Machine Visual Perception
Course Project Report

Alexandru-Vlad Niculae
avn29

Ziyi Liu
zl413

Group number: 4
December 3, 2021

0 Individual contributions

Report

The individual contributions to the report are highlighted at the beginning of each section and subsection. These are usually linked to our contributions to the code.

Code

Alexandru-Vlad Niculae (avn29)

First, I have modified the `BackgroundMattingV2` baseline codebase to integrate CPU support and to fix other minor issues with the baseline (see commit). I have then modified the `Segmenter` codebase (see commit in the Segmenter codebase) and integrated it into the baseline (see commit in the video matting baseline). I have also analysed, designed and implemented the modified ViT architecture which is able to output images of any shapes (instead of classification labels, as in the original paper) - see the committed changes in the `pytorch-ViT` codebase and the commited changes in the `BackgroundMattingV2` baseline. Among these commits, I have also modified the baseline's files used for inference (`inference_images.py`, `inference_video.py`) and the file used for training (`train_base.py`) in order to integrate the different approaches: my overall changes to the `BackgroundMattingV2` baseline can be seen here.

Ziyi Liu (zl413)

All of my model improvements, adaptations, and integration were built on the same `BackgroundMattingV2` baseline codebase (baseline). I first worked on transferring the data input pipelines into the Segmenter codebase (see initial experiments). After we decided to change the direction of integration, I worked on the TransUNet integration to swap out the existing encoder-decoder network (see commit). Then I moved on to adding ViT models into the original CNN-based encoder backbone, where the ViT architecture was originally from this script but I adapted it to our own architecture. The full integration can be seen in this commit. Finally, I set up the inference and evaluation pipeline for quantitative results, shown in this commit. The metrics functions were taken from this script, but the evaluation pipeline for our dataset was implemented from scratch.

1 Introduction and Motivation

1.1 Introduction to the problem

*Subsection written by **Ziyi Liu (zl413)***

Video matting refers to the task of decoupling the foreground from the background captured in videos. It represents a well-studied problem in the field of computer vision and is often used to apply special effects on the background for entertainment industries or to enhance security and privacy by blurring the background while retaining the fine-matted foreground figure [1]. Using a green screen for easier matting has been a canonical approach, but it is often inconvenient or impractical to have a green screen available, and many times we would like to extract the foreground in a real-world environment.

For image matting, we can formulate the problem as finding the optimal alpha matte and F for each pixel on image I, such that

$$I_i = \alpha_i \times F_i + (1 - \alpha_i) \times B_i$$

where F and B denotes background and foreground intensity, each i subscript represents a pixel, and all quantities on the right hand side are unknown [18]. Traditionally, we are dealing with an under-constrained problem, and many methods solve for alpha, F, and B simultaneously with iterative optimization [19], but this problem can now also be approached with neural networks that have the capability to capture local details for high-resolution matting.

Video matting, as an extension to image matting, shares a similar problem setting, but further requires temporal consistency. An intuitive approach is to apply an image matting model to each frame of the video, but this may lead to flickering and inability to process fast-moving objects. Methods that take temporal relations into account on top of image matting have been developed to improve video matting quality without further assistive information.

1.2 Background and related work

*Subsection written by **Alexandru-Vlad Niculae (avn29)***

As mentioned above, any previous work on image matting can be transferred to video by simply applying the algorithms on each individual frame of the video. Early work on image matting did not perform any learning, but rather applied traditional computer vision techniques that can be classified into 2 categories: sampling-based methods [8, 9, 10] and affinity-based methods [11, 12, 13]. However, considering individual frames only does not use the additional information that is provided by the numerous video frames, which results in temporal inconsistency. The first papers that considered the relationship between frames [14, 15] required the user to provide manual annotations for blocks of frames in order to indicate which pixels are part of the background and which are part of the foreground. These methods were satisfactory at the time of release, but cannot be used at scale in today's real-time software application. More recent techniques apply deep learning techniques such as convolutional neural networks in order to gain contextual information; for example, the work by Lin et al. [1] shows satisfactory real-time performance by requiring the user to capture only a single additional frame of the background (without any subjects). Other recent papers based on deep learning, however, do not require any additional user priors at all and show satisfactory real-time performance ([16]).

Moreover, in the past year, well-studied computer vision tasks such as Object Classification, Image Segmentation, Detection or Few-shot learning have seen improvements by replacing and/or enhancing the convolutional neural networks with the Visual Transformers (ViTs) blocks proposed by Dosovitskiy et al. [5]. While still being a new technique, visual transformers proved impressive results in extracting the contextual dependencies from within images.

1.3 Overview of the idea

*Subsection written by **Alexandru-Vlad Niculae (avn29)***

The aim of this project is to explore the benefits of visual transformers in the context of video matting. Hence, the goal is to add transformer blocks into an existing state-of-the-art video matting codebase and to analyse the changes produced in the training and validation stages. This is particularly interesting because previous work shows how classical computer vision tasks were improved by applying ViTs blocks; however, with video matting making no exemption from the list of classical computer vision tasks, there is currently no work that applies visual transformers to it. Moreover, the versatility of visual transformers also motivates this project, since they can be applied to replace or enhance any existing parts of the current convolutional neural networks that are being used to perform video matting.

2 Method

2.1 Baseline algorithm

*Subsection written by **Alexandru-Vlad Niculae (avn29)***

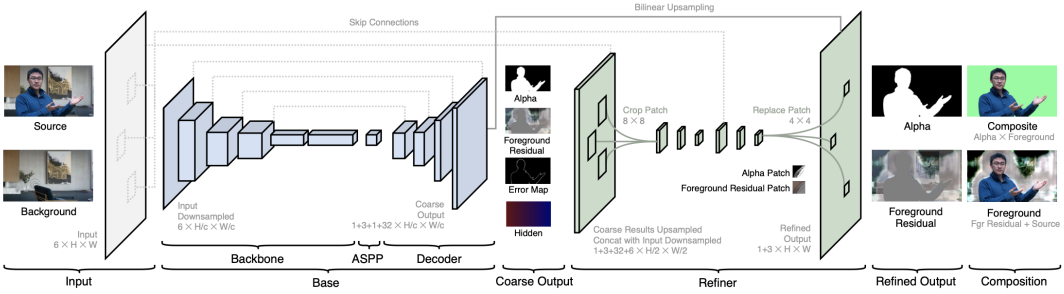


Figure 1: The baseline architecture proposed by Lin et al. [1] that was used throughout this project.

