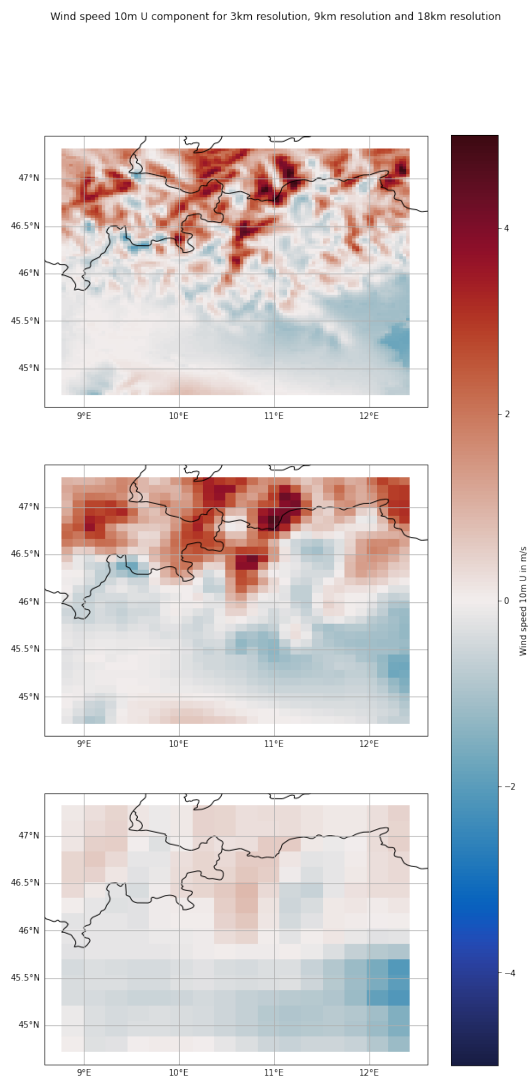
Advanced Topics in Climate Dynamics
Project 2

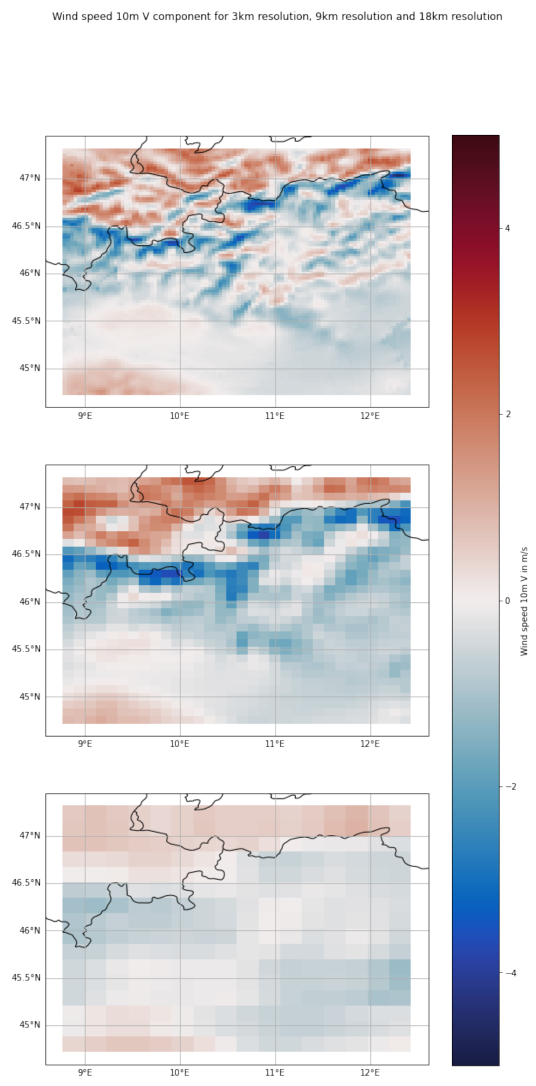
L. Vulto
4685784
l.vulto@students.uu.nl

June 29, 2023

(a) The data can be seen in Figure 1 and 2. The HR data is the first picture from above, the MR data is the middle picture and the LR data is the lowest picture. From Figure 1 we can observe wind foëhns in the V field, southerly winds in the south of the alps and northerly winds in the north of the alps Miralles et al., 2022. In the lower resolution this gets less clear.


Wind speed 10m U component for 3km resolution, 9km resolution and 18km resolution
47°N
46.5°N
46°N
45.5°N
45°N
9°E
10°E
11°E
12°E
Wind speed 10m U in m/s
4
2
0
−2
−4

(a) Zonal wind speed


Wind speed 10m V component for 3km resolution, 9km resolution and 18km resolution
47°N
46.5°N
46°N
45.5°N
45°N
9°E
10°E
11°E
12°E
Wind speed 10m V in m/s
4
2
0
−2
−4

(b) Meridional wind speed

Figure 1: From top to bottom: wind speed for 3kmx3km resolution, 9kmx9km resolution and 18kmx18km resolution

Furthermore, in Figure 2 we see higher temperatures in the valley, and lower temperatures in the Alps. This is also what we expected, since the Alps are quite high in altitude. The temperature gradient is quite well conserved also in lower resolution, but the valleys can only really be seen on the High resolution image, and somewhat in the middle resolution.

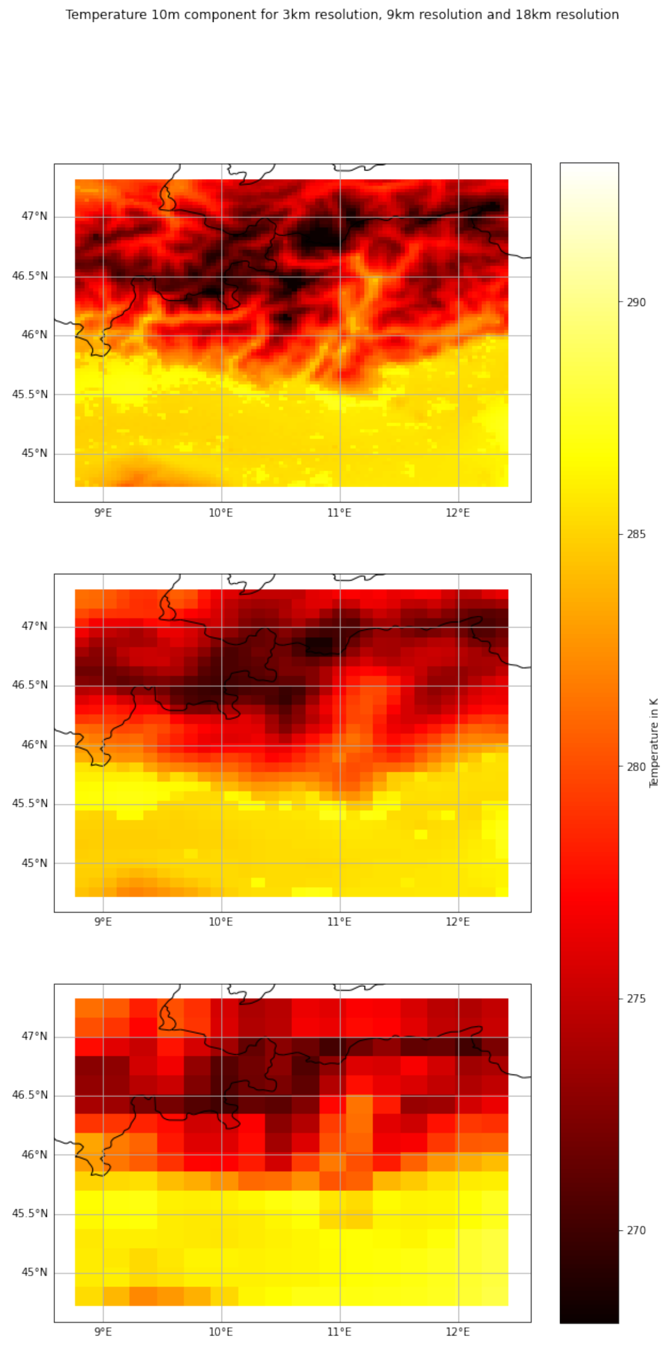

Temperature 10m component for 3km resolution, 9km resolution and 18km resolution
47°N
46.5°N
46°N
45.5°N
45°N
9°E
10°E
11°E
12°E
290
285
280
275
270
Temperature in K

Figure 2: Mean T_{2m} for HR, MR, and LR data.

