Lab Vision Systems WS22/23
Video Semantic Segmentation

Siarhei Sheludzko, Patrick Schwemmer, Sebastián Gómez Ruiz

University of Bonn

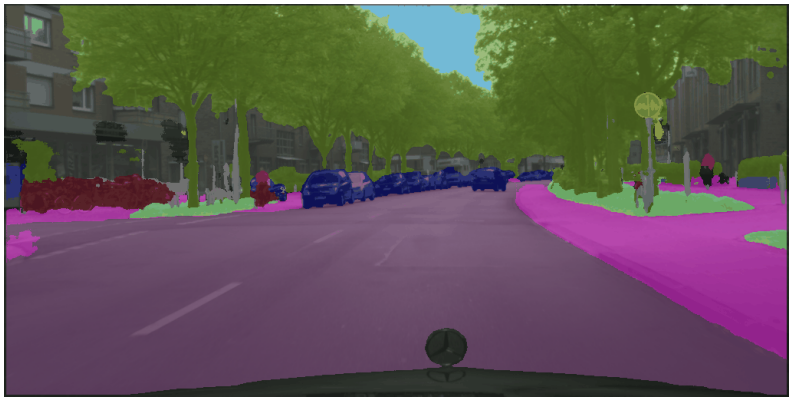


Fig. 1: Segmentation of a street scenery in a city.

Abstract. Semantic segmentation consists in predicting a semantic class for every single pixel in an image, which when transferred to the video domain implies the realization of this task for the different sequence of frames that conform a video. We tackled this task with the usage of a Recurrent UNet-like model in order to obtain advantage of the temporal characteristic of the data. Multiple configurations of the model were implemented and their different results are thus given here, including metrics, visualizations, and a detailed analysis. The training and test were completely executed on the well known Cityscapes dataset.

1 Introduction

Semantic segmentation and video semantic segmentation are dense-prediction vision tasks, which goal is to predict a semantic class for every single pixel of an image, or respectively for every pixel for every frame that forms a video sequence. Its motivation comes from the wide range of applications they can be used at, such as object detection and tracking, which plays an important role for e.g. autonomous driving systems. We focus on video semantic segmentation, which

brings additional challenges in comparison to its non-temporal variant. One of them being temporal consistency, meaning that coherence of the segmentation results of adjacent frames should be present while taking into account the state of an object, such as its motion and appearance. Other difficulties arise directly from objects' movement, such as occlusion or motion blur.

We use an UNet based architecture neural network [9] to tackle this task, which we adapt with convolutional temporal modules in order to take advantage of the temporal nature of the data and overcome the previously mentioned challenges. Multiple variations of the models are implemented, as well as trained and evaluated on the Cityscapes dataset [2], a benchmark large-dataset that provides semantic and pixel-wide annotations for urban street scenery. The different model configurations and results are presented and analyzed in detail in further sections, as well as the process itself. In which we contrast a baseline result obtained by using the model to process the data in a frame by frame manner, with the results obtained by using the model to process the data with the temporal modules in a sequenced (multiple-frames) manner. Analysis is given through the whole method itself and results, for the model design and its evaluation, we also discuss about limitations during the development of the project itself, difficulties and challenges of the process and task as a whole, as well as possible aspects of improvement.

2 Methodology

In this section we describe the methodology used to perform sequential semantic segmentation and evaluate the performance of the proposed methods. We start by giving an overview of the neural network architectures, followed by a description of the loss function used for training and the evaluation metrics.

2.1 Neural Network Architecture

In this chapter we will describe the neural network architecture and its components used to tackle the task of sequential semantic segmentation.

Model Architectures In our approach we extended a UNet architecture [9] by incorporating convolutional recurrent neural network units (cRNN) [7] and convolutional gated recurrent units (cGRU) [4] as temporal units. Our proposed architecture can be decomposed into multiple encoding and decoding layers, each consisting of multiple typical convolutional block that include 2D-Convolutions, batch-normalization, an activation function, each encoder block including down-sampling done with a convolution and each decoder block including up-sampling done with a transposed convolution. At the end of each encoder block we include one of the convolutional temporal cells, either cRNN or cGRU to capture the temporal dependencies in the image data, while at the decoder blocks we concatenate with the hidden state given by either cRNN or cGRU.

