Institute of Computer Science at the Freie Universität Berlin

Dahlem Center for Machine Learning and Robotics

Bachelor Thesis

Evaluating Self-Supervised Vision Transformers For Traffic-Sign Classification Using The "DINO" Method

Hisham Chbeib

Matrikelnummer: 4932721

hisham.chbeib@fu-berlin.de

First Examiner: Prof. Dr. Daniel Göhring

Supervisor: Claas-Norman Ritter

Berlin, 21.12.2022

Abstract

Computer vision plays an essential role in the perceiving surrounding environment in the field of autonomous driving. One of the main focuses of this field is the reliable detection and classification of traffic signs, to be able to abide by traffic laws and provide a safe autonomous product.

For this detection and classification, many machine learning based approaches exist. In this thesis, a self-supervised method for training a vision transformer, a recent deep learning architecture, called "self-distillation with no labels" is discussed and evaluated on the German Traffic-sign Recognition Benchmark dataset. Moreover, the method is evaluated using a small dataset from a prototype vehicle at the Dahlem Center for Machine Learning. In total 3 models are evaluated. A model pretrained on ImageNet1K, a model further trained on the GTSRB dataset using the weights of the first model, and a model trained from-scratch exclusively on the GTSRB dataset.

With these models a k-NN classification on the GTSRB dataset containing 43 classes is performed, producing precision and recall averages of 88.61% and 84.06% for the first model respectively. The second model output better precision and recall averages of 97.77% and 96.37%. The third model achieved comparatively worse with precision and recall averages of 77.91% and 72.46%.

Contents

List of Figures

List of Tables

1 Introduction

Autonomous vehicles have been receiving a significant amount of investment recently. Many aspects surrounding autonomous driving are the subject of constant research. One of the main goals of advanced driver assistance systems (ADAS) is to increase road safety, whether by providing systems that intervene in dangerous situations, or by producing systems that provide comfort to a driver by assuming driving responsibilities in, for example, certain cases of severe weather. In order to provide a robust autonomous solution, several problems need to be solved reliably. This will enable autonomous driving to become a mainstream task and widespread service. One important aspect is adherence to traffic laws, which is mainly imposed by traffic signs present on the roads. Thus, it is essential for ADAS to be able to detect and interpret traffic signs in a real-time manner and adjust driving behaviour accordingly. This is typically handled by computer vision applications.

Many solutions to this problem have existed for a while, and most common are detection methods based on color- or shape-segmentation. Furthermore, convolutional neural networks (CNN) are often employed for both detection and classification of traffic signs. Since the introduction of transformers for natural language processing and their apparent successful impact on this application, many studies have proposed and presented adaptions of this concept for application in computer vision tasks. While convolutional neural networks are currently the go-to system for the vision-based research area, a number of different implementations of vision transformers have delivered equally good results, sometimes exceeding those of CNNs in tasks such as image classification. One downside is that transformer-based models require much more training data to allow them to be compared to CNNs; however, some improved implementations have increased efficiency with relatively smaller training datasets. Nonetheless, processing large training datasets can be cumbersome. By utilizing self-supervised learning approaches, the effort required to annotate and prepare large amounts of data for training can be greatly reduced.

This thesis is structured as follows: Chapter 2 briefly discusses related work in vision transformers and and traffic-sign detection and classification. In Chapter 3, a background on machine learning types, algorithms, and applications is presented. The structure of the dataset used, and adjustments made to it are discussed in Chapter 4. Chapter 5 explains the self-supervised method "self-**di**stillation with **no** labels" (DINO) employed in this thesis.The different models trained with the DINO method are described in Chapter 6, and the experiments are presented.

2 Related Work

This section presents examples of a few related research projects into vision transformers, and a range of studies utilizing the GTSRB dataset for traffic-sign detection and classification will be discussed.

Kolesnikov et al. [1] introduced the vision transformer (ViT) architecture by adapting the original transformer architecture with minimal adjustments as described in Section 3.1.3. Although this showed good results, it required pretraining on very large datasets such as the proprietary JFT-300M dataset [1], since use of smaller datasets did not deliver promising results. Many studies have aimed to optimize this aspect. One example is the work proposed by *Touvron et al.* [2] where they use knowledge distillation of a pretrained teacher model that is a CNN to transfer knowledge to the student transformer model via attention. This approach enabled the model to deliver a top-1 accuracy of 83.1% on the ImageNet benchmark by training the transformer on ImageNet1K [3], which contains 1.2 million images.

Li et al. [4] present an efficient self-supervised ViT (EsViT), a multi-stage ViT architecture where tokens are merged along the stages which reduces complexity at the cost of not being able to capture detailed relations between image sections. To circumvent this issue, they introduce a novel region-matching pretraining task, allowing the model to restore detailed relations between image regions. This approach achieves 81.3% on the ImageNet linear evaluation.

Many studies research traffic-sign detection and classification systems with varying approaches. These are commonly color- and shape-based approaches. *Moutarde et al.* [5] present a two-stage process and utilize a shape-based detection method to detect circular and rectangular traffic signs, while employing a neural network for character recognition and classification, after filtering digit characters from the traffic sign. Their system therefore avoids the effects of color inconsistencies and lighting conditions. They employ Hough transformation for circular shape detection and a specially designed method for the edge detection of rectangular shapes. They achieved 90% successful detection and a 99% classification success rate.

Color-based approaches are implemented widely due to the nature of traffic signs in general, which have striking features such as bright concentrated colors. In [6], the researchers propose a segmentation method by adaptively adjusting the color threshold. The threshold is dynamically chosen via a cumulative distribution function on the histogram. They also seek solutions to illumination

disruptions by using a normalization method which reduces extreme brightness in image backgrounds and they designed a detection algorithm to detect the symmetry of traffic signs relying on statistical hypothesis testing. This study achieves a detection accuracy of 94% on the GTSDB [7] dataset.

Other traffic-sign detection and classification approaches rely on machine learning as well. *Aghdam et al.* [8] present a lightweight convolutional neural network for for traffic-sign detection which incorporates dilated convolutions to employ a sliding window detection method. They also use an additional CNN for the classification task, achieving an average precision of 99.89% in detecting signs and 99.55% in classification on the GTSRB dataset [9].

3 Background

In this section, the fundamentals of this thesis will be explained. This first subsection starts with the explanation of the different machine learning types, followed by a description of the transformers and the vision transformers which are based on the transformer. In the next subsection, the results as well as approaches from previous studies will be briefly presented.

3.1 Artificial Intelligence and Machine Learning

In recent years, artificial intelligence has become the basis of many research fields, and it is applied in many areas. By utilizing statistical methods, classification and prediction tasks can be developed to produce accurate results. There is a wide variety of application areas where artificial intelligence is being actively utilized, such as in medical, economic, or industrial fields. This section will investigate the basic concepts of artificial intelligence and machine learning (ML) and provide the essential information for this thesis.

3.1.1 Types of Machine Learning

Over the years, many approaches for creating machine learning systems have been developed. One of the ways the different types of machine learning can be distinguished is according to the process by which the training of a model is performed and the amount of supervision required [10].

Teaching a model includes adjusting the weights of the model's components. Through changes in the weights of neurons during the training process, a neural network—for example—acquires knowledge on a specific area. The network learns from a curated and processed set of examples; the higher the quality of these examples, the higher rate of learning achieved by the network. These examples comprise what is known as the training data [11].

There are several different methods to train a model. These can be differentiated based on how the dataset is processed and how the training is run. Figure 3.1 gives an overview on ML methods and algorithms.

In **reinforcement machine learning**, the focus is on learning from the past experiences. A type of reward is given to the program for a correct result. This is followed by a kind of motivation to increase this beneficial outcome. Reinforcement learning is mainly used in robotics [11].

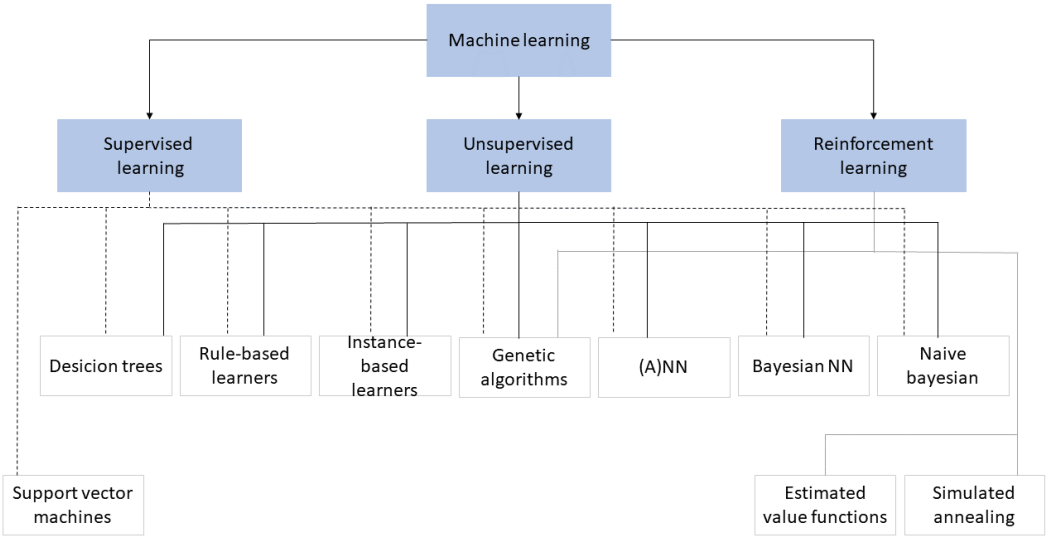

Figure 3.1: A brief overview of some machine learning types and common algorithms [12]

Unsupervised machine learning works without an already known partitioning of the input data, i.e., the output data remain unknown to the model. In this process, the model recognizes structures in the data and transforms them into actionable information. An important method of unsupervised learning is clustering. In clustering, those data that have similarities are grouped together [11].

