

***Dissertation***

On

**Hyper Realistic Rain Video Simulation and Study on  
Deep Learning Based Deraining**

*Thesis Submitted in the partial fulfilment of the requirements for the degree of*

**Master of Engineering**

In

**Computer Science and Engineering**

Submitted By

**Aishik Chanda**

**Registration number: 154166 of 2020-2022**

**Roll Number: 002010502042**

Under Guidance of

**Prof. (Dr.) Ujjwal Maulik**  
**Jadavpur University**

Dept. of Computer Science & Engineering  
Faculty Council of Engineering and Technology  
JADAVPUR UNIVERSITY  
KOLKATA – 700032  
2021 – 2022

**Department of Computer Science & Engineering**  
**Faculty Council of Engineering and Technology**  
**JADAVPUR UNIVERSITY, KOLKATA – 700032**

**Certificate of Recommendation**

This is to certify that Aishik Chanda (Roll number: 002010502042, Registration Number: 154166) has completed his dissertation entitled “Hyper realistic rain video simulation and study on deep learning based deraining”, under the supervision and guidance of Prof. Ujjwal Maulik, Jadavpur University, Kolkata. We are satisfied with his work, which is being presented for the partial fulfillment of the degree of Master of Engineering in Computer Science & Engineering, Jadavpur University, Kolkata – 700032.

---

**Prof. Ujjwal Maulik**  
Teacher in Charge of Thesis  
Professor, Dept. of Computer Science & Engineering  
Jadavpur University, Kolkata – 700032

---

**Prof. Anupam Sinha**  
HOD, Dept. of Computer Science & Engineering  
Jadavpur University, Kolkata – 700032

---

**Prof. Chandan Mazumdar**  
Dean, Faculty Council of Engineering and Technology  
Jadavpur University, Kolkata – 700032

**Faculty Council of Engineering and Technology**  
**JADAVPUR UNIVERSITY, KOLKATA – 700032**

**Certificate of Approval\***

The foregoing thesis entitled “Hyper realistic rain video simulation and study on deep learning based deraining”, is hereby approved as a creditable study of Master of Engineering in Computer Science & Engineering and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion therein but approve this thesis only for the purpose for which it is submitted.

Final Examination for Evaluation of the Thesis

Signature of Examiners

\_\_\_\_\_  
\_\_\_\_\_

\*only in case the thesis is approved

## Declaration of Originality and Compliance of Academic Ethics

I hereby declare that this thesis contains literature survey and original research work by the undersigned candidate, as part of her Master of Engineering in Computer Science & Engineering.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct.

Name	Aishik Chanda
Roll Number	002010502042
Registration Number	154166
Thesis Title	Hyper realistic rain video simulation and study on deep learning based deraining

I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Signature with date:

Signature with date:

# **Acknowledgements**

This dissertation would not have been possible without the support of many people. First and foremost, I would like to thank my advisor, Dr. Ujjwal Maulik, Professor, Department of Computer Science and Engineering, Jadavpur University, Kolkata. He has provided me with the perfect balance of guidance and freedom. He is the first person who introduced me to the world of deep learning and provided me with the guidance that was essential in making this dissertation. I also want to thank him for his perspective and for helping me pursue and define projects with more impact.

I would also like to thank Mr. Srimanta Kundu, Research Scholar, Department of Computer Science and Engineering, Jadavpur University for his immense contribution to my knowledge of deep learning and Image processing. A lot of this dissertation would not have happened without the long, brainstorming sessions and the technical discussions we had during the coffee breaks at the university. He has questioned me at every step of the way, allowing me to make numerous improvements on my project. In many ways, he has shown me the path for how to apply, develop and understand deep learning.

Finally, I would like to thank Jadavpur University for providing me with the opportunity to work in such a productive environment. I would also like to thank all the other professors and research scholars of the department who have extended their helping hands whenever I needed help.

Date:

Place:

---

Aishik Chanda, M.E in CSE,

Exam Roll: 002010502042

# **Content**

<b>Acknowledgements</b>	<b>5</b>
<b>Content</b>	<b>6</b>
<b>List of Figures</b>	<b>8</b>
<b>List of Tables</b>	<b>10</b>
<b>Abstract</b>	<b>11</b>
<b>Introduction</b>	<b>12</b>
Organization of this thesis	14
<b>Chapter 1</b>	<b>15</b>
Introduction	17
Background	17
Attention	17
Transformer	17
Vision Transformer	18
Literature Survey	20
Vit Model	27
Transformer	27
Self-Attention	31
Self-attention technical view	32
Matrix calculation	35
Multiheaded attention	37
Residual Connections	43
Inside the decoder	46
Output	47
Vision Transformer	48
Image splitting	48
Flatten	49
Position Encoding	49
Transformer Encoder	49

Training	50
VitMAE	51
Experimental Setup	53
Results	55
<b>Chapter 2</b>	<b>57</b>
Introduction	59
Dataset Creation	62
Theory	65
Size	65
Shape	65
Forces	67
Appearance	68
Our setup	71
Results	74
<b>Chapter 3</b>	<b>75</b>
Conclusion	76
References	78

# **List of Figures**

Figure 1. The architecture of DerainCycleGAN

Figure 2. Synthetic rain images

Figure 3. The architecture of URAD

Figure 4. The architecture from the paper JORDER

Figure 5. Attentive Generative Adversarial Network for Raindrop Removal from A Single Image (Architecture)

Figure 6. The architecture of our contextual autoencoder. Multiscale loss and perceptual loss are used to help train the autoencoder.

Figure 7. Abstract architecture of a transformer.

Figure 8. Encoder and decoder stacking inside a transformer

Figure 9. Single encoder from the transformer

Figure 10. Decoder from the transformer

Figure 11. Self attention to the feed forward network.

Figure 12. Output of the first encoder to the next encoder.

Figure 13. The Key, Query and Value vectors

Figure 14. Calculating the score in the self-attention

Figure 15. Performing softmax on the scores

Figure 16. The final output of the self-attention

Figure 17. Matrix representation of the Key, Query and Value vectors

Figure 18. The entire self-attention operation condensed in a single formula

Figure 19. The multi headed attention

Figure 20. Multiple output from multiple attention heads.

Figure 21. The operations inside the self-attention to get the final low dimensional attention

Figure 22. All the steps inside the self-attention illustrated

Figure 23. Demonstrating the positional encoding

Figure 24. An example of a 4-dimensional position encoding

Figure 25. The positional encoding visualized

Figure 26. Demonstrating residual connections

Figure 27. Visualizing the vector and layer norms inside the encoder



Figure 28. Visualizing the vector and layer norms inside the decoder

Figure 29. Converting the decoder output to words.

Figure 30. The Vision Transformer architecture

Figure 31. The transformer encoder.

Figure 32. The abstract ViTMAE architecture

Figure 33. Sample of synthetic rain streaks and rain images.

Figure 34. Sample from the “Attentive Generative Adversarial Network for Raindrop Removal from A Single Image” paper and their ground truth.

Figure 35. Change of the drop diameter and shape as it falls.

Figure 36. The water droplet on a flat surface.

Figure 37. Picture of a clear raindrop.

Figure 38. Figure showing how light travels through a drop of water on a flat surface

Figure 39. A different view of our setup in unity. a) top view, b) 3/4th view

Figure 40. A screenshot from unity when simulating the rain

## **List of Tables**

Table 1: The output of our model on our light rain dataset

Table 2: The output of our model on our heavy rain dataset

Table 3. Comparison to existing state of the art

Table 4. Comparison of increase in performance with a VGG loss on our light rain dataset

Table 5. Comparison on our heavy rain dataset

Table 6. Comparison on our light rain dataset

Table 7. Sample from our light rain dataset

Table 8. Sample from our heavy rain dataset

# **Abstract**

*Raindrops adhered to windshields of cars can severely hamper the visibility of the background and degrade any image or video captured by the driving assistance systems to an extent that it no longer is able to detect background objects accurately. The problem is intractable since the regions occluded by the raindrops are not given and since any information about the occluded regions are completely lost. It is difficult to develop any supervised model as no dataset with proper ground truth exists for rainy videos. In this paper we attempt to develop a method to restore the video frame by frame by first creating a dataset of rainy images with their ground truth and then developing a model to restore rainy images.*

# **Introduction**

Raindrops attached to a windshield can affect the visibility of the background and also affect the performance of any object detection or segmentation performed on the image. Even though the affected area may look completely unrecoverable, the raindrops affected regions are formed from rays from the background refracted through the water drop, the information about the background is still present in the area but distorted. In our work, we attempt to create a deep learning model which can recover these raindrop-affected regions from images. We study various methods that have previously attempted to model the raindrop on a surface.

Our main focus will be on the case where the rain drops are hitting the surface of the windshields of cars. Self-driving cars heavily depend on computer vision techniques such as segmentation and object detections, and these techniques are affected by the raindrops. If we can recover the rain affected images we can improve the performance of self-driving cars in bad weather.

Deep learning is a set of machine learning techniques that extracts complex features from the input as opposed to task-specific algorithms. The inspiration for deep learning techniques comes from the mammalian brain. Deep learning techniques have been used in computer vision tasks, image analysis tasks, natural language processing tasks, time-series data analysis, biomedical and computational biology and many other tasks.

Deep learning models have a fundamental architecture which is made of several perceptrons or artificial neurons arranged in layers, and these layers are stacked. These artificial neurons behave as non linear processing units organized in layers. These layers are stacked one after the other and each layer successively extracts more complex features. The parameters in each of the neurons are commonly initialized randomly thus they need to be updated. The technique used to update the parameters is called backpropagation in which, after calculating the loss in forward pass of the model the loss is propagated backward through the model and updates the parameters accordingly. An artificial neural network that has more than 2 hidden layers is called a deep artificial neural network. It is observed that with increasing numbers of these hidden layers the network often fails. Generally, deep learning models are computationally intensive and do not guarantee optimality.

Recently deeplearning technique have been combined with other machine learning technique to build new hybrid model which give better output and better performance as compared to vanilla artificial neural networks. Once such technique is where they added the concept of attention to encoder decoder models for sequential data, this model is called transformer. Transformers [1] have both cross attention and self attention arranged in layers. Models like Bert [6] and GPT3 [5] have shown their dominance in the field of natural language processing. Recently ViT [2] have proven their dominance in image processing tasks such as object detection and caption generation. In this study we discuss a new flavor of ViT, the ViTMAE [3] this has been trained with masked images and has displayed better performance in several computer vision tasks.

Most commonly supervised Deep learning models depend on labeled data or for this type of case good ground truth images. In our case, it is clear that we would require rainy as well as nonrainy images of the same scene, which in practice is almost impossible to capture. We will discuss various attempts of doing the same, but there exists no dataset where the background of the rainy image matches exactly with the background of the ground truth. In this study, we create an artificial dataset in unity by simulating rain on dashboard camera videos of cars, with the help of fluid simulation.

The most common deraining technique deals with the rain streaks. In this paper, we deal with the raindrops sticking to the surface of a transparent glass instead. There exists various deeplearning technique based on GANs and cycleGANS which try to solve this issue, the most prominent one is [14].

In this study, we discuss a new deeplearning model the ViTMAE which is a masked vision Transformer. Transformers have been traditionally used for NLP tasks, but recently they have displayed their dominance in some computer vision tasks. Specifically, the ViTMAE is a vision transformer trained with masked images similar to BERT.

The transformer architecture is also made of encoders and decoders, but unlike traditional sequential mode, the transformer has several encoders and decoders stacked. This is so that we can compute the attention between different segments of a sequence. When the transformer is used with text the input string is first tokenized and then passed through a learnable embedding layer, so that

the words when converted into vectors can still maintain some semantic information in the space they are embedded to. The embedded tokens are then added to a positional encoding to give them some positional information and then passed to the encoders.

In the paper [2] they first convert the images into sequences of small patches, flattens them, produce lower dimensional linear embedding and add a positional encoding before passing it to the encoder stack. The transformer needs to be first pretrained and then fine-tuned to a specific task. In case of text transformer the pre-training was done without supervision, two tasks were used to achieve the same, the first was where random words from the input sentence were masked and the transformer had to predict the masked words and the second was where in a group of sentence the transformer had to predict the next sentence given on sentence.

### Organization of this thesis

This thesis is organized into three major chapters:

In chapter one we discuss the model we use, that is the ViTMAE, and discuss how the masked autoencoder helps to reconstruct the rain-affected images.

In chapter two we discuss how we created the synthetic rainy dataset with the help of a game development tool Unity.

In the last chapter we conclude the dissertation.

## **Chapter 1**

## **Section 1: Introduction**



## Introduction

### Background

Seq2seq models are Deep learning models which take a sequence of items as inputs and the output is also a sequence of Items. Classical sequence to sequence models depends on a vector called the context vector. The encoder processes each item in the sequence it has seen into this vector. The encoder after processing the sequence sends the context vector to the decoder to generate the new item.

The context vector was somewhat a bottleneck for sequence-to-sequence models for processing large sequences because the model had to wait for the previous items to be processed before processing the current items. This did not allow parallelization of the models. A solution to this was proposed in [1], these papers use a technique called Attention, which improved the quality of machine translation systems. Attention allowed the model to focus on any item in the sequence as needed when processing an item.

### Attention

The attention allows the sequence-to-sequence model to focus on the relevant items in the entire sequence when processing a single item in two ways, first, the encoder captures a lot more data, instead of capturing information of only the last hidden state the encoder passes information from all the hidden state to the decoder, the decoder chooses the hidden state the item is most related to.

### Transformer

The transformer model essentially consists of an encoding component and a decoding component. The encoding component is made of a couple of encoders stacked over each other, and similarly, the decoding component is made of a couple of decoders stacked over each other. In the paper they decided to stack 6 encoders and 6 decoders. The encoders share a common architecture each but have different weights. The encoders can be broken down into two sublayers, a self-attention layer which allows the encoder to look at the entire sequence when encoding a specific item and the output is fed to a feed forward neural network. The decoder has a similar architecture but

between the attention layer and the feedforward layer there is another attention layer which allows the decoder to focus on specific items in the input sequence.

### Vision Transformer

The ViT is a model for computer vision tasks based on the transformer architecture which was originally for the text based tasks. The original transformer takes a sequence of words as input, similarly, the image is broken into a series of image patches before feeding into the ViT model. The ViT is trained to directly predict the labels of the images. The ViT has demonstrated a huge performance boost over state-of-the-art CNNs with respect to computational resources when trained on enough data.

It is able to embed information globally across the entire image due to the self-attention player in ViT. This also learns to encode the position of each patch from the training data.

The ViTMAE model was proposed in Masked Autoencoders Are Scalable Vision Learners by Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, Ross Girshick [3]. This paper demonstrates that pretraining a ViT to reconstruct pixel values for masked patches can give better results compared to supervised pre-training, when fine-tuned.

## **Section 2: Literature Survey**

## Literature Survey

It is important to model the rain pattern first before developing a method to remove them. Even though there is no good way of determining which model more accurately represents actual rain, we look into a few of the existing models.

In the DerainCycleGAN [10] paper, the authors decided to work with a simple model of rain.

$$X = R + B.$$

Where  $R$  is the rain streaks and  $B$  is the background image, they add them to get the rainy image.

