotation(15)`,
- `transforms.RandomResizedCrop((224, 224), antialias=True)`,
- `transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2)`,
- `transforms.RandomPerspective(distortion_scale=0.2, p=0.5)`.

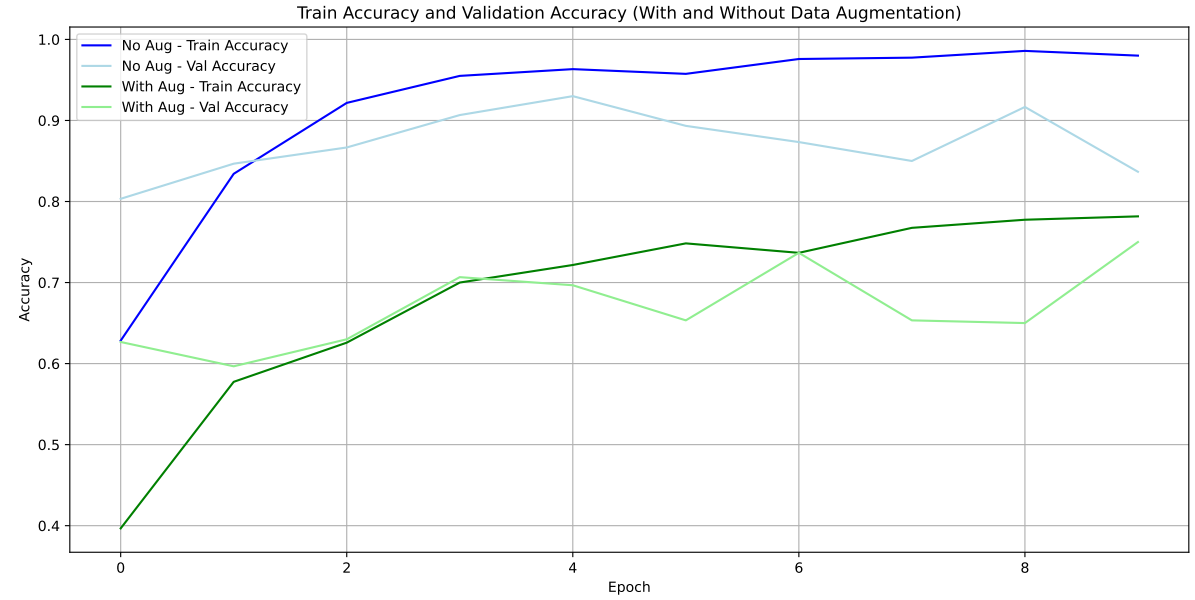
Figure 1.9: Examples of the applied transformation to augment data

The dataset was divided into training, validation, and test sets, with an allocation of 80% of the data for training, 20% for validation, and a separate set for testing. Importantly, the transformations were applied only to the training and validation sets to evaluate the model's ability to generalize to unmodified test data.
For this experiment, we utilized the VGG-16 model, modified to classify into 15 categories instead of its original 1000. The model was configured with a cutoff at the ReLU 7 layer, and the final layer was adapted for the 15 Scene dataset classification. We did not employ an SVM in this setup. The model was trained over 10 epochs for each scenario: with and without data augmentation.
During the training process, we meticulously logged the following metrics:

- Training Loss and Validation Loss,
- Training Accuracy and Validation Accuracy,

for both scenarios (with and without data augmentation). Additionally, we evaluated and recorded the final test accuracy on unseen data to measure the model's generalization capabilities. Here are the results obtained:





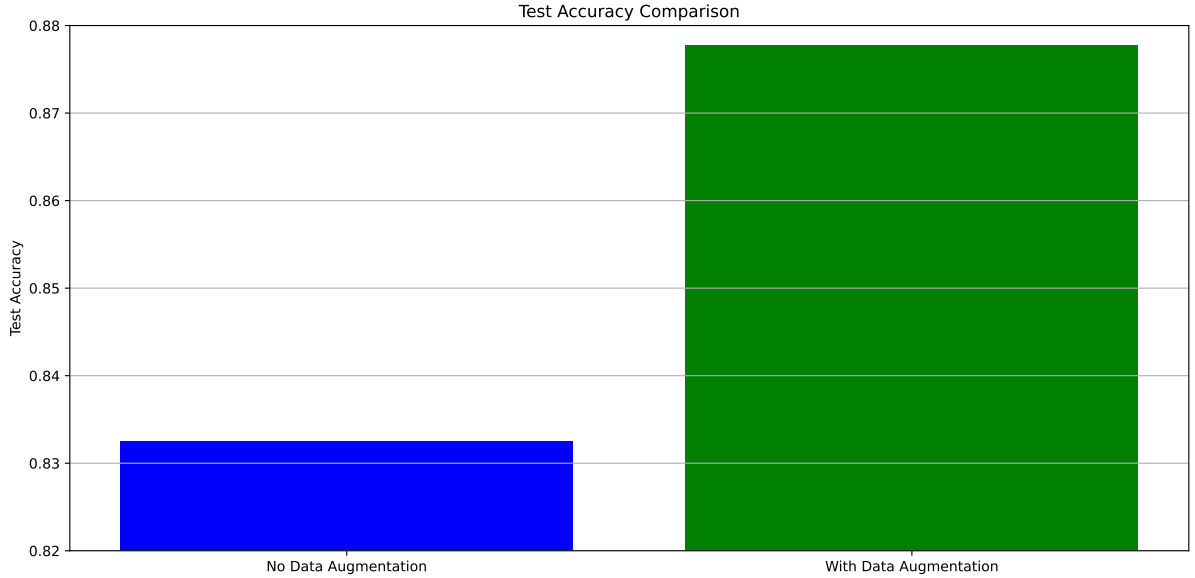


Figure 1.10: Train, validation and test results with and without Data Augmentation

Observations: The figure above yielded the following notable observations:

- **Higher Training and Validation Loss with Data Augmentation:** At the 10th epoch, training loss was observed to be higher when data augmentation was employed, as opposed to the scenario without data augmentation, where both training and validation losses were approaching zero. This phenomenon can be attributed to the increased complexity and variability introduced by data augmentation. The model, when exposed to augmented data, faces a more challenging learning environment, thus resulting in higher loss values during training.
- **Lower Training and Validation Accuracy with Data Augmentation:** Consistent with the higher loss, the training and validation accuracies were lower when data augmentation was used. This outcome initially suggests that data augmentation might be detrimental to the model's performance.
- **Improved Performance on Unseen Test Data with Data Augmentation:** Contrary to the initial impression, the model trained with data augmentation outperformed the one without data augmentation on the unseen test set, achieving approximately 88% accuracy compared to 83%. This indicates that despite lower performance metrics during training and validation, the augmented model developed a better generalization capability.

Conclusion: The experiment underscores the significant benefits of data augmentation in deep learning, particularly for tasks involving image classification. While data augmentation may lead to higher training and validation losses, and seemingly lower accuracies during the training phase, this should not be misconstrued as an indication of poor model performance. Rather, it reflects the model's exposure to a more diverse and challenging training environment, which ultimately enhances its ability to generalize to new, unseen data. This is evidenced by the improved accuracy on the test set. Thus, the findings reinforce the idea that training and validation results are not always the definitive indicators of a model's effectiveness, especially in scenarios involving data augmentation. The key measure of success in such cases lies in the model's performance on unseen data, which is a true test of its generalization capability and robustness.

1.6.4 Analyzing the Impact of Freezing VGG-16 Layers

Context: Freezing layers in a neural network is a technique where certain layers are prevented from updating during backpropagation. This means their weights remain constant, and the gradient flow through these layers is effectively halted. Freezing layers can be beneficial, particularly in transfer learning scenarios, as it allows the model to retain the knowledge gained from pre-training on a large dataset. This can be especially useful when the subsequent training dataset is relatively small or less diverse. However, it also means that the model's ability to adapt to new, possibly unique features of the new dataset is limited, as only a part of the network is fine-tuned.

Experiment: In our experiment, we modified the VGG-16 architecture to adapt it to the Scene 15 dataset. We froze all the layers of VGG-16 except for the last one. This was achieved by setting the 'requires_grad' attribute of the parameters in all layers except the last to 'False', ensuring that these layers do not update during training. Consequently, only the last layer, which we replaced with a new classification layer suitable for 15 output classes, was trainable. This approach allowed us to leverage the pre-learned features of VGG-16 while fine-tuning the model to the specific requirements of the Scene 15 dataset.

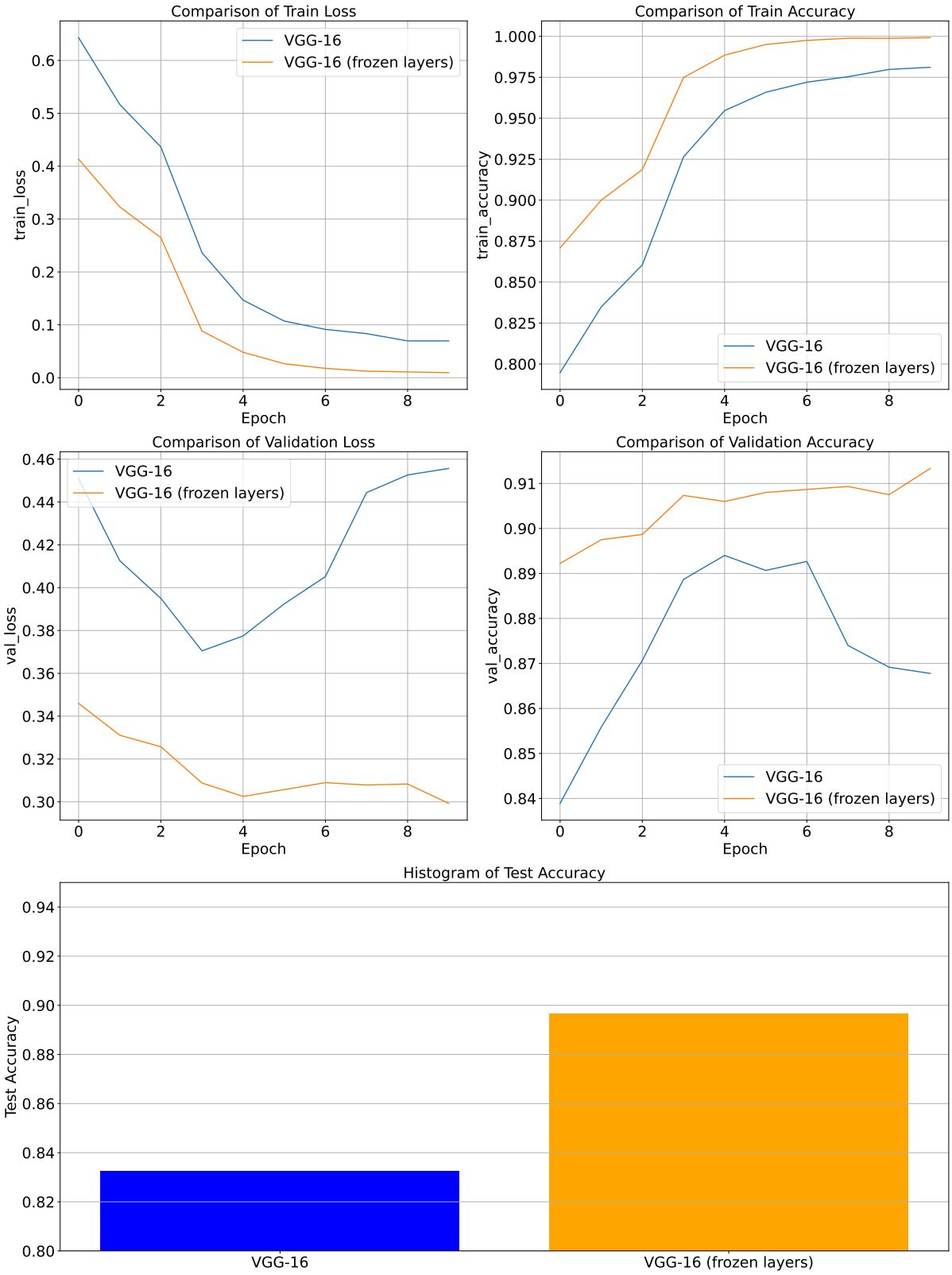


Figure 1.11: Train, validation and test results with and without the freezing process

Observations: The variant of VGG-16 with frozen layers exhibited superior performance compared to its fully trainable counterpart. Specifically, we observed:

- Lower training/validation loss, Higher training/validation accuracy

- An overall higher test accuracy, achieving 90% compared to 83% by the non-frozen model.

Conclusion: This enhanced performance can be attributed to the stability and reliability of the pre-trained features in VGG-16. By freezing the layers, we effectively utilized the robust feature extraction capabilities developed through pre-training on extensive datasets (ImageNet). This prevented overfitting to the training data, a common issue in deep networks, especially when trained on relatively smaller datasets. The last trainable layer, being tailored to the Scene 15 dataset, provided the necessary adaptability, enabling the model to fine-tune its predictions effectively for this specific task.

1.6.5 Comparison with other Pre-Trained Vision Models: ResNet, Inception, AlexNet...

Context: In the realm of computer vision, several pre-trained models have gained prominence, each with its unique architecture and strengths. Among the most well-known are variants of VGG, ResNet, and Inception. While VGG has been detailed previously, it is insightful to explore ResNet and Inception for their distinctive features.

ResNet (Residual Networks): ResNet, short for Residual Networks, revolutionized deep learning architectures through its introduction of "residual connections." Its main particularity lies in addressing the vanishing gradient problem, which often plagues very deep networks. ResNet achieves this by incorporating shortcut connections that bypass one or more layers. These connections allow gradients to flow through the network directly, facilitating the training of much deeper networks than was previously possible. This architectural innovation has enabled ResNet to achieve deeper network structures, leading to significant improvements in performance.

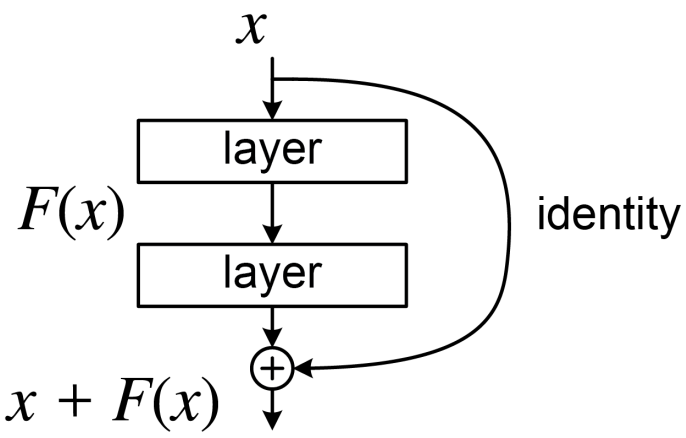


Figure 1.12: A Residual Block in a deep Residual Network. Here the Residual Connection skips two layers.

Inception Network: The Inception network, on the other hand, is known for its complex and efficient architecture. Its main particularity is the use of "Inception modules," which allow the network to choose from multiple filter sizes (1x1, 3x3, 5x5) within the same layer. This design enables the network to capture information at various scales, increasing its adaptability and efficiency. Furthermore, Inception incorporates auxiliary classifiers to combat the vanishing gradient problem and includes batch normalization to accelerate training.