To further evaluate a UNet-like architecture in this setting we chose to adapt the UNet architecture in such a way that the encoder and decoder blocks can be easily replaced by other convolutional bocks such as suggested by the ResNet architecture [6], or the ConvNeXt architecture [8].
The general model diagram is presented in figure 2, for the specific values of the multiple tested configurations refer to the experimental details 3.

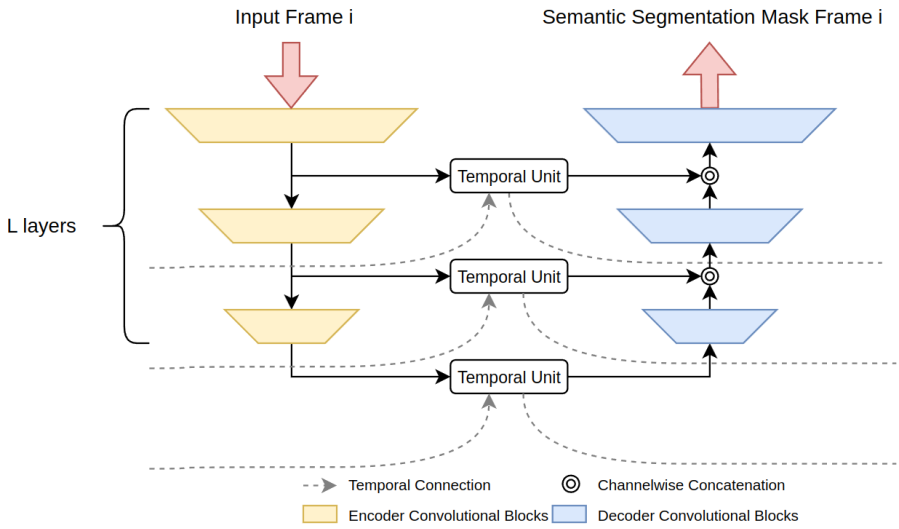


Fig. 2: The figure presents the template of the temporal UNet architecture. The skip connections are replaced by a temporal unit, which functions as replacement for the concatenation input and to capture sequential information.

Our neural network architecture components can be replaced by a total of two temporal cells with cRNN and cGRU and three encoder and decoder block types, being the vanilla blocks suggested by UNet [9], ResNet blocks [6] and ConvNeXt blocks [8]. The resulting architecture aims to be more capable of capturing both spatial and temporal information in sequential data.
The output of the network is a probability map, where each pixel is assigned to a probability value for a given class.

Convolutional Temporal Units Temporal Units like RNN units [3] and GRU units [1] are a class of neural network units that have been designed to capture the temporal dependencies and general temporal information of sequential data e.g. videos or audio data such as speech, and have shown to be effective in many

tasks of sequential data processing.

The mathematical equation for the convolutional units for both RNN and GRUs follow a similar structure as their non convolutional version, but with an addition of convolutional operations in their recurrent connection. To show how they are modified the following sections starts by stating the standard equations of RNN and GRU, followed by equations that show how they are modified to include convolutional operations.

Convolutional RNN To describe the exact operations we start with the equation of the RNN unit and the equation of the convolutional RNN unit:

$$h^{(t)} = f(W_x x^{(t)} + W_h h^{(t-1)} + b_h) \tag{1}$$
$$y^{(t)} = f(W_y h^{(t)} + b_y) \tag{2}$$

Where h_t denotes the hidden state at a time step t, x_t is the input at time step t, W_x, W_h are learned weight matrices,f is a generic activation function, $y^{(t)}$ is the output of that RNN unit, with b_h, b_y being the bias vectors of the hidden state and the output. The cRNN [7] which we adjusted us can be mathematically described as follows:

$$y^{(t)} = f(C_x * x^{(t)} + C_h * y^{(t-1)}) \tag{3}$$

C_x, C_h in equation 3 denote the convolutions applied on the input $x^{(t)}$ and $y^{(t)}$. One key difference to the conventional RNN is that $y_{(t)}$ is both the output and the hidden state of this cRNN unit. This adaption is made to have a lightweight version of an RNN without a third convolution replacing W_y while still keeping the aspect of temporal grasps.

