Optical-Flow-Guided Pretrained Video Instance Segmentation

Leitung von vortrainierter Videoinstanzsegmentierung durch Optischen Fluss

Master thesis by Melda Ekşi

Date of submission: 28.09.2023

1. Review: Dr. Sc. Simone Schaub-Meyer
2. Review: M.Sc. Dustin Carrión
3. Review: M.Sc. Oliver Hahn
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Computer Science Department

Visual Inference Lab

visual inference

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit erkläre ich, Melda Ekşi, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Stuttgart, 28.09.2023

Melda Ekşi

M. Ekşi

Contents

Abstract

Video instance segmentation (VIS) is seen as a fundamental problem for other video-analysis tasks which generally requires many computational resources. Current research on vision problems present big pretrained foundational models that work well in many downstream tasks. Motivated by this, we exploit such a self-supervised feature extractor in the context of video instance segmentation. In our experiments we shed light on which modifications are necessary to ensure its effectiveness in a VIS setting. Furthermore, we show that fundamental concepts like optical flow can be used to improve the tracking performance or speed up the testing process and propose methods to do so. Although the results show that we cannot compete with current state-of-the-art methods, they show that this is a promising direction of future research.

Die Videoinstanzsegmentierung (VIS) gilt als ein grundlegendes Problem für andere Videoanalyseaufgaben, das im Allgemeinen viele Rechenkapazitäten erfordert. Aktuelle Forschung zu Bildverarbeitungsproblemen präsentiert große vortrainierte Grundlagenmodelle, die gut in vielen nachgelagerten Aufgaben eingesetzt werden können. Aus diesem Grund betrachten wir einen solchen selbstüberwachten Merkmalsextraktor im Zusammenhang mit der Segmentierung von Videoinstanzen. In unseren Experimenten beleuchten wir, welche Modifikationen notwendig sind, damit sie in einem VIS-Problem funktionieren. Darüber hinaus zeigen wir, dass grundlegende Konzepte wie der optische Fluss verwendet werden können, um die Tracking-Leistung zu verbessern oder den Testvorgang zu beschleunigen, und schlagen entsprechende Methoden vor. Obwohl die Ergebnisse zeigen, dass wir nicht mit den aktuellen Spitzenmethoden konkurrieren können, zeigen sie, dass dies eine vielversprechende Richtung für zukünftige Forschung ist.

1 Introduction

Recently, automated driving was authorized in multiple countries [9]. Moreover, approximately 330k hours of video content is uploaded to YouTube daily [7, 1]. Similarly, TikTok accumulates millions of posted videos per day [35]. Thus, moving picture content/videos are almost endless, and the consumption and creation of such media are rising.

This rise creates multiple challenges: How can we make video consumption more accessible? How can we describe and categorize these huge masses of video content? How can we learn from them? And many more. Consequently, it is no surprise that many research areas have emerged from these challenges and opportunities. For instance, action recognition [29] and temporal action segmentation [11] can help in the annotation of video segments. Video retrieval [32] can help in video recommender systems and video captioning [40] can make videos more accessible to visually impaired people. Even if not explicitly targeted at videos, other research areas also have to deal with video data. Automated driving uses visual cues for safety and functional purposes [15]. Augmented reality must also deal with changing environments [14].

While many methods exist for extracting information from images, moving pictures provide more visual and semantic cues that can benefit many tasks. In essence, it all boils down to successfully representing video data well and extracting valuable information from it because, as an example, all beforementioned problems could benefit from extracted video-level object masks. Extracting these masks is the primary goal of video instance segmentation (VIS), where individual object detection, segmentation, and tracking are performed simultaneously [42].

Current models either perform well in accuracy but require lots of computational resources or are resource-efficient but less performant. Therefore, we want to tackle this problem by leveraging big pretrained foundational models for efficiency and taking advantage of their favorable properties. However, these pretrained foundational models require modifications to work satisfactorily.

The contributions of this work are as follows:

- We shed light on which design choices are necessary to use valuable foundational models for the VIS problem and introduce new methods with these findings.
- We introduce optical flow as guidance in the testing procedure to speed up inference and achieve higher accuracies.
- We show that methods with significantly less computational demands are realizable when pretrained foundational models are utilized.

We will begin by giving an overview about the theoretical background and the current lines of research of the VIS problem in Section 2.1. Furthermore we present the foundational models and concepts that our method is based on. Chapter 3 describes the methods that are developed in this thesis. We first describe our self-supervised vision transformer backbone in Section 3.1. Afterwards, in Section 3.2 we present two different concepts for how optical flow can be used as guidance in the testing procedure. In Chapter 4, we conduct extensive experiments on the design choices and the performance of our DINO backbones (Section 4.4.1), as well as our guidance with optical flow (Section 4.4.2). Lastly, we summarize and discuss our findings and give an outlook on possible future work in Chapter 5.

2 Related Work

In this chapter, we formally introduce the video instance segmentation task and summarize the research directions of the problem. Furthermore, we describe architectures that will be further discussed in later chapters.

2.1 Video Instance Segmentation

Video instance segmentation (VIS) [42] is a problem introduced in 2019 as an extension to related problems like image instance segmentation and video object detection and segmentation. In VIS, we not only label instances in video frames but also segment each instance and assign them an instance ID. In other words, we want to determine to what class the object belongs to, where it lies, and its instance identification. It is worth clarifying that in VIS, objects with the same class should be separately detected (segmented and labeled) and tracked along the video.

More formally, as visualized in Figure 2.1, given an input video with T frames with a total of I object instances, we get the following output for each instance hypothesis j:

- Class label: $\widetilde{c}_j$
- Segmentation map per frame: $\widetilde{m}^j_{\widetilde{p}...\widetilde{q}}$ with $\widetilde{p}$ and $\widetilde{q}$ referring to the beginning end ending frame of the hypothesis, respectively
- Instance ID: j
- Bounding box per frame: $\widetilde{b}^j_{\widetilde{p}...\widetilde{q}}$
- Confidence score: $\widetilde{s}_j$

Alongside the VIS problem definition, MaskTrack R-CNN [42] was proposed as the first solution, which we will describe in detail in Section 2.2. Nowadays, VIS research can be categorized into two lines of research: offline and online/real-time VIS.

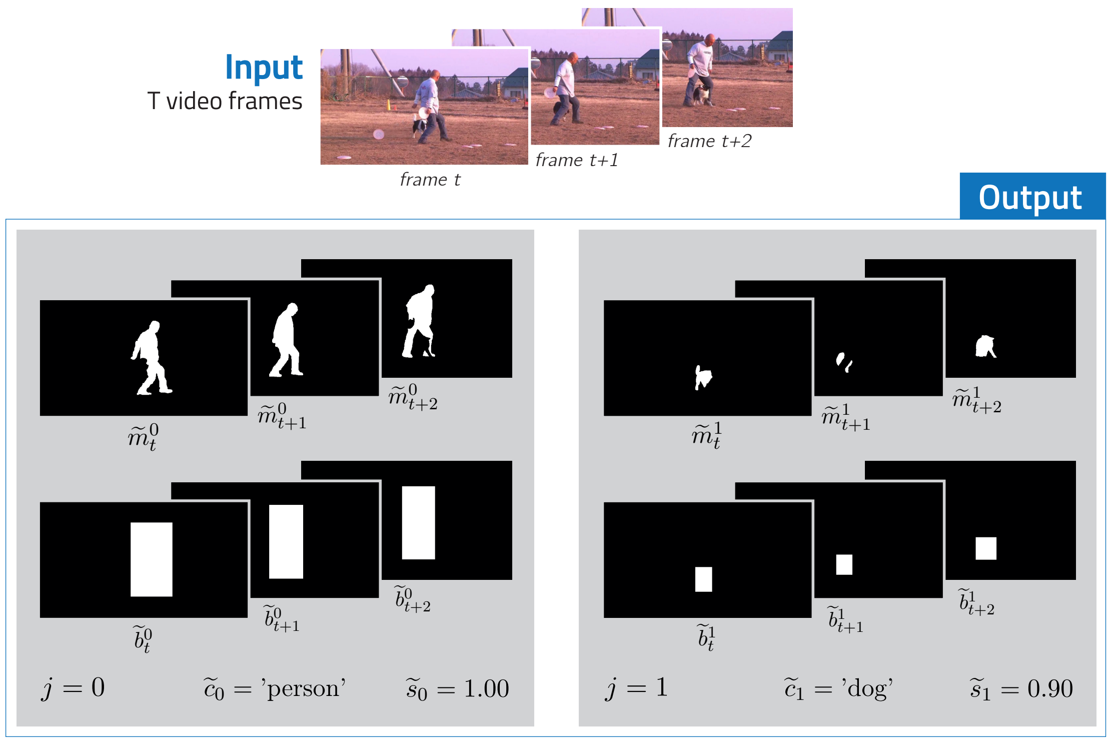

Figure 2.1: Visualization of the video instance segmentation problem. Video frames are fed into the algorithm. As an output, we receive instance hypotheses (grey boxes) with predicted segmentation masks and bounding boxes, class label and detection confidence.

Solutions based on vision transformers are offline, and currently, they are the state-of-the-art methods for VIS. In this type of solution, frames of a whole video are input into a vision transformer, which treats it as a sequence decoding/prediction problem. VisTR [37] was the first work to propose using vision transformers for VIS. Then, SeqFormer [38] was built upon VisTR by modifying the model to attend to different areas in different frames corresponding to the same instance. A meaningful video-level instance embedding is calculated by weighting the feature embeddings extracted from the frame-level instance bounding boxes. Although transformer-based methods are SOTA, they have one main disadvantage: they are very computationally expensive because we have to feed whole video sequences into the models. Consequently, these methods are limited because they can only deal with short or low-resolution videos.

The second main line of research deals with the opposite problem: how to make VIS more computationally efficient. One real-time solution was proposed with the InstanceFlow [23] method. While popular Mask-R-CNN-based methods always rely on a first stage with bounding box detection, InstanceFlow calculates a geometric center for each instance mask and tracks the instances by calculating the flow between these mask centers. Another work [43] tries to make video instance segmentation faster by omitting the calculation of segmentation masks for each frame. Instead, they propagate the calculated segmentation mask iteratively with a dense optical flow prediction for in-between frames. The problem with online solutions is that they cannot compete with offline methods in accuracy. Consequently, VIS methods always have a tradeoff between accuracy and inference speed.

2.2 MaskTrack R-CNN

Mask R-CNN [18] is a widely used foundational semantic segmentation model built on Faster R-CNN [31]. It consists of three branches, *i.e.*, classification, bounding box regression, and binary segmentation. The architecture consists of two stages; the first is for proposing regions of interest, which are then used in the branches of the second stage.

The complete procedure of Mask R-CNN is as follows: a backbone generates a feature map from the input image. This feature map is then fed into a region proposal network (RPN), which proposes bounding boxes for regions of interest in the feature space. From these bounding box regions of interest (RoIs), features are then extracted from the feature embedding with RoIAlign and processed in task-specific branches. These were classification and bounding box detection for Faster R-CNN [31], which was extended for image segmentation with a binary segmentation branch later in Mask R-CNN.

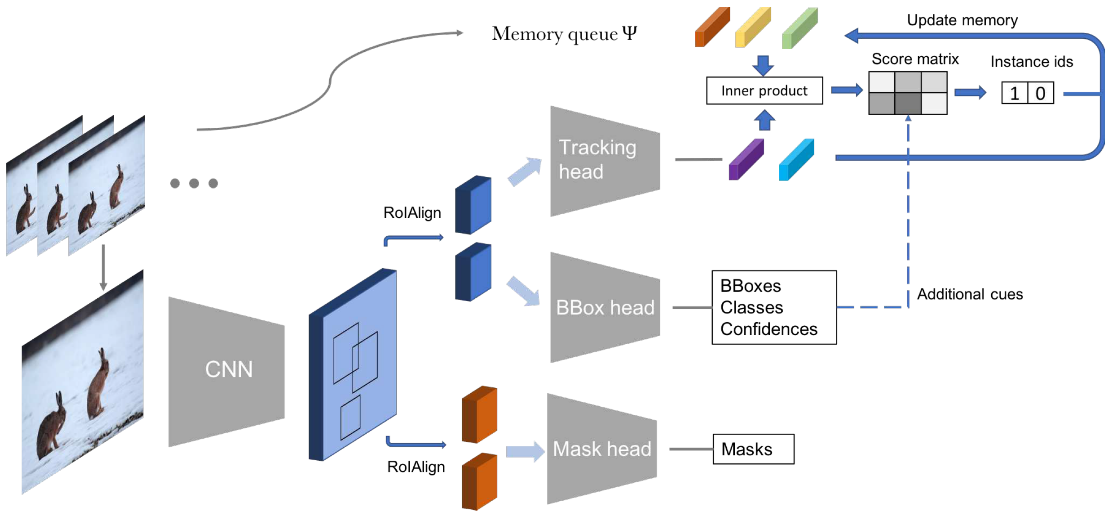

Figure 2.2: Architecture of MaskTrack R-CNN, copied from [42].