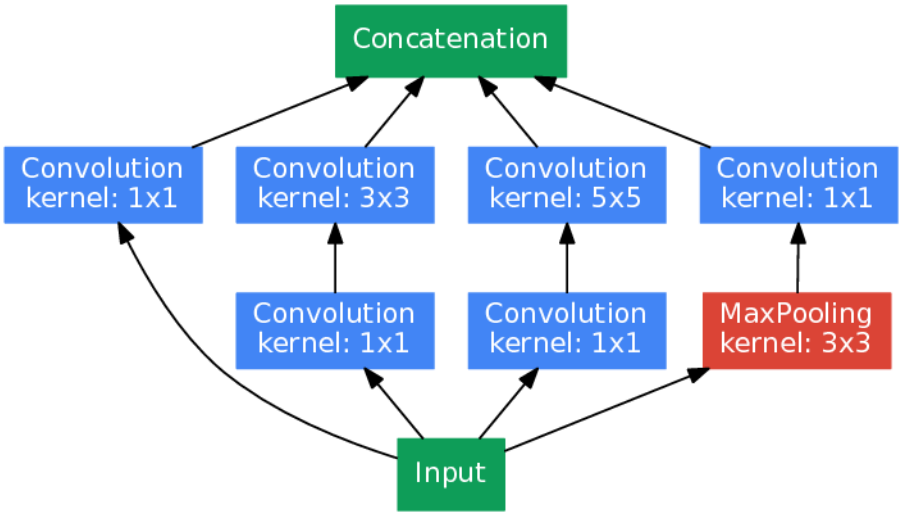


Figure 1.13: Building block of the Inception architecture

Moving away from FCNs: Both ResNet and Inception mark a departure from architectures like VGG, AlexNet, or LeNet, particularly in their reliance on Fully Connected Networks (FCNs) at the end. Unlike VGG, which uses large FC layers at the end of the network, ResNet and Inception use global average pooling to reduce the dimensions before the final classification layer. This approach reduces the total number of parameters, diminishing the risk of overfitting and enhancing computational efficiency. The reduction in parameters also makes these models more adaptable and scalable for various applications.
While VGG remains a powerful and straightforward model, the advances in architectures like ResNet and Inception offer more efficient and flexible alternatives for tackling complex computer vision tasks. The choice among these models largely depends on the specific requirements of the task at hand, including the complexity of the dataset, computational resources, and the need for model scalability and adaptability.
Experimental Setup with Various Pre-Trained Models
In our comprehensive analysis, we conducted experiments using a variety of well-known pre-trained models. The list includes:

- AlexNet
- VGG16, VGG19
- ResNet50, ResNet152
- Inception v3

For each of these models, we modified the original architecture by removing the top (decapitating) and replacing it with a new classification layer tailored to output 15 categories, aligning with our dataset. The image input sizes were adjusted according to the specific requirements of each model. For AlexNet, VGG16, VGG19, and both ResNet variants, the input images were resized to 224x224 pixels. Inception v3, due to its unique architecture, required the images to be resized to 299x299 pixels.
The training process for all models was standardized to ensure consistency in comparison. Each model was trained over a duration of 10 epochs. The training dataset was split, with 80% used for actual training and the remaining 20% allocated for validation. The entire separate test set was utilized for evaluating model performance.

During the training and validation phases, we meticulously logged the following metrics for each model:

- Training Loss and Accuracy per Epoch
- Validation Loss and Accuracy per Epoch

Upon completion of the training, we also evaluated and recorded the test accuracy for each model. It is important to note that no data augmentation techniques were applied to the dataset in these experiments. This approach allowed us to assess the baseline performance of each model in a controlled and uniform setting.

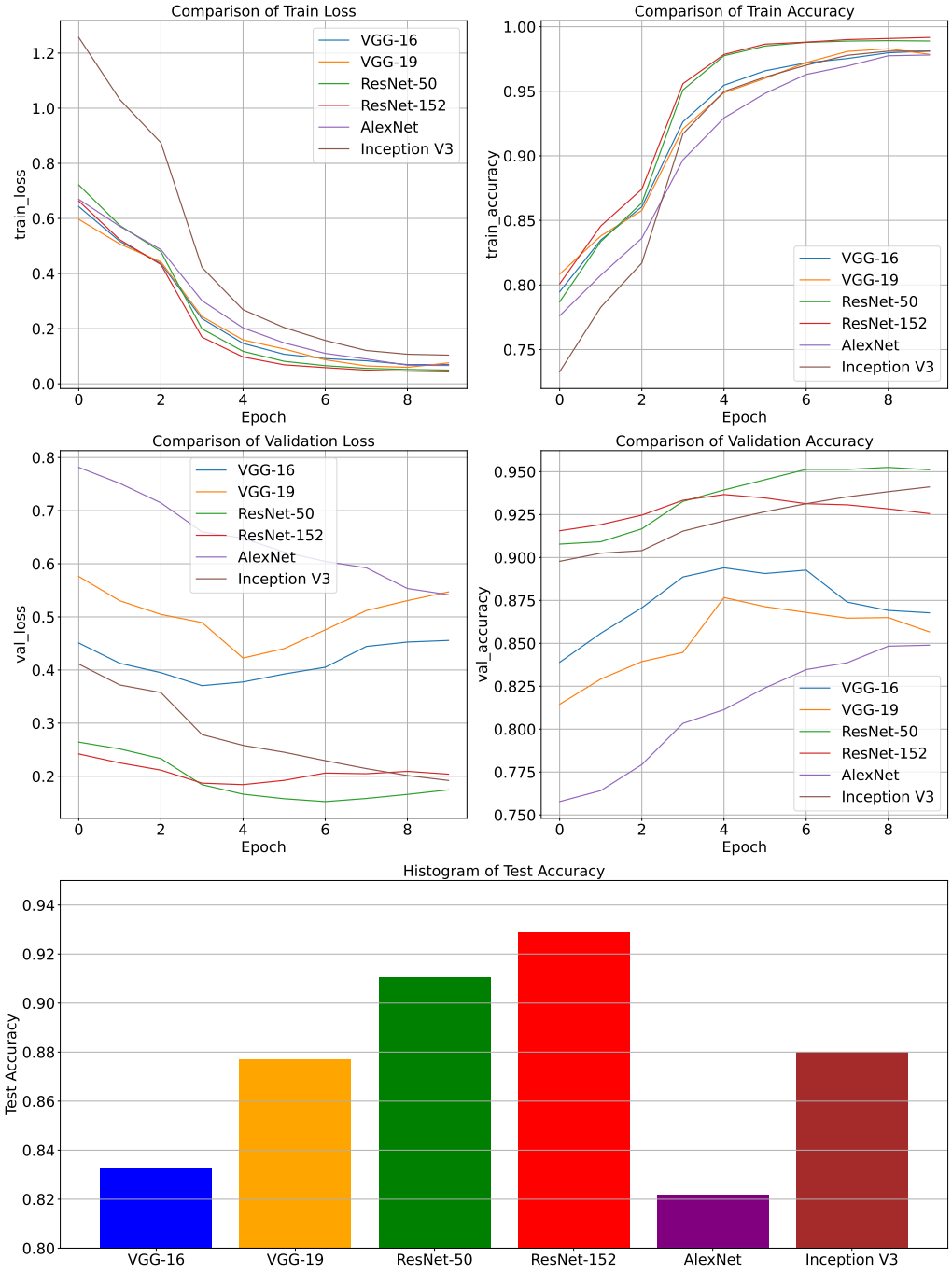


Figure 1.14: Comparison of various models' performance on 15 Scene dataset

Observations: Our experiment with various pre-trained models yielded several key insights:

- **AlexNet Performance:** AlexNet, being one of the earliest deep learning models, showed the least impressive performance among the tested models. Its architecture is not as deep as VGG, and it relies heavily on Fully Connected Networks (FCNs) towards the end. This design makes it more susceptible to overfitting, as evidenced by its comparably high training accuracy and low training loss. The limitations of AlexNet stem from its simpler and now somewhat outdated architecture, which lacks the advancements in depth and regularization seen in more recent models.

- **VGG16 vs. VGG19:** In the comparison between VGG16 and VGG19, the latter exhibited a marginally superior performance. This can be attributed to the additional convolutional layers in VGG19, which enable it to capture more complex features. The presence of FCNs in both models contributes to a noticeably longer training time compared to ResNet or Inception. Research indicates a saturation point in performance improvement with increasing depth in VGG models, beyond which the model's effectiveness diminishes, paving the way for architectures like ResNet.

- **ResNet50 vs. ResNet152:** Among the ResNet variants, ResNet152 outperformed ResNet50, which in turn surpassed VGG19. The superiority of ResNet152 is a direct result of its deeper architecture with residual connections, allowing it to learn more complex features without the risk of vanishing gradients or overfitting. The efficiency of ResNet models is further enhanced by the use of global average pooling towards the end, replacing the large FCNs and reducing the overall training time. ResNet152, with its profound depth, demonstrated the highest performance in our tests, achieving nearly 93% accuracy on the test set. With ResNet models, the deeper the better!

- **Inception Performance:** The Inception model outperformed both VGG16 and VGG19, likely due to its innovative architecture that avoids FCNs at the end and utilizes a mix of convolutional kernel sizes to capture a wide range of features. However, it did not surpass the performance of the ResNet models. This could be due to the Inception model's complex architecture, which, while efficient, may not capture as deep or as nuanced features as the very deep ResNet models.

Conclusion: The comparative analysis of these models highlights the evolution and advancements in deep learning architectures. While earlier models like AlexNet and VGG have been foundational, their limitations become apparent when compared to more advanced architectures like ResNet and Inception. ResNet, with its deep layers and residual connections, emerges as a particularly effective architecture, capable of handling deep networks without overfitting, thereby achieving superior performance. Inception, with its unique approach to convolutional layers, also demonstrates its efficacy, especially over traditional architectures like VGG. These insights are invaluable for guiding the choice of architecture in various computer vision applications, emphasizing the importance of considering both the depth and efficiency of the model in relation to the specific requirements of the task.

Chapter 2

Visualizing Neural Networks

2.1 Introduction

This practical work centers around demystifying the internal mechanisms of convolutional neural networks (CNNs), a cornerstone in deep learning for image processing tasks. Despite their widespread success, understanding the decision-making process of these networks remains a significant challenge. This "black box" nature hinders our ability to fully trust and optimize these models, especially in critical applications. The aim of this practical work is to leverage various visualization techniques to interpret and analyze the behavior of CNNs.

2.2 Context and Problem Statement

Convolutional neural networks, while highly effective in image classification and other tasks, often operate without providing intuitive insights into their decision-making processes. The complexity of these networks, characterized by multiple layers and high-dimensional parameter spaces, makes it difficult to decipher how input data is transformed into output predictions. This lack of transparency is particularly problematic in domains where understanding the rationale behind a decision is as important as the decision itself.

2.3 Roadmap, Approaches and Techniques

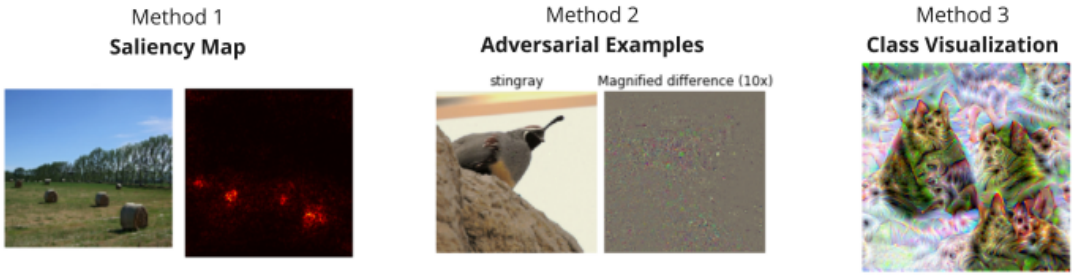


Figure 2.1: Illustration of the different methods

Saliency Maps

Objective The first part of the practical work involves generating saliency maps to visualize which regions of an input image most significantly influence the network's classification decision.
Methodology Following the method proposed by Simonyan et al. (2014), we compute the gradient of the network's output with respect to the input image. This technique highlights the pixels that have the most significant impact on the class prediction.

Insights Saliency maps offer a window into understanding which features of an image are deemed important by the network. This visualization helps in identifying the network's focus areas and provides clues about its learned features.

Adversarial Examples

Objective We explore the network's susceptibility to adversarial attacks, where imperceptible alterations to an image lead to incorrect classifications.
Methodology Inspired by Szegedy et al. (2014), we modify input images based on gradient information to deceive the network into making wrong predictions, highlighting its vulnerabilities.
Relevance This exercise underlines the importance of robustness in neural networks and their potential weaknesses, crucial for improving security and reliability in practical applications.

Class Visualization

Objective The goal here is to visualize what patterns or features a network associates with specific classes.
Methodology We generate images that maximize the class score for a given class, modifying an initial image (like random noise) to enhance features characteristic of that class. This approach is based on the work of Simonyan et al. (2014) and Yosinski et al. (2015).
Application These visualizations provide insights into what the network 'believes' each class should look like, offering a unique perspective on its learned representations.

2.4 Saliency Map

2.4.1 Q1. Show and interpret the obtained results.

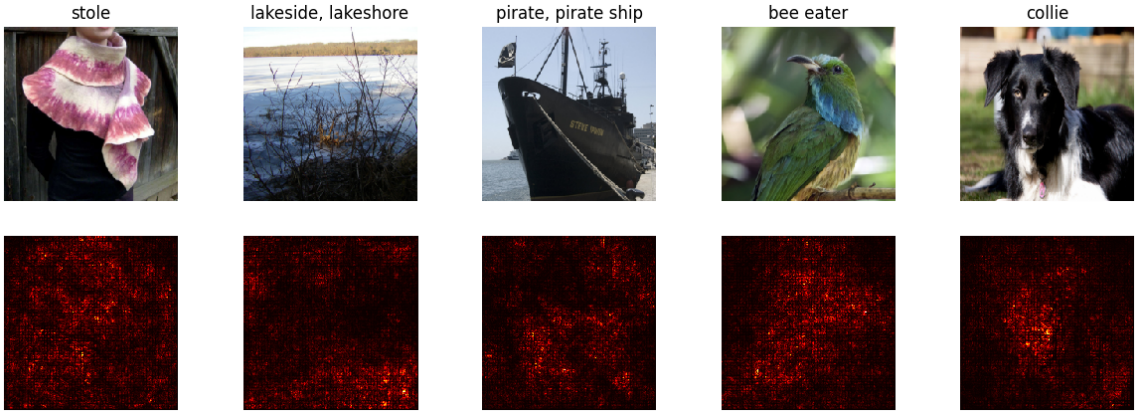


Figure 2.2: Saliency Map for the first five images

Interpretation: Saliency maps are a visualization technique used in computer vision to highlight the regions of an image that a deep learning model focuses on when making a decision, such as classifying an object or scene.
Let's analyze each pair:

- Stole: The original image shows a person wearing a stole. The saliency map, typically generated through techniques like backpropagation, highlights the parts of the image that were most influential in the model's decision-making process. Here, it seems to focus on the texture and edges of the stole.
- Lakeside/Lakeshore: The photo shows a lakeside scene, and the saliency map highlights the regions along the boundary where the land meets the water and the contrast is strongest, suggesting these features are significant for the model.
- Pirate/Pirate Ship: This image of a pirate ship with the pirate flag is paired with a saliency map that emphasizes the contours of the ship and the flag, likely because these are distinctive features for identifying such a ship.
- Bee Eater: The original shows a bee-eater bird, and the corresponding saliency map is lit up in the regions that define the bird's beak and plumage, which are probably key identifiers for the species in the image recognition task.
- Collie: The last pair features a collie dog, and its saliency map shows heightened attention to the face and fur pattern of the dog, which are critical for recognizing the breed.

2.4.2 Q2. Discuss the limits of this technique of visualizing the impact of different pixels.

While saliency maps provide an intuitive visualization of pixel-level contributions to model predictions, they have inherent limitations:

- **High-Dimensionality Reduction:** The technique simplifies the contribution of each pixel to a single scalar value, potentially discarding nuanced interactions between pixels and color channels.

- **Noise Sensitivity:** The reliance on gradient information can make saliency maps sensitive to noise in the image, possibly highlighting regions that are not semantically relevant to the class identification.
- **Class Discrimination:** Saliency maps might not effectively discriminate between classes with similar visual features, as they focus on the most activating features, which might not be class-specific.
- **Overemphasis on Edges:** Since gradients are often high around edges, saliency maps may overemphasize edge pixels, overlooking more subtle, yet important, features.
- **Model Dependence:** The interpretation provided by the saliency map is tightly coupled with the model architecture and parameters, which means different models may yield very different saliency maps for the same image.

2.4.3 Q3. Alternative Uses of Saliency Techniques

Beyond interpreting the workings of a neural network, saliency techniques can be repurposed for tasks such as:

- **Data Augmentation:** By highlighting salient regions, one can generate perturbations focused on the most influential parts of the image, possibly enhancing training.
- **Weakly Supervised Object Localization:** Saliency maps can indicate the location of objects in images without requiring detailed annotations, aiding in localization tasks.
- **Model Debugging:** Discrepancies in saliency maps can signal issues with data or the model itself, such as unintended biases or incorrect feature associations.
- **Adversarial Attack Analysis:** Visualizing the impact of adversarial perturbations on model predictions can help in developing more robust networks.

These alternative applications demonstrate the flexibility of saliency map techniques beyond the primary goal of network interpretation.

2.4.4 Q4. Bonus: Test with a different network, for example VGG16, and comment.

When applying saliency map techniques to VGG16, a network known for its depth and use of small convolutional filters, several points should be considered:

- **Layer Selection:** VGG16 has 16 layers, and selecting which layer's gradients to visualize can impact the saliency map. Early layers may highlight textures and edges, while deeper layers may reflect more complex patterns and class-specific features.
- **Depth of Features:** Due to its depth, VGG16 may capture more hierarchical and abstract representations. Saliency maps might thus reflect a more refined focus on features pertinent to the class prediction.
- **Channel Richness:** VGG16's use of multiple filters at each layer produces a rich set of channel activations. The maximum value across channels may represent a complex amalgamation of features.