The `BackgroundMattingV2` [2] GitHub repository was used as a baseline and represents the implementation of the paper proposed by Lin et al. [1], their architecture being pictured in Figure 1. For each processed video, the codebase requires an additional frame of the background - this can be observed on the left hand side of the diagram above, in the `Input` section. The `Base` section consists of a CNN encoder-decoder architecture which operates on downsampled images and which outputs four components, as follows:

- **pha** $(\mathbf{B}, \mathbf{1}, \mathbf{H}, \mathbf{W})$ [1]: represents the alpha prediction, with pixel values normalised to $[0, 1]$.
- **fgr** $(\mathbf{B}, \mathbf{3}, \mathbf{H}, \mathbf{W})$ [1]: represents the RGB foreground prediction, with pixel values normalised to $[0, 1]$.
- **err** $(\mathbf{B}, \mathbf{1}, \mathbf{H}, \mathbf{W})$ [1]: represents the error prediction, with pixel values normalised to $[0, 1]$. The error is used in the `Refiner` component to refine the deficient areas of the predicted image.
- **hid** $(\mathbf{B}, \mathbf{32}, \mathbf{H}, \mathbf{W})$ [1]: represents the hidden layers, which are used in the `Refier` component.

The four outputs are then passed into the `Refiner` component, which applies convolution in two stages to recover the details of the most deficient estimated pixel areas. The new estimated pixels are used to replace the old pixels in the `Alpha` and `Foreground` modules, which are then outputted as the final prediction.

[1] B - batch size; H, W - height and width of the downsampled frames.

2.2 Algorithm improvements

*Subsection written by **Alexandru-Vlad Niculae (avn29)** and **Ziyi Liu (zl413)***

In order to improve the algorithm, we looked into methods to replace the `Base` CNN-based encoder-decoder component of the baseline with a new, transformer-based backbone. One solution we have taken was to integrate transformers designed for semantic segmentation that are able to output multi-channel images. For this purpose, we have integrated two models: Segmenter [3] and TransUNet [4].

Another solution that was carried out involved modifying the original Visual Transformer block proposed by Dosovitskiy et al. [5] (pictured in figure 18) to make it generate tensors with an arbitrary number of channels (instead of classification labels, as in the original design). To achieve this, a MLP block was applied on the patches outputted by the transformer block to upsample the number of channels, and then a linear projection was performed to unflatten the patches. The modified ViT architecture can be visualised in Figure 2.

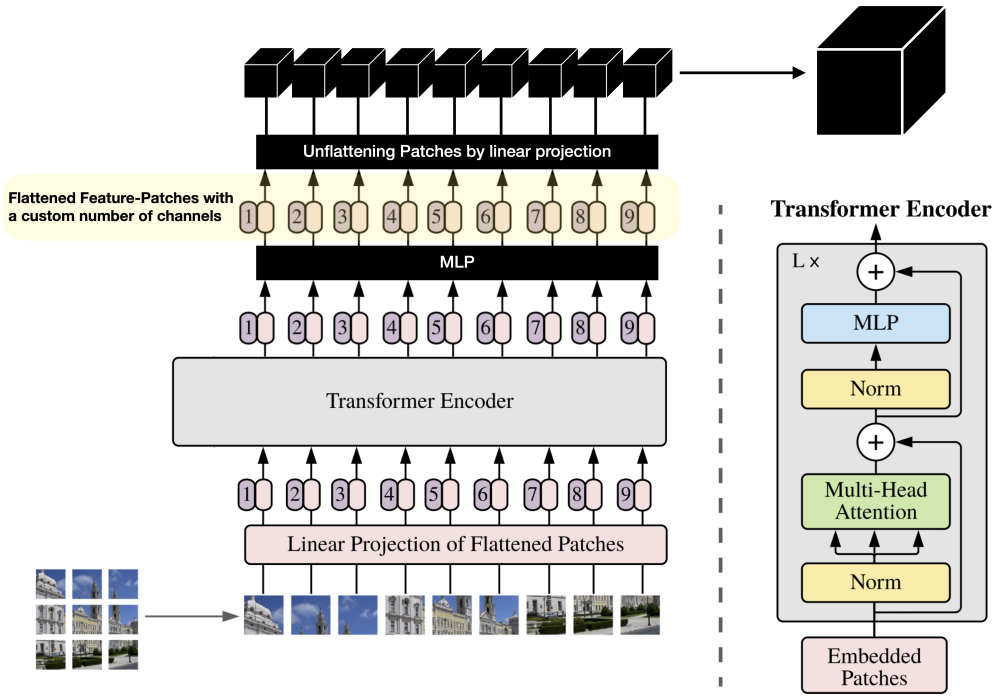


Figure 2: Our modified ViT architecture.

Finally, a hybrid model that integrates transformer layers into the original CNN-based architecture was proposed by us. The key idea is to see if inserting new transformer layers while retaining the original architecture can improve the model performance. Ideally, we could insert one ViT block [5] between each of the CNN blocks by matching the input and output shapes, but in order not to increase the model size by too much in our experiments, we only plant one such layer between the first and the second CNN blocks to demonstrate the feasibility. An illustration of the modification is shown in Figure 3.

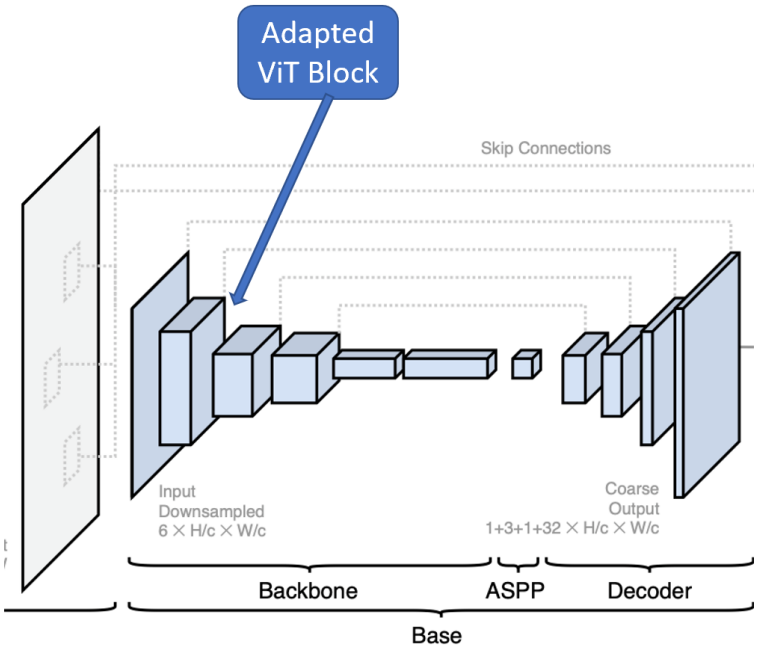


Figure 3: Our proposed ViT-CNN hybrid model structure

2.3 Implementation details

In this section we describe the implementation details of each approach outlined in the previous section.

