ISTANBUL TECHNICAL UNIVERSITY

FACULTY OF SCIENCE AND LETTERS

PHYSICS ENGINEERING

Synthetic Image Dataset Generation for CNN Training

Efekan Mutlu Altay Avcı

Supervisor: Prof. Dr. Emre Onur Kahya

May 2024

Contents

1 Introduction

This study addresses an approach aimed at generating synthetic image data to train Convolutional Neural Network (CNN) models. Focusing on the difficulty of creating a unique dataset for training CNN models due to industry-specific challenges, the approach seeks to augment existing images with a complex structure developed using diffusion models to increase the volume of a specific dataset.

Diffusers are models that transform the stepwise noisy distribution, obtained from random noise, into the intended distribution using a scheduler and U-net model. The intended distribution can be a two-dimensional image, a three-dimensional image, or a sound wave output. All distributions we aim to obtain in our study are two-dimensional image distributions with three-color channels, which will be used in the training of CNN models. While theoretically any diffusion model could generate an unlimited number of images with random noise, in practice, as the size of the augmented dataset obtained with diffusers increases, the contextual similarity of the generated images tends to cause overfitting in CNN training.

The primary focus of this study is to address the increasing similarity among the outputs of diffusers, differentiating itself from standard augmentation methods by altering the background image, position, and perspective of each image element in the dataset, thus providing contextual diversification. Through this pipeline, we aim to overcome the image scarcity and monotony, which are the major disadvantages of a limited dataset. Additionally, an efficiency analysis will be conducted by comparing the outputs obtained using our designed pipeline with those obtained using the Stable Diffusion XL (SDXL) pipeline. The objective of this approach is to present a significant methodology for researchers aiming to enhance CNN models using synthetic image data by comparing the context manipulation ability of our established pipeline with the widely used open-source diffusion model, SDXL.

In this study, before introducing the diffuser pipeline we have developed, essential components and concepts crucial for understanding the structure of diffusers and various models used in generating synthetic image data, which we will encounter in later sections, will be explained. Subsequently, the architecture of the complex pipeline elements, including its outputs, code implementation, and other various aspects, will be individually examined.

In the later parts of our project, data sets containing images produced with the pipeline we designed will be created and these will be used in CNN model training. After that, these results will be compared with the results of the CNN model with the same hyperparameters trained with real data. During this examination process, both standalone synthetic image datasets and versions mixed with real images will be used. The reason for this is to determine the most efficient way synthetic images can be utilized in CNN model training.

In the subsequent sections of the thesis, the dataset created for training will first be examined, followed by an explanation of the CNN model and training parameters to be used, and an analysis of the training and test results. In the final section, a general evaluation of the entire study will be conducted to assess performance.[2]

2 Diffusion Method

2.1 U-net

U-Net stands out as a significant deep learning model, particularly in the field of biomedical imaging. Fundamentally, it presents an architecture that is effective in image segmentation tasks. One of the main advantages of this model is its ability to provide high precision and accuracy in localization tasks, thanks to the connections added between down-sampling and up-sampling stages.[13]

U-Net has a symmetric structure comprising convolutional and transpose convolutional layers. The down-sampling stage includes convolutional and pooling layers to extract essential features from the input image, allowing the model to learn general characteristics. The up-sampling stage incorporates transpose convolutional layers to restore the learned features to the original input dimensions, facilitating precise localization.

U-Net has been successfully employed, particularly in the segmentation of tissues and organs in the field of medical imaging. The model facilitates the merging of lower-level features with higher-level features through skip connections, contributing to the preservation of details and enhancing overall learning performance.

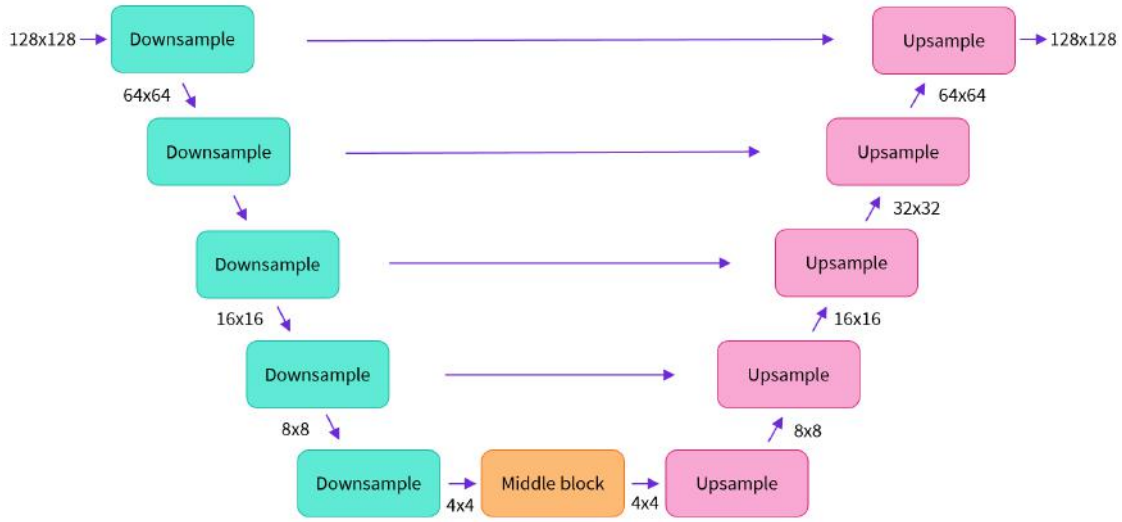


Figure 1: U-Net architecture representation

When combined with diffusion models, the role of U-Net becomes even more crucial. The robust feature extraction and segmentation capabilities provided by U-Net are critically important for diffusion models. As we will see in later sections in more detail, the fundamental goal of diffusion models is to manipulate the initially given random distribution, referred to as noise, based on the input provided by the user. The function of the U-Net architecture in this process involves identifying key features by directing the distribution towards the desired distribution. New information layers are established by associating and correlating these key features. Additionally, the U-Net architecture plays a crucial role in determining and grouping information about the shape, position, and color of these features on the distribution. Skip connections, in particular, play a critical role in preserving low-level details and generating high-quality synthetic images.

2.2 Scheduler

After extracting features with the U-Net model and establishing the overall context of the distribution, a scheduler structure is employed for the transition process from the current state to the target distribution. Schedulers are functions that take the output of a model (the example upon which the diffusion process operates) and a time step, returning a denoised example from the noise.

To address the inadequacy of controlling the noisy distribution in a single iteration of the scheduler, the approach involves denoising or adding noise to the distribution from the previous U-Net output and then refeeding it into the U-Net model. This allows the extraction of updated features by the U-Net, resulting in a more detailed context. The output is then processed again by the scheduler, and this process is repeated for a specific limit step. Within the scheduler,

the iterations of stochastic and ordinary differential equations can be evaluated, enabling the probabilistic distribution to evolve and be controlled. [1]

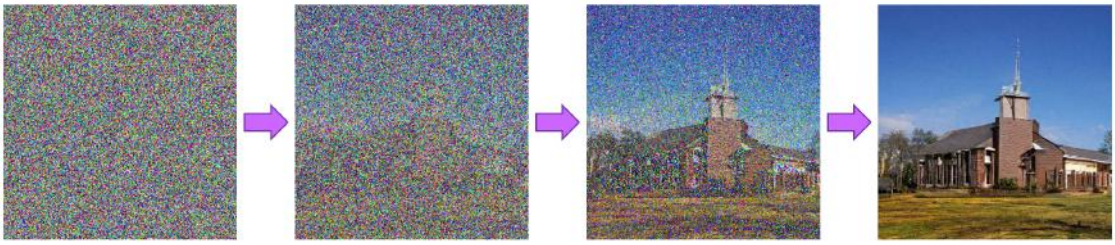


Figure 2: Image evolution from noise with Schedular

During training, a scheduler is employed to add noise to an example, and this may involve various algorithms to train a diffusion model. During inference, a scheduler defines how to update an example based on the output of a pre-trained model.

Schedulers provide the ability to regulate the learning process and inference of diffusion models. Depending on the value of the timestep, these functions determine how noise addition or example updating will occur, playing a crucial role in controlling the diffusion process.

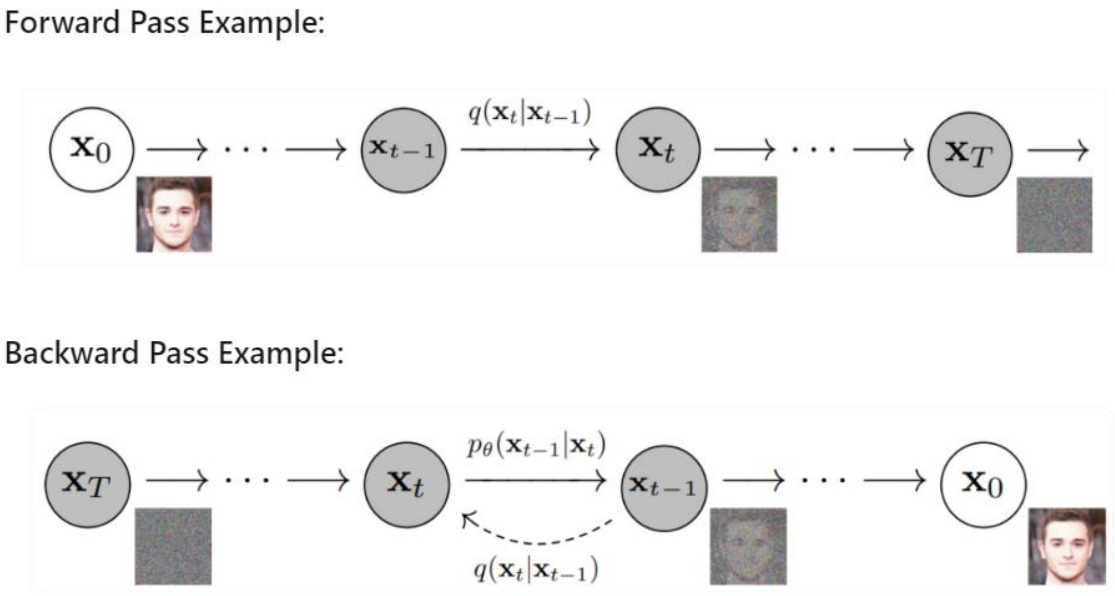


Figure 3: Adding and removing noise from the distribution using Schedular

2.2.1 UniPC Multistep Scheduler

UniPC demonstrates a promising capability in the field of high-resolution image synthesis generated by diffusion models. However, taking an example from a pre-trained diffusion model is time-consuming due to multiple evaluations of the denoising network. As a solution to this challenge, UniPC has been developed, incorporating a corrector (UniC) that can be applied after existing schedulers and a predictor (UniP) that supports an additional model evaluation without any extra regularization.[23]

UniPC presents a Unified Predictor-Corrector framework, particularly effective in significantly improving the sample quality compared to previous methods in very few steps. This model provides a

unified analytical form for any regularization and can notably enhance the sample quality, especially when taking samples in very few steps, compared to previous methods.

The experiments with UniPC demonstrate that it can not only reduce production time but also significantly enhance sample quality. These gains will provide both speed and processor optimization for our complex pipeline, which will be discussed in more detail in the future, enabling the system to operate dynamically. Moreover, it will improve image quality by tolerating decreases in image quality that occur due to the randomness in diffusion models.

2.2.2 Euler Ancestral Discrete Scheduler

Although Euler Ancestral Discrete (EulerA) scheduler operates on the same fundamental principle as other schedulers in the industry, its advantageous features include optimizing the sampling mechanism to be lighter. This feature will increase the operational speed of our designed pipeline and reduce the processing power on the GPU. Additionally, its efficient operation of the stochastic sampler enhances image quality.[3]

This scheduler will be utilized in the outpainting module of our designed pipeline. The fundamental reasons for this choice, as mentioned above, include its lightweight nature, ensuring it does not burden the processor during the outpainting process, a cornerstone of mass production. Furthermore, in this module, it helps reduce the potential decrease in image quality due to condition inadequacy and provides balanced outputs for the augmented dataset for CNN by maintaining the image quality consistency across all generated examples.

2.3 Variational Autoencoder

Variational Autoencoders (VAE) are models used in the field of generative modeling, capable of learning data distributions and generating new data samples. VAEs are designed to represent data in a code space, ensuring that data points with similar features have similar codings. Their fundamental structure consists of two main components: an encoder and a decoder.

The encoder, taking the input data, typically attempts to map it to a multivariate normal distribution. This process involves learning a statistical representation of the data, and the encoder produces a distribution parameter containing the mean and variance representing the input data. The decoder, on the other hand, takes a point in the code space and tries to transform this point into a reconstructed version of the input data. At this stage, the goal is to transform samples in the code space into realistic data samples. During the training of VAE, both the encoder and decoder are optimized simultaneously. The logic of transforming the initial image into an abstract latent layer at a bottleneck point and then reconstructing the image back to its original dimensions with the decode part can be likened to the U-Net architecture in this respect. [4]

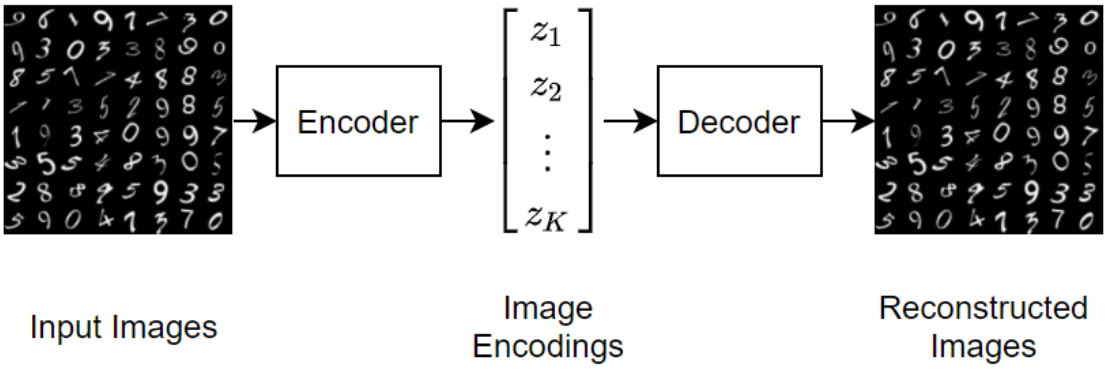


Figure 4: Representation of working principle of VAEs

In comparison to other models, Variational Autoencoders (VAEs) offer significant control over modeling our latent distribution. The advantages provided by the ability to sample from the distribution, followed by decoding and generating new data after training, make VAEs attractive compared to standard autoencoders. VAEs, in contrast to standard autoencoders, provide a closer match of the latent space to the data distribution, offering more effective learning and generation capabilities.

2.3.1 Autoencoder KL

Kullback-Leibler (KL) Divergence, which is present within the autoencoder, plays a crucial role in training a Variational Autoencoder (VAE) model. KL Divergence is an information theory concept that measures the difference between two probability distributions. While a VAE learns the representation of a data point in the latent space, it aims to preserve how close this representation is to a certain distribution. This is where KL Divergence comes into play. KL Divergence measures the difference between the learned distribution and a reference distribution.[5] Mathematically, it is expressed as follows:

$$D_{KL}(P \,||\, Q) = \sum_i P(i) \log \left(\frac{Q(i)}{P(i)} \right) \tag{1}$$

Here, P represents the true distribution, and Q represents the desired reference distribution. The logarithmic expression in this formula measures the proportional differences between the two distributions. During training, the VAE uses this KL Divergence term to ensure how close the obtained representation is to a certain distribution. Thus, the learned representation effectively captures the data and preserves a certain structure. KL Divergence serves as a regularization mechanism for VAE, preventing overfitting and promoting more generalizable learned representations.

2.3.2 Asymmetric Autoencoder KL

The most important feature that distinguishes Asymmetric Autoencoder from other VAE architectures stems from its asymmetric VQGAN structure.

Asymmetric VQGAN stands out with two fundamental designs. Firstly, it adds an additional branch by making the VQGAN decoder conditional to support local editing tasks. Secondly, it employs deeper or wider decoders similar to the complexity of the encoder to enhance the capabilities of the decoder. It is noted that these designs improve the ability to better preserve unedited regions and recover more details from quantized outputs. This feature places the asymmetric autoencoder

in a crucial position for the outpainting process in our designed pipeline. As we will examine in more detail in the upcoming sections, in the outpainting process, a derivative of the inpainting method, the root image containing the object is preserved with an automated mask condition and iteratively shifted on an empty canvas. Preserving the image of our object in the resulting new image is crucial for CNN training. The autoencoder has been observed to preserve the object thanks to its asymmetric VQGAN structure.[24]

Furthermore, due to the low training and inference costs of asymmetric VQGAN, it only requires training a new asymmetric decoder, eliminating the need to modify the original StableDiffusion and VQGAN encoders.

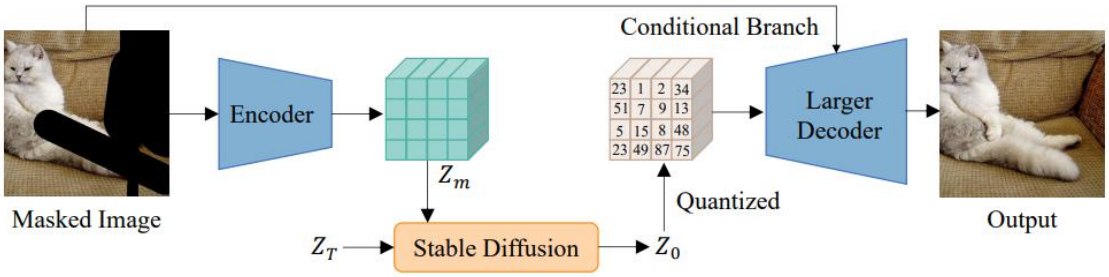


Figure 5: Representation of Asymmetric Autoencoder KL working principle

2.4 Text and Image Encoders

Text and image data integration allows artificial intelligence models to provide effective solutions across a wide range of applications. In this context, the Contrastive Language-Image Pre-training (CLIP) model stands out as a pre-trained model that integrates text and image representations. CLIP represents text and image data in the same space, providing a structure that can match text-image pairs with similar semantic features. This model can be utilized in various applications with its ability to find image descriptions matching a given image or images matching a given text description.[11]

In the process of combining CLIP's text and image representations, the integration with diffusion models, particularly effective in image manipulation tasks, holds significant importance. Diffusion models are primarily successful generative models used in tasks such as image synthesis and manipulation. The integrated diffusion model with CLIP maintains text-image matching capabilities while also delivering effective performance in image manipulation or synthesis tasks.

During the training process, the model learns text and image representations using the pre-trained weights of CLIP. Subsequently, a fine-tuning step is performed for a specific application. In this stage, the diffusion model is utilized for image manipulation or synthesis processes, while CLIP generates image representations consistent with text descriptions during this process.

The integrated model can take text descriptions as input and manipulate images or synthesize new images accordingly. The diffusion model is employed to optimize these manipulation or synthesis processes, and CLIP obtains image representations consistent with text descriptions during this process. Through this integration, the ability to understand complex relationships between text and image data in-depth allows the model to be effectively applied across a wide range of applications.

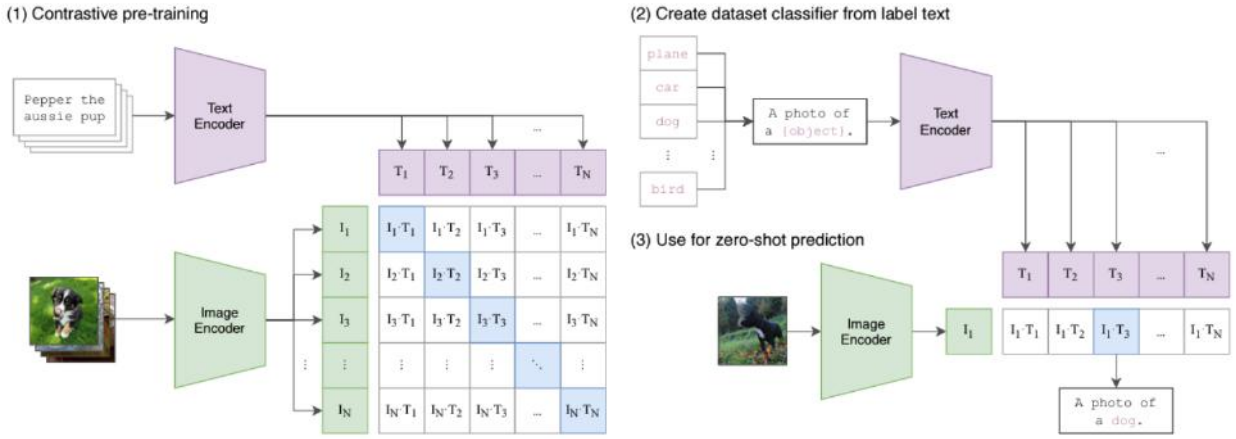


Figure 6: Approach Summary: Unlike standard image models that simultaneously train an image feature extractor and a linear classifier for label prediction, CLIP jointly trains an image encoder and a text encoder to predict correct pairings in a batch of (image, text) training examples. During testing, the trained text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the classes in the target dataset.

