Anpassung und Erweiterung verschiedener Verfahren zur Frameratenerhöhung von Videosequenzen

von Fabian Brand

Februar 2017

Betreuer:
Dipl.-Ing. Michel Bätz
Contents

1 Introduction 1

2 Volume Alignment 3
   2.1 Concept of Volume Alignment .............................................. 4
   2.2 Changes to 3D-FSE ........................................................... 4
   2.3 Hybrid Approach ............................................................. 6
   2.4 Simulations and Results .................................................. 7
      2.4.1 Non Motion Compensated FRUC ..................................... 7
      2.4.2 Motion Compensated FRUC ........................................... 10
      2.4.3 Multi Layer Consistency Check ................................. 12

3 Neural Network FRUC 14
   3.1 Concept ................................................................. 14
   3.2 Implementation .......................................................... 15
      3.2.1 HDF5 Input Layer .................................................... 16
      3.2.2 Slice Layers ......................................................... 16
      3.2.3 Concatenation Layers ............................................... 17
      3.2.4 Silence Layers ...................................................... 17
      3.2.5 Convolutional Layers .............................................. 18
      3.2.6 ReLU Layers ........................................................... 19
      3.2.7 Euclidean Loss Layers ............................................. 19
   3.3 Networks ................................................................. 19
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4 Training Data</td>
<td>20</td>
</tr>
<tr>
<td>3.5 Evaluation</td>
<td>22</td>
</tr>
<tr>
<td>4 Summary</td>
<td>25</td>
</tr>
<tr>
<td>Literature</td>
<td>26</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

There are numerous algorithms to solve the problem of frame rate up-conversion (FRUC). These algorithms are needed to convert videos from a certain frame rate to a higher rate, for example when a video is displayed on a screen with a higher refresh rate than the recorded frame rate. Often this is implicitly done by just repeating the same frame until a new one is available, which does not fully exploit the performance of the screen.

It is possible to get estimates for the missing frames by looking at the surrounding frames. Modern well-performing algorithms usually employ motion estimation and subsequent motion compensation. Since the motion cannot be estimated at the to-be-interpolated frame, it can not be performed in an ideal way. Therefore holes and overlaps occur. There are various ways to deal with them or also to avoid them by using vector field retiming or bilateral motion estimation, both usually resulting in suboptimal motion vector fields.

In this thesis, three algorithms are examined and improved. The first two algorithms are based on the algorithms developed in [1], namely NRS-3DFSE and RS-MC-CC-3DFSE which showed the best results depending on the content. The main goal was to modify the 3DFSE so that the model generation takes motion information into account. This process will be called Volume Alignment (VA). The third algorithm tries to modify
the neural-network-based super-resolution algorithm developed in [2] to be applicable for FRUC.
Chapter 2

Volume Alignment

As described in [1], NRS-3DFSE and RS-MC-CC-3DFSE rely on three dimensional frequency extrapolation (3D-FSE) for interpolation. In NRS-3DFSE the missing frames are interpreted as normal missing pixels in the context of 3D-FSE and are interpolated. The key point here is, that the sequence is irregularly sampled to provide more temporal frequency information. In [1] it was shown, that this is necessary in order to obtain good results. On the other hand, RS-MC-3DFSE partially fills the missing frames with motion compensated data. This offers enough spatial support to perform 3DFSE on a temporally regular sampling grid. The algorithm for motion compensation is a modified version from Partial Averaging Motion Compensation (PAMC) as described in [3]. The motion data is filtered with a consistency check to find outliers. Therefore the motion compensated parts contain a large amount of holes which is then filled with 3D-FSE.

In 3D-FSE a sparse frequency model for a given volume is created using the surrounding information as support [4]. This support area is typically a cube with the to-be-interpolated volume in the center. The idea of volume alignment is now to use a motion compensated support volume instead. This idea was adapted from [5].
2.1 Concept of Volume Alignment

In general, volume alignment requires only one additional step in the 3D-FSE. Usually, the support area is taken out of the raw lossy input data. Now before extracting the data, all relevant frames (i.e., all frames up until a certain distance to the loss volume) are motion compensated to the loss frame. The support area is then extracted from the motion compensated, or aligned, volume. The concept of VA is shown in Fig. 2.1 for a block error. Since the lossy frame is only incomplete, or in the case of NRS-3DFSE completely empty, the lossy frame is estimated from the two adjacent frames using a conventional FRUC algorithm like PAMC or motion compensated linear averaging (MCLA). This estimated frame is then used as reference for the further volume alignment.