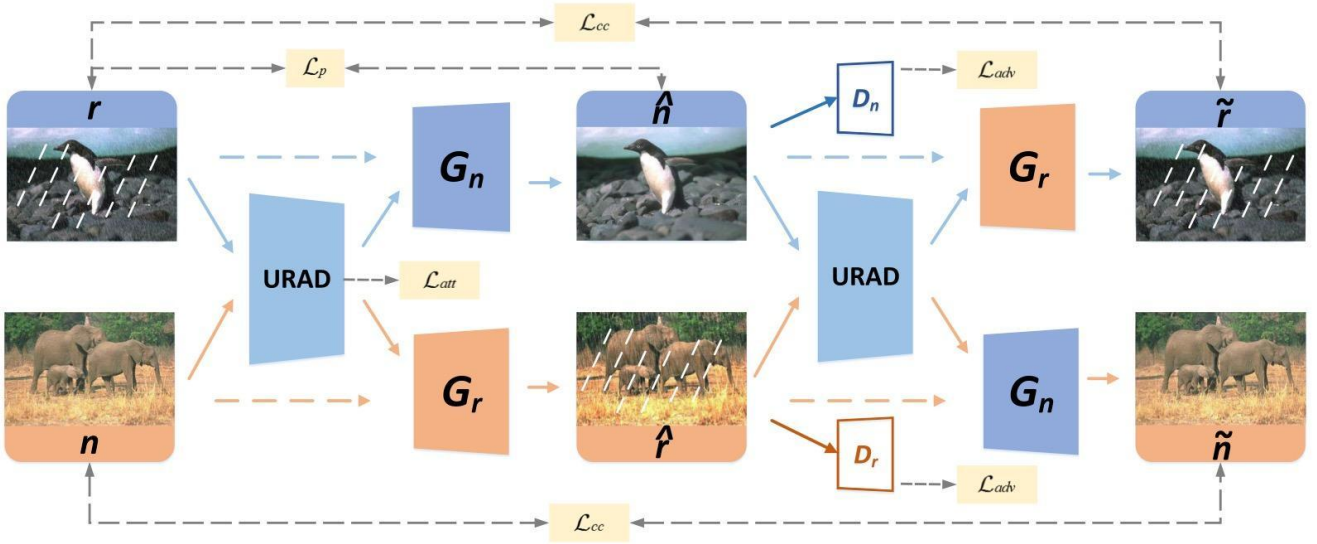


Figure 1. The architecture of DerainCycleGAN that has three parts: (1) unsupervised rain attentive detector (URAD) that pays attention to the rain informations in both rain and rain-free images; (2) two generators  $G_n$  and  $G_r$  that generate rain-free and rainy image, respectively; (3) two discriminators  $D_n$  and  $D_r$  to distinguish the real image from fake image produced by generators. This network has two constrained branches: (1)  $r \rightarrow \hat{n} \rightarrow \tilde{r}$ , where the rainy image is used to generate rain-free image and is then reconstructed by the generators; (2)  $n \rightarrow \hat{r} \rightarrow \tilde{n}$ , where rain-free image is used to generate rainy image and is then reconstructed by the generators.

In the paper JORDER [12] they discuss mainly two issues, first the rain streak accumulation in various regions and second overlapping rain streaks with different directions, and proposed the following model for rain.

$$\mathbf{O} = \alpha \left( \mathbf{B} + \sum_{t=1}^s \check{\mathbf{S}}_t \mathbf{R} \right) + (1 - \alpha) \mathbf{A},$$

$\mathbf{R}$  is a region-dependent variable to indicate the location of individually visible rain streaks.  $\check{\mathbf{S}}_t$  is a layer of rain streaks that have the same direction.  $t$  is the index of the rain-streak layers, and  $s$  is the maximum number of rain-streak layers.  $\mathbf{A}$  is the global atmospheric light, and  $\alpha$  is the atmospheric transmission.

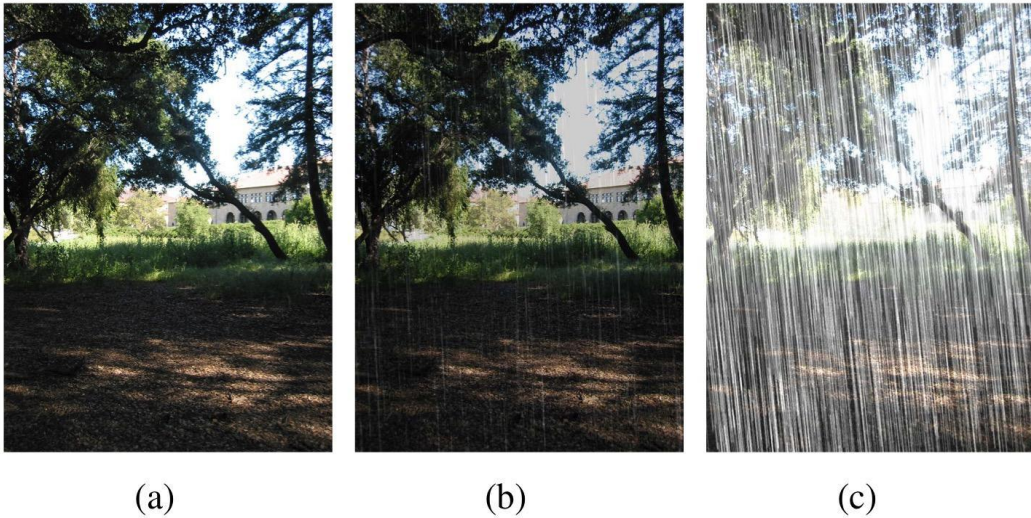


Figure 2. Synthetic rain images, (a) is Groundtruth (b) is light rain and (c) is heavy rain.

The rain model that looks most relevant for our case, i.e recovering the region of the frames from the dash cam recording occluded by the raindrops, is the model defined in the paper Attentive Generative Adversarial Network for Raindrop Removal from A Single Image [8].

where  $I$  is the colored input image and  $M$  is the binary mask. In the mask,  $M(x) = 1$  means the pixel  $x$  is part of a raindrop region, and otherwise means it is part of background regions.  $B$  is the background image and  $R$  is the effect brought by the raindrops, representing the complex mixture of the background information and the light reflected by the environment and passing through the raindrops adhered to a lens or windscreen. Operator  $\odot$  means element-wise multiplication.

The next step is to come up with a good technique to perform the deraining. We look into few supervised as well as few unsupervised techniques using various deep learning models.

Some unsupervised techniques exists like the UDGAN [8], they use a GAN [16] like architecture , they assume that the rain is mainly a high-frequency noise thus they use Gaussian blur with different mean to separate the background from rain streaks reconstruct a image and use two discriminators to train the generator, one for the background and one for the rain streak.

The DerainCycleGAN [10]and the Clearing the skies [9] paper both use a CYCLEGAN [17] to generate a rain drop free image as desired and again reconstruct the rainy image to train the generator. The DerainCycleGAN uses an attentive [13] CYCLEGAN [17], it uses a LSTM [18] based model to extract the attention from the rainy image.

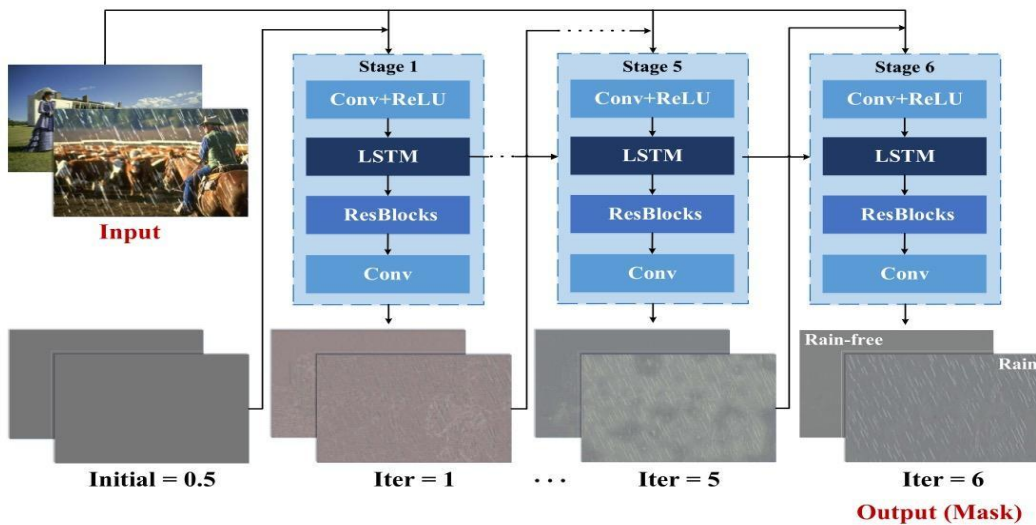


Figure 3. The architecture of URAD [10]

It is difficult to develop a dataset for training deraining models because

It is difficult to capture both the rainy and it's non rainy version at the same time. Yet researchers have used photoshop to develop various rainy image datasets with ground truth.

We looked into few such supervised models, like the JORDER [12], They have defined a new a rain model, they use a contextualised dilated network to extract rain features, then they perform convolution to get the rain mask , using the features extracted and the rain mask they obtain the rain streaks, using the rain streak and the rain mask they obtain the background image from the rainy image.

$$\mathbf{O} = \alpha \left( \mathbf{B} + \sum_{t=1}^s \tilde{\mathbf{S}}_t \mathbf{R} \right) + (1 - \alpha) \mathbf{A},$$

In the Clearing the Skies paper [9], they make an observation that the high frequency elements are mainly constituted of the rain noise, they separated the high frequency element with the help of a low pass filter and passed the high frequency elements through a CNN to separate the rain elements and add the remaining details back to the low frequency elements to get the recovered image.

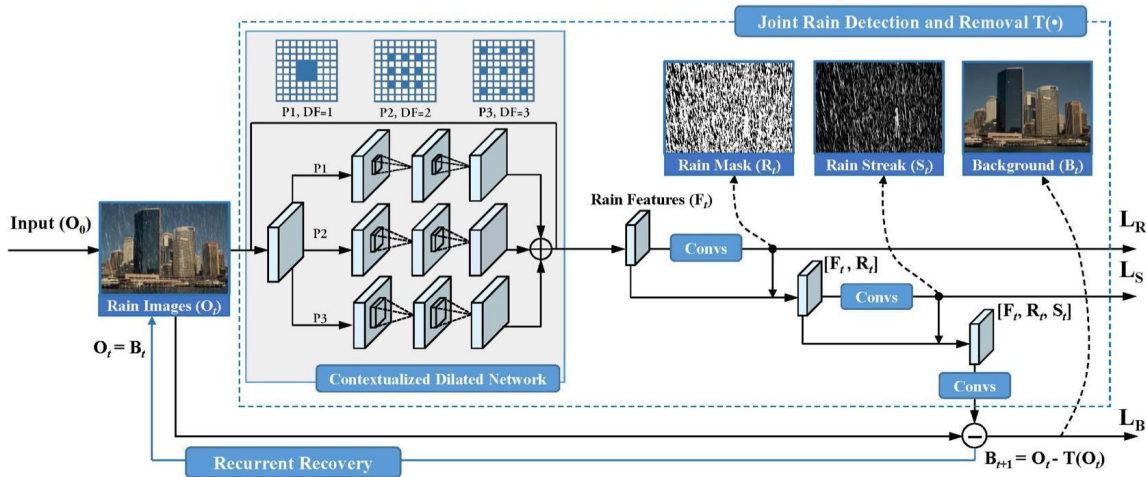


Figure 4. The architecture from the paper JORDER. The architecture of our proposed attentive GAN. The generator consists of an attentive-recurrent network and a contextual autoen-coder with skip connections. The discriminator is formed by a series of convolution layers and guided by the attention map. Best viewed in color.[12]



The paper “Attentive Generative Adversarial Network for Raindrop Removal from A Single Image” [14] tries to recover the image where the water droplets are falling on a transparent surface and thus occluding the background when seen through the transparent surface. This approach is best suited for our case where we are trying to recover the regions occluded by raindrops in a dashcam video.

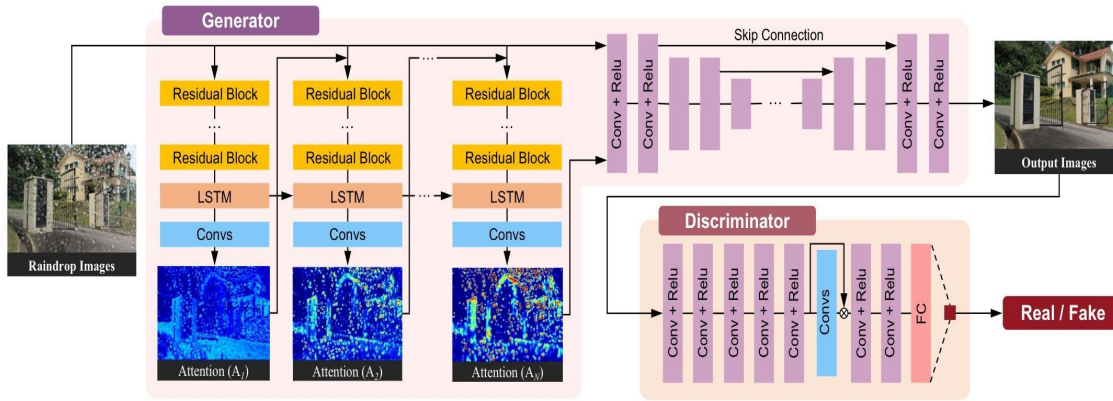


Figure 5. Attentive Generative Adversarial Network for Raindrop Removal from A Single Image (Architecture)[14]

As the name suggests the architecture used in the paper is a GAN [16]. They first extract a mask of the rain occluded region by comparing the degraded image to the ground truth and use this mask  $M$  to train a LSTM [18] to generate the attention [19] in the generator. The attention along with the degraded image is passed to a contextual auto encoder which generates the background image. This is passed to the discriminator which is formed by a series of convolution layers, to train the generator. The extracted background and the ground truth is passed through a pre-trained VGG-16 network, the two sets of features thus obtained are compared and used as an additional loss to train the generator.



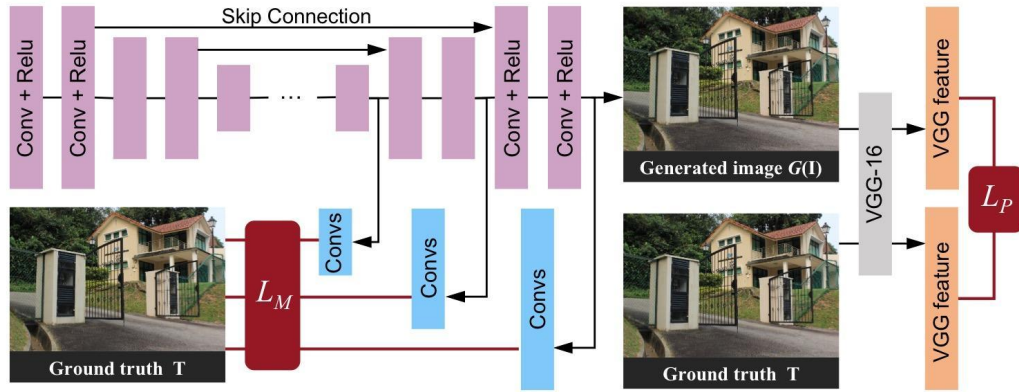


Figure 6. The architecture of our contextual autoencoder. Multiscale loss and perceptual loss are used to help train the autoencoder.[14]

## **Section 3: Vit Model**

## Vit Model

### Transformer

The Transformer is made of an encoding component and a decoding component with connections between them.

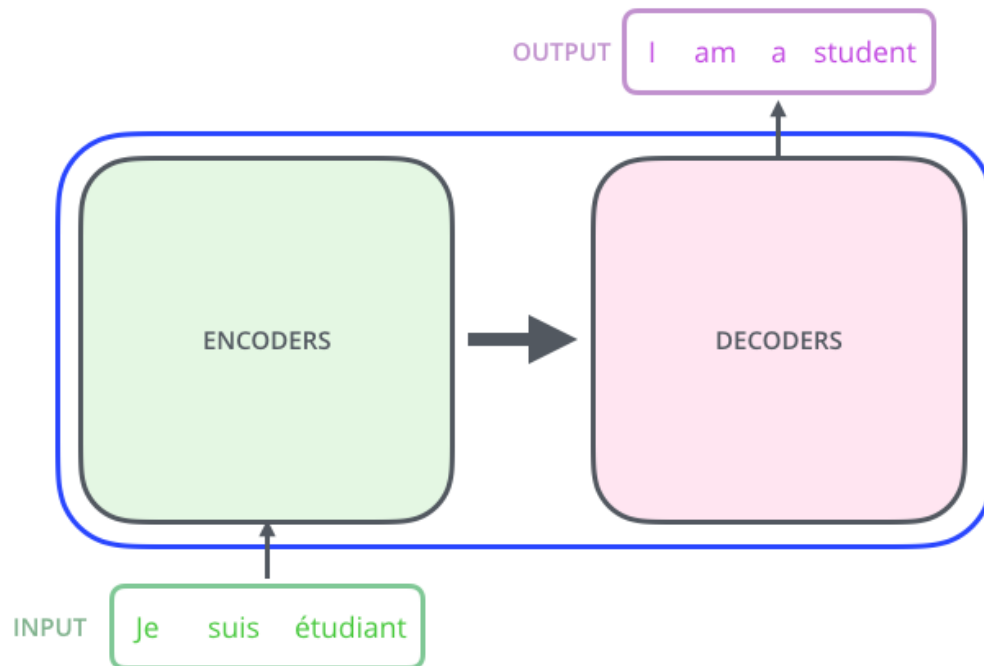


Figure 7. Abstract architecture of a transformer. [4]

The encoding component contains encoders stacked over each other, similarly, the decoding component contains decoders stacked over each other. In the paper, they decided to stack 6 encoders and 6 decoders.