Convolutional GRU To contrast the way we implemented the convolutional GRU Unit and the conventional GRU unit we first begin to state the equations for the typical GRU unit.

$$r^{(t)} = \sigma(W_{xr} x^{(t)} + b_{xr} + W_{hr} h^{(t-1)} + b_{hr}) \tag{4}$$
$$z^{(t)} = \sigma(W_{xz} x^{(t)} + b_{xz} + W_{hz} h^{(t-1)} + b_{hz}) \tag{5}$$
$$n^{(t)} = tanh(W_{xn} x^{(t)} + b_{xn} + r^{(t)} \cdot (W_{hn} h^{(t-1)} + b_{hn})) \tag{6}$$
$$h^{(t)} = (1 - z^{(t)}) \cdot n^{(t)} + z^{(t)} \cdot h^{(t-1)} \tag{7}$$

In equation 7 $\cdot$ denotes the Hadamard product, which is element-wise multiplication. W_{lc} is the weight matrices and b_{lc} is the biases of a layer l and

component c. The different components of a GRU unit are a reset gate $r^{(t)}$, an update gate $z^{(t)}$, the candidate hidden layer $n^{(t)}$ and the hidden layer $h^{(t)}$. The proposed convolutional GRU by [4] is as follows:

$$r^{(t)} = \sigma(C_r * (x^{(t)} \odot h^{(t-1)})) \tag{8}$$
$$z^{(t)} = \sigma(C_z * (x^{(t)} \odot h^{(t-1)})) \tag{9}$$
$$n^{(t)} = tanh(C_n * (x^{(t)} \odot (r^{(t)} \cdot h^{(t-1)}))) \tag{10}$$
$$h^{(t)} = (1 - z^{(t)}) \cdot n^{(t)} + z^{(t)} \cdot h^{(t-1)} \tag{11}$$

C_r, C_z, C_n denote the convolutions of the according components. For the input of the given convolution in each component concatenation of the prior hidden state and the input is used . The concatenation operation that concatenates two matrices is denoted as $\odot$.

For a parameter description of the convolutions and activation functions used we refer to the appendix A.2.

2.2 Convolutional Blocks

We evaluate three types of convolutional blocks that replace the convolutional blocks implemented in the UNet structure. The three types of convolutional blocks we chose are the original UNet convolutional blocks by [9], residual convolutional blocks [6] and lastly convolutional blocks given by the ConvNeXt architecture [8] which are optimized for the UNet structure proposed by [5]. For more detail regarding the convolutional blocks we refer to the appendix A.1.

2.3 Loss Function

To measure the difference between predicted output and true output of a multiclass segmentation task the loss function we used is the cross entropy loss. Cross entropy loss is a popular choice for multiclass classification problems as it measures the dissimilarity between the predicted probability distribution and the true probability distribution for each pixel. In our case the probability distribution is a probability map, where each pixel is assigned to a probability value for each class. The ground truth probability distribution is a one-hot encoded map, where each pixel is assigned a label corresponding to the ground truth class.

Cross entropy loss penalizes the predicted map if it assigns a low probability to the correct classes and a high probability to the incorrect classes. It can be computed as follows:

$$\mathcal{L}_{CE} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} log(p_{i,c}) \tag{12}$$

where N is the number of pixels given, C is the number of classes, $y_{i,c}$ is the ground truth label for a pixel i and class c, and $p_{i,c}$ is the predicted probability for the same pixel i and class c.