- **Resolution and Receptive Field:** The architecture's design results in a different distribution of receptive field sizes, which can influence the resolution of the saliency map. This aspect may cause VGG16's saliency maps to be more focused on localized regions compared to networks with larger receptive fields per layer.
- **Comparison with Original Network:** By comparing saliency maps from VGG16 with those from the original network (e.g., AlexNet or a custom CNN), differences in feature prioritization and abstraction level can be observed, potentially highlighting the architectural influences on feature learning.

Commentary on VGG16 Saliency Maps

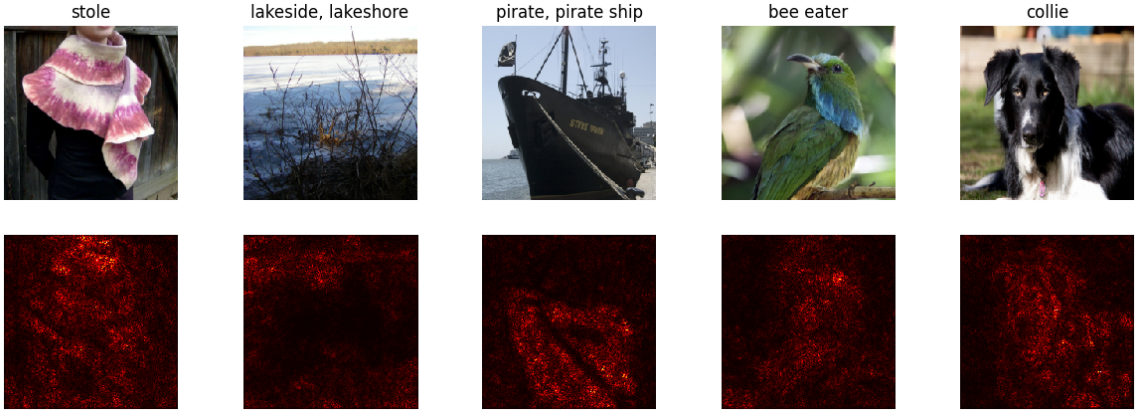


Figure 2.3: Saliency Map for the first five images

Upon generating saliency maps using VGG16, one might observe the following:

- **Detailed Texture and Pattern Emphasis:** VGG16's numerous layers might create saliency maps that emphasize detailed textures and complex patterns that are less apparent in shallower networks.
- **Class-specific Regions:** Given VGG16's capacity to capture more complex features, the saliency maps may more distinctly delineate class-specific regions, aiding in a clearer understanding of the network's decision-making process.
- **Adaptation to Adversarial Examples:** When testing with adversarial examples, VGG16's saliency maps could reveal how these perturbations alter the network's focus, potentially informing strategies for enhancing model robustness.

Conclusion: The use of VGG16 for generating saliency maps underscores the role of network architecture in interpretability. The depth and complexity of VGG16 allow for potentially more nuanced and informative saliency maps, though they may also necessitate more careful analysis to understand the multitude of features being highlighted.

2.5 Adversarial Examples

Another aspect of analyzing neural networks is studying adversarial examples (also known as fooling samples), a practice since a few years. Using an existing pre-trained network and a correctly classified image, this method consists in modifying the image as little as possible such that it becomes misclassified by the network.

2.5.1 Q5. Show and interpret the obtained results.

The provided figures illustrate the outcome of an adversarial example on a convolutional neural network (CNN). The attack subtly alters an image to cause misclassification while remaining imperceptible to human observers.

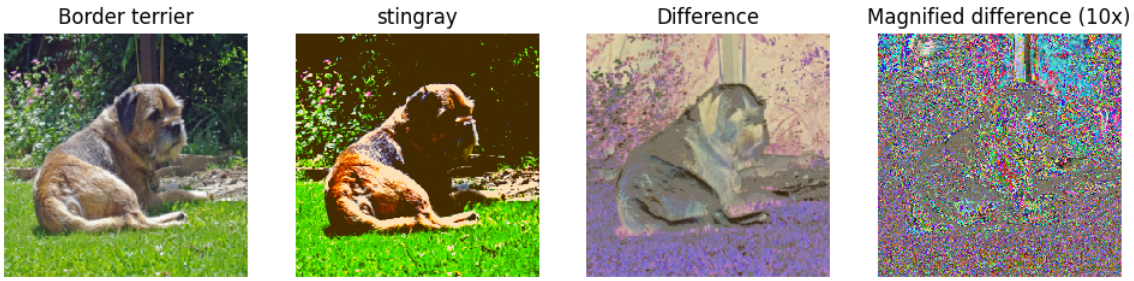


Figure 2.4: Adversarial example

- The **Original Image** depicts a "Border terrier" that the CNN classifies correctly.
- The **Adversarial Image** is the result of the adversarial modification. Visually similar to the original, it is incorrectly classified as a "stingray" by the CNN.
- The **Difference** shows the pixel-level changes applied to the original image, which are almost indiscernible without magnification.
- The **Magnified Difference** amplifies the adversarial perturbations, revealing the targeted pattern that confuses the CNN.

The adversarial example was crafted by performing gradient backpropagation to compute the gradient of the target class score with respect to the input image. The image was then iteratively modified by a scaled addition of the gradient, making minimal changes to the pixel values until the CNN misclassified the perturbed image.
This experiment highlights the fragility of neural networks to adversarial noise and poses significant implications for the robustness of such systems. The stark difference in perception between CNNs and humans raises questions about the interpretability and trustworthiness of decisions made by these networks.

2.5.2 Q6. Consequences of Adversarial Methods on CNNs

Adversarial examples reveal critical vulnerabilities in the way Convolutional Neural Networks (CNNs) process and interpret data. These vulnerabilities have several technical consequences that affect the robustness and generalization capabilities of CNNs.
Impact on Generalization CNNs are designed to generalize from training data to unseen examples. Adversarial attacks demonstrate that:

- Small perturbations can lead to significant performance degradation, indicating that CNNs might be learning to classify based on brittle features that do not generalize well.
- The models might be overfitting to peculiarities in the training data, lacking the ability to capture the underlying semantic meaning of the input.

Robustness of Learned Features Adversarial examples suggest that the features learned by CNNs are not robust:

- The high-dimensional feature spaces of CNNs are susceptible to adversarial perturbations, which can navigate the spaces in directions imperceptible to humans but recognizable by the model.
- CNNs may prioritize high-frequency features that are not as relevant to humans, leading to a misalignment between human visual perception and model decision-making.

Optimization Instability The optimization processes used to train CNNs may not converge to stable solutions:

- The non-convexity of the optimization landscape in deep learning allows for the existence of adversarial examples within close proximity to legitimate data points.
- The gradient-based optimization methods may focus on minimizing loss in a manner that is overly sensitive to specific data distributions, failing to encapsulate broader data variations.

Confidence and Uncertainty Calibration Adversarial perturbations challenge the confidence calibration of CNNs:

- A model's high confidence in its predictions can be misguided when faced with adversarial inputs, questioning the reliability of confidence scores provided by CNNs.
- The inability to recognize and quantify uncertainty in the presence of adversarial inputs limits the deployment of CNNs in scenarios where confidence is crucial.

Direction for Future Research The susceptibility of CNNs to adversarial examples directs future research towards:

- Developing training regimes and architectures that prioritize the learning of semantically meaningful, robust features over those that are fragile and susceptible to adversarial manipulation.
- Investigating novel optimization techniques that are less prone to adversarial exploitation and that lead to more stable and generalizable models.
- Enhancing the interpretability of CNNs to provide insights into model decisions, allowing for better understanding and detection of adversarial vulnerabilities.

The phenomenon of adversarial examples serves as a crucial benchmark for the robustness and reliability of CNNs. Addressing the technical challenges posed by adversarial methods is imperative for advancing the field of deep learning and ensuring the safe application of CNNs in real-world systems.

2.5.3 Q7. Limitations and Alternatives to Naive Adversarial Image Construction

The naive approach to adversarial image generation, characterized by iterative gradient updates, has several limitations. It often leads to images that, while misleading to the model, may not be truly indistinguishable to human observers due to perceptible noise. Moreover, the method may be less effective against robustly trained models or those with defensive mechanisms in place.

Limitations

- **Perceptibility**: Changes may be perceptible, reducing the attack's stealthiness.
- **Transferability**: These adversarial images may not transfer well across different models
- **Robustness**: Models trained with adversarial training may resist naive adversarial perturbations.

Alternatives Recent research proposes several advanced techniques to address these limitations:

- **Projected Gradient Descent (PGD)**: A more powerful attack that considers a set of perturbations bounded by an ℓ_∞ norm, ensuring imperceptibility.
- **Carlini & Wagner (C&W) Attack**: Produces adversarial examples with minimal perturbation, posing a significant threat even to robust models.
- **Generative Adversarial Networks (GANs)**: Can generate more natural adversarial examples by training a generator network to fool the classifier.

While the naive approach provides a foundation for understanding adversarial images, advancements in adversarial machine learning offer more sophisticated methods to create imperceptible and transferable adversarial examples, necessitating continuous development of defensive strategies.

2.6 Class Visualization

In order to visualize the type of patters likely to produce a particular class prediction, and so indirectly analyzing what a network detects, Simonyan et al. (2014) proposes a simple approach, further developed by Yosinski et al. (2015).

2.6.1 Q8. Show and interpret the obtained results.

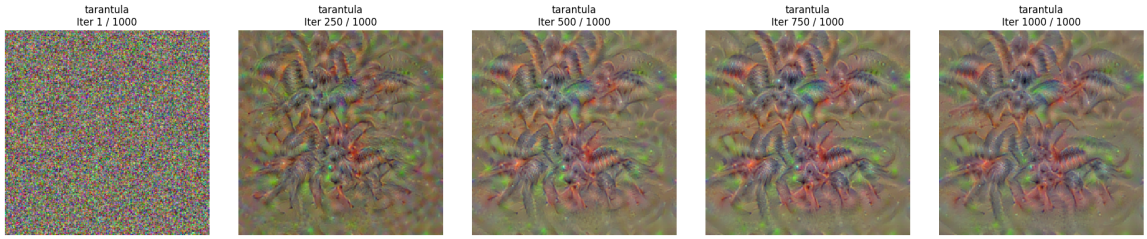


Figure 2.5: Class visualization: started from random noise, Tarantula class. default regularization parameters

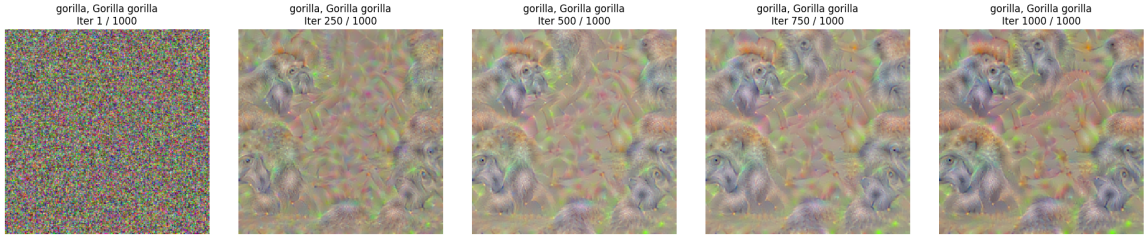


Figure 2.6: Class visualization: started from random noise, Gorilla class. default regularization parameters

Result Interpretation The first visualization shows the progression of a class visualization technique applied to a Convolutional Neural Network (CNN), with the goal of maximizing the response for the "tarantula" class. This technique begins with random noise and iteratively adjusts the input image to increase the activation of neurons associated with the target class. Here's a step-by-step interpretation:

Initialization with random noise The process starts with a random noise image, which contains no discernible features or patterns. This is typical for class visualization methods, as it allows the network to impose its own structure on the data.

Progressive Pattern Formation As the iterations progress, we observe the emergence of patterns and structures that the CNN associates with the tarantula class. This typically involves highlighting textures, edges, and colors that are characteristic of tarantulas as understood by the network.

Regularization and Learning Rate Impact

- **Regularization:** A regularization factor (10^{-3}) is quite low, which suggests that the emphasis is on allowing the network to freely express the features it associates with tarantulas without imposing too much constraint on the smoothness or simplicity of the image.
- **Blurring:** Blurring every ten epochs acts as a form of spatial smoothing, preventing the optimization from amplifying high-frequency noise that the network may falsely interpret as relevant

features.

- **Learning Rate:** A learning rate of 5 is relatively high, indicating that each iteration makes significant changes to the image. This can lead to faster convergence but also risks overshooting optimal activations.

Detailed Feature Evolution

- **Early Iterations (1-250):** We see only slight deviations from noise, indicating that the network is beginning to latch onto low-level features it associates with the target class.
- **Mid Iterations (250-750):** The features become more pronounced. We might see textures or patterns resembling the hairy legs or segmented bodies of tarantulas, reflecting the network's internal representation of these features.
- **Later Iterations (750-1000):** The visualization shows complex, tarantula-like structures with multiple limbs and segments, although still in an abstract form. This suggests that higher-level features that the CNN uses to identify tarantulas are distributed and superimposed in various orientations and scales.

Final Image Interpretation (Iteration 1000) The final image is a complex, abstract representation of what the CNN identifies as a tarantula. It is not meant to be a realistic picture but a manifestation of the most stimulating pattern for the tarantula class neurons. We typically see exaggerated and repeated features like legs, eyes, or body textures, which indicates the key features the network uses for classification.

Summary In summary, the class visualization demonstrates that the CNN has learned to recognize tarantulas by a combination of specific patterns and textures, rather than by matching to an exact image of a tarantula. The visualization reveals the network's abstract understanding of the class, which can be composed of multiple, sometimes overlapping, features that are not necessarily spatially or proportionally accurate to a real tarantula. This demonstrates the power of CNNs to extract and amplify features relevant to the task of classification, although it also highlights that their perception can be quite different from human vision.

2.6.2 Q9. Try to vary the number of iterations and the learning rate as well as the regularization weight.

Number of iterations

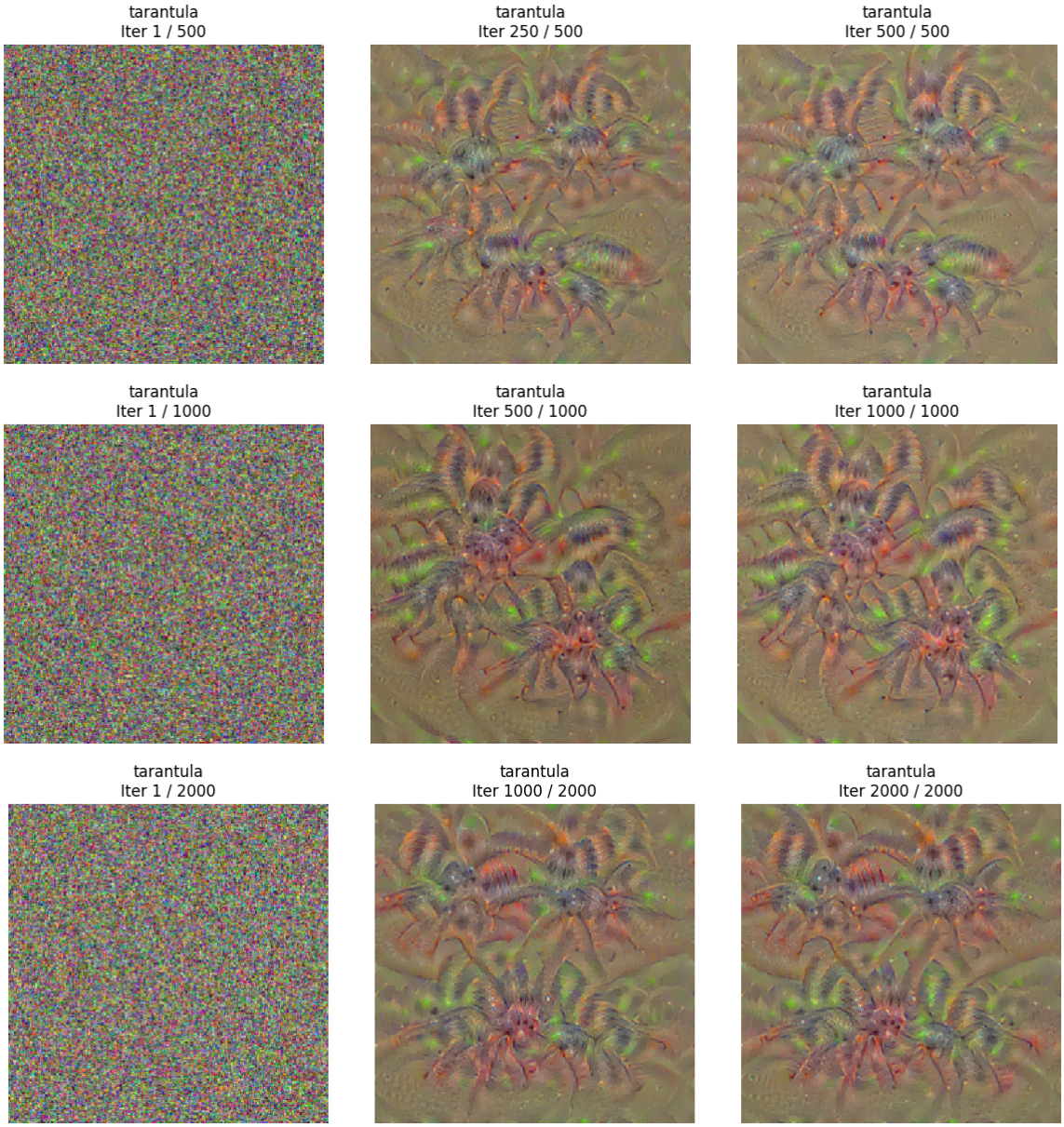


Figure 2.7: Class visualization: Varying the number of iterations (500-1000-2000) with default regularization parameters

In our experiments, we've observed that the quality of the visualizations improves as they go through more epochs, but only up to a point. The key is to find that sweet spot where the image is crisp and detailed before it starts to 'overtrain' and no longer shows any real improvement, which could lead to a kind of visual clutter or noise in the image.