(b) In Figure 3 the probability density distribution is plotted. The temperature distribution agree quite okay, but the RMSE of the wind especially in the V field is higher. Note that the RMSE especially for the wind doesn't show too much, since the lower resolution can just be the average.

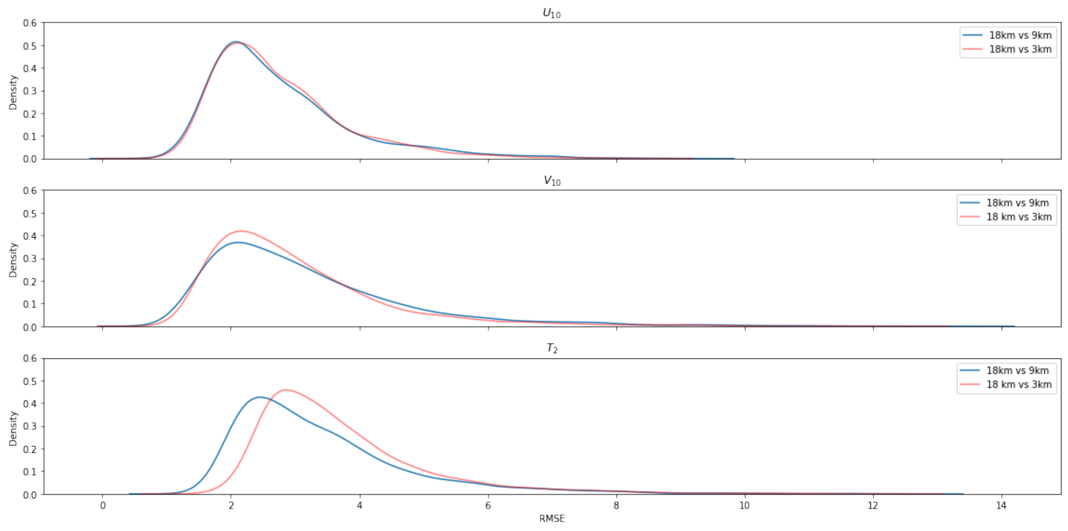
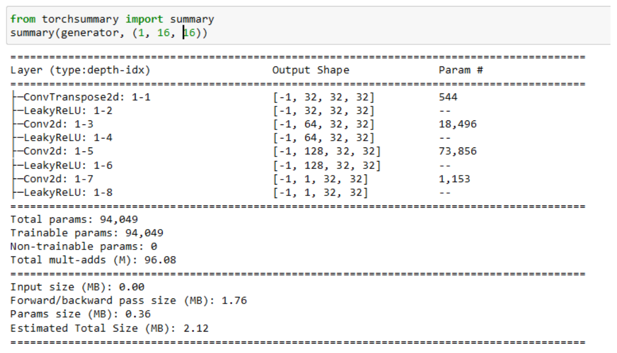


Figure 3: RMSE of the training data(18km x 18km) versus both the MR 9km x 9km data and training data vs HR 3km x 3km data.

These distributions are a baseline for what would happen if we would interpolate the data from a coarse resolution to a higher resolution. This, since we regridded the coarse resolution to a higher resolution via the helperfunction file. Therefore these are the scores for the network to beat.

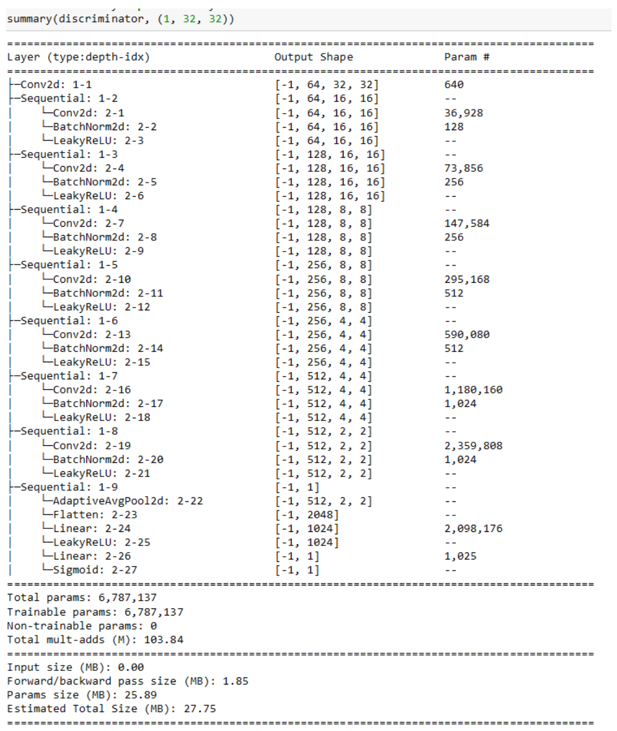
(c) The layers of the generator and the discriminator can be seen in Figure 4.



```
from torchsummary import summary
summary(generator, (1, 16, 16))
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─ConvTranspose2d: 1-1                   [-1, 32, 32, 32]          544
├─LeakyReLU: 1-2                         [-1, 32, 32, 32]          --
├─Conv2d: 1-3                            [-1, 64, 32, 32]          18,496
├─LeakyReLU: 1-4                         [-1, 64, 32, 32]          --
├─Conv2d: 1-5                            [-1, 128, 32, 32]         73,856
├─LeakyReLU: 1-6                         [-1, 128, 32, 32]         --
├─Conv2d: 1-7                            [-1, 1, 32, 32]           1,153
├─LeakyReLU: 1-8                         [-1, 1, 32, 32]           --
==========================================================================================
Total params: 94,049
Trainable params: 94,049
Non-trainable params: 0
Total mult-adds (M): 96.08
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 1.76
Params size (MB): 0.36
Estimated Total Size (MB): 2.12
==========================================================================================
```