2.4 Evaluation Metrics

In order to asses the performance of the different combination of models proposed by us we chose to adapt mean accuracy (mAcc) and mean intersection over union (mIoU) as a metric to evaluate on a validation set.
Accuracy is commonly used as a metric to measure the percentage of correctly classified pixels in the predicted segmentation map. The accuracy and the mean accuracy is computed as:

$$\text{Accuracy}_c = \frac{TP_c + TN_c}{TP_c + TN_c + FP_c + FN_c} \tag{13}$$

$$\text{Mean Accuracy} = \frac{1}{C} \sum_{c=1}^{C} \text{Accuracy}_c \tag{14}$$

Intersection over Union (IoU) is widely used as evaluation metric for semantic segmentation. It measures the overlap between predicted segmentation and the ground truth segmentation. In contrast to accuracy IoU takes spatial relationships of pixels into account and therefore is more informative than accuracy for evaluation of semantic segmentation models. Similarly mIoU can be computed from a confusion matrix as follows:

$$\text{Mean IoU} = \frac{1}{C} \sum_{c=1}^{C} \frac{TP_c}{TP_c + FP_c + FN_c} \tag{15}$$

3 Experimental Details

In this section we describe multiple model configurations that are combinations of the components explained in 2.1. These each also vary in dimensions of the convolution blocks and convolutional temporal cells. We present three configurations of the UNet size and show their dimensionalities. Lastly we discuss the spatial and temporal augmentations used for the cityscapes data set and image resolution chosen.

3.1 Model Configurations

We explored several model configurations for two primary reasons. Firstly, we aimed to test the capabilities of the model with various parameters and settings. Secondly, we faced hardware limitations as we could only utilize machines

equipped with RTX 2080Ti GPUs of 11 GiB and GTX TITAN X Pascal of 12GiB memory capacity.

Initially we trained a model, which consisted of a 3 layer deep UNet with the encoder having 3 blocks the decoder also having 3 blocks respectively, without temporal units and with the bigger size feature maps of all the tested models. This model is what we call our baseline model with original size. Then we propose two different models of smaller (small) feature channels, one with 3 layers as well (shallow) and another with 4 layers (deep). For both of these models three different convolutional blocks configurations were available, those being Vanilla, ResNet18 based, and ConvNext based, while also having the two different temporal units possibilities mentioned previously, cRNN, and cGRU. The convolutional blocks dimensionalities are shown in table 1 As observable the decoder blocks initial channels are double the output of the last encoder block, that is due to the concatenation of the result given by the temporal unit.

Convolution Blocks		**Number of Channels**		
Block	**Convolution Size**	**Original**	**Small Shallow**	**Small Deep**
1	3×3	-	-	16
	3×3	-	-	16
2	3×3	64	16	32
	3×3	64	16	32
3	3×3	128	32	64
	3×3	128	32	64
4	3×3	256	64	128
	3×3	256	64	128
5	3×3	512	128	256
	3×3	256	64	128
6	3×3	256	64	128
	3×3	128	32	64
7	3×3	128	32	64
	3×3	64	16	32
8	3×3	64	16	32
	3×3	n	n	16
9	3×3	-	-	16
		-	-	n

Table 1: A table of the UNet architecture dimensions. The configuration and their sizes are shown as well as the convolution kernel sizes used. The original and small shallow size is 3 Layers deep and small deep size is 4 layers deep. Block 1-4 are encoder blocks and Block 4-8 are decoder blocks and the last block is the encoding to a probability map. n is the number of classes with each class having a probability map.

3.2 Cityscapes dataset

The Cityscapes dataset [2] is a large-scaled dataset for semantic understanding. It provides dense semantic segmentation annotations for images of urban scenery in Germany, consisting of 20000 images with coarse annotations, and 5000 images with fine annotations. The images correspond to video sequences each consisting of a total of 30 frames, with the semantic mask ground truth annotation given for the 20th frame of the sequence. The annotations are provided for a total of 30 classes, however only 19 classes are actually used, and one extra class is setup for the non-used classes, yielding a total of 20 possible labels.