Regularization Weight

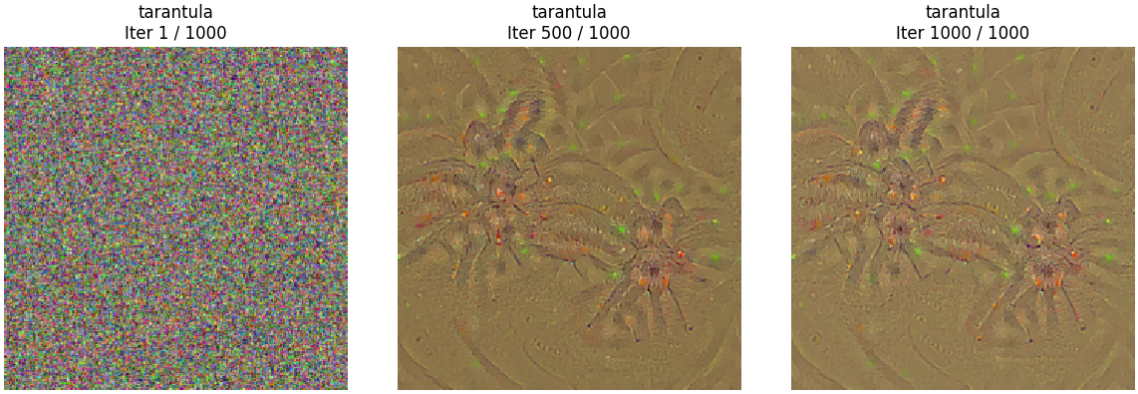


Figure 2.8: Class visualization: Impact of regularization weight (1e-1)

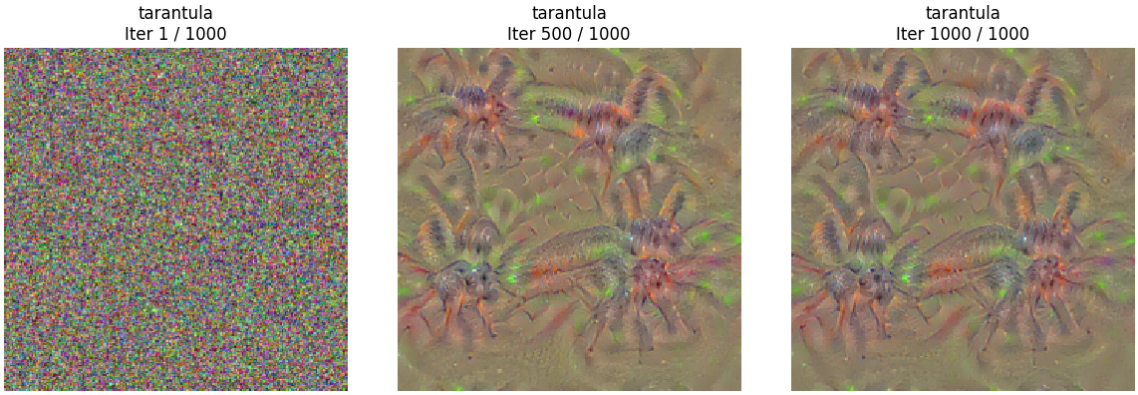


Figure 2.9: Class visualization: Impact of regularization weight (1e-2)

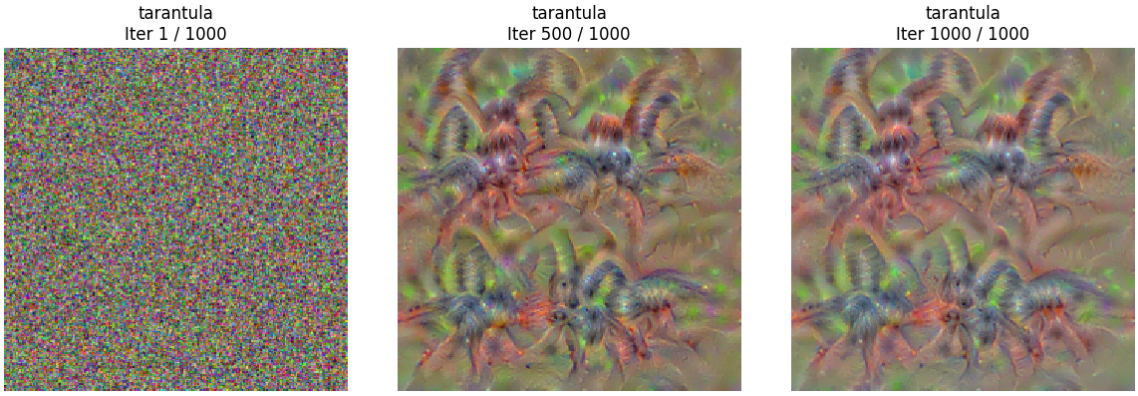


Figure 2.10: Class visualization: Impact of regularization weight (1e-5)

Three distinct regularization weights were tested: 10^{-1}, 10^{-2}, and 10^{-5}.

Results

- **High Regularization** (10^{-1}): Resulted in images with limited feature development, retaining a simplified structure even after 1000 iterations.
- **Moderate Regularization** (10^{-2}): Achieved a harmonious balance, revealing distinct tarantula features with minimal noise, indicating an optimal regularization weight for clear visualizations.
- **Low Regularization** (10^{-5}): Produced more elaborate and vibrant features, but accompanied by increased noise, suggesting a compromise between detail richness and image clarity.

The comparative analysis underscores the importance of regularization weight selection. Moderate regularization (10^{-2}) emerges as the most effective in producing detailed and noise-controlled class visualizations, whereas high regularization is overly constraining, and low regularization allows excessive noise.

Learning rate

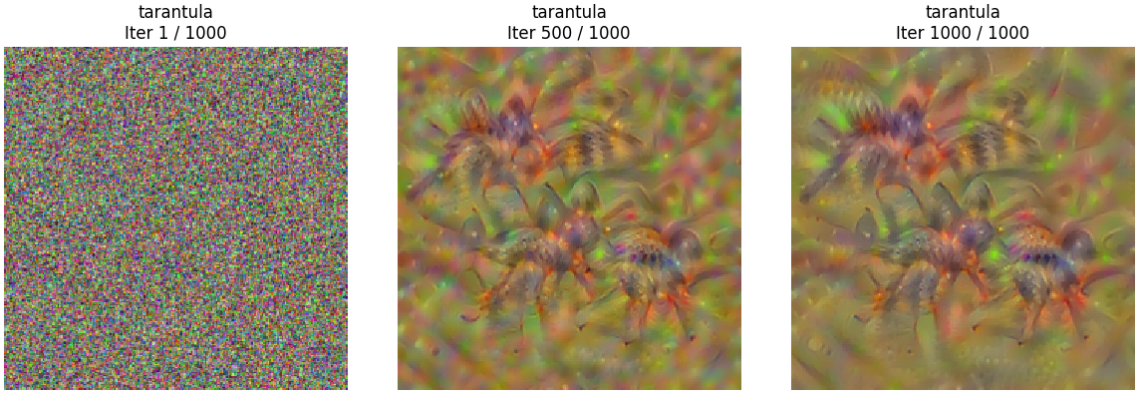


Figure 2.11: Class visualization: Impact of learning rate (1)

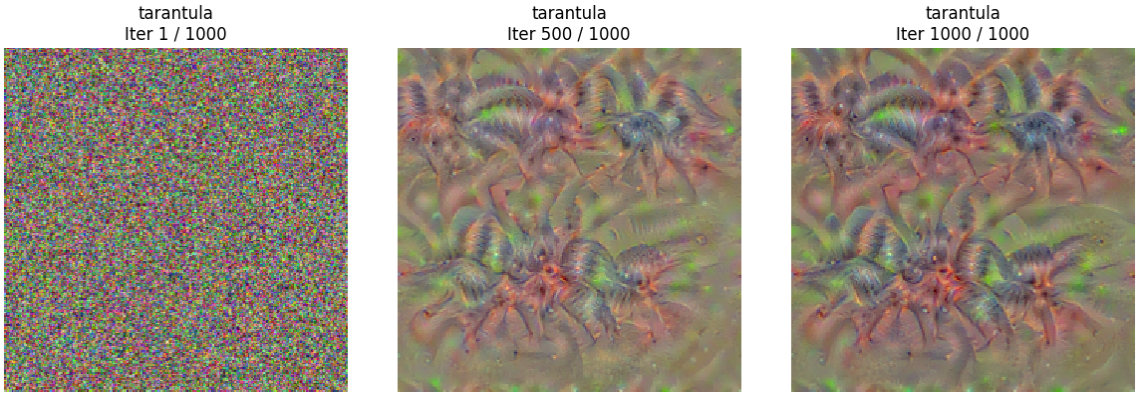


Figure 2.12: Class visualization: Impact of learning rate (5)

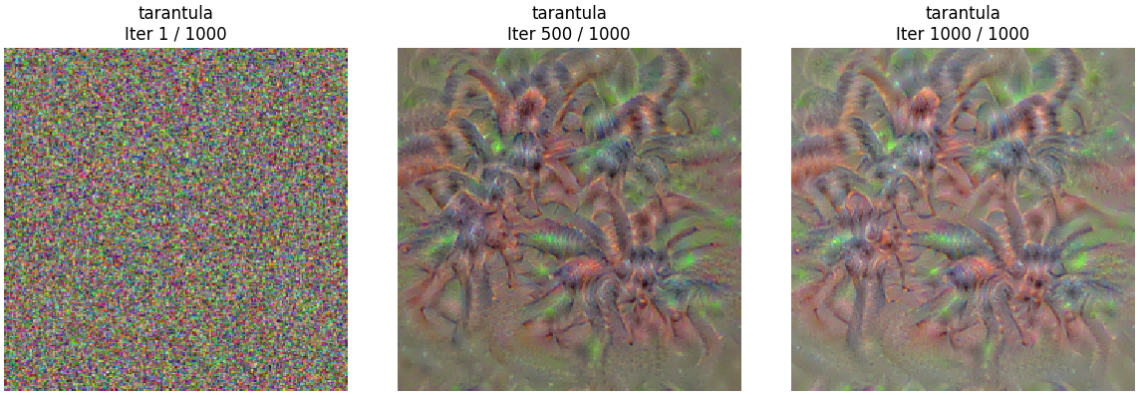


Figure 2.13: Class visualization: Impact of learning rate (9)

The class visualization process was conducted using three distinct learning rates: 1, 5, and 9, each observed over 1000 iterations.

Comparative Analysis

- **Learning Rate 1 (Figure 1.11)** The visualization shows a steady feature evolution, culminating in a structured yet subdued depiction of the tarantula class by the 1000th iteration.

- **Learning Rate 5 (Figure 1.12)** This rate achieved a faster convergence to detailed features, with a clear representation apparent by the 500th iteration and further refined by the 1000th.
- **Learning Rate 9 (Figure 1.13)** At this high rate, features emerged swiftly but possibly at the expense of increased noise, as suggested by the 1000th iteration's image quality.

Conclusion

A comparative assessment reveals that a moderate learning rate (5) offers a pragmatic compromise between the slow yet controlled feature development at a lower rate (1) and the rapid but potentially less stable progression at a higher rate (9).

Blurring factor

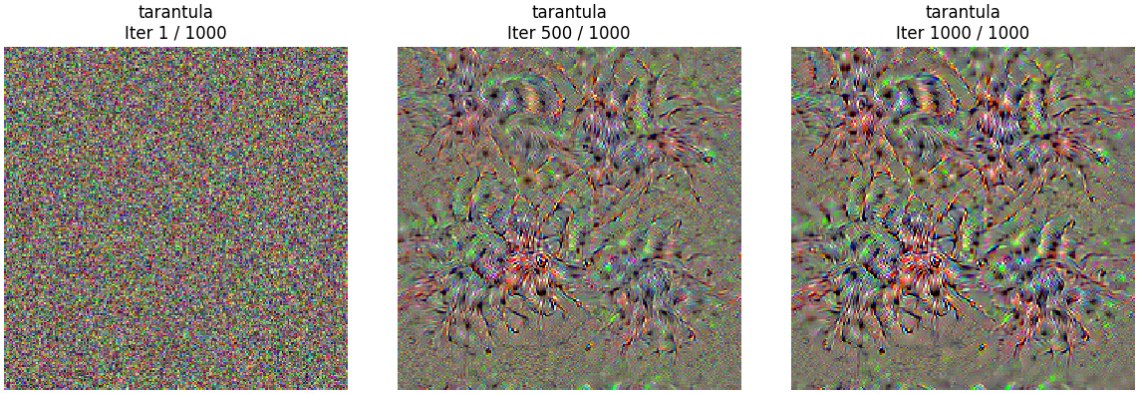


Figure 2.14: Class visualization: No Blurring

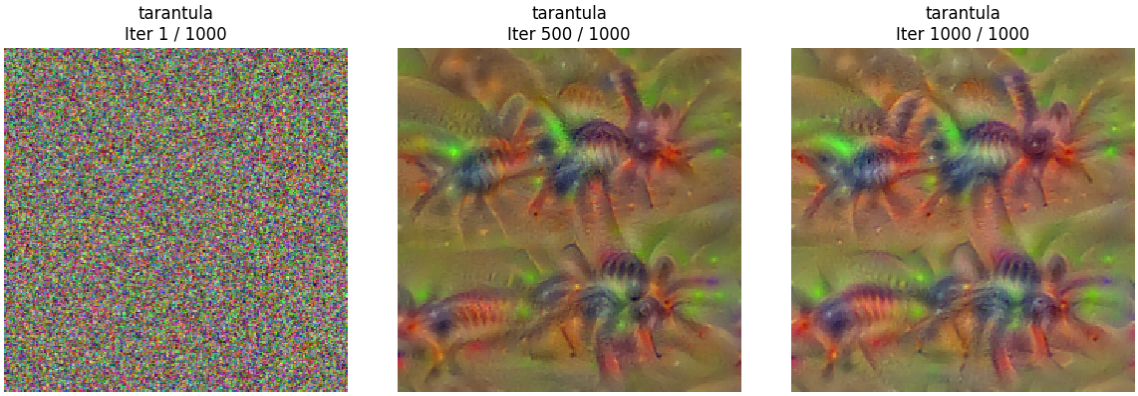


Figure 2.15: Class visualization: Blurring Factor (2)

In our evaluation, the second image, where blurring is applied every two steps, presents more refined and distinct features.

The application of blurring seems to inhibit the spread of redundant patterns surrounding the visualization. By reinforcing the pixels already altered by the model and preserving their trajectory, blurring aids in crafting a more authentic and intricate representation of the class.

2.6.3 Q10. Try to use an image from ImageNet as the source image instead of a random image. You can use the real class as the target class. Comment on the interest of doing this.

Starting with an ImageNet-derived image proves to be an advantageous strategy. It offers a substantive baseline for gradient-driven visualization, guiding the initial gradients toward pertinent features and

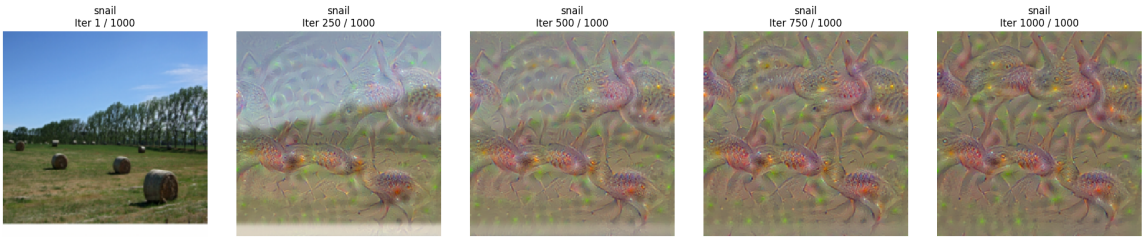


Figure 2.16: Class visualization: Initialization with ImageNet Hay Image, Target : Tarantula

patterns. This method helps to avoid the dispersal of early gradients into random, irrelevant paths, which can lead to a more coherent and decipherable visualization.