Segmenter integration
*Subsection written by **Alexandru-Vlad Niculae (avn29)***

The integration of Segmenter became straightforward once we gained enough insights into the codebase. One important aspect was the modification of the number of output channels in the Segmenter model, which was changed to 37 (for the four outputs required by the `Refiner` component). The other main modification was performed in the `forward` function of the model, with the following additions:

```
1 def forward(self, im_fgr, im_bgr):
2           H_ori, W_ori = im_fgr.size(2), im_fgr.size(3)
3           im_fgr = padding(im_fgr, self.patch_size)
4           im_bgr = padding(im_bgr, self.patch_size)
5           H, W = im_fgr.size(2), im_fgr.size(3)
6
7           x = self.encoder(torch.cat([im_fgr, im_bgr], dim = 1),
            ↪  return_features=True)
```

It can be observed that line 7 performs the concatenation between the foreground and the background images, as employed in the `BackgroundMattingV2` codebase, which results in a 6-channel image.

TransUNet integration

*Subsection written by **Ziyi Liu (zl413)***

Another transformer model integration was based on TransUNet [4] backbone, a hybrid CNN-Transformer architecture that takes advantage of both the capability of CNN for capturing local features and the global context provided by Transformer model. As shown in the main architecture, the linear projection from CNN as input embedding was passed into 12 transformer blocks in sequence, the result of which was in turn concatenated with CNN outputs before being passed into the CNN-based decoder architecture (fig. 4).

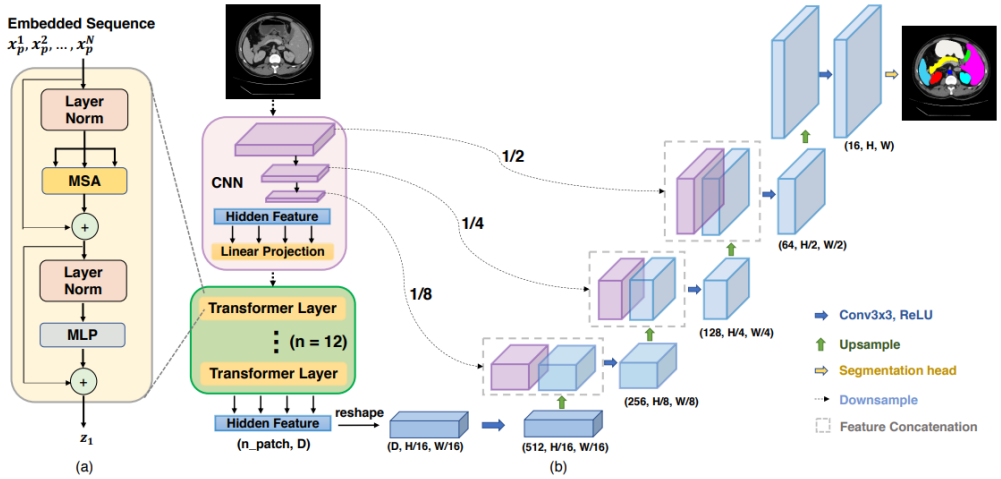


Figure 4: The TransUNet hybrid Transformer-CNN Architecture

From the code snippet below of the modified `Transformer` for TransUNet, we can see on lines 20-23 that the concatenated source-background pair is sent into Transformer encoder and CNN decoder, with a 37-channel output that fits into the `BackgroundMattingV2` training pipeline:

```
1  class VisionTransformer(nn.Module):
2      def __init__(self, config, img_size=512, num_classes=21843, zero_head=False,
       ↪  vis=False):
3          super(VisionTransformer, self).__init__()
4          outchans = 37
5          self.num_classes = num_classes
6          self.zero_head = zero_head
7          self.classifier = config.classifier
8          self.transformer = Transformer(config, img_size, vis)
9          self.decoder = DecoderCup(config)
10         self.segmentation_head = SegmentationHead(
11             in_channels=config['decoder_channels'][-1],
12             out_channels=outchans,
13             kernel_size=3,
14         )
15         self.config = config
16
17     def forward(self, x, x_bgr):
```

```
        if x.size()[1] == 1:
            x = x.repeat(1,3,1,1)
        x = torch.cat([x, x_bgr], dim=1)
        x, attn_weights, features = self.transformer(x)  # (B, n_patch, hidden)
        x = self.decoder(x, features)
        logits = self.segmentation_head(x)
        print(logits.shape)
        return logits[:,:1,:,:], logits[:,1:4,:,:], logits[:,4:5,:,:],
        ↪  logits[:,5:,:,:]
```

Our Modified ViT

*Subsection written by **Alexandru-Vlad Niculae (avn29)***

To implement the modified ViT architecture that would be capable of outputting tensors of any shapes (rather than classification labels), the `pytorch-vit` GitHub repository [6] was used as a baseline. The `forward` function was modified as follows:

```
1  def __init__(self, ...)
2      ...
3      patch_dim = channels * self.patch_height * self.patch_width
4      ...
5      self.mlp_head = nn.Sequential(
6          nn.LayerNorm(dim),
7          nn.Linear(dim, patch_dim // channels * out_channels),
8          nn.Linear(patch_dim // channels * out_channels,
9                      patch_dim // channels * out_channels)
10     )
11
12 def forward(self, im_fgr, im_bgr):
13     x = self.to_patch_embedding(torch.cat([im_fgr, im_bgr], dim = 1))
14     b, n, _ = x.shape
15
16     x += self.pos_embedding[:, :n]
17     x = self.dropout(x)
18
19     x = self.transformer(x)
20     x = self.mlp_head(x)
21
22     x = Rearrange('b (h w) (p1 p2 c) -> b c (h p1) (w p2)',
23                     h = im_fgr.shape[2] // self.patch_height,
24                     w = im_fgr.shape[3] // self.patch_width,
25                     p1 = self.patch_height,
26                     p2 = self.patch_width)(x)
27
28     return x[:,:1,:,:], x[:,1:4,:,:], x[:,4:5,:,:], x[:,5:,:,:]
```

Line 13 concatenates the two inputs required by the `BackgroundMattingV2` baseline and passes the 6-channel image to the linear projection. Line 21 applies our custom MLP block (defined on line 5) that upsamples the number of channels in each of the outputted feature-patches, while lines 22 to

26 apply a linear projection that unflattens the embedded patches. Finally, line 28 outputs the four components (`pha, fgr, err, hid`), as required by the baseline.