3.3 Data Augmentation

Different possibilities of forming the L-length sequences were possible, such as taking for example the ground truth frame as the ending of it, thus taking the sequence from the frame number $(20 - L)$ to the 20th one. Nevertheless, in order to have more possible different sequences, we opted for allowing the ground truth frame to be in any possible position of the sequence, giving a total of L possible sequence variations for every original given sequence. The position of the ground truth frame within the shortened sequence is randomly selected by the loader every time the full original sequence is obtained by the data loader. This mechanism is the temporal data augmentation method we implemented. On the other hand, for spatial data augmentations, we incorporated random crop and horizontal flipping, both for the reason of data augmentation in itself, but also by having in mind the generalization of the model for different resizing resolutions of the given image input.

3.4 Training configuration

During the development of the whole process we use the fine grained annotation images, from which we take a total of 2975 frames images for the training set, and a total of 500 images for the validation set. For the training stage sequences of length 5 were provided, while for the validation stage a sequence length of 12, in both cases with a frame resolution of (512×1024). In the implementation itself, it was decided to keep always the frame corresponding to the ground truth label in the given sequence, thus the 20th frame was always part of the sequence given as input to the model.
Each model was trained until convergence with a maximum of 100 epochs. The temporal models were fed a sequence of 5 images for training and a sequence of 12 images for evaluation. In contrast to the temporal models, the baseline models were fed a single image and its respective ground truth image for both training and evaluation. By only feeding the baseline 1 image at a time we could increase the baseline models size, while we simultaneously had to decrease model sizes to compensate for temporal models.
For each model we used an Adam optimizer with a learning rate of 0.0003

(3×10^{-4}), while the default arguments were kept for the other optimizer parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-8}$, and weight decay set to 0.
To not run intro problems with single batch learning we later implemented gradient accumulation, although prior models seemed to not be limited by single step learning in terms of performance and loss minimization.

4 Evaluation

In this section we present our results and evaluate the quality of the models. Specifically, we compare the performance of different temporal UNet models and aim to analyze evaluation metrics discussed in 2.4 as well as evaluate architectural choices such as model depth and model size. We also provide analysis of experimental findings of predicted segmentation sequences.

	Block	Size name	Temporal Unit	Num. Params	mIoU%	mAcc%
Base models	Vanilla	Original	-	1,958,996	**46.40**	**87.36**
	Vanilla	Small shallow	-	125,732	36.72	84.45
	Vanilla	Small deep	-	507,940	41.17	86.83
Temporal models	Vanilla	Small shallow	cRNN	222,724	37.39	84.60
	Vanilla	Small shallow	cGRU	416,372	37.06	84.24
	ResNet	Small shallow	cGRU	423,556	40.74	86.43
	ResNet	Small shallow	cRNN	229,908	37.19	84.96
	ResNet	Small deep	cRNN	760,420	40.02	85.03
	ConvNext	Small deep	cGRU	1,544,020	39.00	86.31
	ResNet	Small deep	cGRU	1,707,460	**42.43**	**87.48**