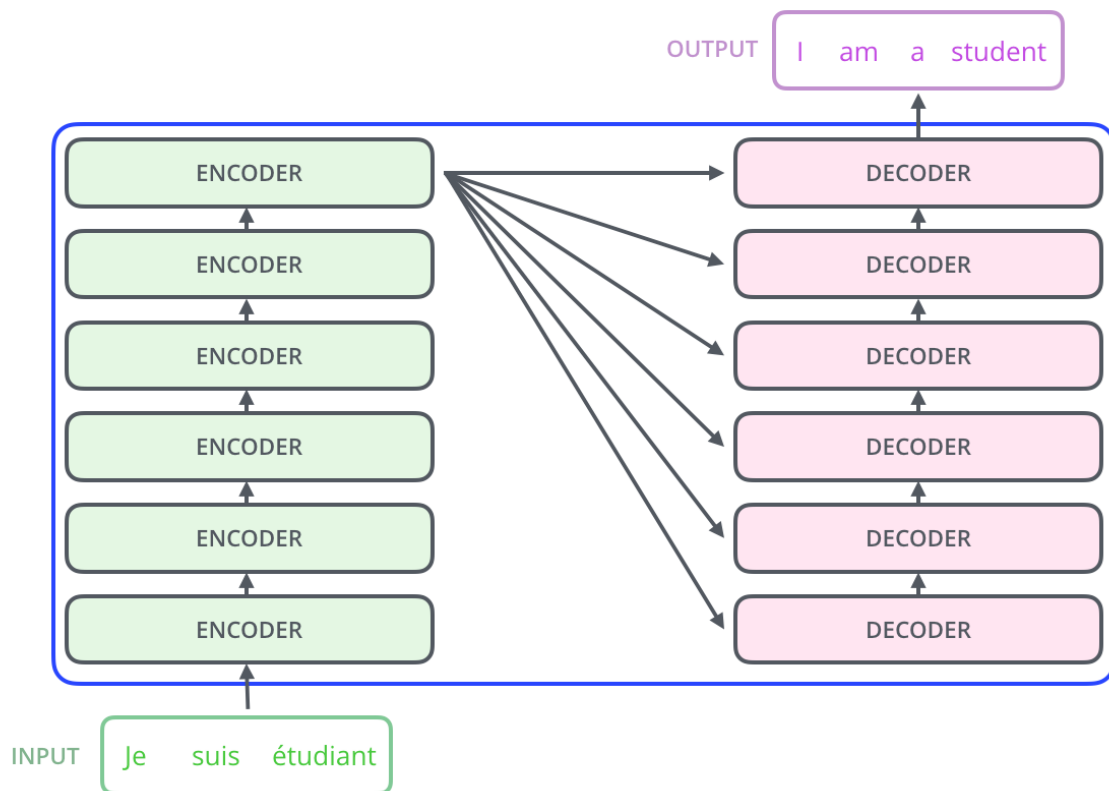


Figure 8. Encoder and decoder stacking inside a transformer [4]

The decoders have the same component and architecture but they do not share weights. The encoder contains two sub-layers, a self-attention and a feed forward neural network.

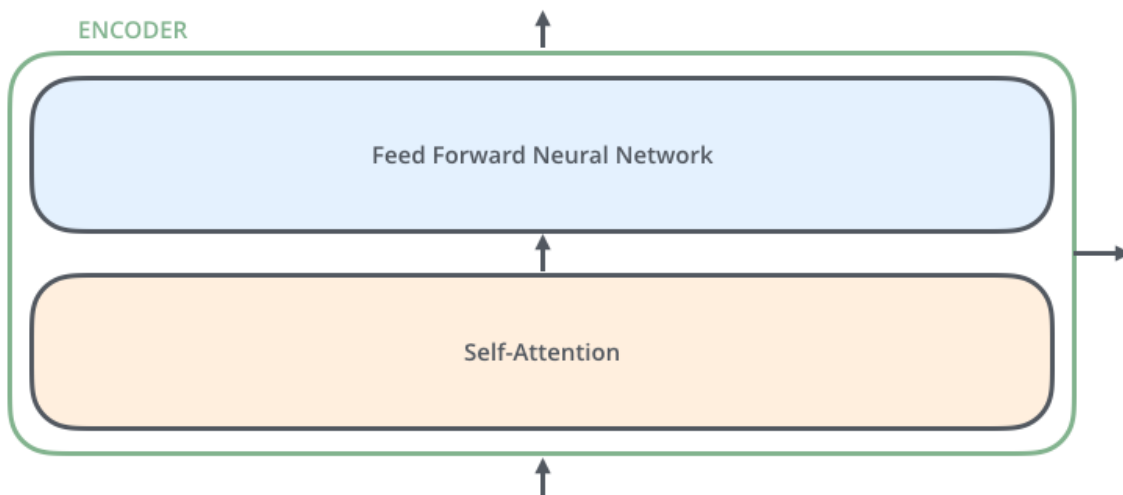


Figure 9. Single encoder from the transformer [4]

The embedded inputs first travel through the self-attention layer that allows the model to focus on all other relevant items when processing a specific item in the sequence. The output of self-attention is fed to a feed-forward neural network.

The decoder has a similar structure, but has an extra attention layer, allowing the decoder to pay attention to relevant parts of the input sentence.

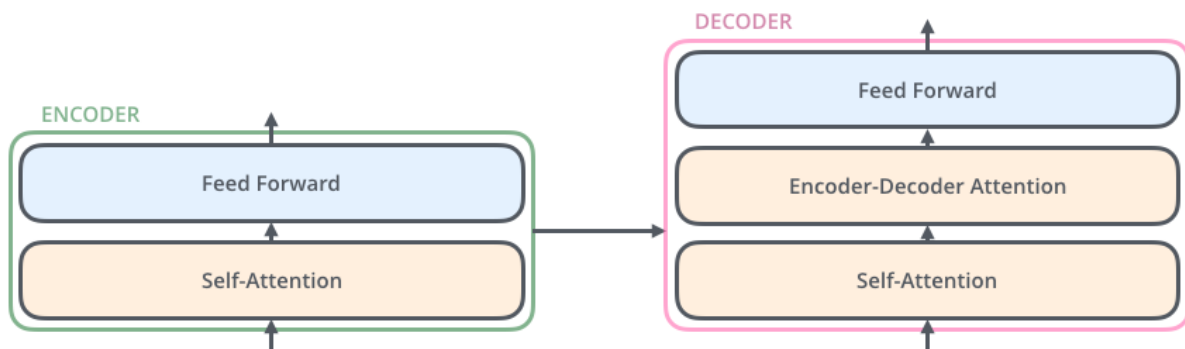


Figure 10. Decoder from the transformer [4]

The transformer needs the input to be a sequence of embedded vectors, so in our case, the first step is to convert the images into a sequence of patches and embed them.

The embedding is done only in the first encoder, the abstraction common in all the encoders is that they receive a list of embedded vectors. In the first encoder, it would be the embedded input tokens, in the rest, the input would be the output of the previous encoder. The size of this list is a hyperparameter we can set according to our requirements.

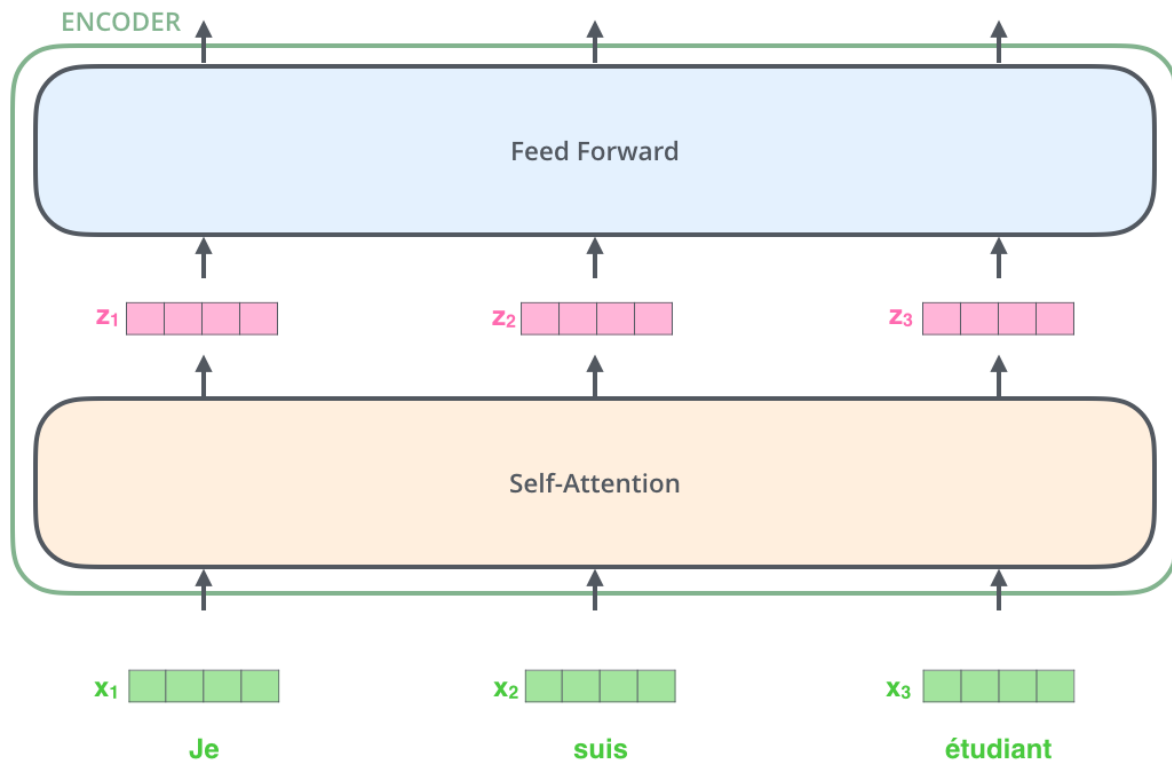


Figure 11. Self attention to the feed forward network. [4]

Once the word is embedded into an input sequence, each of them is passed through the two layers in the encoder. This is a key property of the Transformer, that is each embedded word flow through its own path within the encoder. There are dependencies between these paths in the self-attention layer. However these dependencies does not exist in the feed-forward layer, and because of this different words embedding can flow through their own path in parallel while flowing through the feed-forward layer.

The encoder takes a list of vectors as input, and processes this input by first passing them through a self-attention layer and then through a feed-forward neural network, then they pass the output to the next encoder.

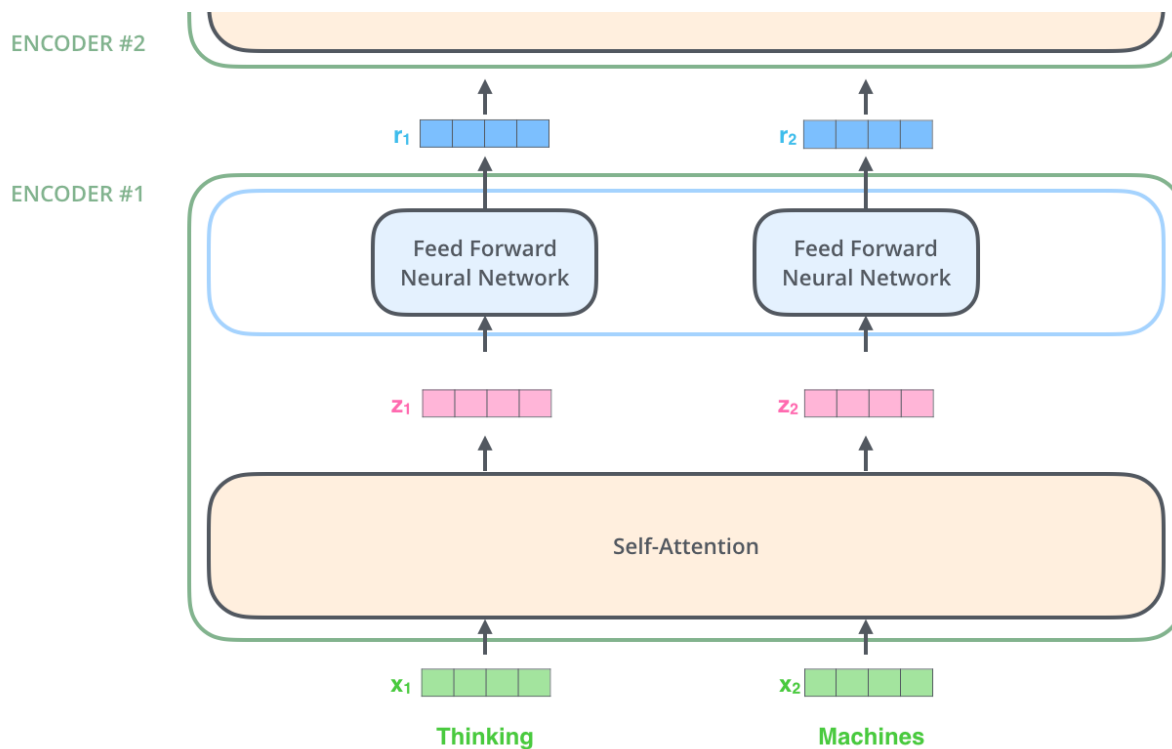


Figure 12. Output of the first encoder to the next encoder. [4]

## Self-Attention

"The cat didn't cross the road because it was blind"

In the above sentence, to understand the meaning it is important to understand what "it" refers to, and does it refer to the cat or the road. This is easy for a human to understand, but not for a deep learning model. When processing the word "it", self-attention allows the model to associate "it" with "cat".

As the model is processing each item in an input sequence, the self-attention allows the model to look at all other items in the sequence to find clues that might help embed more information into the encoder's output encoding for that specific item.

This is similar to what RNN tries to achieve, by maintaining the hidden state of previous words it allows the RNN to embed the information of the previous words into the representation of the word it is currently processing. The Transformer achieves this with

self-attention, it allows the transformer to bake information of other relevant words with respect to the specific word it is processing into the embedding of the specific word.

### Self-attention technical view

To calculate the self-attention we first create three vectors for each of the inputs in the sequence to the encoder. The three vectors are the Query vector, Key vector, and Value vector. These vectors are created by multiplying the input embedding by three learnable matrices.

The dimension of the new vectors is generally smaller than the input embedding, for the original Transformer [1] the input embedding was 512 while the dimension for the three vectors was 64, this is done so as to keep the computation of the multiheaded attention as constant as possible.