Our ViT-CNN hybrid architecture

*Subsection written by **Ziyi Liu (zl413)***

The central idea to the implementation of this hybrid architecture is restructuring the output of the original ViT architecture to have it fit into the subsequent CNN layer. The original ViT has a MLP head that produces a linear final output, so we need to reshape it to match the halved image size (128, 128), as shown from line 29. The full encoder architecture is also shown in lines 4-19, as the decoder structure was unchanged.

```
1  """
2  forward function from Encoder Class
3  """
4  def forward(self, x):
5          x0 = x  # 1/1
6          x = self.conv1(x)
7          x = self.bn1(x)
8          x = self.relu(x)
9          x1 = x  # 1/2
10         x = self.vit(x)
11         x = self.maxpool(x)
12         x = self.layer1(x)
13         x2 = x  # 1/4
14         x = self.layer2(x)
15         x3 = x  # 1/8
16         x = self.layer3(x)
17         x = self.layer4(x)
18         x4 = x  # 1/16
19         return x4, x3, x2, x1, x0
20
21 """
22 forward function from modified ViT Class as our Hybrid Model Class
23 """
24 def forward(self, img):
25         x = self.to_patch_embedding(img)
26         b, n, _ = x.shape
27         ...
28         x = self.mlp_head(x)
29         x = x.reshape(b, self.channels, self.image_height, self.image_width)
30
31         return x
```

2.4 Data pipelines

*Subsection written by **Ziyi Liu (zl413))***

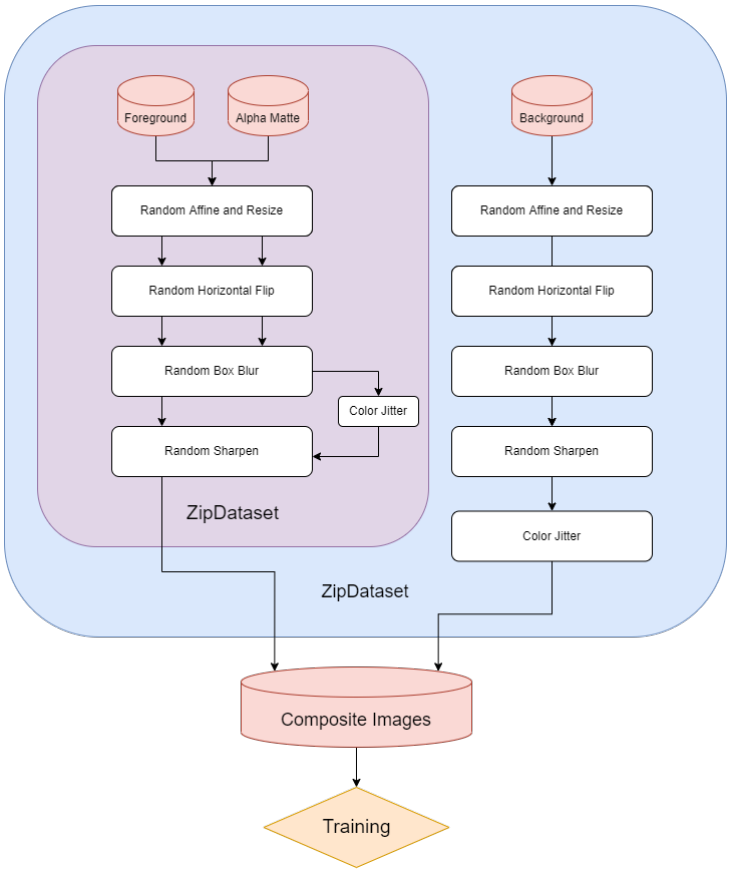


Figure 5: Training data pipeline and augmentations applied (our diagram).

The original codebase processed the three images (foreground, alpha matte, and background) by storing them in a nested `ZipDataset`, where a sequence of data transformation functions were applied (fig. 5) for individual augmentations.

Since the background is sampled independently from the foreground, the training source ('real' image) is composed by $source = F * \alpha + B * (1 - \alpha)$, followed by more augmentation transformations (fig. 5). This data pipeline was retained with minor changes to `random_crop` function to meet the input image size requirements for some transformer architectures.

We also implemented a separate data pipeline integration in the experiments where we attached the main existing pipeline to the `Segmenter` backbone. However, the majority of the experiments were conducted with the first approach.

2.5 Training procedures

*Subsection written by **Alexandru-Vlad Niculae (avn29)***

For training purposes, we employed the Adam optimizer [7] for achieving faster convergence, with a set learning rate of 0.0001. By modifying every transformer-based backbone to accommodate the characteristics of the `BackgroundMattingV2` codebase, little changes had to be made in the training procedures. As mentioned previously, for each batch of 8 images, several augmentation techniques are applied to the frames in order to improve the pixel-wise diversity of the dataset: techniques include affine transformation and noise/shadow/jitter addition. Beside these augmentation techniques, random-sized

crops are applied to the images before inputting them into the model in order to enhance the model's ability to perform well on random-sized video inputs. However, the visual transformers must be able to split the images into patches without any remaining pixels outside the patches. Hence, the image dimensions must be multiples of the patch dimensions. To avoid padding, we restricted the sizes of the random-sized crops to multiples of patch dimensions only. With this restriction in place, no overhead was introduced whilst having random-sized images during training. Lastly, the loss employed by the baseline during training was left untouched:

```
1 def compute_loss(pred_pha, pred_fgr, pred_err, true_pha, true_fgr):
2     true_err = torch.abs(pred_pha.detach() - true_pha)
3     true_msk = true_pha != 0
4     return F.l1_loss(pred_pha, true_pha) + \
5            F.l1_loss(kornia.sobel(pred_pha), kornia.sobel(true_pha)) + \
6            F.l1_loss(pred_fgr * true_msk, true_fgr * true_msk) + \
7            F.mse_loss(pred_err, true_err)
```

In order to learn the `alpha` component, lines 4 and 5 employ the L1 loss over the entire alpha matte and its (Sobel) gradient. Line 6 ensures the foreground layer is learned correctly by employing the L1 loss only over the pixels where the true `alpha` component is positive. Finally, line 7 computes the mean squared error loss between the predicted and true error maps, component which is used during Refinement.

2.6 Testing and validation procedures

*Subsection written by **Ziyi Liu (zl413)***

Our validation pipeline is coupled with the training procedure, where we run the model on the validation set after each training iteration. Similar to training, three of the four outputs are kept: predicted alpha matte, predicted foreground residuals (shortened as foreground from this point onwards), and predicted error map, which correspond to the first 5 channels in the model outputs. To illustrate the validation results, validation loss was plotted after each minibatch training.

For inference, we have tested two main procedures: inferring on a video and on a series of frames of images. The two approaches are fundamentally the same, but they require separate pipelines for input data. In `inference_images.py`, the testing pipeline takes in the source and background images directly without augmentation or compositions, as we would expect for real life applications, and they are passed into the model for inference straightforwardly. Since we adapted the `Base` backbone for our own implementations and experiments, the output items are identical to those in validation.

The predicted test results are in turn stored in disk for evaluation on various metrics, including MSE (mean squared error) for both alpha matte and foreground, and SAD (sum of absolute difference), Grad (spatial gradient metric), Conn (connectivity) for alpha matte only [1]. Temporal coherence of inferred stream can be tested with dtSSD metiric [17]; however, this will not be a major focus for our experiments.