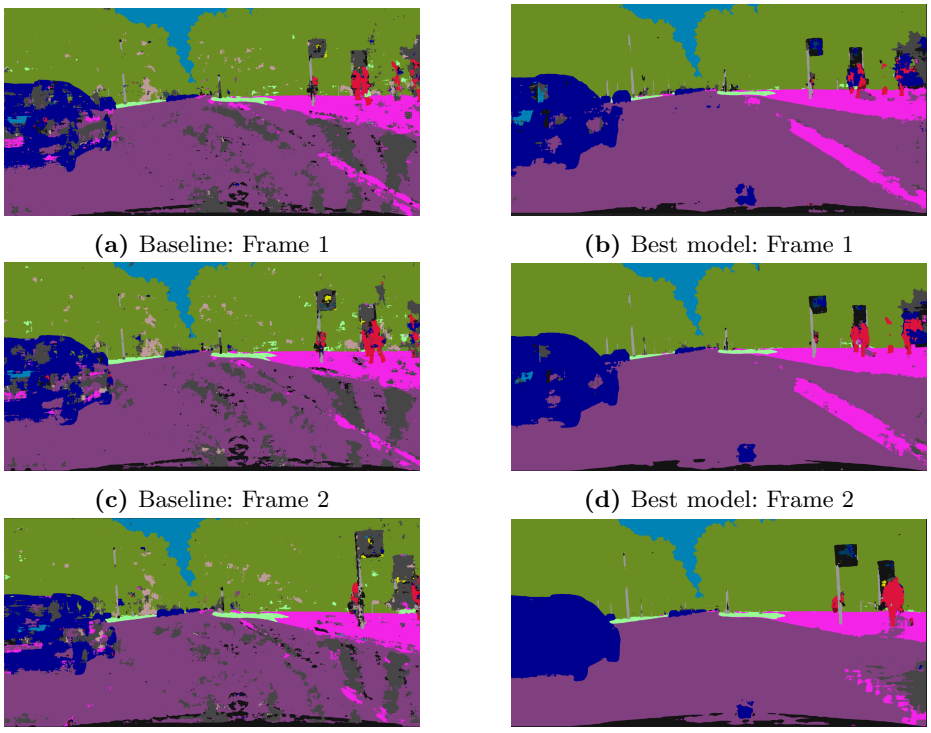
Table 2: The table shows performance comparison of temporal UNet-based video sequence classification models with varying block types, sizes, and temporal units. The two best models are the UNet baseline model and the ResNet cGRU deep shallow model.

4.1 Runtime

The training time of a model heavily depended on its number of parameters,due to its architecture, layer size, the given image resolution and also highly impacted by the temporal module, with cGRU being way more computationally expensive. Baseline models were the fastest, taking in between around 10 and 20 minutes per epoch, because they were fed only 1 image and its respective ground truth at a time. Vanilla temporal models have a training time of about 50 minutes per epoch, and Resnet based models have a training time of about 80 and 130 minutes per epoch.

	Model Parameters		
Model	**Small Shallow**	**Small Deep**	**Original**
Vanilla UNET	125,732	507,940	1,958,996
Vanilla cRNN	222,724	900,100	3,508,180
Vanilla cGRU	416,372	1,683,700	6,605,204
ResNet cRNN	229,908	923,860	3,598,356
ResNet cGRU	423,556	1,707,460	6,695,380
ConvNeXt cRNN	-	760,420	2,933,588
ConvNeXt cGRU	-	1,544,020	6,030,612

Table 3: This table shows the number of parameters of each configuration. Increasing the sizes and dimensions linearly lead to more parameters, noticeable is the huge parameter increase of cGRU cells when compared to cRNN cells.



(a) Baseline: Frame 1 **(b)** Best model: Frame 1

(c) Baseline: Frame 2 **(d)** Best model: Frame 2

(e) Baseline: Last frame **(f)** Best model: Last frame

Fig. 3: Semantic segmentation comparison of vanilla original baseline (a, c, e) against the best obtained model, ResNet cGRU small deep (b, d, f). We can see that the

4.2 Performance Analysis

While accuracy is a simple and intuitive metric, it does not take into account the spatial relationships between pixels and may not be the best metric for evaluating semantic segmentation models. On the other hand, IoU takes into account the spatial relationships between pixels and is a more informative metric than accuracy for evaluating semantic segmentation models.

As we can observe in table 2, the model in which we obtain the best mIoU corresponds to the one with ResNet based convolutional blocks, with a small deep size, and with convolutional GRU as the temporal units, achieving a 42.43% mIoU and a 87.48% mAcc. Clearly it does not beat the baseline model with original sizes, but its baseline counterpart by a total of 1.26% increase in mIoU.