2.6.4 Q11. Bonus: Test with another network, VGG16, for example, and comment on the results.

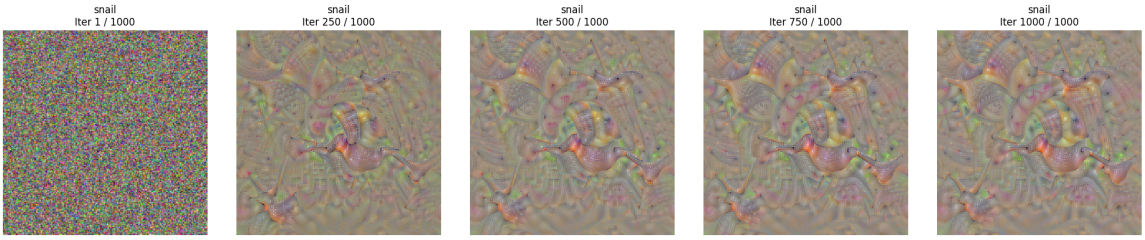


Figure 2.17: Class visualization:Using VGG-16 net with random noise init

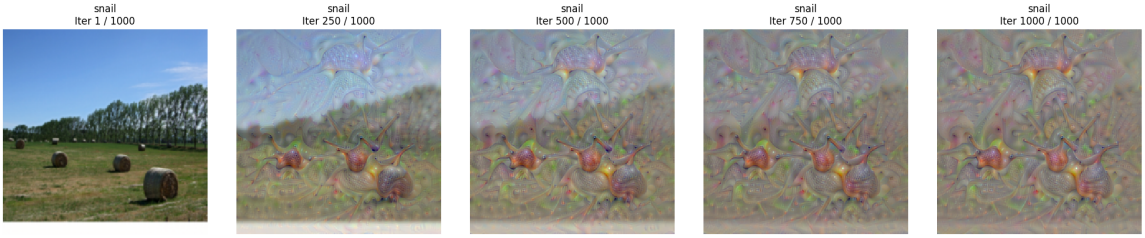


Figure 2.18: Class visualization:Using VGG-16 net with ImageNet init

Visualizations are often superior due to VGG16's overall improved performance.

2.7 Conclusion & Personal Insights

Through these practical exercises, we gained a deeper understanding of the interpretability of CNNs. The visualization techniques employed serve as powerful tools for unraveling the complex decision-making process of these models. They not only aid in comprehension but also reveal areas where models might be lacking, particularly in terms of robustness against adversarial attacks. This knowledge is instrumental in developing more reliable and interpretable models for future applications.

Chapter 3

Domain Adaptation

3.1 Context

In this practical work on Domain Adaptation, we addressed a central challenge in machine learning and computer vision: effectively transferring knowledge from one domain (source) to another (target), where direct application of a model trained on the source domain performs poorly on the target due to differences in data distribution. This scenario is emblematic in real-world applications such as autonomous driving, where models trained on simulated environments must adapt to real street scenarios without extensive retraining.

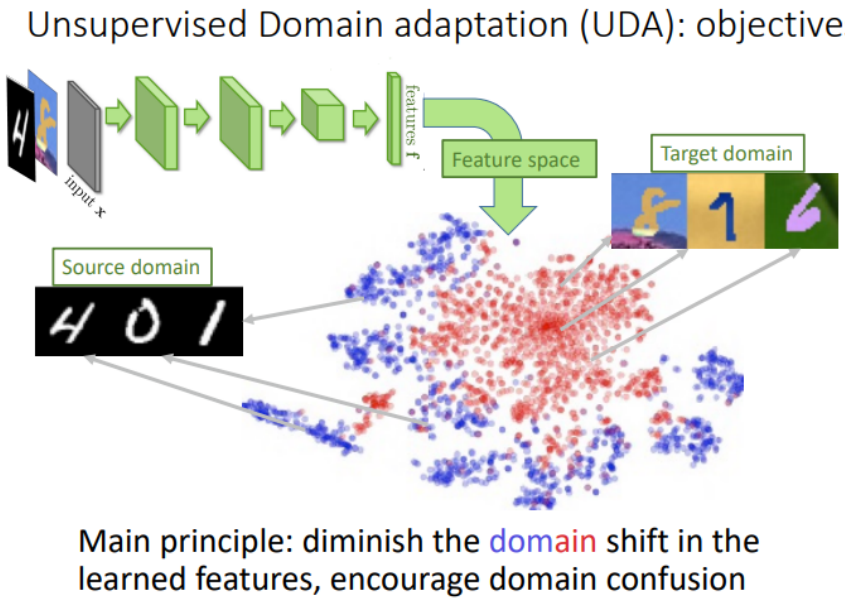


Figure 3.1: Domain Adaptation objective

Our focus was on the MNIST to MNIST-M adaptation, where the goal was to train a model on the well-labeled MNIST dataset and ensure robust performance on the MNIST-M dataset, which features color and texture variations. To achieve this, we explored the Domain-Adversarial Neural Network (DANN), an innovative architecture that introduces a domain classifier and a Gradient Reversal Layer (GRL). The GRL is a novel concept designed to confuse the domain classifier, thereby encouraging the feature extractor to develop domain-invariant features.

Our approach was iterative and experimental. Initially, we replicated the DANN model to understand the baseline performance without domain adaptation. Then, by incorporating the GRL, we observed the shift in performance, noting particularly how and why there might be a slight degradation in source domain accuracy—a sacrifice for greater generalization.

A significant portion of our work was dedicated to fine-tuning: adjusting the number of epochs to allow the model more learning iterations, updating the learning rate more frequently to refine the optimization

landscape, and experimenting with different data normalization techniques to find the most effective input standardization. We also manipulated the GRL's scaling factor to identify the optimal rate at which the model should become domain-agnostic.

The journey didn't end there; our explorations also included alternative techniques such as pseudo-labeling, adding an additional layer of complexity and potential to the adaptation process. This technique uses the model's own predictions on the target domain data to augment training, pushing the boundaries of unsupervised learning.

Ultimately, each component of the practical work—whether it was dissecting the DANN model's architecture, tweaking training hyperparameters, or investigating alternative domain adaptation methods—contributed to a nuanced understanding of the delicate balance between domain specificity and domain agnosticism. The practicals were not just about achieving parity with existing benchmarks but about developing a deep comprehension of the underlying principles that govern domain adaptation.

3.2 DANN and the GRL layer

In this practical session, we tackled domain adaptation using the MNIST (source) and MNIST-M (target) datasets. Our goal was to train a model on labeled MNIST data and evaluate its performance on the unlabeled MNIST-M dataset, which presents a shift in domain characterized by variations in digit color and background.

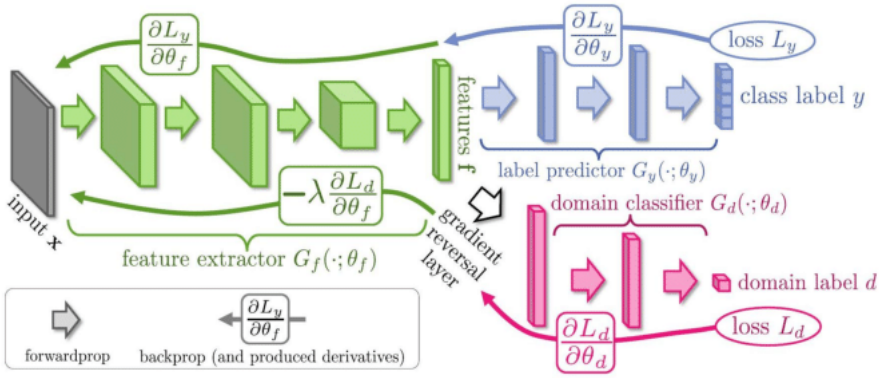


Figure 3.2: DANN Architecture

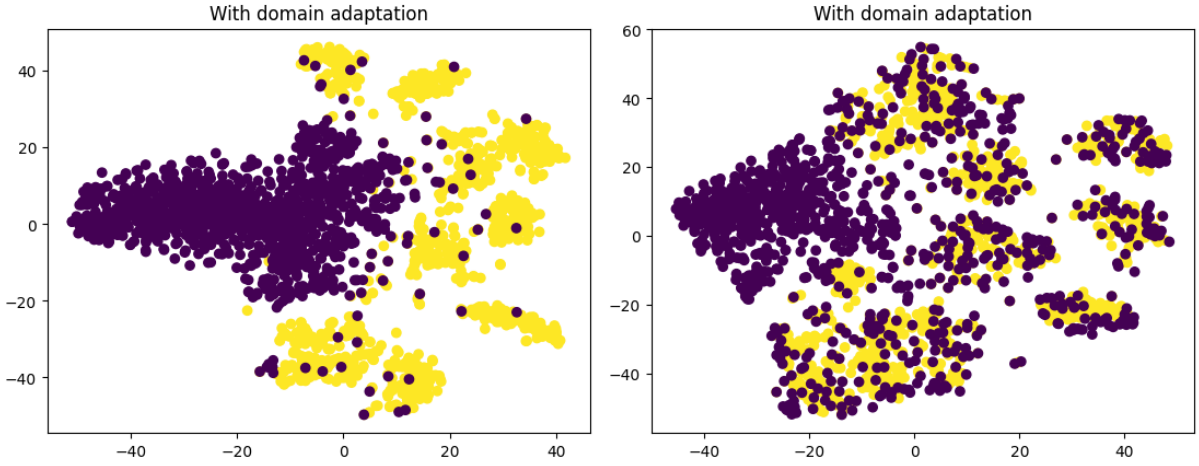
We implemented the Domain-Adversarial Neural Network (DANN) model, following the architecture proposed by Ganin and Lemptisky. This model comprises a feature extractor (convolutional neural network in green), a classifier for the task at hand (blue layers), and a domain classifier (pink layers). The key component of DANN is the Gradient Reversal Layer (GRL), which sits between the feature extractor and the domain classifier. The GRL inverts the gradient during backpropagation, compelling the feature extractor to learn domain-agnostic features. As a result, the model is encouraged to find a representation where the domain is indistinguishable, hence improving generalization to the target domain.

3.3 Practice

3.3.1 Q1. If you keep the network with the three parts (green, blue, pink) but didn't use the GRL, what would happen ?

Without the Gradient Reversal Layer (GRL), the domain classifier became proficient at distinguishing between the source and target domains. This specialization would inadvertently train the feature extractor to emphasize domain-specific features.
As a result, the neural network is optimized for the source domain, achieving high accuracy there. However, this specialization is counterproductive for domain adaptation, leading to poor performance on the target domain as the learned features do not generalize well.
The absence of the GRL would thus defeat the purpose of domain adaptation, which is to learn domain-invariant features that perform equally well on both source and target domains.

Experimental Study.



(a) Without GRL (b) With GRL

Figure 3.3: DANN Applied with/without GRL Layer.

With GRL:

- Source Domain: The class accuracy is high (98.42%), but not perfect. The domain accuracy is significantly lower (80.54%), indicating that the feature extractor is not emphasizing features unique to the source domain, which aligns with the goal of domain adaptation. The domain loss is relatively high (0.51769), reflecting this confusion.
- Target Domain: The class accuracy is reasonably good (73.56%), a significant achievement considering this domain's data were not used for labeled training. The domain accuracy is close to the source's (75.24%), suggesting that the model does not strongly differentiate between domains, supporting the notion of domain-agnostic feature learning. The domain loss is lower (0.45681) compared to the source, indicating effective confusion in domain classification.

Without GRL:

- Source Domain: Class accuracy is near perfect (99.14%), and domain accuracy is absolute (100%). This suggests the model is highly specialized in identifying and classifying source domain features.

The domain loss is almost zero (0.00004), indicating the model is confidently and correctly identifying the source domain.

- Target Domain: There is a significant drop in class accuracy (41.79%), showing the model's inability to generalize to the target domain. The domain accuracy is extremely high (99.96%), with a very low domain loss (0.00075), which means the model can almost perfectly identify the target domain as being different from the source, but this does not translate into effective digit classification.

Analysis

- Impact of GRL: The inclusion of the GRL disrupts the model's ability to learn domain-specific features. This is evident from the lower domain accuracies and higher domain losses when the GRL is present. The model with GRL learns features that are more generalizable, as indicated by the much higher target domain classification accuracy.
- Trade-off: There is a trade-off observed in using GRL. While it decreases the performance slightly on the source domain, it significantly boosts performance on the target domain, which is the primary goal of domain adaptation.
- Generalization vs. Specialization: Without GRL, the model becomes highly specialized to the source domain, leading to excellent performance there but poor generalization to the target domain. The GRL encourages the model to learn more generalized features, beneficial for performance on the target domain.

In conclusion, the GRL plays a pivotal role in domain adaptation by encouraging the model to learn features that are not specific to the source domain, thus enhancing the model's ability to generalize to the target domain. This experimental evidence strongly supports the theoretical underpinnings of the GRL's role in domain-adaptive models.

3.3.2 Q2. Why does the performance on the source dataset may degrade a bit ?

In domain adaptation, particularly in models employing a Gradient Reversal Layer (GRL), a slight performance degradation on the source dataset is often observed. This phenomenon can be attributed to the following factors:

Trade-off Between Generalization and Specialization The primary goal of domain adaptation is to enable the model to generalize well to a target domain, which inherently involves a trade-off. As the model learns to become more domain-agnostic:

- It tends to lose some of the specialized capabilities that were solely effective on the source domain.
- This loss in specialization can manifest as a minor drop in performance metrics like accuracy on the source dataset.

Feature Extractor's Role The feature extractor in a domain-adaptive model plays a crucial role:

- It is trained to extract features that are relevant for both the source and target domains, diverging from the typical approach of learning features that are highly specific to the source domain.
- This shift in focus from domain-specific to domain-invariant feature extraction is what causes the model to be slightly less effective on the source domain, albeit more adaptable to the target domain.

Implications Understanding this trade-off is vital in domain adaptation:

- It guides the model's training process, emphasizing the balance between retaining enough discriminative power for the source domain and achieving adequate generalization for the target domain.
- It also informs the iterative process of model refinement, where adjustments in architecture or training parameters are made to achieve the desired balance.

Conclusion The slight performance degradation on the source dataset during domain adaptation is a consequence of the model's increased focus on generalization. This understanding is essential for effectively training domain-adaptive models and achieving optimal balance between source domain performance and target domain generalization.

3.3.3 Q3. Discuss the influence of the value of the negative number used to reverse the gradient in the GRL.

The Gradient Reversal Layer (GRL) plays a crucial role in domain adaptation by imposing 'confusion' on the domain classifier. This is controlled by its negative scaling factor. The following points elaborate on its impact:

Function of the Scaling Factor The GRL's scaling factor negatively scales the gradients during the backpropagation process. Its absolute value determines the intensity of this effect:

- A **higher absolute value** results in stronger penalization of domain-specific feature learning, nudging the feature extractor towards domain-agnostic features.
- Conversely, a **lower absolute value** may not sufficiently encourage the model to disregard domain-specific characteristics.

Consequences of Scaling Factor Magnitude The magnitude of the scaling factor influences the model's learning dynamics:

- **Too Large**: An excessively large factor can destabilize training, leading to underfitting in both source and target domains. This occurs because the strong negative scaling overly diminishes the importance of domain-specific features, which are still somewhat relevant for accurate classification.
- **Optimal Balance**: An appropriately calibrated scaling factor achieves a balance, promoting the learning of features that are informative for the classification task yet invariant across domains.

Implications for Model Performance The choice of the scaling factor is thus a critical decision in model design:

- It necessitates a careful balance to ensure the model is encouraged to learn representations that are generalizable across domains without losing its ability to capture relevant features for the classification task.
- This balance is key to achieving effective domain adaptation.

Conclusion The GRL's negative scaling factor significantly impacts the efficacy of domain adaptation models. Its careful calibration is essential to enable the model to learn domain-invariant features while maintaining performance on the classification task, embodying the core challenge of domain adaptation.

3.3.4 Q4. Another common method in domain adaptation is pseudo-labeling. Investigate what it is and describe it in your own words.

Pseudo-labeling, as a technique in domain adaptation, harnesses the model's own predictions on unlabeled data for further training. This semi-supervised approach is particularly advantageous in scenarios with limited labeled data in the target domain.

Mechanics of Pseudo-Labeling

The process of pseudo-labeling unfolds in the following steps:

1. The model, initially trained on a labeled source domain, is applied to the unlabeled target domain data.
2. Predictions made by the model on the target data are converted into 'pseudo-labels,' especially those predictions made with high confidence.
3. These pseudo-labels are then used as if they were true labels to retrain the model, thus refining its ability to classify data from the target domain.