2.5 Attention

The attention mechanism is a widely used structure, especially in transformer-based models. This mechanism enables determining and highlighting the relationship of a word with other words. Attention helps achieve better performance in natural language processing tasks by learning the interaction of a word with the other words in a sequence. In text encoders, the attention mechanism, specifically referred to as self-attention or intra-attention, determines the interaction of a word with other words within the same sequence. Self-attention allows a word to understand and learn its relationships within its own context, enabling the model to better grasp connections between words.[22]

In diffusion models, the attention mechanism plays a significant role, especially in complex tasks like text-to-image generation. In understanding the connections between text and images, attention allows specific words and features in the text to influence the visual content. This is particularly useful in text-based tasks, for example, to understand how a description impacts an image.

Diffusion models, especially models like StableDiffusion, can successfully integrate visual information related to text using the attention mechanism. This allows the information provided by text to influence the visual content, resulting in the generation of more realistic and meaningful images.

2.5.1 Self Attention

The attention mechanism is a critical component commonly used in learning models in natural language processing and computer vision fields. This mechanism is designed to understand relationships between input signals by focusing on specific elements. Initially, matrices defined through the terms Query, Key, and Value are employed.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2}$$

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \\ \text{where} \quad \text{head}_i &= \text{Attention}(QW_{Q_i}, KW_{K_i}, VW_{V_i}) \end{aligned} \tag{3}$$

In natural language processing models, such as the CLIP model, the attention mechanism is utilized to determine relationships between text and image inputs. The model applies the Query matrix representing text queries to image keys, performing a weighted sum to identify similarities. This allows the model to understand and learn connections between text and images.[18]

In diffusion models, particularly in models like Stable Diffusion, the attention mechanism plays a crucial role during the learning process. The attention mechanism improves the image by focusing on specific pixels during the transformation process in various steps.

The utilization of the attention mechanism in diffusion models, especially compared to previous models, introduces a regularization that provides lower computational cost and more effective learning. By regulating the mechanism's ability to understand the distance and similarity between pixels, it assists the model in achieving more consistent and high-quality results.

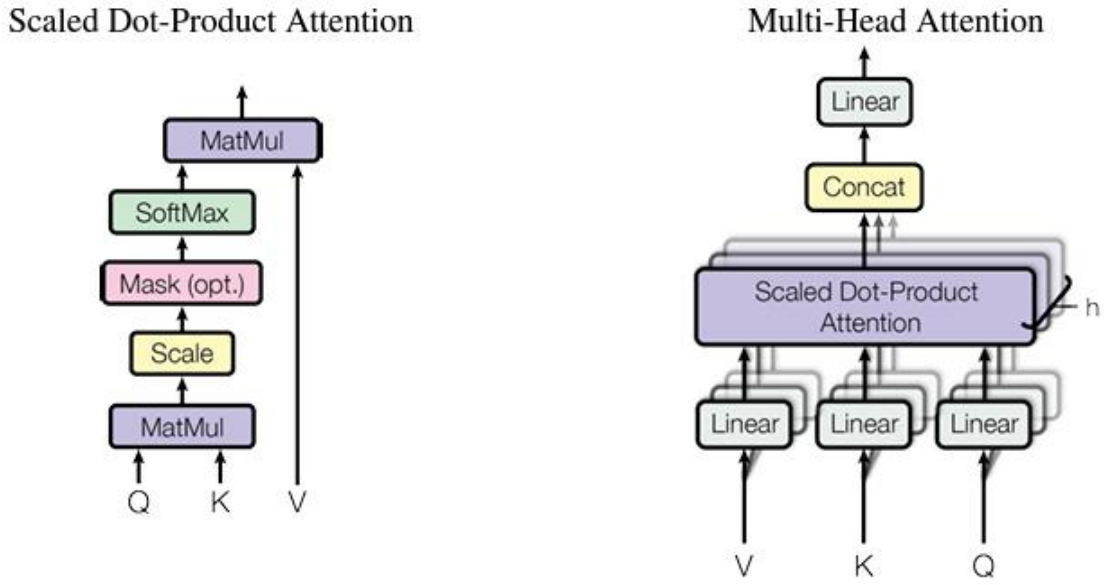


Figure 7: (left) ScaledDot ProductAttention. (right) Multi Head Attention consists of several attention layers running in parallel.

2.5.2 Cross Attention

Cross attention is a mechanism used by a model to understand relationships between two different input modes and make inferences based on these relationships. Apart from the inputs, the calculation of cross attention is the same as self-attention. Cross-attention combines asymmetrically two separate embedding sequences of the same dimension. In contrast, self-attention's input is a single embedding sequence. While one sequence serves as the query input, the other functions as the key and value inputs.[7]

Exactly, in tasks like text-to-image generation, cross attention is employed to understand the connections between text and image inputs. For instance, it can be used to determine how objects or concepts described in a text caption are represented in an image. This enables the model to better comprehend the relationships between text and images, leading to more consistent and meaningful outputs.

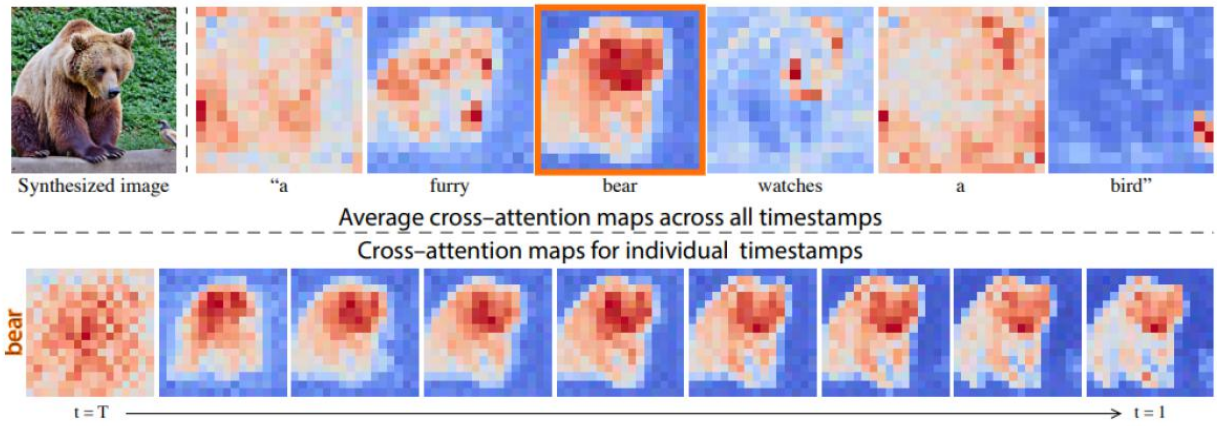


Figure 8: Cross attention maps of a text-conditioned diffusion image generation.

Indeed, the cross-attention mechanism typically involves calculating similarity scores between two different modules. For example, the similarity score between a word vector and an image region determines whether that word is associated with a significant feature in the image. These similarity scores are then used in a weighted manner to obtain a fused representation between the two modules. This process allows the model to capture meaningful relationships between different types of input modalities, enhancing its ability to generate coherent outputs.

2.6 Overall

The building blocks of standard diffuser models have been explained above. In this section, we can now fully examine the working mechanism of the stable diffusion pipeline.

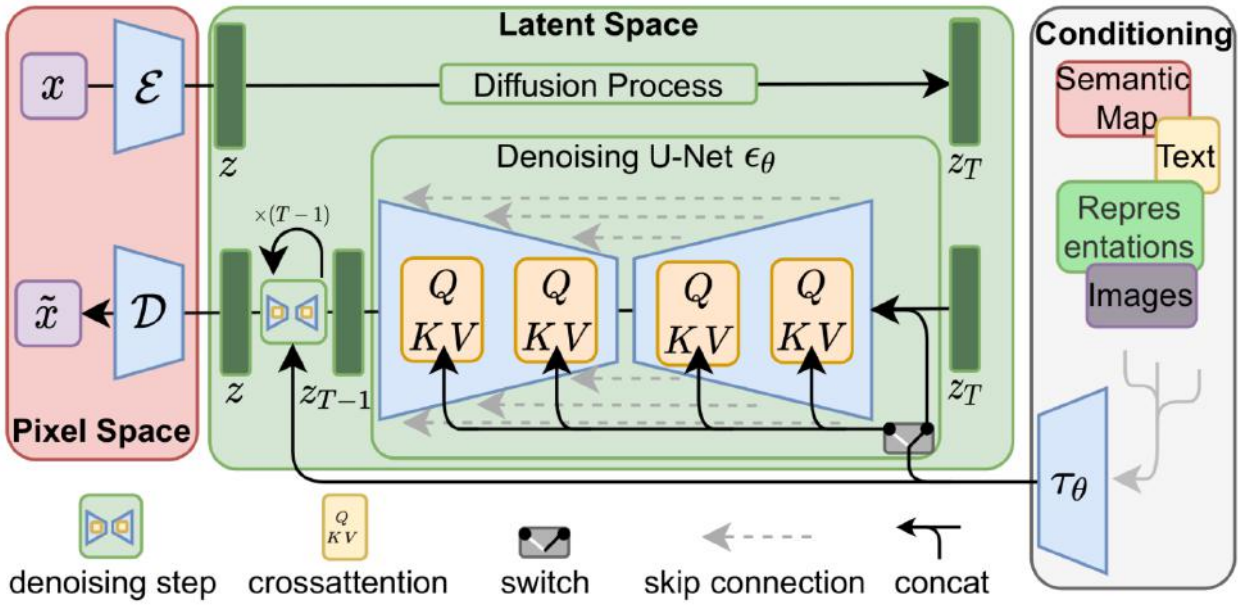


Figure 9: Representation of the Stable Diffusion working principle

The process starts with a random noise distribution, serving as a noisy starting point that is regenerated independently at each use in the pixel space of the model. This noise will undergo

an encode-decode process with the U-Net module, as mentioned earlier. In addition to the U-Net operations, text embedding conditions from the text encoder are introduced at this stage and become part of the matrix multiplications in the U-Net. This system, referred to as U-Net with attention, produces output that is sequentially passed to the scheduler module. Within the scheduler, utilizing the ODE and SDE structures in the sampler, the initial noise, considering the conditions, is iteratively updated to converge towards the target image.[12]

The exchange of samples between U-Net and the scheduler occurs for a specified "t" time step between U-Net and the scheduler. This loop structure will iteratively evolve the distribution, step by step, transforming the noise into the desired image. During this process, external conditions from the user, such as CLIP text embeddings or the embeddings of an example image distribution, can be provided to the U-Net model and used as conditions.

At the end of the time step, the obtained sample is transferred to the relevant variational autoencoder (VAE) module. Through this process, the final output is generated, representing the image in a latent space with two dimensions (width and height). The VAE aids in refining this representation, resulting in a final output image that will be presented to the user.

To train the U-net model within the diffusion model to generate visually coherent content, the reverse of the denoising process described earlier is applied. In this process, random noise is incrementally added to images from a training dataset until the time step reaches the upper limit. This results in obtaining a random noise distribution. The training procedure follows the standard neural network training processes, incorporating the U-net model and, if deemed necessary, the text encoder during training for potential future training of text-to-image models.

3 Pipe Architecture

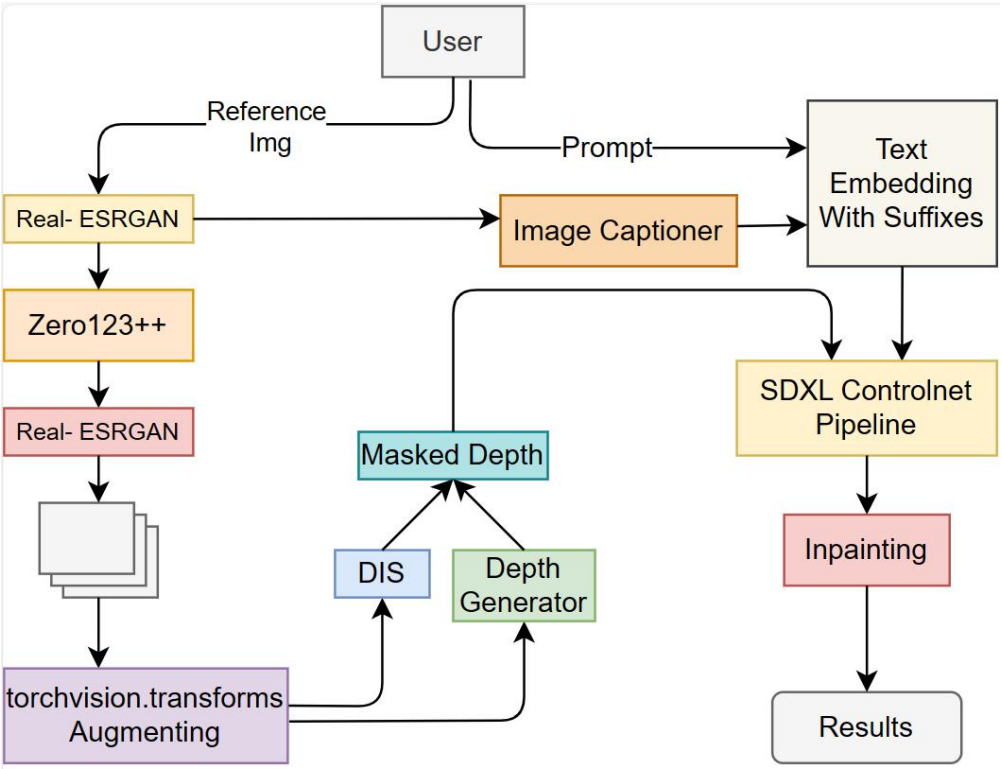


Figure 10: Representation of the pipeline architecture we designed

The foundational principle of the pipeline we have designed is based on acquiring only the image requested to be augmented from the user, obtaining augmentation parameters, receiving a prompt condition to manipulate the context in the desired direction, and automating the remaining procedures through algorithmic processes and artificial intelligence models. The user will be able to upload only one image and prompt at a time. To augment each image in the dataset, our pipeline needs to run within a loop, systematically processing the specified images within the file one by one.

Our pipeline initially scales the reference image received from the user by a factor of 2 using the Real-ESRGAN module and enhances its resolution. Subsequently, the image is forwarded to the Zero123++ module, enabling the generation of potential perspectives of the image from different angles.

The Zero123++ module inherently generates six different images from the provided input image as a standard procedure. These images encompass various perspectives of the main object, which is the focal point for training the CNN model, appearing against a simplified background. The images obtained from this process are then sent back to the Real-ESRGAN module, where they undergo a fourfold scaling, leading to an increase in resolution. This scaling and resolution enhancement process contributes to generating detailed and high-quality outputs for the anticipated diffusion results in the future.

Each of the images scaled with Real-ESRGAN undergoes augmentation individually using the primary methods provided by the torchvision library, a part of the PyTorch framework. These methods include operations such as rotating and shifting, which are currently employed in augmenting the dataset. The same augmentation procedures are applied to each image. The user can adjust the augmentation factor, denoted as t, determining the extent of augmentation. Consequently, each image obtained from the Zero123++ module will be duplicated t times, reflecting the specified augmentation level.

Following this step, both a mask image and a depth map will be generated for each image using the DIS module. Subsequently, these two images will be combined to create a masked depth image for each image. The purpose of this process is to establish suitable conditions for future background generation and to alter the context as much as possible without compromising the structure of the object, which is the focal point for CNN training, and its relationship with the surrounding environment.

Simultaneously, while these processes are taking place, the Real-ESRGAN outputs scaled by a factor of 2 are sent to the image captioning step, utilizing the BLIP model. This allows for the automatic analysis of the image uploaded by the user and its conversion into single-sentence texts. The objective is to establish the necessary conditions for the Stable Diffusion XL ControlNet module. In this scenario, to better control the image context with both initial input from the user and the combination of image captioning and previously defined prompt suffixes, we obtain the system's context. We then convert all the obtained text data into text embeddings for the Stable Diffusion XL ControlNet module.

The masked depth and text embedding conditions are connected to the Stable Diffusion XL ControlNet module to generate the main content of the image in the desired direction. Simultaneously, utilizing text embedding, the background image is diffused. By default, synthetic data can be generated with one altered background for each image. However, the user has the option to increase this default value, thereby augmenting the number of synthetic data generated for each image. Consequently, in this step, we can increase the previously obtained number of images according to demand by a factor of c.

The background obtained from the Stable Diffusion XL ControlNet module transitions to another variation of the Stable Diffusion pipeline, namely the Stable Diffusion Inpaint module, for final processing. The fundamental purpose of this step is to alter the background. Shifting the focus of the CNN model, which is centered on the object, will diversify the content by displacing the object

within the image. This process differs from asymmetric padding, as in the padding operation, there will be thin empty pixels around the image, and the image will generally remain centered. However, in the inpainting process, whose details will be explained in later sections, the newly formed pixel layer will be both thick and suitable for the background. Thus, the main object can be displaced to the outermost points of the image. The mentioned inpainting process will be applied to each image obtained from the preceding step, and each image will be augmented by a constant factor of k^2. The square expression here is a result of the designed square kernel grid structure.

In summary, augmentation processes will occur in four distinct steps based on the operations mentioned above: 7 images from Zero123++ (6 generated + 1 original image), t images from torchvision.transform, c images from Stable Diffusion XL ControlNet, and finally, k^2 images from inpainting. Thus, the total number of augmented images obtainable from a single original image will be equal to:

$$Augmentation\ factor = 7 \cdot t \cdot c \cdot k^2 \tag{4}$$

The models in our pipeline are all open source models. REAL-ESRGAN and DIS pre-trained models from Github repositories; Zero123++,Depth Generator,Image Captioning,SDXL ControlNet and Stable Diffusion Inpainting The pre-trained weights of the models were obtained from the Huggingface library.

In future sections, the process steps and separate pipeline modules within our pipeline will be explained in more detail.

3.1 Real-ESRGAN

The most prominent aspect of the pipeline we designed, which takes center stage in CNN training, can be considered as the AI-driven reproduction of the content within the images. Despite being an intermediate step throughout the process from start to finish, the resolution enhancement method, although not as perceptible in its impact as the others, has proven through our experiments to enhance the model's efficiency.

We use the Real-ESRGAN model as the super resolution model. This model is actually the same as the ESRGAN model in its working principle, but it only allows us to determine the scaling dimensions of the output obtained in more detail.

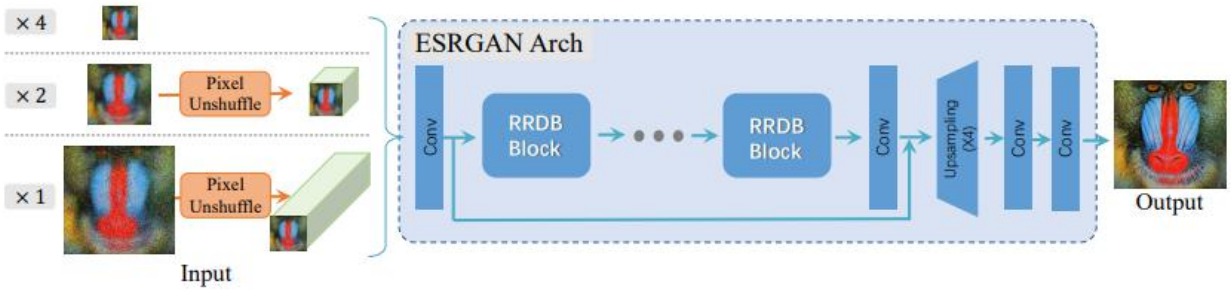


Figure 11: Real-ESRGAN working principle representation

The fundamental principle of ESRGAN involves the utilization of the Generative Adversarial Network (GAN) architecture. The generative model of ESRGAN takes low-resolution input images and endeavors to transform these images into high-resolution counterparts. Concurrently, the discriminative model strives to differentiate between the generated images and real high-resolution images. This competition fosters the generative model to produce more realistic and high-quality outcomes.[19]

Another crucial feature behind the success of ESRGAN is the Convolutional Neural Network architecture. During the training process, the model learns from a diverse training dataset consisting of pairs of low-resolution and high-resolution images.