Unlike in unsupervised machine learning, in **supervised machine learning** both the input data and the output data are known. The model in supervised machine learning is trained to produce a desired output with input data. This involves linking the known input data to the targeted output data to form a general rule. When new data are input to the model, this rule should be employed to generate new outputs [11].The desired output is referred to as labels [10].

The two important methods of supervised learning are classification and regression. Both are predictive models with which predictions about the future can be made. Regression allows predictions about continuous values. The difference between classification and clustering (unsupervised machine learning) is that in clustering, classes are created by identifying similarities in the data and grouping them together. In classification, the classes are already known, and similarities are searched for in the data set, and the data that have similarities are assigned to an already known class [11].

Semi-Supervised machine learning falls between the supervised and unsupervised ML. The main concept of semi-supervised learning is that the data are partially labeled. Despite the partial labeling, the data can be used to create

accurate models for various applications. This method can be appealing because partial preparation of training data is a time-saving measure [10].

Self-supervised machine learning is a machine learning process where the model trains itself. The goal here is to learn one part of the input based on a different part of the input. This involves identifying the hidden part of the input from each non-hidden part of the input. In self-supervised learning, the data are not labeled. The labels are automatically generated during the process. So, with this process, the unsupervised problem is converted into a supervised problem. It is necessary to set the learning objectives properly at the beginning of the process to be able to use the significant amount of unlabeled data and acquire the monitoring from the data [13].

3.1.2 Transformers

In the past, natural language processing (NLP) mainly relied on recurrent neural networks (RNNs). NLP is applied in various tasks such as translation, sentiment analysis, and speech recognition. The transformer model is a recently conceived deep learning architecture which was introduced in 2017 by a research group led by Google [14]. Figure 3.2 illustrates the entire transformer model architecture.

Transformers are tasked with solving problems concerning sequence transduction, i.e., converting an input sequence into an output sequence. In a sentence, word dependencies play a vital role in the semantic meaning that the sentence is trying to deliver. This is mostly achieved by using RNNs [15].

However, one of the main drawbacks of RNNs is the sequential operation. During the encoding stage, the words of a sentence pass through encoders based on their position in the sentence. Each encoder generates a state that is forwarded to the next encoder. The state resulting from the last encoder is then forwarded to a decoder which outputs the translation of the first word and the following state, which is in turn forwarded to the next decoder [15].

This mechanism allows for dependencies between the states to emerge. Thus, hindering parallel operation and negatively affecting training performance. While there have been some improvements, this constraint is still present [14].

With transformers, the constraints of sequential processing are resolved. Much like RNNs, the transformer architecture utilizes encoders and decoders. Figure 3.3 shows the structure of an encoder or decoder layer in a transformer. The contrast is that it allows the input sequences to be forwarded in parallel. All words are passed simultaneously, and all their embeddings are also calculated simultaneously. As previously mentioned, a word's is vital to its meaning, so an additional positional encoding vector, which has information on distances

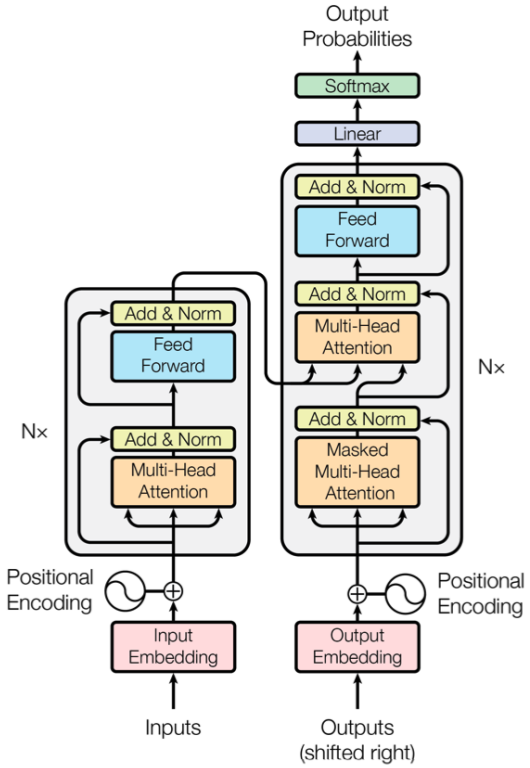

Figure 3.2: The architecture of the transformer model. Words are converted into embeddings, and positional information is attached to them. In parallel, they flow through encoder sub-layers on the left, and resulting states are used as input to the decoder. The decoder has similar sub-layers with an additional multi-head attention layer. At the output of the decoder on the right, a linear transformation occurs, and a softmax is applied, converting output into predicted probabilities [14].

between words, is added to the word's input embedding beforehand. The result is a word-embedding with positional information [14].

Encoder: The word-embeddings pass through the encoder block as vectors. The entire block contains six identical layers (encoders), and each one consists of a multi-head self-attention sublayer and a fully connected feed-forward sublayer. After each sublayer, normalization is applied. Word-embeddings pass through the attention sublayer followed by the feed-forward sublayer, which generates an output that is forwarded to the next encoder layer [14].

Decoder: The decoder block has the same number of layers (decoders) as the encoder block, with the addition of a third multi-head self-attention sublayer that is applied over the output of the encoder block. The self-attention layer in the decoder applies masking, together with offsetting output embeddings by one position to prevent positions from laying attention on consequent positions. Similarly, all layers are connected and after each one, normalization occurs [14].

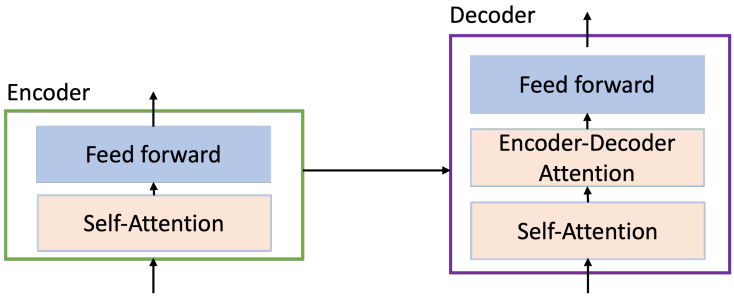

Figure 3.3: One layer of a transformer encoder containing a self-attention layer and a fully connected feed-forward network. A decoder layer has the same structure, with an additional multi-head attention layer [16, 14].

Attention: By mapping a set of key-value pairs and a query to an output, the attention function is generated. The output, key-value pairs and the query are represented as vectors for each word-embedding. By summing the weights of the values, the output is calculated, and these weights are outputted by a function which takes the query and its corresponding key as an input. It calculates the dot product of the query vector with all other key vectors, then each product is divided by the square root of the key vectors' dimension. This value is chosen to derive stable gradients. After division, a softmax is used to normalize the values. After this operation, these softmax values are multiplied by the value vectors, and the value vectors are summed to provide the self-attention layer as an output for a word [14, 16].

Put simply, self-attention relates different positions of an input sequence with each other and a score in order to compute a representation of the sequence. While processing a word, this process forms dependencies to the other words to build a form of understanding.

3.1.3 Vision Transformers (ViTs)

Given the promising results of the transformer model architecture in the field of natural language processing in areas such as computation efficiency and scalability, it was adapted to be applied in computer vision. CNN architectures had previously dominated computer vision. Vision transformer (ViT) architecture was introduced in 2020 by Google researchers with the intention for it to be utilized on a large scale [1]. The left side of Figure 3.4 portrays the vision transformer architecture, while the right side presents a layer of a transformer encoder.

The application of transformers to images requires adaption, however. Images contain a lot more information and data than text, and the main unit of operation is the pixel. Thus, applying the self-attention mechanism pixel-wise without adjustments prevents realistic utilization of the transformer architecture, as it is extremely costly to compute attention between each pixel [1].

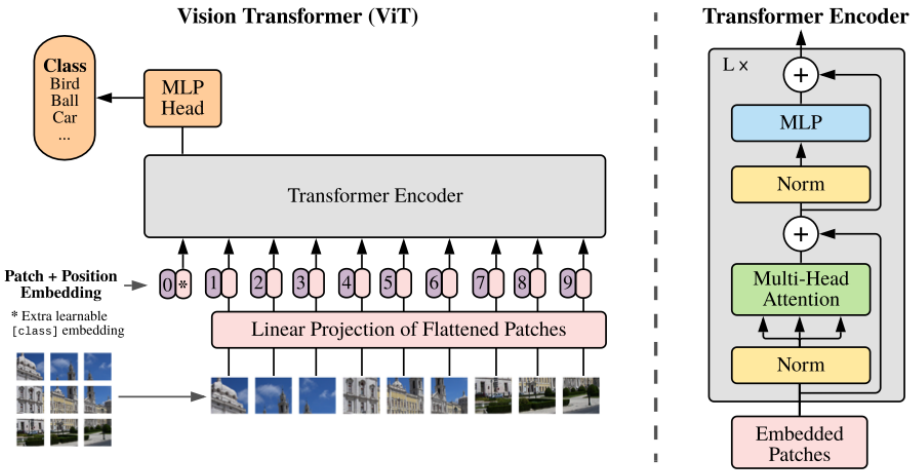

Figure 3.4: The architecture of a vision transformer. An image is divided into patches of identical size. Patches are then linearly projected, after which a positional encoding is attached. A learnable class token (CLS token) is prepended which the MLP Head uses for classification. This is then used as input to the encoder [1].