This result leads us to believe that the temporal model with original sizes would outperform the respective baseline, unfortunately due to resource limitations the training of such model was not possible.

On the other hand, even though the baseline model with original sizes obtains better quantitative metrics, when we compare the results in a frame by frame manner the difference made by the temporal module is noticeable in the segmentation results. We can observe this in figure 3, noticing that through the progression of the sequence the segmentation results improve for the temporal model, which is visible for the second frame, and even more for the last frame of the sequence, while they seem to remain low quality for the baseline frame by frame model. The baseline model outputs heavily artifacts for the segmentation mask. Here we can see the importance of integrating temporal awareness into the model.

4.3 Model Sizes

Judging from the table 2 we can see that the model with the highest mIoU and mAcc metrics is the ResNet small deep cGru model. This suggest that increasing the number of parameters can improve performance. However we observe from table 3 that the number of parameters is not a linear relationship and that using cGRU cells lead to a much larger increase in parameters than the cRNN cells. Besides the two best models, baseline original size and the ResNet cGRU small deep, smaller models seem to be decently performing e.g. ResNet cRNN small deep and vanilla small deep baseline. In conclusion we observe a trade off between performance and parameter size when the model size can not exceed a threshold.

5 Future Work

Due to limitations in machines and GPU memory we had to compromise in some aspects like training larger model sizes or trying bigger architectures. Therefore

future work can include training larger models to really filter out the capabilities of the different components. Improvements for the models were also not realized because we wanted to preserve comparable results of all training runs, including runs that were computed before new features were realized, hence we were just limited to implementing smoothing the loss with e.g. gradient accumulation or increasing efficiency. These improvements can includes to add additional temporal augmentation such as frame skipping and add temporal regularization by integrating a loss that penalizes non-smooth changes between the predicted sequence segmentation mask.
Furthermore future work is using advanced segmentations like mixup to see if it can boost the quality of a model. Lastly including pre-training on a dataset such as coco may improve model quality.

6 Conclusion

Concluding our report we overcame hurdles of memory capacity, not being able to pretrain our model and designed our architecture accordingly. Our designed model beat the baseline, and with that we reach as well the goal of the project. Even though the model and training can be greatly improved, we observe how taking advantage of the temporal qualities of the data by adapting the model with recurrent modules yields better results, especially visual results, as the ones produced in a frame by frame manner, improving significantly the temporal coherency.
We found out that even though IoU can be high e.g. for our baseline model the sequence prediction can lead to heavy artifacts because of a lack of temporal awareness. Ultimately a trade-off between performance and parameter size needs to be taken into account when including temporal units that grant an architecture temporal awareness.

References

1. Nicolas Ballas, Li Yao, Chris Pal, and Aaron C. Courville. Delving deeper into convolutional networks for learning video representations. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.
2. Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding, 2016.
3. Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.
4. Nelly Elsayed, Anthony S. Maida, and Magdy A. Bayoumi. Deep gated recurrent and convolutional network hybrid model for univariate time series classification. *CoRR*, abs/1812.07683, 2018.

5. Zhimeng Han, Muwei Jian, and Gai-Ge Wang. Convunext: An efficient convolution neural network for medical image segmentation. *Knowledge-Based Systems*, 253:109512, 2022.
6. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.
7. Gil Keren and Björn Schuller. Convolutional rnn: an enhanced model for extracting features from sequential data, 2017.
8. Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s, 2022.
9. Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

A Appendix

A.1 Encoder-Decoder-Implementation

The encoder and decoder parts are visible in 5. Parameters for the modules can be read from the figure. After each encoder block downsampling is applied by a convolution with kernel 2×2 and stride 2. The upsampling on the other side is applied before a decoder block, using a Transposed 2D convolution with kernel 2×2 and stride 2.
The convolution blocks are implemented as visible in 5. The blocks of the encoder and decoder are completely replaceable by two of the other block types in 5.