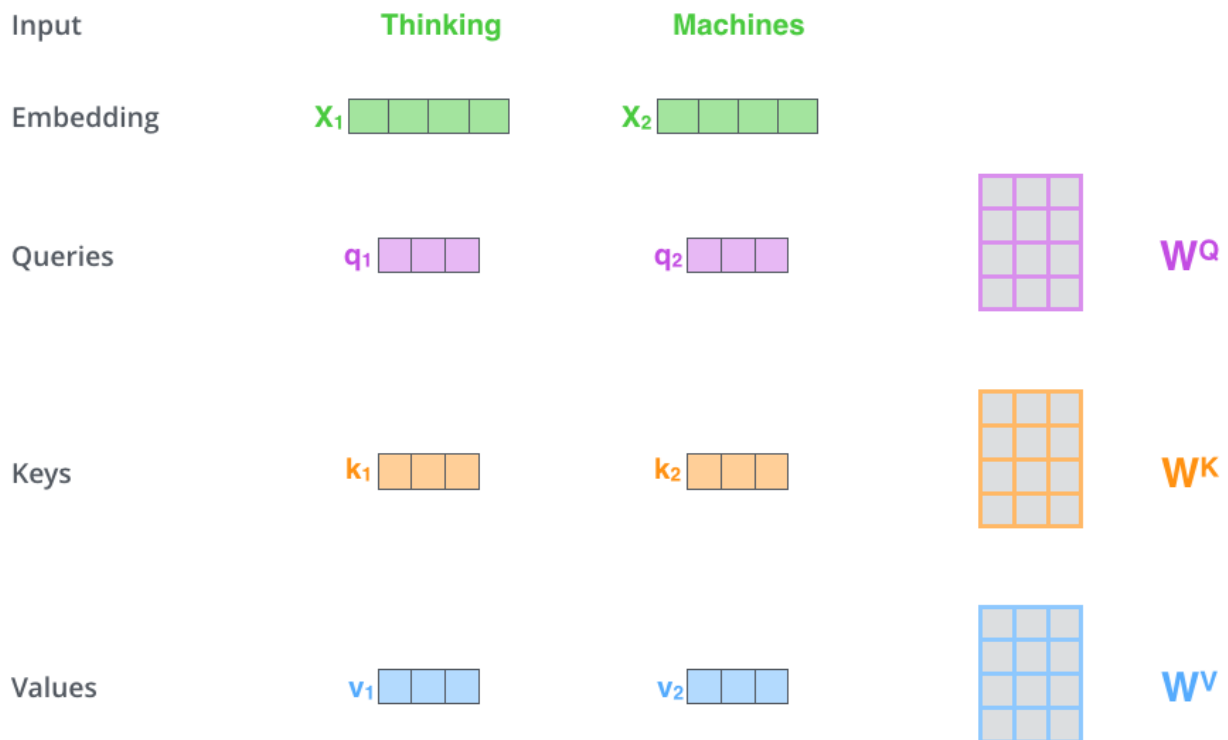


Figure 13. The Key, Query and Value vectors [4]

Next, we calculate a score, the score determines how much attention is to be given to the other items in the sequence. Say we are calculating the self-attention for the first



word, we need to calculate a score for all other words in the sequence with respect to the first word.

The score for each word is calculated by taking the dot product of the query vector of the specific word and the key vector of all the words in the input sequence. If we want to process the self-attention for the first word that is in position one, the first score will be calculating the dot product between  $q_1$  and  $k_2$ , and the second score would be calculated by the dot product of  $q_1$  and  $k_2$ .

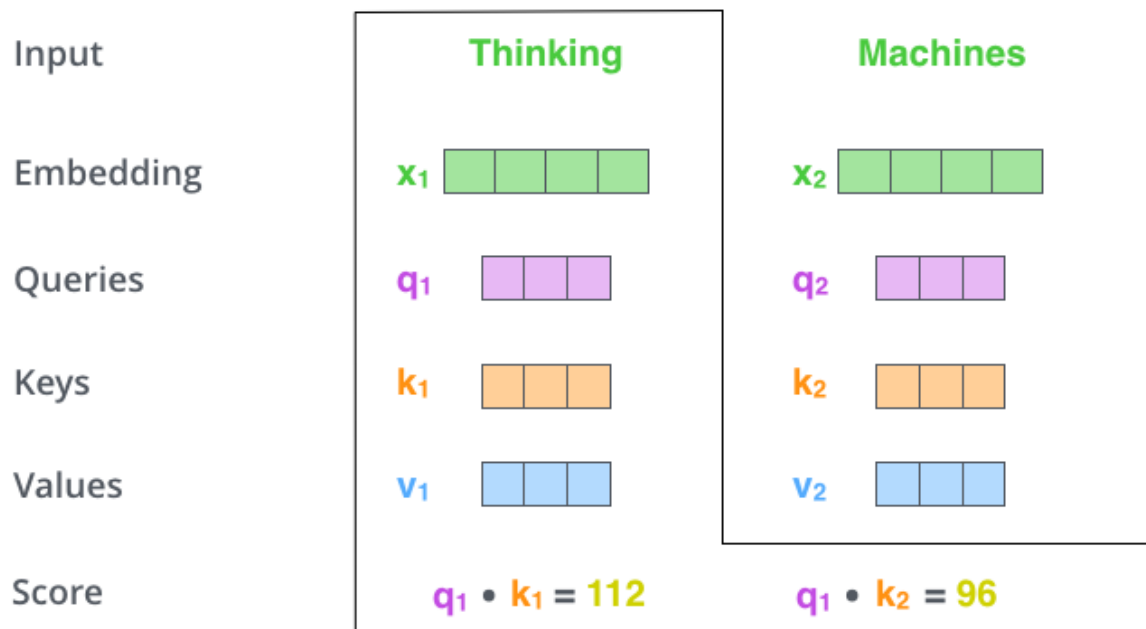


Figure 14. Calculating the score in the self-attention [4]

After calculating the scores we divide the resulting values by the square root of the dimension of the key vector, in the paper [1] it is 8, this results in having more stable gradients. The output is then passed through a softmax layer, softmax normalizes the scores and makes sure the values are positive, and adds up to 1.

The softmax score tells the model how much influence each word in the sequence has on this particular position. Clearly, the word itself will have the highest influence on its position, but often it will attend to other words based on the score as found relevant.

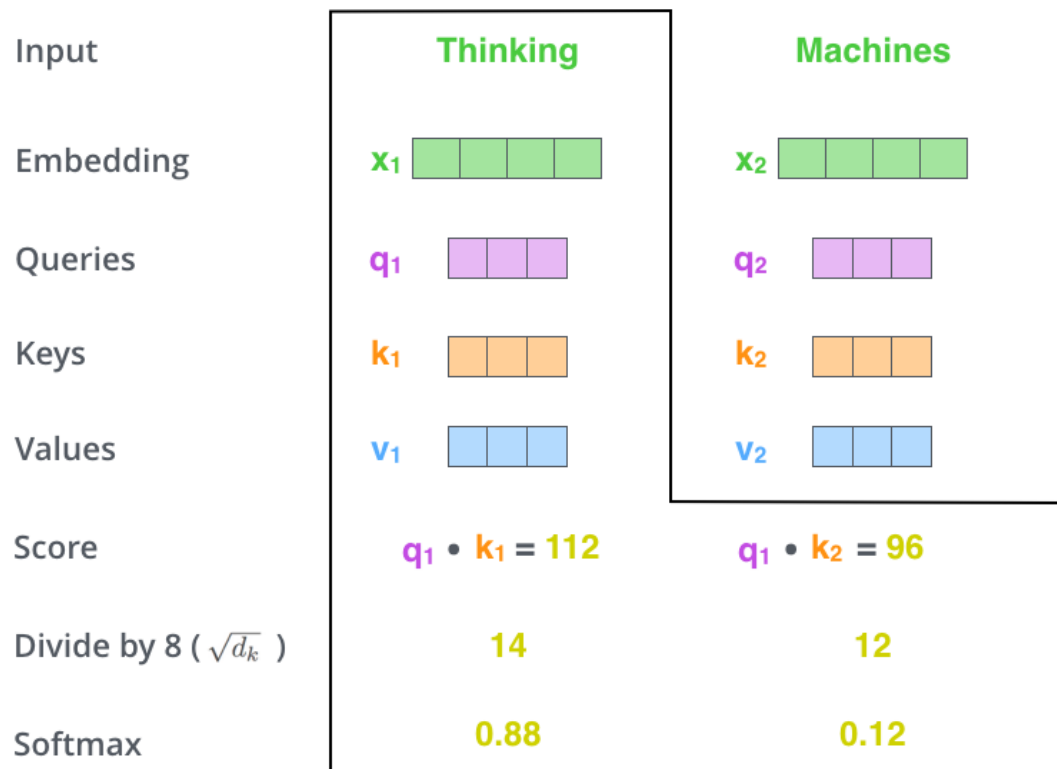


Figure 15. Performing softmax on the scores [4]

The output of the softmax is then multiplied with the value vector,, this is done to keep the value of the words relevant to the current word being processed and drown-out irrelevant words. The weighted value vector thus obtained is added up, this produces the output of the self-attention layer at this position. This is the complete calculation for self-attention. The resulting vector is forwarded to the feed-forward neural network.

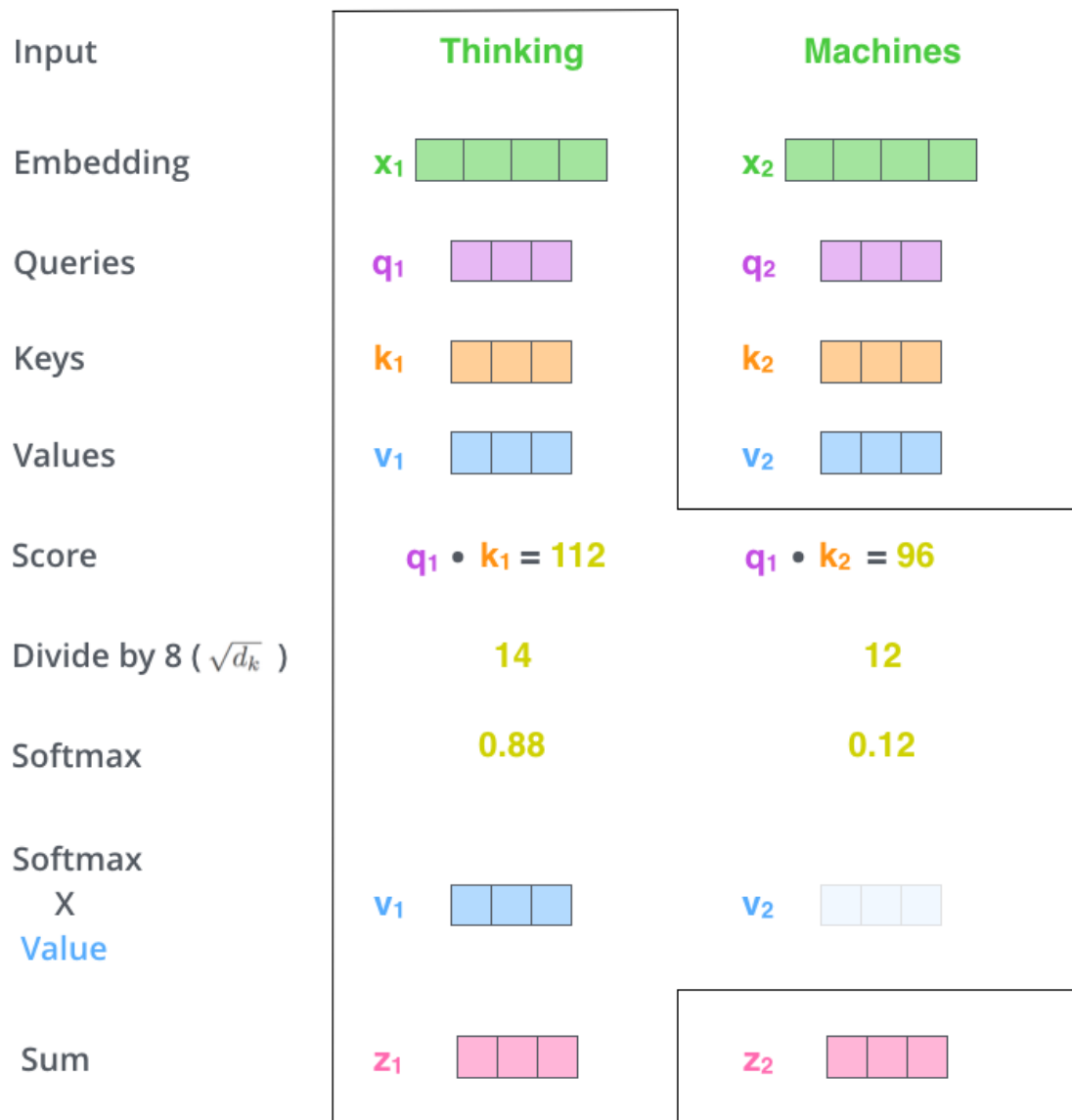


Figure 16. The final output of the self-attention [4]

### Matrix calculation

In the original paper [1] for faster calculation the above calculations are done in matrix form. First, we calculate the Query, Key, and Value matrices. We stack the embedded input into a matrix X and multiply it by learnable weighted matrices.

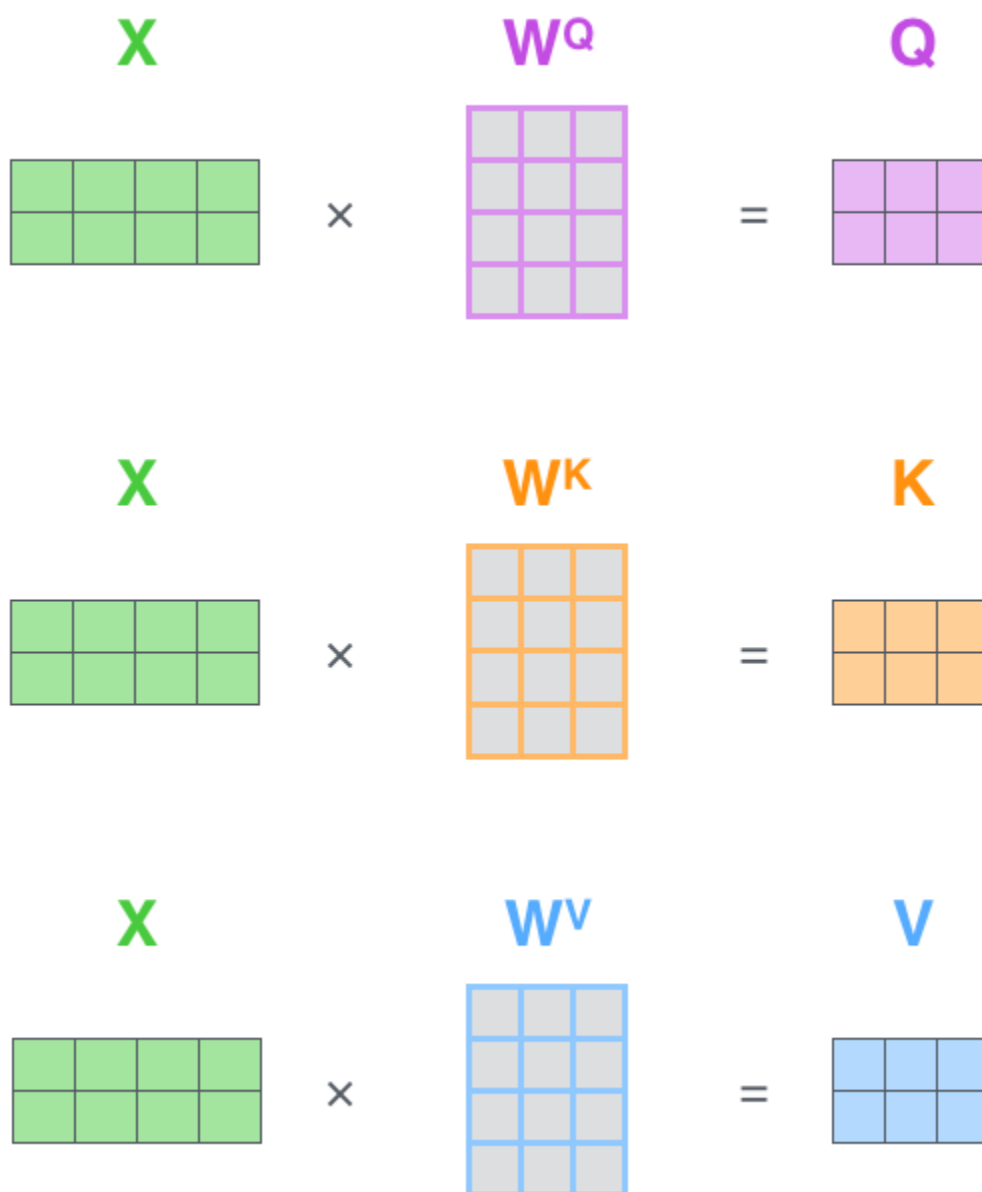


Figure 17. Matrix representation of the Key, Query and Value vectors [4]

The scoring of the embedded words, then the normalization and then finally multiplying with the scored value matrix and adding the obtained weighted value vectors can be condensed into a single formula to get the output of the self-attention layer.

$$\begin{aligned}
 & \text{softmax} \left( \frac{\begin{matrix} \text{Q} \\ \begin{matrix} \square & \square & \square \\ \square & \square & \square \end{matrix} \end{matrix} \times \begin{matrix} \text{K}^T \\ \begin{matrix} \square & \square \\ \square & \square \\ \square & \square \end{matrix} \end{matrix}}{\sqrt{d_k}} \right) \begin{matrix} \text{V} \\ \begin{matrix} \square & \square & \square \\ \square & \square & \square \end{matrix} \end{matrix} \\
 & = \begin{matrix} \text{Z} \\ \begin{matrix} \square & \square & \square \\ \square & \square & \square \end{matrix} \end{matrix}
 \end{aligned}$$

Figure 18. The entire self-attention operation is condensed in a single formula [4]

### Multiheaded attention

The self-attention layer is further refined by the multi-headed attention. The multiheaded attention works in two ways, It allows the model to focus on a different part of the input encoded sequence. Even though the output of the self-attention already does this, the output of the self-attention may be dominated by the item being processed itself. It allows the output of the attention layer to be represented in multiple subspaces. The multiheaded attention has multiple sets of Query, Key, and Value weights. In the original [1] they used 8 attention heads, so there are 8 sets of Query, Key, and Value matrix for each encoder and decoder. Each of these matrices is randomly initialized, after training, each of these project the input of the encoder or decoder to a different representation subspace.