2.2 Changes to 3D-FSE

To make volume alignment work, some changes have to be implemented in the 3D-FSE. The two major changes are the change of the cube size and the implementation of a different processing order.
2.2. CHANGES TO 3D-FSE

The temporal block size has to be set to 1 if volume alignment is to be used, since motion compensation can only be done with respect to one single frame. If a cube would have a larger size it would be ambiguous which frame shall be used as reference for VA. In any way, there would be inconsistencies.

Considering a previous cube size of $4 \times 4 \times 4$, each cube containing two loss frames, one cube is now replaced by two $4 \times 4 \times 1$ cubes at the position of the loss frames. For each of the cubes one whole model has to be created. This increases the complexity of 3D-FSE by a factor of two, just by the use of another block size.

To restrain the complexity another change has been made in the processing order. In previous implementations, each cube was selected from the whole volume to meet the conditions of the optimal processing order, as described in [4]. In principle this is also possible with volume alignment, but the following problem occurs: Since the whole interpolation is based on motion compensated data, prior to the processing of the block, motion compensation has to be performed. This has to be performed for at least the whole support area. It is easy to see that, since the support volumes overlap, a lot of the necessary calculations have to be performed multiple times. However, since a video usually consists of a very large amount of data, it is impractical to save all the information for the whole runtime of the algorithm. Therefore a new processing order is introduced, which first finishes interpolating one frame before computing cubes from other frames. This way it is possible to first completely compensate the surrounding frames. The single support volumes are then extracted from this compensated volume.

The order, in which the frames are processed is determined by the number of supporting pixels, thus matching the original optimized processing order. Also the processing order of the cubes inside the volume mainly fulfills the criterion introduced in [4]. Instead of looking only at the support on a cube level, now also the amount of supporting pixels in each supporting cube is taken into account. Another major difference is that the cubes, which are blocked for processing to avoid a race condition, are now determined by looking if they are in the respective support volume, whereas the previous implementation only blocked all neighboring blocks. This is mainly due to the fact
that in non motion compensated FRUC algorithms blocks which influence each other should not be processed directly after each other without additional support [1].

2.3 Hybrid Approach

The tests showed, that VA improves the result for sequences with large motion and declines for small motion. Probably this is because motion compensation uses interpolation methods which decrease the sharpness of the result. Usually a sequence consists of regions with different amounts of motion. The motion is therefore used as a decision rule to decide between the use of VA and no VA. If the motion is bigger than a certain threshold, VA is used, otherwise no VA is used. This way a hybridization of no VA and VA approaches is achieved. However, it is not trivial to determine the motion at the intermediate frame.

Three hybridization approaches are introduced. One non motion compensated and one motion compensated scheme and one using vector field retiming from [6]. In all approaches, the computation is done in a block wise fashion, with the spatial cube size of 3D-FSE as block size (in this case 4). From all motion vectors in this block the 90th percentile is compared with the threshold.

In the non motion compensated hybridization the motion vectors from the corresponding blocks in the closest support frames are used. This yields the double amount of motion vectors, compared to the other schemes. This scheme will be referred to as H-NMC.

The other two algorithms merge the motion vector fields using either vector field retiming or MCLA to motion compensate the motion information. They will be referred to as H-VFR and H-MC respectively. H-MC was also tested with pixel accuracy, i.e. a block size of 1. This will be referred to as H-MCP.

For benchmarking purposes tests are also performed with an oracle hybridization, in which case the two blocks will be compared to ground truth and the best performing block is used. This is used to indicate the highest achievable gain by hybridization.
They are tested for a block size of 4 and 1 and will be called H-O4 and H-O1, respectively.

2.4 Simulations and Results

Preliminary tests showed, that VA performs best when the reference frame is estimated with PAMC and no consistency check is used in VA. (However, as shown in [1], the consistency check is used for the motion compensation of the sequence before entering it into 3D-FSE.)

The main focus of the following experiments is determining the best way of hybridization. Experiments were conducted of the first 51 frames of the PartyScene, BQSquare, BlowingBubbles and RaceHorses sequences. So in total 25 frames are interpolated. BQSquare has only very small motion, whereas RaceHorses has very large motion.