As previously explained, MaskTrack R-CNN [42] was introduced alongside the VIS problem definition. It essentially extends Mask R-CNN [18] with a tracking branch to allow the tracking of object instances across frames. Figure 2.2 shows the architecture of the model.

With the introduction of the tracking task, the training objective is now a minimization of a joint loss function:

$$L = L_{cls} + L_{box} + L_{mask} + L_{track}. \tag{2.1}$$

During training, one randomly sampled frame is input to the architecture as the query frame and one additional random frame is used as a reference frame. Reference frames provide tracking features extracted from ground-truth regions. These features are computed by extracting the feature embeddings of regions of interest via RoIAlign and feeding them into two fully connected layers. Then, query frames match their candidate bounding boxes, that have enough bounding box overlap with detected instance regions, with instance IDs.

During training, the tracking branch computes similarity between instance candidates and already detected instances which are stored in an external memory by computing the dot product of the candidates and detected instances stored tracking features.

Formally, the instance identification problem is treated as a multiclass classification problem. Here we have $N+1$ ‘classes,’ which actually refer to instances. The label assignment probability is calculated with

$$p_i(n) = \begin{cases} \frac{e^{f_i^T f_n}}{1+\sum_{j=1}^{N} e^{f_i^T f_n}} & n \in [1, N] \\ \frac{1}{1+\sum_{j=1}^{N} e^{f_i^T f_n}} & n = 0 \end{cases} \tag{2.2}$$

If instances were not seen before, their tracking features will be saved in the external memory directly. If we see the same instance in later frames again, the tracking feature of this instance will be overwritten with its newer tracking feature embedding.

The tracking loss L_{track} with y_i as the ground truth instance label is is calculated with $L_{track} = -\sum log(p_i(y_i))$.

The testing procedure differs from the training one since it not only relies on the similar appearance of the features but also tries to ensure that labels, masks, and bounding boxes between instance candidates and saved features align well. This is computed with

$$v_i(n) = \log p_i(n) + \alpha \log s_i + \beta IoU(b_i, b_n) + \gamma\delta(c_i, c_n) \tag{2.3}$$

with α, β, and γ being hyperparameters that regulate the influence of the different cues.

The final instance label is the majority vote of instance labels, and the confidence scores are calculated via an averaging of detection scores.

2.3 Vision Transformers

Transformers were introduced in the NLP domain first, including highly popular architectures like BERT [10] or different versions of GPT [30, 4], which the omnipresent tool ChatGPT is based on, and adapted to vision problems afterward because of their good performance.

While natural language can be fed in a sequence encoding/decoding architecture relatively straight-forward, we need to rethink the process for visual inputs. Vision transformers [12] generally take an input image and separate them into patches of a specific size, *i.e.*, 16×16 pixel patches. These patches are then transformed into sequence patch embeddings, for example, via a linear projection, and the sequence is then wholly fed as input to the transformer model. However, positional information of the patches must be explicitly modeled to preserve spatial information, which is often done by adding positional encodings to the patch embeddings, as otherwise, this information would be lost.

The effectiveness of the transformer architecture comes from its multi-head self-attention mechanism [36], where attention is calculated for every query-token pair. This means that for a patch in the input image (query), attention is calculated for all other image patches (keys). In this context, attention refers to the relationship of a patch with another patch and is computed via a similarity measure like dot-product. Afterward, these are normalized with softmax to obtain scores. A single patch representation will, therefore, be a weighted sum of the representations of all other patches, thus encoding global information [24]. Multi-head attention refers to using multiple attention heads to model different relationships of the input, with each head having its own weight matrices.

Formally, attention $Z_i \in \mathbb{R}^{n \times d_v}$ of a single head is calculated as follows [36, 24]

$$Z_i = softmax\left(\frac{Q_i K_i^T}{\sqrt{d_q}}\right) V_i \tag{2.4}$$

where Q_i (Queries), K_i (Keys) and V_i (Values) denote the input sequence $X = (x_1, x_2, ..., x_n) \in \mathbb{R}^{n \times d}$ projected onto different weight matrices W_i^Q, W_i^K, W_i^V, *i.e.*,

$$\begin{aligned} Q_i &= XW_i^Q \\ K_i &= XW_i^K \\ V_i &= XW_i^V \end{aligned}$$

with $W_i^Q \in \mathbb{R}^{d \times d_q}$, $W_i^K \in \mathbb{R}^{d \times d_k}$, $W_i^V \in \mathbb{R}^{d \times d_v}$, $d_q = d_k$ and $i \in \{0, \dots, (h-1)\}$ for h attention heads.

The attention outputs of the different heads are then concatenated to $Z = [Z_0, Z_1, \dots, Z_{h-1}] \in \mathbb{R}^{n \times h \cdot d_v}$. We introduce an additional weight matrix $W^O \in \mathbb{R}^{h \cdot d_v \times d}$ that the attention gets projected on to obtain the multi-head attention output

$$M = ZW^O \tag{2.5}$$

A vision transformer [12] includes such multi-head attention modules in its encoder blocks together with normalization, MLP, and residual connections. These encoder blocks are stacked to form a powerful transformer encoder. An extra learnable token is added to the input sequence that would otherwise only consist of combined positional and patch embeddings. The whole architecture of a vanilla vision transformer is depicted in Figure 2.3.

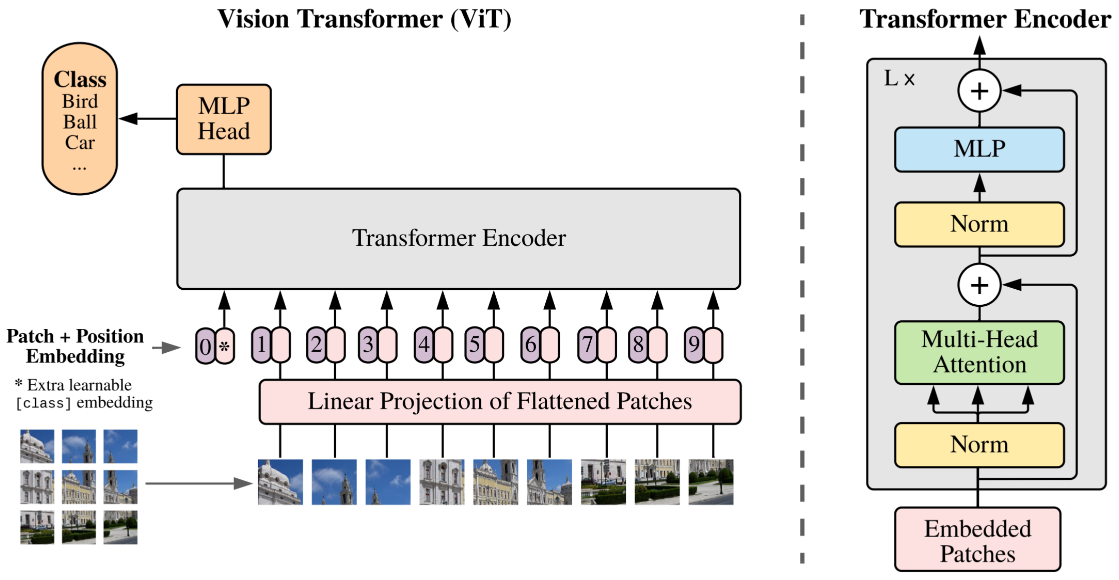

Figure 2.3: Architecture of a vanilla vision transformer, copied from [12].

Although transformers are shown to work well, they have three main limitations [24]:

- As the whole input is fed into the encoder as a sequence, we cannot deal with high-resolution image inputs.
- The attention mechanism is very costly.
- Vision transformers need to be trained on very large datasets to perform well.

These limitations make them slow during inference and training and require a lot of computational resources.

Self-distillation with no labels One challenge of vision problems is that getting annotated datasets is hard and costly. To overcome such problems, many works have used self-supervision paradigms to train architectures without using annotations. In vision problems, such self-supervised methods are mainly contrastive learning methods, *i.e.,* similar images should be moved together and dissimilar images further apart, which is often achieved by treating an augmented version of the input image as a positive sample and other images as negative samples [26]. Popular contrastive methods are the variants of MoCo [17, 8]. However, there are also popular non-contrastive methods like BYOL [16] or SWAV [5] that only use positive samples. Regardless of the approach, the main idea of all methods is to learn a good representation of the data that can benefit downstream tasks and compensate for the need for large, labeled datasets.

As described earlier, transformer architectures generally need to be pretrained on large datasets to work well. Thus, they could benefit from self-supervision. For this reason, in self-distillation with no labels [6], commonly abbreviated as DINO, the authors analyze what kind of properties self-supervised vision transformer features convey. For that, they propose a self-supervised procedure inspired by knowledge distillation [19]. We refer to the original DINO paper for the exact differentiation of the method to regular knowledge distillation [6].

Two identical vision transformers with projection heads, referred to as student g_{Θ_s} and teacher g_{Θ_t}, are introduced. These have different weights, while the teacher weights are computed as a moving average of student networks from past iterations with centering and sharpening afterwards to prevent mode collapse.

Self-supervision is introduced by multi-crop [5]. Here an input image gets differently cropped multiple times: 2 ‘global’ crops with higher resolution and several ‘local’ crops, which have smaller resolutions. The teacher architecture receives the global crops as input, while the student architecture receives all crops.

The learning target is minimizing the difference between the two architectures distributions

$$\min_{\Theta_s} \sum_{x \in x_1^g, x_2^g} \sum_{\substack{x' \in V \\ x' \neq x}} -P_t(x) \log P_s(x') \tag{2.6}$$

where the distributions over K dimensions are calculated with

$$P_s(x)^{(i)} = \frac{\exp(g_{\Theta_s}(x)^{(i)}/\tau_s)}{\sum_{k=1}^{K} \exp(g_{\Theta_s}(x)^{(k)}/\tau_s)}, \tau_s > 0 \tag{2.7}$$

The same formula applies to the teachers distribution P_t, but with τ_t and g_{Θ_t}.

DINO is trained on ImageNet without labels and with more augmentations in addition to multi-crop and is trained with 8×8 or 16×16 patches on differently sized architectures that align with ViT-small, ViT-base, and ViT-large [12]. A result of this work is that self-supervised vision transformers encode information about the semantic segmentation of images although they were not trained for this task, which is not inherent in other self-supervised methods or supervised methods. The authors also show that their pretrained, fixed DINO features can be used as meaningful feature embeddings for downstream tasks like classification or semantic segmentation.

2.4 Optical Flow

Optical flow estimation is a well-explored area of research that originates in deep-learning-free methods and denotes the apparent motion between images [20]. Early research on optical flow reaches back to the Horn and Schunck method [20], where the authors assume that the representation of a pixel stays constant between frames, which is called the ‘brightness constancy assumption’. Lucas and Kanade add the assumption that pixels of the same neighborhood move in the same direction [27]. These assumptions are modeled into an energy minimization problem.

Later on, FlowNet [13] introduced the first deep-learning-based architecture where convolutional neural networks learn outputting optical flow fields in a supervised fashion. Additionally, the authors introduced the synthetically generated FlyingChairs dataset because previous datasets were comparably small, and CNNs rely on having a sufficient amount of training data to perform well. However, extensions were necessary because the models could not generalize well to real data and could not beat the performance of traditional methods. For this reason, FlowNet2.0 [21] stacks multiple FlowNet architectures to estimate large and small motions in an iterative fashion.

Traditional methods for optical flow estimation are still important for newer methods that are based on deep learning. For instance, PWC-Net [33] uses three of such important concepts in optical flow estimation: image pyramids, warping, and cost volume calculation. These are combined by using learnable feature pyramids. Large motion is estimated with a warping layer and cost volumes are computed in another layer. Contrary to previous methods, PWC-Net works in real-time (35 FPS) and can be trained end-to-end.

Optical flow computation is generally computationally expensive regardless of the method because it is a 2D search problem where large 4D cost volumes are computed [41]. This is especially a problem when dealing with high-resolution images where calculating such a cost volume is often infeasible. To overcome this problem, Flow1D [41] treats the 2D problem as two separate 1D problems. Here, the 2D search space is approximated into two 1D directions via 1D attention and 1D correlation in orthogonal directions. This architecture can calculate flow for input images with resolutions up to 8K, contrary to PWC-Net, which can only run on small resolutions.