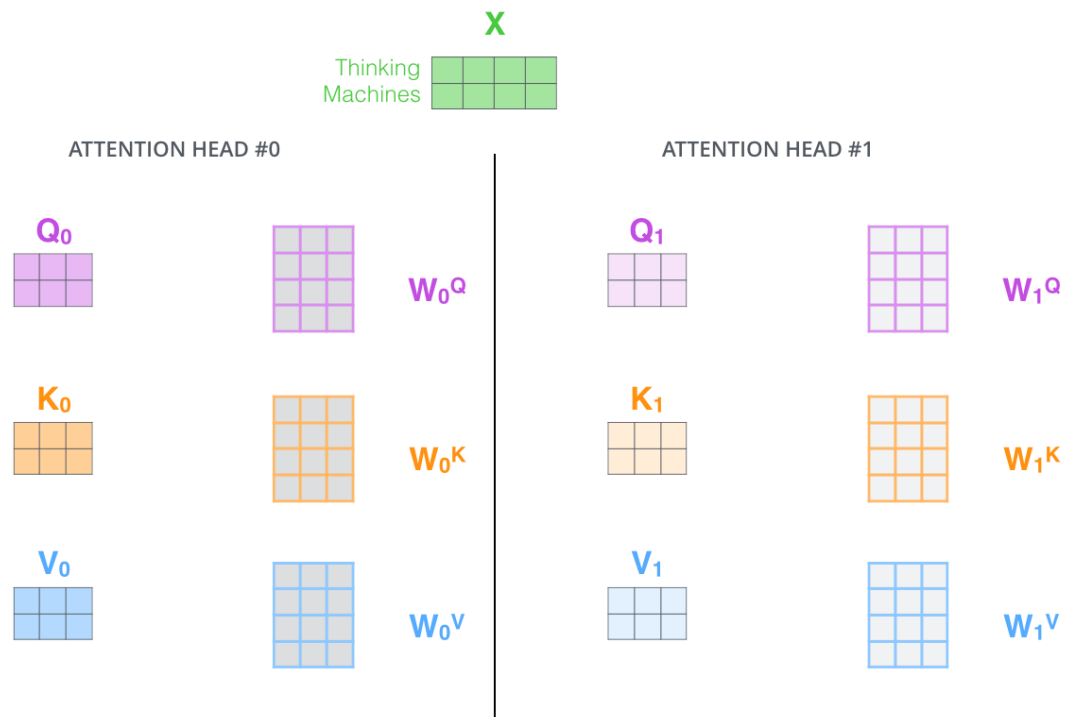


Figure 19. The multi headed attention [4]

The output of multi-head attention generates different attentions for all the different sets of Query, key, and value matrices, this needs to be condensed into a single vector before passing to the feed-forward layer. This is achieved by first contacting all the different attentions and then multiplying it by a learnable weight matrix.



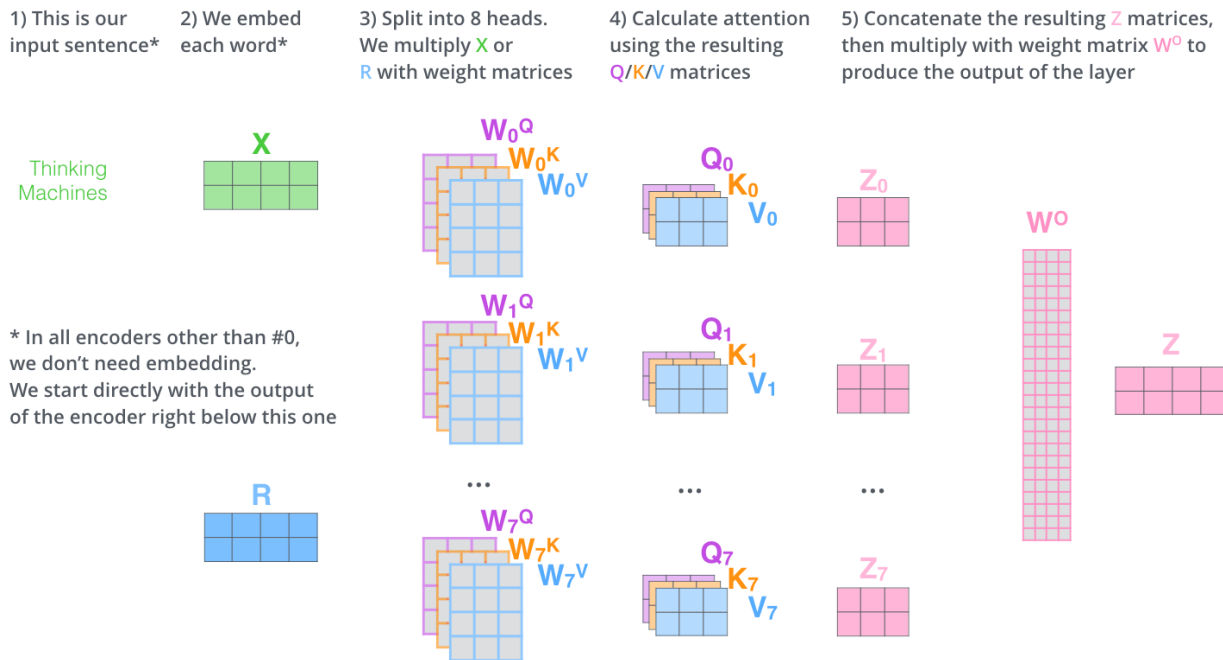


Figure 22. All the steps inside the self-attention illustrated [4]

An important part of the input embedding still is to embed the position of each item in the sequence with respect to each other into the sequence itself. This is done by adding a special vector to each input embedding. This adds a specific pattern to the input embedding so that the model can learn to understand the position of each item in the sequence relative to each other. The intuition here is to add meaningful distance to the embedding before they are projected into query, key, and value vectors and before the dot-product attention is calculated so that the mutual distance between the embedding can help the model to understand the position of the embedding with respect to each other.



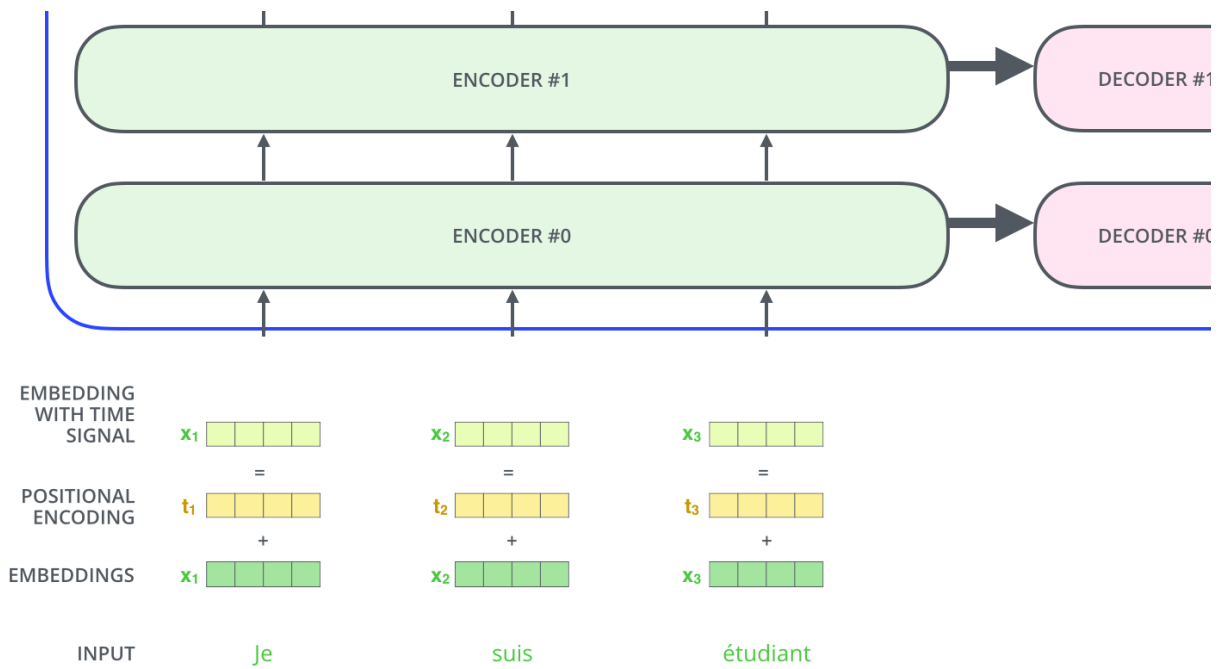


Figure 23. Demonstrating the positional encoding [4]

For example, for a 4-dimensional embedding, the positional encoding would look like

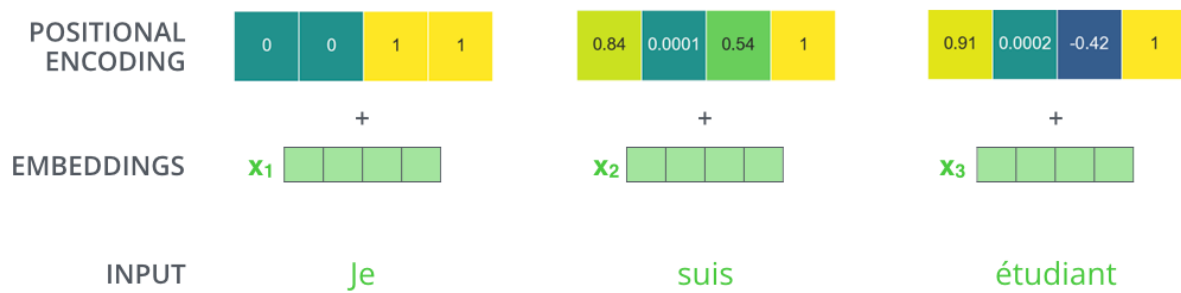


Figure 24. An example of a 4-dimensional position encoding [4]

In the figure, each row corresponds to a positional encoding for a vector, the first row denotes the positional encoding that is added to the first embedded word. The length of each row is 512 values corresponding to the 512 dimensions in the word embedding each with a value between 1 and -1.

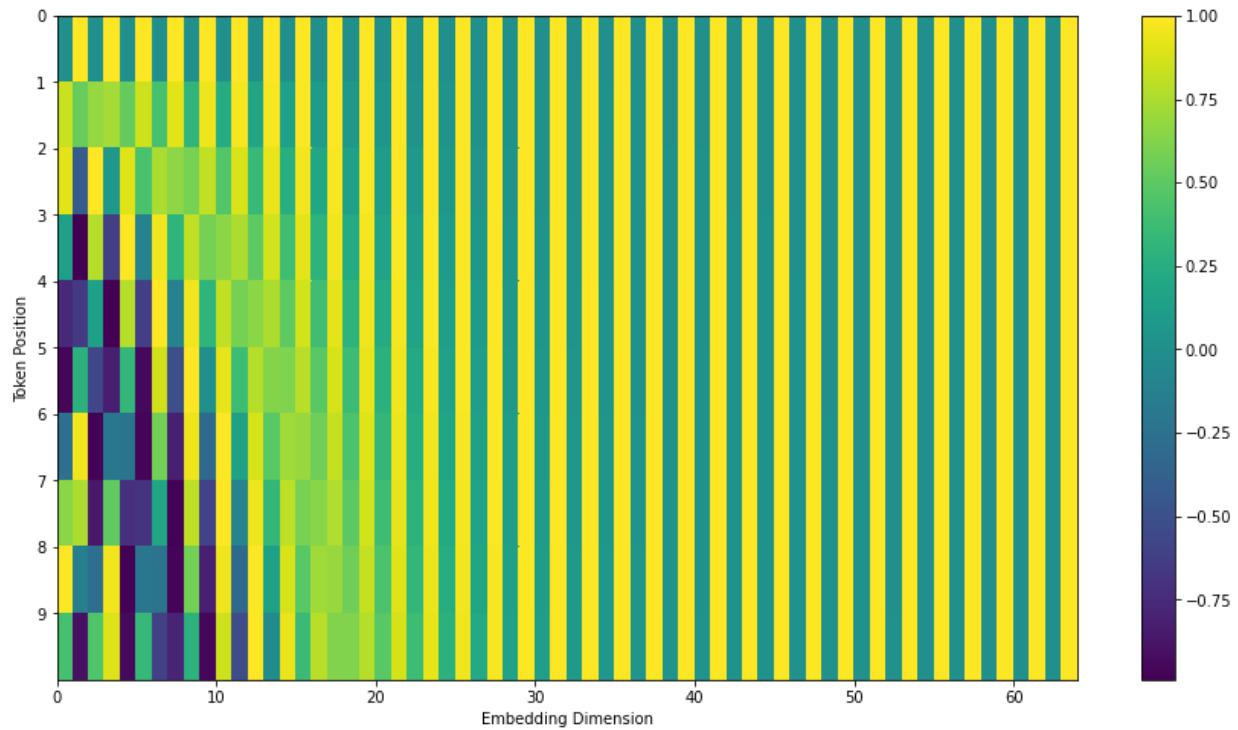


Figure 25. The positional encoding visualized [4]

## Residual Connections

The sublayers in each of the encoders and decoders are connected with residual connections, along with a layer-normalization.