The ViT introduced follows the original transformer architecture, which is represented in Figure 3.2, from [14] as much as possible, while adapting the method for images. 2D images are divided into equally sized patches; these patches are handled sequentially. Patches have a resolution of (P, P); this results in a total number of patches of $N = Height \times Width/P \times P$, which represents the length of the input sequence, in a similar manner to the length of a sentence. Similarly, patches represent words in a sentence. Using a trainable linear projection, patches are converted to patch-embedding vectors. Furthermore, a learnable class-token embedding is then prepended to the sequence of patch embeddings. This token contains global information of the entire image and is employed to predict the class of the input. This is fed into the encoder, and only its state at the output of the encoder is forwarded to a multi-layer perceptron head for classification [1]. To preserve the positional integrity of the patches relative to the original image, positional embeddings are added to the patch embeddings. These help to encode distances between feature. The resulting embeddings represent the input to the encoder.

4 Dataset

This chapter will present the structure of the dataset used. Furthermore, Section 4.3 contains a description of the adaptations that were performed on the dataset.

4.1 Overview

For self-supervised model training, a suitable dataset is required. In the past few years, many datasets containing different traffic-sign scenes from around the world have been prepared and provided for creating and training machine learning detection and recognition models [17]. For this study, the most relevant dataset which was chosen is the German traffic-sign recognition benchmark (GTSRB) dataset [9].

Provided by the Institut Für Neuroinformatik at Ruhr-Universität Bochum, this publicly available dataset was created to be utilized in a benchmarking competition for detection and recognition models. As this dataset mainly consists of traffic signs on German roads, it was deemed the best-fit for training this vision transformer model.

4.2 Structure

This dataset is divided by it's authors into a training set and a testing set. In total more than 50,000 images are provided for training and testing. The training set contains a total of 39,209 images in Portable Pixmap format (PPM); this represents about 80% of the entire dataset. These images are divided into 43 directories, and each directory represents a single class. The traffic-sign classes are displayed in Figure 4.1. The order of the images corresponds to the class index. The first image at the top on the left side (speed limit 20 km/h) represents the class zero. A mapping of the class IDs to the different traffic signs can be found in Table 7.1 in the Appendix [9].

The training images were created by extracting frames from recorded videos taken while passing by a sign. Each class directory contains multiple tracks of a traffic-sign instance, and each track is a sequence of 30 images of one physical traffic sign. Images include a 10% border around the sign, and this border is from the actual environment the image was captured from. As the images were collected while driving on roads, there are many variations to the appearance

Figure 4.1: The 43 traffic-sign classes [9].

of the signs, e.g., the visibility and clarity of the signs are sometimes affected by lighting conditions, weather conditions, and motion blur as well as certain obscuring factors such as tree branches [9].

The classes can be interpreted into categories based on their purpose, i.e., speed, prohibitory, mandatory, de-restriction and danger signs. Image dimensions vary between 15×15 px and 250×250 px, and they are RGB color images. While the dataset provides a CSV file for each class containing annotations, e.g., image dimensions and ROIs (regions of interest), these were irrelevant for this thesis. The number of images varies between classes [9]. Figure 4.2 depicts the distribution of images before the split.

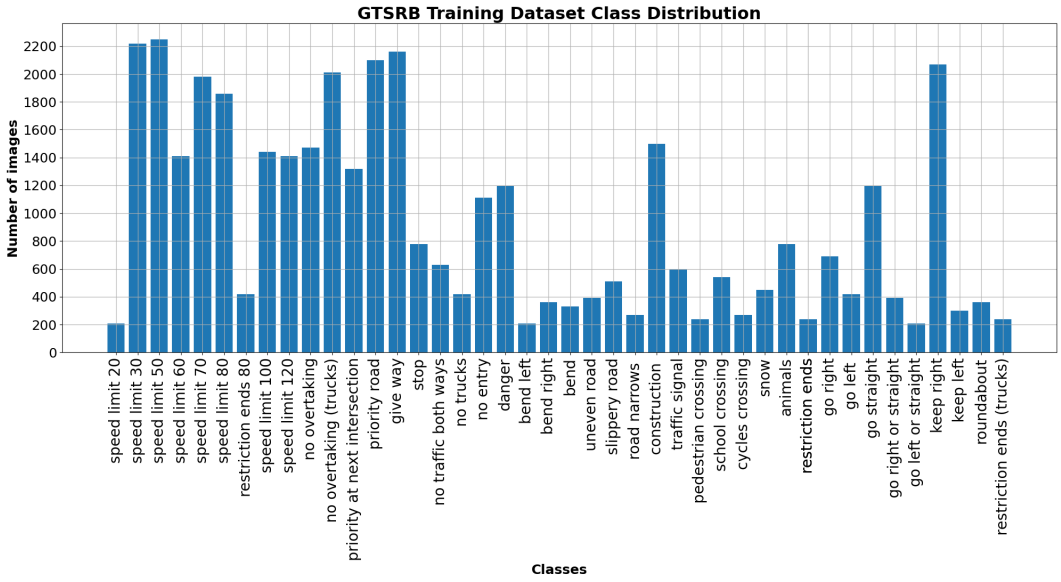

Figure 4.2: Class distribution of the GTSRB training set before the split.

Image files are named as follows: xxxxx_yyyyy.ppm. The first part is the track

number (xxxxx) and the second part is the sequence number of the frame.

In order to use this dataset for training the model with DINO, certain adjustments to it were required.

4.3 Adjustments to the Dataset

Only the training dataset from [9] was considered. Thus, the total number of images in the dataset used in this work is 39,209. To train the model using DINO, 80% of the images in the training set had to be compiled into one directory, without maintaining the class-folder structure. This posed a challenge due to the duplicate image file names.

Renaming approximately 39,000 images manually was not feasible, so this was solved by creating a python script. This script would take the path of the classes directories and iterate over each class directory. Within each directory, the script iterated over every image and prepended the class number to the image name. In this way, all the images received a unique filename with an indication of to which class they belonged.

To copy the images out of the class directories and into the main training directory, another simple python script was used. All the remaining images in the training set were used to test the model. To insert this test dataset—which is 20% of the training set—into the k-NN classifier, it had to be split into two folders, each containing 50% of the test images, i.e., 10% of the training set.

5 Self-Distillation with no Labels (DINO)

5.1 Overview

In this thesis, the focus is on the method presented by researchers at Meta AI and their application of the vision transformer architecture [18]. Their research proposes a self-supervised learning method called self-**di**stillation with **no** labels (DINO). This is used to train a vision transformer.

Knowledge Distillation: The knowledge distillation technique from [19] used for training DINO is different. In other implementations, training a smaller network designated as the student relies on a network previously trained on labeled data designated as the teacher that is also larger than the student network. While training, knowledge is transferred from the teacher-network to the student-network. The goal is to train the student-network to match the teacher-network's output. DINO builds on the approach in [2]: however, with DINO, the teacher network is constructed during training and is built from past versions of the student-network [18].

Teacher and Student: The researchers' strategy involves using two networks with an identical architecture, e.g., a vision transformer, one designated as a student $g_{\theta s}$, and the other as a teacher $g_{\theta t}$. However, both have different sets of parameters θ_s and θ_t. Both networks are initialized with the same weights. During training, only the student is updated with a stochastic gradient descent and the parameters of the teacher are updated by the student by employing an exponential moving average (EMA) on the student weights, where the update rule is $\theta t \leftarrow \lambda \theta t + (1 - \lambda)\theta s$, and where λ follows a cosine schedule while training. A feature vector of dimension K is what both networks output. This is normalised using a temperature softmax over the entire feature dimension to produce the probability distributions P_s, P_t. Sharpness of the distributions is controlled by the temperature parameters $\tau_s > 0$ and $\tau_t > 0$ for both networks. Distributions are matched between the student and the teacher by minimizing the cross-entropy loss with reference to the student's parameters. To ensure only the student's gradient is updated, a stop-gradient is applied at the teacher's output [18]. Figure 5.1 summarizes the functionality of DINO.

Avoiding collapse: With self-supervised learning, it is essential to prevent the model from collapsing, and avoid mapping inputs to a trivial output. While this has been achieved previously with various methods, such as through contrastive loss [20] and batch normalization [21], the authors [18] relied exclu-

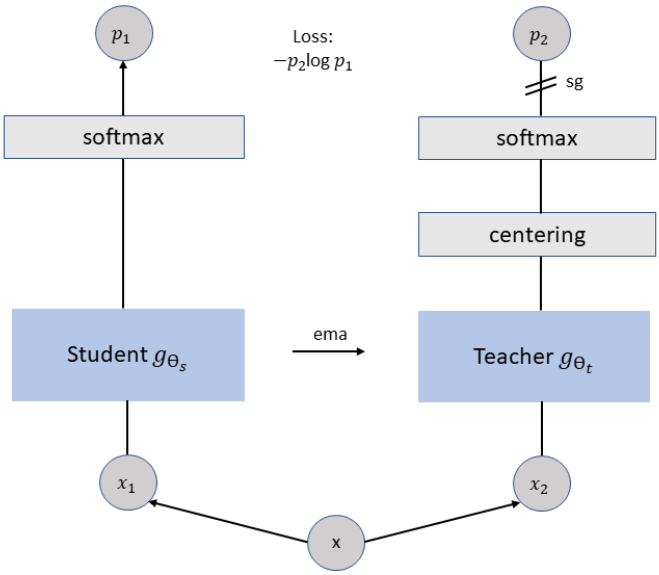

Figure 5.1: Operation concept of DINO. Two different random views of an image are fed into both the student and teacher networks having the same architecture with differing parameters. Only on the teacher's output centering occurs. Feature outputs of dimension K of both networks are normalized using a temperature softmax to obtain probability distributions P_1 and P_2. Distributions are matched by minimizing the cross-entropy loss. By applying a stop-gradient on the teacher, gradients are updated only via the student. The parameters of the teacher are adapted by applying an exponential moving average of the student parameters. [18].

sively on performing two operations during training, i.e., centering and sharpening of the teacher outputs. Sharpening and centering play contrasting roles. While sharpening encourages the domination of one dimension and prevents collapse to a uniform distribution, centering affects these aspects in an opposite manner by preventing one dimension from dominating and encouraging a uniform distribution. Together with updating the *EMA*, these operations prevent model collapse [18].