(a) Generator layout



```
summary(discriminator, (1, 32, 32))
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Conv2d: 1-1                            [-1, 64, 32, 32]          640
├─Sequential: 1-2                        [-1, 64, 16, 16]          --
|    └─Conv2d: 2-1                       [-1, 64, 16, 16]          36,928
|    └─BatchNorm2d: 2-2                  [-1, 64, 16, 16]          128
|    └─LeakyReLU: 2-3                    [-1, 64, 16, 16]          --
├─Sequential: 1-3                        [-1, 128, 16, 16]         --
|    └─Conv2d: 2-4                       [-1, 128, 16, 16]         73,856
|    └─BatchNorm2d: 2-5                  [-1, 128, 16, 16]         256
|    └─LeakyReLU: 2-6                    [-1, 128, 16, 16]         --
├─Sequential: 1-4                        [-1, 128, 8, 8]           --
|    └─Conv2d: 2-7                       [-1, 128, 8, 8]           147,584
|    └─BatchNorm2d: 2-8                  [-1, 128, 8, 8]           256
|    └─LeakyReLU: 2-9                    [-1, 128, 8, 8]           --
├─Sequential: 1-5                        [-1, 256, 8, 8]           --
|    └─Conv2d: 2-10                      [-1, 256, 8, 8]           295,168
|    └─BatchNorm2d: 2-11                 [-1, 256, 8, 8]           512
|    └─LeakyReLU: 2-12                   [-1, 256, 8, 8]           --
├─Sequential: 1-6                        [-1, 256, 4, 4]           --
|    └─Conv2d: 2-13                      [-1, 256, 4, 4]           590,080
|    └─BatchNorm2d: 2-14                 [-1, 256, 4, 4]           512
|    └─LeakyReLU: 2-15                   [-1, 256, 4, 4]           --
├─Sequential: 1-7                        [-1, 512, 4, 4]           --
|    └─Conv2d: 2-16                      [-1, 512, 4, 4]           1,180,160
|    └─BatchNorm2d: 2-17                 [-1, 512, 4, 4]           1,024
|    └─LeakyReLU: 2-18                   [-1, 512, 4, 4]           --
├─Sequential: 1-8                        [-1, 512, 2, 2]           --
|    └─Conv2d: 2-19                      [-1, 512, 2, 2]           2,359,808
|    └─BatchNorm2d: 2-20                 [-1, 512, 2, 2]           1,024
|    └─LeakyReLU: 2-21                   [-1, 512, 2, 2]           --
├─Sequential: 1-9                        [-1, 1]                   --
|    └─AdaptiveAvgPool2d: 2-22           [-1, 512, 2, 2]           --
|    └─Flatten: 2-23                     [-1, 2048]                --
|    └─Linear: 2-24                      [-1, 1024]                2,098,176
|    └─LeakyReLU: 2-25                   [-1, 1024]                --
|    └─Linear: 2-26                      [-1, 1]                   1,025
|    └─Sigmoid: 2-27                     [-1, 1]                   --
==========================================================================================
Total params: 6,787,137
Trainable params: 6,787,137
Non-trainable params: 0
Total mult-adds (M): 103.84
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 1.85
Params size (MB): 25.89
Estimated Total Size (MB): 27.75
==========================================================================================
```

(b) Discriminator layout

Figure 4: Structure of the generator and discriminator used.

Generator

The input of the generator is a 3D tensor with shape (N_p, S, S). N_p = number of predictors (which is 1), S is patch size. The output is a 3D tensor with double the resolution, as can be seen in Figure 4.

For the generator, a MSELoss function is used. As can be seen in Figure 4, 1 Convolutional transpose layer is used and 3 convolutional layers. Furthermore, the LeakeReLU activation function is used as activation function. The inspiration for this was Stengel et al., 2020.

Discriminator

The input of the discriminator is a high-resolution input tensor in shape (N_p, S, S). The discriminator looks if the input is a good match and if the high-resolution temperature field is from the generator or the 9km input field. The output of the discriminator is $(N_p x 1)$. For the discriminator, a Binary Cross Entropy loss function is used. Because we use the BCE Loss function, we can use a Sigmoid activation function in the fully connected layer.

Training

For the training, for computational purposes $\frac{1}{4}$th of the data is used. If the whole dataset was used, probably better results would have been achieved.

The hyperparameters that were used for training were $\alpha = 0.1$, $n_{epoch} = 100$, $l_g = 1 \times 10^{-3}$ and $l_d = 1 \times 10^{-4}$. The trainig was done in batches of $b = 64$. These hyperparameters yielded the best result for me.

First, the network is pretrained for 5 epochs. This is to improve the generator before the discriminator gets too good. This choice was made because in the beginning the discrimnator was trained way too quickly.

Also, following Stengel et al., 2020, a dynamical layer has been implemented that trains the generator more often than the discriminator if the discriminator trains too quickly for the generator. This makes sure the generator is trained well enough.

The adverserial loss is bigger in this research than in that of Stengel et al., 2020. The adverserial loss is the captures the ability of the generator network to fool the discriminator. This loss for my network thus was more important to obtain better results, and I hypothesise that this is because my generator is not as good as Stengel et al., 2020 uses. Therefore, by increasing α the network is rewarded extra if the adverserial loss decreases. The generator has a higher learning rate than the discriminator because the generator needed to be trained a lot for good results. This choice yielded the best results. 100 epochs were used because it looked like the loss wasn't decreasing anymore (Figure 9).

(d) See figure 5 for the RMSE loss. The network seems to perform a bit better than the baseline in b). Though, this is only looking at the RMSE of the average wind, not looking at the pixels. For the downscaled temperature patterns in 2021 see Figures 6, 7,8.

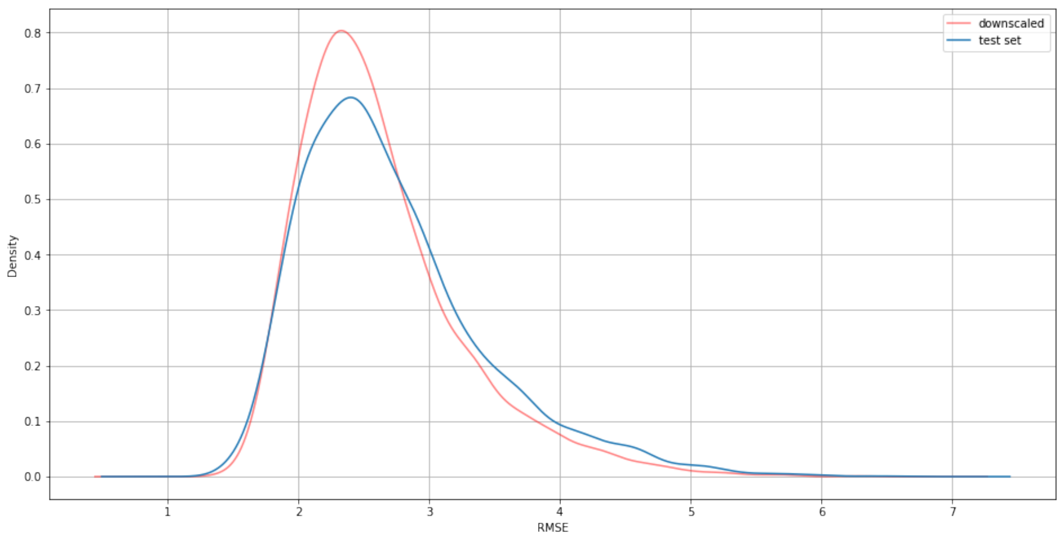


Figure 5: RMSE for SRGAN generated image versus RMSE of 18km vs 9km on the test dataset.