3 Methodology

This chapter provides a conceptual overview of the methods that were developed for this work. First, we present a new backbone that leverages self-supervised DINO features. In addition, we propose methods for using optical flow to guide the testing process. Implementation details are discussed in Chapter 4.

3.1 Pretrained Video Instance Segmentation

Vision transformers, trained in a self-supervised fashion as proposed in DINO [42], were shown to be able to provide meaningful feature embeddings for downstream tasks like classification and semantic segmentation. Considering the resemblence of these downstream tasks to video instance segmentation, we suggest integrating DINO as a feature extractor in video instance segmentation architectures.

We want to leverage foundational methods to solve VIS, hence our method is a variation of the MaskTrack R-CNN architecture. Furthermore, our method is based on the findings of [24]. In this work, vision transformers were used as backbones in Mask R-CNN architectures. However, the authors introduce multiple changes in the Mask R-CNN modules and training procedures. Although other self-supervised methods are featured in this work, DINO vision transformers are also not addressed.

Because we want to show the effectiveness of DINO backbones for video instance segmentation, we want to keep other components of MaskTrack R-CNN as-is as much as possible. MaskTrack R-CNN generally uses ResNet backbones as feature extractors, which are then fed into a feature pyramid network. ResNet backbones are fundamentally different to vision transformers, because they provide multi-scale features. Contrary to that, the scale and embedding dimension of vision transformer features stay constant throughout. We

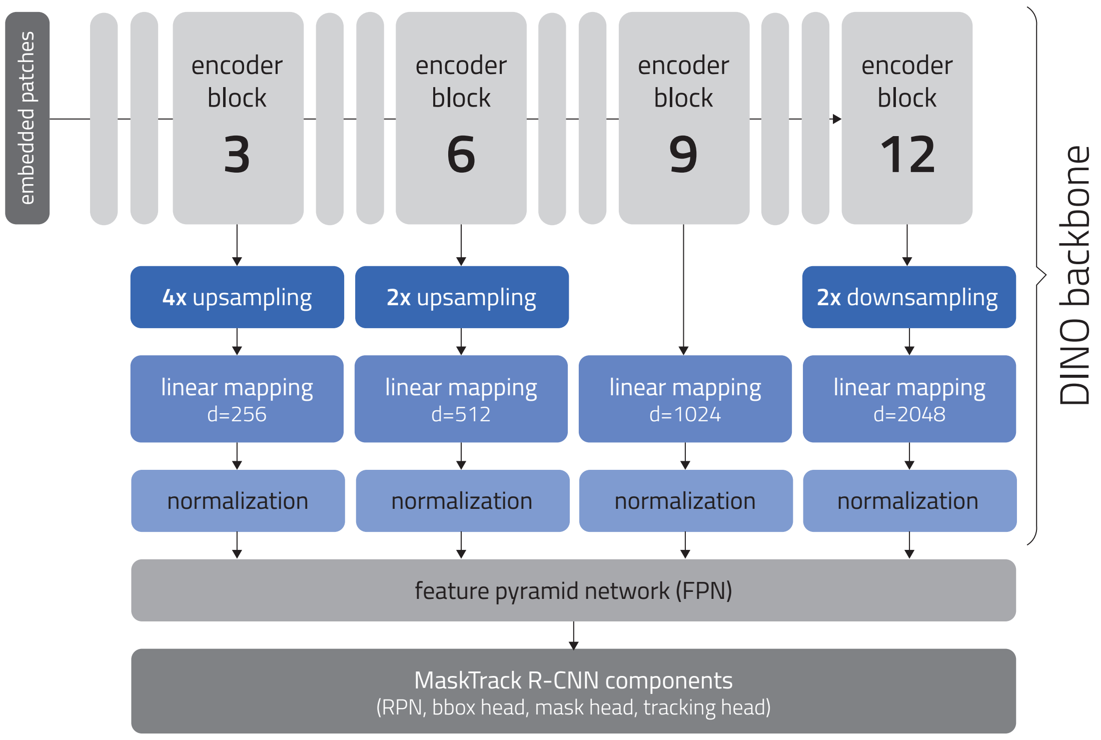

Figure 3.1: Visualization of the DINO MaskTrack R-CNN architecture, adapted from [24]. Features are extracted from every third encoder block of a DINO vision transformer, on which we perform resampling, linear mapping and normalization. This forms the DINO backbone.

therefore propose a method to imitate the outputs of a ResNet to adhere to the rest of the MaskTrack R-CNN architecture with minimal changes.

First, we extract features after every third DINO encoder block. Afterwards, these block outputs are resampled as in [24] to obtain multi-scale DINO features. The third block output will be upsampled with a factor of 4. This upsampling is performed via 2x2 transposed convolutions with a stride of 2, a GeLU non-linearity and another 2x2 transposed convolution. Outputs of block six are upsampled with a factor of 2, which is again performed with a 2x2 transposed convolution with a stride of 2. We keep the feature output scale of block nine as-is and downsample the output of block twelve with 2x2 max pooling with a stride of 2. With every ResNet block, the scale of a feature embedding

would be halved, while the embedding dimensionality is doubled. Vision transformers provide feature embeddings where the embedding dimension is constant. To match the ResNet behavior, we use 1x1 convolutions to map to the ResNet-50 embedding dimensions 256, 512, 1024 and 2048.
Lastly, because we have shown empirically that feature normalization is necessary, all feature embeddings are normalized with group normalization with the default group size of 32. A depiction of the explained architecture is shown in Figure 3.1.

All these operations in combination form our basic working DINO backbone for MaskTrack R-CNN. We will refer to the whole architecture as 'DINO MaskTrack R-CNN'. Alternate design choices will be explored in Chapter 4.

3.2 Optical-Flow-Guided Video Instance Segmentation

We introduce optical flow in video instance segmentation to achieve two goals: on one hand, the testing procedure will be sped up, while model performance will be improved in terms of accuracy on the other hand.

If we know where an object instance is located in one frame and also know the movement between frames, we can calculate where it moved to. This idea can be exploited to achieve the first goal. We modify the tracking branch of MaskTrack R-CNN by using flow propagation as an additional cue in the testing procedure. This idea is based on [43], where segmentation masks computed with Mask R-CNN are propagated into later frames. However, that work dealt with the problem of semantic instance segmentation and therefore reported performance on the Cityscapes dataset among others.
More specifically, we start by computing segmentation masks, bounding boxes, prediction confidence and class labels with the Mask R-CNN portion of MaskTrack R-CNN for the first frame and repeat this procedure for every frame in a specified interval. For every intermittent frame, we compute its dense optical flow between the previous frame and propagate all hypotheses of the previous frame into the current one. This is performed by a forward warping of the hypothesis segmentation masks with the computed dense optical flows. Additionally, we recompute the bounding boxes and copy the previously computed prediction confidence, class and instance labels. We will refer to this method as 'FlowProp'. Figure 3.2 shows this setup when the prediction frequency is 2. FlowProp will certainly lead to a degradation in segmentation performance, but speed up the testing process.

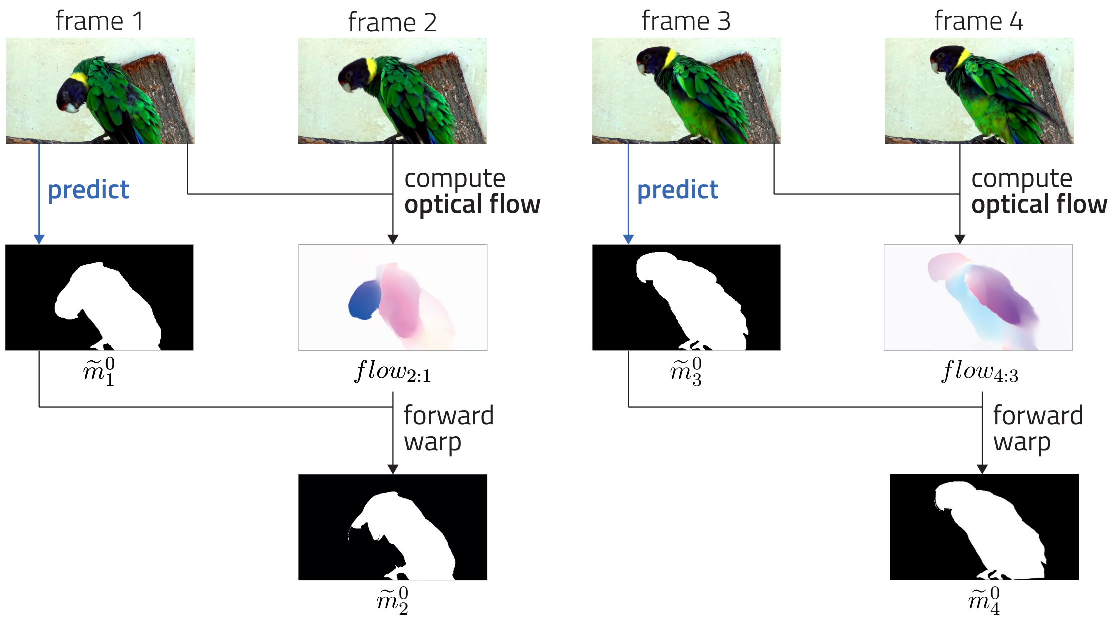

Figure 3.2: Visualization of the FlowProp method. We start by predicting instances with regular MaskTrack R-CNN for the first frame. For every second frame, instance masks from the previous frame are forward warped with the optical flow between the two frames.

Instead of speeding up VIS, we can also use optical flow to improve the tracking performance, as it can be a suitable cue for instance detection. We will output predictions with the Mask R-CNN parts of MaskTrack R-CNN for every frame to obtain accurate segmentation masks. Tracking is supported by using the optical flow between the current frame and the previous frame. With this, we forward warp the segmentation masks of the previous frames hypotheses and calculate the overlap between the warped segmentation mask and the predicted ones. If the mask overlap is high, the objects should be of the same instance. We extend the regular MaskTrack R-CNN tracker with this additional cue

$$\begin{aligned} v_i(n) = \log p_i(n) + \alpha \log s_i + \beta IoU(b_i, b_n) + \gamma\delta(c_i, c_n) \\ + \varepsilon \cdot sIoU(m_i, flow_{n:i}(m_n)) \end{aligned} \tag{3.1}$$

where $flow_{n:i}$ denotes the optical flow between the current frame and the previous frame and m_i and m_n represent the segmentation masks of the current and previous frame respectfully, while ε controls the influence of the cue.

We adapt the soft IoU calculation from [3] over the mask pixels p, which is:

$$sIoU(m_i, m_n) = \frac{\sum_p m_i(p)m_n(p)}{\sum_p m_i(p) + m_n(p) - m_i(p)m_n(p)} \tag{3.2}$$

We refer to this optical flow concept as ‘MapMatch’ and visualize it in Figure 3.3.

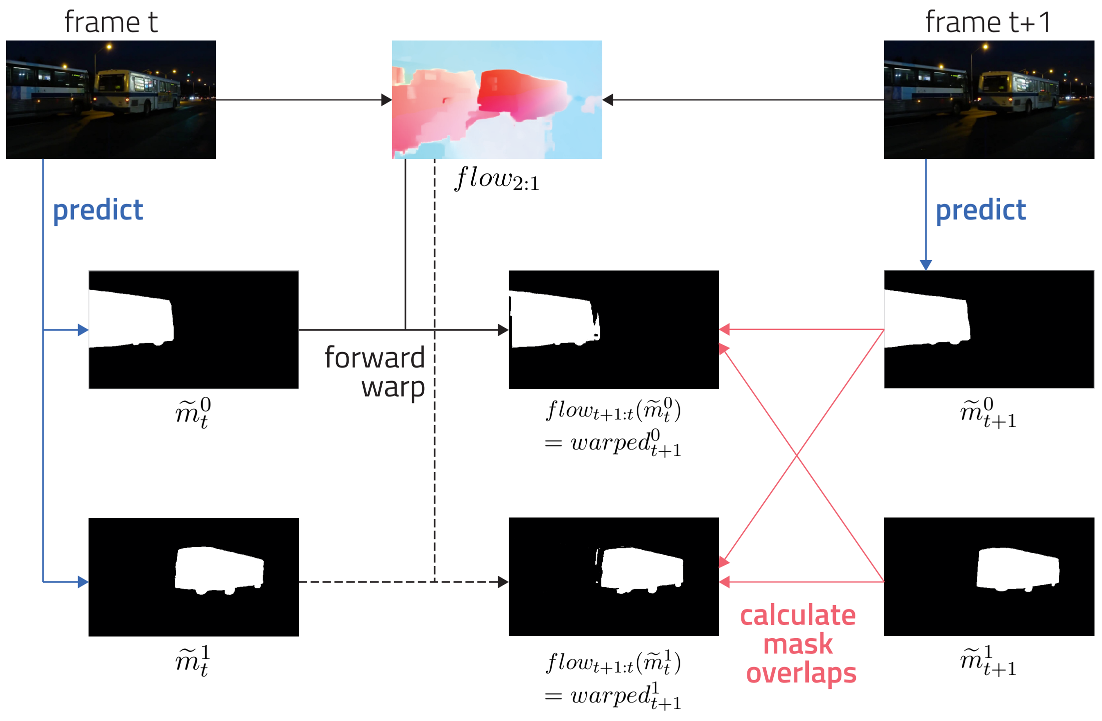

Figure 3.3: Visualization of the MapMatch method. Masks of previous instance detections are forward warped to the next frame with optical flow. Mask overlaps between all warped masks and currently predicted masks are calculated in terms of soft IoU. These scores act as an additional cue that should benefit tracking in the testing procedure.