Advantages and Strategic Importance

The key benefits and strategic importance of pseudo-labeling include:

- **Data Utilization**: It optimizes the use of available data, especially when labeled data in the target domain is scarce.
- **Model Refinement**: By iteratively training with pseudo-labels, the model continually enhances its understanding and adaptability to the target domain.
- **Improving Generalization**: This technique helps bridge the gap between source and target domains, thereby improving the overall generalization of the model.

Conclusion

Pseudo-labeling emerges as a powerful tool in domain adaptation, enabling models to better transfer learned knowledge from a source to a target domain. Its effectiveness lies in creating a dynamic learning environment where the model self-improves by learning from its own high-confidence predictions.

Chapter 4

Generative Adversarial Networks (GANs)

4.1 Introduction

Generative Adversarial Networks (GANs), since their introduction in 2014, have revolutionized the field of computer vision. Moving beyond traditional classification models that focus on predicting labels from inputs ($P(Y|X)$), GANs explore the joint distribution of data and labels ($P(X,Y)$). This shift towards generative models has significant implications in image generation and editing, allowing for the creation of realistic images from limited or no labeled data.

4.2 Context and Problem Statement

The traditional focus in computer vision has been on discriminative models requiring extensive labeled datasets. This approach is limiting due to the scarcity and expense of obtaining such data. GANs, however, offer a novel solution by generating new, realistic data samples. A standard GAN generates an image from a noise vector (z), while its extension, the conditional GAN (CGAN), incorporates an additional input (y) like a class label or image for more controlled generation. This capability of GANs not only pushes the boundaries of artificial intelligence in image creation but also raises important ethical questions about the authenticity and use of synthetic images.

4.3 Roadmap

This roadmap provides a concise overview of our practical study on Generative Adversarial Networks (GANs) and Conditional GANs (CGANs). The practical is structured into two main sections.

1. **GANS:** In the first section, we focus on GANs. We begin by addressing common questions about their fundamental principles and architecture. Our exploration starts with training a Deep Convolutional GAN (DCGAN) on the MNIST dataset, following the default settings from the DCGAN paper. Subsequently, we delve into the impact of various hyperparameters on the model's performance, including ngf (number of feature maps in the generator), ndf (number of feature maps in the discriminator), and n_z (size of the input noise vector). We examine the effects of different initialization methods for DCGAN weights, comparing the default PyTorch initialization with the custom method proposed in the DCGAN paper. Additionally, we experiment with replacing the generator's loss function with the original GAN equation and explore linear interpolation within the GAN framework. Our experimentation extends to other datasets, such as CIFAR10 and CelebA.

2. **Conditional GANS:** In the second section, we shift our focus to Conditional GANs (CGANs). We start with a brief introduction to CGANs, highlighting their unique features. This is followed by a detailed discussion on their implementation and experimentation on the MNIST and CelebA datasets.

4.4 Section 1.1: General Principle of GANs

4.4.1 Q1. Interpret the equations (6) and (7). What would happen if we only used one of the two ?

Equations (6) and (7) are key components of the training dynamics in Generative Adversarial Networks (GANs). They represent the loss functions for the generator (G) and discriminator (D), respectively. Let's interpret each equation and discuss the implications of using only one of them.

1. **Equation (6): Generator's Objective**

$$\max_G \mathbb{E}_{z\sim P(z)}[\log D(G(z))]$$

   - **Goal:** This equation represents the generator's objective in a GAN. The generator, G, tries to generate data that is indistinguishable from real data.
   - **Mechanism:** It does so by taking a noise vector z (sampled from some probability distribution $P(z)$) and transforming it into a sample $G(z)$ that resembles the real data.
   - **Objective:** The generator aims to maximize the probability of the discriminator, D, being mistaken, i.e., the discriminator classifying the fake data $G(z)$ as real. In essence, $\log D(G(z))$ becomes larger when D is more likely to classify the fake samples as real.

2. **Equation (7): Discriminator's Objective**

$$\max_D \mathbb{E}_{x^*\in \text{Data}}[\log D(x^*)] + \mathbb{E}_{z\sim P(z)}[\log(1 - D(G(z)))]$$

   - **Goal:** This equation represents the discriminator's objective in a GAN. The discriminator, D, attempts to correctly classify real data x^* (from the dataset) and generated data $G(z)$ (from the generator).
   - **Mechanism:** The discriminator learns to maximize the probability of correctly identifying real and fake samples.
   - **Objective:** The first term, $\mathbb{E}_{x^*\in \text{Data}}[\log D(x^*)]$, encourages D to recognize real data (increasing its output for real samples), while the second term, $\mathbb{E}_{z\sim P(z)}[\log(1 - D(G(z)))]$, encourages D to recognize fake data generated by G (decreasing its output for fake samples).

3. **Implications of Using Only One Equation:**

   - **Using Only Equation (6):** If the training process includes only the generator's objective without the discriminator, the generator would lack feedback on the quality of its output. There would be no mechanism to compare the generated data against real data, leading to the generator not learning effectively how to create realistic data.
   - **Using Only Equation (7):** Without the generator, the discriminator's training would also be meaningless. The discriminator is meant to distinguish between real and generated data, and without the generator producing fake data, it would only learn to always output a high probability for any input, assuming it to be real. It would not learn to discriminate effectively, as there would be no varying data to challenge its decision-making capability.

In a GAN framework, both equations are crucial and interdependent. The generator and discriminator are trained simultaneously in a competitive setup where each one's improvement provides a learning signal for the other. This dynamic, adversarial relationship is what allows GANs to generate highly realistic data and makes them effective for tasks like image generation, style transfer, and more.

4.4.2 Q2. Ideally, what should G transform the distribution $P(z)$ to ?

In a GAN framework, the generator G aims to transform the input noise distribution $P(z)$ into a distribution that mirrors the real data distribution $P_{\text{data}}(x)$. Mathematically, this transformation is represented as $G : z \sim P(z) \to x \sim P_{\text{data}}(x)$. The goal is for the generated data distribution $P_G(x)$, where $x = G(z)$, to approximate $P_{\text{data}}(x)$ as closely as possible. This ideal outcome implies that the generator has successfully learned to produce synthetic data indistinguishable from real data.

4.4.3 Q3. Remark that the equation (6) is not directly derived from the equation 5. This is justified by the authors to obtain more stable training and avoid the saturation of gradients. What should the "true" equation be here ?

In the original Generative Adversarial Network (GAN) framework, the generator's objective is often stated as:

$$\min_G \mathbb{E}_{z \sim P(z)}[\log(1 - D(G(z)))]$$

However, this formulation presents a challenge at the early stages of training, specifically related to the behavior of the discriminator D when the generator G is not yet proficient at generating realistic images. The issue can be explained as follows:

- At the beginning, G is inefficient at generating images that resemble the real data, meaning D will easily classify them as fake. This situation implies that $D(G(z)) \approx 0$.
- The derivative of the original objective function, considering $\sigma = 1 - D(G(z))$, is:

  $$\frac{d}{d\theta_G} \log(\sigma) = \frac{1}{\sigma} \cdot \frac{d\sigma}{d\theta_G}$$

  As $D(G(z)) \to 0$, we find that $\sigma \to 1$, leading to:

  $$\frac{d}{d\theta_G} \log(\sigma) \approx \frac{d\sigma}{d\theta_G}$$

  Given that $D(G(z)) \approx 0$, $\frac{d\sigma}{d\theta_G} \approx 0$, resulting in:

  $$\frac{d}{d\theta_G} \log(1 - D(G(z))) \approx 0$$

- This results in a vanishing gradient issue for the generator. The generator receives little to no feedback for improvement, causing it to be stuck in the initial state of generating poor samples.

To mitigate this problem, an alternative objective is often used in practice:

$$\max_G \mathbb{E}_{z\sim P(z)}[\log D(G(z))]$$

or equivalently,

$$\min_G -\mathbb{E}_{z\sim P(z)}[\log D(G(z))]$$

This alternative objective provides stronger gradients for the generator, especially early in training, and helps to avoid the vanishing gradient problem.

4.5 Section 1.2: Architecture of GANs and their implementation

Deep Convolutional Generative Adversarial Networks (DCGANs) are a class of GANs that primarily use convolutional neural networks (CNNs) for both the generator and discriminator. DCGANs were introduced by Alec Radford, Luke Metz, and Soumith Chintala in their 2015 paper titled "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". This innovative architecture made several key contributions to the field of generative modeling. It established a set of architectural guidelines for stable training of GANs, emphasizing the use of strided convolutions in the discriminator and fractional-strided convolutions in the generator, batch normalization in both the generator and discriminator, and the use of the ReLU activation function in the generator (except for the output layer which uses Tanh) and LeakyReLU in the discriminator. These modifications significantly improved the quality and stability of the generated images and allowed for more complex and detailed generation tasks, making DCGANs a foundational architecture in the field of deep learning and generative models.

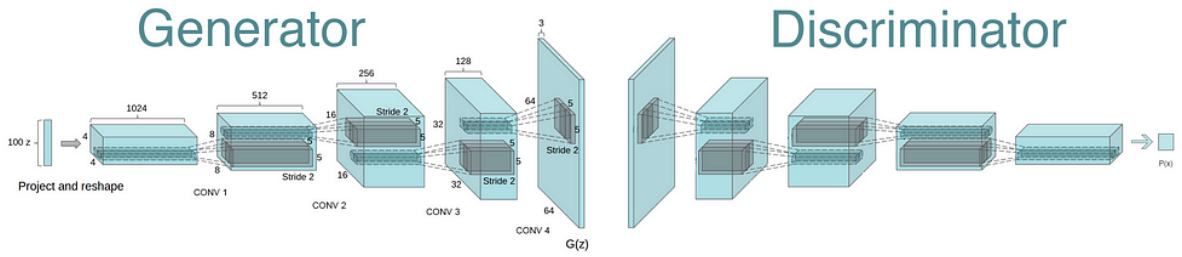


Figure 4.1: DCGAN Architecture

Note: For all the experiments that will be conducted in the upcoming parts, we will consistently adopt the DCGAN architecture, following the specific configuration and hyperparameter settings documented in the DCGAN research paper and the guidelines mentioned in the practical notebook. This includes parameters like the learning rate, alpha values for the optimizer, and other relevant architectural choices as specified in those references.

4.5.1 Training the GAN with default settings

We will first train the GAN with the following default settings using the MNIST dataset:

- n_z = **100**: Size of z latent vector (i.e., size of generator input).
- $batch_{size}$ = **128**: Number of images per batch
- ngf = **32**: Size of feature maps in generator.
- ndf = **32**: Size of feature maps in discriminator.
- lr_d = **0.0002**: Learning rate for discriminator.
- lr_g = **0.0002**: Learning rate for generator.
- β_1 = **0.5**: β_1 hyperparam for Adam optimizers.

- $n_{channels} = \mathbf{1}$: Since we're using the MNIST dataset.

After 10 epochs, we obtain the following results:

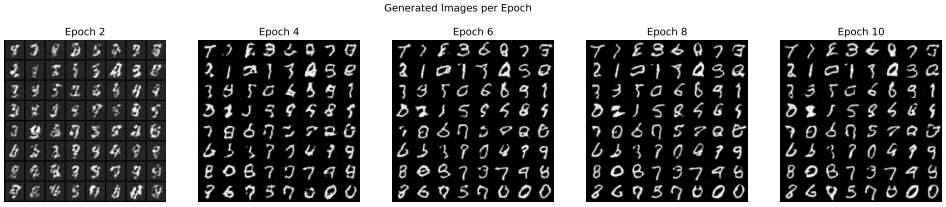


Figure 4.2: Generated images per epoch (every 2 epochs)

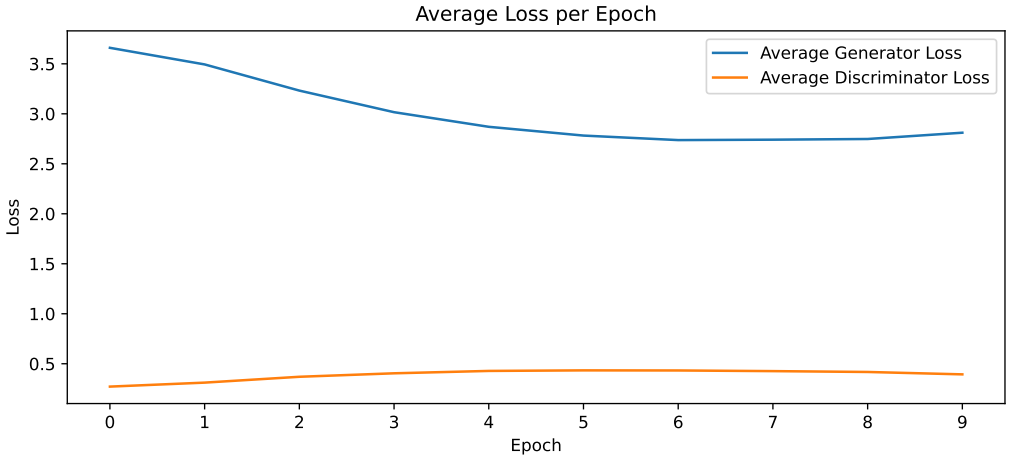


Figure 4.3: Average loss per epoch ($n_{epochs} = 10$)

Observations:: Analyzing the images produced across various epochs reveals an intriguing trend: while the generator has shown improvement in generating certain digits, its performance on others has diminished. A notable observation is the inverse relationship between the losses of the generator and discriminator – as the generator's loss decreases, the discriminator's loss tends to increase. This suggests that the generator is on the right path of learning to produce more realistic synthetic data, but it's challenging to conclusively determine its limits without extending the training duration. The generated images indicate potential for further enhancement, hinting at the necessity of experimenting with different parameters and training the model for longer periods to optimize the its performance.

4.5.2 Effects of ngf and ndf

In this experiment, we will decrease the number of feature maps in one component of the GAN while substantially increasing it in the other component. This will allow us to observe and analyze the impact of such a drastic change on the training process.

- $ngf = \mathbf{256}$ | $ndf = \mathbf{8}$ **:** The main consequence of such an imbalance is that the discriminator becomes too weak to effectively challenge and critique the generator's output. As a result, the generator may not receive adequate feedback to push its learning boundaries further. This lack of rigorous challenge can hinder the generator's ability to improve and diversify its outputs, potentially

leading to a stagnation in the quality of the generated data. To further confirm this, here are the results of the training process using the specified ngf and ndf values:

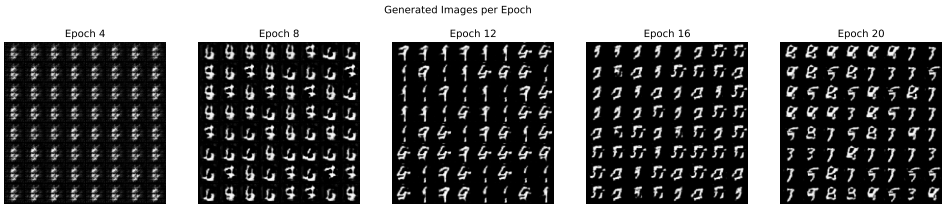


Figure 4.4: Generated images per epoch (every 4 epochs)

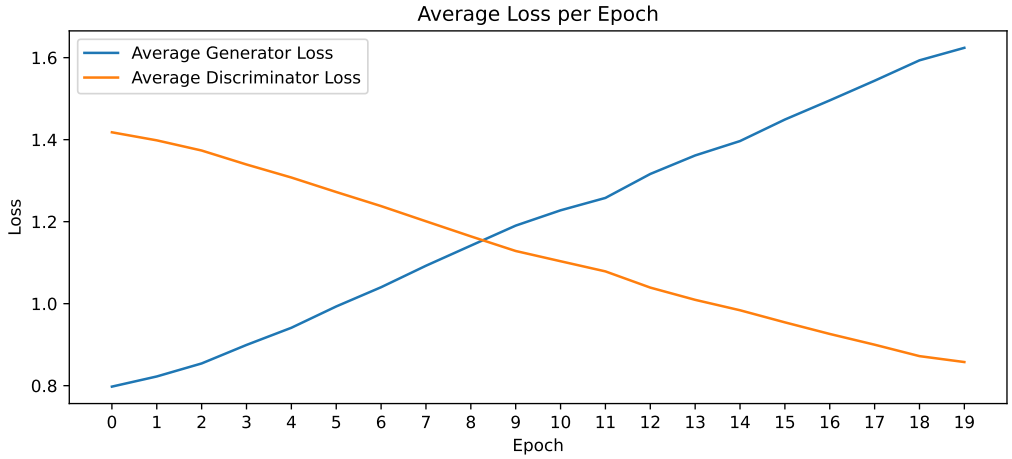


Figure 4.5: Average loss per epoch ($n_{epochs} = 20$)

Observations: It is evident that the training process is not progressing as desired. Instead of the expected scenario where the generator loss decreases while the discriminator loss increases, we are observing the opposite outcome. This suggests that the generator is struggling to produce realistic images, primarily because it is not being sufficiently challenged by the discriminator. Hence, it is crucial to strike a balance in the number of feature maps between the generator and discriminator. Excessive feature maps in the generator or a significant deficit of feature maps in the discriminator can hinder the training process, preventing the generator from improving its data generation and potentially leading to catastrophic learning.