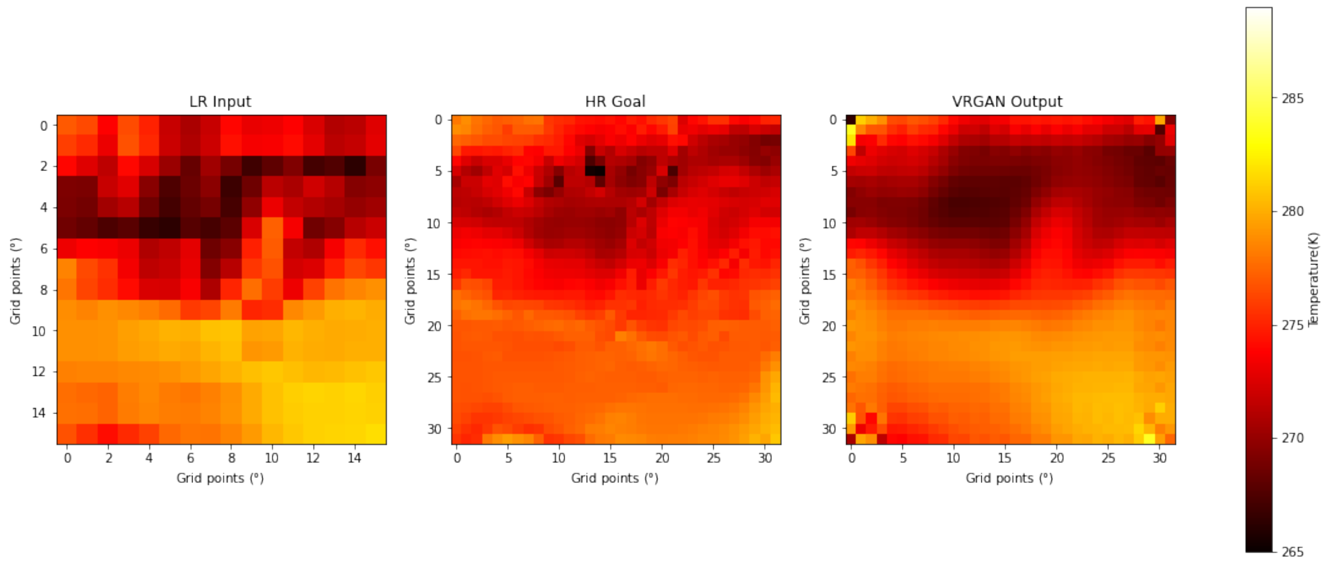


Figure 6: Input, target & Output for SRGAN LR $\rightarrow$ MR

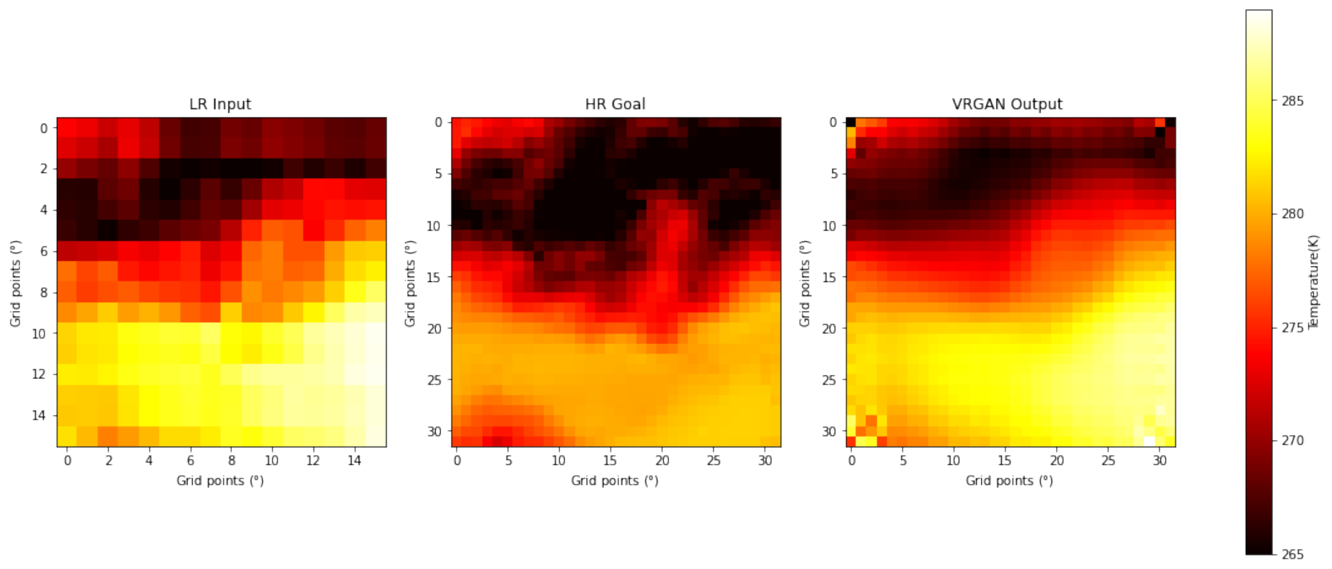


Figure 7: Input, target & Output for SRGAN LR $\longrightarrow$ MR

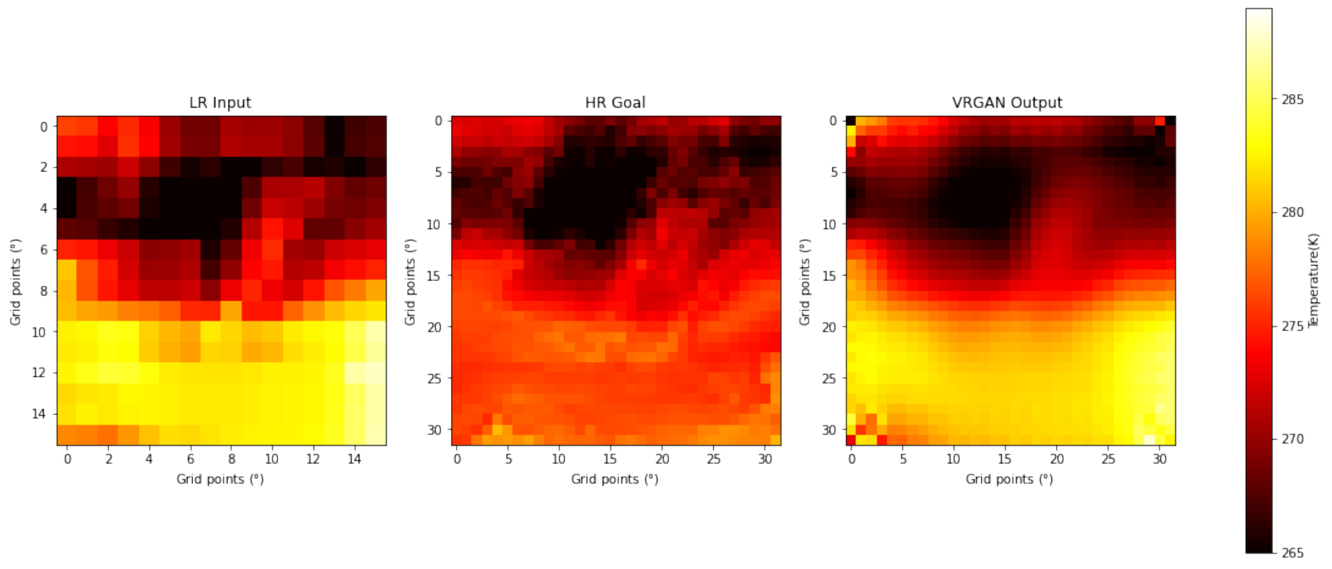
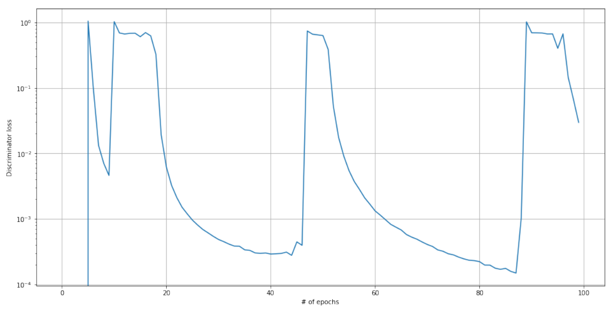


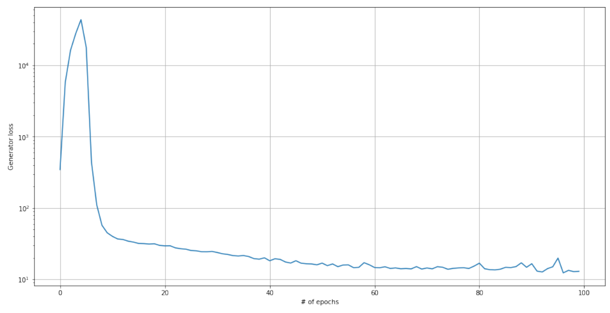
Figure 8: Input, target & Output for SRGAN LR $\longrightarrow$ MR

For Figures 6, 7,8 the left figure is the LR NCEP9 Input, the middle picture is the HR WRF9km Goal, and the right picture is the SRGAN output.

It can be seen that the SRGAN is pretty good at recognising the lakes and the mountains, but this can also be distinguished from the LR input data.



(a) Generator loss



(b) Discriminator loss

Figure 9: Loss of the generator and discriminator.

The mean of the temperature of the test set is compared as well in figure 10.