4 Experiments

4.1 Implementation

We base our implementation on the MMTracking toolbox [28], which is a part of the OpenMMLab project and designed explicitly for video perception problems. This toolbox was chosen because it includes the MaskTrack R-CNN [42] baseline implementation, the YouTube-VIS datasets [42] and evaluation methods, unlike the popular Detectron2 framework [39].

We extend the framework with a custom DINO backbone that is based on the official DINO vision transformer implementation. Dense optical flow is computed with the high-resolution pretraining of the original implementation of Flow1D [41], where the extracted flow map dimensions match the original video resolutions. During evaluation, predictions will be resampled to the original frame dimensions to calculate the metrics. We perform forward warping in our optical-flow-guided methods with full resolution flow maps.

Training Setup We mostly follow the original training setup of the official MaskTrack R-CNN implementation [42]. The methods are trained with a batch size of 8 for 12 epochs. The initial learning rate is 0.00125 with a linear warmup for 500 iterations and a warm-up ratio of $\frac{1}{3}$. Learning rate decay is performed with a factor of 10 at epochs 8 and 11. Furthermore, we use the original hyperparameter choices of MaskTrack R-CNN, which are $\alpha = 1$, $\beta = 2$ and $\gamma = 10$. During training and validation, we downsample the original input frames to 640×352 pixels. The input dimensions are divisible by 16, which means that the entire image can be divided into 8×8 or 16×16 pixel patches. Otherwise, vision transformers would disregard the remainders of the image. Additionally, no padding is performed on the inputs. All experiments were performed on a NVIDIA RTX A6000 (49140MiB) GPU.

4.2 Datasets

This thesis makes use of the YouTube-VIS 2019 [42] dataset. It is based on the YouTube-VOS dataset, but it is labeled exhaustively and extended with instance information to adhere to the video instance segmentation problem.

The 2019 version has 40 classes and consists of approximately $2\,900$ high-resolution videos, of which $2\,238$ belong to the training split and 302 videos comprise the validation dataset. The test dataset consists of 343 videos. Videos are object- or scene-centric and can feature (multiple) animate objects like animals or people, as well as inanimate objects such as skateboards or trains. The datasets consist of video frames that are extracted in an interval of five frames in a frame rate of 30fps. Figure 4.1 shows some examples.

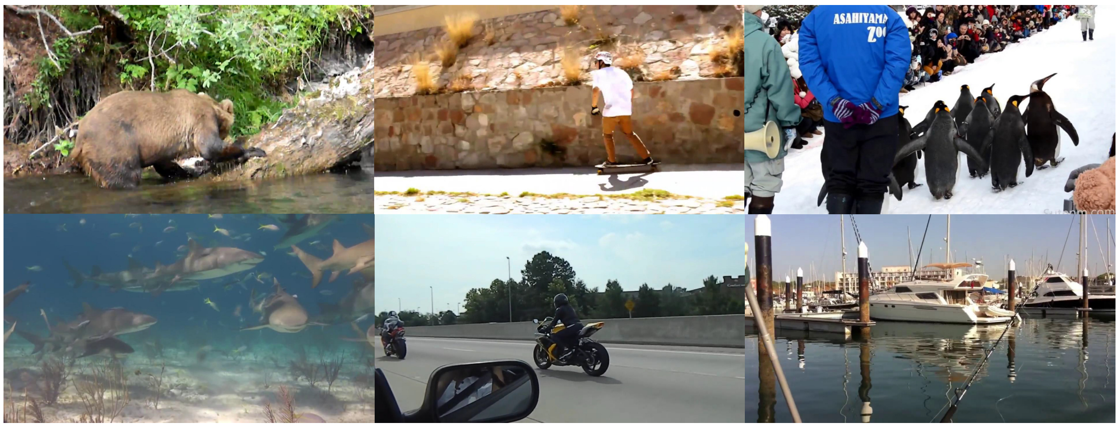

Figure 4.1: Frames extracted from videos of the YouTube-VIS 2019 dataset [42].

In total $130k$ object masks from 4800 objects are provided. Ground-truth annotations are only provided for the training split and hidden for the other splits. Metrics are therefore calculated on a Codalab server.

Although newer versions of the dataset exist, most works still report on the 2019 version since it is already an established baseline for the VIS problem. Due to the extensive computational needs of VIS models, we will, therefore, also focus our experiments to this version. The original paper reports on the validation and test dataset, but metric calculation on the test set is no longer possible. Hence we will, just like in other works, report our results on the validation set only.

4.3 Metrics

We evaluate our methods on the standard VIS metrics described in more detail below. Nevertheless, as validation annotations are not provided, performance metrics on the validation dataset are calculated by uploading the predictions to the Codalab server[1].

Average Precision Average precision (AP) is a widely used metric in image instance segmentation and denotes the area under the precision-recall curve. The precision-recall curve is calculated over varying detection confidence thresholds. The area under the curve is computed over 101 recall thresholds, which are 0 to 100% with increments of 1%. The original COCO metric [25] calculates AP over 10 IoU thresholds between 50% and 95% in 5% increments by averaging the AP scores obtained from the single thresholds.

In video instance segmentation, it is especially important to model and assess the relationship of the instance representation across frames. A performance metric for VIS must, therefore, be able to represent such relations. IoU in image instance segmentation would be normally computed for single images. However, just extracting frames from videos and treating them as separate entities would defeat the purpose. The standard AP metric must be, therefore, modified, which was done by redefining the IoU computation as [42]:

$$IoU(i,j) = \frac{\sum_{t=1}^{T} |m_t^i \cap \widetilde{m}_t^j|}{\sum_{t=1}^{T} |m_t^i \cup \widetilde{m}_t^j|} \tag{4.1}$$

In other words, IoU is computed as the overlap between the real instance segmentation masks $m_{p\ldots q}^i$ and those of the hypotheses $\widetilde{m}_{\widetilde{p}\ldots\widetilde{q}}^j$, summed and averaged over all T frames of a video. As explained in [42], if the object masks are detected successfully, but the model fails to track the objects across frames, it will get a low IoU.

The above calculation is limited to 100 hypotheses per video that have the highest detection confidence. Like in regular COCO evaluation, AP is first computed per category and then averaged over all categories. In addition to **AP**, two additional variations are calculated: **AP50** and **AP75**. These variations report the average precision over single IoU thresholds, which are 0.5 and 0.75, respectively.

[1] https://codalab.lisn.upsaclay.fr/competitions/6064

Average Recall Contrary to the AP calculation, average recall (AR) is defined as the maximum recall we can achieve over a fixed amount of detection hypotheses. While all AP metrics are calculated over 100 hypotheses per video, AR is reported over two different fixed numbers of hypotheses. **AR1** uses only one highest-scoring hypothesis per video, while **AR10** examines 10 detections. Apart from that, the IoU and recall thresholds are the same as for AP, and the AR variants are computed per category first and then averaged afterward again.

4.4 Experiments

This chapter describes different experiments and their results in detail. In Section 4.4.1 we will explore different design choices for DINO-based MaskTrack R-CNN backbones. Section 4.4.2 will show the experiments for our proposed optical-flow-based tracking concepts, which were described in Section 3.1 and 3.2.

4.4.1 DINO MaskTrack R-CNN

In Chapter 3 we introduce our initial DINO-based backbone for MaskTrack R-CNN by imitating the architecture of the original ResNet-50 backbone. These initial working baselines are created by defining small datasets consisting of a single video or five videos of the original training split and ensuring that the model is able to overfit on both of them. However, the initial design choices might not be best-suited and could be improved upon. Futhermore, most design choices introduce more learnable parameters and increase model complexity. Thus, we will explore which modifications are necessary to be able to use DINO backbones in MaskTrack R-CNN and which additional design choices are beneficial for the model performance.

The results of all experiments will be reported over two random seeds. Preferably one would report over a bigger amount of random seeds, but this is not feasible because of the computational overhead. All models were trained on the training split of the YouTube-VIS 2019 dataset. The AP and AR metrics, which are described in Section 4.3, are reported on the YouTube-VIS 2019 validation split unless stated otherwise.

DINO variants The authors of DINO provide different vision transformers that were trained with their self-supervised training setup, which is explained in Section 2.3. These pretrained backbones are either using 8×8 or 16×16 pixel patches and have different embedding dimensions, which are 384 for the small models and 768 in the base models. We refer to the small models with ViT-S and to the base models as ViT-B.

We will test out the different pretrained backbone variants to see which are the best performing ones in conjunction with a MaskTrack R-CNN architecture. For this experiments, we train the models on the whole training dataset, as the difference of architectures becomes only apparent with a usage of bigger amounts of training data. When using smaller splits, the results become more similar, which could be also caused by the not fully deterministic training procedure. While the MaskTrack R-CNN baseline initializes the Mask R-CNN parts with pretrained weights, we compare the DINO architectures performance to training from scratch. The reasoning for this is the mismatch of dimensions for some of the components, where the pretrained weights cannot be loaded fully. Also the 16×16 patch variants might benefit from the pretraining more because it maps the input image into the same sizes as the original ResNet backbone would. To ensure comparability even further, we freeze the original ResNet backbone in our experiments. The MaskTrack R-CNN baseline only froze the first stage of the backbone, which amounts to 225,344 of the 23,508,032 parameters. This corresponds to about 0.96% of the total backbone parameters. Consequently, the ResNet-50 backbone is mostly trainable in the baseline setup. Furthermore, we use the original initialization of the ResNet when MaskTrack R-CNN is trained from scratch. We present our results on the YouTube-VIS 2019 validation split in Table 4.1.

The results show that the baseline model with a frozen backbone is not able to perform as well as any of the models that have a DINO vision transformer feature extractor, causing a loss of AP between approx. 2 and 6% compared to the DINO variants. ViT-S/16 performs 1.33% worse than ViT-S/8 in terms of AP, while ViT-B/16 performs 3.16% worse than ViT-B/8. The architectures therefore benefit from smaller patch sizes, probably due to their higher patch granularity, which should lead to more detailed instance segmentations. ViT-B/16 has 1.43% more AP than ViT-S/16, while the improvements in AP50 is noteably large (+3.16%) and the improvement in AP75 only minor (+0.4%). In terms of AR1 and AR10, ViT-B/16 performs better than ViT-B/16 (+1.06% AR1 and 1.99% AR10). However, when looking at the 8×8 patch variants, ViT-B/8 performs 3.26%, 3.22% and 5.07% better in terms of AP, AP50 and AP75 respectively than ViT-S/8. AR1 and AR10 are also better (+3.12% and +3.39%). We can therefore see that bigger embedding dimensions of DINO vision transformers leads to a better performance than their smaller counterparts. This is probably because of their higher amount of parameters and model complexity,

Method	Arch.	AP	AP50	AP75	AR1	AR10
MaskTrack R-CNN	ResNet-50	0.2481	0.4579	0.2534	0.2722	0.3299
Frozen MaskTrack R-CNN	ResNet-50	0.1448	0.3276	0.1218	0.1744	0.2044
DINO MaskTrack R-CNN	ViT-S/16	0.1632 $\pm$0.0039	0.3394 $\pm$0.0124	0.1349 $\pm$0.0007	0.2143 $\pm$0.0059	0.2509 $\pm$0.0132
DINO MaskTrack R-CNN	ViT-B/16	<u>0.1775</u> $\pm$0.0017	<u>0.3710</u> $\pm$0.0135	0.1389 $\pm$0.0060	<u>0.2249</u> $\pm$0.0018	<u>0.2708</u> $\pm$0.0015
DINO MaskTrack R-CNN	ViT-S/8	0.1765 $\pm$0.0062	0.3663 $\pm$0.0222	<u>0.1424</u> $\pm$0.0093	0.2173 $\pm$0.0152	0.2631 $\pm$0.0087
DINO MaskTrack R-CNN	ViT-B/8	**0.2091** $\pm$0.0069	**0.3985** $\pm$0.0231	**0.1931** $\pm$0.0001	**0.2485** $\pm$0.0043	**0.2970** $\pm$0.0061

Table 4.1: MaskTrack R-CNN results for different DINO ViT configurations and baseline models for comparison on the YouTube-VIS 2019 validation dataset. All backbone feature extractors are initialized with pretrained weights, while the rest of the components are initialized randomly. Best scores of DINO MaskTrack R-CNN methods are formatted in **bold**, while second-best scores are <u>underlined</u>.

especially when we keep in mind that the embedding dimensionality will be mapped to up to 2048 in the backbone later. At a further glance on the results, ViT-B/16 and ViT-S/8 perform almost similarly well (higher AP, AP50, AR1 and AR10 for ViT-B/16 and higher AP75 for ViT-S/8). Increasing the embedding dimensionality and decreasing the patch size lead to similar improvement. To conclude, both an increase in the embedding dimensionality and decrease of the patch sizes in conjunction cause a significant increase in performance, making VitB-8 the best performing variant. Nevertheless, the original MaskTrack R-CNN model still performs 4 to 8% AP better than our proposed MaskTrack R-CNN derivates with frozen DINO backbones. Compared to the 10% performance loss in AP that fully freezing the original MaskTrack R-CNN backbone causes, the experiments hence validate the effectiveness of the self-supervised DINO vision transformer features for downstream tasks, which are not apparent in the supervised features of the ResNet.