The parameters of 3D-FSE are equal to the parameters determined in [1]. The parameters which stay the same for all performed tests are shown in Tab. 2.1. In particular the borderWidth parameter was set to $4 \times 10$ for NRS-3DFSE and $10 \times 4$ for RS-MC-CC-3DFSE, respectively. Additionally, the multi-layer consistency check introduced in [7] was added to RS-MC-CC-3DFSE as a separate test. This algorithm will be called RS-MC-MLCC-3DFSE.

Before the results are compared, a 10 pixel margin is removed, since VA introduces artifacts close to the border. Afterwards, the PSNR is used to compare the algorithms.

2.4.1 Non Motion Compensated FRUC

In a first experiment, VA is applied on NRS-3DFSE. For RaceHorses VA yields an improvement of 5.3dB, but the PSNR of BQSquare drops by 5.3dB. Also visually, it can be seen, that the sharpness of BQSquare is decreased. For PartyScene and BlowingBubbles the PSNR increases around 0.8dB. Table 2.2 shows detailed results for all tests.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Values</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta$</td>
<td>0.2</td>
<td>Weight of the interpolated pixels</td>
</tr>
<tr>
<td>FFT-size</td>
<td>32</td>
<td>FFT-size used for 3D-FFT in FSE</td>
</tr>
<tr>
<td>Cube-size</td>
<td>$4 \times 4 \times 1$</td>
<td>Size of the cubes dividing the sequence</td>
</tr>
<tr>
<td>Frequency weighting</td>
<td>off</td>
<td>Frequency weighting in 3D-FSE</td>
</tr>
<tr>
<td>Weighting function</td>
<td>centroid</td>
<td>Weighting function of 3D-FSE</td>
</tr>
<tr>
<td>$\rho$</td>
<td>0.7</td>
<td>Decay factor of the weight function</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>0.5</td>
<td>Orthogonality correction</td>
</tr>
<tr>
<td># iterations</td>
<td>500</td>
<td>Number of iterations</td>
</tr>
</tbody>
</table>

Table 2.1: This table shows the test parameters.

It can be seen that for $BQSquare$ and $RaceHorses$ the PSNR moves towards the better result for the proposed hybridization schemes. For the other two sequences, the PSNR is better than both previous results. In average the gain compared to no VA is increased from 0.6dB to 2.2dB by vector field retiming hybridization with a threshold of 2. In figure 2.2 and 2.3 a visual comparison for selected schemes for the $RaceHorses$ and $PartyScene$ sequences is shown.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>No VA</th>
<th>VA</th>
<th>H-NMC</th>
<th>H-VFR</th>
<th>H-MC</th>
<th>H-MCP</th>
<th>H-O1</th>
<th>H-O4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>BQSquare</td>
<td>38.67</td>
<td>33.36</td>
<td>37.17</td>
<td>38.42</td>
<td>37.70</td>
<td>38.67</td>
<td>37.23</td>
<td>38.66</td>
</tr>
<tr>
<td>RaceHorses</td>
<td>18.75</td>
<td>25.03</td>
<td>25.03</td>
<td>25.01</td>
<td>25.03</td>
<td>25.00</td>
<td>25.03</td>
<td>24.87</td>
</tr>
<tr>
<td>Mean</td>
<td>28.10</td>
<td>28.73</td>
<td>29.84</td>
<td>30.17</td>
<td>30.01</td>
<td>30.23</td>
<td>29.86</td>
<td>30.09</td>
</tr>
</tbody>
</table>

Table 2.2: Results for NRS-3DFSE with VA.
2.4.2 Motion Compensated FRUC

In this section, VA is used to improve upon RS-MC-CC-3DFSE. In average VA improves the PSNR by 0.1dB which is less than in the previous case. Compared to NRS-3DFSE, where VA can change the entire frame, here it can only effect smaller areas between 0.7% in BQSquare and 24% in RaceHorses [1]. Also the effect of the hybridization is smaller. The gain can be improved to 0.14dB with H-MC and a threshold of 2. Also the oracle tests show that only a 1dB hybridization gain is possible. However, RS-MC-CC-3DFSE shows better absolute results than NRS-3DFSE. Table 2.3 shows the detailed results for all tests.