The full post-processing procedure was implemented independently, which adapts the inferred image results for later evaluation pipeline, as shown below:

```
1 def src_compose(fgr_dir, bgr_path, pha_dir):
2
3     fgrs = sorted(os.listdir(fgr_dir))
4     phas = sorted(os.listdir(pha_dir))
```

```
    assert len(fgrs) == len(phas)

    for i in range(len(fgrs)):
        fgr_path = os.path.join(fgr_dir, fgrs[i])
        pha_path = os.path.join(pha_dir, phas[i])
        true_fgr = Image.open(fgr_path).resize((600,400))
        true_pha = Image.open(pha_path).resize((600,400))
        true_bgr = Image.open(bgr_path).resize((600,400))
        true_src = Image.composite(true_fgr, true_bgr, true_pha.convert("L"))
        true_src.save("inference_data/src_%s.jpg" % str(i).zfill(5), "JPEG")

def rename(dir):
    for file in os.listdir(dir):
        os.rename(os.path.join(dir, file), os.path.join(dir, 'src_' + file))

def crop_images(dir, size):
    for img in os.listdir(dir):
        image = Image.open(os.path.join(dir, img)).resize((size, size))
        image.save(os.path.join(dir, img))
```

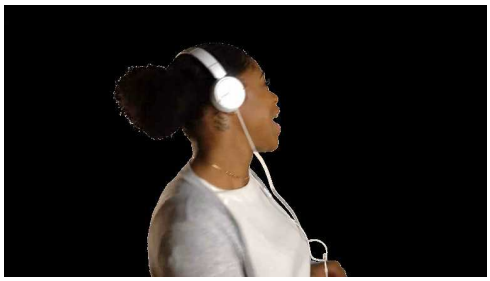
3 Experiments and Evaluation

3.1 Datasets

*Subsection written by **Ziyi Liu (zl413)***

Our main dataset for the following experiments was VideoMatte240K[1]. This consists of 484 pairs of HR (high resolution) alpha matte and foreground video clips broken down into individual frames (images), where 479 pairs are used for training and 5 for validation. Each video clip typically contains 300-1000 frames, contributing to a total amount of 240,000+ images in total. Both alpha matte and foreground data were extracted from the original green screen streams with Adobe After Effects [1].

(a) An example of the green screen video frames paired with their alpha matte



(b) An example of the actual foreground extracted from original dataset for training purposes

Figure 6: Examples from VideoMatte240K dataset

The main reason for adopting this dataset is that it is one of the largest public video matting datasets available to us with a mixture of HD and 4K resolution videos. Also, the two main components we wish to infer (alpha matte and foreground F) have already been pre-extracted, which saves the effort of further pre-processing. The backgrounds are stored separately in a different dataset containing 200 HR images with a very diverse range of environments, which can be easily combined with VideoMatte240K data to form 'real-life' videos.

One main limitation of the training data is that the foreground edges for fast-moving examples can be ragged due to the software extraction procedure, as we have no access to the original green screen video clips. Furthermore, the videos are ”synthetic” in nature, which may neglect certain background noises in real environments, such as a highly dynamic background. However, the diversity of our static backgrounds helps reduce the impact of this bias by effectively augmenting the training data.

3.2 Training and testing results

3.2.1 Training results

*Subsection written by **Ziyi Liu (zl413)***

Our training pipeline, as described above, was applied to each of our adapted or improved model architectures. In our setting with a batch size of 8, we have 29749 minibatches in each epoch, where each minibatch takes 40 seconds to train on a CPU on average, which would cost at least 13.8 days to complete one epoch. Since we had a handful of experiments to run and very limited resources, we only ran each training process for around 10-300 minibatches, This inevitably leads to suboptimal results, where the training loss has not converged, but it is a necessary compromise for this project.

We illustrate some of the training results with training loss curve as an intuitive measure of model improvement, and for each training iteration, we also evaluate on the validation set to produce a validation loss curve.

Segmenter Integration

*Subsection written by **Alexandru-Vlad Niculae (avn29)***

Figure 7 shows the validation loss obtained by the Segmenter model. The model fails to progress, which is most probably due to the poor fine-tuning, but also due to its systematic limitations.

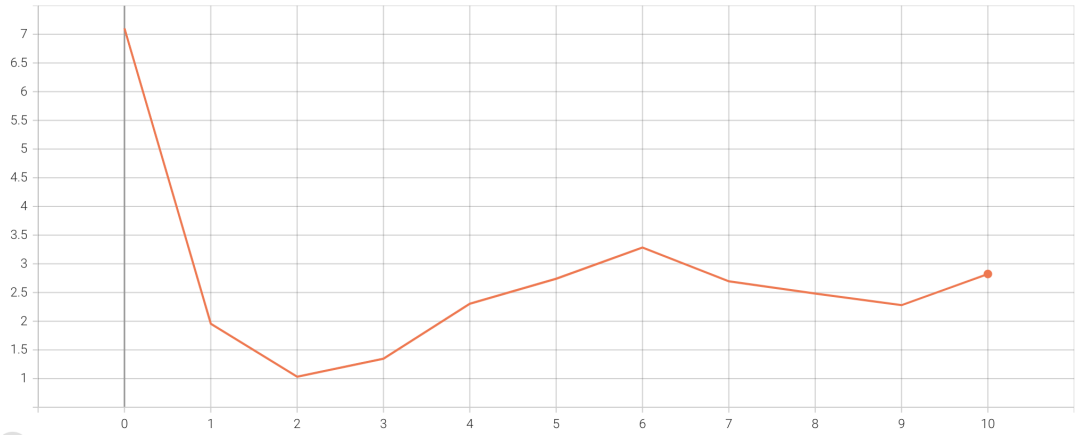
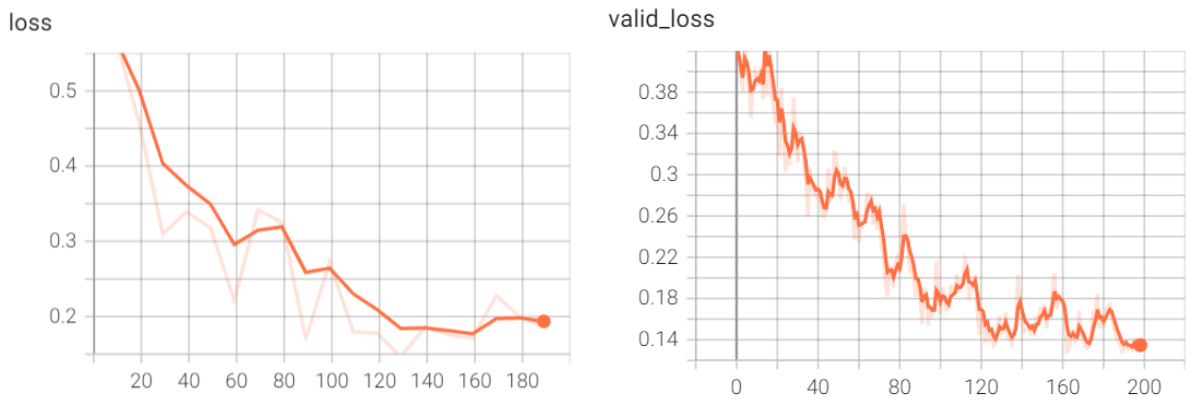


Figure 7: Validation loss curve obtained by running the Segmenter integration. (The Ox axis represents the batch index, while the Oy axis represents the loss employed by the baseline, as documented in Section 2.5).