All of the following experiments will be conducted on the MaskTrack R-CNN architecture that has a DINO ViT-S/8 backbone (later abbreviated as DINO-MT-S/8), because it has a similar amount of backbone parameters as ResNet-50, performs better than ViT-S/16, which has the same amount of feature extractor parameters. This will give us the performance gain of using 8×8 patches, but not the negative effect of longer training times that would come with ViT-B/8.

Updown vs. no Resampling Different block outputs of the ResNet provide feature embeddings in different sizes as a result of the convolutional architecture, which are then used in a Feature Pyramid Network. We therefore want to see if feature maps in different resolutions are beneficial for a MaskTrack R-CNN pipeline when using DINO backbones or if resampling can be omitted. Different vision transformer blocks normally always have the same size, contrary to a ResNet backbone. We therefore conduct an experiment where we omit the upsampling/downsampling of the features and use the extracted DINO features in their original resolutions. The strides are therefore modified to 16 everywhere. The results are presented in Table 4.2.

Method	AP	AP50	AP75	AR1	AR10
DINO-MT-S/8	<u>0.1775</u> ±0.0017	<u>0.3710</u> ±0.0135	<u>0.1389</u> ±0.0060	<u>0.2249</u> ±0.0018	<u>0.2708</u> ±0.0015
DINO-MT-S/8 without resampling	0.0272 ±0.0070	0.0935 ±0.0150	0.0064 ±0.0038	0.0382 ±0.0060	0.0387 ±0.0067
DINO-MT-S/8 with bilinear interpolation	**0.1847** ±0.0204	**0.3896** ±0.0299	**0.1572** ±0.0349	**0.2343** ±0.0135	**0.2783** ±0.0174

Table 4.2: Results of DINO ViT-S/8 MaskTrack R-CNN for different resampling configurations. Best scores are formatted in **bold**, while second-best scores are <u>underlined</u>.

Table 4.2 shows that the model performs poorly when resampling is omitted. It causes a loss of 15% in AP, which leads to an AP of 2.72% which is extremely bad. 3.4% is both the score for AR1 and AR10. The model is therefore not able to make even a single confident prediction, diminishing the usefulness of the backbone. Resampling therefore is a necessity for DINO backbones in Mask R-CNN architectures. This is probably caused by the architecture of feature pyramid networks, where the features are inputted into.

As we have seen, resampling is a necessary component for making Mask R-CNN architectures work well with DINO backbones. Resampling the feature embeddings via upconvolutions or downconvolutions introduce more learnable parameters for the backbone. We will test if interpolation, which does not introduce any more trainable parameters to the backbone, suffices for resampling the feature embeddings to different scales. We use bilinear interpolation to upsample and downsample the feature embeddings of the different block outputs. The output embeddings will have the same size as they had with the convolution-based resampling method. Table 4.2 shows that bilinear interpolation is indeed a sufficient replacement for the convolution-based resampling and even performs slightly better. We achieve a 0.72% increase in AP, 1.86% increase in AP50 and 1.83% increase in AP75. The AR metrics also show an improvement of 0.94 and 0.75% in AR1 and AR10 respectively. The standard deviation of the results however show noteably more variance in model performance when using bilinear interpolation compared to the initial version with up- and downconvolutions. A resampling like in **??** is obviously suitable for vision-transformer-based backbones, but in case of self-supervised feature extractors that provide valuable features themselves already, might not the best choice. We expect them to work better in regular supervised settings where the backbone will be trained fully as well. Concluding, resampling is anecessary step to make MaskTrack R-CNN work with DINO feature extractors.

No mapping The outputs of ResNet blocks are differently sized because when downsampling the feature dimension, the embedding dimension is doubled. Vision transformers however have a constant embedding dimension. We test if a mapping to the ResNet embedding dimensions is necessary.

Method	**AP**	**AP50**	**AP75**	**AR1**	**AR10**
DINO-MT-S/8	**0.1775** ±0.0017	**0.3710** ±0.0135	**0.1389** ±0.0060	**0.2249** ±0.0018	**0.2708** ±0.0015
DINO-MT-S/8 without linear mapping	0.1620 ±0.0030	0.3359 ±0.0001	0.1365 ±0.0129	0.2052 ±0.0033	0.2549 ±0.0017

Table 4.3: Results of DINO ViT-S/8 MaskTrack R-CNN with and without linear mapping. Best scores are formatted in **bold**, while second-best scores are underlined.

Table 4.3 shows that ViT-S/8 with linear mapping has a higher AP with +1.55% and higher AP50 with +3.51%. AP75 is however only slightly better in the version with linear mapping (+0.25%), while the difference in AR1 and AR10 are again larger (1.97% and 1.59%). Thus MaskTrack R-CNN based archictectures with DINO feature extrators can work without a linear mapping, but possibly perform better when used because we have introduced more learnable parameters.

Block selection Initially, we defined our DINO backbone in a way that used the outputs of every third encoder output of the vision transformer as done in [24]. In this experiment we test if the last four block outputs provide more meaningful features for VIS. Table 4.4 shows the results of the block selection experiments. We see that the version using the last four block outputs performs worse than the original version that uses the DINO output of every third block. With every encoder block the vision transformer will produce more semantically meaningful features. However, we suspect that the worse performance is caused by the diminishing difference between the representations when using the last encoder blocks. This adds less value to the different outputs and thus to a lower performance. We also think that the design choices in [24] are carefully chosen and modifications will not lead to much betterment.

Method	AP	AP50	AP75	AR1	AR10
DINO-MT-S/8	**0.1775** ±0.0017	**0.3710** ±0.0135	**0.1389** ±0.0060	**0.2249** ±0.0018	**0.2708** ±0.0015
DINO-MT-S/8 with last 4 block outputs	0.1554 ±0.0125	0.3337 ±0.0155	0.1261 ±0.0193	0.2116 ±0.0087	0.2563 ±0.0123

Table 4.4: Results of DINO ViT-S/8 MaskTrack R-CNN for varying encoder block selections. Best scores are formatted in **bold**, while second-best scores are underlined.

Normalization Normalization of features is a crucial step for deep learning architectures, which leads to the representations of input data to have similar ranges, because otherwise learning would be harder. We try different popular normalization options of the DINO block outputs and present the results in Table 4.5.

Introducing no normalization prevents the model from learning and the model loses the ability to make any confident predictions as a consequence. Because the experiment on the small dataset consisting of five training videos already did not work, and the ability of the model, that was trained on the whole training split, to make predictions diminished completely, we do not report any metrics for this experiment. Batch normalization [22] barely allows the model to localize the instances in terms of bounding box and class detection, but fails in segmenting the instances, which causes poor AP and AR performance. Batch normalization is highly dependent on the batch size and generally works better with rising batch sizes. Video data, especially if it is high-resolution, cannot be processed in large batch sizes due to their high computing needs, so batch normalization performs second worse. Instance normalization [34] actually allows the model to segment instances

Method	**Normalization**	**AP**	**AP50**	**AP75**	**AR1**	**AR10**
DINO-MT-S/8	GroupNorm 32	**0.1775** ±0.0017	0.3710 ±0.0135	0.1389 ±0.0060	**0.2249** ±0.0018	**0.2708** ±0.0015
DINO-MT-S/8	GroupNorm 16	0.1715 ±0.0073	**0.3712** ±0.0157	**0.1462** ±0.0208	0.2113 ±0.0057	0.2555 ±0.0091
DINO-MT-S/8	GroupNorm 8	0.1728 ±0.0042	0.3677 ±0.0038	0.1437 ±0.0071	0.2091 ±0.0018	0.2575 ±0.0065
DINO-MT-S/8	LayerNorm	0.1715 ±0.0074	0.3570 ±0.0030	0.1437 ±0.0200	0.2122 ±0.0040	0.2543 ±0.0022
DINO-MT-S/8	InstanceNorm	0.1178 ±0.0053	0.2602 ±0.0098	0.0980 ±0.0036	0.1653 ±0.0134	0.2043 ±0.0193
DINO-MT-S/8	BatchNorm	0.0022 ±0.0000	0.0092 ±0.0001	0.0000 ±0.0001	0.0106 ±0.0002	0.0109 ±0.0002

Table 4.5: Results of DINO ViT-S/8 MaskTrack R-CNN for different normalization methods on the YouTube-VIS 2019 validation dataset. Best scores are formatted in **bold**, while second-best scores are underlined.

and performs better than BatchNorm. Layer normalization [2] allows the model to actually learn valuable features that can be used for the segmentation of instances. Group normalization in all variants is performing best and the different group size selections do not lead to large differences in performance, which aligns with the original paper. We can therefore conclude that feature normalization is absolutely necessary when using DINO features in Mask R-CNN based archictures as well and group normalization is best suited in this case. We show that the more granular normalization is, i.e. normalization is performed over smaller areas, the better the results are.

Input Scale Videos in high-resolution cannot be processed easily due to their need for extensive computational resources. In Mask Track R-CNN input images are downscaled to half of high-resolution, i.e. 640×360 pixels, such that a computation of those features is possible in sufficient batch sizes. Smaller input resolutions would speed up training and inference and it is therefore worth exploring how much the model performance suffers when downscaling the inputs even further. Table 4.6 shows the results of our input resolution experiment.

We have tried to test out higher input resolutions as well, but computing time rises exponentially, so training a model with 960×536 pixels would have taken over 20 days and was therefore infeasible. Training with an input dimension of 800×440 pixels was feasible, but the model lost the ability to make any confident predictions along proceeding training epochs.

Method	Input dimensions	AP	AP50	AP75	AR1	AR10
DINO-MT-S/8	640×352	0.1775 ±0.0017	0.3710 ±0.0135	0.1389 ±0.0060	0.2249 ±0.0018	0.2708 ±0.0015
DINO-MT-S/8	480×264	<u>0.1987</u> ±0.0061	<u>0.4011</u> ±0.0039	**0.1830** ±0.0203	**0.2504** ±0.0033	**0.3015** ±0.0013
DINO-MT-S/8	320×176	0.1926 ±0.0092	0.3985 ±0.0090	0.1520 ±0.0165	0.2361 ±0.0037	0.2797 ±0.0018
DINO-MT-S/8	224×120	**0.2082** ±0.0074	**0.4152** ±0.0030	<u>0.1827</u> ±0.0200	<u>0.2484</u> ±0.0040	<u>0.2903</u> ±0.0022

Table 4.6: Results of DINO ViT-S/8 MaskTrack R-CNN for varying input dimension, where the same input dimension is used during training and validation, on the YouTube-VIS 2019 dataset. Best scores are formatted in **bold**, while second-best scores are <u>underlined</u>.