Multi Cropping and Augmentation: Images in the dataset are processed such that a set of views containing two distinguishable representations of the image, i.e., crops, is generated. Researchers can apply a multi-crop strategy as in [22]. The aim of this is to adapt the cross-entropy loss minimization to self-supervised learning.

Each set of views of an input image contains two global crops, each covers at least 50% of the image. Global crops may overlap, and they provide a general representation of the original image. The views also contain a sequence of smaller patches called local crops which provide a more local representation of the image. Local crops cover between 5% and 50% of the image. These could

overlap as well, and the number of local crops to be generated for training can be modified. The student network receives both global and local crops, whereas the teacher network only receives global crops in order to promote "local-to-global" correlation of the extracted features. The views are passed through an augmentation process similar to [21]. Augmentations include random rotation, color adjustments such as solarization, and the application of Gaussian blurring [18].

6 Experiments

This section focuses on the different experiments, and the different models that were trained are described. The k-NN classifier results are also presented for these different models.

6.1 Trained Models

6.1.1 Pretrained Model

Overview

This is the model provided by the researchers from Meta AI. The dataset used to train this model is ImageNet1K [3]. This dataset contains 1,000 classes and more than 1.2 million training images, 50 thousand validation images, and 100 thousand test images. There are around 600 to 3,000 images per class in the training set, 50 images per class in the validation set, and 100 images in the test set.

This model's architecture is a *small* vision transformer (ViT-S/8), meaning it has 384 feature dimensions, 6 multi-attention heads, and 21 million parameters. It was trained without any labels. Training was run on 64 GPUs distributed on 8 machines, with 8 GPUs per machine. It was run for 800 epochs. The model has a patch resolution of 8×8 px and batch size per GPU of 16, totaling 1024 images per batch. While a ViT-S/16 with a patch size of 16×16 px is also provided, the model with the smaller patch size was chosen, given that it delivers better performance [18].The dimension of the output K is 65536. The teacher temperature parameter τ_t is set to 0.04 and is linearly increased during the first 30 Epochs to 0.07, while the student temperature τ_s is set to 0.1 through the entire training. Learning rate is also increased during the first 10 epochs following a linear scaling rule [23] $lr = 0.0005 * \text{batch size}/256$; afterwards, the learning rate is decreased using a cosine schedule [24]. Weight decay also follows a cosine schedule, starting at 0.04 and ending with 0.4. For this training, multi-cropping was employed by setting the number of local crops to 10, while global crops remained at a default count of 2. The global crops scale ranged from 40% to 100% of the original image. The local crops scale ranged from 5% to 40%. Global crops were set to a resolution of 224×224 and local crops to a resolution of 96×96 [18].

The optimizer used for training is *adamw* [25], which is an improvement of the optimization algorithm *adam* [26] achieved by applying enhancements to the

loss function's optimization steps and decoupling weight decay.

Attention Visualization

By utilizing the provided `visualize_attention.py` utility [18], a visualization of the self-attention of the CLS token on the attention heads from the last layer can be generated. Each head focuses on a different region or object in the image, be it a character boundary or an object within the image. This makes it possible to observe which regions and objects of an image are weighted most heavily for the model, with areas of higher attention being brighter and areas of lower attention being darker.

Self-attention maps in Figure 6.1 through Figure 6.4 were checked against several example images from the GTSRB dataset. These images were not used for training the model. It was noticed how DINO "views" different aspects of a sign and segments objects. The leftmost picture is the input image for which a self-attention visualization is generated, while the rightmost picture depicts a mean representation of all six generated heads. This was generated by converting the six attention images to arrays containing image RGB values of each pixel and compiling them into one array. On this array, the NumPy function `numpy.mean()` is applied; this calculates the mean RGB array representation, that is then converted back to an image using the `PIL.Image.fromarray()` function.

The images represent different traffic-sign shapes, colors, and contents, where it can also specifically be seen that the hexagonal boundaries of Figure 6.2 are outlined, as well as the triangular shape of Figure 6.3. In Figure 6.4, the focus on the arrow object is present in all heads, but for example, head 0 relates to the disfigurement with the sticker, where head 1 relates more to the area in the background of the image and the traffic-sign pole.

6.1.2 Post-trained Model

Overview

This section discusses whether further training of the pretrained DINO model on the GTSRB dataset yields better results for application of ViTs on traffic-sign images.

It was possible to take the last checkpoint from the pretrained model, which was trained on ImageNet1K for 800 epochs, and train it further on more than 31 thousand images from the GTSRB training dataset. As mentioned in Chapter 4, these images amount to 80% of the GTSRB training dataset. All images were provided in a single folder with no labels as input.

The model was trained for 50 epochs on a machine in the Dahlem Center

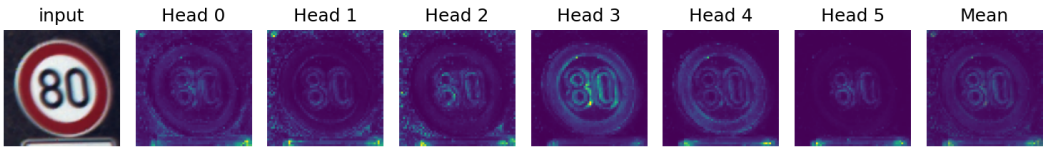

Figure 6.1: Input image, generated attention heads, and a mean representation of a speed-limit-80 sign – pretrained model.

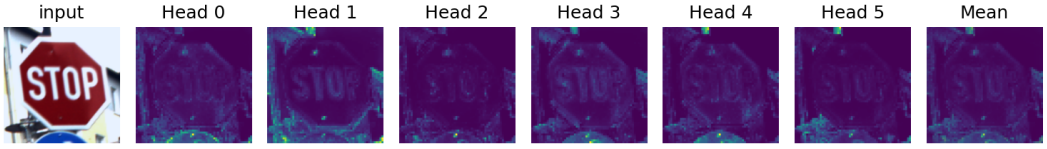

Figure 6.2: Input image, generated attention heads, and a mean representation of a stop sign – pretrained model.

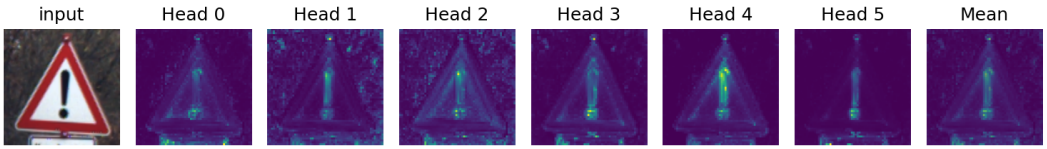

Figure 6.3: Input image, generated attention heads, and a mean representation of a danger sign – pretrained model.

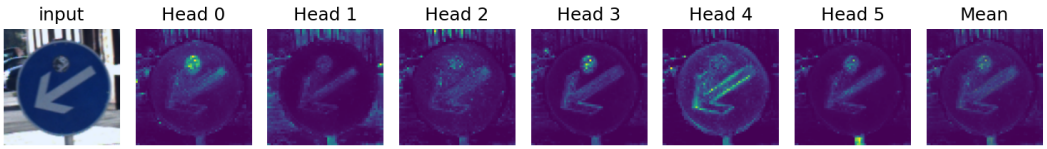

Figure 6.4: Input image, generated attention heads, and a mean representation of a keep-left sign – pretrained model.

for Machine Learning and Robotics (DCMLR). The machine had two Nvidia GeForce GTX 1080 Ti GPUs, both having 11 GB of GDDR5X RAM. The GPUs ran CUDA Version 11.8. Training this model for 50 epochs took two days and 14 hours.

The parameters for training this model were identical to those for the pretrained model, as we trained and adjusted the weights that had been saved. The one difference was that the teacher temperature was increased in the first ten epochs from 0.04 to 0.07. Moreover, due to the hardware limitations in comparison with the pretrained model, the largest possible batch size per GPU without exceeding memory capacity was six images. The same optimizer, *adamw*, was used for training.

Attention Visualization

For the same set of example images of traffic signs, the self-attention maps were visualized to compare how the different models notice the characteristics of the images. Given that this model was trained further on traffic-sign images, a few differences were noted.

It is noticeable in Figure 6.6 compared to Figure 6.2 that there is increased focus on the sign in the different heads. Most heads focus on the outline of the sign, as well as the *STOP* text within the sign. In head 0, the entire sign shape, text, and color are captured, while the cut out of the blue sign below it and the features in the background of the image are ignored in Figure 6.6 in contrast to Figure 6.2.

Similarly, it can be observed in Figure 6.7 that this model focuses more on the sign than on other objects, specifically the top part of the white sign below it. When head 5 from Figure 6.7 is compared with Figure 6.3, it can be noted that there is no focus on the sign below, in contrast to the same image in the previous model where some attention is dedicated to the boundaries of the sign below in all heads.