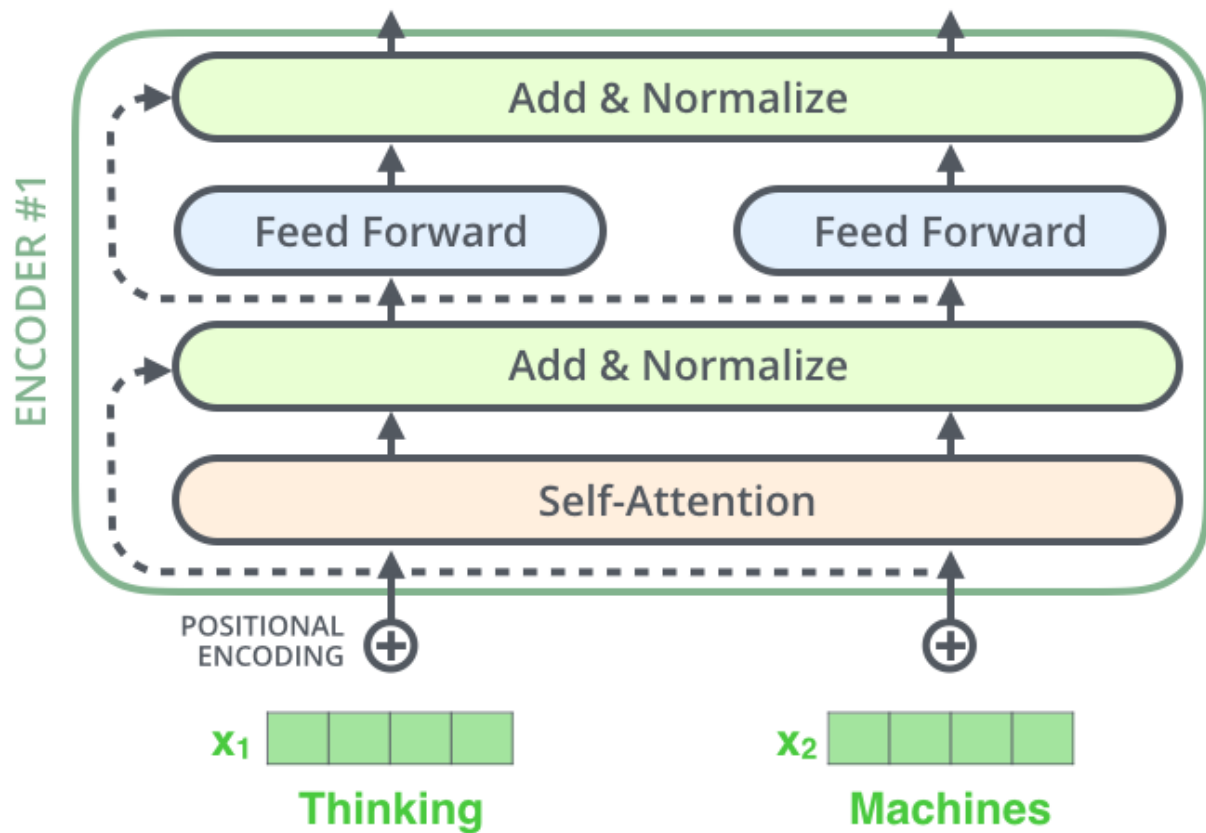


Figure 26. Demonstrating residual connections [4]

Visualizing the vectors and layer norm operation from the self-attention inside the encoder.

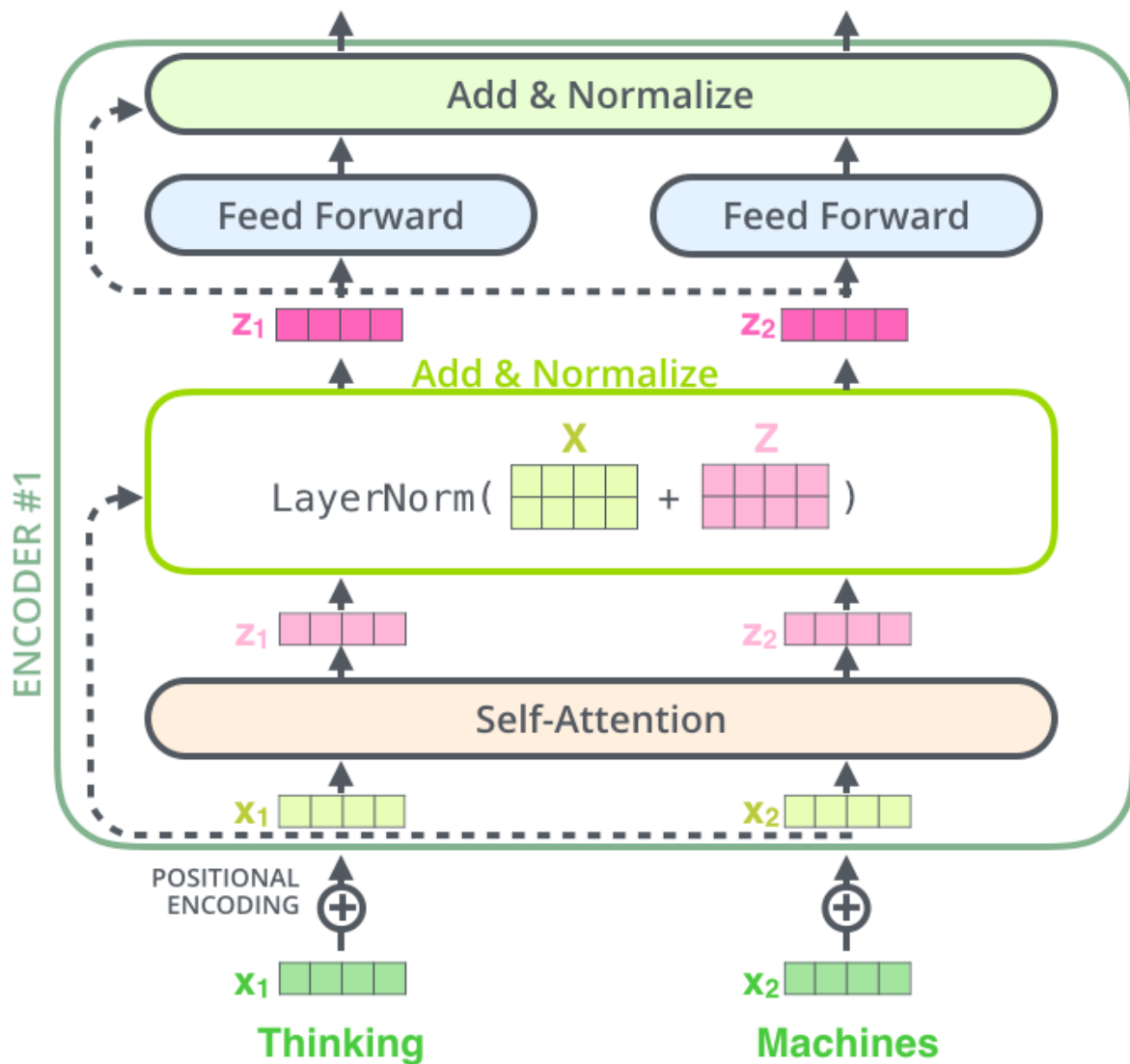


Figure 27. Visualizing the vector and layer norms inside the encoder [4]

Visualizing the vectors and layer norm operation from the self-attention inside the decoder.

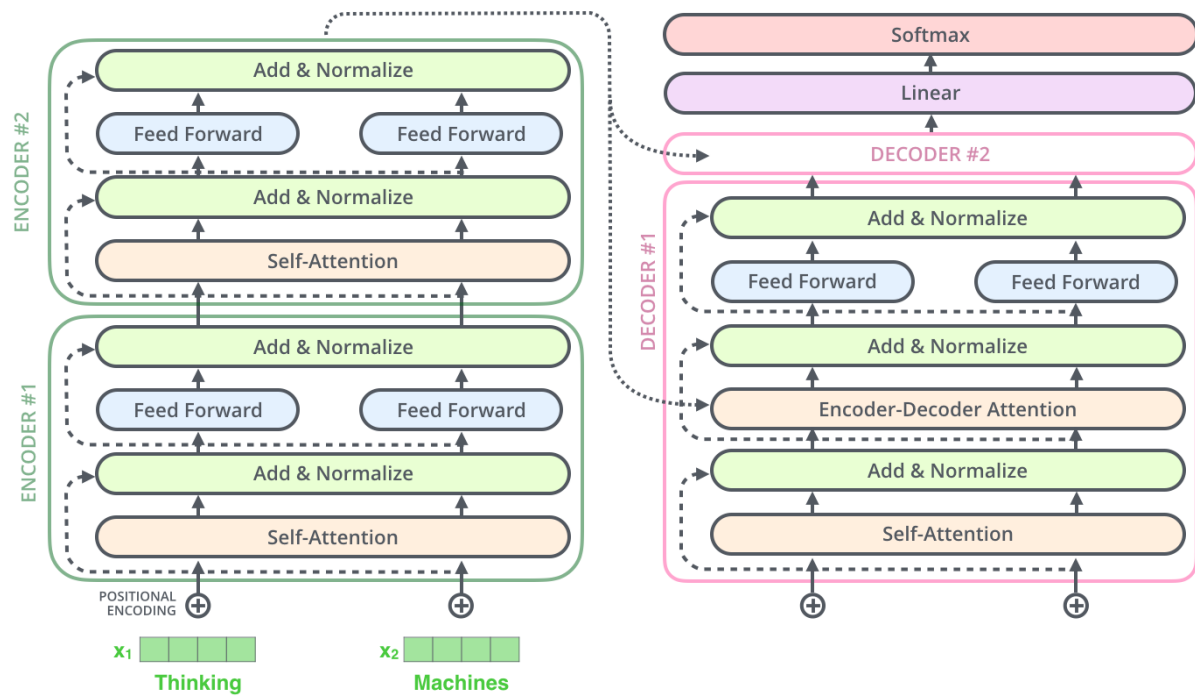


Figure 28. Visualizing the vector and layer norms inside the decoder [4]

## Inside the decoder

The encoder processes the input sequence. The output of the encoder is then transformed into a set of attention vectors  $K$  and  $V$ . The decoder uses these vectors to focus on relevant words from the input in its encoder-decoder attention layer.

The output of each step in the decoder is fed back into the decoder, and the decoder generates an output. This is repeated till a special token is generated by the decoder. Similar to the encoder, the output of the decoder is embedded and a positional encoding is added to the output before passing it back to the decoder.

The self-attention in the decoder is different from the self-attention in the encoder. Unlike in the encoder, in the decoder the self-attention is only allowed to look at the previous elements of the item it is currently processing, this is done by masking the future positions, and setting the score to negative infinity before passing it to the softmax layer in the self-attention, this is called the masked attention.

The Encoder-decoder attention works similarly to the multiheaded self-attention. The only difference is that it creates the Query matrix from the layer below it, and takes the Keys and Values matrix from the output of the encoder.

## Output

The decoder stack outputs a vector, which needs to be converted into words. The output is passed through a fully connected neural network which projects the vector into a much larger vector called the logits vector.

For example, the model learns 10000 unique English words which are called the output vocabulary, that it learns to form the training dataset, this makes the logits vector 10000 cells wide, where there will be a cell corresponding to each unique word. The softmax converts the scores into a probability distribution. The cells with the highest probability are taken, and the word corresponding to that cell is produced.

Which word in our vocabulary  
is associated with this index?

am

Get the index of the cell  
with the highest value  
(argmax)

5

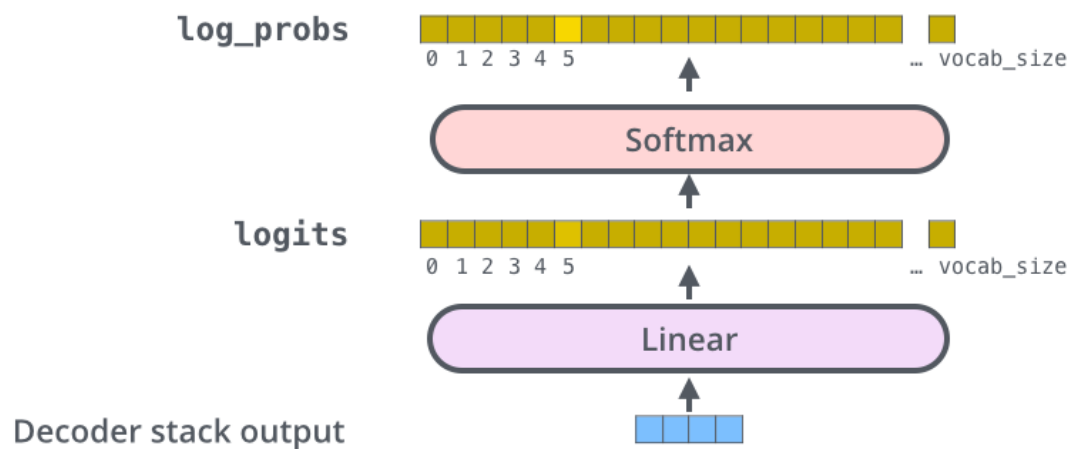


Figure 29. Converting the decoder output to words. [4]

## Vision Transformer

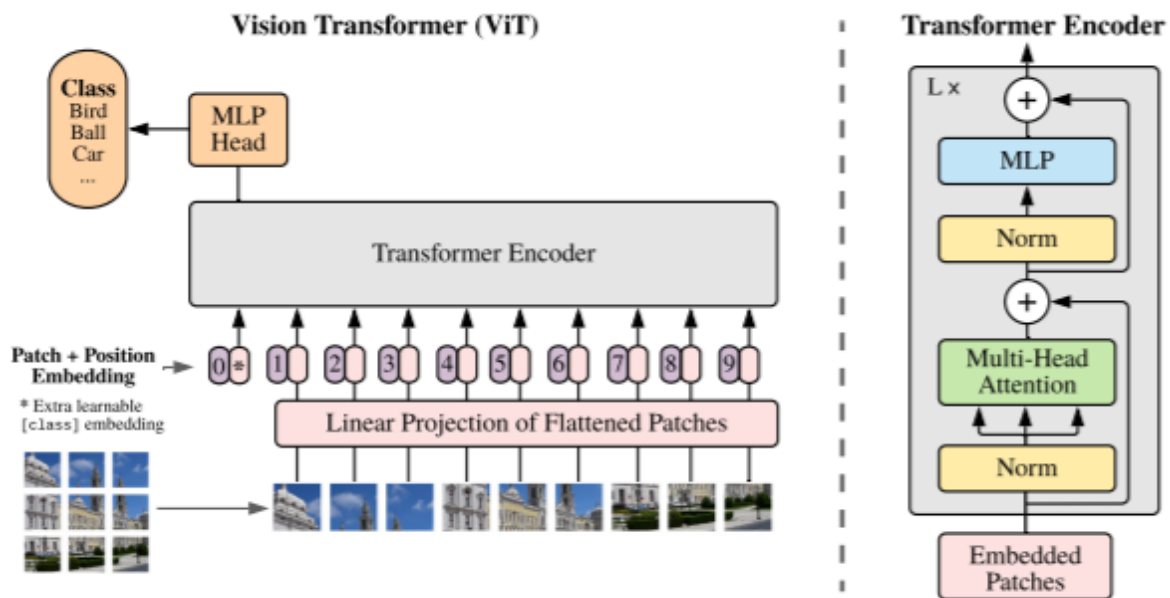


Figure 30. The Vision Transformer architecture.[2]

The transformer has demonstrated great success in NLP tasks, they have now been applied to vision-based tasks. While CNN uses arrays of pixels, Visual Transformer divides the images into visual tokens.

### Image splitting

The images are first split into smaller fixed-sized tokens. The 2D images of size say  $H \times W$  is split into  $N$  patches where  $N = H \times W / P^2$ . The self-attention has quadratic time



complexity. It would be a very costly operation if the self-attention had to pay attention to each pixel. This is why the images are split into smaller patches.

### Flatten

The 2D patches are then flattened into 1D patch embedding and then linearly embedded. Each of the patches is flattened into 1d patch embedding by first concatenating all the channels into a patch and then linearly projecting it to the desired input dimension.

### Position Encoding

Similar to the original transformer, positional encoding is added to the patch encoding to add positional information. The transformer is agnostic to the structure of the input, adding the learnable position embedding to each patch will allow the model to understand the structure of the image.

An extra learnable classification token is added to the sequence to patches at the starting of the sequence.

