Transformer applied to Facial Landmark detection

Francesco Bertolotti

April 11, 2021

1 Abstract

Facial landmark detection is the task of accurately localize fiducial points on a human face. Several important applications rely on the accuracy of models to detects these landmarks. face recognition, head pose estimation, sentiment analysis are only few of these applications. Many approaches have been proposed to address this task. Among the most promising techniques for this task are based on deep learning models. Usually, Convolutional neural network are used to either directly predict fiducial points (numbered from 1 to 68 in Fig. 4) or to obtain features to be input to another model. In this work, we take the latter approach. We feed features collected by a convolutional neural network to a vision transformer. We will show that such simple architecture is enough to achieve results close to the state-of-the-art.

2 Introduction

One of the most popular deep learning (DL) architectures for natural language processing (NLP) is the Transformer architecture. The Transformer was popularized in the famous paper "Attention is all you need" [15]. From the original proposal, many architectural improvements have been proposed, e.g. [2, 9, 17] and many others. Recently, the Transformer is being applied also to task for which wasn't originally designed. One notable example is image processing. In particular, [4] adapt the Transformer architecture to process images. This represents one of the firsts instances of convolution-free architecture to achieve comparable results with the state-of-the-art (SOTA). The same idea is further explored in [16] using a pyramidal-like Transformer architecture. This architecture achieves further improvements surpassing results obtained by ResNets architectures [5]. While the Transformer architecture is gaining further traction in image processing tasks the potentialities remain mostly unexplored. With this project, we aim to apply and adapt the Transformer for the task of "Facial Landmark Detection". While most of the previous works aim to convolution-free architectures, we aim to combine the ResNets and the Transformers hoping to gain further improvements.

3 Vision Transformer

In this section, we are going to expose the inner workings of the Transformer architecture. Recall that it was introduced for NLP tasks. Often, sentences have words of which the meaning is dependent on their context. This is the problem that the transformer tries to address. Firstly, words are encoded into embeddings which capture their isolated semantic. Next, words are mixed with each other using self-attention. Self-attention, is an attention-mechanism where elements in a sequence are combined together based on their similarity. Intuitively, a word attends to (meaning "pays attention to") other similar words in the sentence. When a word u attends to a word v collects and incorporates

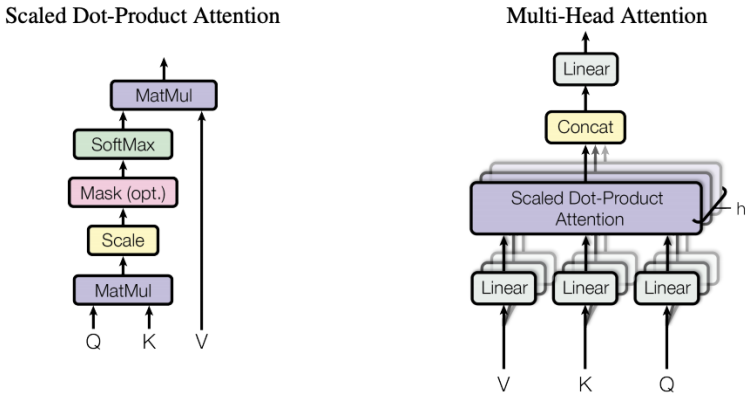


Figure 1: Flowchart of the main component of the transformer, self-attention.

the embedding of v into its own. Practically, u incorporates the meaning of v into itself. After several iterations of this mechanism a word obtains an embedding that it is not only representative of its own semantics, but also, it is representative of its surrounding.

An example may help to understand both the problem and the solution. Consider the phrase "bank of a river". The word "bank" alone has a semantic that is very different from the semantic that has in this phrase. During the previous steps of the transformer. The embedding of "bank" would get mixed with the embedding of "river" in such a way that, "bank" becomes representative of the contextual meaning of the word. Another way of seeing the same approach is the following: Some *complex meanings* can be only obtained by combining other *simple meanings* together. Self-attention is a mechanism for combining meanings.