In figure 2.4 and 2.5 a visual comparison for selected schemes for the RaceHorses und PartyScene sequences is shown. There is not much difference between VA and H-MC, since the areas with little motion are usually included in the motion compensated part of the image, which is the same for VA and no VA.
2.4. SIMULATIONS AND RESULTS

Figure 2.3: Visual Comparison for the *PartyScene* sequence with *NRS-3DFSE*

Figure 2.4: Visual Comparison for the *RaceHorses* sequence with *RS-MC-CC-3DFSE*
2.4.3 Multi Layer Consistency Check

As a last test, VA was combined with the multi-layer consistency check (MLCC) as introduced in [7]. Unfortunately VA does not improve upon the gain from MLCC and rather decreases the PSNR. One reason for that might be that MLCC gives additional motion compensated spatial support to the loss areas, whereas the temporal support offered by VA is the result of a different motion compensation based on interpolated data. Both kinds of support may not be consistent with each other, which may lead to some distortion. Table 2.4 shows the results for MLCC with VA.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>No VA</th>
<th>VA</th>
<th>H-NMC</th>
<th>H-VFR</th>
<th>H-MC</th>
<th>H-MCP</th>
<th>H-O1</th>
<th>H-O4</th>
</tr>
</thead>
<tbody>
<tr>
<td>PartyScene</td>
<td>29.96</td>
<td>29.84</td>
<td>29.84</td>
<td>29.86</td>
<td>29.85</td>
<td>29.86</td>
<td>29.84</td>
<td>29.88</td>
</tr>
<tr>
<td>BQSquare</td>
<td>37.84</td>
<td>37.71</td>
<td>37.72</td>
<td>37.77</td>
<td>37.82</td>
<td>37.84</td>
<td>37.73</td>
<td>37.84</td>
</tr>
<tr>
<td>BlowingBubbles</td>
<td>30.12</td>
<td>30.18</td>
<td>30.18</td>
<td>30.19</td>
<td>30.17</td>
<td>30.17</td>
<td>30.18</td>
<td>30.19</td>
</tr>
<tr>
<td>Mean</td>
<td>31.08</td>
<td>31.18</td>
<td>31.18</td>
<td>31.20</td>
<td>31.21</td>
<td>31.22</td>
<td>31.19</td>
<td>31.22</td>
</tr>
</tbody>
</table>

Table 2.3: Results for RS-MC-CC-3DFSE with VA.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>No VA</th>
<th>VA</th>
<th>H-NMC</th>
<th>H-VFR</th>
<th>H-MC</th>
<th>H-MCP</th>
<th>H-O1</th>
<th>H-O4</th>
</tr>
</thead>
<tbody>
<tr>
<td>PartyScene</td>
<td>30.30</td>
<td>29.66</td>
<td>29.66</td>
<td>29.68</td>
<td>29.68</td>
<td>29.73</td>
<td>29.66</td>
<td>29.76</td>
</tr>
<tr>
<td>BQSquare</td>
<td>37.83</td>
<td>37.63</td>
<td>37.67</td>
<td>37.76</td>
<td>37.80</td>
<td>37.83</td>
<td>37.70</td>
<td>37.83</td>
</tr>
<tr>
<td>BlowingBubbles</td>
<td>30.35</td>
<td>30.01</td>
<td>30.01</td>
<td>30.01</td>
<td>30.00</td>
<td>30.01</td>
<td>30.01</td>
<td>30.04</td>
</tr>
<tr>
<td>Mean</td>
<td>31.44</td>
<td>31.03</td>
<td>31.04</td>
<td>31.07</td>
<td>31.07</td>
<td>31.10</td>
<td>31.04</td>
<td>31.11</td>
</tr>
</tbody>
</table>

Table 2.4: Results for RS-MC-MLCC-3DFSE with VA
Chapter 3

Neural Network FRUC

In this chapter, a neural network (NN) based FRUC algorithm is introduced. The algorithm follows the ideas from [2] for super-resolution. In this chapter only regular temporal sampling with an up-sampling factor of 2 is considered.

3.1 Concept

The basic idea of the algorithm is to use a neural network to combine several candidates for the intermediate frame. One of them is created with MCLA from the two adjacent frames. Another two candidates are created with PAMC from the two adjacent frames. PAMC is only performed until forward and backward estimation are combined. Instead the holes in both frames are interpolated separately and the frames are used as candidates. The forward and backward estimated frames will be called MC-F and MC-B, respectively. The last two candidates are created in the same way, just from the frames three time steps from the intermediate frame. Those candidates are omitted in some schemes.