### Transformer Encoder

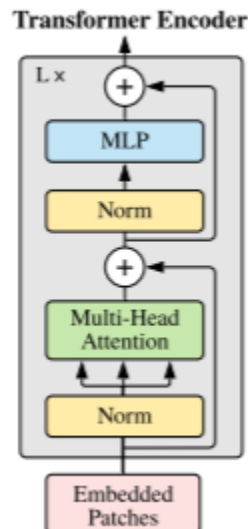


Figure 31. The transformer encoder.[2]

In the paper [2] the transformer encoder consists of:

- A Multi-head self-attention layer to project the multiple self-attention outputs to the desired dimension. The multi-headed self-attention allows the model to learn both local and global dependencies in the image.
- A multi-layer perceptron contains a two-layer GELU (Gaussian Error Linear Unit)
- A Layer Norm is applied before each block to avoid introducing new dependencies between the training images. This helps in improving the training time and generalization performance.
- Residual connections are added after each block to allow the flow of gradients through the network without passing the non-linear activations.

The higher layers of the Vit learn the global features, the lower layers learn both local and global features, this allows the Vit to learn generic patterns.

### Training

The Vit are generally pre-trained on large datasets and then finetuned on a smaller dataset similar to a classical transformer.

The transformer has no prior knowledge of the image and its structures and hence takes a long time to train and requires a large dataset for the training.

## VitMAE

In the paper [3], they have demonstrated that masked autoencoders are scalable self-supervised learning models for computer vision. They mask random patches and trained the model to reconstruct the missing patches. They based their architecture on two core designs, they developed an asymmetric encoder-decoder architecture, with the encoder set to only operate on the visible subset of patches but not the masked tokens, and the lightweight decoder reconstructs the original image from the masked token and latent representation. They observed that masking a high proportion of input images as high as 75% yields a nontrivial and meaningful self-supervised task. Coupling these two core design concepts allowed them to train large models efficiently and effectively. They accelerated training by almost three times more and improved accuracy. This scalable approach allows learning high-capacity models that generalize well like the vanilla ViT-Huge model and achieves the best accuracy of 87.8% among methods that use only ImageNet-1k data. The transfer performance in downstream tasks outperforms supervised pre-training and shows promising scaling behaviors.

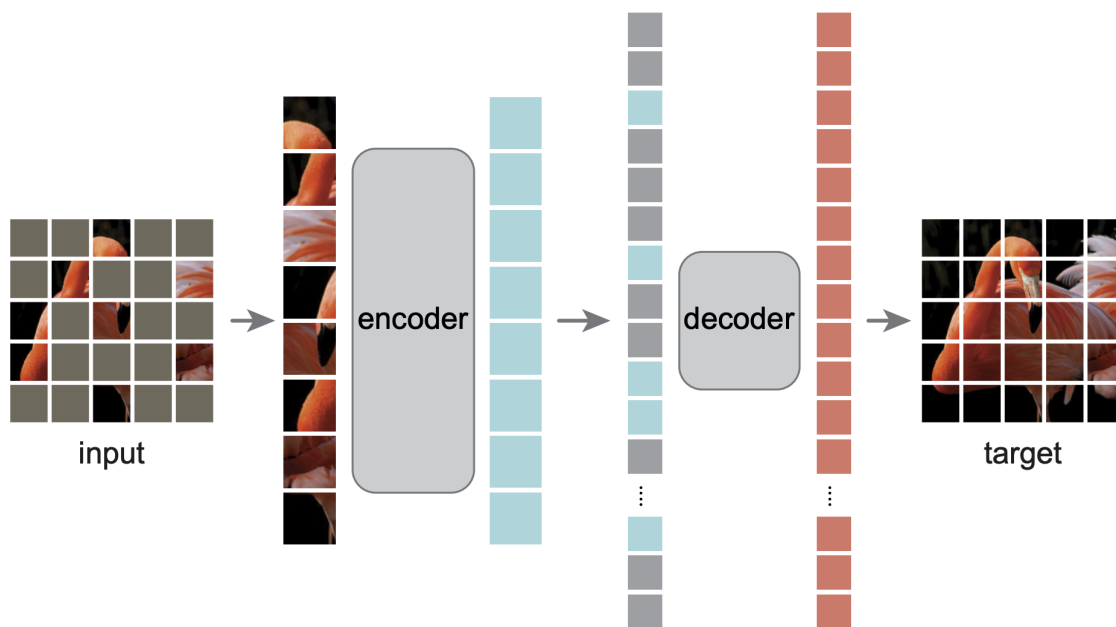


Figure 32. The abstract ViTMAE architecture. [3]

## **Section 4:**

### **Our Experiment**

## Experimental Setup

Our task is to reconstruct images affected by raindrops, that is to reconstruct the regions lost because of the rain in the image. Here an assumption is that the ViTMAE [3] has been trained on a reconstruction task very similar to ours. The training is fast and highly scalable, so it should also perform better for our task. We finetune the pretrained ViTMAE with our synthetic dataset. The results were promising showing a significant improvement over the previous state-of-the-art deraining model [14].

We use three basic loss, one is the patch-wise mean square error, one is a VGG loss same as [14] and we add one extra loss where we finetune a ViT pretrained model with our dataset to classify rainy image and non rainy image and use it as a discriminator to our model.

## **Section 5: Results**

## Results




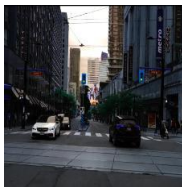

Rainy					
Predicted					
Ground Truth					
	a)	b)	c)	d)	e)

Table 1: The output of our model on our light rain dataset





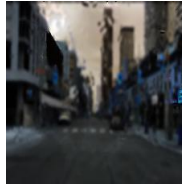






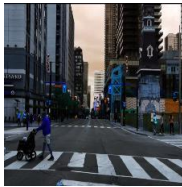


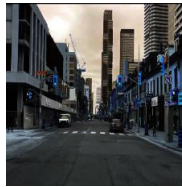
Rainy					
Predicted					
Ground Truth					
	a)	b)	c)	d)	e)

Table 2: The output of our model on our heavy rain dataset

Referring to the Table 1 and Table 2, this display the output of out model for our heavy and light rain datasets. The model when trained on a mixture of the two datasets reports and average PSNR of 27.3. When the model is trained on the light rain dataset it reports a PSNR value of 33.4 when tested on the same dataset. While when trained on the heavy rain dataset the model reports a average of 24.3 PSNR value when tested on same dataset.

Model Name	PSNR	SSIM
Original	24.09	0.8518
Eigen13[21]	28.59	0.6726
Pix2Pix[21]	30.14	0.8299
Qian et al.(no att.)[14]	30.88	0.8670
Qian et al.(full att.)[14]	31.51	0.9213
Horia Porav.et al.[21]	31.55	0.9020
Ours(ViTMAE finetuned)	28.3	0.86

Table 3. Comparison to existing state of the art

Experiment	PSNR	SSIM
Without VGG loss and discriminator	24.4	0.83
With VGG loss and discriminator	32.3	0.93

Table 4. Comparison of increase in performance with a VGG loss on our light rain dataset

From Table 4 it can be observed that adding a VGG loss and a ViT discriminator can significantly increase the image quality.



Experiment	PSNR	SSIM
Qian et al.(full att.)[14]	17.4	0.67
VitMAE	24.3	0.83

Table 5. Comparison on our heavy rain dataset

Experiment	PSNR	SSIM
Qian et al.(full att.)[14]	24.2	0.82
VitMAE	32.3	0.93

Table 6. Comparison on our light rain dataset

The paper [14] does not have a training code attached to it so the above data Table 5 may not reflect the actual performance of the paper.

## **Chapter 2**

## **Section 1: Introduction**

## Introduction

In most supervised deeplearning models dealing with deraining a dataset containing rainy, as well as their clean ground truth, is required. It is almost impossible to capture the exact same background affected by rain and also not affected by rain at the same time. In papers [14] they have attempted to capture by fixing a glass panel and first capturing the background without any noise or rain and then they sprinkle the glass panel with water and then again capturing the background through it, even though this is a good attempt to achieve the rain effect but the resulting clean and rainy image pair often do not have the same background, due to environmental noise such as dust or environmental lighting changing. In the paper [21] they develop a device that can be fixed in front of a car and has two cameras one with water droplets rolling in front of the camera and another camera placed next to it with a clear screen, the resulting image pairs resemble a rainy and nonrainy image pair, this method suffers from a slight perspective shift in the image pair. It can be concluded there exists no method where we can capture natural rainy and nonrainy images of the same background. For our study, we develop a synthetic dataset where we take dashboard videos of cars and we simulate rain using unity and fluid simulation.

Modelling natural phenomena, like fire, clouds, and water, is not an easy task due to their multiformity and complex motion. Studying the behaviour of liquids, in particular, can be traced far back, but methods for animating such phenomena were first proposed around 1980, with the rise of the computer graphics field. Rain is in many ways a complex phenomenon with many components, and how one should approach to model and render it depends on the characteristics of the problem and the limitations of the hardware. The largest discrepancy in the approaches is realism versus performance - some simulations strive to find a valid mathematical description of a phenomenon, focusing on precision and numerical accuracy, while others, for example in entertainment media, focus on a convincing visual representation and performance. In our case, we are not concerned with performance as we are not trying to achieve real-time simulation instead we are trying to achieve a hyperrealistic rain simulation.

Our main focus is on simulating rain on a surface of a windshield of a car, so we study various properties of the raindrops and how it interacts with a flat surface

and how the light travels through it to the camera. We study the several forces acting on the drop of water because they will affect the shape and size of the water drop, we also pay attention to how a water drop interacts with other water drops when they come in contact.

Unity is a game engine and was developed for the purpose of game development, but in recent years it has been used in various simulation tasks. Game engines have a physics engine that helps developers simulate various physical behaviors in nature such as gravity, wind, acceleration, and many other physical phenomena. One such physical behaviour is the fluid simulation, this effect is achieved by simulating several particles which behave like water particles and display properties like, they stick together when in contact, the interparticle motion is controlled by a fixed parameter, and the particles experience gravitational forces, etc. When large numbers of such particles are generated and come in contact they can demonstrate fluid-like behaviors. These have been widely used in game development. In this study, we use this fluid simulation technique to simulate rain drops on a car's windshield.

Modeling the way light travels through the rain particle is also important. Most game development engines provide the ability to write shaders for any 3D model or geometry, shaders are codes that define how the model behaves when displayed on the screen, mostly shaders run in the GPU. To achieve the raindrop effect we use Obi fluid's shaders which are able to replicate almost exactly how light travels through water and give a hyperrealistic look to the simulation.

Our main purpose is to recreate how the cameras in self-driving cars are affected in bad rainy conditions. Unity allows us to place the camera in front of a video exactly and we place a transparent plane that behaves as the windshield between the camera and the video, we simulate the fluid on the glass panel and observe how the fluid interacts with the glass. This is sufficient to recreate the effect of rain hitting the surface of a car's windshield. We capture the images from the unity camera and that becomes our rainy image and the ground truth is the video itself.

## **Section 2: Dataset Creation**

## Dataset Creation

Due to the lack of real-world paired data, various synthetic datasets have been developed like the Rain12, Rain100L and Rain100H [12,15] by combining background images and suitable rain streaks with the help of tools like photoshop. The Rain12 includes 12 synthesized rain images with only one type of streak, the Rain100L is a synthesized data set with only one type of rain streak and the Rain100H is synthesized with five streak directions.

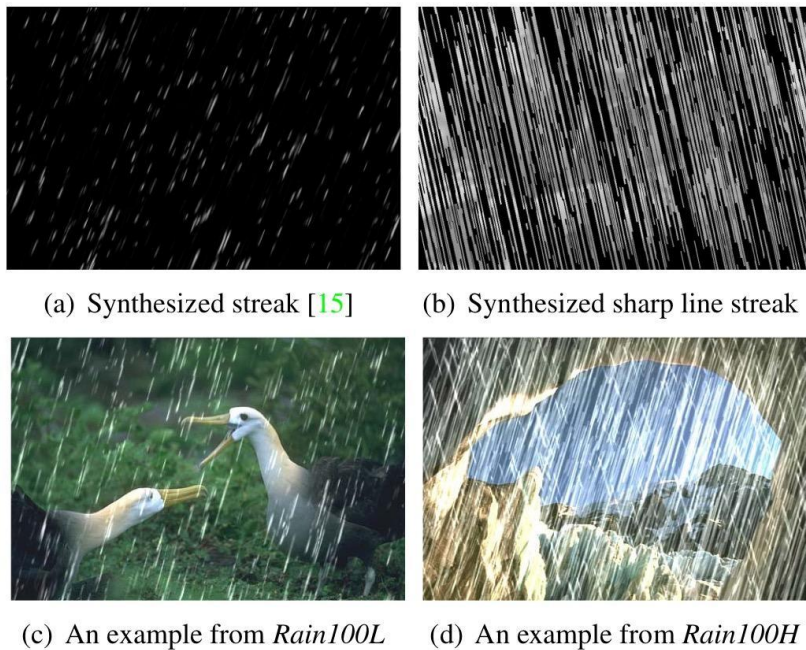


Figure 33. Sample of synthetic rain streaks and rain images.[15]

In the paper “Attentive Generative Adversarial Network for Raindrop Removal from A Single Image” [14] they note that their model just like any other deep learning model requires a huge amount of paired data. They decided to create a new dataset. They used two identical pieces of glass, one was kept clean while the other was sprayed with water. The identical glasses were of exactly the same thickness thus ensuring no misalignment caused due to different refractive indices.



Figure 34. Sample from the “Attentive Generative Adversarial Network for Raindrop Removal from A Single Image” paper and their ground truth.[14]

They captured the data with the distance between the glass and camera varying from 2 to 5 cm to generate different raindrop images and to minimize the reflection effect.

We require a similar [8] dataset but we require a video dataset and it is difficult to capture real-world paired rain video, so we decided to create a video database for training our deep learning model.

In the Master’s thesis of Katerina Koblik “Simulation of rain on a windshield” they simulate rain on the surface of a windshield of a car. They accurately simulated the effects, and key takeaways from the simulation are the appearance of a single drop, the motion of individual drops, trails that each drop may leave and the distribution of the drops on the windshield's surface.



## **Section 3: Theory**

## Theory

Simulation mostly tries to replicate the physics accurately but often opt to modify these behaviors to simplify implementation and improve performance. In this section, we discuss some physical properties that we try to simulate.

### Size

It is observed that falling raindrops have a diameter roughly of 0.5mm and 5mm, but this depends on the type of rain. The drop size distribution can be explained by the Marshall-Palmer distribution [22].

This distribution model explains that the amount of drops of diameter  $D$  can be written as

$$N(D)_{MP} = N_0 e^{-AD}$$

Where  $N_0 = 0.8 \text{ cm}^{-4}$ ,  $A = 41R^{0.21} \text{ cm}^{-1}$  and  $R$  is the rate of rainfall in mm/hour. This gives a linear correlation between  $\log^{10}N(D)$  and the diameter of the drops, which gives an exponential drop size distribution.

On the other hand, the drop diameter on a surface is harder to estimate. Due to the cohesion effect, in absence of any interfering forces, the drops will start merging together if they touch one another. If forces like wind or non-perpendicular gravity are affecting the drops they may separate into smaller drops, making it harder to determine the diameter of the individual drops.

### Shape

While falling drops are frequently pictured as having a "tear" shape, this is far from the truth; in actuality, they are more like spheres since this reduces their surface area.

Aerodynamic forces cause larger drops to distort into parachute-like forms with the majority of their volume at the base. Drops with a diameter of less than 2 mm often maintain their spherical shape as they descend through the air.

A drop will typically split into several smaller drops at a diameter of over 5 mm, as seen in Figure 1.

Depending on size, the speed of the falling droplets varies, although they frequently have a terminal velocity between 0.7 m/s and 13 m/s.