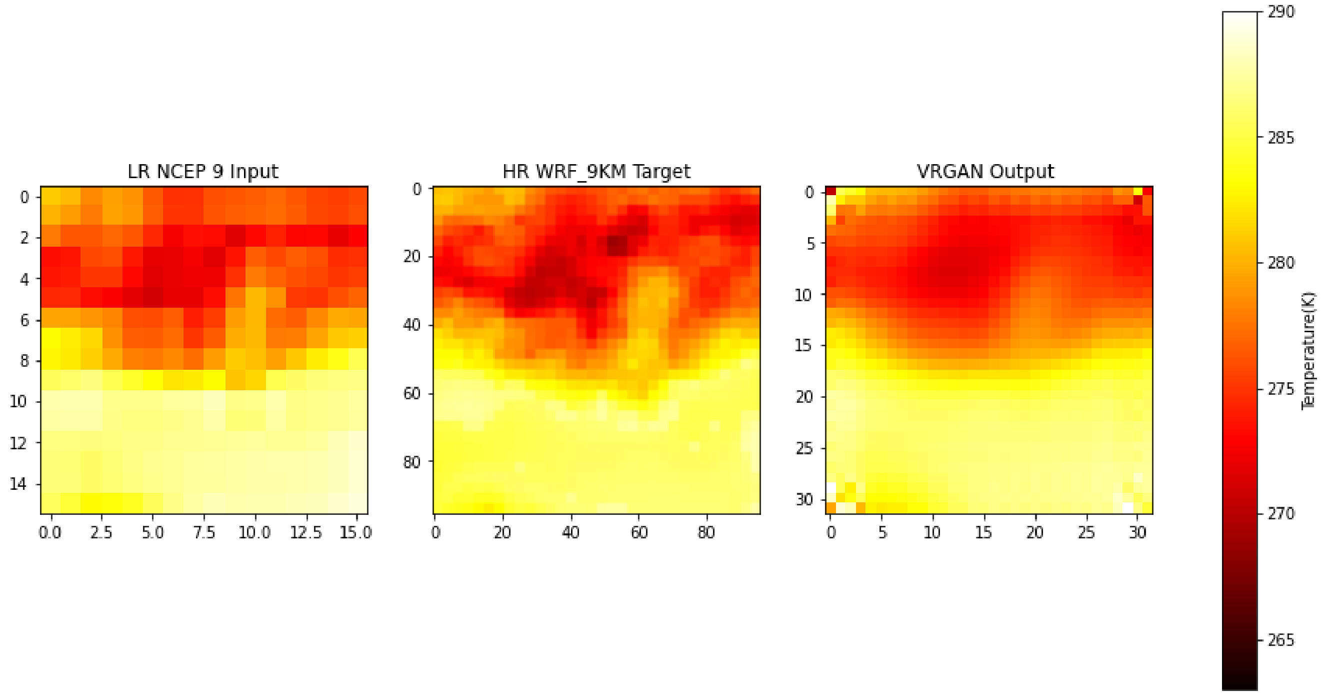


Figure 10: Mean of the LR inupt, HR goal and SRGAN output. Please don't subtracht for the typo in 'VRGAN', this should be 'SRGAN' in third subplot.

Also, the RMSE per opixel is computed. This is shown in Figure 11.

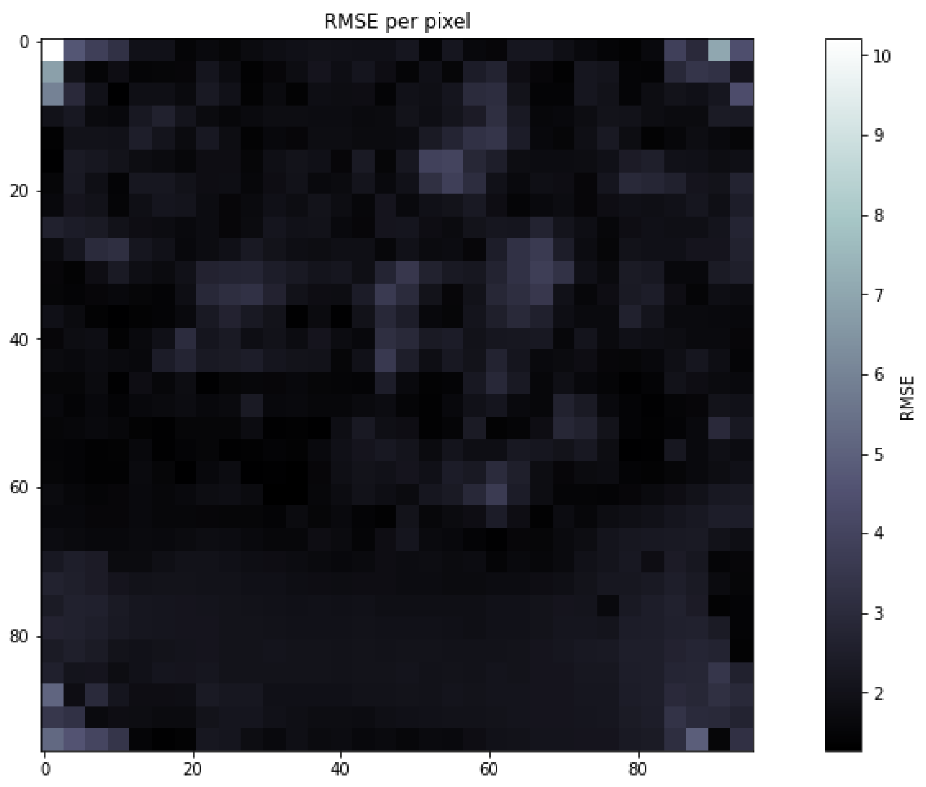
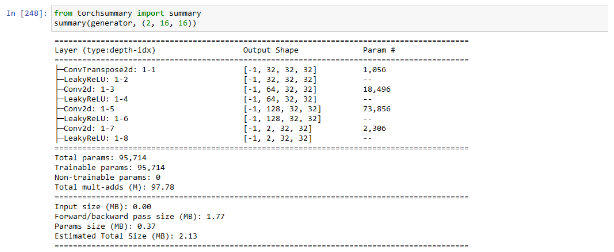


Figure 11: RMSE per pixel

It can be seen that the RMSE is smallest in the valley, and highest in the mountains. This is logical, since the big topography differences in the mountains as can be seen in a). But, overall, the GAN performs pretty good.

(e) For the generator and the discriminator network, in principle the same GAN was used, but now with (2, 16, 16) input. Our input is still (N_p, S, S) where N_p = number of predictors (which is 2 now), S is patch size. For the detailed description of the network, see Figure 12