6.1.3 From-scratch Model

Overview

This model was trained solely using the GTSRB training dataset, i.e., 80% of the training set.

The training was run for 50 epochs on the same machine mentioned previously, with two GTX 1080 Ti GPUs and a total of 22GB of memory. The duration of the training was two days 18 hours and 45 minutes.

This model also has a ViT-S architecture resulting in an output dimension K of 65536 with the same resolution of 8×8 px. The number of local crops is set to eight local crops plus two global crops, with the identical global and local crop scales to those of the previous models of (0.4, 1.0) for global crops and (0.05, 0.4) for local crops. The teacher temperature parameter is increased in the first 20 epochs from 0.04 to 0.07. Similar to all models, the learning rate is increased in the first ten epochs, and weight decay is updated via a cosine schedule starting from 0.04 and ending at 0.4.

Attention Visualization

As in the previous section, the self-attention maps of the same set of images, generated by using the weights of the third model, are analyzed.

In these examples, a sharper attention can be noted in Figure 6.9 through

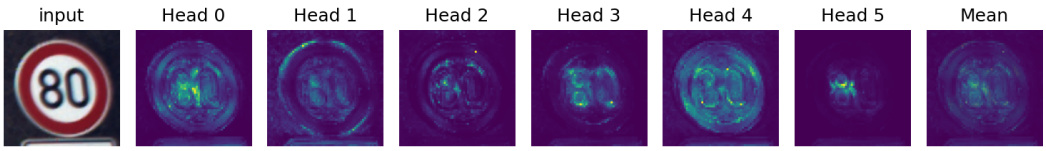

Figure 6.5: Input image, generated attention heads, and a mean representation of a speed-limit-80 sign – post-trained model.

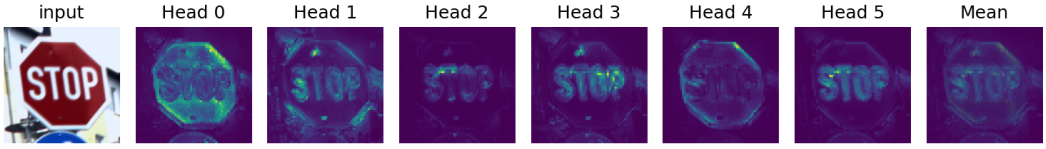

Figure 6.6: Input image, generated attention heads, and a mean representation of a stop sign – post-trained model.

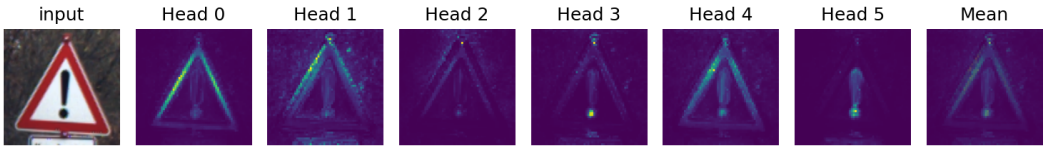

Figure 6.7: Input image, generated attention heads, and a mean representation of a danger sign – post-trained model.

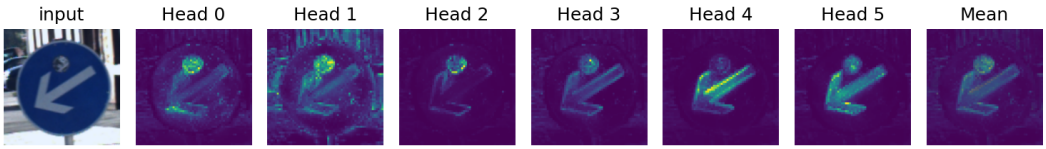

Figure 6.8: Input image, generated attention heads, and a mean representation of a keep-left sign – post-trained model.

Figure 6.12. The model focuses on finer details of the images; for example, in Figure 6.10 it can be noted that head 4 focuses on the shadow effect of the input image. Furthermore, a sharp focus can be seen to be paid to the triangular shape of the danger sign in Figure 6.11 in almost all heads, in particular in head 2, where the color changes occur, and in head 4, where the entirety of the red outline of the danger sign can be seen.

Similar observations can be made regarding Figure 6.9, where the circular shape, as well as the text and the changes in color are outlined more than in the previous two models. Presumably, this is due to the fact that this model had not seen any different objects or colors other than ones that are present in traffic signs.

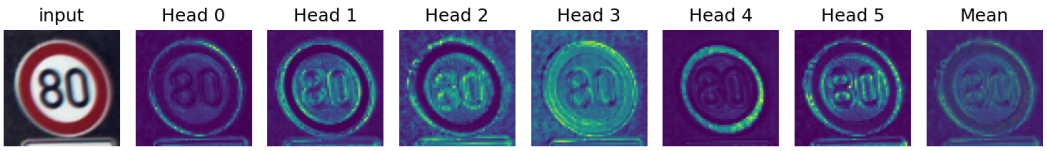

Figure 6.9: Input image, generated attention heads, and a mean representation of a speed-limit-80 sign – from-scratch model.

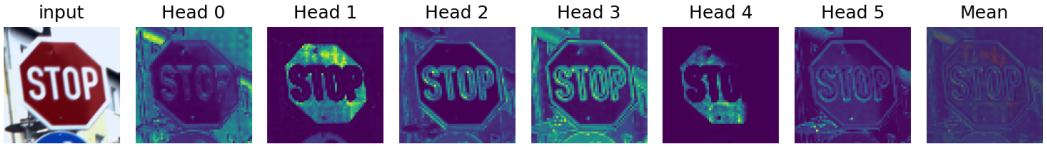

Figure 6.10: Input image, generated attention heads, and a mean representation of a stop sign – from-scratch model.

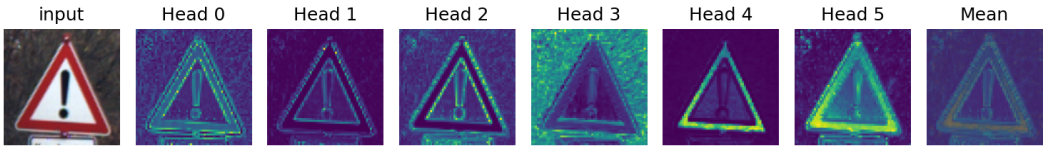

Figure 6.11: Input image, generated attention heads, and a mean representation of a danger sign – from-scratch model.

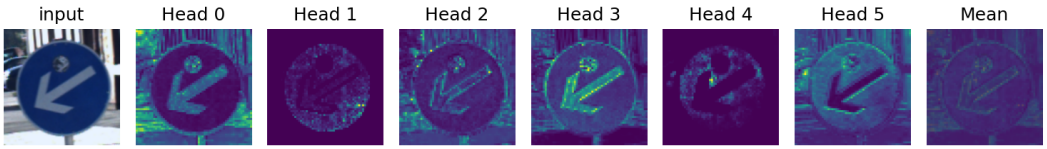

Figure 6.12: Input image, generated attention heads, and a mean representation of a keep-left sign – from-scratch model.

6.2 k-NN Evaluation

One of the main assumptions by the authors of DINO is that the ViT models trained with DINO and self-supervision provide a good level of performance in classification tasks, specifically with a nearest-neighbors classifier.

The input for the k-NN classifier had to be structured in a certain way. It required two folders, *train* and *val*, to be fed into the classifier. Within each folder, there was a set of subfolders containing images representing the different classes. The training dataset of the GTSRB had this structure. As mentioned in Chapter 4, an 80/20% split of the images was performed. The 20% was subsequently also split in half and 10% of the images were put in the *train* folder, and the rest 10% were put in the *val* folder. In the following section, images in the *val* folder will be referred to as test images, and images in the *train* folder will be referred to as train images.

The provided classifier `eval_knn.py` [18] receives the frozen weights from the

trained model and runs them through a feature extraction pipeline, where the images are loaded via a data loader for each set. This returns two feature tensors for each dataset: (3921,384) for the test images, and (3290,384) for the train images. Both feature tensors are then normalized. Furthermore, two tensors containing train and test labels are also derived from the subfolder structure; these represent the classes.

After feature extraction, both train and test feature tensors, along with their respective labels, are fed through the classification function. The classification function calculates the dot product of the test features and the train features and retrieves the top k similar train images for each test image along with their indices by sorting in descending order and fetching the top k elements. The retrieved distances of nearest neighbors are transformed by dividing by the temperature parameter $\tau = 0.07$ per default and applying the exponential function e^x. Probabilities of predicted classes for each test image are retrieved and sorted. Indices of the highest probabilities are also retrieved and saved in a tensor which is named predictions and which will represent classes with highest probabilities. Finally, an equality check is performed on the predictions tensor against a tensor containing the actual classes of the test images in order to calculate the percentage of correctly predicted classes. The classification function iterates over batches of images, where the batch size is 1% of the total number of 3,921 test images; this is rounded down to 39 images per iteration.

Due to the fact that the classifier only outputs the percentage of correct class predictions from the frozen features, some adjustments were applied. In order to match the train images to the test images and map their similarity score, the feature extraction pipeline was adjusted to also extract the full list of images from the data loader and pass them forward to the classification function. For each batch, the relevant images, labels, and similarities were extracted from the tensors and comprised into lists. In order to collect this information for all images over the batches, a simple function was implemented that took the tensors as an input. These were transferred from the GPU to the CPU and then converted into a NumPy array; then, the contents of the array were appended to the respective list aggregating all the information. Another function was implemented to extract pairs of test image and train image, and their similarity score, along with the actual class and the predicted class of the test image from the above-mentioned lists, and these were exported to a CSV file. Due to resource restrictions, it was not possible to extract all $3,921 \times 3,920$ as the resulting matrix would have contained 3,920 entries for each of the 3,921 test images, resulting in a total number of over 15 million entries in the output file, which exceeds the machine's resource capacity, i.e., the 32GB RAM available on the machine. So, the entries of the output file were capped to 100 test images $\times$ 100 train images. Furthermore, for each of the 3,921 test images, the top ten similar train images were also exported. Similarly, this CSV file

Model	Macro Average Precision	Macro Average Recall
Pre-trained model	88.61%	84.06%
Post-trained model	97.77%	96.37%
From-scratch model	77.19%	72.46%

Table 6.1: Summary of precision and recall values for the different models when classifying GTSRB dataset.

also contained image paths, actual classes, and predicted classes of test images, along with the similarity scores and the actual classes of the train images.