The experiment results show that the model benefits from smaller input dimensions. The model with an input dimension of 224×120 pixels performs best (+3.07% AP), while scale 0.375 also leads to an improvement of approximately 2 percent in AP, AR1 and AR10 compared to scale 0.5. The performance difference for AP50 and AP75 is bigger with approx. 4%. Scale 0.25 still performs better than the original input size of MaskTrack R-CNN, but worse than scale 0.375. Originally, vision transformers and DINO were trained with 224×224 pixel inputs, which would explain the performance gain when using similar input dimensions. For the rest, smaller input dimensions will lead to larger neighborhoods in the patches, which can benefit the detection performance and decrease the segmentation performance. Scale 0.375 and 0.25 perform similarly well. Scale 0.25 surpasses scale 0.375 in AP and AP50 by 0.95% and 1.41%, while falling behind minimally in AP75 and AR. Due to the benefit of faster training and inference times and smaller model size, scale 0.25 should be the preferred input scale. Table 4.7 shows that the performance improvement comes mostly from the change of input dimensions in training. When only lowering the input dimensions during validation, the versions with scale 0.375 and 0.25 perform much worse.

Method	Input dimensions	AP	AP50	AP75	AR1	AR10
DINO-MT-S/8	640×352	<u>0.1775</u> ±0.0017	<u>0.3710</u> ±0.0135	<u>0.1389</u> ±0.0060	<u>0.2249</u> ±0.0018	**0.2708** ±0.0015
DINO-MT-S/8	480×264	**0.1817** ±0.0026	**0.3864** ±0.0043	**0.1521** ±0.0060	**0.2269** ±0.0020	<u>0.2700</u> ±0.0029
DINO-MT-S/8	320×176	0.1249 ±0.0011	0.3078 ±0.0110	0.0866 ±0.0247	0.1766 ±0.0014	0.2040 ±0.0042
DINO-MT-S/8	224×120	0.0659 ±0.0016	0.1935 ±0.0013	0.0252 ±0.0102	0.0986 ±0.0009	0.1091 ±0.0001

Table 4.7: Results of DINO ViT-S/8 MaskTrack R-CNN for varying input dimension during validation on the YouTube-VIS 2019 dataset. During training, the input dimension is 640×352 pixels in all methods. Best scores are formatted in **bold**, while second-best scores are <u>underlined</u>.

Pretraining As the original architecture relies on a pretrained Mask R-CNN and [24] also claims that pretraining always leads to faster convergence, we want to explore how finetuning on pretrained weights effects the performance. Table 4.8 shows the results when finetuning pretrained weights and without. Finetuning is shown to be beneficial for Mask R-CNN-based architectures with DINO backbones as well, even though the pretrained weights are calculated with architectures without DINO backbones and not all weights are applicable due to dimension mismatchs in the backbones It causes an increase of almost 4% in AP, 5% in AP50 and 6% in AP75. AR1 is also improved by 3.59% and AR10 by 4.1%. The model performance is therefore significanty increased through pretraining. These results are in-line with recent research in general and especially with the findings in [24].

4.4.2 FlowProp and MapMatch

The MaskTrack R-CNN tracking branch favors videos with small motion and instances with similar appearance due to its design of the testing procedure. However, not every instance will only have small displacements between frames and they can also change their appearance severely. To counteract this problem, we propose using optical flow as guidance in the testing procedure. As described in Chapter 3, we call this method 'MapMatch'.

Method	AP	AP50	AP75	AR1	AR10
DINO-MT-S/8	0.1775 ±0.0017	0.3710 ±0.0135	0.1389 ±0.0060	0.2249 ±0.0018	0.2708 ±0.0015
DINO-MT-S/8 with pretraining	**0.2151** ±0.0096	**0.4226** ±0.0196	**0.2006** ±0.0063	**0.2608** ±0.0032	**0.3118** ±0.0015

Table 4.8: Results of DINO ViT-S/8 MaskTrack R-CNN with and without pretrained Mask R-CNN components besides the backbone. The backbones are initialized with DINO weights for both. Best scores are formatted in **bold**, while second-best scores are underlined.

The other shortcoming of MaskTrack R-CNN is its long inference time. Thus, as presented in **??**, optical flow can be used to speed up the model during testing time, which we will call 'Flow Propagation'. Flow propagation will most certainly lead to a decrease in prediction accuracy due to non-perfect flow computations. Additionally, if objects disappear and reappear later, it will not be able to re-detect the instance. However, speedups can be substantial, so we want to see how much performance loss is to be expected when using the proposed method.

Oracle flow We compute an upper-bound for the model performance in an oracle experiment. For that, the annotations provide us with ground-truth segmentations, instance labels and bounding boxes. These will be used instead of the predictions of the other MaskTrack R-CNN branches. Unfortunately validation annotations are not provided and we will therefore perform the oracle experiment on a random split of the training data. This split consists of 302 videos, just as the validation split.

The best-achieveable model performance is calculated with the ground-truth annotations. To measure the influence of the flow procedure, we configure the model to calculate Mask R-CNN predictions only on every n-th frame. For the remaining frames we will copy the predictions of the previous frame and warp them with optical flow to obtain predictions for the current frame. The more frames skip calculating predictions with Mask R-CNN, the worse the accuracy should get. In order to validate the effectiveness of introducing flow in this propagation procedure, we omit segmentation warping with flow by just copying the predictions of the previous frame for frames where we do not calculate the predictions with Mask R-CNN. Figure 4.2 shows the results of the procedure with varying prediction frequencies on the beforementioned training split.

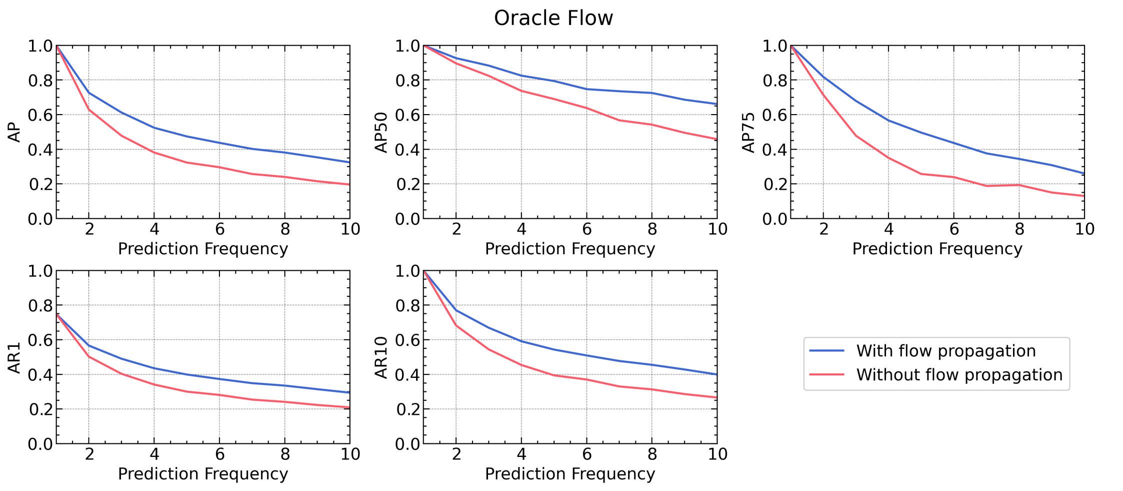

Figure 4.2: Results of the oracle flow experiment on a training data split. Flow propagation improves the performance by up to 20 percent.

We observe that the bigger the prediction frequency, the worse the model accuracy. However, introducing flow indeed causes a better-performing model. The architecture with flow propagation performs up to 20% better than the version without flow propagation. When looking at the AR1 metric, we observe that it could not reach 100 percent, even though we use ground-truth annotations. This is not an error, but in fact can be traced back to the definition of the metric. AR1 cannot be 1 if there is more than one instance in a video. If there are more than 10 instances in a video, AR10 will not be 1 either.

FlowProp Returning to FlowProp, now that we have computed the upper performance bound with the oracle experiment, we want to see the influence on real predictive models. Just as in Section 4.4.1, we will use our baseline MaskTrack R-CNN with a DINO ViT-S/8 vision transformer backbone. We first report the performance on the training split that was used in the oracle experiment. Afterwards, we report on the validation dataset. The results on the training split, as presented in Table 4.9 are naturally lower than in our oracle experiment, but better than when calculating metrics on the validation set, because the model cannot generalize well to unseen data.

Table 4.10 shows the results of FlowProp on the validation dataset. When only making predictions on every second frame and using optical flow to warp the predictions of the previous frame on every intermittend frame, we only lose 2.15% AP. AP50 and AP75

Method	Prediction Frequency	AP	AP50	AP75	AR1	AR10
DINO-MT-S/8	1	**0.4050** ±0.0057	**0.6900** ±0.0170	**0.4175** ±0.0304	**0.3630** ±0.0085	**0.4825** ±0.0078
DINO-MT-S/8 FlowProp	2	<u>0.3555</u> ±0.0007	<u>0.6630</u> ±0.0085	<u>0.3385</u> ±0.0035	<u>0.3200</u> ±0.0071	<u>0.4290</u> ±0.0042
DINO-MT-S/8 FlowProp	3	0.3315 ±0.0007	0.6465 ±0.0021	0.3115 ±0.0120	0.3000 ±0.0028	0.3990 ±0.0014
DINO-MT-S/8 FlowProp	4	0.3065 ±0.0092	0.6135 ±0.0120	0.2565 ±0.0049	0.2770 ±0.0057	0.3715 ±0.0007
DINO-MT-S/8 FlowProp	5	0.2850 ±0.0071	0.5955 ±0.0120	0.2440 ±0.0141	0.2625 ±0.0049	0.3475 ±0.0021
DINO-MT-S/8 FlowProp	6	0.2765 ±0.0049	0.5865 ±0.0064	0.2335 ±0.0064	0.2585 ±0.0078	0.3365 ±0.0007

Table 4.9: Results of DINO ViT-S/8 MaskTrack FlowProp for varying prediction frequencies on a training split of the YouTube-VIS 2019 dataset. Best scores are formatted in **bold**, while second-best scores are <u>underlined</u>.

suffers by 1.42 and 2.41%. We also lose 2.38% AR1 and 3.31% in AR10. This performance loss was to be expected. With rising prediction frequencies, performance differences get smaller and smaller. When we compare prediction frequency 6 and the one with frequency 1, we only lose approximately 5-6% in every metric besides AR10. AR10 suffers the most with 7.65%. If inference speed is only a slight concern, it is advisable to omit predictions on every second frame. But if speed is the main issue, bigger prediction frequencies can be of good use. This will however come with a decrease in accuracy.

MapMatch An improvement in accuracy might be achieved with the MapMatch flow method that is described in Chapter 3. In this method, we add an optical-flow-based cue in the testing phase to the existing cues of MaskTrack R-CNN. The influence of this cue is regulated with the hyperparameter ε. Because a whole hyperparameter search is infeasible, we base our experiment on the original hyperparameters that were chosen in MaskTrack R-CNN and combine them with our own MapMatch addition. The original hyperparameters were 1 for the detection confidence cue, 2 for the IoU cue and 10 for the label consistency cue. We explore the best hyperparameter choice for ε in the following experiment. Again,

Method	Prediction Frequency	AP	AP50	AP75	AR1	AR10
DINO-MT-S/8	1	**0.1775** ±0.0017	**0.3710** ±0.0135	**0.1389** ±0.0060	**0.2249** ±0.0018	**0.2708** ±0.0015
DINO-MT-S/8 FlowProp	2	<u>0.1560</u> ±0.0006	<u>0.3568</u> ±0.0255	<u>0.1148</u> ±0.0114	<u>0.2011</u> ±0.0041	<u>0.2377</u> ±0.0042
DINO-MT-S/8 FlowProp	3	0.1460 ±0.0051	0.3327 ±0.0061	0.1116 ±0.0064	0.1887 ±0.0108	0.2271 ±0.0100
DINO-MT-S/8 FlowProp	4	0.1355 ±0.0010	0.3327 ±0.0085	0.0994 ±0.0027	0.1713 ±0.0040	0.2127 ±0.0001
DINO-MT-S/8 FlowProp	5	0.1321 ±0.0115	0.3197 ±0.0106	0.0950 ±0.0170	0.1732 ±0.0016	0.2041 ±0.0030
DINO-MT-S/8 FlowProp	6	0.1264 ±0.0032	0.3115 ±0.0105	0.0874 ±0.0098	0.1653 ±0.0018	0.1943 ±0.0018

Table 4.10: Results of DINO ViT-S/8 MaskTrack FlowProp for varying prediction frequencies on the YouTube-VIS 2019 validation dataset. Best scores are formatted in **bold**, while second-best scores are <u>underlined</u>.

we use the DINO-MT-S/8 model to analyze our custom tracker. Figure 4.3 shows the result of the experiments for ε between 0 and 15. Detailed values are shown in the Appendix in Table 6.1

MapMatch leads to an improvement of at least 0.49% in AP and at most 2.42%. All other metrics improve too, with up to 4.45% in AP50, 3.58% in AP75, 2.15% in AR1 and 2.81% in AR10. We achieve an improvement of at least 0.71% in AP50, 0.91% in AP75, 0.81% in AR1 and 0.69% in AR10. No matter which ε we choose, the performance improves. MapMatch is therefore an effective approach to improve the performance of DINO MaskTrack R-CNN through optical flow. The best choice for ε lies between 12 and 14. Because $\varepsilon = 13$ performs best in AP50, AP75, AR10 and second-best in AR50, we choose it as the ideal hyperparameter.