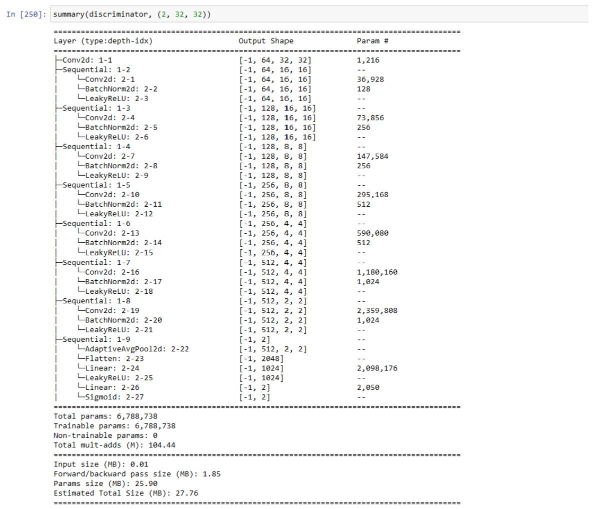
Furthermore, for the training different hyperparameters were used than in c). Namely, $\alpha = 0.01$, $n_{epoch} = 300$, $l_g = 9 \times 10^{-3}$ and $l_d = 1 \times 10^{-5}$. Batches were still trained in batch sizes of 64. These hyperparameters where chosen because they gave the best results. Also, the dynamical learning from c) is still used, where the generator or discriminator is trained separately if one of the two variables trains too quickly. I set the generator learning rate higher in comparison to c) because with the old hyperparameters my generator was learning too slow. I hypothesised that the wind would need more training since it is harder to predict than temperature. The generator loss is also decreasing slower than in d) (Figure 18).



```
In [248]: from torchsummary import summary
          summary(generator, (2, 16, 16))
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─ConvTranspose2d: 1-1                   [-1, 32, 32, 32]          1,056
├─LeakyReLU: 1-2                         [-1, 32, 32, 32]          --
├─Conv2d: 1-3                            [-1, 64, 32, 32]          18,496
├─LeakyReLU: 1-4                         [-1, 64, 32, 32]          --
├─Conv2d: 1-5                            [-1, 128, 32, 32]         73,856
├─LeakyReLU: 1-6                         [-1, 128, 32, 32]         --
├─Conv2d: 1-7                            [-1, 2, 32, 32]           2,306
├─LeakyReLU: 1-8                         [-1, 2, 32, 32]           --
==========================================================================================
Total params: 95,714
Trainable params: 95,714
Non-trainable params: 0
Total mult-adds (M): 97.78
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 1.77
Params size (MB): 0.37
Estimated Total Size (MB): 2.13
==========================================================================================
```

(a) Generator layout Wind



```
In [250]: summary(discriminator, (2, 32, 32))
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Conv2d: 1-1                            [-1, 64, 32, 32]          1,216
├─Sequential: 1-2                        [-1, 64, 16, 16]          --
|    └─Conv2d: 2-1                       [-1, 64, 16, 16]          36,928
|    └─BatchNorm2d: 2-2                  [-1, 64, 16, 16]          128
|    └─LeakyReLU: 2-3                    [-1, 64, 16, 16]          --
├─Sequential: 1-3                        [-1, 128, 16, 16]         --
|    └─Conv2d: 2-4                       [-1, 128, 16, 16]         73,856
|    └─BatchNorm2d: 2-5                  [-1, 128, 16, 16]         256
|    └─LeakyReLU: 2-6                    [-1, 128, 16, 16]         --
├─Sequential: 1-4                        [-1, 128, 8, 8]           --
|    └─Conv2d: 2-7                       [-1, 128, 8, 8]           147,584
|    └─BatchNorm2d: 2-8                  [-1, 128, 8, 8]           256
|    └─LeakyReLU: 2-9                    [-1, 128, 8, 8]           --
├─Sequential: 1-5                        [-1, 256, 8, 8]           --
|    └─Conv2d: 2-10                      [-1, 256, 8, 8]           295,168
|    └─BatchNorm2d: 2-11                 [-1, 256, 8, 8]           512
|    └─LeakyReLU: 2-12                   [-1, 256, 8, 8]           --
├─Sequential: 1-6                        [-1, 256, 4, 4]           --
|    └─Conv2d: 2-13                      [-1, 256, 4, 4]           590,080
|    └─BatchNorm2d: 2-14                 [-1, 256, 4, 4]           512
|    └─LeakyReLU: 2-15                   [-1, 256, 4, 4]           --
├─Sequential: 1-7                        [-1, 512, 4, 4]           --
|    └─Conv2d: 2-16                      [-1, 512, 4, 4]           1,180,160
|    └─BatchNorm2d: 2-17                 [-1, 512, 4, 4]           1,024
|    └─LeakyReLU: 2-18                   [-1, 512, 4, 4]           --
├─Sequential: 1-8                        [-1, 512, 2, 2]           --
|    └─Conv2d: 2-19                      [-1, 512, 2, 2]           2,359,808
|    └─BatchNorm2d: 2-20                 [-1, 512, 2, 2]           1,024
|    └─LeakyReLU: 2-21                   [-1, 512, 2, 2]           --
├─Sequential: 1-9                        [-1, 2]                   --
|    └─AdaptiveAvgPool2d: 2-22           [-1, 512, 2, 2]           --
|    └─Flatten: 2-23                     [-1, 2048]                --
|    └─Linear: 2-24                      [-1, 1024]                2,098,176
|    └─LeakyReLU: 2-25                   [-1, 1024]                --
|    └─Linear: 2-26                      [-1, 2]                   2,050
|    └─Sigmoid: 2-27                     [-1, 2]                   --
==========================================================================================
Total params: 6,788,738
Trainable params: 6,788,738
Non-trainable params: 0
Total mult-adds (M): 104.44
==========================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 1.85
Params size (MB): 25.90
Estimated Total Size (MB): 27.76
==========================================================================================
```

(b) Discriminator layout

Figure 12: Structure of the generator and discriminator used for the wind.

The RMSE of the U and V 10m component of the SRGAN are plotted against the baseline results obtained in b) in Figure 13.

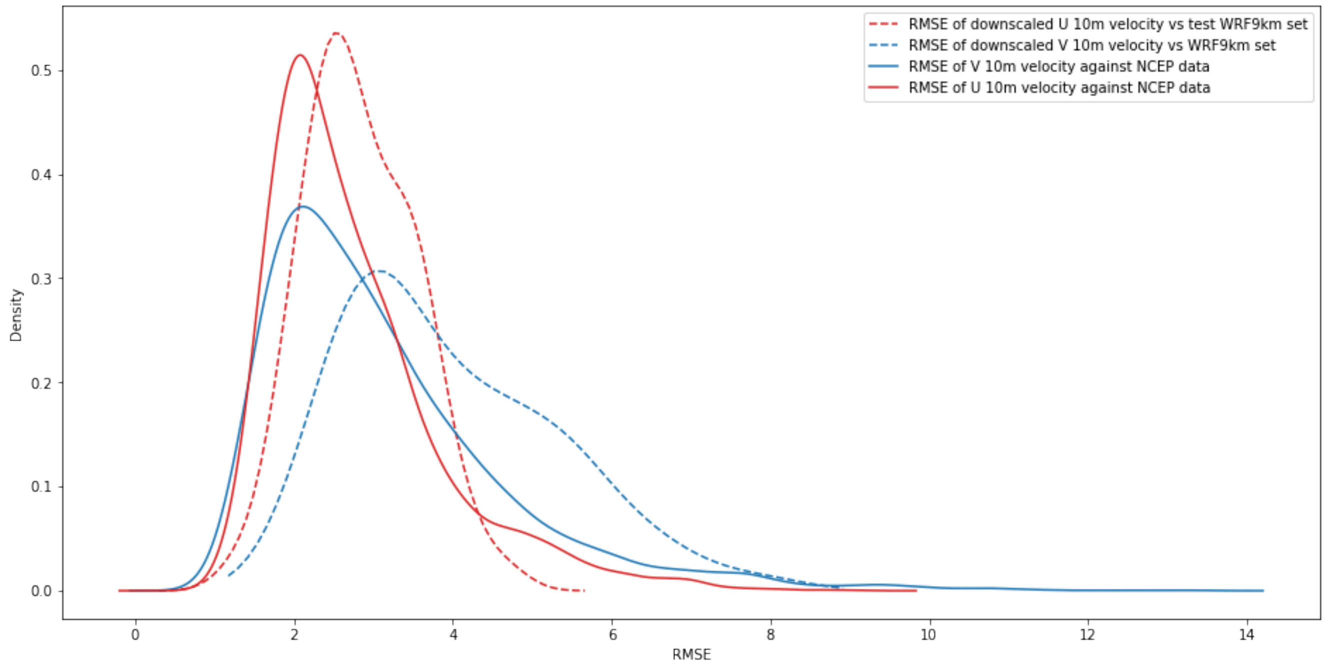


Figure 13: RMSE of U and V 10m wind speed component, downscaled LR → MR velocity vs test WRF9km set versus the baseline