ESRGAN also incorporates a loss function called Perceptual Loss. This is utilized to assess how closely the generated images resemble real high-resolution images. Consequently, the perceptual characteristics of the human eye can be better captured, leading to more realistic outcomes.

Finally, the high-resolution images obtained after training are typically subjected to a post-processing step. This step is employed to reduce undesirable artifacts and achieve cleaner results. ESRGAN stands out as an artificial intelligence model that significantly enhances the quality of low-resolution images by combining these steps.

The Real-ESRGAN model is employed to enhance the resolution of the output from the user-provided reference image. This serves as a beneficial starting point to achieve higher-quality results for subsequent processes, including the Zero123++ pipeline and other models that may follow in sequence.

The Real-ESRGAN model works with the following code blocks:

```
def init():
    global upsampler

    if not os.path.exists("scaler/weights"):
        os.mkdir("scaler/weights")
        url = 'https://github.com/xinntao/Real-ESRGAN/releases/download/v0.1.1/
    RealESRNet_x4plus.pth'
        response = requests.get(url)
        with open('scaler/weights/RealESRNet_x4plus.pth', 'wb') as f:
            f.write(response.content)
    model = RRDBNet(num_in_ch=3, num_out_ch=3, num_feat=64,
                    num_block=23, num_grow_ch=32, scale=4)
    upsampler = RealESRGANer(
        scale=4, model_path="scaler/weights/RealESRNet_x4plus.pth",
        model=model, device="cuda"
        )


def upscale(image, outscale):
    original_numpy = np.array(image)
    original_opencv = cv2.cvtColor(original_numpy, cv2.COLOR_RGB2BGR)

    output, _ = upsampler.enhance(original_opencv, outscale=outscale)
    upscaled = Image.fromarray(cv2.cvtColor(output, cv2.COLOR_BGR2RGB))

    return upscaled
```

Listing 1: Coding the Real-ESRGAN module

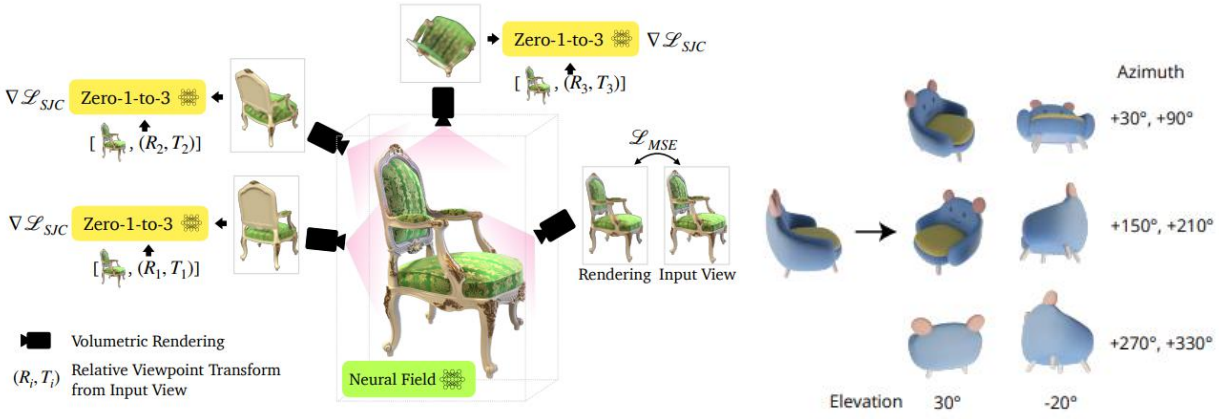
3.2 Multiview Zero123++

The Zero123++ model is essentially an enhanced version of the Zero 1-to-3 model, achieved through fine-tuning to construct a three-dimensional structure and improve the ability to remove distortion artifacts. It operates on the same principle as the underlying pipeline it is based on. Additionally, various enhancements have been implemented to ensure compatibility with the increasingly prevalent stable diffusion models and the controlnet architecture.

The operation mechanism of both the Zero 1-to-3 and its derivative, the Zero123++ model, relies on diffusers and the Neural Radiance Fields (NeRF) system. In terms of diffusers, the models exhibit proximity by manipulating a two-dimensional noise distribution through U-net and

schedular based on the user's text or image input condition, resulting in a two-dimensional image output. Additionally, a key distinguishing feature of the Zero 1-to-3 and Zero123++ models is the incorporation of the NeRF method used during training, setting them apart from other conventional diffusers. [8]

NeRF is an artificial intelligence model that generates photo-realistic images of 3D objects. This model employs a two-stage architecture called Rendering Network and Scene Representation Network. The Scene Representation Network is a neural network that predicts the color and density of points in the scene for each ray, providing a detailed representation of the scene in 3D space. The Rendering Network, using this representation, predicts the color value of each pixel, resulting in a realistic image. During training, the model compares the gathered ray information with real images and minimizes errors, allowing NeRF to learn the complexity and surface details of scenes and generate high-quality 3D images through ray-based rendering methods.[9]



(a) Zero 1-to-3 training (b) input to outputs

Figure 12: Zero123++ pipeline result format

During the training of the Zero 1-to-3 and Zero123++ models, this NeRF structure is utilized for the two-dimensional diffusion process. Traditional diffusion models, when trained to create a specific object, often disregard qualities such as angle and perspective since they are not crucial when assembling various images of the relevant object. Consequently, three-dimensional continuity is not preserved. In contrast, the Zero123++ model prioritizes maintaining the 3D structure of the object. It achieves this by obtaining images from all angles or creating 3D meshes in a virtual environment and capturing images of the object from different perspectives. This approach enables the model to better learn the contextual integrity of the object from all viewpoints.The acquired knowledge is processed and stored as weights in the diffusion module during training. When necessary, the learned information can be used to generate the desired image in 2D from the desired angle, with appropriate conditions.[8] [15]

The Zero123++ model used in our pipeline standardizes the output format in comparison to the Zero 1-to-3 model. It consistently generates images from specific azimuth angles and serves them in a single output containing six visuals. This output format facilitates the automation of our system, ensures uniformity among the metrics of the obtained outputs, alleviates the need for additional algorithm development, and contributes to the system's efficiency by streamlining its operation.

Below, the operation of the Zero123++ pipeline occurs with the following code blocks:

```
pipe_zero123 = DiffusionPipeline.from_pretrained(
    "sudo-ai/zero123plus-v1.1", custom_pipeline="sudo-ai/zero123plus-pipeline",
    torch_dtype=torch.float16
    ).to('cuda')
```

```
5
6  pipe_zero123.scheduler = EulerAncestralDiscreteScheduler.from_config(
7      pipe_zero123.scheduler.config, timestep_spacing='trailing'
8      )
9
10 pipe_zero123.vae = AsymmetricAutoencoderKL.from_pretrained(
11     "cross-attention/asymmetric-autoencoder-kl-x-2",
12     torch_dtype=torch.float16
13     ).to('cuda')
```

Listing 2: Coding the Zero123++ module

3.3 Dichotomous Image Segmentation (DIS)

The first necessary step for background replacement (or creating a new background from scratch) on the images from various perspectives obtained from the reference image in the Zero123++ module is the segmentation of points. At this point, our goal is to preserve the content of the main object image that will be used in the training of the CNN model to the maximum extent during future diffusion processes. To achieve this, we can obtain the mask image of the relevant object using a segmentation model and use it for the StableDiffusionXLControlNet pipeline.

The DIS model operates similarly to other segmentation models on the market, such as SAM (Segment Anything like Me). However, it possesses key features that distinguish it from others and have led to its inclusion in our designed pipeline. One of these features is its training process with a comprehensive and detailed dataset, performed bidirectionally. This method allows the encoder-decoder architecture to generate the final mask result from the input image, while simultaneously extracting features from the ground truth mask value of the relevant input image through the encoder-decoder blocks. Both output features are compared using a feature synchronization method, enabling a thorough learning process. Another reason for its inclusion is its lightweight design compared to the widely used SAM model. This results in quick processing without overburdening the processor, making it more efficient in comparison. [10]

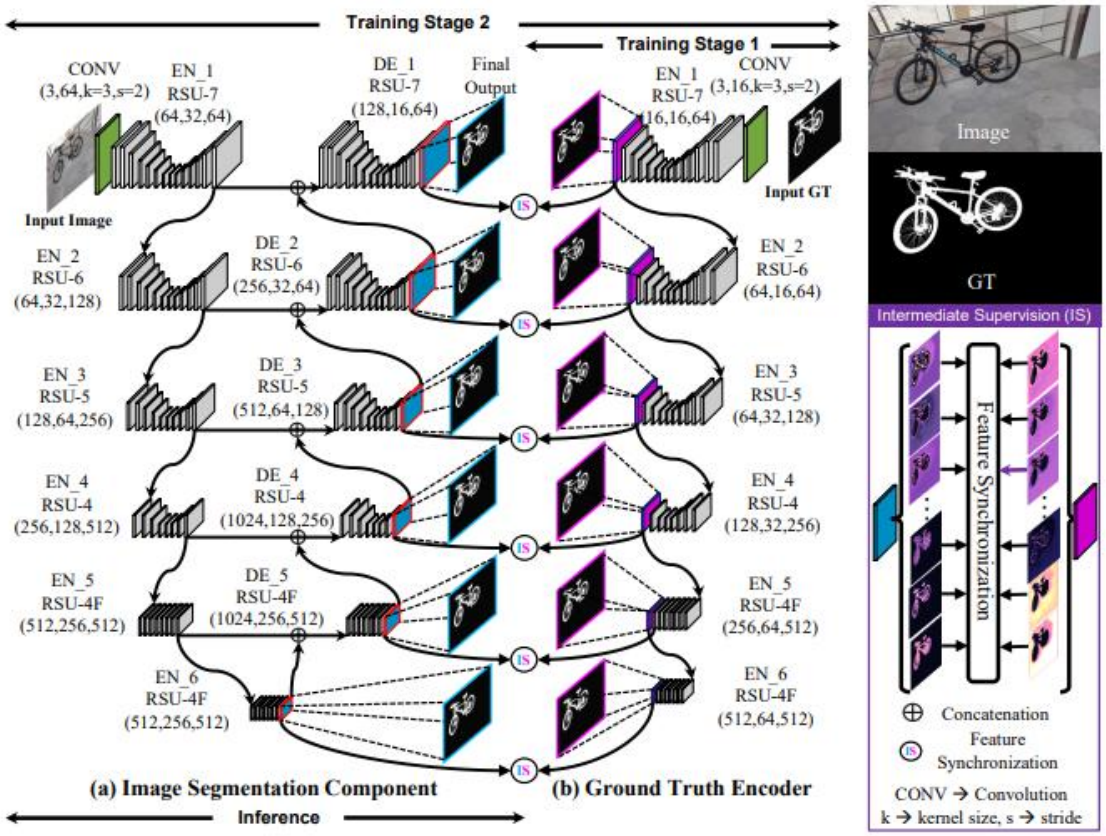


Figure 13: DIS Training Process

Using the mask data, various operations can be performed in diffusion processes: keeping the background constant while changing the object, keeping the main object constant while changing the background, or cropping the object from the reference image and pasting it onto a newly generated image. Unfortunately, the first option deviates from our focus on generating synthetic images for CNN training, as the main object to be detected would be lost. The second and third options are suitable for our objectives.

The method we employ is a variation of the second option. Our devised approach involves segmenting the multiview outputs with DIS to obtain mask data. Subsequently, we combine this mask data with depth data to obtain masked depth. This is because the general blurriness of depth images can distort the boundaries of the object. Combining it with mask data helps to define the boundaries more accurately. When using this depth masked condition in the StableDiffusionXLControlNet pipeline, it will reconstruct the entire image. However, since the three-dimensional structure of our shape and the perspective context relative to the background are preserved, this does not pose an issue.

At this point, cropping the main object from the reference image using mask data and pasting it onto the image generated by the diffuser is also a logical option. However, for our designed pipeline, we did not find it necessary and opted not to make the structure overly complex. Yet, this option is defined as optional in the pipeline for future cases requiring precision, such as logo, sign, specific character detection, or image segmentation tasks. Below the installation and operation of the DIS model is illustrated by code block:

```
def predict(net, inputs_val, shapes_val, hypar, device):
    '''
    Given an Image, predict the mask
```

```
    '''
    net.eval()

    if (hypar["model_digit"] == "full"):
        inputs_val = inputs_val.type(torch.FloatTensor)
    else:
        inputs_val = inputs_val.type(torch.HalfTensor)

    inputs_val_v = Variable(inputs_val, requires_grad=False).to(
        device)

    ds_val = net(inputs_val_v)[0]
    pred_val = ds_val[0][0, :, :, :]
    pred_val = torch.squeeze(F.upsample(torch.unsqueeze(
        pred_val, 0), (shapes_val[0][0], shapes_val[0][1]), mode='bilinear'))

    ma = torch.max(pred_val)
    mi = torch.min(pred_val)
    pred_val = (pred_val-mi)/(ma-mi)

    if device == 'cuda':
        torch.cuda.empty_cache()
    return (pred_val.detach().cpu().numpy()*255).astype(np.uint8)


def segment(image):
    image_tensor, orig_size = load_image(image, hypar)
    mask = predict(net, image_tensor, orig_size, hypar, device)

    mask = Image.fromarray(mask).convert('L')
    im_rgb = image.convert("RGB")

    cropped = im_rgb.copy()
    cropped.putalpha(mask)

    return [cropped, mask]
```

Listing 3: Coding the DIS module

3.4 Depth Generator

The second step necessary for the background replacement process on the images from different perspectives obtained from the reference image in the Zero123++ module is to obtain the depth image.

In our experiments, when we only took the mask data of the images and performed diffusion based solely on that, we generally observed a decrease in quality. We attribute this primarily to the U-net model not adequately grasping the spatial context of the generated image, resulting in outputs that lack a depth perception and resemble a general cartoon style. Additionally, since we aim to automate our devised system as much as possible, we prefer minimal user intervention. Manipulating depth perception with prompt inputs would introduce a trial-and-error burden on the user.

For these reasons, we developed a depth module that operates for each image, aiming to generate the depth data of the relevant images. This approach not only preserves the spatial context of the future diffusion process's output but also adds depth perception to the main object, preventing potential geometric flaws in the future.

The code we wrote to generate depth is as follows:

```
def init():
    global depth_estimator, feature_extractor

    depth_estimator = DPTForDepthEstimation.from_pretrained(
        "Intel/dpt-hybrid-midas").to("cuda")
    feature_extractor = DPTFeatureExtractor.from_pretrained(
        "Intel/dpt-hybrid-midas")


def get_depth_map(image):
    original_size = image.size

    image = feature_extractor(
        images=image, return_tensors="pt").pixel_values.to("cuda")

    with torch.no_grad(), torch.autocast("cuda"):
        depth_map = depth_estimator(image).predicted_depth

    depth_map = torch.nn.functional.interpolate(
        depth_map.unsqueeze(1),
        size=original_size[::-1],
        mode="bicubic",
        align_corners=False,
    )

    depth_min = torch.amin(depth_map, dim=[1, 2, 3], keepdim=True)
    depth_max = torch.amax(depth_map, dim=[1, 2, 3], keepdim=True)
    depth_map = (depth_map - depth_min) / (depth_max - depth_min)
    image = torch.cat([depth_map] * 3, dim=1)

    image = image.permute(0, 2, 3, 1).cpu().numpy()[0]
    image = Image.fromarray((image * 255.0).clip(0, 255).astype(np.uint8))

    return image
```

Listing 4: Coding the Depth Generator module

3.5 Image Captioning

In addition to adding input images and masked depth conditions to the StableDiffusionXLControlNet module, there is another crucial condition we need to fulfill: prompt embeddings.

As mentioned earlier, in our devised system, the user will provide the image they aim to augment as input at the beginning of the system. In addition to this, the user needs to provide an input prompt condition. When preparing the dataset for CNN model training, if the user wants to change the semantic context of the image, it is more beneficial for both the user and our pipeline to briefly describe it by writing a prompt rather than seeking other image conditions.

In our experiments, relying solely on the prompt data from the user posed a risk to the quality of the model's output. Additionally, there might be scenarios where the user does not want to write a prompt. For these reasons, we found it appropriate to use an image captioner module that automatically generates prompts by analyzing the reference image uploaded by the user.

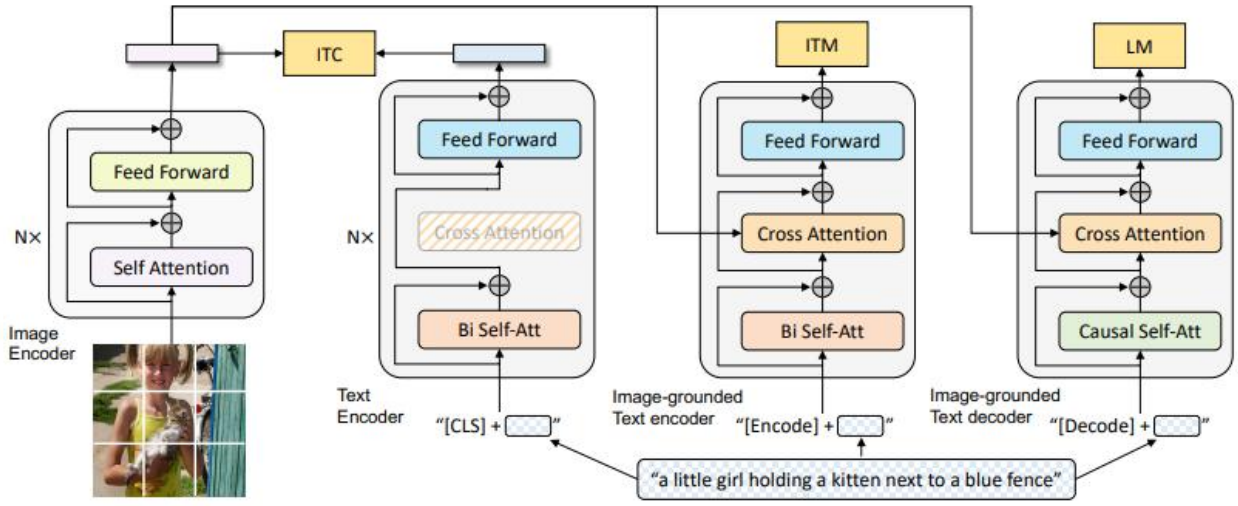


Figure 14: BLIP working principle representation

The core of the Image Captioner module is based on BLIP (Bimodal Learning with Image-Text Pre-training), a unified Vision-Language Pre-training (VLP) framework designed to learn from noisy image-text pairs. The model architecture of BLIP includes a visual transformer and a Multimodal Mixture of Encoder-Decoder (MED) model for multitask learning. The visual transformer breaks down the input image into parts and encodes them as an embedding sequence. The MED model can perform three functionalities in a single forward pass, including (1) single-modal encoding, separately encoding the image and text, (2) image-based text encoding, adding visual information to each block of the text encoder, and (3) image-based text decoding, replacing bidirectional attention layers in the text encoder with causal attention layers. BLIP optimizes three pre-training objectives: Image-Text Contrastive Loss (ITC), Image-Text Matching Loss (ITM), and Language Modeling Loss (LM). These objectives, working in conjunction with forward passes that activate different functionalities, aim to learn and understand from noisy data. This makes BLIP an excellent candidate for both processing images and matching them with appropriate text. Furthermore, its lightweight design allows for quick results without overwhelming the processor. A single-sentence textual output obtained from the Image Captioner is often sufficient for the model, and any misunderstandings that may arise from positive and negative suffix prompts can be tolerated. [6]

The code we wrote for image captioner is as follows:

```
def init():
    global captioner

    captioner = pipeline(
        "image-to-text",
        model="Salesforce/blip-image-captioning-base",
        prompt=PROMPT,
        device=0
    )


def derive_caption(image):
    result = captioner(image, max_new_tokens=20)
    raw_caption = result[0]["generated_text"]
    caption = raw_caption.lower().replace(PROMPT.lower(), "").strip()
    return caption
```

Listing 5: Coding the Image Captioner module

3.6 Stable Diffusion

Stable Diffusion, introduced in 2022, is a deep learning-based text-to-image transformation model. Developed by the CompVis group, it originated at LMU Munich and was released with contributions from EleutherAI and LAION as a project supported by Stability AI, CompVis LMU, and Runway. Initially designed for generating detailed images based on text, this model can also be employed for various tasks such as changing or expanding the content of images.