MapMatch ablations A whole ablation study where every cue combination is computed is infeasible. Thus we perform the experiment on binary combinations of the different cues. Appearance similarity is set per default, as it is the only cue that is used in the

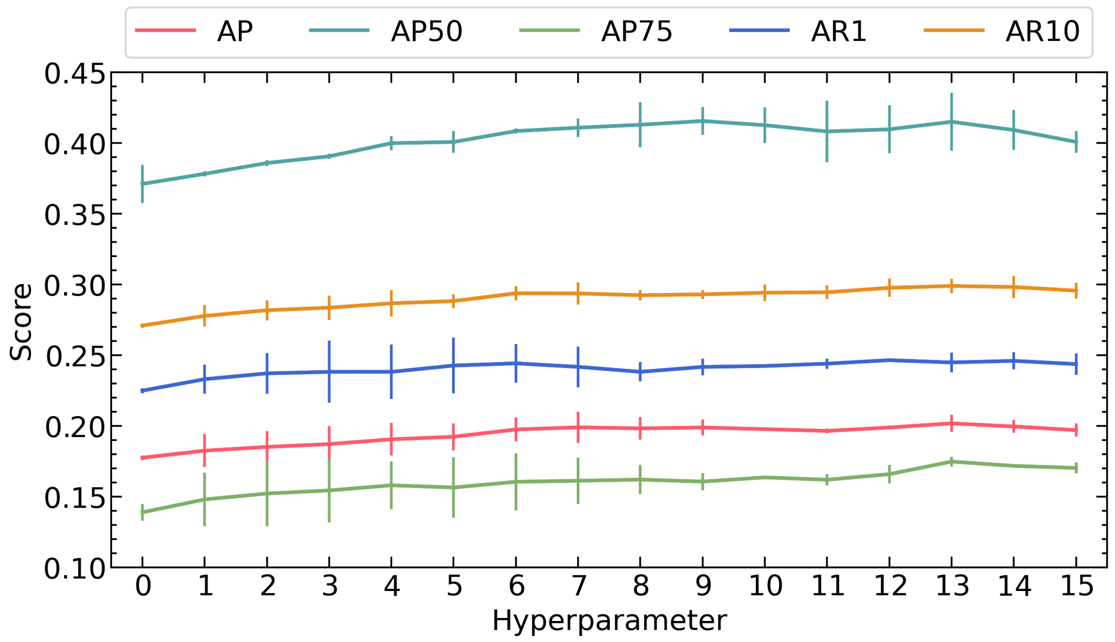

Figure 4.3: Effects of the hyperparameter choice for ε that controls the influence of the MapMatch cue.

training procedure. We then combine it with MapMatch and one of the other original cues. Table 4.11 shows the results of this experiment.

Cues	AP	AP50	AP75	AR1	AR10
Appearance + Flow	0.0738 ±0.0060	0.1628 ±0.0008	0.0524 ±0.0033	0.0904 ±0.0063	0.1358 ±0.0016
Appearance + Detection + Flow	0.0980 ±0.0096	0.2165 ±0.0354	0.0703 ±0.0061	0.1193 ±0.0037	0.1744 ±0.0017
Appearance + IoU + Flow	0.0810 ±0.0021	0.1749 ±0.0043	0.0601 ±0.0001	0.0996 ±0.0052	0.1412 ±0.0085
Appearance + Label + Flow	<u>0.1893</u> ±0.0045	<u>0.3875</u> ±0.0059	<u>0.1619</u> ±0.0094	<u>0.2369</u> ±0.0000	<u>0.2861</u> ±0.0071
Appear. + Det. + IoU + Label + Flow	**0.2017** ±0.0062	**0.4149** ±0.0203	**0.1747** ±0.0035	**0.2448** ±0.0069	**0.2989** ±0.0049

Table 4.11: Ablation results of DINO ViT-S/8 MaskTrack MapMatch for selected cue combinations on the YouTube-VIS 2019 validation dataset. Best scores are formatted in **bold**, while second-best scores are <u>underlined</u>.

We see that appearance similarity and our flow-based cue do not suffice as cues because that version performs almost 13% AP worse than the version with all cues. AP50 is affected the most (-25.21%). In comparison, bounding box overlap measured in IoU only boosts the performance by 0.72% AP, while the detection confidence improves AP by 2.42%. The most valuable addition is the label consistency cue, which raises AP by 11.55%. Furthermore, it only performs 1.24% AP worse than the version with all cues. Detection confidence and bounding box overlap therefore only cause a litte improvement. In conclusion, the combination of all cues accounts for the good performance.

DualFlow Lastly, we would like to combine all the methods we have introduced into one solution. Table 4.12 shows the influence of each of our introduced flow-based methods and its combination thereof on our DINO MaskTrack R-CNN architecture.

When we combine our flow-based methods, we can achieve almost the performance of DINO MaskTrack R-CNN, where we did not use our flow-based guidance. AP is only 0.6% lower, while AP75, AR1 and AR10 are lower by 0.62%, 0.93% and 1.44% respectively. AP50 is even 0.71% better with our combined flow guidance. The performance loss that

Method	**AP**	**AP50**	**AP75**	**AR1**	**AR10**
DINO-MT-S/8	<u>0.1775</u> ±0.0017	0.3710 ±0.0135	<u>0.1389</u> ±0.0060	<u>0.2249</u> ±0.0018	<u>0.2708</u> ±0.0015
DINO-MT-S/8 FlowProp	0.1560 ±0.0006	0.3568 ±0.0255	0.1148 ±0.0114	0.2011 ±0.0041	0.2377 ±0.0042
DINO-MT-S/8 MapMatch	**0.2017** ±0.0062	**0.4149** ±0.0203	**0.1747** ±0.0035	**0.2448** ±0.0069	**0.2989** ±0.0049
DINO-MT-S/8 MapMatch + FlowProp	0.1715 ±0.0063	<u>0.3781</u> ±0.0280	0.1327 ±0.0023	0.2156 ±0.0005	0.2564 ±0.0014

Table 4.12: Results of DINO ViT-S/8 MaskTrack for different flow-guidance setups on the YouTube-VIS 2019 validation dataset. Best scores are formatted in **bold**, while second-best scores are <u>underlined</u>.

would be caused by the omission of Mask R-CNN predictions on every second frame can therefore almost be compensated through its combination with the MapMatch method.

Best combinations In Section 4.4.1, we have shown which design choices can be exploited to make DINO-based backbones work well in a VIS setting. We combine our findings in two models: one that corresponds to ViT-S and one to ViT-B. Both models are trained on 8×8 patches, extract the features of every third encoder block and use bilinear interpolation as their resampling method. Additionally, linear mapping is added and group normalization on the feature embeddings is performed with a group size of 32. The input dimensions are also lowered to 224×120 pixels and the model is finetuned on Mask R-CNN weights. Moreover, we show the effects of optical-flow-based guidance on these models. Table 4.13 presents the results of this experiment.

Our goal was to minimize the performance gap between our proposed DINO MaskTrack R-CNN architecture and the baseline MaskTrack R-CNN architecture. While we can not match the performance of the original baseline, the performance difference is only approximately 5% in AP for both variants that use optical-flow-guidance in form of MapMatch. The difference between DINO MaskTrack R-CNN Base and Small is only 0.5% in AP, 0.9% in AP50, 1.8% in AP75, 1.2% in AR1 and 1% in AP10. However, DINO MaskTrack R-CNN has approximately 40% less trainable parameters than the baseline architecture.

Method	AP	AP50	AP75	AR1	AR10
MaskTrack R-CNN	**0.303**	**0.511**	**0.326**	**0.310**	**0.355**
DINO-MT-S/8 MapMatch	0.225 ±0.003	0.436 ±0.009	0.211 ±0.012	0.259 ±0.012	0.305 ±0.007
DINO-MT-S/8 MapMatch	0.246 ±0.007	0.459 ±0.001	0.232 ±0.027	0.270 ±0.013	0.329 ±0.014
DINO-MT-B/8 MapMatch	0.242 ±0.008	0.447 ±0.027	0.234 ±0.007	<u>0.283</u> ±0.000	0.327 ±0.001
DINO-MT-B/8 MapMatch	<u>0.251</u> ±0.001	<u>0.468</u> ±0.018	<u>0.250</u> ±0.024	0.282 ±0.013	<u>0.339</u> ±0.010

Table 4.13: Results of DINO ViT-S/8 and ViT-B/8 MaskTrack MapMatch for the best architecture combination (outputs of every third encoder block, resampling via bilinear interpolation, with linear mapping, trained and evaluated with an input dimension of 640×352 pixels) and initialized with pretrained MaskR-CNN weights) on the YouTube-VIS 2019 validation dataset. Best scores are formatted in **bold**, while second-best scores are <u>underlined</u>.

4.4.3 Qualitative Results

In this section we analyze the performance of our DINO MaskTrack R-CNN MapMatch architecture in a qualitative fashion. The original MaskTrack R-CNN baseline mainly suffered from changes in object appearance and the occlusion of objects [42].

While changes in object appearance can be handled relatively well in our method, occlusions of similar appearing objects still cannot be handled well. In this case, segmentation masks overflow into each other or the objects are treated as one joint instance, as shown in b) and d) in Figure 4.4. In the second frame of b) the elephants on the left are treated as a single instance. In the second frame of d), both apes are also merges into one detection. The method performance positively stands out in videos with single objects which have a different appearance than their backgrounds. These can be segmented and tracked precisely even when motions are large, as shown in a). Our method can track instances well even if they disappear in some frames, as shown in c). However, the quality of YouTube-VIS videos varies. In blurry videos, the method has a hard time with segmenting instances accurately. But compared to the baseline method we can observe that segmentation masks in our method are generally not as detailed. Additionally, the method is sometimes not confident in their class detection if objects appear similar to multiple classes, as shown in e).

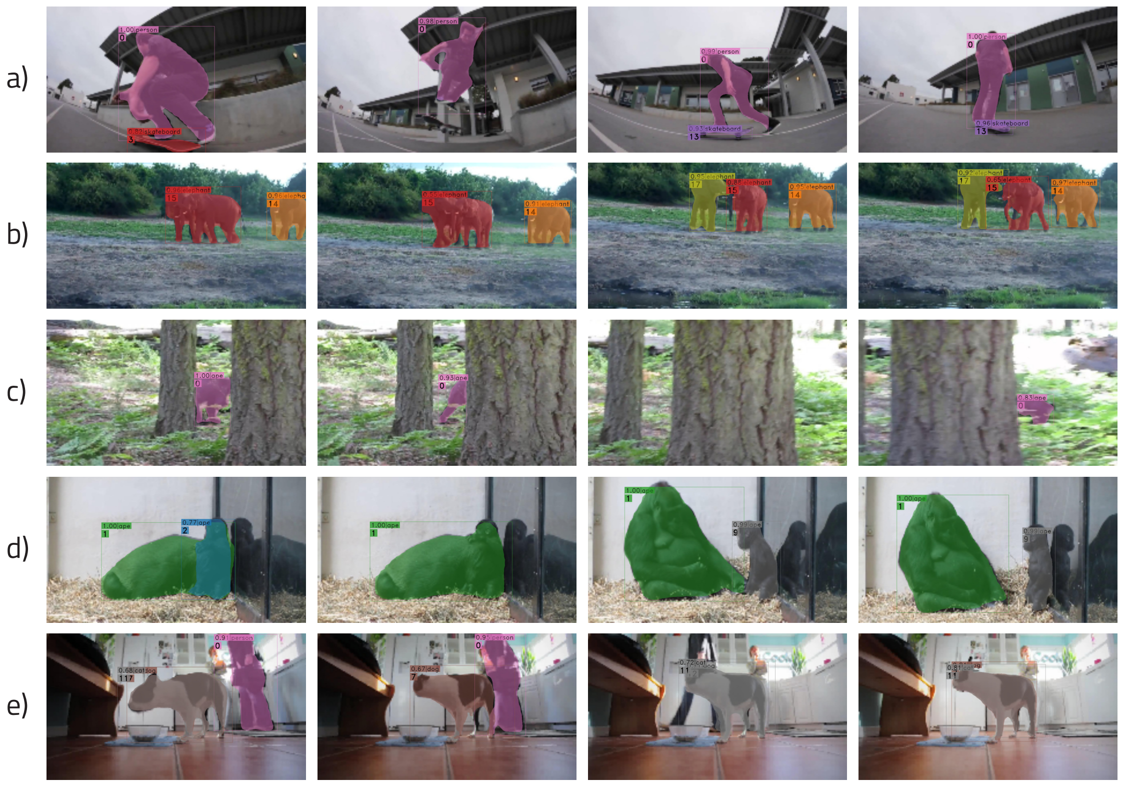

Figure 4.4: Examples generated with DINO ViT-S/8 MaskTrack R-CNN MapMatch on the validation dataset.