Now, let us introduce the architecture by starting from the input. The Transformer takes as input n embeddings of size d. In our case, embeddings will be patches of an image, as shown by Fig. 2. Let us call $X = [x_1, \ldots, x_n] \in \mathbb{R}^{n \times d}$. Now that we have our patches, we can feed them to the first Transformer Encoder layer. Let us introduce three parameter matrices $W_Q, W_K, W_V \in \mathbb{R}^{n \times d}$. We proceed to compute the so-called queries, keys, and value vectors (shown as inputs in Fig. 1):

$$Q = X^T \cdot W_Q, K = X^T \cdot W_K, V = X^T \cdot W_V$$

Once we have computed these matrices representing three different linear transformations of the input, we proceed by applying the so-called self-attention (shown in 1). But firstly, let us introduce the softmax non-linearity. The softmax function takes as input a vector $v \in \mathcal{R}^{1 \times k}$ and outputs a vector $v' \in \mathcal{R}^{1 \times k}$

such that $\sum_i v'_i = 1$ and $v_i \leq v_j \implies v'_i \leq v'_j \forall i, j$:

$$softmax(v)_i = \frac{e^v_i}{\sum_j e^v_j}, v' = [softmax(v)_1, \ldots, softmax(v)_k]$$

Finally, we can compute the self-attention (depicted as Scaled Dot-Product Attention in Fig. 1):

$$Z = softmax(\frac{Q \cdot K^T}{\sqrt{d}}) \cdot V$$

This represents the heart of the Transformer architecture. By $Q \cdot K^T$, we are computing the dot product between each row-vector in Q and each row-vector in K. This returns a similarity matrix scoring each query against each key. Afterward, we turn these scores (by applying the softmax function row-wise) into probability distributions. By now, we have a probability distribution for each patch in our image. Each probability distribution tells us how similar is a patch wrt. the others. The last matrix multiplication is going to use these distributions to sum all the patches weighted by the probability score. Ultimately, self-attention is a way to reroute information between vectors based on their similarity. The last component of the encoder layer applies a standard feed-forward network and a skip connection with the original input. Now, multiple encoders can be stacked together to achieves, usually, higher results.

You may have noticed that the Transformer Encoder has no way of knowing if a patch comes from the center of the image or comes from the left corner of the image. To mitigate this issue, we introduce positional embeddings. These are simply parameter vectors $[e_1, \ldots, e_n] \in \mathcal{R}^{d \times n}$. These vectors are meant to represent positions and are summed altogether with our input. We can redefine the Transformer Encoder input as follows:

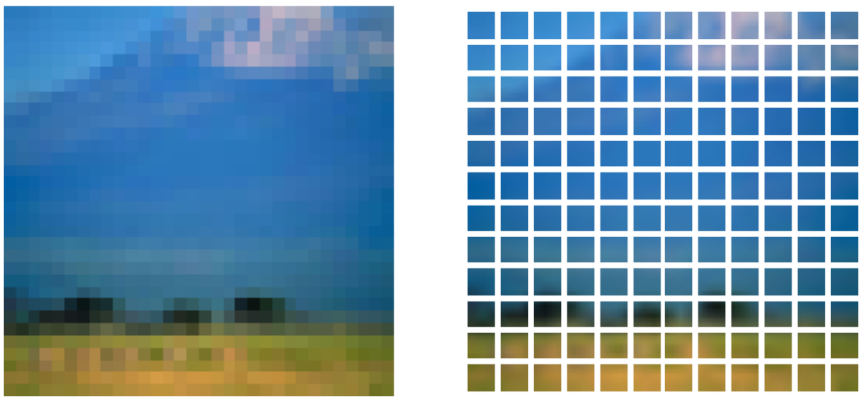
$$X = [x_1 + e_1, \ldots, x_n + e_n]$$

Another important component is the Transformer Decoder. However, the final architecture of this project will not be used. Thus, we are not discussing this component. Nonetheless, additional material about the transformer can be found `http://jalammar.github.io/illustrated-transformer/`.

With these considerations, our overview of the Transformer architecture is complete.

4 ResNet

As said before, we are going to combine both the architectures of ResNets and Transformers. While we have covered most of the concepts of the Transformers, we are still lacking the necessary concepts about ResNets. Let us start with the input of this architecture: an image $X \in \mathbb{R}^{C \times H \times W}$, where C represents the number of channels, H represents the image height and W represents the image width.



(a) full image

(b) image divided in $12 \times 12 = n$ patches

Figure 2: both figures are taken from the Keras vision transformer tutorial available at `https://keras.io/examples/vision/image_classification_with_vision_transformer/`