One key aspect of the Stable Diffusion model is its open-source nature. As a result, our intricate pipeline benefits from this by avoiding any copyright issues when using the model and the generated images commercially. Furthermore, the open-source advantage allows us to leverage a wide community for ready-to-use pre-trained U-net model weights, instead of training the model from scratch for the desired output type. We built the components and pre-trained weights of all diffuser models in our pipeline using the open-source Python libraries PyTorch and Hugging Face.

3.6.1 StableDiffusionXLControlnet Pipeline

When the Controlnet architecture is used in conjunction with stable diffusion, it provides a system that offers users more flexibility in controlling the context of images. Normally, the standard stable diffusion pipeline manipulated the distribution step by step, taking conditions from the user, including both the image encoder and text encoder, in line with these objectives. However, due to the inadequacy of the embedded U-net module to adapt to a wide range of desired tasks, pre-training was necessary. Especially when generating contextually distant images, the existing weights of the U-net architecture localized to a specific task could not meet these demands. This posed challenges for the producer in both preparing the training dataset and providing computational power. The contribution of Controlnet plays a crucial role precisely at this point. Thanks to its shell structure, it enables obtaining the desired outputs with almost an overfitting-like resemblance to the stable diffusion model with very few input conditions, without the need to train the U-net.[21]

As can be seen more clearly in the figure below, an analogy of a shell can be made for the ControlNet architecture. Functioning as only the encoder part of a U-net (where the remaining decoder part contains zero convolution blocks), it collaborates with the middle block of the stable diffusion's U-net module to generate images based on the targeted task. With this structure, instead of training the entire U-net model, we can relieve the user from a significant training burden by training the ControlNet module and providing it with specialized conditions. However, if it is still considered that the desired flexibility is not achieved, one can opt to train only the ControlNet instead of the complete stable diffusion. During the training process of ControlNet, U-net blocks in the stable diffusion model can be frozen, allowing us to exclusively train the ControlNet blocks.

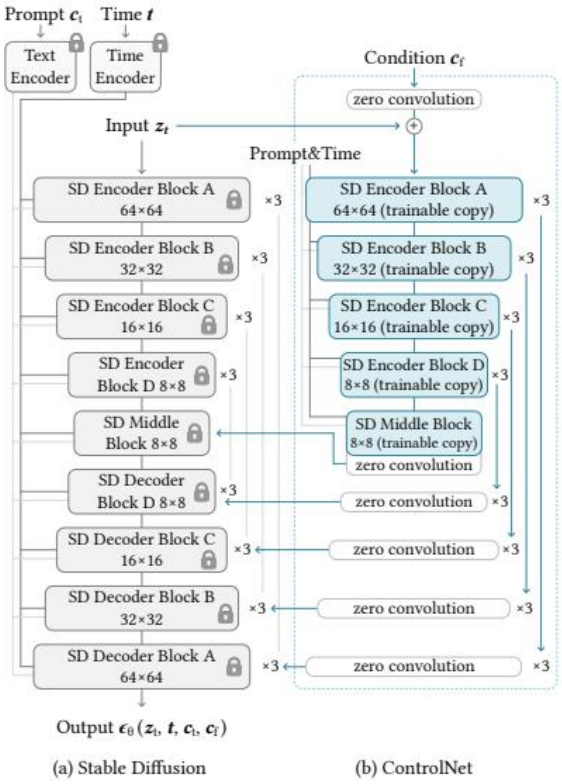


Figure 15: Representation of ControlNet architecture working with Stable Diffusion

The main conditions used in the ControlNet module are shown below.

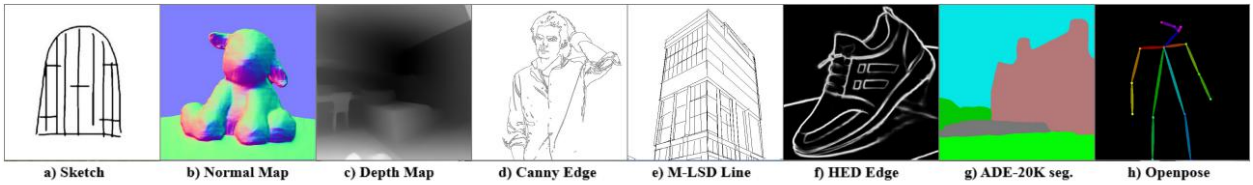


Figure 16: Major ControlNet conditions

We employ the Stable Diffusion ControlNet pipeline in the background replacement task within our developed pipeline. By centering on object images generated from different perspectives provided by the Reference image and the Zero123++ module, we replace the background if available or create a new background from scratch otherwise. We merge the results of the suffix, input prompt, and captioner into a single final prompt, transforming it into a text embedding using Compel. To control the background and content, we utilize text embedding. Throughout this process, we preserve the edges and depth structure of our object by using the mask and depth conditions obtained previously, ensuring not to compromise the structure of our focus, the CNN-trained detection persona, within the ControlNet architecture. Consequently, although our object is regenerated in the background production process from scratch, its context is preserved according to the constraints of ControlNet.

Based on our experiments, we can state that distortions, whether geometric or color-related, occurring in the object, the focal point of the image, as a result of background changes have successfully passed initial assessments made through human vision. It is likely that these distortions will harmonize within the context during future CNN training sessions, and there doesn't seem to

be a negative impact on the model's learning state.

Using the StableDiffusionXLContolnet pipeline, we can perform the background change mentioned above with the following code blocks:

```
compel = Compel(
    tokenizer=[pipe_controlnet_sdxl.tokenizer,
                pipe_controlnet_sdxl.tokenizer_2],
    text_encoder=[pipe_controlnet_sdxl.text_encoder,
                    pipe_controlnet_sdxl.text_encoder_2],
    returned_embeddings_type=ReturnedEmbeddingsType.
    PENULTIMATE_HIDDEN_STATES_NON_NORMALIZED,
    requires_pooled=[False, True]
    )

final_positive_prompt = [f"({caption}){captioner_prompt_weight}, ({positive_prompt
    }){positive_prompt_weight}, ({POSITIVE_PROMPT_SUFFIX}){
    positive_prompt_suffix_weight}"]
positive_prompt_embds, pooled_positive_prompt_embds = compel(
    final_positive_prompt
    )
final_negative_prompt = [f"({negative_prompt}){negative_prompt_weight}, ({
    NEGATIVE_PROMPT_SUFFIX}){negative_prompt_suffix_weight}"]
negative_prompt_embds, pooled_negative_prompt_embds = compel(
    final_negative_prompt
    )
```

Listing 6: Coding the text embeddings for SDXL ControlNet module

```
depth_controlnet = ControlNetModel.from_pretrained(
        "diffusers/controlnet-depth-sdxl-1.0-small",
        use_safetensors=True,
        torch_dtype=torch.float16
    ).to("cuda")

vae = AutoencoderKL.from_pretrained(
        "madebyollin/sdxl-vae-fp16-fix",
        torch_dtype=torch.float16
    ).to("cuda")

scheduler = UniPCMultistepScheduler.from_config("diffusers/controlnet-depth-sdxl
    -1.0-small",
                                                    subfolder="scheduler"
                                                    )
pipe_controlnet_sdxl = StableDiffusionXLControlNetPipeline.from_pretrained(
        ""diffusers/controlnet-depth-sdxl-1.0-small"",
        controlnet=[depth_controlnet],
        scheduler=scheduler,
        vae=vae,
        variant="fp16",
        use_safetensors=True,
        torch_dtype=torch.float16,
    ).to("cuda")
```

Listing 7: Coding the SDXL ControlNet module

3.6.2 StableDiffusionInpaint Pipeline

If the goal is to use only the mask condition on Stable Diffusion, instead of employing the ControlNet architecture, the image encoder within the standard stable diffusion model can be used.

Since the mask data carries only two types of pixel values (black and white), pixels on the mask overlapping with the input condition image will correspond to the reference image's coordinates for white pixels. While the coordinates corresponding to white pixels undergo changes in the reference image, the remaining pixel regions will remain constant. This method is referred to as inpainting, and it serves to achieve more precise outputs by applying diffusion only on the desired region, rather than altering all components of the existing image.

We integrated the inpainting method into our pipeline based on the anticipation that, during the training of Convolutional Neural Network (CNN) models in the future, the convolutional network would tend to focus solely on the main object at the center of the dataset after a certain point. By aligning with the approach that suggests this behavior could persist in various future instances, resulting in a decline in accuracy, we aim to facilitate a more effective understanding of context by enabling the CNN model to explore different positions of the image in the primary image space, which has been shifted.

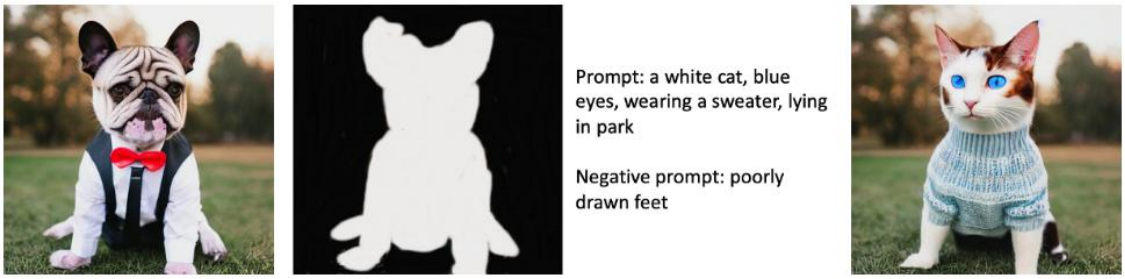


Figure 17: Representative working principle of the inpainting process

In our designed pipeline, we will employ the inpainting method to generate additional regions around the image and perform diffusion operations on these selected regions. In this process, initially, we create a padding area containing black pixels around the images with altered backgrounds, preserving the entire area of the images obtained from the previous step of the StableDiffusionXLControlnet pipeline. Subsequently, in the next step, we generate a mask where white pixels correspond to the pixels in the padding area, and black pixels correspond to the pixels in the reference image. This way, a new background is created around the reference image with a stable background, while maintaining the altered background, and additional contextually suitable information (assisted by prompts, potentially incorporating text encoder conditions for a more comprehensive output) is generated.

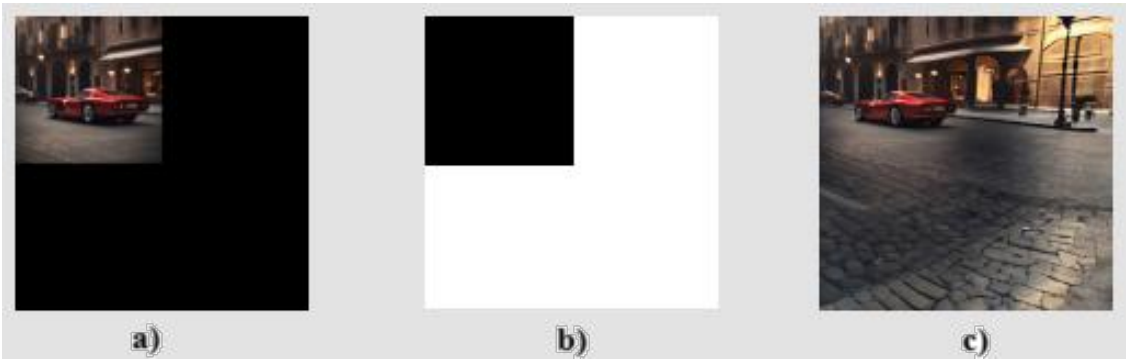


Figure 18: Changing the position of the reference object in the image using the Inpainting method. a) Padding with black pixels b) Relevant mask to match the position and dimensions of the reference image c) Result

By using this process, we can increase the number of augmented images by iteratively changing the position of the reference image coming from the controlnet pipe in the padding area. The codes required for the inpainting process are as follows.

```
compel = Compel(tokenizer=pipe_outpaint.tokenizer,
                text_encoder=pipe_outpaint.text_encoder
                )
positive_prompt_embeds = compel(f"({positive_prompt},{POSITIVE_PROMPT_SUFFIX}).
    blend({positive_prompt_weight},{positive_prompt_suffix_weight})")
negative_prompt_embeds = compel(f"({negative_prompt},{NEGATIVE_PROMPT_SUFFIX}).
    blend({negative_prompt_weight},{negative_prompt_suffix_weight})")
```

Listing 8: Coding the text embeddings for Stable Diffusion Inpaint module

```
pipe_outpaint = DiffusionPipeline.from_pretrained(
    "stabilityai/stable-diffusion-2-inpainting",
    torch_dtype=torch.float16,
    revision="fp16"
    ).to('cuda')

pipe_outpaint.scheduler = EulerAncestralDiscreteScheduler.from_config(
    pipe_outpaint.scheduler.config
    )

pipe_outpaint.vae = AsymmetricAutoencoderKL.from_pretrained(
    "cross-attention/asymmetric-autoencoder-kl-x-2",
    torch_dtype=torch.float16
    ).to('cuda')
```

Listing 9: Coding the Stable Diffusion Inpaint module

The outputs we obtained at the end of this pipeline are the final results of the complex pipeline we designed, and now we can use them to prepare an augmented data set to be used in CNN training.

4 Analyzing of Results

In summarizing our accomplishments thus far, in Section Two, we elaborated on the components of diffusion models and their operational principles, introducing our pipeline to the readers. We prepared the third section by duplicating the reference image received by our pipeline's Zero123++ module, generating numerous perspective images for a comprehensive view. Subsequently, we obtained mask, depth, and masked depth conditions from these images, incorporating them into the stable diffusion XL ControlNet pipeline. Additionally, we incorporated user prompts as an extra condition, acquiring prompts both with prompt suffixes and through an image captioner, utilizing them as text embeddings in the controlnet. The manipulated image with altered background was then processed in the inpainting module, iteratively modifying the positional information and ultimately changing the background one last time to achieve our final augmented results.

In this paper, we share with you the main outputs for evaluation purposes. We will examine the visual and context quality of the outputs. We will use the following 3 separate reference images in both the SDXL Img2Img pipeline and the pipeline we designed and publish the outputs.

(a) The Bag (b) The Cat (c) The Fire Hydrant

Figure 19: Reference images to use for the SDXL img2img pipeline

In addition, the prompts we will use with these images are as follows; for the bag: *"in a display window in the store"*, for the cat: *"on the streets of New York"*, for the fire hydrant: *"on a snowy, romantic street"*

4.1 Stable Diffusion XL Img2Img Pipeline Results

The outputs of the examples we created initially using reference images with the Stable Diffusion XL image to image model are listed below. Outputs are grouped according to the reference images they belong to. Since the synthetic image is produced in one go, there are no intermediate steps.

Figure 20: Synthetic bag images produced by the SDXL Img2Img pipeline

Figure 21: Synthetic cat images produced by the SDXL Img2Img pipeline

Figure 22: Synthetic fire hydrant images produced by the SDXL Img2Img pipeline

When we look at the images of the synthetic bags produced, we see that they are all very similar to the reference image: they all have the same front face and strap handle stance. They are all aligned in the center of the image. Two of them have more background elements, adding extra context to the images, but the others are against a plain background. For this reason, depth perception also weakens. Despite this, the texture of the bags is well produced and the picture looks photographic quality. Changing the brand text on the front of some bags has added diversity.

When we look at the synthetic cat images produced, we see that they are very similar to the reference image. The stance and breeds of cats are very similar to each other. They are all aligned in the center of the image. The model automatically added depth perception to the image by blurring the background.

When we look at the images of synthetic fire hydrant produced, we see that the diversity comes from the change in the location and shape of the valves of the hydrant. They are all aligned in the center of the image. Depth perception is enhanced because the model reproduces the background as a city.

Upon conducting a qualitative analysis of the content of all these images, we believe that augmenting the dataset prepared for training a CNN model may prove insufficient and lead to overfitting. This is attributed to the fact that even the six images generated from a reference image closely resemble each other in terms of content. When the task of augmenting a dataset heavily relies on SDXL Img2Img, we consider the augmentation factor to be crucial. Without an adequate augmentation factor, the CNN model is likely to work consistently on similar examples. Moreover, since the model tends to focus on the center of the images, overfitting may occur over time.

4.2 Our Pipeline Results

The results of our pipeline are listed below. Images that occur in the intermediate step from the reference image to the final output have also been added. Images are grouped according to their belonging to their reference images. The order of the figures in each group is the same as the working order of the intermediate steps of the pipeline.

Figure 23: Multiple images of the bag image courtesy of Zero23++

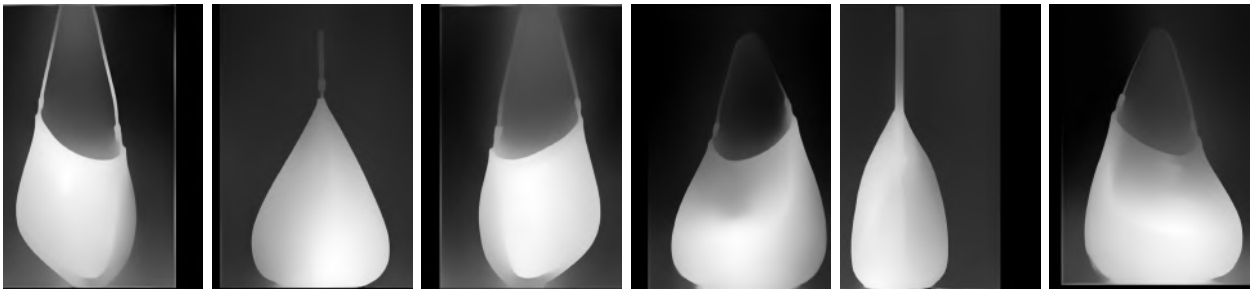


Figure 24: Depth images of the bag multiple images

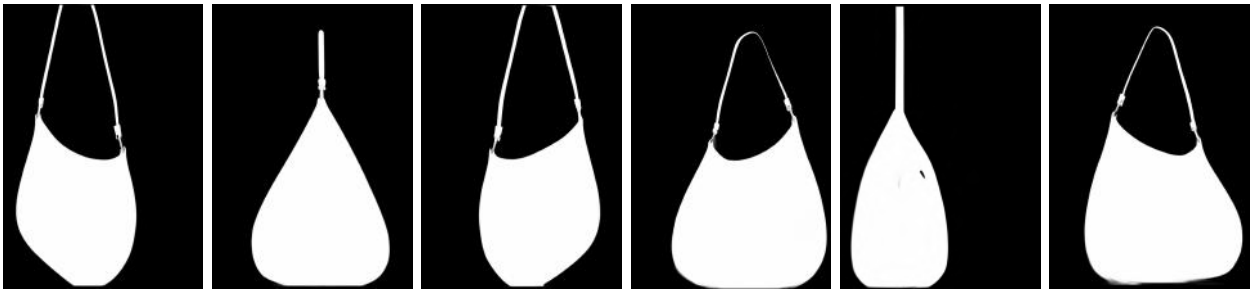


Figure 25: Mask images of the bag multiple images

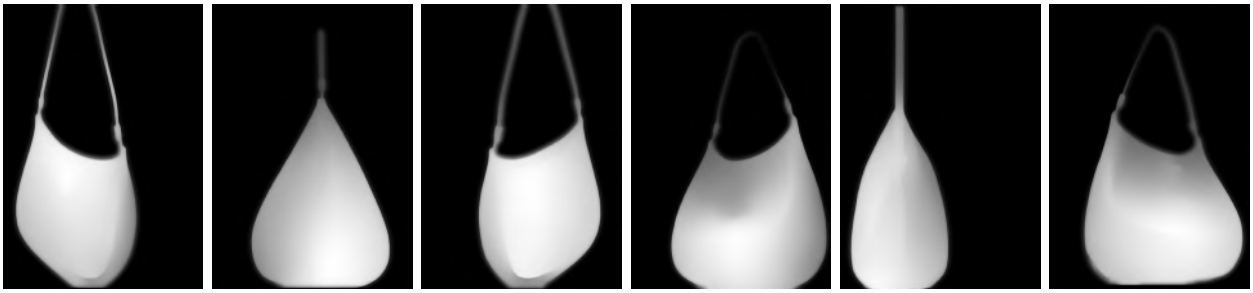


Figure 26: Combining the bag mask and depth images

Figure 27: Creating backgrounds for the bag images

Figure 28: Shifting the bag object to the corners of the image with Inpainting

It is evident from the above images that the Zero123++ pipeline has successfully replicated the reference bag. Consequently, the depth, mask, and masked depth images have been generated successfully. The lines appearing on the sides in the depth images are a result of cropping the outputs of Zero123++, but naturally, they disappear during the mask process. In the background generation, a new bag is created along with the background based on the masked depth, resulting in significant stylistic differences in the bag. Subsequently, during the inpainting process, the bags are effectively positioned at the extremes of the new image and seamlessly integrate with the background, maintaining a cohesive visual appearance.