A.2 Temporal Units Details

The convolutions C_x, C_h of the cRNN use a 3×3 kernel with stride $= 1$ and padding $= 1$. Both convolutions have the same input dimensions and output dimensions, the input dimensions being the output dimensions of each encoder block respectively.
For the cGRU the convolutions C_r, C_z are computed as one convolutionC_{rz} of added input and hidden size. The output dimension is twice the hidden dimension s.t. C_{rz} can be split into both $r^{(t)}, z^{(t)}$ respectively. C_n is computing as stated in the equations. Equivalent to cRNN the convolutions have 3×3 kernel with stride $= 1$ and padding $= 1$.

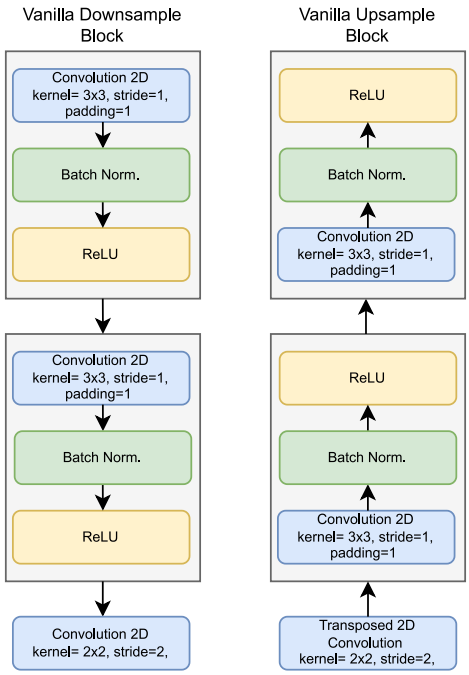


Fig. 4: Encoder block on the left and the Decoder block on the right hand side. Both blocks are of vanilla type

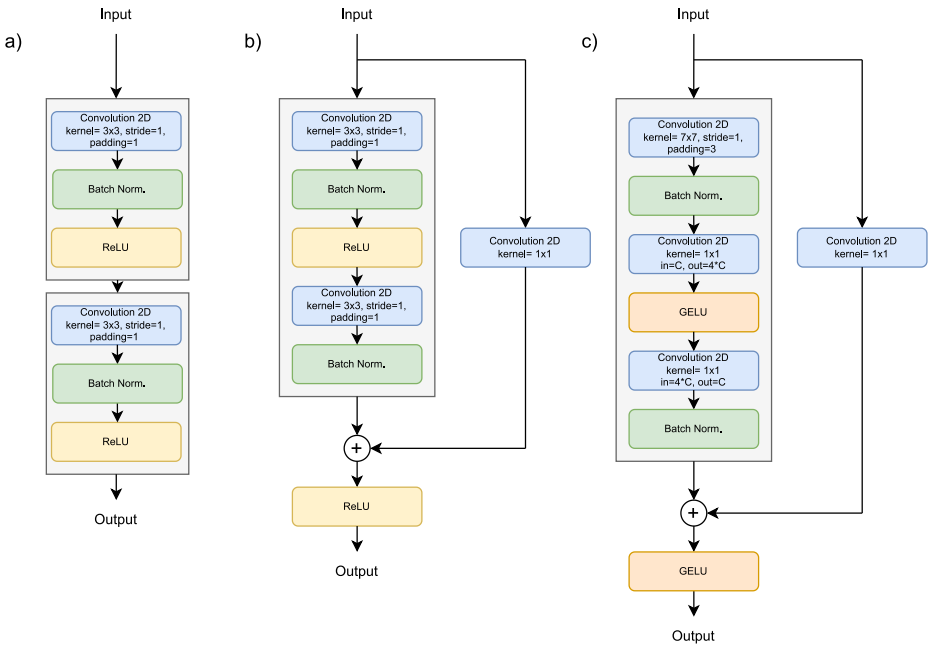


Fig. 5: This figure show all three types of convolution blocks. a) is the vanilla block, b) is the ResNet block, c) is the ConvNeXt block.