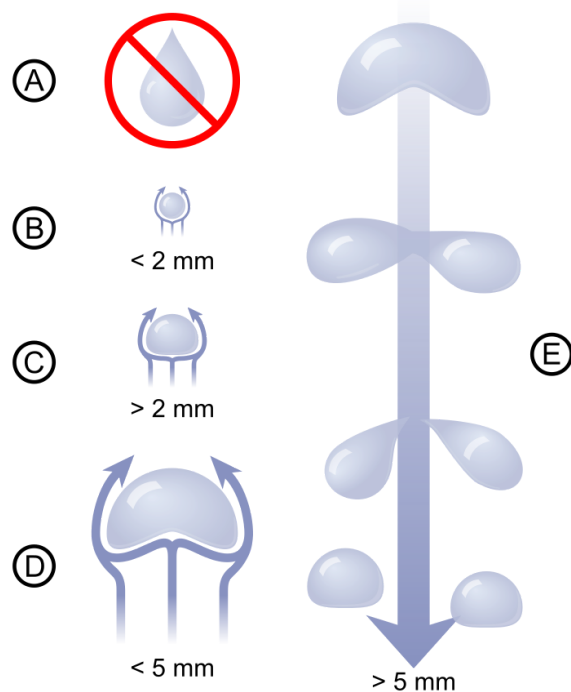


Figure 35. Change of the drop diameter and shape as it falls.[27]

If the surface on which the drop is placed is uniform and horizontal, the drop will still leave a circular "footprint."

The shape is no longer circular and is now determined by the drop's contact angle with the surface.

The drop behaviour will be classified as hydrophilic or hydrophobic based on this contact angle, which is related to the interface-induced tensions between the air, the liquid, and the surface.

Additionally, the drops' attempts to minimize their energy will cause them to deform as a result of external forces like gravity and wind drag. Larger drops will have their spherical forms flattened by gravity, and if the surface is inclined, gravity will also change the drops' contact angles. The contact angle will now vary around the base of the drop when viewed from above, and one can then identify the advancing (a) and receding (r) contact angles defined by the largest and smallest contact angles, respectively. Figure 36 also demonstrates these. These two angles can be compared to determine whether the drop will remain stationary, move, or completely separate from the surface.

A final word on drop shape: it's important to remember that a number of other factors, such as dust particles or surface damage, can also have an impact on the way that drops are shaped and how they move across a surface. Applying noise to the drops' movement and/or footprint is a common solution to this behaviour, which is much harder to model accurately.

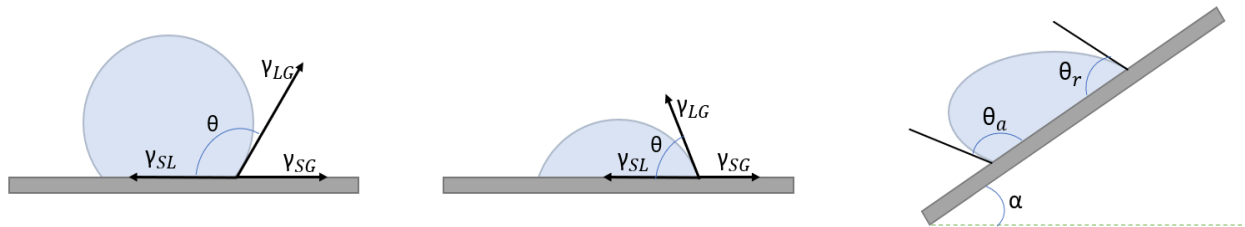


Figure 36. The water droplet on a flat surface.[27]

## Forces

The motion of drops on a car side-window is modelled in an article by J. André et al. [23] , which served as the main inspiration for the model used for force calculations in this report. Due to its relative simplicity, this model was chosen for the project's needs. The article by N. Nakate et al. [22] also provided some information.

It's crucial to keep track of the forces because they determine how the drop itself deforms and moves. We will examine the external forces—gravity and wind drag—as well as an internal opposing force that works to balance the external forces in this report. The adhesive qualities of water, which make it "stick" to other surfaces, are primarily responsible for the resisting force.

## Appearance



Figure 37. Picture of a clear raindrop.[27]

Water droplets act as lenses, refracting and reflecting their surroundings because of their shape and transparency.

For an illustration of how the background might appear through a drop, see Figure 3. A light ray will split into a refracted (also known as a transmitted) ray and a reflected ray when it travels from air to water.

The incident angle,  $\theta_i$  of the ray, and the refractive indices of the two media affect how much light is reflected and transmitted. Most light is transmitted and most is reflected for water-to-air rays in the range  $\text{abs}(\theta_i) < 49$ . This range is a little bit less for glass. In other words, if one knows the incident angle, one can ignore the splitting of the ray and only pay attention to the one moving in the direction where the majority of light travels. For reflected rays incident angle  $\theta_i$  and reflected angle  $\theta_r$  are the same.

For refracted ray, the relation is given by snell's law:

$$n_i \sin \theta_i = n_r \sin \theta_r$$

Where  $n_i$  and  $n_r$  are the refractive indices of the two mediums

There are several steps that need to be taken into account when examining the paths of light rays when looking at a background through a glass-covered in drops. Four possible ways for a ray to pass through are shown in Figure 4. Observe how the background is

drawn from an (arbitrary) viewpoint while the light is actually coming from outside the car and entering the eye/camera. The figure is based on the typical calculation method for ray tracing in 3D rendering.

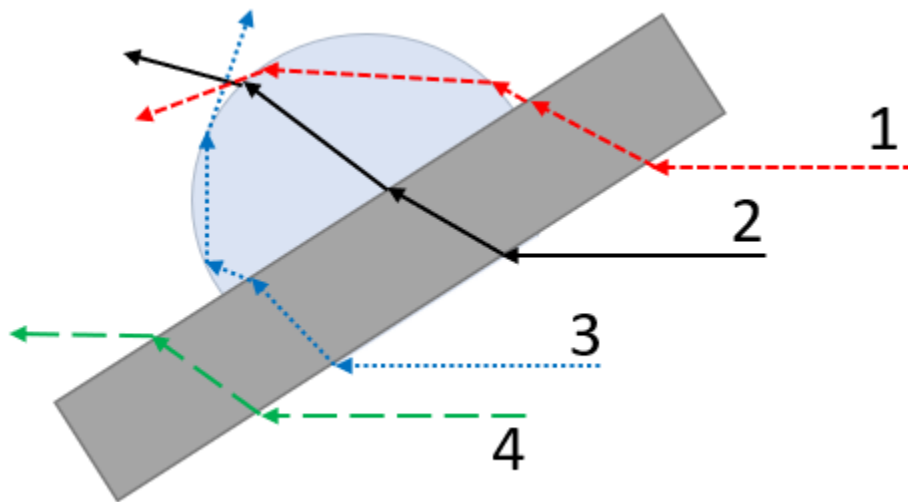
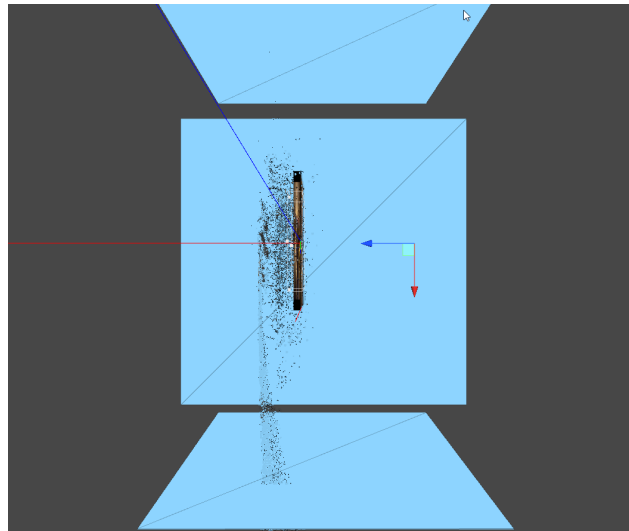


Figure 38. Figure showing how light travels through a drop of water on a flat surface[27]

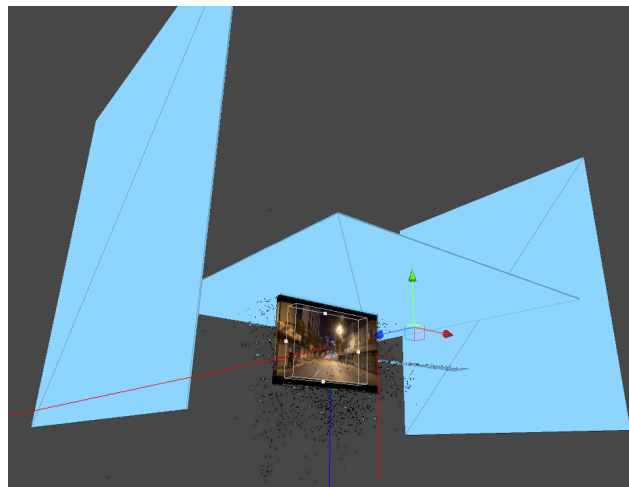
It's also important to note that the water drops' opacity and the intensity of the light that affects them are related.

## **Section 4: Our Setup**

## Our setup



a)



b)

Figure 39. A different view of our setup in unity. a) top view, b) 3/4th view





Figure 40. A screenshot from unity when simulating the rain

The figure 39 shows our setup in unity. We use a library called the obi fluid which uses a fluid simulation techniques to simulate fluid particles. The Obi fluid implements technique from papers [24,25,26] which helps to model fluid behaviour almost accurately. This technique is slower than the technique used in [27] but the effect is much more realistic. The paper [27] mainly focuses on real-time simulation which is not a concern for our test case.

## **Section 5: Results**

## Results

We generate two different sets of data. One is the light rain dataset where the area adhered by raindrops is less, and the other is a heavy rain dataset where the area adhered by the raindrops is more (more than 80% of the background is covered).

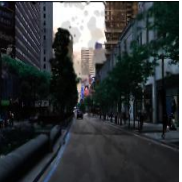
Clean					
Rainy					

Table 7. Sample from our light rain dataset




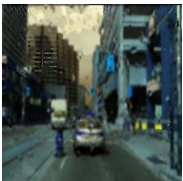
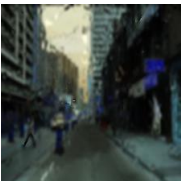
Clean					
Rainy					

Table 8. Sample from our heavy rain dataset

## **Chapter 3**

## Conclusion

In this study we discuss the issue of environmental conditions affecting the results of computer vision tasks, particularly environmental conditions like rain. Rain adhered to the surface of a camera lens or the windshield of self-driving cars can affect the outcome of computer vision tasks to a large extent. The regions of the image affected by the rain are almost unrecoverable. Almost all deep learning algorithms depend on a good input image and their ground truth pair. Capturing rain-affected pictures and capturing their clean background is a difficult task. We first develop a simulation in unity to replicate the effect of the rain on a windshield. We then use this dataset to train a Vision Transformer-based model to reconstruct the rain drop affected images.

## **Section 6: References**

## References

1. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In NIPS, 2017.
2. Dosovitskiy, Alexey & Beyer, Lucas & Kolesnikov, Alexander & Weissenborn, Dirk & Zhai, Xiaohua & Unterthiner, Thomas & Dehghani, Mostafa & Minderer, Matthias & Heigold, Georg & Gelly, Sylvain & Uszkoreit, Jakob & Houlsby, Neil. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.
3. He, K., Chen, X., Xie, S., Li, Y., Dollar, P., & Girshick, R. (2022). Masked Autoencoders Are Scalable Vision Learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 16000-16009).
4. Jay Alammar. The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. Jalammar.github.io. <https://jalammar.github.io/illustrated-transformer/>. [Online;accessed 20-June-2021]
5. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., & Amodei, D. (2020). Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems* (pp. 1877–1901). Curran Associates, Inc..
6. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (cite arxiv:1810.04805Comment: 13 pages)
7. J. Pu, X. Chen, L. Zhang, Q. Zhou and Y. Zhao, "Removing rain based on a cycle generative adversarial network," 2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA), 2018, pp. 621-626, doi: 10.1109/ICIEA.2018.8397790.
8. X. Jin, Z. Chen, J. Lin, Z. Chen and W. Zhou, "Unsupervised Single Image Deraining with Self-Supervised Constraints," 2019 IEEE International Conference on Image Processing (ICIP), 2019, pp. 2761-2765, doi: 10.1109/ICIP.2019.8803238.
9. Fu, Xueyang & Huang, Jiabin & Ding, Xinghao & Liao, Yinghao & Paisley, John. (2016). Clearing the Skies: A Deep Network Architecture for Single-Image Rain Removal. IEEE Transactions on Image Processing. PP. 10.1109/TIP.2017.2691802.

10. Y. Wei et al., "DerainCycleGAN: Rain Attentive CycleGAN for Single Image Deraining and Rainmaking," in *IEEE Transactions on Image Processing*, vol. 30, pp. 4788-4801, 2021, doi: 10.1109/TIP.2021.3074804.
11. Cai, Linwen & Li, Si-Yao & Ren, Dongwei & Wang, Ping. (2019). Dual Recursive Network for Fast Image Deraining. 2756-2760. 10.1109/ICIP.2019.8803308.
12. Wenhan Yang, Robby T. Tan, Jiashi Feng, Jiaying Liu, Zongming Guo, Shuicheng Yan; Deep Joint Rain Detection and Removal From a Single Image; *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1357-1366
13. Fu, Xueyang & Huang, Jiabin & Zeng, Delu & Huang, Yue & Ding, Xinghao & Paisley, John. (2017). Removing Rain from Single Images via a Deep Detail Network. 1715-1723. 10.1109/CVPR.2017.186.
14. Qian, Rui & Tan, Robby & Yang, Wenhan & Su, Jiajun & Liu, Jiaying. (2018). Attentive Generative Adversarial Network for Raindrop Removal from A Single Image. 2482-2491. 10.1109/CVPR.2018.00263.
15. Y. Li, R. T. Tan, X. Guo, J. Lu and M. S. Brown, "Rain Streak Removal Using Layer Priors," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 2736-2744, doi: 10.1109/CVPR.2016.299.
16. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'14)*. MIT Press, Cambridge, MA, USA, 2672–2680.
17. Jun-Yan Zhu\*, Taesung Park\*, Phillip Isola, and Alexei A. Efros. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", in *IEEE International Conference on Computer Vision (ICCV)*, 2017.
18. Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. *Neural computation*. 9. 1735-80. 10.1162/neco.1997.9.8.1735.
19. Vaswani, Ashish, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser and Illia Polosukhin. "Attention is All you Need." *ArXiv abs/1706.03762* (2017): n. pag.
20. N. Nakata, M. Kakimoto, and T. Nishita, "Animation of water droplets on a hydrophobic windshield," 06 2012.
21. Porav, Horia & Bruls, Tom & Newman, Paul. (2019). I Can See Clearly Now : Image Restoration via De-Raining.
22. Nakata, M. Kakimoto, and T. Nishita, "Animation of water droplets on a hydrophobic windshield," 06 2012.
23. J. André, C. Brochet, Q. Louis, A. Barral, A. Guillen, F.-T. Goh, A. Prieto, and T. Guillet, "Motion of rain drops on a car side window," *Emergent Scientist*, vol. 3, p. 3, 2019.
24. Miles Macklin, Matthias Müller, *Position Based Fluids*, *ACM Transactions on Graphics (SIGGRAPH 2013)*



25. Iván Alduán, Ángel Tena, Miguel A. Otaduy, *DYVERSO: A Versatile Multiphase Position-Based Fluids Solution for VFX*, (Computer Graphics Forum 2017)
26. Bo Ren, Xiao Yan, Chen-Feng Li, Tao Yang, Ming C. Lin, Shi-Min Hu, *Fast SPH Simulation for Gaseous Fluids* (2015)
27. Katerina Koblik, Daniel Wiberg, Andreas Nordenström. Simulation of rain on a windshield: Creating a real-time effect using GPGPU computing, Master's Thesis in Engineering Physics. (20-June-2021)
28. J Utah, Toronto 4K - Skyscraper District - Driving Downtown, YouTube (Jul 16, 2019)
29. J Utah, Toronto 4K - Driving Downtown - New York City 4K - Midtown Morning, YouTube (Nov 29, 2021)
30. J Utah, Toronto 4K - Driving Downtown - New York City 4K - USA, YouTube (Mar 25, 2018)