Another noteworthy observation is that as the training progresses, the generated images from the generator appear to become increasingly uniform and less diverse. This phenomenon is commonly referred to as **"mode collapse"**, where the generator seems to believe it can deceive the discriminator by producing images that belong to a limited subset of classes, disregarding the full diversity present in the training dataset.

- $ngf = 8$ | $ndf = 256$ **:** This gives the discriminator more capacity to distinguish between real and fake images. A stronger discriminator can force the generator to improve its output quality. However, if the discriminator becomes too strong (ndf much bigger than ngf), it might easily reject generator outputs without giving it a chance to learn effectively. To further confirm this, here are the results of the training process using the specified ngf and ndf values:

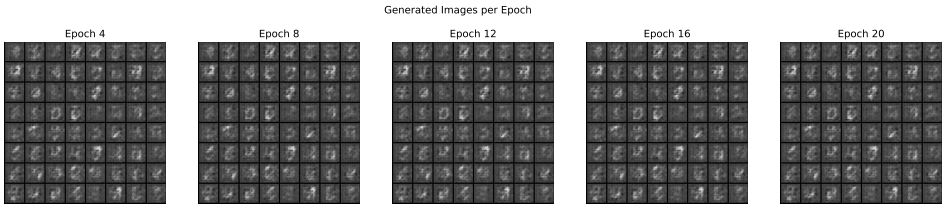


Figure 4.6: Generated images per epoch (every 4 epochs)

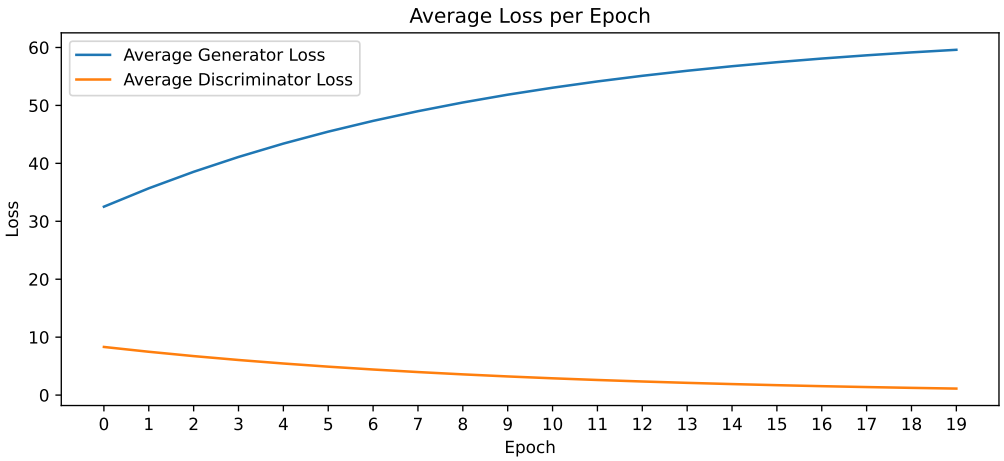


Figure 4.7: Average loss per epoch ($n_{epochs} = 20$)

Observations: It's evident that the discriminator's strength is overwhelming the generator's learning process. This is notably reflected in the generated images, which seem to remain trapped in their initial noise vector state, unable to evolve or improve.

4.5.3 Changing the weight initialization

In the DCGAN paper, a specific weight initialization strategy was adopted instead of using the default PyTorch initialization for several reasons:

1. **Stability in Training**: GANs are known for being hard to train due to issues like mode collapse, vanishing gradients, and failure to converge. The choice of weight initialization can significantly affect the stability and convergence of GAN training. A careful initialization strategy helps in stabilizing the learning process.

2. **Normalization Technique**: The DCGAN paper recommended initializing all weights from a zero-centered Normal distribution with a standard deviation of 0.02. This specific initialization is a form of small random weights initialization, which helps in breaking the symmetry in the learning process and ensures that each neuron in the network initially contributes a small but random effect, preventing layers from getting stuck during the learning process.

3. **Consistency Across Layers**: By initializing weights in a certain way, it ensures consistency across different layers of the network. This consistency is crucial for deep networks like DCGANs, where multiple layers of convolutions and transposed convolutions are stacked together.

4. **Preventing Overshooting and Vanishing Gradients**: Proper initialization helps in preventing the gradients from becoming too large (exploding) or too small (vanishing) during the initial stages of training. Both of these issues can severely hinder the learning process.

5. **Empirical Evidence**: The choices in the DCGAN paper were based on empirical evidence. The authors experimented with various architectures and settings, and the recommendations were those that led to stable and successful training in their experiments.

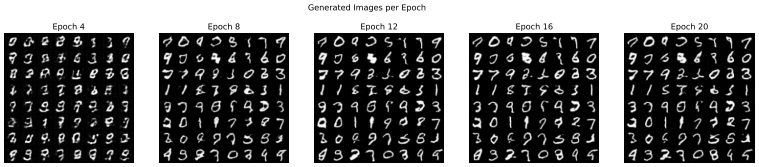


Figure 4.8: Generated images per epoch (every 4 epochs - **Default PyTorch initialization**)

Figure 4.9: Generated images per epoch (every 4 epochs - **Paper initialization**)

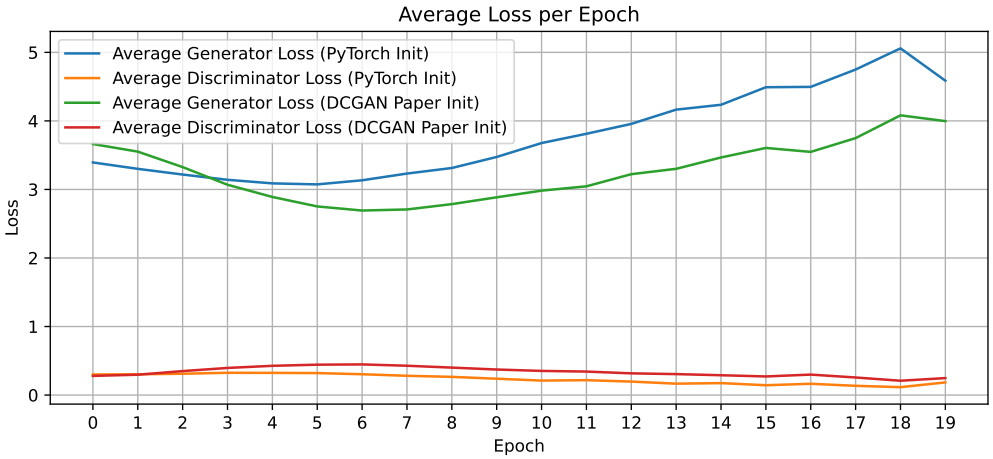


Figure 4.10: Average loss per epoch ($n_{epochs} = 20$)

Observations: It is evident from our experiments that when using the weight initialization recommended by the authors, the generator's loss is consistently lower compared to the default PyTorch initialization. On the other hand, the discriminator's losses remain largely similar, with a slight increase when using the custom initialization. These observations align with the extensive experiments conducted by the authors of the DCGAN paper. The custom weight initialization method generally provides more stable learning outcomes, which is of paramount importance given the notorious difficulty of training GANs.

4.5.4 Replacing the Generator's Training Loss with the Original Loss

In this section, we aim to replace the loss function that the generator minimizes with the original one, as described in question 3 under section 1.1 "General Principle of GANs".
In practice, the generator updates its weights using Equation (6), the generator's objective:

$$\max_G \mathbb{E}_{z\sim P(z)}[\log D(G(z))]$$

The reason for using this objective rather than the original part of the GAN objective function:

$$\min_G \mathbb{E}_{z\sim P(z)}[\log(1 - D(G(z)))]$$

is due to the vanishing gradient problem at initialization. To confirm the theoretical details provided in that answer, we conduct an experiment by modifying the generator's loss function to optimize the original equation. We kept all other parameters at their default values, only changing the loss function that the generator minimizes. The obtained results are as follows:

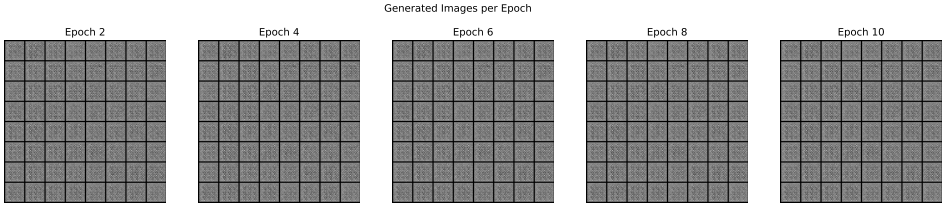


Figure 4.11: Training GAN using original loss

Observations: As expected, the GAN is unable to learn anything and remains stuck in complete and utter noise. This experiment demonstrates that the original equation is not suitable in practice, despite its theoretical foundation.

4.5.5 Impact of Noise Vector Dimension n_z

In their study titled "Effect of Input Noise Dimension in GANs" (2020), Padala et al. investigated the influence of the noise vector's dimension on the performance of Generative Adversarial Networks (GANs). They concluded that the dimension of the input noise vector plays a significant role in the generation of images. To achieve high-quality data generation, the choice of n_z should be tailored to the dataset and the specific GAN architecture being used.
To further validate these findings, we conducted experiments with the DCGAN architecture while varying the n_z values, specifically setting them to 2, 10, and 1000. We then visualized the generated images for each n_z value.

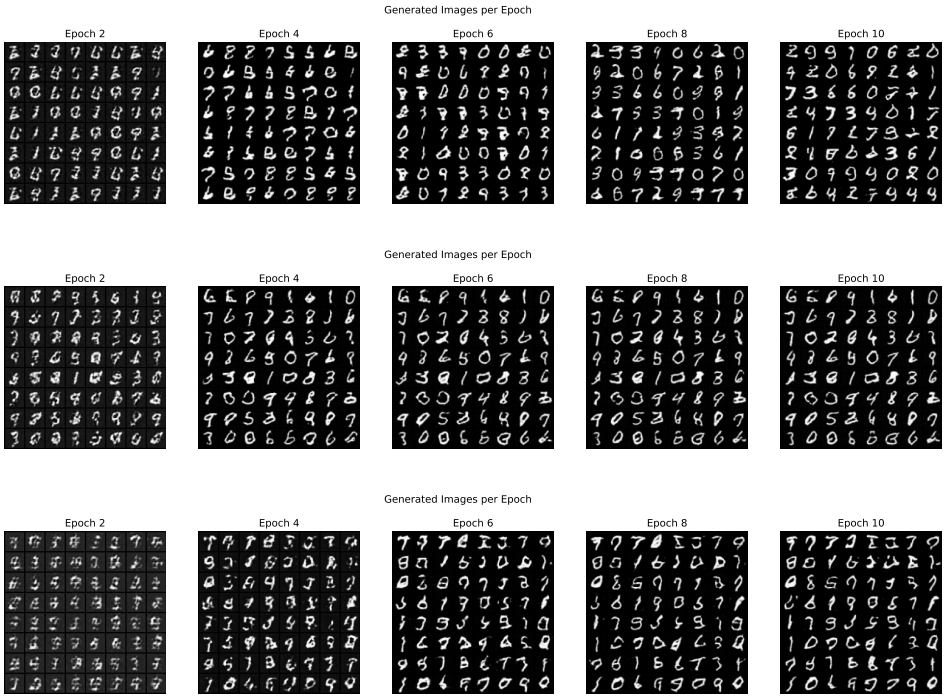


Figure 4.12: Generated images per epoch (every 2 epochs) | n_z **= 2, 10, 1000 respectively**

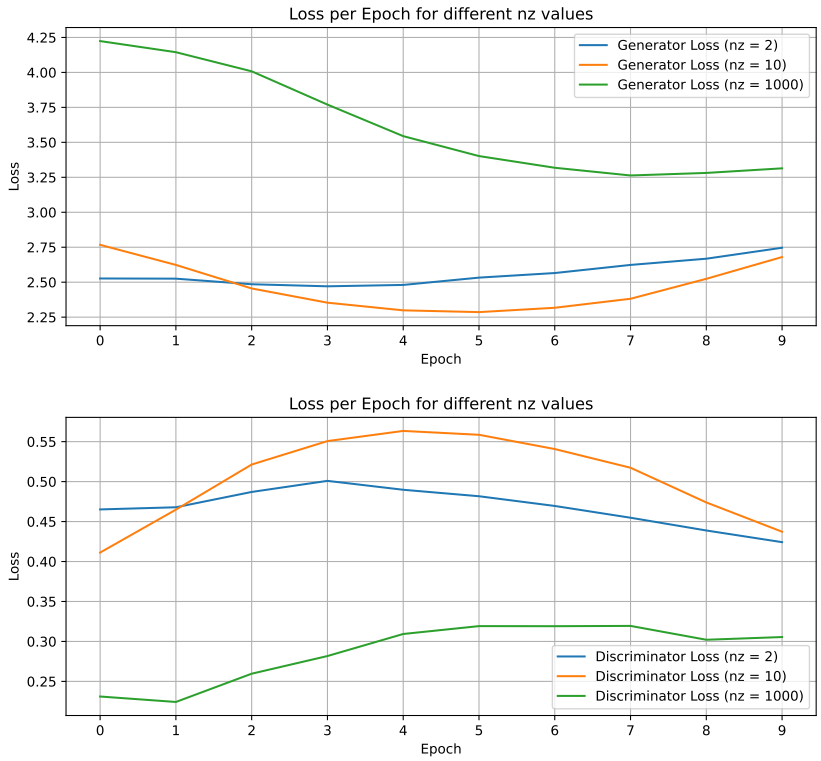


Figure 4.13: G and D loss per epoch for different n_z values ($n_{epochs} = 10$)

Observations: The results exhibit significant variations in terms of loss and generated images. Particularly in this example, $n_z = 2$ seems to give the best results, whereas $n_z = 1000$ gives the worst, but establishing a straightforward correlation between n_z values and the quality of results is not evident. It

is worth noting that the commonly used value of $n_z = 100$ often yields satisfactory results in many GAN applications.

Conclusion : The size of the noise vector in a GAN can exert a notable influence on its performance and the characteristics of the generated data. However, the relationship is not always linear, and its impact varies depending on factors such as dataset complexity and GAN architecture.

4.5.6 Linear interpolation with GANs

In this section, we focused on exploring the latent space of a GAN through the technique of linear interpolation between two noise vectors, z_1 and z_2. The process is as follows:

1. **Creation of Noise Vectors (z_1 and z_2):** Two distinct random noise vectors, z_1 and z_2, were generated ($n_z = 100$). These vectors act as seeds for the GAN's generator to create data samples and represent unique points in the model's latent space.

2. **Linear Interpolation Formula:** We used the formula $\alpha z_1 + (1 - \alpha)z_2$, where α varies from 0 to 1. This formula creates a series of intermediate vectors between z_1 and z_2. Each vector represents a blend of $z1$ and $z2$, with α controlling the extent of influence from each vector.

We obtain the following results:

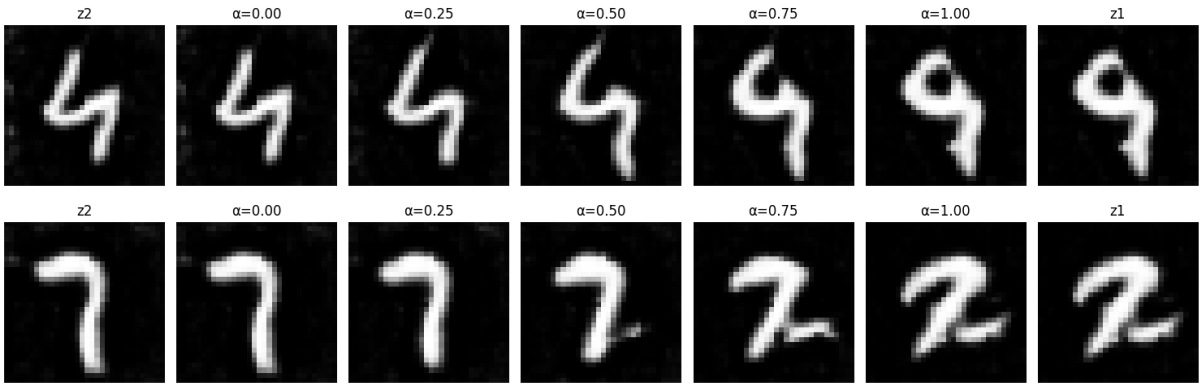


Figure 4.14: **Linear interpolation for varying α values:** z_1 on the right, z_2 on the left.