5 Discussion and Future Work

Video instance segmentation presents a fundamental problem for other video-analysis tasks. However, VIS methods are shown to be very computationally expensive or in do not perform as well in efforts for speed-up.

In this work, we tried to exploit big pretrained foundational models and their favorable properties in a VIS setting. Several works have shown that DINO features provide stable features that can be used as-is in downstream tasks. This motivated us to apply them in the VIS problem as well. To the best of our knowledge, our proposed DINO MaskTrack R-CNN is the first architecture that utilizes self-supervised DINO features in the context of video instance segmentation. Within the course of our experiments, we found out that certain modifications are necessary for them to work well within foundational architectures that are designed for object segmentation. As a consequence, we have modeled a DINO-based backbone for the MaskTrack R-CNN model. To further tackle problems of VIS, we propose methods that introduce guidance via optical flow in the testing procedure. Our experiments show that the exploitation of a fundamental concept like optical flow can improve the tracking performance or speed up the testing process.

Generally, our results show that self-supervised DINO features can work in dense tasks such as VIS, even though they were not trained on such a task. However, the detection performance can be still improved and our method cannot compete with current state-of-the-art methods in terms of accuracy. Still, our findings indicate that this method could be applied to other feature extractors that provide self-supervised representations in future work. Throughout our experiments, we only made modifications on the backbone and tracking head of MaskTrack R-CNN. Nevertheless, improvements could be made by tweaking the architecture of other components as well, similarly to [24]. Our methods for flow-guidance are also rather simple and future work in this direction is also possible.

6 Appendix

Method	ε	**AP**	**AP50**	**AP75**	**AR1**	**AR10**
DINO-MT-S/8	0	0.1775 ±0.0017	0.3710 ±0.0135	0.1389 ±0.0060	0.2249 ±0.0018	0.2708 ±0.0015
DINO-MT-S/8 MapMatch	1	0.1824 ±0.0116	0.3781 ±0.0018	0.1480 ±0.0190	0.2330 ±0.0104	0.2777 ±0.0076
DINO-MT-S/8 MapMatch	2	0.1851 ±0.0113	0.3858 ±0.0021	0.1522 ±0.0233	0.2371 ±0.0145	0.2817 ±0.0071
DINO-MT-S/8 MapMatch	3	0.1871 ±0.0127	0.3905 ±0.0018	0.1543 ±0.0224	0.2382 ±0.0220	0.2835 ±0.0087
DINO-MT-S/8 MapMatch	4	0.1905 ±0.0115	0.3998 ±0.0050	0.1580 ±0.0167	0.2382 ±0.0193	0.2866 ±0.0093
DINO-MT-S/8 MapMatch	5	0.1923 ±0.0096	0.4006 ±0.0076	0.1564 ±0.0213	0.2426 ±0.0196	0.2882 ±0.0048
DINO-MT-S/8 MapMatch	6	0.1974 ±0.0085	0.4084 ±0.0019	0.1604 ±0.0202	0.2442 ±0.0137	0.2937 ±0.0050
DINO-MT-S/8 MapMatch	7	0.1989 ±0.0111	0.4107 ±0.0064	0.1612 ±0.0164	0.2417 ±0.0144	0.2936 ±0.0079
DINO-MT-S/8 MapMatch	8	0.1982 ±0.0081	0.4128 ±0.0159	0.1620 ±0.0103	0.2382 ±0.0068	0.2923 ±0.0036
DINO-MT-S/8 MapMatch	9	0.1988 ±0.0056	**0.4155** ±0.0098	0.1606 ±0.0061	0.2416 ±0.0058	0.2929 ±0.0031
DINO-MT-S/8 MapMatch	10	0.1977 ±0.0003	0.4125 ±0.0127	0.1635 ±0.0011	0.2423 ±0.0003	0.2941 ±0.0059
DINO-MT-S/8 MapMatch	11	0.1964 ±0.0017	0.4081 ±0.0218	0.1619 ±0.0042	0.2440 ±0.0036	0.2944 ±0.0049
DINO-MT-S/8 MapMatch	12	0.1988 ±0.0013	0.4096 ±0.0169	0.1658 ±0.0066	**0.2464** ±0.0007	0.2976 ±0.0066
DINO-MT-S/8 MapMatch	13	**0.2017** ±0.0062	0.4149 ±0.0203	**0.1747** ±0.0035	0.2448 ±0.0069	**0.2989** ±0.0049
DINO-MT-S/8 MapMatch	14	0.1995 ±0.0045	0.4091 ±0.0141	0.1717 ±0.0012	0.2459 ±0.0060	0.2981 ±0.0078
DINO-MT-S/8 MapMatch	15	0.1970 ±0.0047	0.4007 ±0.0078	0.1702 ±0.0040	0.2436 ±0.0075	0.2956 ±0.0056

Table 6.1: Caption

Bibliography

[1] Muninder Adavelli. *How many videos are uploaded on YouTube every day?* July 27, 2023. URL: https://techjury.net/blog/how-many-videos-are-uploaded-to-youtube-a-day/.

[2] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. “Layer Normalization”. In: *arXiv preprint: 1607.06450* (2016).

[3] Gedas Bertasius and Lorenzo Torresani. “Classifying, Segmenting, and Tracking Object Instances in Video with Mask Propagation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9736–9745.

[4] Tom B. Brown et al. “Language Models are Few-Shot Learners”. In: *Advances in Neural Information Processing Systems*. 2020, pp. 1877–1901.

[5] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. “Unsupervised Learning of Visual Features by Contrasting Cluster Assignments”. In: *Advances in Neural Information Processing Systems*. 2020, pp. 9912–9924.

[6] Mathilde Caron et al. “Emerging Properties in Self-Supervised Vision Transformers”. In: *Proceedings of the International Conference on Computer Vision*. 2021, pp. 9630–9640.

[7] L. Ceci. *Hours of video uploaded to YouTube every minute as of February 2022*. Sept. 5, 2023. URL: https://www.statista.com/statistics/259477/hours-of-video-uploaded-to-youtube-every-minute/.

[8] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. “Improved Baselines with Momentum Contrastive Learning”. In: *arXiv preprint: 2003.04297* (2020).

[9] Johannes Deichmann, Eike Ebel, Kersten Heinecke, Ruth Heuss, Martin Kellner, and Fabian Steiner. *Autonomous driving’s future: Convenient and connected*. Jan. 6, 2023. URL: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/autonomous-drivings-future-convenient-and-connected.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2019, pp. 4171–4186.

[11] Guodong Ding, Fadime Sener, and Angela Yao. “Temporal Action Segmentation: An Analysis of Modern Techniques”. In: *arXiv preprint: 2210.10352* (2023).

[12] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *Proceedings of the International Conference on Learning Representations*. 2021.

[13] Alexey Dosovitskiy et al. “FlowNet: Learning Optical Flow with Convolutional Networks”. In: *Proceedings of the International Conference on Computer Vision*. 2015, pp. 2758–2766.

[14] Austin Erickson, Kangsoo Kim, Gerd Bruder, and Greg Welch. “A review of visual perception research in optical see-through augmented reality”. In: *International Conference on Artificial Reality and Telexistence and Eurographics Symposium on Virtual Environments*. 2022.

[15] Di Feng et al. “Deep Multi-Modal Object Detection and Semantic Segmentation for Autonomous Driving: Datasets, Methods, and challenges”. In: *IEEE Transactions on Intelligent Transportation Systems* 22.3 (2021).

[16] Jean-Bastien Grill et al. “Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. 2020, pp. 21271–21284.

[17] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. “Momentum Contrast for Unsupervised Visual Representation Learning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2020.

[18] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. “Mask R-CNN”. In: *Proceedings of the International Conference on Computer Vision*. 2017, pp. 2980–2988.

[19] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the Knowledge in a Neural Network”. In: *arXiv preprint: 1503.02531* (2015).

[20] Berthold K.P. Horn and Brian G. Schunck. “Determining optical flow”. In: *Artificial Intelligence* 17.1 (1981), pp. 185–203.

[21] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. “FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1647–1655.

[22] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the International Conference on Machine Learning*. 2015, pp. 448–456.

[23] Xiang Li, Jinglu Wang, Xiao Li, and Yan Lu. “Video Instance Segmentation by Instance Flow Assembly”. In: *IEEE Transactions on Multimedia* (2022), pp. 1–10.

[24] Yanghao Li, Saining Xie, Xinlei Chen, Piotr Dollar, Kaiming He, and Ross Girshick. “Benchmarking Detection Transfer Learning with Vision Transformers”. In: *arXiv preprint: 2111.11429* (2021).

[25] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *European Conference on Computer Vision*. 2014, pp. 740–755.

[26] Xiao Liu et al. “Self-supervised Learning: Generative or Contrastive”. In: *IEEE Transactions on Knowledge and Data Engineering* 35.1 (2020), pp. 857–876.

[27] Bruce D. Lucas and Takeo Kanade. “An Iterative Image Registration Technique with an Application to Stereo Vision”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*. 1981, pp. 674–679.

[28] MMTracking Contributors. *MMTracking: OpenMMLab video perception toolbox and benchmark*. `https://github.com/open-mmlab/mmtracking`. 2020.

[29] Preksha Pareek and Ankit Thakkar. “A survey on video-based Human Action Recognition: recent updates, datasets, challenges, and applications”. In: *Artificial Intelligence Review* 54.3 (2020), pp. 2259–2322.

[30] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. “Language Models are Unsupervised Multitask Learners”. In: (2019).

[31] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (2017), pp. 1137–1149.

[32] Newton Spolaôr, Huei Diana Lee, Weber Shoity Resende Takaki, Leandro Augusto Ensina, Claudio Saddy Rodrigues Coy, and Feng Chung Wu. “A systematic review on content-based video retrieval”. In: *Engineering Applications of Artificial Intelligence* 90 (2020), p. 103557.

[33] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. “PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8934–8943.

[34] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. “Instance Normalization: The Missing Ingredient for Fast Stylization”. In: *arXiv preprint: 1607.08022* (2016).

[35] Raj Vardhman. *13 Insightful Statistics on How Many Videos are Uploaded to TikTok Daily*. July 31, 2023. URL: `https://techjury.net/blog/how-many-videos-are-uploaded-to-tiktok-daily/`.

[36] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5998–6008.

[37] Haochen Wang, Xiaolong Jiang, Haibing Ren, Yao Hu, and Song Bai. “SwiftNet: Real-Time Video Object Segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2021, pp. 1296–1305.

[38] Jialian Wu et al. “Efficient Video Instance Segmentation via Tracklet Query and Proposal”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2022, pp. 949–958.

[39] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. *Detectron2*. `https://github.com/facebookresearch/detectron2`. 2019.

[40] Zuxuan Wu, Ting Yao, Yanwei Fu, and Yu-Gang Jiang. “Deep learning for video classification and captioning”. In: *Frontiers of Multimedia Research*. 2018, pp. 3–29.

[41] Haofei Xu, Jiaolong Yang, Jianfei Cai, Juyong Zhang, and Xin Tong. “High-Resolution Optical Flow from 1D Attention and Correlation”. In: *Proceedings of the International Conference on Computer Vision*. 2021, pp. 10478–10487.

[42] Linjie Yang, Yuchen Fan, and Ning Xu. “Video Instance Segmentation”. In: *Proceedings of the International Conference on Computer Vision*. 2019, pp. 5187–5196.

[43] Otilia Zvorişteanu, Simona Caraiman, and Vasile-Ion Manta. “Speeding Up Semantic Instance Segmentation by Using Motion Information”. In: *Mathematics* 10.14 (2022).