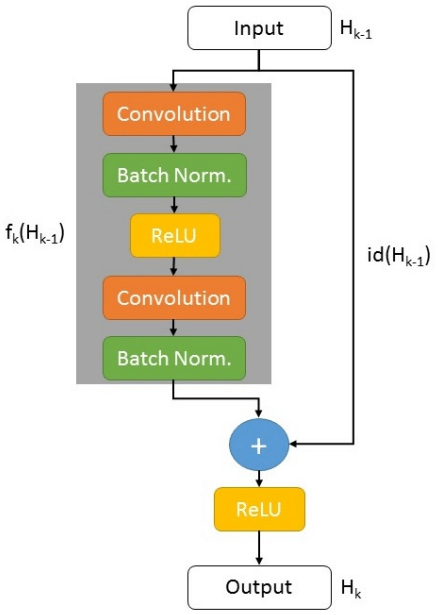
The input image will go through a series of convolutional blocks. Each convolutional block is shown in Fig. 3a. As you can see, the input (a tensor of dimension $C \times H \times W$) goes through convolutional layers an activation layer, and normalization layers. After this, the tensor is summed with the input and another activation is applied. One of the most important things to note is the skip connection named `id`. This type of connection, discovered to be effective by [5], is responsible for improving performance, reduce the vanishing/exploding gradient problem, and smoothing the loss landscape [8]. Following this simple structure, usually, a max-pooling layer is applied to reduce the number of features per channel.

A full depiction of a resnet architecture is shown in Fig. 3b. As you can see, many convolutional blocks are applied one after another increasing the number of channels from the original number (usually 3) to 512. The final depicted block is responsible for outputting the classification.

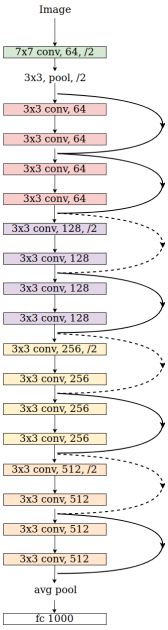
While the full intuition behind this architecture is extremely important it also out of the scope of this report. However, it is worth spending few more words about convolution. It is known that initial convolutional layers capture low-level features of the image e.g. lines, corners, and so on. Instead, higher convolutional layers use the low-level features to capture more and more abstract features like eyes, mouths, and so on.

5 Dataset, Applications and Metrics

Dataset In this section, we are going to discuss the dataset and the task that is proposed i.e. Facial Landmark Detection. Our dataset of choice is the 300W



(a) Depiction of a convolutional block from [7]



(b) Depiction of a ResNet from [5]

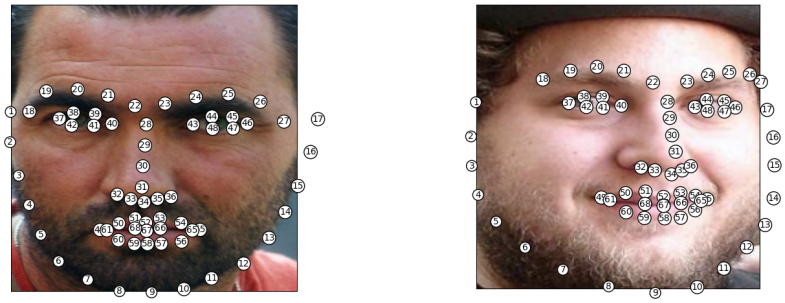
dataset [11, 12, 10]. Many other datasets are available, however, this one is one of the most popular. 300W comes already split into three parts: Test, Validation and Train each composed of $\sim 600, 130, 3700$ respectively. Each sample in the dataset is composed of an image and 68 landmark points on the image. The objective is to train a model to match the given points to the image. Fig. 4 shows two examples.

Applications Facial landmark detection can be applied to a variety of important tasks. For example, *face animation* and *face reenactment* can both exploit the automatic generation of facial landmarks. Other important tasks are *driver status tracking* and *emotion classification* and *face recognition* [6].

Metrics The goal of this task is to predict accurate landmarks on the human face. Therefore, a metric to measure the goodness of a model must take into account this fact. The most popular metric is the *mean error*:

$$ME = \frac{1}{68} \sum_{i=1}^{68} ||y_i - \hat{y}_i||$$

Where $y_i \in \mathbb{R}^2$ represents the prediction of i-th landmark and $\hat{y}_i \in \mathbb{R}^2$ represents the true i-th landmark position. Despite seeming perfectly reasonable as metric ME lacks an important normalization factor. In fact, An error of 10 pixel in $4K$ images can be negligible while the same error on images of size 224×224



(a) Example 1 (b) Example 2

Figure 4: Two example with respective annotations from the 300W dataset

can be considered a poor classification. To overcome this issue a normalization factor is introduced d. While the importance of d is out of discussion there is not a true consensus among which normalization to adopt. In some cases, d is the bounding box dimension, in other cases is the distance between the pupils but, most commonly, it is the distance between the 37-th landmark and 46-th landmark. In this report, we will consider only the latter.