Observations: These results provide an evident demonstration of the **continuity** and **structure** within the model's **latent space.** This continuity is a crucial aspect of a well-trained GAN, allowing the user to mix two generated images in order to create a new one that's somewhere in between.

4.5.7 Experimenting with other datasets

In this exploration, we extend the application of DCGAN to two well-known datasets: the CelebA face dataset and the CIFAR-10 dataset. Both datasets present unique challenges and opportunities for generative modeling.

CelebA Dataset: The CelebA (Celebrities Faces Attributes) dataset is a large-scale collection of celebrity portraits. It's widely used for benchmarking in facial attribute recognition, face detection, and landmark (facial keypoints) localization. The dataset features images with a resolution of 178x218 pixels, predominantly in color, which we resize to 32x32 and normalize to accommodate our model's architecture. CelebA's rich and diverse set of facial images makes it an ideal candidate for testing the capabilities of DCGAN in generating realistic human faces.

Figure 4.15: The CelebA dataset

CIFAR-10 Dataset: CIFAR-10 is another popular dataset in the field of machine learning and computer vision. It consists of 60,000 32x32 color images in 10 different classes, with 6,000 images per class. The classes include various objects like animals and vehicles, providing a good test-bed for a model's ability to generate diverse images across multiple categories.

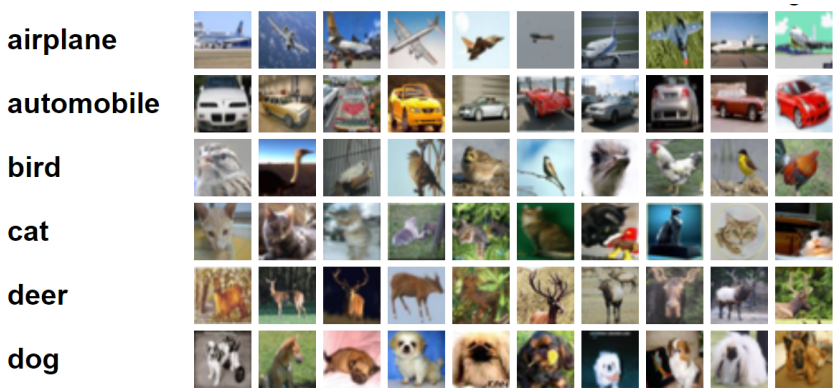


Figure 4.16: The CIFAR-10 dataset

For both datasets, we maintain consistency in our approach, using the same default parameter values and image preprocessing steps. The primary difference lies in adapting the input channel size to accommodate the RGB color space of these images. We train the model for 50 epochs on each dataset and capture the evolution of the generated images every 10 epochs. This progression can be visualized through GIF animations, easily generated by setting `show_animation = True` in the `plot_gan_training` function. These visualizations offer a dynamic view of how the model learns and adapts, gradually improving its ability to generate convincing images that capture the essence of the respective datasets. We obtain the following results:

Figure 4.17: DCGAN results with **CelebA**

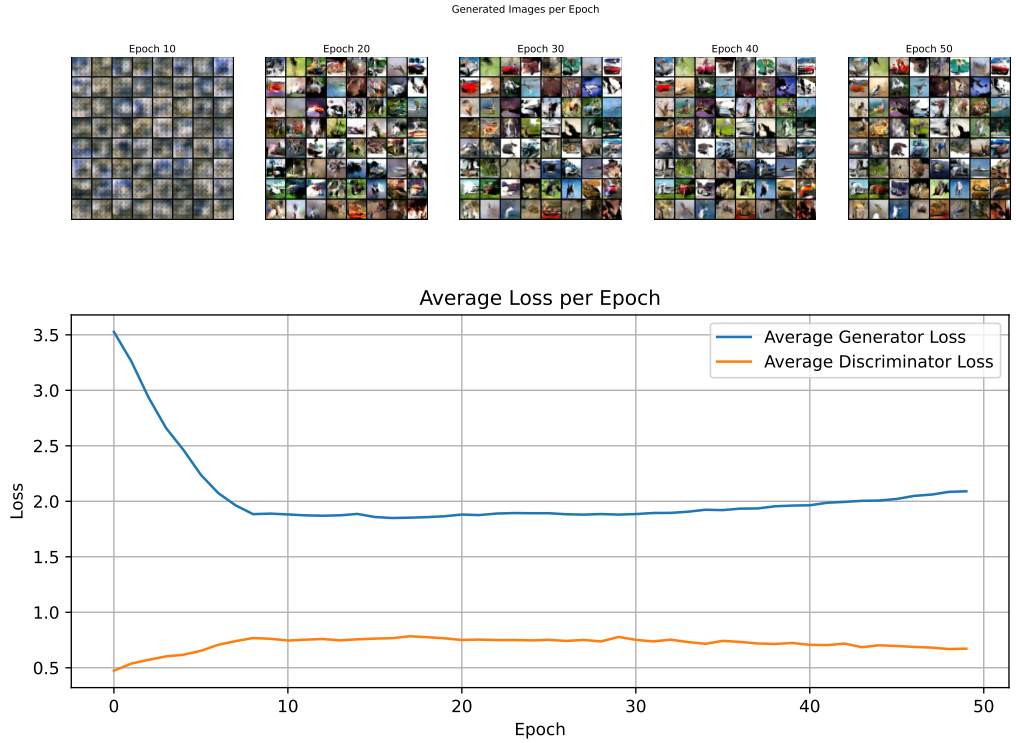


Figure 4.18: DCGAN results with **CIFAR-10**

Observations: Upon closer examination of the generated images, we observe a significant improvement in quality for both the CelebA and CIFAR-10 datasets. However, when analyzing the generator and discriminator losses, we note distinct patterns:

1. **CelebA**: In the case of the CelebA dataset, the generator loss initially decreases rapidly, suggesting

that the generator momentarily outperforms the discriminator. However, the discriminator quickly adapts and becomes proficient at identifying fake images. The subsequent increase in the generator's loss does not necessarily indicate poor image generation but rather signifies the discriminator's improved ability to detect fake samples. Counter-intuitively , at epoch 5, the generator's loss was at its lowest while the discriminator's loss was at its highest. In contrast, around epoch 50, the generator's loss was notably higher, yet the images generated at Epoch 5 are significantly worse than those at epoch 50. This underscores the complexity of monitoring GAN training, where achieving a delicate balance between generator and discriminator performance is essential. Nonetheless, it is important to note that the primary objective of this practical is not to find the optimal DCGAN parameter combination but to explore the training process.

2. **CIFAR-10**: In the case of the CIFAR-10 dataset, the training process appears more stable. The generator's loss consistently decreases, indicating its ability to generate images that deceive the discriminator. The images generated around epoch 50 show clear progress in learning. This relatively stable learning can be attributed to the default parameters we used, which align well with the characteristics of the CIFAR-10 dataset. It emphasizes the importance of adjusting parameters based on the specific dataset and GAN architecture in the context of GAN training.

4.6 Section 2: Conditional GANs (CGANS)

4.6.1 A brief overview of Conditional GANs

Conditional Generative Adversarial Networks (CGANs) represent an evolutionary leap in the world of generative models. While traditional GANs generate data from random noise, CGANs condition the data generation process on additional information, typically in the form of labels or tags. This conditioning allows for more controlled and targeted data generation. For instance, in image generation, a CGAN can be conditioned to produce images of specific categories, like different types of objects or facial expressions.

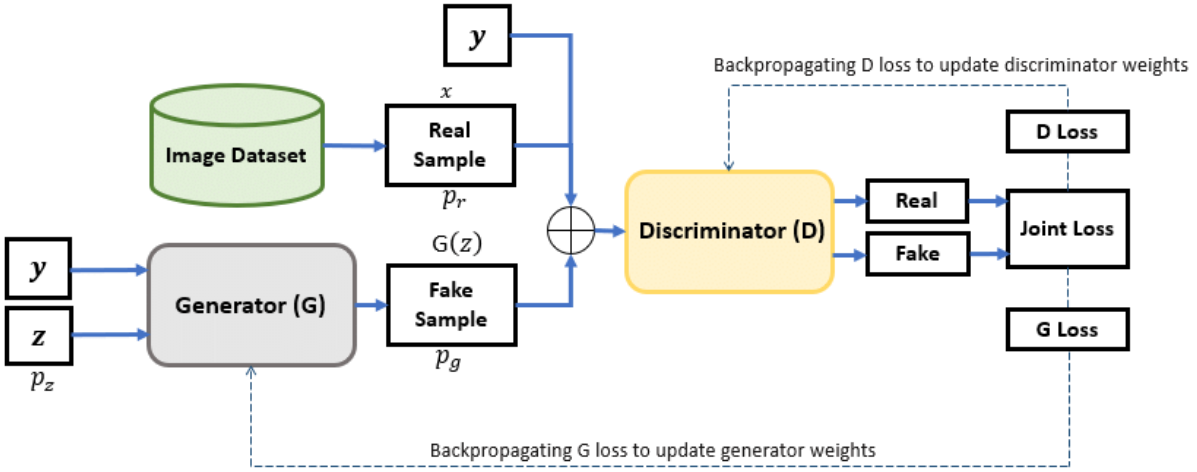


Figure 4.19: Basic architecture of a CGAN

The key distinction between GANs and CGANs lies in this ability to direct the generation process. In a standard GAN, the generation is a random process influenced solely by the latent space. In contrast, CGANs use both the latent space and external information (like class labels) to guide the generation, leading to more specific and diverse outputs.

4.6.2 Implementation of Conditional GANs

In our implementation of CGANs, we intricately combine class information with the latent vector to guide both the generator and discriminator:

4.6.2.1 Conditional Generator

- We start with two key inputs for the generator: a latent vector (random noise) and class information. The class information is encoded as a one-hot vector (size `[`$batch_{size}$`, 10]` for a dataset like MNIST with 10 classes, or `[`$batch_{size}$`, 2]` for CelebA).
- Both the latent vector and the one-hot encoded class vector are transformed into **'images'** by adding dimensions, resulting in two separate 4D tensors.
- The next step involves upscaling these **'images'** using deconvolutional layers (`ConvTranspose2d` in PyTorch). Each modality (latent vector and class vector) passes through its respective upscaling block.

- Post-upscaling, we concatenate these feature maps along the channel dimension and pass them through subsequent upscaling blocks. The final output is a generated image, conditioned on the input class.

4.6.2.2 Conditional Discriminator

- Similarly, the discriminator takes both an image and its corresponding class label as inputs. The class label is also represented as a one-hot vector.
- We downscale both the input image and the one-hot encoded class vector separately using convolutional layers (`Conv2d` in PyTorch), resulting in two distinct feature maps.
- These feature maps are then resized to ensure matching dimensions and concatenated. This combined feature map is then fed through additional downscaling blocks.
- The final output of the discriminator is a single value representing the likelihood of the input image being real or generated, informed by both the image and its class label.

This dual conditioning in CGANs, both at the generator and discriminator levels, establishes a rich training environment. It enables the model to generate more specific outputs and provides a finer understanding of the features associated with different classes, leading to enhanced generative capabilities and broader applicability in tasks requiring targeted data generation.

4.6.3 CGAN experiments on MNIST and CelebA

Using this architecture, we train the model for 20 epochs and resize all inputs to 32x32, for both datasets.

4.6.3.1 CGAN with MNIST dataset

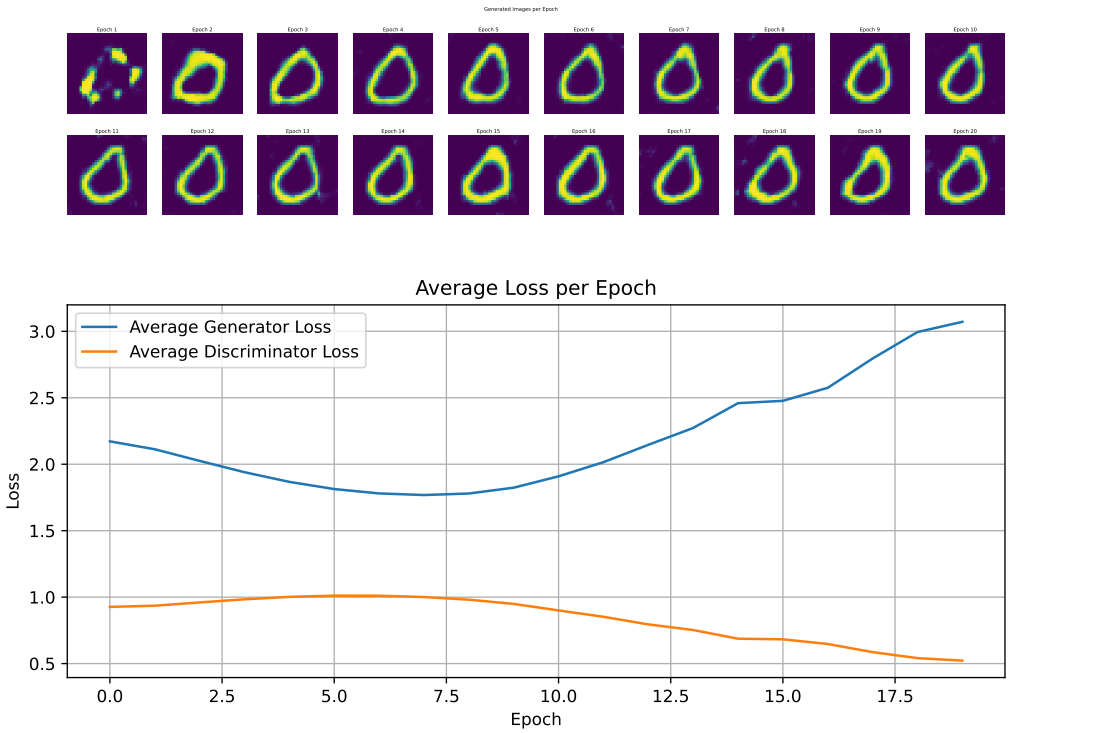


Figure 4.20: **CGAN train results on MNIST**

Observations: By incorporating class-specific one-hot vectors (representing digits 0 through 9), our CGAN demonstrates remarkable control over the generated outcomes. The experiment on the MNIST dataset yielded 20 images for each digit class. The results clearly exhibit consistency and accuracy in generating images corresponding to the specified class labels. This level of control is a significant advancement over traditional GANs, where output generation lacks predictability.

4.6.3.2 CGAN with CelebA dataset

Expanding our exploration to the CelebA dataset, we conditioned the CGAN on specific facial attributes. Utilizing the `train_CGAN_celebA` function, we targeted different attributes by assigning values to the `id_att` parameter. This approach enabled us to direct the GAN output based on selected attributes. For instance:

- `id_att = 31`: Targets facial expression, generating images categorized as 'Smiling' or 'Not Smiling'.
- `id_att = 4`: Focuses on hair attributes, distinguishing between 'Bald' and 'Not Bald'.
- `id_att = 14`: Addresses the presence of accessories, specifically 'Eyeglasses' or 'No Eyeglasses'.
- ...and so on for other attributes.

In our specific experiment, the CGAN was conditioned on the **31st attribute (Smiling or Not Smiling)** of the CelebA dataset, demonstrating the versatility and effectiveness of the model in generating conditioned images.

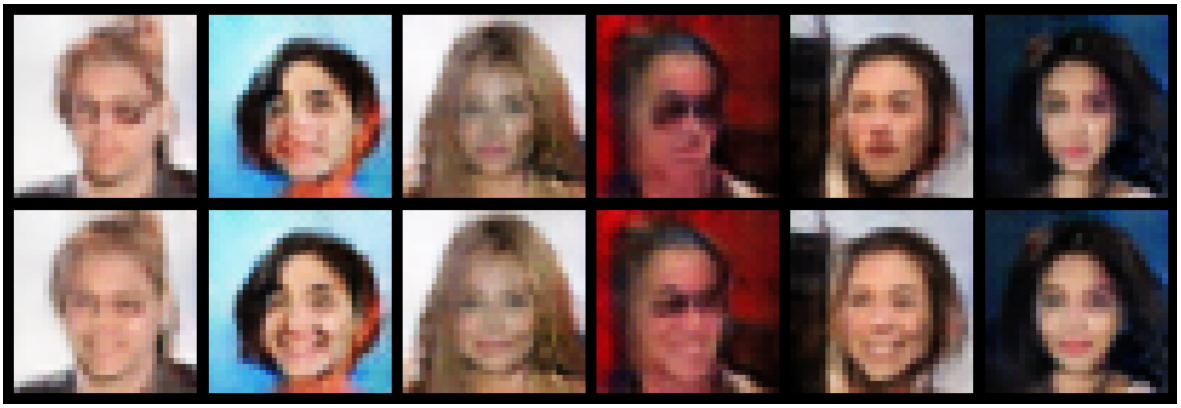


Figure 4.21: **CGAN train results on CelebA (class = Smiling | Not Smiling)**