Figure 29: Multiple views of the cat image courtesy of Zero23++

Figure 30: Depth images of the cat multiple images

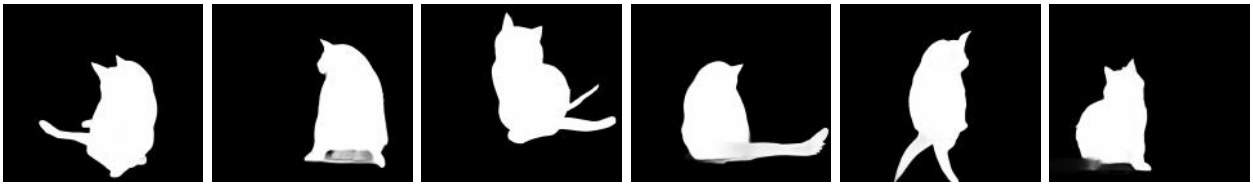


Figure 31: Mask images of the cat multiple images

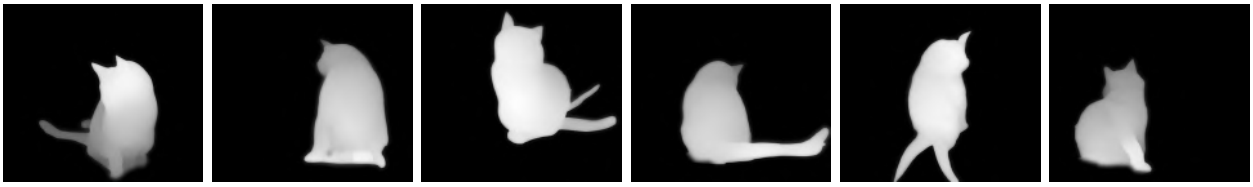


Figure 32: Combining the cat mask and depth images

Figure 33: Creating backgrounds for the cat images

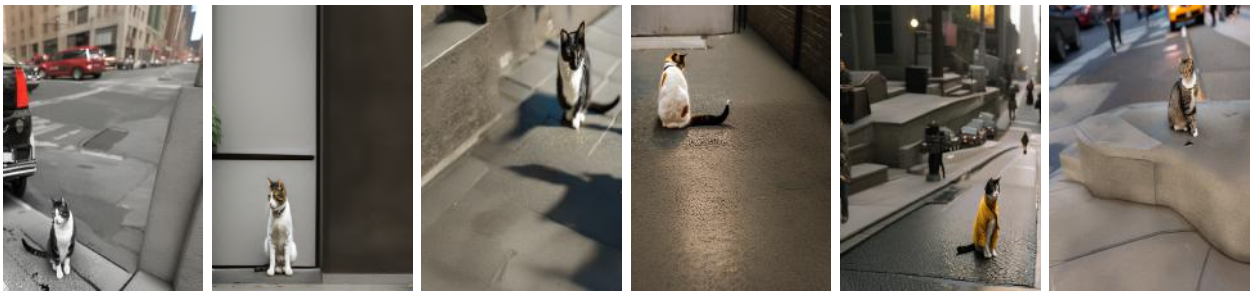


Figure 34: Shifting the cat object to the corners of the image with Inpainting

The images above reveal that Zero123++ has produced distorted visuals while generating cats. Although the degree of distortion in the resulting images is not excessively high, efforts have been made to mitigate this distortion in the depth, mask, and masked depth images. Subsequently, during the background replacement process, cats, along with the background, have been recreated. Through the process of recreating the image from scratch, it is evident that the outputs of Zero123++ can be tolerated. Following this, the outpainting process has been employed to disperse the cats into street scenes, thereby solidifying the contextual coherence.

Figure 35: Multiple views of the fire hydrant image courtesy of Zero23++

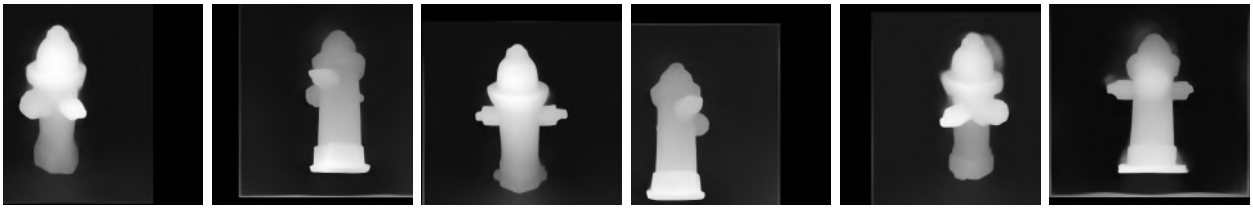


Figure 36: Depth images of the fire hydrant multiple images

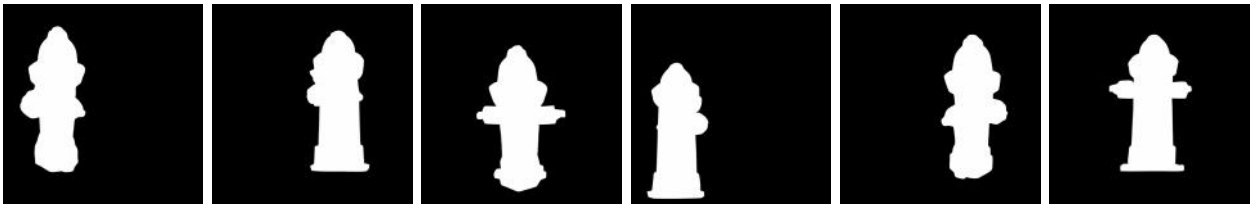


Figure 37: Mask images of the fire hydrant multiple images

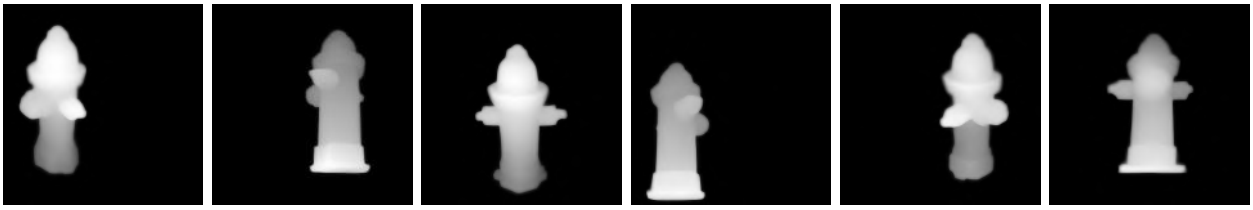


Figure 38: Combination of the fire hydrant mask and depth images

Figure 39: Creating backgrounds for the fire hydrant images

Figure 40: Shifting the fire hydrant object to the corners of the image with Inpainting

As clearly observed in the above images, Zero123++ has successfully replicated the fire hydrant. The primary reason for this success can be attributed to the simplistic geometric structure of the fire hydrant. In the subsequent depth processing, it is evident that blurring and dispersal occur around the valves and top of the hydrant. However, these distortions are effectively tolerated through the distortion, mask, and the associated masked depth processes. As a result, both the background

and, consequently, the fire hydrant itself have been successfully reproduced. The image quality of the generated outputs is remarkably clear, and even in the subsequent inpainting step, despite the reduction in size within the image, the fire hydrant maintains its realism and contributes to the overall visual coherence.

Upon a final comparison between our own pipeline and the SDXL Img2Img pipeline, the following statements can be made unequivocally: Our designed pipeline possesses advantages that the SDXL Img2Img pipeline can never independently achieve, as it rotates the main object, alters its structure through recreation, and effectively utilizes the entire surface of the image by shifting within. The SDXL Img2Img pipeline, by only introducing slight modifications on and around the reference image, is prone to significant overfitting issues in future CNN training when augmenting the dataset. This is likely to result in a substantial decrease in accuracy. In contrast, we believe that our pipeline, despite duplicating the image, consistently generates new elements while maintaining contextual coherence. Consequently, we are confident that it can successfully train the CNN model without inducing overfitting.

Another crucial point, as evident in the above examples, is that our pipeline, with its inherent cascading step structure, acts as a kind of safety belt. It creates a safety net that can tolerate errors that may arise in intermediate steps, providing resilience. Thanks to this feature, our pipeline gains a significant advantage over the standard SDXL Img2Img pipeline, which lacks any intermediate steps.

As a result of all these evaluations, we conclude that our intricately designed pipeline, during the data set augmentation stage for training a CNN model, outperforms Stable Diffusion XL, one of the most well-known artificial intelligence models in the market for generating visuals, in terms of differentiating and duplicating reference images.

5 Gathering of the Datasets

We can divide the type of data to be used in our project into two: real images obtained by web scraping over the internet, and synthetic image data produced with our generation pipeline. We chose the content of our data as pants, t-shirts and shoes for this project.

We can obtain our real dataset from the product catalogs of various online clothing stores. For this purpose, the web addresses of the most popular shopping sites were used, and images were randomly selected, including both men's and women's clothing. Afterwards, duplicate images were cleaned up first. Subsequently, the available images underwent basic manual filtering based on their content. One of the main goals here was to remove unnecessary elements from the images obtained from shopping sites, which could negatively affect the learning of the CNN model, such as advertisements, product tables, or extreme zoom shots. Another goal was to minimize the possibility of multiple labels for different classes in a single image when tagging the images. To achieve this, examples that maximized the appearance of the relevant class in the image were preferred.

6 Analyzing of the Datasets

We divided the dataset to be used for the CNN model into three categories: a dataset composed entirely of real images, a dataset composed entirely of synthetic images, and a mixed dataset created by combining real and synthetic images in a 1:1 ratio.

The primary purpose of creating and using the mixed dataset in CNN model training is to determine the most effective use of synthetic data. By doing so, we can closely examine the model's tolerance to the realism distortion of synthetic data and better define the potential industrial applications of this pipeline in the future.

The dimensions of the images in all datasets are the same, 768x1024 pixels. Additionally, each dataset has an equal volume, with 467 images in each set. Before being input into the CNN model, these images will be resized while preserving the aspect ratio and then used for training.

6.1 Real Dataset

For the real datasets, we tried to select examples that best represent each class. We aimed to minimize the presence of different classes within the same image to prevent potential labeling errors in the future. Additionally, we filtered out data with numerous other objects in the background, adhering strictly to the context of commercial clothing catalog photography. Below is an example for each class:

(a) T-Shirt (b) Pants (c) Shoes

Figure 41: One Example for Each Real Data Class

- *Image Dimensions*: 1200x1800
- *Number of Data in Each Class*: 467

6.2 Synthetic Dataset

During the creation of real image sets, we adjusted the text embeddings to emphasize the subject heading in a commercial fashion as much as possible. For reference images, we created a synthetic image set by selecting random samples for each class from the data set we obtained above.

(a) T-Shirt (b) Pants (c) Shoes

Figure 42: One Example for Each Synthetic Data Class.

- *Image Dimensions*: 768x1024
- *Number of Data in Each Class*: 467

6.3 Real-Synthetic Mixture Dataset

This dataset was created with a half-mix of synthetic and real images from each class. Two examples from each class, one real and one synthetic, are listed below.

(a) T-Shirt (Real) (b) Pants (Real) (c) Shoes (Real)

(d) T-Shirt (Synthetic) (e) Pants (Synthetic) (f) Shoes (Synthetic)

Figure 43: Two Samples Each for Mixed Data Set

- *Image Dimensions*: 768x1024
- *Number of Data in Each Class*: 467

7 Training of the Classifiers

In this section, we will first introduce the CNN model we will use. Afterwards, we will evaluate the training process and results.

7.1 Examining of The Model

The CNN model selected for use in this project is EfficientNet. EfficientNet, developed by Google and introduced in 2019, offers the significant advantage of achieving high accuracy with fewer parameters. The primary innovation of EfficientNet is the compound scaling method, which balances scaling the model's width, depth, and resolution. Traditional CNN models typically scale in only one dimension, whereas EfficientNet scales all three dimensions together. This approach makes the model more efficient and allows it to perform exceptionally well on various tasks. Designed using Neural Architecture Search (NAS), EfficientNet optimizes every layer and parameter, resulting in a more compact and effective model. This automated design process ensures that EfficientNet is highly efficient, which is a significant advantage in scenarios with limited computational resources. Additionally, EfficientNet's suitability for transfer learning makes it possible to retrain the model on different datasets and achieve high performance across various tasks[17].

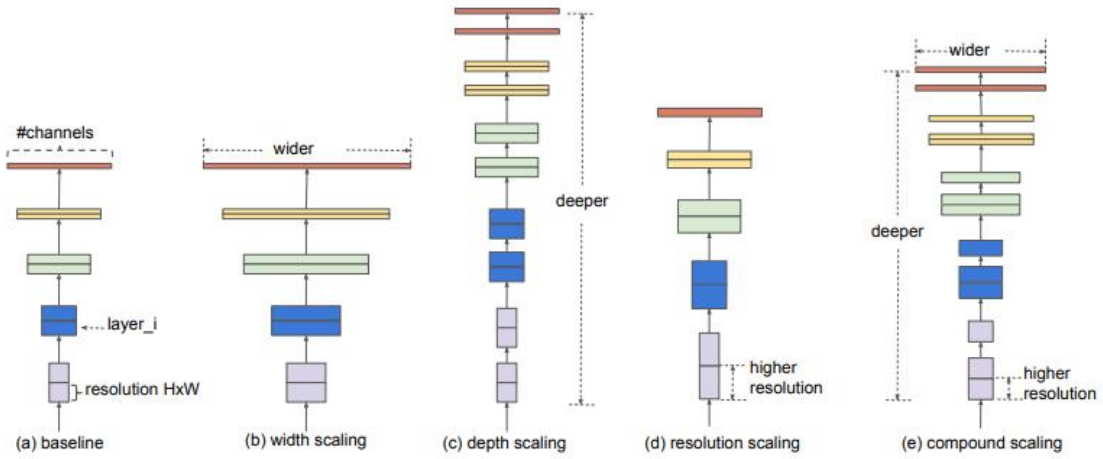


Figure 44: (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

These advantages of EfficientNet will enable our CNN model to effectively grasp the context of images with less GPU power and in a shorter time, despite the fact that our dataset volumes are significantly smaller compared to the standard dataset volumes used in industry training. Additionally, as a result of its efficient performance with smaller datasets, EfficientNet will indirectly reduce the GPU burden on the generation pipeline for producing synthetic data, thereby requiring the generation of less synthetic data for this project.

7.2 Training Phase

Using the EfficentNet model, we can perform our training for three separate cases. The platform on which we conduct training is RunPod[14].

RunPod is a cloud-based platform where machine learning models can be trained and developed. For logging our training sessions, we used Wandb. Wandb is an open-source platform that enables real-time logging of the training and validation processes of artificial intelligence models[20].

The trainings we performed on a virtual machine instance in RunPod were logged in wandb.

7.2.1 Equipment Source

The equipment we used during the training is as follows:

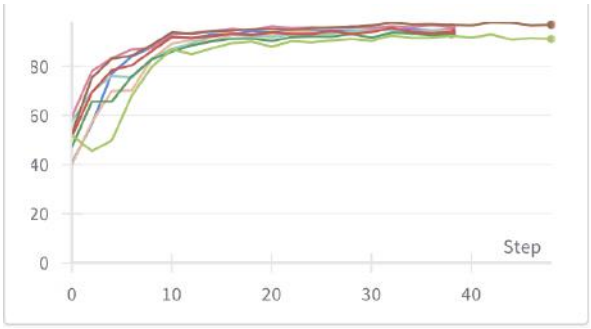
- *CPU count*: 48
- *GPU count*: 1
- *GPU type*: NVIDIA RTX A5000

7.2.2 Training Technics

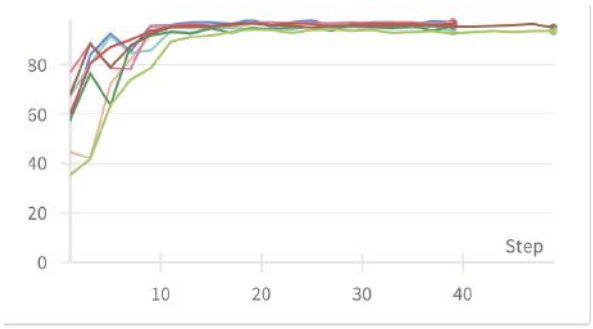
Due to the limited time and number of virtual machine instances, the managed process and applied methods are as follows:

- Training was carried out on a single virtual machine instance.
- Training was done in parallel on a single GPU. For this reason, in order to use GPU VRAM most effectively:
  - The smallest model of the EfficientNet model architecture family was preferred.
  - The method of using the weights of the model trained on the ImageNet-1k dataset (transfer learning) was used.
  - Training, validation and testing data sets were set to be 1501 in total for each training and testing process.

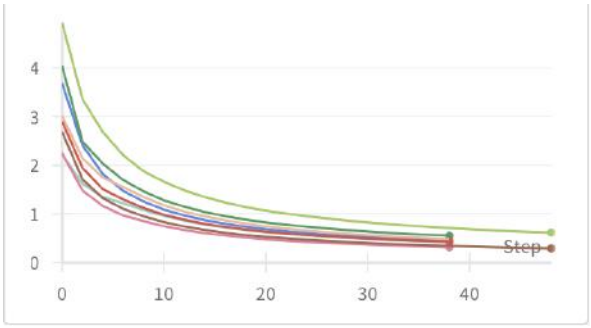
First of all, we focused on the training process using a synthetic data set. In this process, hyperparameter tuning was performed with the transfer learning method mentioned above. After approximately 15 training and testing periods, hyperparameters were determined. Graphics of the hyperparameter tuning process:



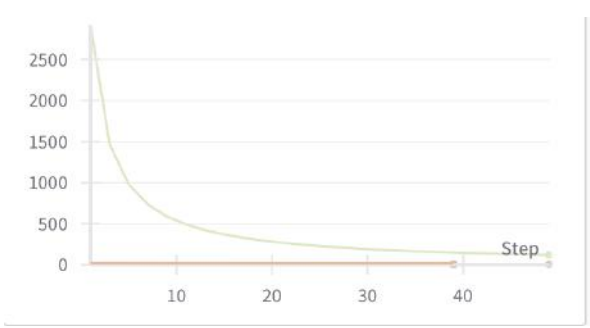
(a) Train Accuracy



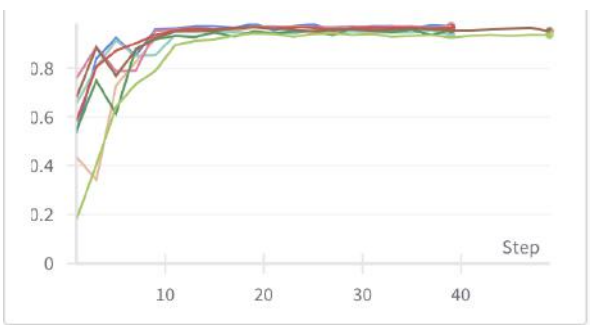
(b) Validation Accuracy



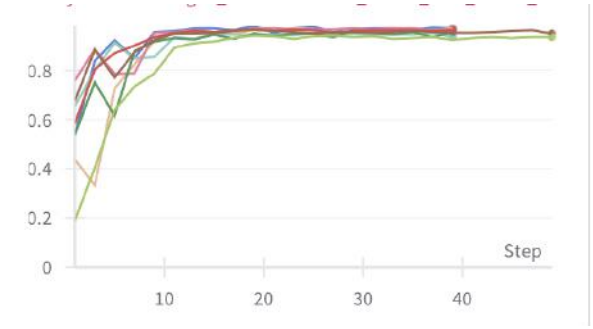
(c) Train Loss



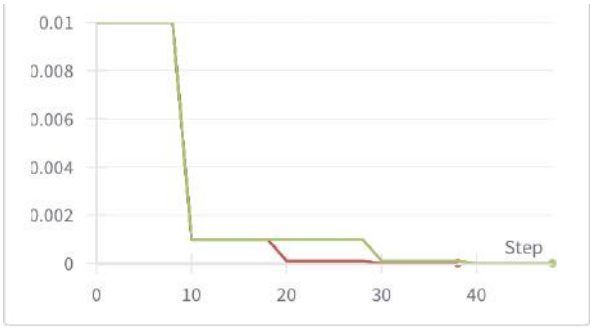
(d) Validation Loss



(e) Macro F1 Score



(f) Weighted F1 Score



(g) Learning Rate Graph

Figure 45: Synthetic Dataset Training with Hyperparameter Tuning graphics

The config file showing the hyperparameters of the model is as follows. These settings remained the same for all three training processes. This is because the focus of our study is not on finding the best training process but rather on examining the results of training conducted under equal conditions with datasets of different contexts[16].

```
{
  "lr": {
    "desc": null,
    "value": 0.01
  },
  "arch": {
    "desc": null,
    "value": "efficientnet_b3.ra2_in1k"
  },
  "root": {
    "desc": null,
    "value": "/workspace/classification"
  },
  "seed": {
    "desc": null,
    "value": 42
  },
  "size": {
    "desc": null,
    "value": 324
  },
  "epoch": {
    "desc": null,
    "value": 20
  },
  "optim": {
    "desc": null,
    "value": "adam"
  },
  "t_max": {
    "desc": null,
    "value": 10
  },
  "_wandb": {
    "desc": null,
    "value": {
      "t": {
        "1": [1,5,41,49,53,55,63,80],
        "2": [1,5,41,49,53,55,63,80],
        "3": [13,16,23],
        "4": "3.10.12",
        "5": "0.17.0",
        "8": [5],
        "13": "linux-x86_64"
      },
      "framework": "torch",
      "start_time": 1717256471,
      "cli_version": "0.17.0",
      "is_jupyter_run": false,
      "python_version": "3.10.12",
      "is_kaggle_kernel": false
    }
  },
  "device": {
    "desc": null,
    "value": "cuda"
  },
  "dataset": {
    "desc": null,
    "value": "syntethic-train-gen"
```

```
  },
  "version": {
    "desc": null,
    "value": "0.16"
  },
  "aug_mode": {
    "desc": null,
    "value": "standart"
  },
  "momentum": {
    "desc": null,
    "value": 0.9
  },
  "save_path": {
    "desc": null,
    "value": "results"
  },
  "scheduler": {
    "desc": null,
    "value": "step"
  },
  "batch_size": {
    "desc": null,
    "value": 16
  },
  "checkpoint": {
    "desc": null,
    "value": ""
  },
  "patience_lr": {
    "desc": null,
    "value": 10
  },
  "wandb_entity": {
    "desc": null,
    "value": "synthetic-training"
  },
  "lr_milestones": {
    "desc": null,
    "value": [5,10,15,40]
  },
  "is_pretrained": {
    "desc": null,
    "value": true
  },
  "scheduler_gamma": {
    "desc": null,
    "value": 0.1
  }
}
```


Listing 10: Training Congig File

7.3 Training Results

We can evaluate the training results for each case separately. The metrics we will consider and the graphs illustrating these metrics are the same for each case: learning rate, train & validation accuracy, train & validation loss, and F1 macro and weighted scores.