TransUNet Integration

*Subsection written by **Ziyi Liu (zl413)***

Figure 8 shows the training and validation loss curves from the integrated TransUNet architecture for the first 200 iterations. We can observe a clear trend of dropping for both losses, which signifies the legitimacy of this integrated model (at least converging to a local minimum).



(a) TransUNet training loss curve for the first 200 iterations

(b) TransUNet validation loss curve for the first 200 iterations

Figure 8: Training results from the integrated TransUNet architecture

Our Modified ViT

*Subsection written by **Alexandru-Vlad Niculae (avn29)***

The training sessions of our modified ViT model provide encouraging results. Figure 9 shows the validation curve obtained by our ViT model with a single fully-connected layer in the MLP head. As the loss is converging only after 50 minibatches, we have run another experiment in which we added one more fully-connected layer in the MLP head. The results obtained by the model with 2 fully-connected layers can be observed in Figure 10: the loss has not yet converged and the model seems to be more capable of learning.

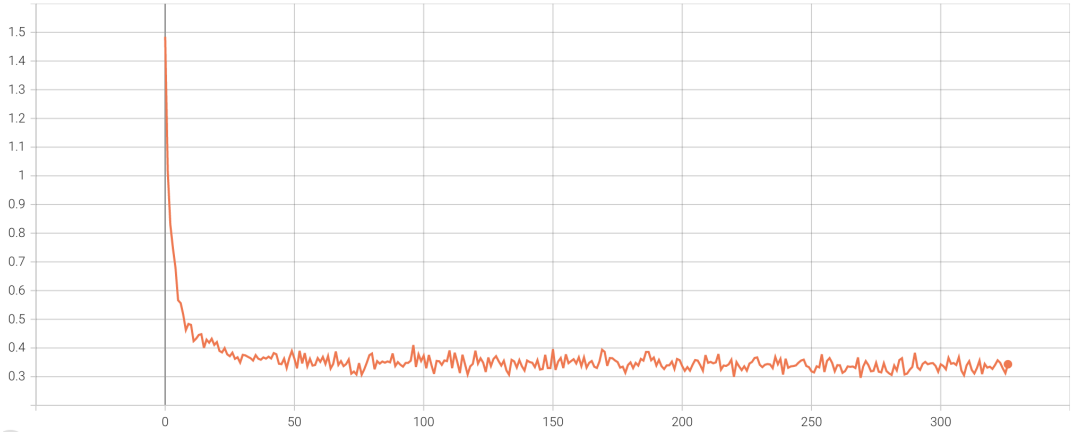


Figure 9: Validation loss curve obtained by running our modified ViT with a single fully-connected layer in the MLP head. (The Ox axis represents the batch index, while the Oy axis represents the loss employed by the baseline, as documented in Section 2.5).

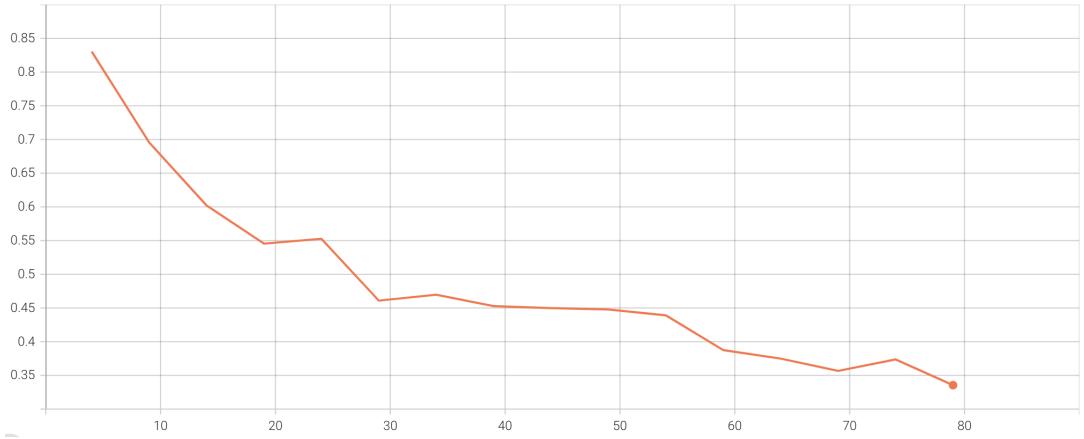


Figure 10: Validation loss curve obtained by running our modified ViT with two fully-connected layers in the MLP head.(The Ox axis represents the batch index, while the Oy axis represents the loss employed by the baseline, as documented in Section 2.5).

Our ViT-CNN hybrid Architecture
*Subsection written by **Ziyi Liu (zl413)***

Finally, we can see from Fig.11 that the valid loss for the hybrid model decreased rapidly for the first 5 iterations and started to fluctuate around 0.35 afterwards. This may be due to a local minimum convergence, or simply because of the low iteration count, as we can still observe the slight tendency for the loss to decrease further.

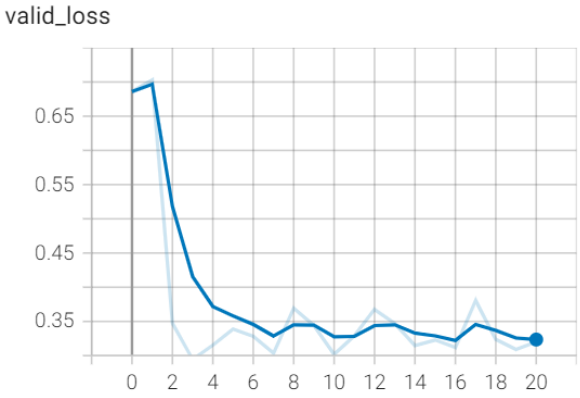


Figure 11: Validation loss curve obtained by running our hybrid ViT-CNN model.

3.3 Qualitative results

*Subsection written by**Alexandru-Vlad Niculae (avn29)** (first half) and **Ziyi Liu (zl413)** (second half)*

With training sessions that lasted only a handful of minibatches, the qualitative results offer only an intuition of what the models can achieve if trained sufficiently. Figure 12 and 13 show the background and the composite image for the the frame that was inputted into the models. Running our modified ViT with only one fully-connected (FC) layer in the MLP head results in the image pictured in figure 14, while figure 15 shows the result of the model with two FC layers. Both results show visible effects obtained from the 32×32 patches employed and highlight the importance of the patch sizes: higher patches imply faster but less accurate models, while smalles patches imply slower but more accurate

models. It is also interesting to observe that even though the multi-FC layer version was trained 3 times less than the single FC layer variant, the former one achieves better matting of the subject, confirming its learning capacity.