6.2.1 k-NN Evaluation on GTSRB Images

k-NN Evaluation – Pretrained DINO Model

Running the k-NN classifier with the weights of the pretrained model yields a top-1 accuracy in the high eighties. The classifier produces accuracies for 10, 20, 100, and 200 nearest neighbors.

Retrieving the ten nearest neighbors, the classifier is able to produce top-1 correct class predictions for 88.39% of the 3,921 of the test images. For the 20 nearest neighbors, it is able to correctly predict images classes for 88.22% of the images. For retrieving the 100 and 200 nearest neighbors, the model achieves top-1 correct class predictions for 86.78% and 86.30% of the 3,921 test images, respectively.

Due to the fact that the dataset is unbalanced, this above-mentioned metric does not give a clear picture on the efficiency of this classification accuracy. Thus, the precision and recall of all classes and a macro average of these values was calculated. Precision resembles the percentage of elements that were actually positively classified out of all elements that the classifier considered correct. Recall on the other hand represents the percentage of actual correct elements that were classified correctly [27]. Precision and recall are given by the following equations:

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{6.1}$$

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{6.2}$$

The 43×43 confusion matrix is calculated by utilising the function `confusion_matrix(targets, predictions)` provided by the `sklearn.metrics` library and subsequently the macro average precision and recall values are computed from this matrix [28]. Table 6.1 summarizes precision and recall values

for all models when using the GTSRB dataset. Overall this model produced an average precision of 88.61% and a recall of 84.06%.

The following are example images, and their top five nearest neighbors. In all figures, the test image and to its right the five train images can be seen. Each train image's corresponding similarity score is displayed above it as well, and class IDs are displayed in parentheses. It is noticeable from these examples that the classifier can make correct predictions for almost all examples, with the exception of Figure 6.16, where the fourth and fifth nearest neighbors are actually a different traffic-sign class, namely a keep-right sign. However, this is very similar to the original class, a keep-left sign.

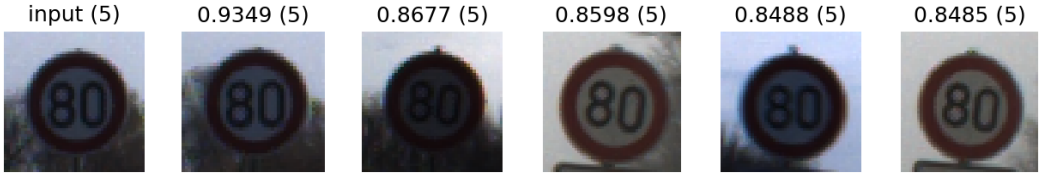

Figure 6.13: Top 5 nearest neighbors of a speed-limit-80 sign – pretrained model.

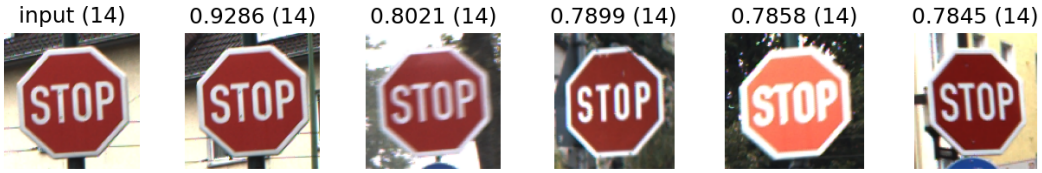

Figure 6.14: Top 5 nearest neighbors of a stop sign – pretrained model.

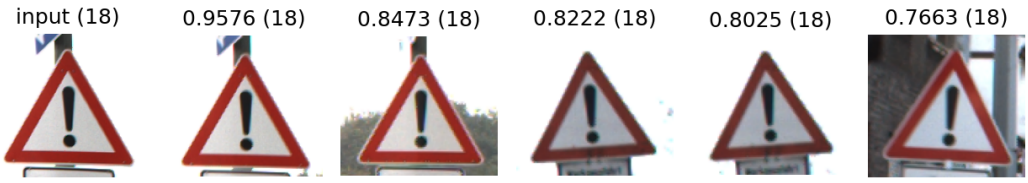

Figure 6.15: Top 5 nearest neighbors of a danger sign – pretrained model.

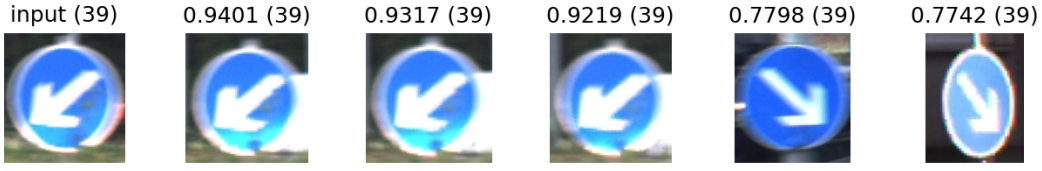

Figure 6.16: Top 5 nearest neighbors of a keep-left sign – pretrained model.

k-NN Evaluation – Post-trained DINO Model

Running the classifier with the frozen weights of the post-trained model with the GTSRB dataset returns significantly better results in comparison with

the previous model. For the top ten and top 20 nearest neighbors, the top-1 prediction was correct in 98.01% of the cases, while for the 100 and 200 nearest neighbors, the model made correct predictions in 97.85% and 97.80% of the cases.

This model produced an average precision of 97.77% and an average recall of 96.37%.

The same set of images and their 5 nearest neighbors can be observed in Figure 6.17 through Figure 6.20 the same set of images and their 5 nearest neighbors. Notably, the similarities are overall higher in comparison with those of the pretrained model. The same observation in Figure 6.20 can be made as was the case in Figure 6.16, where the model retrieved the fourth and fifth most similar images that belonged to a different class, however with a higher confidence level.

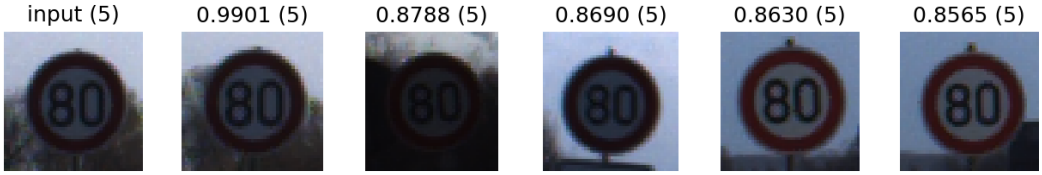

Figure 6.17: Top 5 nearest neighbors of a speed-limit-80 sign – post-trained model.

Figure 6.18: Top 5 nearest neighbors of a stop sign – post-trained model.

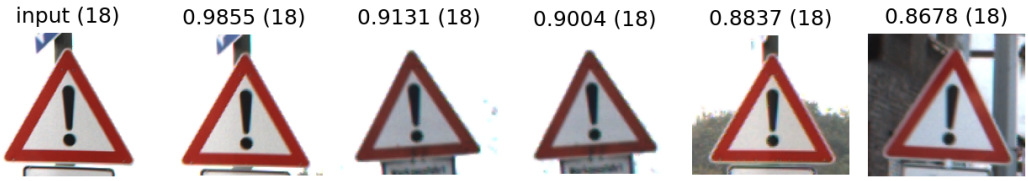

Figure 6.19: Top 5 nearest neighbors of a danger sign – post-trained model.

Figure 6.20: Top 5 nearest neighbors of a keep-left sign – post-trained model.

k-NN Evaluation – From-scratch DINO Model

When using the weights of the model which was trained exclusively with the GTSRB dataset, the classifier produces results that are inferior to those of both prior models. For the ten nearest neighbors, the top-1 prediction was correct 77.30% of the time, while for the 20 nearest neighbors it predicted the correct classes in 76.91% of the cases. Considering the 100 and 200 nearest neighbors, the model provided correct predictions 76.61% of the time in both cases.

This model preformed the worst among the previous models outputting an average precision of 77.19% and an average recall of 72.46%.

This low accuracy compared to the previous models can be seen in the figures. The model is able to retrieve the most similar image that is of the same class in Figure 6.21 with a high accuracy; however, the next four instances are of other classes. Noticeably, the difference between the similarity score of the most similar image and the other four is significant. It is worth mentioning that overall, regardless of the traffic-sign class, the images display very similar characteristics with regards to color, brightness, and elements such as trees in the background.

A similar observation can be made in Figure 6.23, where the model classifies the second nearest neighbor as a danger sign with relatively high accuracy, although it is of a different class, i.e., a no-trucks sign; however, the image has comparable characteristics.

6.2.2 Evaluation on MiG Front-Camera Images

Further analysis has been performed on images from the front camera of the made-in-Germany (MiG) prototype autonomous vehicle at the DCMLR. A small set of images were used as input for the classifier. The images were extracted as frames from video previously recorded while driving the car. The frames in general contain traffic scenery in Berlin.