In these three different training scenarios, the performance of the fully synthetic and synthetic-real mixed datasets will be compared to the performance of the fully real dataset.

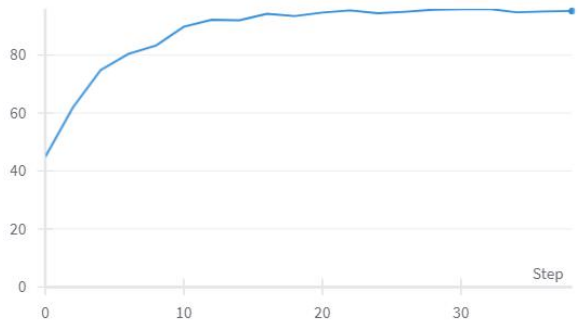
7.3.1 Real Dataset Training Results

First of all, we can see the success of the training more easily by looking at the train & validation loss and accuracy graphs.

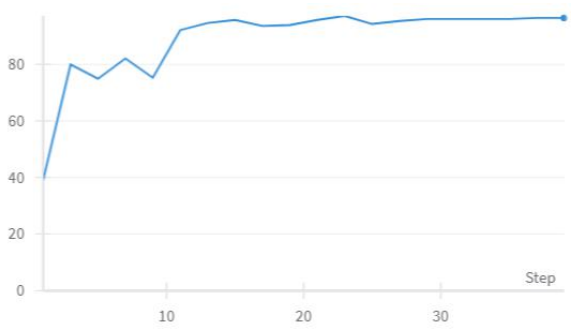
Metric	Value
Train Accuracy %	95.2721
Training Loss	0.4149
Validation F1 Macro Score	0.9642
Validation F1 Weighted Score	0.9644
Validation Loss	0.3625
Validation Accuracy %	96.4286

As seen in the train accuracy graph, it reaches a saturation point around the 10th epoch and remains constant for the remaining epochs. We can also clearly observe a similar behavior in the validation graph. The initial fluctuations give way to stable progress towards saturation around the 15th epoch.

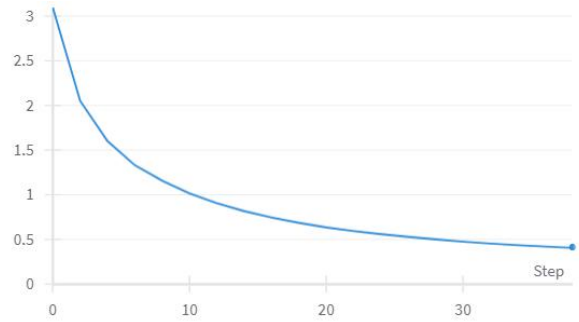
In parallel, the train & validation loss graphs show that the loss values decrease in harmony with each other as the number of epochs increases.



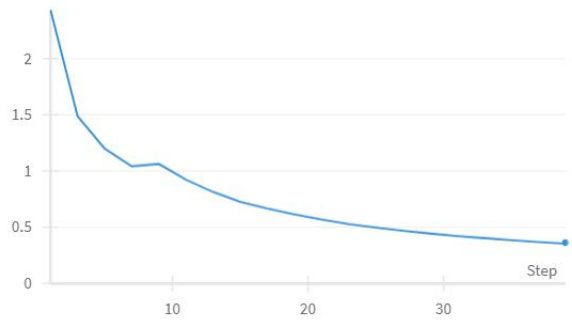
(a) Train Accuracy



(b) Validation Accuracy



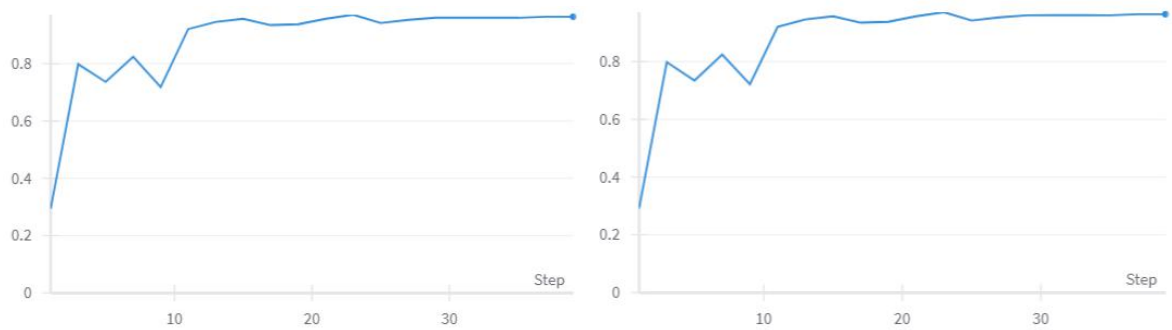
(c) Train Loss



(d) Validation Loss

Figure 46: Real Dataset Train and Valid Loss-Acc. graphics

Based on these accuracy and loss graphs, we can confidently say that the convergence of the train and validation curves over increasing epochs indicates the absence of overfitting. Furthermore, the convergence gaining stability in later epochs indicates that the model is not underfitting.



(a) Macro F1 Score (b) Weighted F1 Score

Figure 47: F1 Score Graphs

F1 macro and weighted scores are important metrics that demonstrate my model's classification ability. When looking at the Macro F1 score, it's evident that the model's ability to classify different classes reaches its peak and saturates within a short time, around the 13th epoch. This pattern is also followed by the weighted F1 score. Since the volumes of classes in our dataset are equal, its result is almost identical to the Macro F1 score.

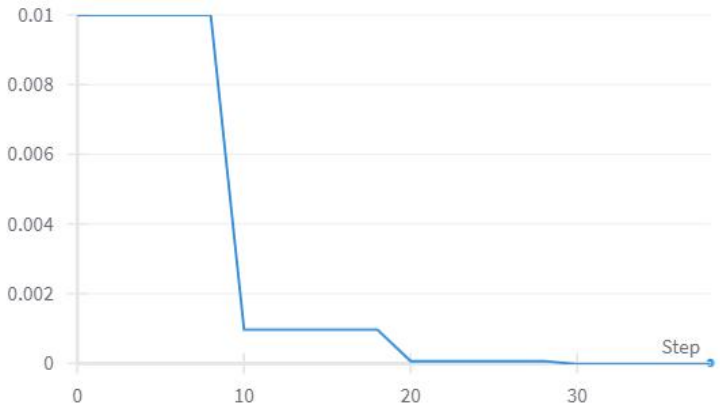


Figure 48: Learning Rate Graph

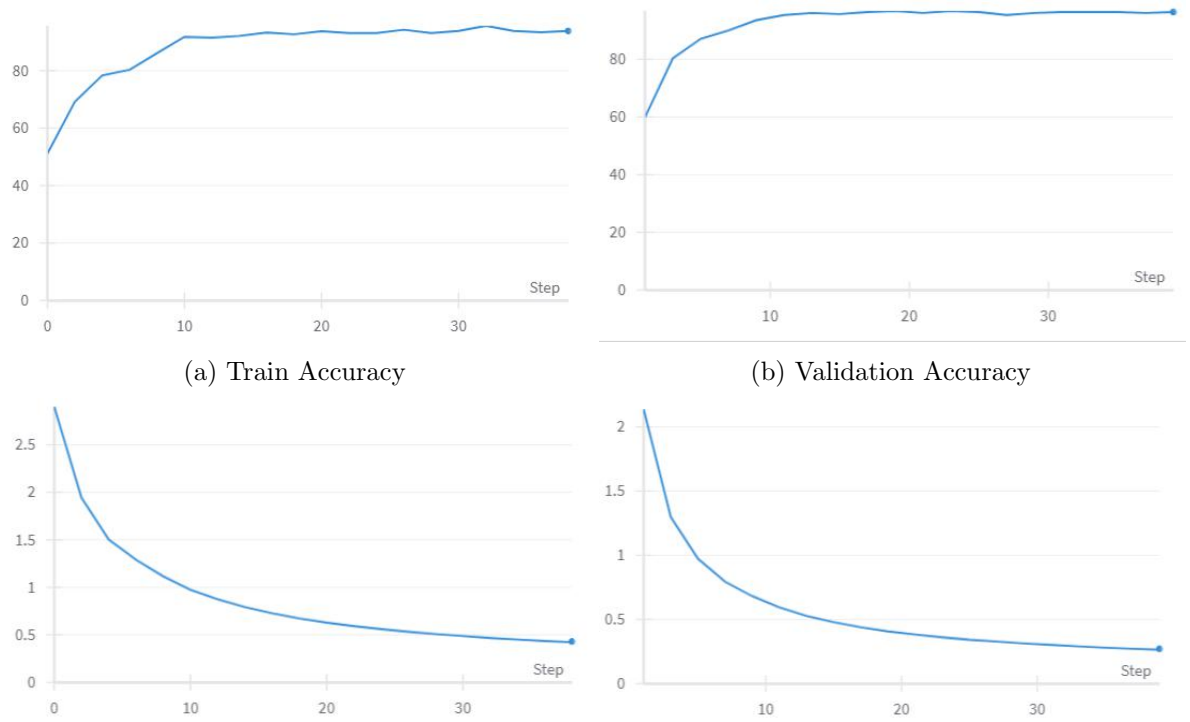
Throughout the training process, the learning rate hyperparameter decreases every 10 epochs, further refining the training process towards the end. The advantage of this is that towards the final epochs, the model has already grasped the overall context of the classes, and the gradient descent process can be further refined to minimize the loss for the remaining contextual details.

7.3.2 Synthetic Dataset Training Results

When we look at the training results of the data set consisting entirely of synthetic images, it can be clearly seen that the results are very similar to the completely real data set.

The train and validation accuracies reach their highest attainable values around the 10th epoch and remain constant for the remaining epochs. Similarly, the train and validation losses exhibit a smooth decrease in parallel. Based on this, we can conclude that the model is neither underfitting nor overfitting.

Metric	Value
Train Accuracy %	93.84
Training Loss	0.4314
Validation F1 Macro Score	0.9640
Validation F1 Weighted Score	0.9641
Validation Loss	0.2715
Validation Accuracy %	96.42



(a) Train Accuracy

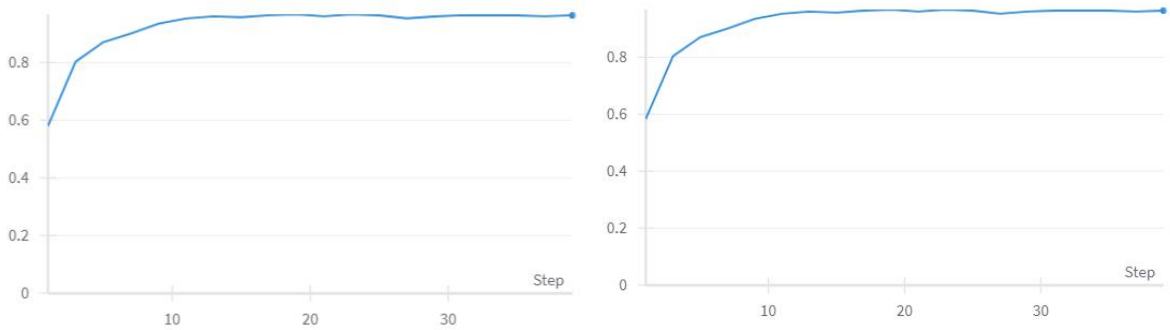
(b) Validation Accuracy

(c) Train Loss

(d) Validation Loss

Figure 49: Synthetic Dataset Train and Valid Loss-Acc. graphics

The sudden increase followed by stability in the progression of both Macro and weighted F1 scores before reaching the 10th epoch, without any fluctuations as seen in the fully real dataset, may indicate that the synthetic dataset brings out a more effective context for the CNN. Due to the homogeneous distribution of the dataset, the weighted F1 score exhibits a performance similar to the Macro F1 score.



(a) Macro F1 Score (b) Weighted F1 Score

Figure 50: F1 Metrics

The steady decrease in the learning rate continued for this training as well.

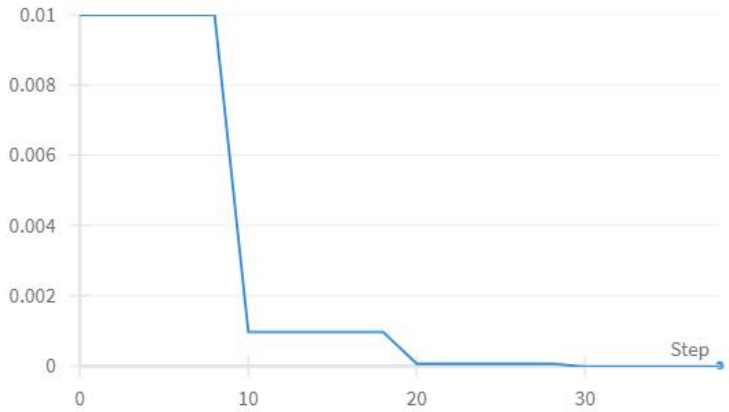


Figure 51: Learning Rate Graph

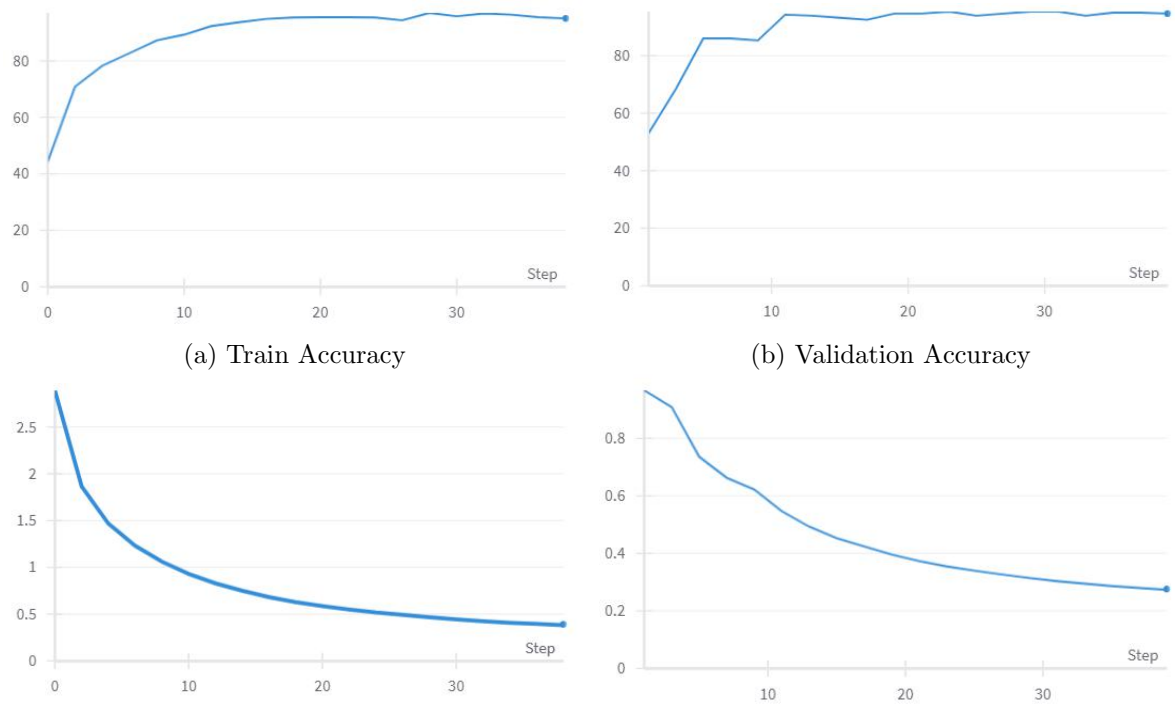
7.3.3 Mixture Dataset Training Results

When we look at the training results of the dataset consisting of a mixture of synthetic and real images, we see that the result we obtained is slightly better than the completely synthetic dataset.

Metric	Value
Train Accuracy %	95
Training Loss	0.3911
Validation 1 Macro Score	0.9463
Validation F1 Weighted Score	0.9463
Validation Loss	0.2755
Validation Accuracy %	94.64

The train and validation accuracies reaching their saturation points quickly and remaining stable there, similar to the others, indicate that the training is quite successful. The initial fluctuations in validation accuracies may result from the diversity introduced into the dataset due to the mixture of contexts, and as observed, the model has optimized with increasing epochs according to the dataset. As seen in the loss graphs, although the validation loss is slightly higher than the train loss,

both have converged smoothly and reached saturation. This indicates that the model is neither overfitting nor underfitting.

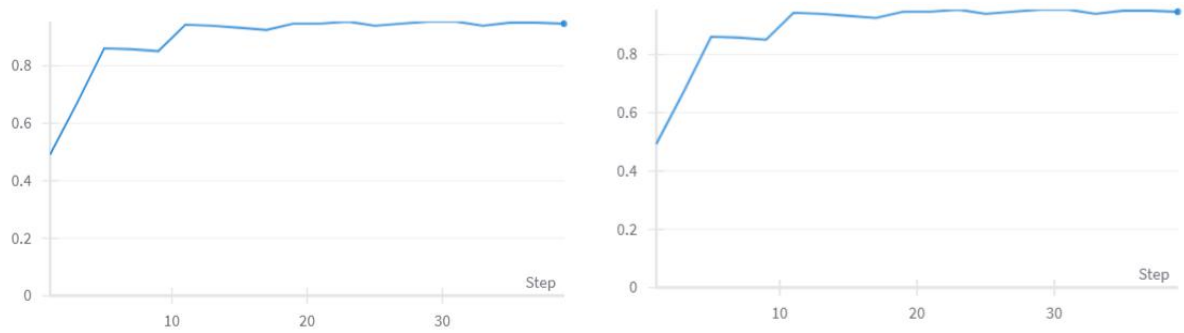


(a) Train Accuracy (b) Validation Accuracy

(c) Train Loss (d) Validation Loss

Figure 52: Real Dataset Train and Valid Loss-Acc. graphics

Unlike the entirely synthetic data, the F1 scores did not increase rapidly at once but rather increased gradually. The reason for this could be the increased diversity in the content of the dataset due to the mixture of contexts mentioned above. This fluctuation has given way to stability from the 10th epoch onwards, and our model's performance in class differentiation has reached its peak.