Here, it can be seen that the baseline performs better. This probably has multiple factors, but to name a few we did not use resevoir computing or made a long short-term memory (LSTM) layer that Miralles et al., 2022 proposes, which means the autocorrelation of the wind is not taken into account. This makes wind very difficult to predict (Miralles et al., 2022). Furthermore, the topography isn't taken into account either in the network, and wind behaves unpredictable in mountainous areas. Furthermore, here are the best plots my SRGAN made:

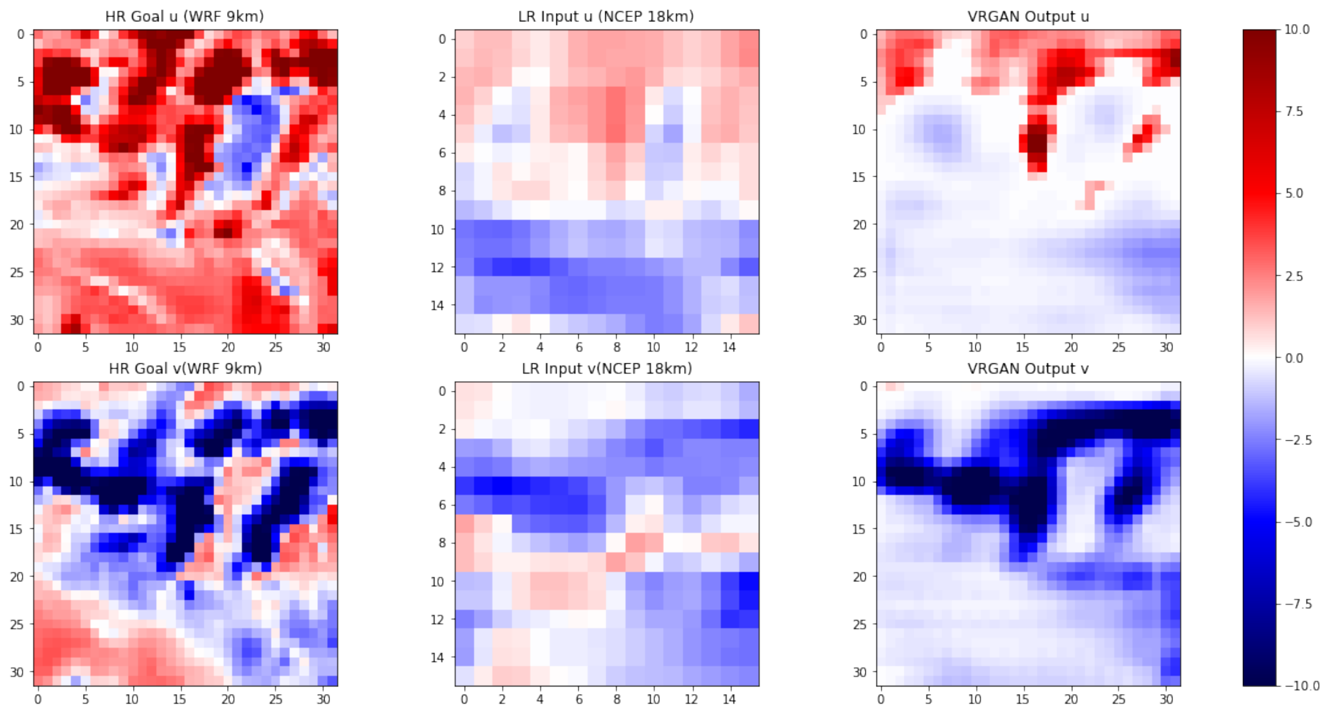


Figure 14: Input, target & Output for SRGAN LR → MR

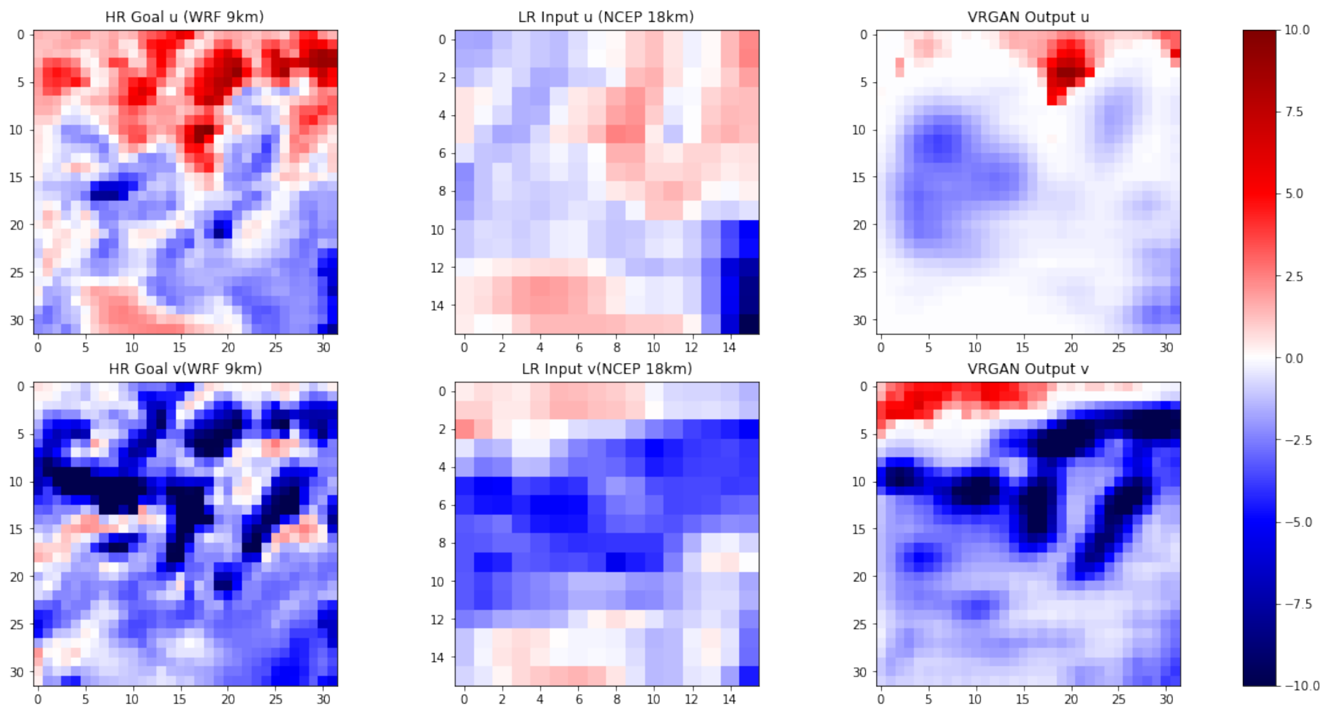


Figure 15: Input, target & Output for SRGAN LR → MR

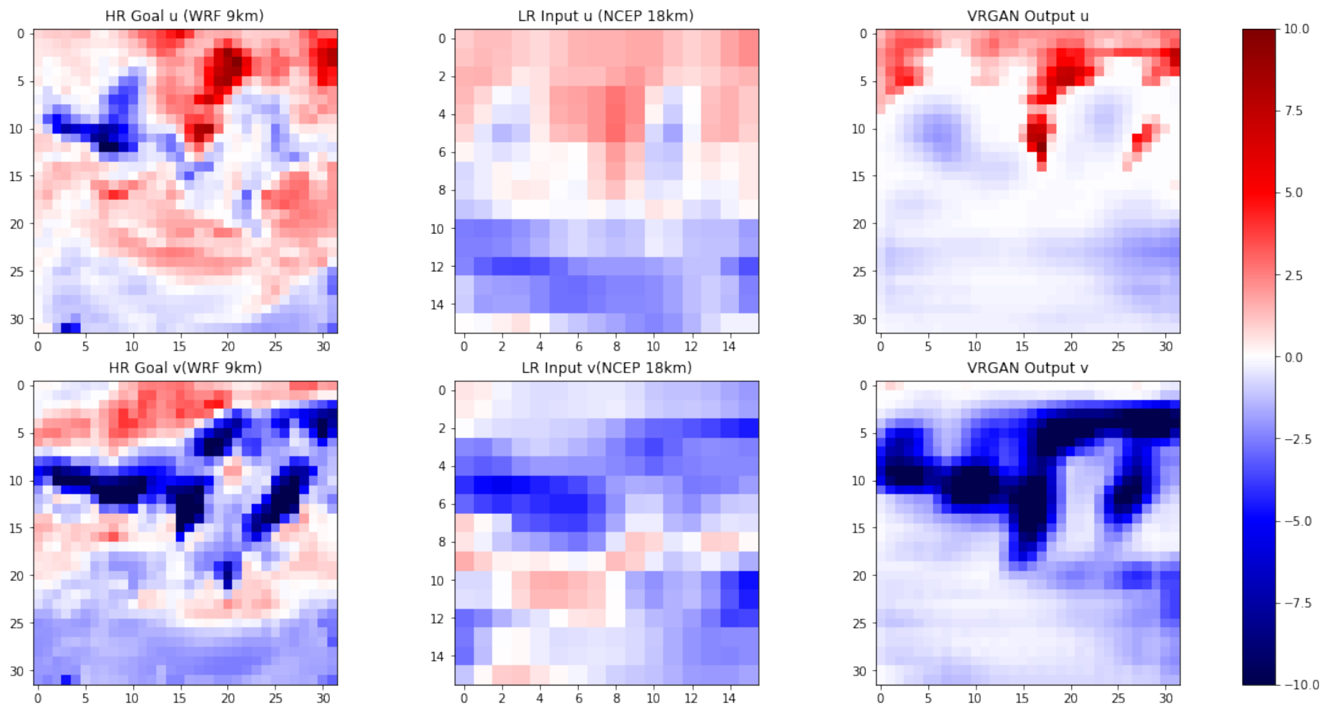


Figure 16: Input, target & Output for SRGAN LR → MR

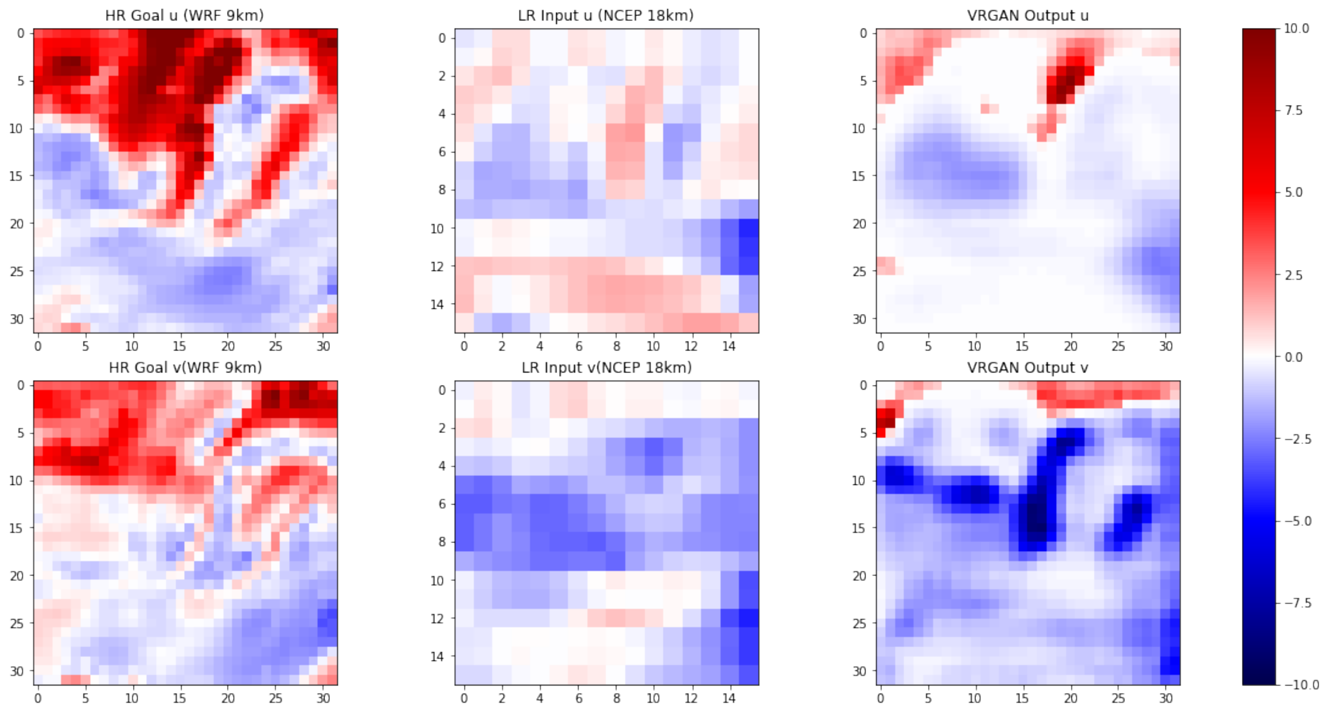
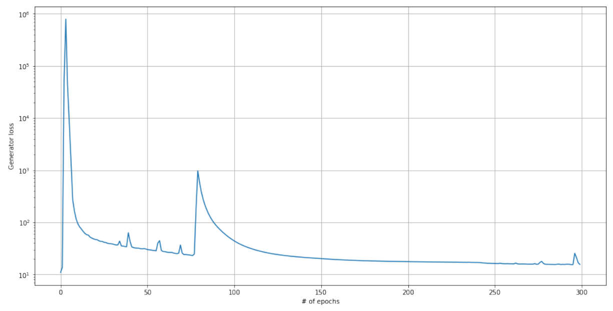
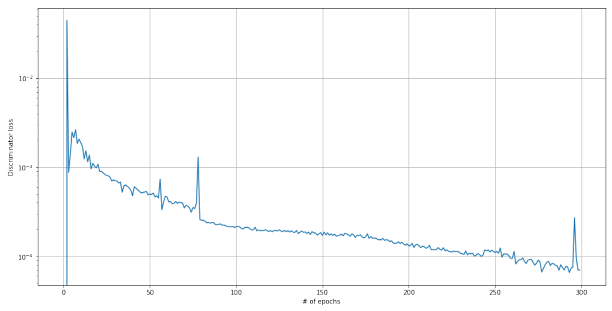


Figure 17: Input, target & Output for SRGAN LR → MR

It can be seen that the output isn't really making a lot of sense and this is in line with Figure 13. The generator loss and discriminator loss are plotted in Figure 18



(a) Generator loss



(b) Discriminator loss

Figure 18: Loss of the generator and discriminator.

(f) I would take the same approach, but in my generator and discriminator I would change the convolutional layer, such that the dimensions are in check. Namely, the imput dimensions need to be (4128,1,32,32) and the output dimentions (4128,1,96,96). Then the SRGAN should produce similar improvements.

Appendix

See notebooks attached. I'm very sorry that I don't have time to make them neat, my dearest apologies.

References

Miralles, O., Steinfeld, D., Martius, O., & Davison, A. C. (2022). Downscaling of Historical Wind Fields over Switzerland Using Generative Adversarial Networks. *Artificial Intelligence for the Earth Systems*, *1*(4). https://doi.org/10.1175/AIES-D-22-0018.1

Stengel, K., Glaws, A., Hettinger, D., & King, R. N. (2020). Adversarial super-resolution of climatological wind and solar data. *Proc. Natl. Acad. Sci. U.S.A.*, *117*(29), 16805–16815. https://doi.org/10.1073/pnas.1918964117