In order to extract traffic signs from the still frames of wide-angle traffic scenes, the implementation of YOLOv7 [29] as part of a bachelor thesis at the DCMLR [30] was employed. In total, 217 cropped images of traffic signs were selected. Images were split, and 126 images were placed in *train* and 91 images in *val* folders, while trying to prevent very similar images appearing in both folders; this may have arisen due to the fact that most of the traffic signs had been cropped from consecutive frames. Images were then separated into subfolders based on classes corresponding to class IDs, similar to the GTSRB dataset, to retain the classifier's input structure and for uniformity purposes. Sample images that were chosen from the MiG front camera belong to the classes 1, 12, 17, 35, 38 and a miscellaneous class which contained random objects other than traffic signs, e.g. a vehicle wheel and a manhole cover.

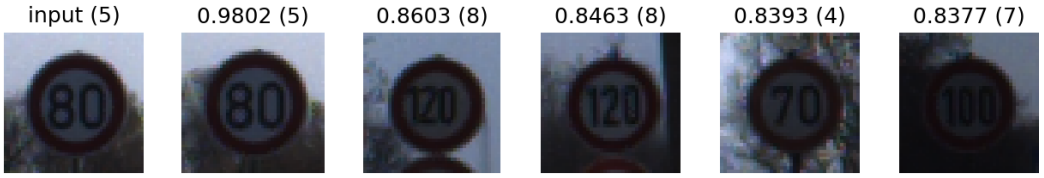

Figure 6.21: Top 5 nearest neighbors of a speed-limit-80 sign – from-scratch model.

Figure 6.22: Top 5 nearest neighbors of a stop sign – from-scratch model.

Figure 6.23: Top 5 nearest neighbors of a danger sign – from-scratch model.

Figure 6.24: Top 5 nearest neighbors of a keep-left sign – from-scratch model.

Attention Visualization: For a few sample images, the attention maps were generated.

With the previously acquired knowledge that the pretrained model on ImageNet1K has, it appears that it struggles to distinguish the features of the traffic signs, as we can see in Figure 6.26, while it can barely focus on the features of the no-entry sign in Figure 6.25.

Using the weights of the post-trained model, it can be noted that more attention is placed on the traffic-sign features overall in the heads. This observation can be made based on Figure 6.27. However, Figure 6.28 demonstrates that the model fails to provide appropriate attention to the traffic-sign characteristics.

Applying the weights of the from-scratch model to generate the attention maps, it can be seen that in most heads the model focuses on the outline of the characters and the shape of the traffic sign more in comparison to the previous

two models. This may be because it had only been trained on such instances of traffic signs. However, in other heads it struggles to provide a logical placement of focus on the features of the traffic sign and focuses either on or around shadows and color discrepancies, as can be seen in head 0 and head 4 in Figure 6.30.

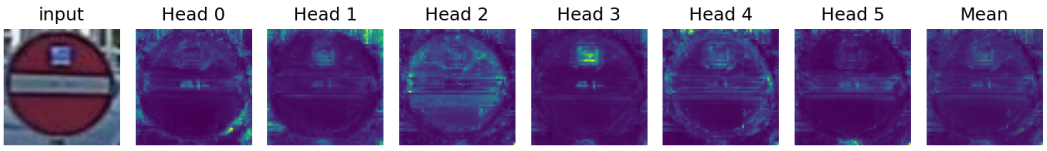

Figure 6.25: Input image, generated attention heads, and a mean representation of a no-entry sign – pretrained model - DCMLR.

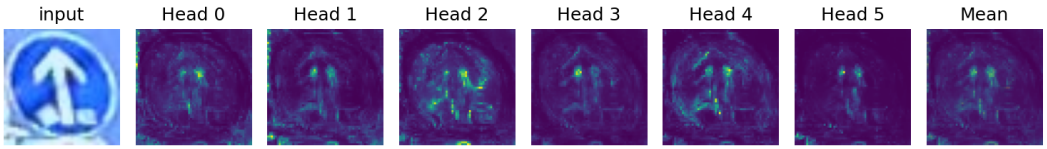

Figure 6.26: Input image, generated attention heads, and a mean representation of a go-straight sign – pretrained model - DCMLR.

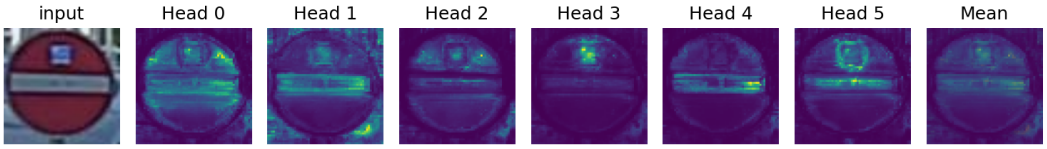

Figure 6.27: Input image, generated attention heads, and a mean representation of a no-entry sign – post-trained model – DCMLR.

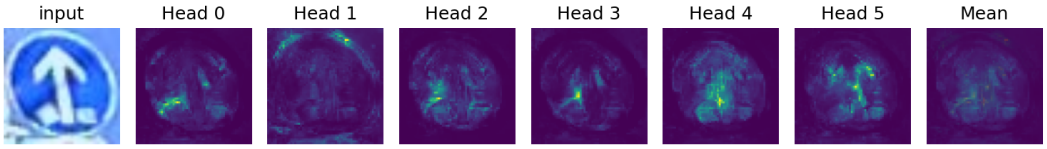

Figure 6.28: Input image, generated attention heads, and a mean representation of a go-straight sign – post-trained model – DCMLR.

k-NN Classification: As described in Section 6.2, features were extracted using the weights of all three models separately, and the classification was executed. For this classification, the five and ten nearest neighbors were inspected, i.e., $k = 5$ and $k = 10$.

Running the classification on the images yielded a top-1 accuracy of 86.81% and 85.71% for 5 and 10 NN respectively, when using the weights of the pretrained model. The post-trained model's weights delivered a top-1 accuracy of 86.81% for 5 NN and 87.91% for 10 NN. The worst performance in classifying these

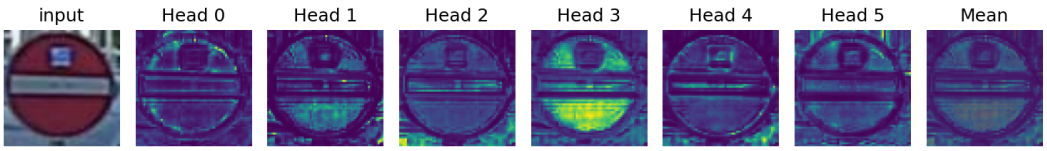

Figure 6.29: Input image, generated attention heads, and a mean representation of a no-entry sign – from-scratch model – DCMLR.

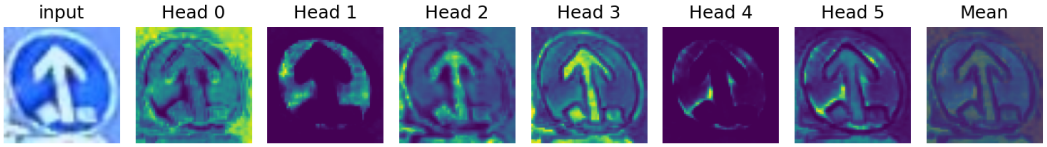

Figure 6.30: Input image, generated attention heads, and a mean representation of a go-straight sign – from-scratch model – DCMLR.

Model	Macro Average Precision	Macro Average Recall
Pre-trained model	88.32%	83.84%
Post-trained model	90.78%	85.89%
From-scratch model	72.41%	63.47%

Table 6.2: Summary of precision and recall values for the different models when classifying the DCMLR test set.

images is observed with the weights of the model trained from scratch, where it achieves a top-1 accuracy of 63.73% for $k = 5$ and 64.83% for $k = 10$.

Considering the precision and recall averages computed on the DCMLR images, a similar trend was observed with regards to which model performed better. Additionally it is noted here too, that the model trained from scratch produced the worst performance achieving an average precision of 72.41% and an average recall of 63.47%. The other two models provided a similar observation to the classification on the GTSRB dataset, where the pretrained model produced an average precision of 88.32% and a recall of 83.84% compared with the slightly better values produced by the post-trained model, scoring an average precision of 90.78% and an average recall of 85.89%. Table 6.2 summarizes the values for the three models.

The retrieved five nearest neighbors of two traffic-sign instances were examined. For all models, the same image was analyzed, together with which top 5 NN that particular model retrieved when running a classification on the images with the weights of the pretrained model. In both examples, this model is able to retrieve the top-5 nearest neighbors which actually belong to the class of the input image which is a priority road sign. In Figure 6.31, it gives the first two images have a high similarity score, which is not surprising given that these images are frames from the same sequence, while the next three have a substantially lower score due to their different backgrounds. The post-trained

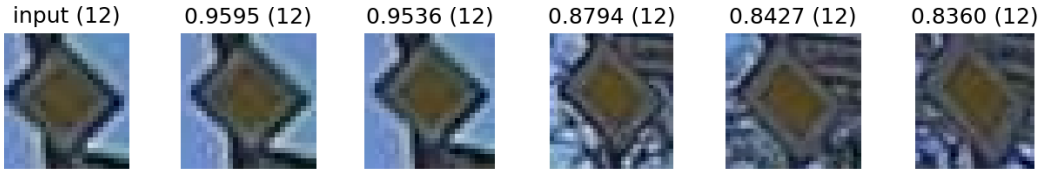

Figure 6.31: Top 5 nearest neighbors of a priority road sign – pretrained model – DCMLR.