(a) Macro F1 Score (b) Weighted F1 Score

Figure 53: F1 Metrics

As in other training cases, the learning rate value decreased every 10 epochs during this training.

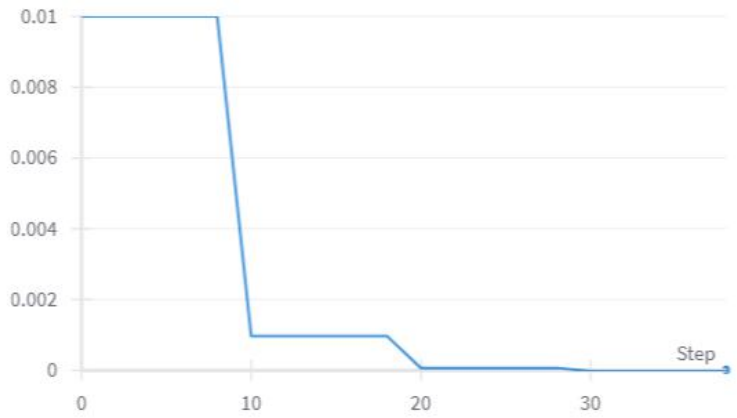


Figure 54: Learning Rate Graph

8 Inference on the Test Dataset

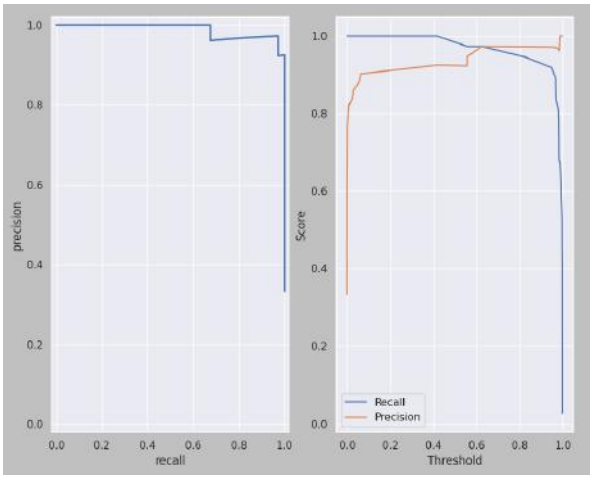
In the previous section, the accuracies of each training case were very close to each other, and the F1 scores indicate that the model performs well in classification. In this section, we will analyze the test data to examine the differences between these three training scenarios more closely. Additionally, we will highlight the differences in the test results between entirely synthetic and synthetic-real mixed data compared to the entirely real training data. We will then draw conclusions based on these differences in the subsequent sections.

8.1 Real Model Test Results

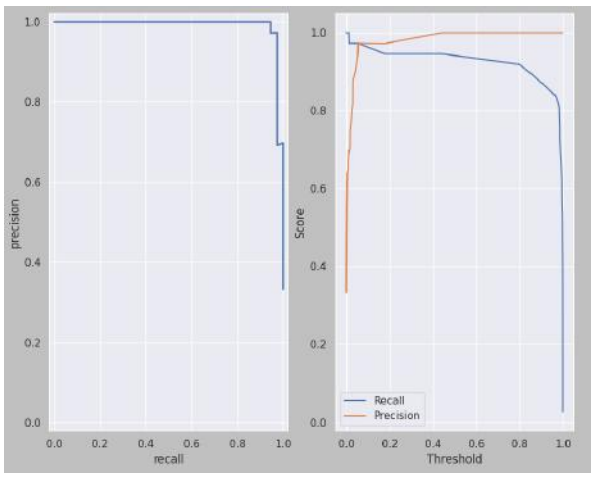
The test results for the dataset composed entirely of real images are as follows. We can begin by interpreting the recall-precision curves for each class. The recall-precision curve on the left shows the model's performance at various thresholds. At the beginning of the curve, the precision value is at 1.0, and the recall value starts low and increases. This indicates that the model initially makes very confident positive predictions, and all of these predictions are correct. As the epochs progress (especially around the 10th epoch), the recall value increases while the precision value starts to decrease slightly. At this stage, the model tries to capture more positive examples, which may include some false positives.

In the graph on the right, we observe the change in precision and recall based on the decision threshold. In the early epochs, the model achieves high recall by identifying a large number of positive examples at low thresholds, but the precision is low. This indicates that the model covers a broad positive set but contains many false positives. As the threshold value increases, especially between the 5th and 10th epochs, precision starts to rise, and recall decreases. This indicates that the model becomes more selective, and false positives decrease. After the 20th epoch, the model's performance becomes more stable, with both precision and recall values remaining consistently high.

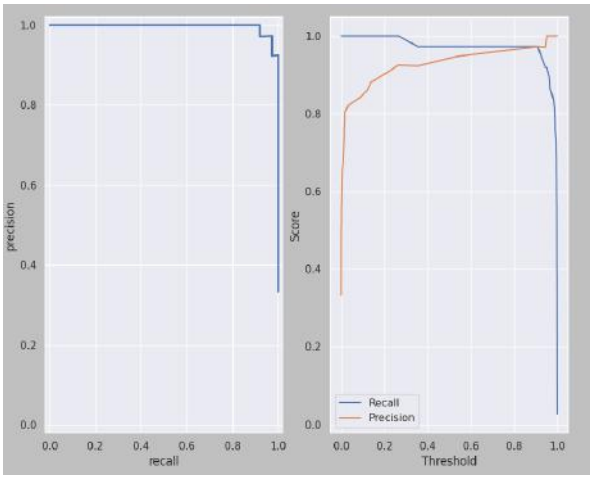
When these two graphs are evaluated together, it is observed that the model's learning curve is steep at the beginning and middle of the training process (epochs 0-20), with significant improvements made. Particularly after the 10th epoch, there is a noticeable improvement and stabilization in both recall and precision metrics. This indicates that errors are reduced during the model's training process, and its predictions become more reliable.



(a) T-shirt Recall-Precission Chart



(b) Pants Recall-Precission Chart



(c) Shoes Recall-Precission Chart

Figure 55: Recall-Precission Graph for Each Class

By looking at the ROC-AUC graph, we can see the classification performance of the CNN model for three separate classes. The area under the curve (AUC) represents the success of the classification, with the curve closer to the top-left corner indicating better classification performance. The area under the curve tells us the ratio of True and False for the relevant class. In this case, the ROC curves being very close to each other suggests that the overall classification for the dataset is nearly homogeneous. It also indicates that the classification performance is slightly better for distinguishing between t-shirts and shoes classes compared to the pants class.

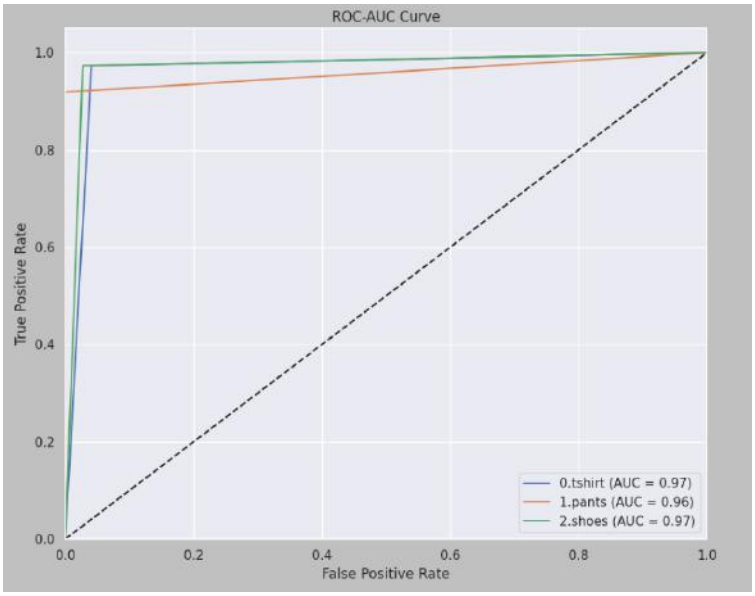


Figure 56: ROC-AUC Graph

The pie chart below visualizes the distribution of results obtained from the classification of the dataset, which was initially evenly distributed in terms of volume, after training. We can easily see that the classification is evenly distributed without any concentration on a particular class.

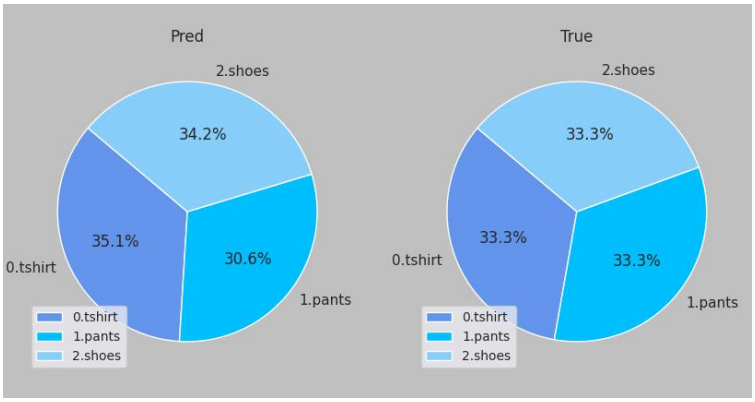


Figure 57: Distribution of Classes Pie Chart

When we look at the confusion matrix, we clearly see that it is diagonalized. This indicates that the training was quite successful. There are few misclassifications in t-shirt and shoe data, but there are no errors in classifying the pants data.

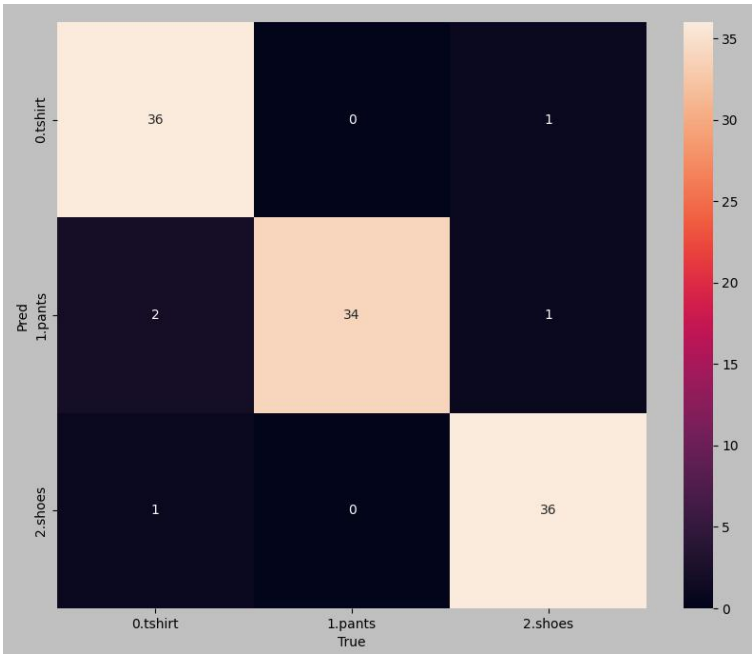


Figure 58: Confusion Matrix

The classification report summarizing all these graphs above is as follows:

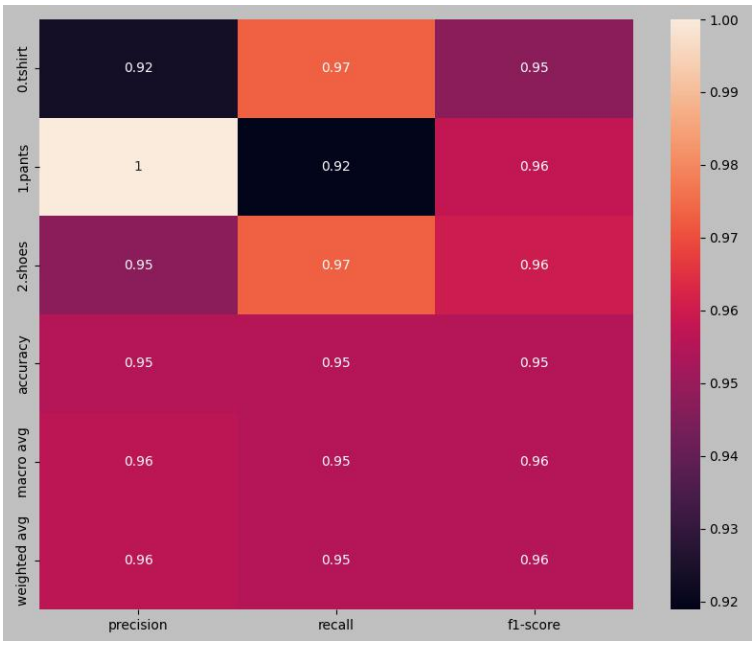


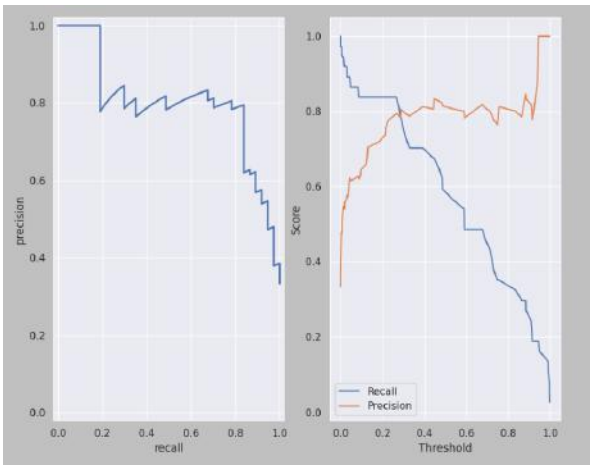
Figure 59: Classification Report

8.2 Synthetic Model Test Results

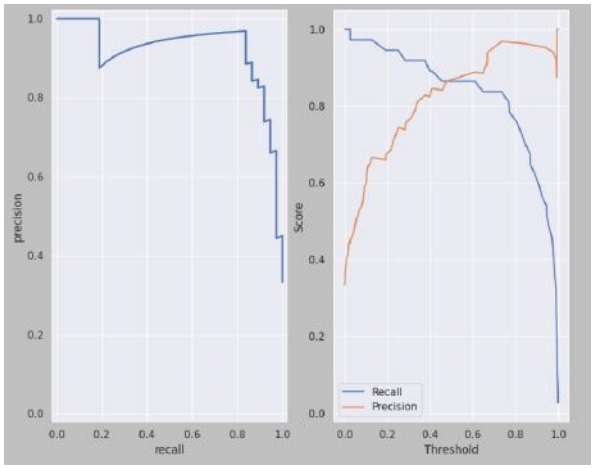
When we look at the recall-precision curves of the CNN model trained on entirely synthetic image data, unlike the entirely real dataset training examined in the previous section, the curves appear to be quite fluctuating and deviate from the ideal curve, which should converge to the

top-right corner. This indicates that our model is showing a tendency towards inconsistency.

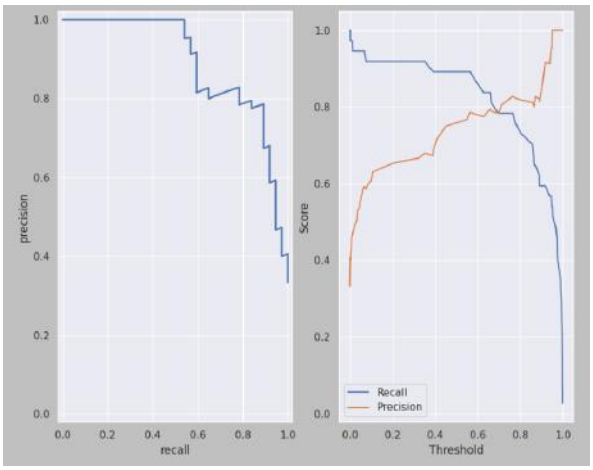
Breaks can be observed across increasing thresholds due to the inconsistent or heterogeneous nature of features in synthetic images. This is an indication of the model's instability and its failure to fully capture the context.



(a) T-shirt Recall-Precission Chart



(b) Pants Recall-Precission Chart



(c) Shoes Recall-Precission Chart

Figure 60: Recall-Precission Graph for Each Class

When we look at the ROC-AUC graph, we see parallel results to the recall-precision curve. The ROC curve tends to be diagonalized, and consequently, the AUC area decreases, indicating a decrease in the ability to distinguish between the relevant classes.

Additionally, in this graph, contrary to the test results of the CNN model trained on entirely real data, we observe a divergence between classes. The model has worsened its classification performance by obtaining inconsistent results in the t-shirt data while better distinguishing pants.

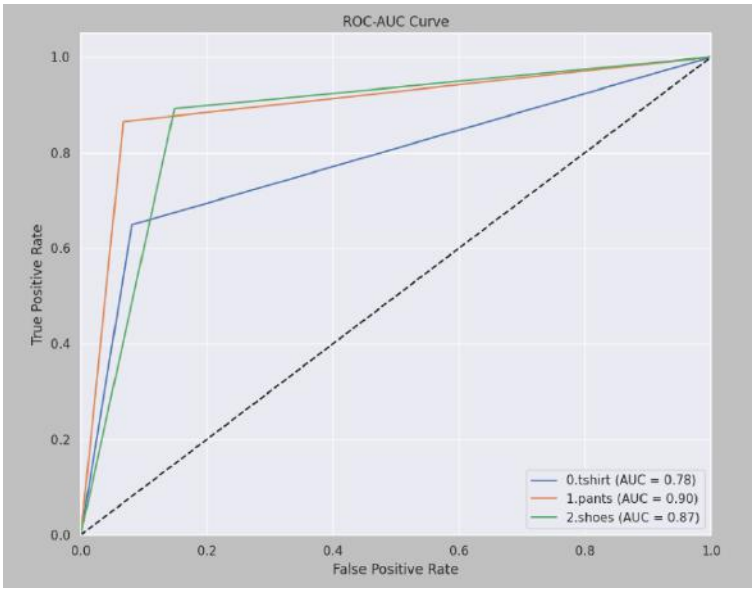


Figure 61: ROC-AUC Graph

This distinction between classes in the ROC-AUC chart can be seen more clearly in the heterogeneous distribution on the pie chart. While most of his guesses were shoes, he had difficulty perceiving t-shirts.

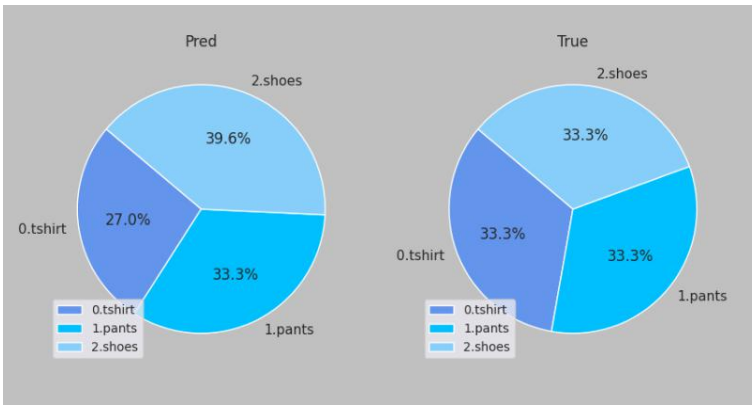


Figure 62: Distribution of Classes Pie Chart

In addition to what was explained above, this situation can also be seen in the confusion matrix. The diagonality that should exist has begun to disappear. The high percentage of shoe predictions also significantly increased the number of misclassifications made in this class.

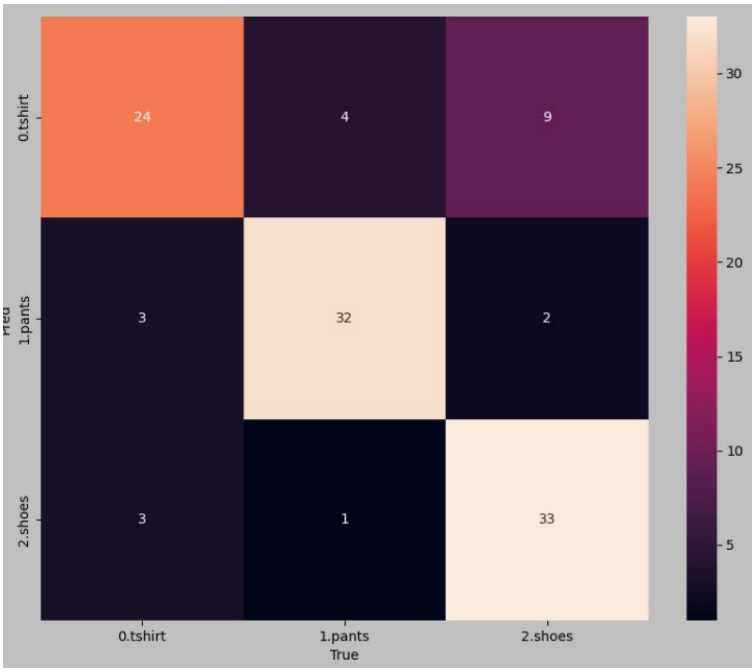


Figure 63: Confusion Matrix

According to the classification report, which shows precision, recall, and F1 score values for each class, the most successful class in this training case has been pants, while the poorest performance results have been obtained for t-shirts.