$$NME = \frac{1}{68} \sum_{i=1}^{68} \frac{||y_i - \hat{y}_i||}{d}$$

6 Combining ResNets with Transformers

In this section, we are going to discuss the chosen architecture. Firstly, we consider a pre-trained ResNet on ImageNet. In particular, ResNet101. We proceed by gaining access to different layers of the ResNet:

```
self.resnet  = torchvision.models.resnet101(pretrained=True, progress=True)
self.resnet3 = torch.nn.Sequential(*list(self.resnet.children())[6:8])
self.resnet2 = torch.nn.Sequential(*list(self.resnet.children())[5:6])
self.resnet1 = torch.nn.Sequential(*list(self.resnet.children())[3:5])
self.resnet0 = torch.nn.Sequential(*list(self.resnet.children())[0:3])
...
res0 = self.resnet0(imgs)
res1 = self.resnet1(res0)
res2 = self.resnet2(res1)
res3 = self.resnet3(res2)
```

In the above listing, we split ResNet101 into four steps. These steps give us access to low-level and high-level features (`res0,res1,res2,res3`). These features will be fed to our Transformer Encoder. In particular, the sizes of these features are: imgs $\in \mathbb{R}^{b\times3\times224\times224}$, $\text{res}_0 \in \mathbb{R}^{b\times64\times112\times112}$, $\text{res}_1 \in \mathbb{R}^{b\times256\times56\times56}$, $\text{res}_2 \in \mathbb{R}^{b\times512\times28\times28}$, $\text{res}_3 \in \mathbb{R}^{b\times2048\times7\times7}$. Where b represents the batch size. Next, we interpret these features as images. Thus, we can subdivide these images into patches.

```
from einops import rearrange as r
```

```
res0 = r(res0,"b c (h1 h2) (w1 w2) -> b (c h1 w1) (h2 w2)",h1=2,w1=2)
res1 = r(res1,"b c (h1 h2) (w1 w2) -> b (c h1 w1) (h2 w2)",h1=2,w1=2)
res2 = r(res2,"b c (h1 h2) (w1 w2) -> b (c h1 w1) (h2 w2)",h1=2,w1=2)
res3 = r(res3,"b c h w -> b (h w) c")
```

Now, `res0,res1,res2`, and `res3` contains more channels with linearized patches. Once again, it is worth looking at the shapes of these tensors. $\text{res}_0 \in \mathbb{R}^{b\times 1024\times 784}$, $\text{res}_1 \in \mathbb{R}^{b\times 1024\times 784}$, $\text{res}_2 \in \mathbb{R}^{b\times 2048\times 196}$, $\text{res}_3 \in \mathbb{R}^{b\times 2048\times 49}$. Now, it should not be too hard to interpret these linearized patches as tokens (much like in NLP) and feed them to a Transformer. However, there are two considerations to be made. Firstly, all patch sizes should be the same. To achieve this, we can employ a feed-forward network. Secondly, we have $1024+1024+2048+2048 = 6144$ tokens which require too much memory. Thus, to reduce the memory footprint, we average several tokens together.

```
from einops import reduce as rd
res0 = rd(ffB0(dropout(gelu(ffA0(res0)))),"b e (c1 c2)->b c1 e","max",c2=16)
res1 = rd(ffB1(dropout(gelu(ffA1(res1)))),"b e (c1 c2)->b c1 e","max",c2=8)
res2 = rd(ffB2(dropout(gelu(ffA2(res2)))),"b e (c1 c2)->b c1 e","max",c2=4)
res3 = rd(ffB3(dropout(gelu(ffA3(res2)))),"b e (c1 c2)->b c1 e","max",c2=1)
```

With the previous snippet, we solved the discussed issues. Now, we have a reasonable amount of tokens (441) each one of them having the same size (512). We can concatenate these tokens together and proceed to add the positional embeddings.

```
self.positions = torch.nn.Embedding(441,512).weight.unsqueeze(0)
...
srcs = torch.cat([res0,res1,res2,res3],1)
srcs += self.positions.repeat(batch_size,1,1)
```

At this point, we have prepared the input for the transformer. We feed the prepared patches to the Transformer. Next, we can downscale the first 68 tokens to only *two* features representing the landmark position on the face.

```
from torch.nn import TransformerEncoder
from torch.nn import TransformerEncoderLayer
self.encoder = TransformerEncoder(
                    TransformerEncoderLayer(512,
                        nhead=8,
                        dim_feedforward=2048,
                        activation="gelu",
                        dropout=.1),
                    num_layers=4)
self.downscale = torch.nn.Linear(512,2)
...
mems = self.encoder(srcs.transpose(0,1)).transpose(0,1)
ldmk = self.downscale(mems[:,:68])
```

With this last piece, we have practically defined the entire architecture. The only thing that remained to discuss is the training procedure. We used an *l1 loss* with *Adam* optimizer, *learning rate* set to $1e^{-4}$, and *batch size* of 32.