Figure 12: Real background

Figure 13: Real image

Figure 14: Output from our modified ViT with 1 FC layer.

Figure 15: Output from our modified ViT with 2 FC layers.

We then show an example of the inferred alpha-matte and foregrounds with TransUNet model after training 200 minibatches (Fig. 16). The composite image Fig. 16a was combined from a background image (church interiors picture) and the true foreground clipped from the green screen, with the true alpha matte as the blending mask (Fig.16b). Note that the final source composite image had to be resized to fit in the inference model. Our foreground and alpha matte prediction results are shown in Fig. 16c,16d. In both cases we can see the outline of the human body separating the background and foreground, where the edges tended to be blurry, indicating higher uncertainties. Although these predictions are still quite far from our desired outputs, it can be concluded that our model is on the right track, and the training certainly optimised the model performance given only 200 minibatches out of almost 30,000 in only one epoch.

3.4 Quantitative results and comparison to SOTA

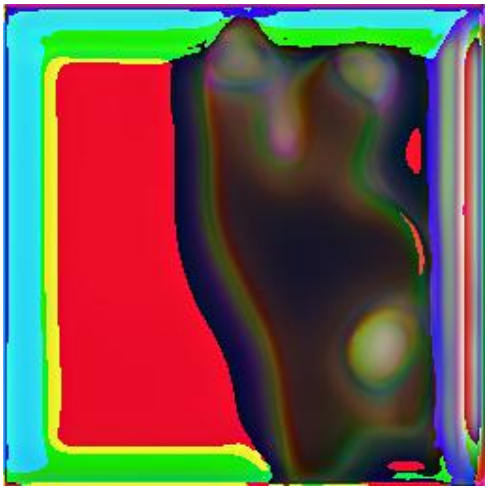
*Subsection written by **Ziyi Liu (zl413)***

Our quantitative results were obtained with the inference on 106 images from the first clip of test images in VideoMatte240K dataset. We used our TransUNet integrated model as it achieved the lowest validation loss during training. The metrics discussed in section 2.6 are listed below (Table 1, 2).

Table 1 shows our numerical results and the comparison with the state of the arts listed from [16] on the Alpha Matter output. We can see that the quantitative results in our case are significantly worse than the SOTA for all metrics as we expected. Similarly, Table 2 shows the two metrics on Foreground outputs, manifesting a similar divergence between the two results. With more computational power and training time, we would expect the evaluation results to be at least in the ballpark of the SOTA results, if not comparable.

(a) The composite source image as inference input combined from the background, true foreground, and true alpha-matte as the mask.

(b) The true alpha-matte for the resized image.



(c) Inferred foreground image, where the rough outlines defining the shape of the body can be seen.

(d) Inferred alpha-matte, where the body shape is clearly carved out, with higher uncertainties around the edges.

Figure 16: Inference example with a frame in VideoMatte240K testing dataset

Architecture	MAD	MSE	GRAD	CONN	dtssd
Our TransUNet	294.93	263.38	56.04	19.50	17.46
Robust [16]	6.08	1.47	0.88	0.41	1.36

Table 1: Evaluation results for Alpha Matte predictions.

Architecture	MAD	MSE
Our TransUNet	218.60	81.91
Robust [16]	N/A	2.60

Table 2: Evaluation results for Foreground predictions.

4 Conclusions and Future Directions

4.1 Conclusions

*Subsection written by **Ziyi Liu (zl413)***

We experimented four main implementations with different modes of transformer integration into the traditional ResNet-based architecture on the tasks of image and video matting, and promising early results were shown. With established data pre-processing and post-processing pipelines, we trained our proposed models on a subset of VideoMatte240K [1] training data for exploratory investigations, and evaluated the results on various sources. Three of the four main implementations showed consistently dropping training and validation loss, and for the model with best validation performance, we evaluated on the test set with a variety of metrics, showing reasonable early results. Qualitative analysis of the inferred alpha-matte and foreground images shows that our models are capable of capturing the edge information as well as the associated uncertainties, despite the limited training.

4.2 Discussion of limitations

*Subsection written by **Alexandru-Vlad Niculae (avn29)** (first half) and **Ziyi Liu (zl413)** (second half)*

The approaches that were carried out have several limitations. First, the integrations of Segmenter and TransUnet present a systematic limitation. Since these models were designed for semantic segmentation, their decoders output one mask for each semantic class. In our integrations, we have used the number of semantic classes as channels for the required output. However, this approach becomes unsuitable when decoders are optimizing for the specific semantic classes: for example, as seen in figure 17, the `Mask Transformer` used by the Segmenter model introduces a learnable embedding for each class (in our case, a class is equivalent to one channel). This design considers each channel independently and does not account for the inter-dependencies between them.

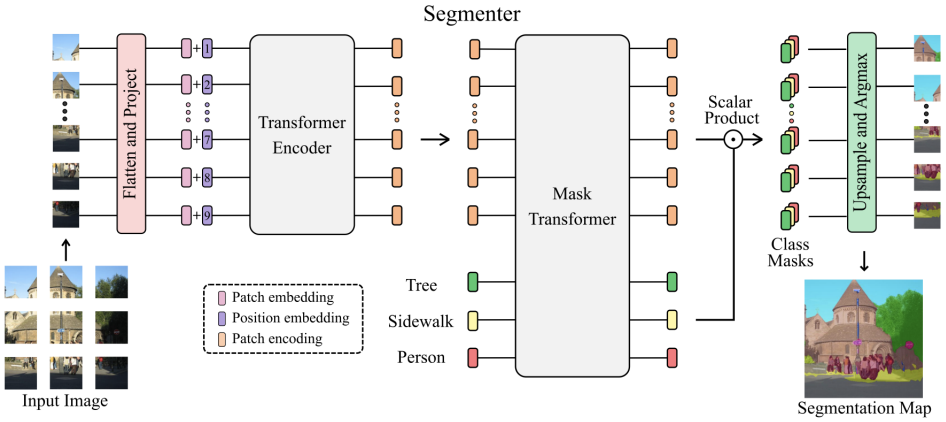


Figure 17: The Segmenter architecure [3]

While our modified ViT architecture does not present systematic limitations, it presents (as all the other transformer-based models) performance limitations that arises from the quadratric overhead introduced by transformers. With the current ViT approaches, there is a trade-off that must be made between the usage of bigger patches, which results in less overhead but also worse accuracy, and smaller patches, which results in better accuracy but also significantly more overhead.