The candidates are then entered into a convolutional neural network (CNN), which combines the candidates to create an optimal output. A CNN is a special architecture of NNs. The two key aspects are the weight sharing and that the layers are not fully connected. Each neuron represents a pixel in the data. In each convolutional layer the
values from a small environment around the corresponding pixel in the previous frame are combined (via a weighted sum) and then fed into a nonlinearity, usually a rectified linear unit (ReLU). The weights are the same for each pixel of that layer. This can be interpreted as a convolution which is used as input for the nonlinearity [8].

Furthermore, the input and the output of each layer usually consists of multiple frames. For every frame of the output volume a 3D-convolution of the input volume is performed. The weights for each frame may differ. Each layer can therefore be characterized by the number of output frames and the kernel size. The kernel size is only given for the spatial dimensions, since the size in the third dimension is already given by the number of input frames, which is given by the output of the previous layer.

3.2 Implementation

For the implementation of the neural network the caffe framework was used. In caffe each network is described by three .prototext files. One of them sets the training parameters while the other two describe the training and the deploy network. It is possible to describe both in one file, but due to different kinds of output (mean square error or actual result) and different input format (here HDF5 files were used for training and the matlab interface was used for deployment) it is simpler to split it up in two files.

The training file is relatively straightforward, as it just defines the standard parameters of NN training. Additionally the paths for the training network and the resulting model have to be defined. Also the solver mode may be set to CPU or GPU.

The definitions of the networks consist mainly on consecutive definitions of layers with certain types. Each layer has to be named first, so the trained parameters can be matched to the correct layer when it is deployed. Each layer may have input blobs (bottom) and output blobs (top). The exact number of inputs and outputs depend on the layer type. A list of all layers can be found in [9]. In the following all used layers are briefly introduced.
3.2.1 HDF5 Input Layer

This layer is named $HDF5\_DATA$ and reads data from a HDF5 container file.

```protobuf
layers {
  name: "data_pair"
  type: HDF5_DATA
  top: "data"
  top: "label"
  hdf5_data_param {
    source: "Test.txt"
    batch_size: 120
  }
  include: { phase: TRAIN }
}
```

The parameter `source` does not link directly to the HDF5 files, but to a text file containing a list of files which shall be used. `batch_size` defines how many data sets shall be read from the files in one time. This is limited by the available memory, since they are stored simultaneously. Most importantly, both outputs must be named as the corresponding group in the HDF5 file. The `include` parameter may be added to every layer and adds this layer only if the network is in a certain phase, namely TEST or TRAIN. Both versions are written in the prototext file for the training network using different sources, but depending on the phase only one is used.

3.2.2 Slice Layers

The $SLICE$ layer splits up the data on defined planes and returns the single frames.

```protobuf
layers {
  name: "slice_frames"
  type: SLICE
  bottom: "data"
  top: "data_backward"
  top: "data_MCLA"
  top: "data_forward"
  slice_param {
    slice_dim: 1
    slice_point: 1
    slice_point: 2
  }
}
```
slice_dim specifies the dimension along which shall be sliced. Note that they do not match the matlab numbering. This example slices along the third dimension in matlab, corresponding to slicing along the single frames of the input volume. slice_point specifies the frames after which the cut shall be made. The number of outputs must be one higher than the number of slice points. In this example the volume consisting of the three candidate frames is sliced, so each frame can be processed individually.

### 3.2.3 Concatenation Layers

The CONCAT layer is the opposite of the slice layer and is used to concatenate frames to one volume.

```plaintext
layers {
    name: "concat"
    bottom: "conv1_sf1"
    bottom: "conv1_MCLA"
    bottom: "conv1_sb1"
    top: "conv1"
    type: CONCAT
    concat_param {
        concat_dim: 1
    }
}
```

The concat_dim parameter sets the dimension of the concatenation. Here the results from previous convolutions are merged to be processed jointly in the next step. Note that each input may have more than one frame already. Here each input has the dimensions $28 \times 28 \times 64$ and the output has the dimension $28 \times 28 \times 192$.