Figure 6.32: Top 5 nearest neighbors of a no-entry sign – pretrained model – DCMLR.

model performs similarly in both examples seen in Figure 6.33 and Figure 6.34, both when classifying the instances as well as when scoring them. The from-scratch model provides similar results in the example shown in Figure 6.35. However, it fails to classify the nearest neighbors of the example image in Figure 6.36 correctly, classifying only one image correctly—with a weak similarity in this instance.

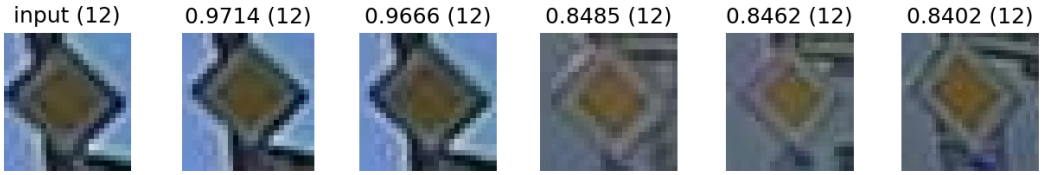

Figure 6.33: Top 5 nearest neighbors of a priority road sign – post-trained model – DCMLR.

Figure 6.34: Top 5 nearest neighbors of a no-entry sign – post-trained model – DCMLR

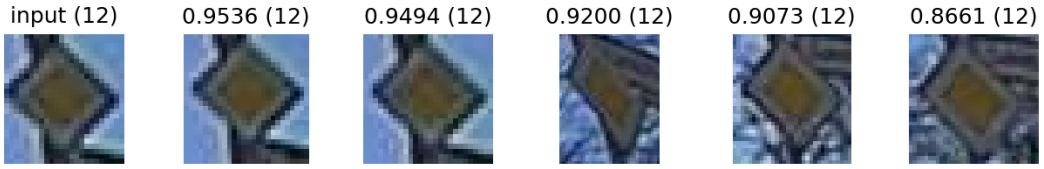

Figure 6.35: Top 5 nearest neighbors of a priority road sign – from-scratch model – DCMLR.

Figure 6.36: Top 5 nearest neighbors of a no-entry sign – from-scratch model – DCMLR

7 Conclusion and Outlook

Transformers have gained popularity in recent years and proved to deliver good results in natural language processing. This led researchers to adapt them for computer vision fields, most notably with the vision transformer from *Kolesnikov et al.* [1].

In this thesis the self-supervised method "self-distillation with no labels" for training a vision transformer was explained. It was evaluated using the German traffic-sign recognition benchmark dataset, as well as a smaller dataset extracted from front-camera images of the Made in Germany autonomous vehicle prototype at the Dahlem Center for Machine Learning. A simple k-NN classifier was used to run classification experiments. Overall, three models were evaluated. The first model was a pretrained model provided by the researchers [22] and was trained on the ImageNet1K dataset containing approximately 1.2 million images. The second model was in essence the first model, but trained further for 50 epochs with the GTSRB training dataset, after some adjustments were made on its structure. The third model was solely trained on the GTSRB dataset for 50 epochs. All models were trained in a self-supervised manner, without the provision of labels or annotations.

Overall, the models were able to provide decent results with regards to the classification of traffic signs. On the GTSRB dataset, we noticed that the second post-trained model performed best with an average precision of 97.77% and an average recall of 96.37%. In comparison the first pretrained model produced an average precision and recall of 88.61% and 84.06% respectively. Notably, the third from-scratch trained model performed worst of all with a precision average of 77.19% and a recall average of 72.46%. This further emphasizes the observation that ViTs rely on large-sized datasets to provide good results, when compared to CNNs. On the DCMLR dataset, the same trend was observed, with post-trained model outperforming the other two, achieving an average precision of 90.78% and recall of 85.89%. The first model produced an average precision and recall of 88.32% and 83.84% respectively and the third output the least values with a 72.41% average precision and a 63.47% average recall over all classes.

For achieving better results, some improvements can be suggested for the future. One aspect that can be improved is the adjustment and fine-tuning of training parameters. During the training of the post-trained model and the from-scratch model, most parameter values from the default trained model were used. These were optimized for the ImageNet1K dataset. Although the

models produced good results, one can look further into fine-tuning the parameters for traffic sign datasets. An additional recommendation is to increase the training time. All models with the exception of the pretrained model were only trained for 50 epochs on medium performance GPUs, where each training process lasted at least two and a half days. Furthermore, a more balanced dataset could equalize precision and accuracy over the different classes.

Bibliography

[1] A. Kolesnikov, A. Dosovitskiy, D. Weissenborn, G. Heigold, J. Uszkoreit, L. Beyer, M. Minderer, M. Dehghani, N. Houlsby, S. Gelly, T. Unterthiner, and X. Zhai, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021.

[2] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *International Conference on Machine Learning*, pp. 10347–10357, PMLR, 2021.

[3] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[4] C. Li, J. Yang, P. Zhang, M. Gao, B. Xiao, X. Dai, L. Yuan, and J. Gao, "Efficient self-supervised vision transformers for representation learning," *International Conference on Learning Representations (ICLR)*, 2022.

[5] F. Moutarde, A. Bargeton, A. Herbin, and L. Chanussot, "Robust on-vehicle real-time visual detection of american and european speed limit signs, with a modular traffic signs recognition system," in *2007 IEEE intelligent vehicles symposium*, pp. 1122–1126, IEEE, 2007.

[6] X. Xu, J. Jin, S. Zhang, L. Zhang, S. Pu, and Z. Chen, "Smart data driven traffic sign detection method based on adaptive color threshold and shape symmetry," *Future generation computer systems*, vol. 94, pp. 381–391, 2019.

[7] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, "Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark," in *International Joint Conference on Neural Networks*, no. 1288, 2013.

[8] H. H. Aghdam, E. J. Heravi, and D. Puig, "A practical approach for detection and classification of traffic signs using convolutional neural networks," *Robotics and autonomous systems*, vol. 84, pp. 97–112, 2016.

[9] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "The German Traffic Sign Recognition Benchmark: A multi-class classification competition," in *IEEE International Joint Conference on Neural Networks*, pp. 1453–1460, 2011.

[10] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow.* " O'Reilly Media, Inc.", 2022.

[11] V. Wittpahl, *Künstliche Intelligenz: Technologien | Anwendung | Gesellschaft*. Springer Berlin Heidelberg, 2018.

[12] T. Wuest, D. Weimer, C. Irgens, and K.-D. Thoben, "Machine learning in manufacturing: advantages, challenges, and applications," *Production & Manufacturing Research*, vol. 4, no. 1, pp. 23–45, 2016.

[13] D. Shah and A. Jha, "Self-supervised learning and its applications." https://neptune.ai/blog/self-supervised-learning, 14.11.2022. last visited on 04.12.2022.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[15] G. Giacaglia, "How transformers work." https://towardsdatascience.com/transformers-141e32e69591, 11.03.2019. last visited on 21.12.2022.

[16] J. Alammar, "The illustrated transformer." http://jalammar.github.io/illustrated-transformer/, 27.06.2018. last visited on 06.12.2022.

[17] S. B. Wali, M. A. Abdullah, M. A. Hannan, A. Hussain, S. A. Samad, P. J. Ker, and M. B. Mansor, "Vision-based traffic sign detection and recognition systems: Current trends and challenges," *Sensors*, vol. 19, no. 9, 2019.

[18] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9650–9660, 2021.

[19] G. Hinton, O. Vinyals, J. Dean, *et al.*, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, vol. 2, no. 7, 2015.

[20] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, "Unsupervised feature learning via non-parametric instance discrimination," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3733–3742, 2018.

[21] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, *et al.*, "Bootstrap your own latent-a new approach to self-supervised learning," *Advances in neural information processing systems*, vol. 33, pp. 21271–21284, 2020.

[22] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, "Unsupervised learning of visual features by contrasting cluster assignments," 2020.

[23] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[24] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[25] I. Loshchilov and F. Hutter, "Fixing weight decay regularization in adam," 2018.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[27] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information processing & management*, vol. 45, no. 4, pp. 427–437, 2009.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[29] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," *arXiv preprint arXiv:2207.02696*, 2022.

[30] B. Suleiman, "Traffic sign detection and classification for autonomous driving," 05.12.2022.

Appendix

Class ID	Traffic sign	Category
0	speed limit 20	prohibitory
1	speed limit 30	prohibitory
2	speed limit 50	prohibitory
3	speed limit 60	prohibitory
4	speed limit 70	prohibitory
5	speed limit 80	prohibitory
6	restriction ends 80	other
7	speed limit 100	prohibitory
8	speed limit 120	prohibitory
9	no overtaking	prohibitory
10	no overtaking (trucks)	prohibitory
11	priority at next intersection	danger
12	priority road	other
13	give way	other
14	stop	other
15	no traffic both ways	prohibitory
16	no trucks	prohibitory
17	no entry	other
18	danger	danger
19	bend left	danger
20	bend right	danger
21	bend	danger
22	uneven road	danger
23	slippery road	danger
24	road narrows	danger
25	construction	danger
26	traffic signal	danger
27	pedestrian crossing	danger
28	school crossing	danger
29	cycles crossing	danger
30	snow	danger
31	animals	danger
32	restriction ends	other
33	go right	mandatory
34	go left	mandatory

Class ID	Traffic sign	Category
35	go straight	mandatory
36	go right or straight	mandatory
37	go left or straight	mandatory
38	keep right	mandatory
39	keep left	mandatory
40	roundabout	mandatory
41	restriction ends (overtaking)	other
42	restriction ends (overtaking (trucks))	other

Table 7.1: Class IDs and corresponding traffic signs as well as the category – GTSRB dataset [9].