Overall, when examined alone, a CNN model trained on entirely synthetic image data demonstrates above-average classification performance. However, when compared to a CNN model trained solely on entirely real data, it falls short, especially in terms of content similarity.

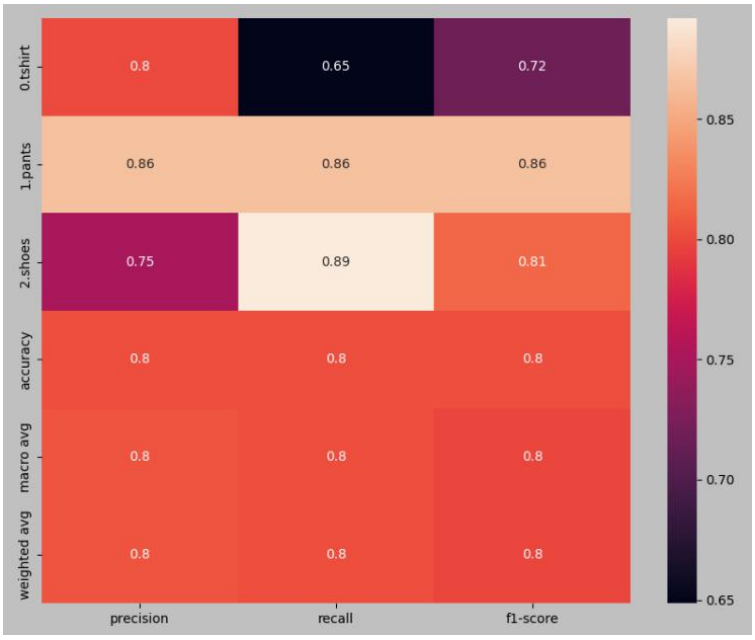
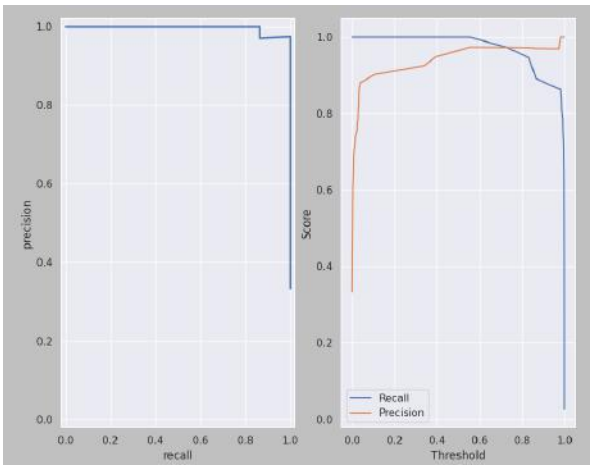


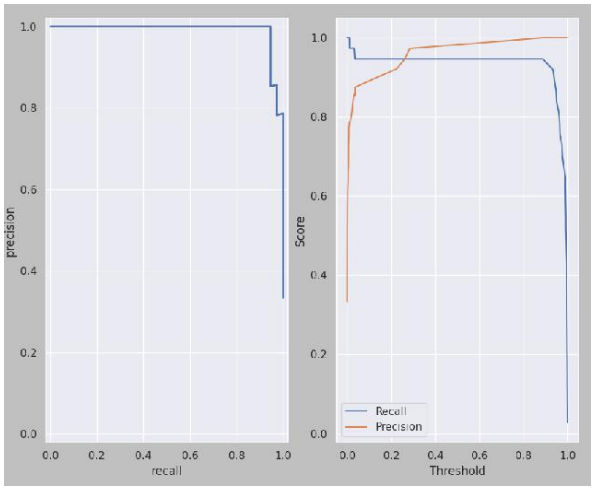
Figure 64: Classification Report

8.3 Mixture Model Test Results

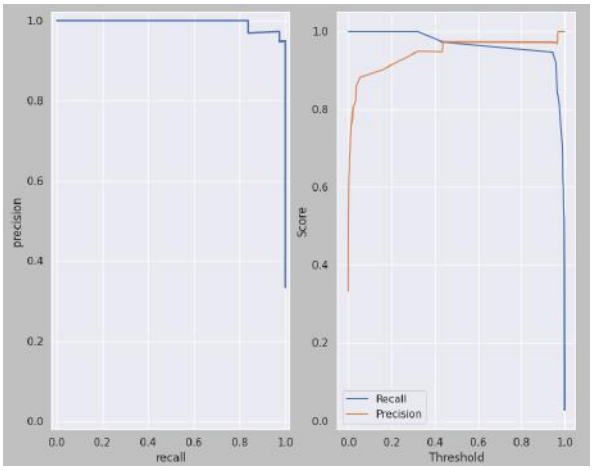
The test results of the CNN model trained on a dataset composed of a mixture of synthetic and real images are as follows. The precision-recall curves closely resemble those of the model trained solely on real data. The impact of threshold increment on the score shows a more consistent increase. Additionally, the curve representing the recall-precision ratio approaches the top-right corner, indicating a close approximation to the ideal results. The absence of a fluctuating pattern is an important factor demonstrating the consistency of the classification results.



(a) T-shirt Recall-Precission Chart



(b) Pants Recall-Precission Chart



(c) Shoes Recall-Precission Chart

Figure 65: Recall-Precission Graph for Each Class

Parallel results to the effects of the regularity of the precision-recall curve can also be observed in the ROC-AUC graph. The AUC scores are close to each other, with only a slight difference from the results of the model trained solely on real data. Here, the curve behaves significantly away from diagonalization, approaching the ideal top-left corner to maximize true positive rates.

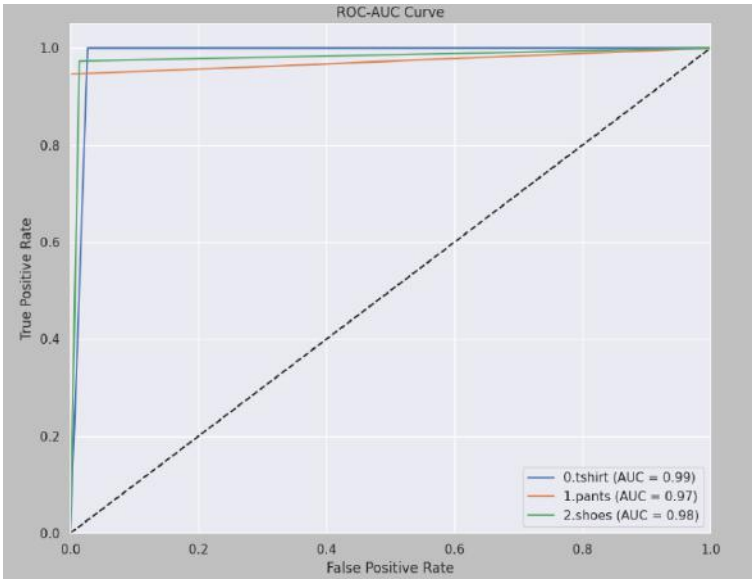


Figure 66: ROC-AUC Graph

We can visualize the implications of the ROC-AUC curves' proximity more clearly on a pie chart. Based on the prediction results here, it wouldn't be an exaggeration to say that the classification ability is more homogeneous and consistent compared to the entirely real model.

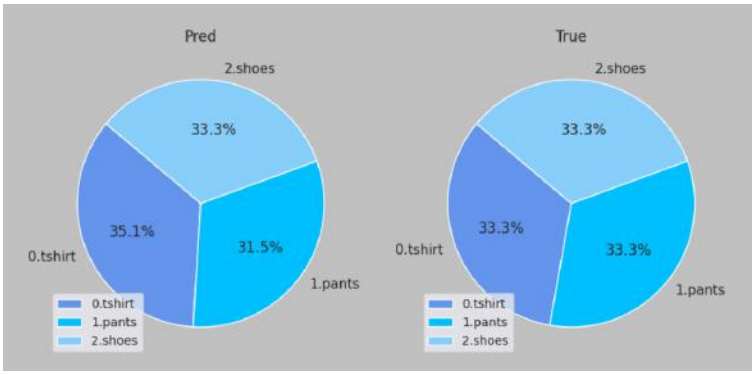


Figure 67: Distribution of Classes Pie Chart

The diagonality on the confusion matrix is at a level that rivals the real data model. There were a few cases of misclassification, but all others were clearly marked correctly.

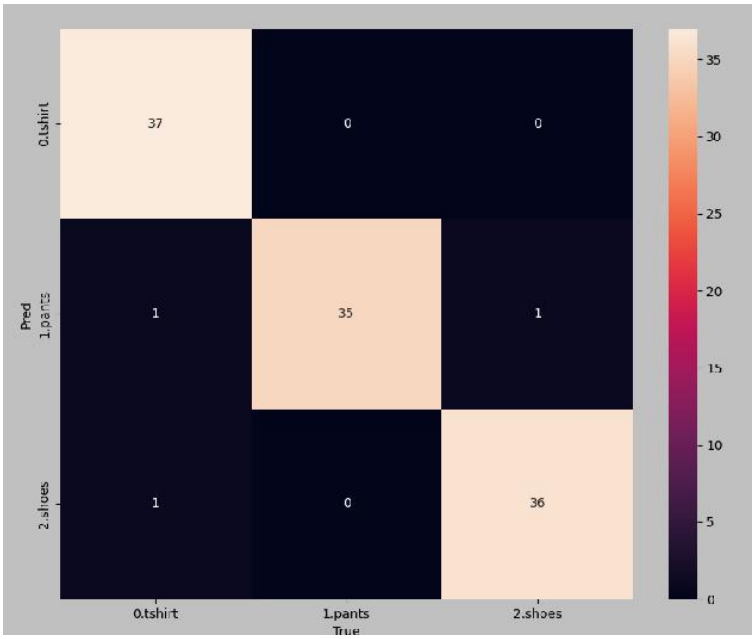


Figure 68: Confusion Matrix

When we look at the classification report, we see that each class is equally successful. We can say that this success is even better than the dataset trained solely on real data. Mixing synthetic data with real images has provided a versatile training opportunity by increasing both the precision and sensitivity of the system.

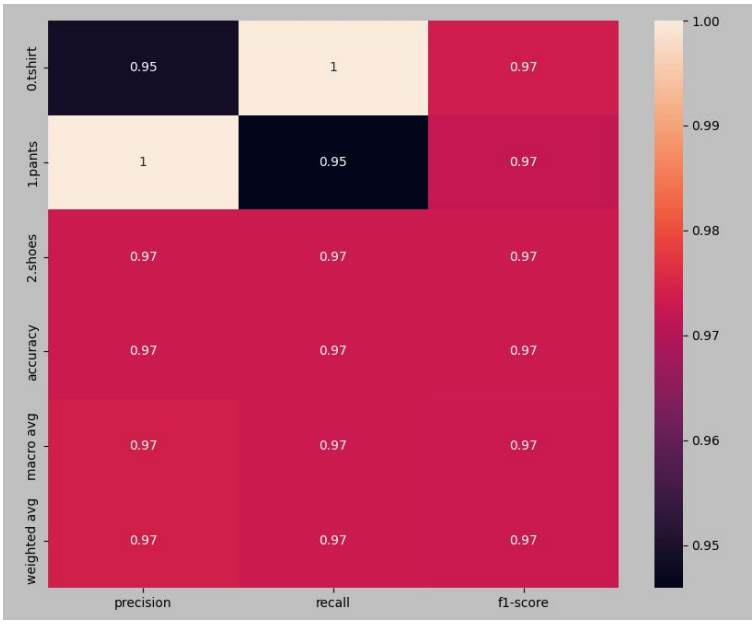


Figure 69: Classification Report

9 Performance Analysis

Firstly, if we analyze the performance of CNN models trained with completely synthetic and completely real image datasets, we can clearly say that training with entirely real image data has dominated, yielding more precise results compared to the other. One of the reasons for this could be the possible overlapping labels in the images, such as the presence of both pants and shoes in the same image. Although this may affect the consistency of the CNN model, the same conditions apply to the real image dataset, and the CNN model still manages to achieve consistent and precise results. One main reason for the synthetic image data producing such heterogeneous and inconsistent results could be the reference image perception by CLIP and the subsequent SDXL ControlNet module. While generating synthetic image data, the clothing data may have led the captioner module to produce erroneous text embeddings due to factors like lighting, background, perspective, or folding, resulting in disconnected images from the context. Another reason could be the direct diffusion models evolving noise with schedulers over time to create a probability-dependent image. Thus, the parts assisting the CNN model in understanding the context of the image may have become mixed and shared across all classes. For example, when designing the lower part of a shoe, extending the upper part to resemble pants is highly probable. Therefore, due to these reasons, the volume of the dataset may have been insufficient for the CNN to learn the context effectively.

When comparing the synthetic-real image dataset with the entirely real dataset, the situation is reversed. Both models have very similar results in both training and testing, with the mixed model slightly outperforming the real dataset. Both models have made extremely consistent and homogeneous predictions. This is likely because mixing synthetic and real data neutralized each other's weaknesses, resulting in a dataset rich in both precision and recall. Real data has tolerated the complex and contextually disconnected aspects of synthetic data, while synthetic data has contributed to the contextual diversity of real data, allowing the CNN model to perceive the context in a multifaceted manner.

Based on this, we can conclude that synthetic data, as our main focus, should be mixed with real data and used in CNN training. This not only serves as an augmentation method to increase data volume but also enriches the context of the dataset, thereby improving classification performance.

10 Potential Application Areas

The use of synthetic data in training CNN models will be particularly beneficial for datasets that are rare, scarce, or composed of uniform content, where CNN models may otherwise struggle. In addition to standard CNN data augmentation techniques, the diversification of data with synthetic data while preserving context will serve as another factor in preventing overfitting. Furthermore, the blending of real and synthetic data to create a mixed dataset will become a primary focus, surpassing the goal of data augmentation in terms of contextual richness.

Companies in the image processing field will either have to use ready-made datasets or create their own datasets for the models they want to train. In cases where the contents of datasets are outdated, of low quality, or lacking variety, synthetic images generated with a selected reference batch can be manipulated within the embedding space to create previously non-existent images that meet the company's specific requirements. Additionally, by incorporating movement of background and objects on the screen, the desired context can be further reinforced.

Another crucial aspect of using synthetic data is copyright laws. In situations where the use of artworks or real-world street views is necessary for training, synthetic data becomes essential, especially when obtaining appropriate official permissions for real-world data may be challenging. Synthetic data generated from legally obtained limited images are currently not subject to copyright

issues, as they are stochastic and originate from the user's original creation. Hence, they do not pose a legal problem for AI models trained on them.

11 Conclusion

Using synthetic image data to train CNN models offers many advantages and shows great promise for the future. In this study, we developed a sophisticated pipeline that replicates and multiplies the input reference image using various artificial intelligence models to maximize efficiency in CNN model training. The primary goals of constructing this pipeline were to synthesize as many images as possible from a reference image and to automate this process. Additionally, we conducted a general comparison of the outputs of our designed pipeline with the most widely used open-source model currently available on the market, Stable Diffusion XL. Following this, we created datasets for CNN model training, using both data obtained via web scraping from online shopping sites and synthetic images produced by our pipeline. We prepared these datasets, which consist of three classes, in three variations: entirely real, entirely synthetic, and a mixture of synthetic and real images. Upon evaluating the outputs of these three cases, we concluded that the most efficient method for preparing the dataset is to mix the generated synthetic images with real images in a 50/50 ratio.

References

[1] Zitong Cheng. *Sampler Scheduler for Diffusion Models*. arXiv:2311.06845 [cs]. Nov. 2023. URL: http://arxiv.org/abs/2311.06845.

[2] Zilin Du et al. *Training on Synthetic Data Beats Real Data in Multimodal Relation Extraction*. arXiv:2312.03025 [cs]. Dec. 2023. URL: http://arxiv.org/abs/2312.03025 (visited on 06/09/2024).

[3] Tero Karras et al. *Elucidating the Design Space of Diffusion-Based Generative Models*. arXiv:2206.00364 [cs, stat]. Oct. 2022. URL: http://arxiv.org/abs/2206.00364.

[4] Diederik P. Kingma and Max Welling. "An Introduction to Variational Autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4 (2019). arXiv:1906.02691 [cs, stat], pp. 307–392. ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000056. URL: http://arxiv.org/abs/1906.02691.

[5] Diederik P. Kingma and Max Welling. *Auto-Encoding Variational Bayes*. arXiv:1312.6114 [cs, stat]. Dec. 2022. URL: http://arxiv.org/abs/1312.6114.

[6] Junnan Li et al. *BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation*. arXiv:2201.12086 [cs]. Feb. 2022. URL: http://arxiv.org/abs/2201.12086.

[7] Hezheng Lin et al. *CAT: Cross Attention in Vision Transformer*. arXiv:2106.05786 [cs]. June 2021. URL: http://arxiv.org/abs/2106.05786.

[8] Ruoshi Liu et al. *Zero-1-to-3: Zero-shot One Image to 3D Object*. arXiv:2303.11328 [cs]. Mar. 2023. URL: http://arxiv.org/abs/2303.11328.

[9] Ben Mildenhall et al. *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*. arXiv:2003.08934 [cs]. Aug. 2020. URL: http://arxiv.org/abs/2003.08934.

[10] Xuebin Qin et al. *Highly Accurate Dichotomous Image Segmentation*. arXiv:2203.03041 [cs]. July 2022. URL: http://arxiv.org/abs/2203.03041.

[11] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. arXiv:2103.00020 [cs]. Feb. 2021. URL: http://arxiv.org/abs/2103.00020.

[12] Robin Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models*. arXiv:2112.10752 [cs]. Apr. 2022. URL: http://arxiv.org/abs/2112.10752.

[13] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. arXiv:1505.04597 [cs]. May 2015. URL: http://arxiv.org/abs/1505.04597.

[14] *RunPod Documentation*. URL: https://docs.runpod.io/.

[15] Ruoxi Shi et al. *Zero123++: a Single Image to Consistent Multi-view Diffusion Base Model*. arXiv:2310.15110 [cs]. Oct. 2023. URL: http://arxiv.org/abs/2310.15110.

[16] Daniel Sonntag et al. *Fine-tuning deep CNN models on specific MS COCO categories*. arXiv:1709.01476 [cs]. Sept. 2017. URL: http://arxiv.org/abs/1709.01476 (visited on 06/09/2024).

[17] Mingxing Tan and Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. arXiv:1905.11946 [cs, stat]. Sept. 2020. URL: http://arxiv.org/abs/1905.11946 (visited on 06/07/2024).

[18] Ashish Vaswani et al. *Attention Is All You Need*. arXiv:1706.03762 [cs]. Aug. 2023. URL: http://arxiv.org/abs/1706.03762.

[19] Xintao Wang et al. *Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data*. arXiv:2107.10833 [cs, eess]. Aug. 2021. URL: http://arxiv.org/abs/2107.10833.

[20] *Weights & Biases Documentation*. URL: https://docs.wandb.ai/.

[21] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. *Adding Conditional Control to Text-to-Image Diffusion Models*. arXiv:2302.05543 [cs]. Nov. 2023. URL: http://arxiv.org/abs/2302.05543.

[22] Tianyi Zhang et al. *A Survey of Diffusion Based Image Generation Models: Issues and Their Solutions*. arXiv:2308.13142 [cs]. Aug. 2023. URL: http://arxiv.org/abs/2308.13142.

[23] Wenliang Zhao et al. *UniPC: A Unified Predictor-Corrector Framework for Fast Sampling of Diffusion Models*. arXiv:2302.04867 [cs]. Oct. 2023. URL: http://arxiv.org/abs/2302.04867.

[24] Zixin Zhu et al. *Designing a Better Asymmetric VQGAN for StableDiffusion*. arXiv:2306.04632 [cs]. June 2023. URL: http://arxiv.org/abs/2306.04632.