Our hybrid architecture made minimal changes to the original architecture, but the insertion of a full ViT network inevitably led to cobbling the model with a huge amount of parameters, and the situation can be significantly worse if we were to add more ViT layers into our model. This can be further optimised by designing a much smaller ViT-esque transformer block suited to the hybridisation.

The main objective for all of our proposed models is to improve the matting quality of images parsed from videos, such that temporal coherence was not considered in most cases. Even though our models may resolve the flickering issues, we still couldn't leverage the temporal information for better single-frame matting performance

In addition, our models and input pipeline requires the background image along with the source image to be matted for both training and inference, but a clean background photo is not always readily available in real life. This problem was dealt with in [16], with a sacrifice on the edge precision.

4.3 Future directions

*Subsection written by **Alexandru-Vlad Niculae (avn29)***

The experiments performed and their results are promising. Most probably, visual transformers are capable of obtaining competitive video matting results when compared to the state-of-the-art methods, if designed and trained carefully. Our modified ViT architecture, as well as the ViT-CNN hybrid architecture showed that transformer-based architectures can be used as a base component for tackling video matting. In order to obtain real-time performance, future work should most probably focus on optimizing the quadratic overhead introduced by transformers and apply them in a new or existing architecture: for example, one such approach might want to apply visual transformers only on patches with the highest predicted error, by using the predicted error map texttterr from the `BackgroundMattingV2` baseline.

A Appendix

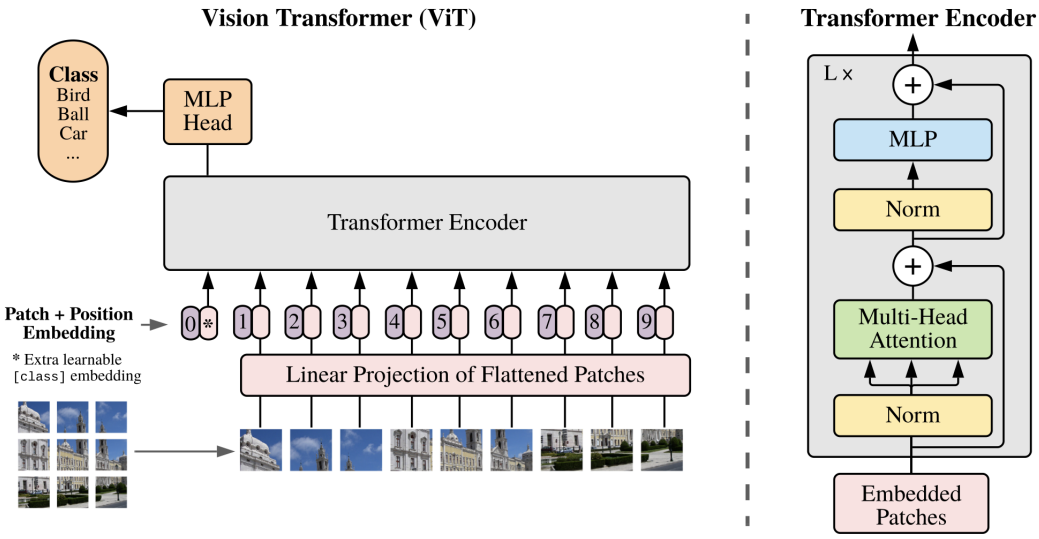


Figure 18: The original ViT architecture, as proposed by Dosovitskiy et al. [5].

References

[1] Lin, S., Ryabtsev, A., Sengupta, S., Curless, B.L., Seitz, S.M. and Kemelmacher-Shlizerman, I. 2021. Real-time high-resolution background matting. *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 8762-8771).

[2] Lin, S., Ryabtsev, A., Sengupta, S., Curless, B.L., Seitz, S.M. and Kemelmacher-Shlizerman, I. BackgroundMattingV2. https://github.com/PeterL1n/BackgroundMattingV2

[3] Strudel, R., Garcia, R., Laptev, I. and Schmid, C., 2021. Segmenter: Transformer for Semantic Segmentation. *arXiv preprint arXiv:2105.05633.*

[4] Chen, J., Lu, Y., Yu, Q., Luo, X., Adeli, E., Wang, Y., Lu, L., Yuille, A.L. and Zhou, Y., 2021. Transunet: Transformers make strong encoders for medical image segmentation. *arXiv preprint arXiv:2102.04306.*

[5] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S. and Uszkoreit, J., 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929.*

[6] Vision Transformer - Pytorch https://github.com/lucidrains/vit-pytorch

[7] Kingma DP, Ba J. Adam. A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.* 2014 Dec 22.

[8] Chuang, Y.Y., Curless, B., Salesin, D.H. and Szeliski, R., 2001, December. A bayesian approach to digital matting. In Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001 (Vol. 2, pp. II-II). IEEE.

[9] Feng, X., Liang, X. and Zhang, Z., 2016, October. A cluster sampling method for image matting via sparse coding. In European Conference on Computer Vision (pp. 204-219). Springer, Cham.

[10] Gastal, E.S. and Oliveira, M.M., 2010, May. Shared sampling for real-time alpha matting. In Computer Graphics Forum (Vol. 29, No. 2, pp. 575-584). Oxford, UK: Blackwell Publishing Ltd.

[11] Aksoy, Y., Oh, T.H., Paris, S., Pollefeys, M. and Matusik, W., 2018. Semantic soft segmentation. *ACM Transactions on Graphics (TOG), 37(4), pp.1-13.*

[12] Aksoy, Y., Ozan Aydin, T. and Pollefeys, M., 2017. Designing effective inter-pixel information flow for natural image matting. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 29-37).*

[13] Bai, X. and Sapiro, G., 2007, October. A geodesic framework for fast interactive image and video segmentation and matting. *In 2007 IEEE 11th International Conference on Computer Vision (pp. 1-8). IEEE.*

[14] Bai, X., Wang, J. and Simons, D., 2011, October. Towards temporally-coherent video matting. *In International Conference on Computer Vision/Computer Graphics Collaboration Techniques and Applications (pp. 63-74). Springer, Berlin, Heidelberg.*

[15] Chuang, Y.Y., Agarwala, A., Curless, B., Salesin, D.H. and Szeliski, R., 2002, July. Video matting of complex scenes. *In Proceedings of the 29th annual conference on Computer graphics and interactive techniques (pp. 243-248).*

[16] Lin, S., Yang, L., Saleemi, I. and Sengupta, S., 2021. Robust High-Resolution Video Matting with Temporal Guidance. *arXiv preprint arXiv:2108.11515.*

[17] M. Erofeev, Yury Gitman, D. Vatolin, Alexey Fedorov, and J. Wang. Perceptually motivated benchmark for video matting. *In BMVC, 2015.*

[18] Ning Xu and Brian Price and Scott Cohen and Thomas Huang Deep Image Matting. *arXiv preprint arXiv:1703.03872.*

[19] Wang, J. and Cohen, M.F. An iterative optimization approach for unified image segmentation and matting. *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1.*