Since the proposed dataset (300W) is quite small, we introduce three augmentation techniques.

1. random rotations.
2. random crop.
3. random jitter.

7 Related Works

Before jumping in the results, we need to present some other successful techniques for the facial landmark task. 3DDE [18] and CFSS [14] use CNN to fit a 3D face model to the sample image. These techniques are particularly effective with unexpressive faces but fail with more extreme expressions. CNNCRF [1] combines CNN and CRF. Adaloss [13] interprets fiducial points as centers of normal distributions with fixed variance. This approach uses a different CNN and Adaloss to model these distributions. SAN-GT [3] uses Generative Adversarial Networks (GANs) and CNN to generate different styles of input images. These styles are fed to a model to detect landmarks. It is believed that using several styles overcomes the difficulties that a specific style may have.

8 results

In this section, we are going to show the results achieved by the proposed architecture in terms of normalized mean error (NME). Table 1 shows the NME compared to other approaches. As you can see, our model is able to achieve results comparable with the state-of-the-art by achieving an NME of 4.26. Compared to the best approach our architecture falls behind only of 1.13.

Next, we present two samples from the test set (see Fig. 5a and 5b). As you can see, our model is capable of obtaining qualitative landmarks even with fairly complex expressions.

model	NME(%)
3DDE[14]	3.13
CNNCRF[1]	3.30
Adaloss[13]	3.31
SAN-GT[3]	3.98
ours	4.26
CFSS[18]	5.76

Table 1: Results from various approaches for the same task *facial landmarks detection*

9 Conclusion & Future Works

In the last section, we have shown that a transformer can be used to achieve results close to the state-of-the-art for the task of Facial Landmark detection. While the results obtained are encouraging there are still imperfections. Consider figures 5, the predicted fiducial points are quite close to expected ones. However, it is evident that the model is biased towards symmetric mouth expressions. Moreover, often predicted points do not align perfectively with the

(a) Example of prediction using our approach

(b) Second example of prediction using our approach

Figure 5: examples of predictions from the test set.

represeted feature. Of course, many other things that could be done or improved to achieve better results. First of all, introducing more augmentation approaches could be beneficial. Secondly, deeper resnets and deeper transformer encoders could boost the performance. It could be worth trying other strategies to extract information from a resnet to be fed to a transformer.

References

[1] Chen, L., Su, H., and Ji, Q. Deep structured prediction for facial landmark detection. *Advances in Neural Information Processing Systems 32* (2019), 2450–2460.

[2] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[3] Dong, X., Yan, Y., Ouyang, W., and Yang, Y. Style aggregated network for facial landmark detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 379–388.

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

[5] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[6] Khabarlak, K., and Koriashkina, L. Fast facial landmark detection and applications: A survey. *arXiv preprint arXiv:2101.10808* (2021).

[7] Kumra, S., and Kanan, C. Robotic grasp detection using deep convolutional neural networks.

[8] Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913* (2017).

[9] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[10] Sagonas, C., Antonakos, E., Tzimiropoulos, G., Zafeiriou, S., and Pantic, M. 300 faces in-the-wild challenge: Database and results. *Image and vision computing 47* (2016), 3–18.

[11] Sagonas, C., Tzimiropoulos, G., Zafeiriou, S., and Pantic, M. 300 faces in-the-wild challenge: The first facial landmark localization challenge. In *Proceedings of the IEEE International Conference on Computer Vision Workshops* (2013), pp. 397–403.

[12] Sagonas, C., Tzimiropoulos, G., Zafeiriou, S., and Pantic, M. A semi-automatic methodology for facial landmark annotation. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops* (2013), pp. 896–903.

[13] Teixeira, B., Tamersoy, B., Singh, V., and Kapoor, A. Adaloss: Adaptive loss function for landmark localization. *arXiv preprint arXiv:1908.01070* (2019).

[14] Valle, R., Buenaposada, J. M., Valdés, A., and Baumela, L. Face alignment using a 3d deeply-initialized ensemble of regression trees. *Computer Vision and Image Understanding 189* (2019), 102846.

[15] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *arXiv preprint arXiv:1706.03762* (2017).

[16] Wang, W., Xie, E., Li, X., Fan, D.-P., Song, K., Liang, D., Lu, T., Luo, P., and Shao, L. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *arXiv preprint arXiv:2102.12122* (2021).

[17] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., and Le, Q. V. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237* (2019).

[18] Zhu, X., Lei, Z., Liu, X., Shi, H., and Li, S. Z. Face alignment across large poses: A 3d solution. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 146–155.