### 3.2.4 Silence Layers

A SILENCE layer has arbitrarily many inputs and no outputs. The only reason to use this layer is to clean up the network. This is necessary since blobs which are not fed into any other layers will be used as output of the network.

```plaintext
layers {
    name: "silence_sb2"
    bottom: "datax_sb2"
}
```
3.2.5 Convolutional Layers

The core of each CNN are the convolutional layers. They are named \textit{CONVOLUTION} and perform a convolution with a trained kernel. Note, that no nonlinearity is included.

layers {
  name: "conv1_MCLA"
  type: CONVOLUTION
  bottom: "data_MCLA"
  top: "conv1_MCLA"
  blobs_lr: 10
  blobs_lr: 20
  weight_decay: 1
  weight_decay: 0
  convolution_param {
    num_output: 64
    kernel_size: 9
    pad: 0
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}

\textit{blobs\_lr} and \textit{weight\_decay} define the learning parameters for weights and bias separately. Therefore both parameters are listed twice. \textit{num\_output} sets the number of filters which shall be learned, which also equals the number of output frames. \textit{kernel\_size} defines the size of the filter. \textit{pad} is set to 0 in learning and \(\frac{\text{kernel\_size} - 1}{2}\) for deploying. This is to keep the training clear of border effects and on the other hand not to change the size of the image when deploying the network. \textit{weight\_filler} and \textit{bias\_filler} set the strategy to initialize the the layer. The \textit{stride} should always be set to zero for image and video post processing.
3.2.6 ReLU Layers

The RELU layer applies a rectified unifier on each pixel. It is usually used after a convolutional layer.

```text
layers {
    name: "relu1"
    type: RELU
    bottom: "conv1_sf1"
    top: "conv1_sf1"
}
```

3.2.7 Euclidean Loss Layers

The Euclidean_loss layer computes the error between the output of the network and the ground truth. To match machine learning convention, the ground truth is called label.

```text
layers {
    name: "Euclidean_loss"
    bottom: "conv4"
    bottom: "label"
    type: EUCLIDEAN_LOSS
}
```

This layer has two inputs and one optional output. If the layer is used for training no output is required. However, in the test phase another Euclidean loss layer with output can be used to output the error.

3.3 Networks

The tests were performed on four different networks, which are all based on the structure shown in Fig 3.1. Each candidate is filtered with one convolutional layer and a ReLU before all layers are combined and processed with another three convolutional layers. The last layer produces only one output, which is the resulting frame. The structure is inspired by the proposed structure in [2].
The difference of the schemes lies in the number of candidate frames. As previously discussed, either three or five candidates are used. Either only frames with distance 1 or frames with distance 1 and 3 are used. For five input candidates, the part before the concatenation layer are just duplicated.

Also side information about the candidates can be passed along. In this case each input does not consist of a single frame but a frame and a binary mask giving information whether a specific pixel was obtained from motion compensation (1) or from subsequent interpolation (0). This is also tested for three or five input candidates. Those masks only exist for the PAMC candidates, so the input consists of 2 or 4 volumes consisting of two frames and one additional frame for MCLA.

### 3.4 Training Data

The network was trained on patches extracted from H.265 sequences, namely *BasketballDrive, Kimono, BQTerrace, ParkScene* and *Cactus*. For each frame of the sequences (an exception is the *Kimono* sequence where only the frames up to the cut were used) the candidates were calculated including the masks. From the resulting volume obtained for each frame patches of the size $36 \times 36 \times 9$ were extracted. The number 9 comes from 5 candidates plus 4 masks. 10000 patches of this kind were randomly chosen and concatenated in the 4th dimension, thus giving a $36 \times 36 \times 9 \times 10000$ matrix.

From the original sequences $20 \times 20 \times 1$ patches were extracted serving as ground truth. The difference in size is necessary, since no padding is performed in the convolutional layer in the training step. The ground truth is also concatenated, yielding a $20 \times 20 \times 1 \times 10000$ matrix. Both of them are then written into a HDF5 file using the HDF5 matlab framework.

In total 150 HDF5 files each containing 10000 training patches were created, so in total 1.5 Million patches are used for training. Another 20000 patches are used in the test set.
Figure 3.1: This figure shows the basic architecture of the used NN. The parameters of the convolutional layers are given by (kernel size, number of output frames).
3.5 Evaluation

The neural networks were tested on four sequences from the H.265 database, namely a down-sampled version of RaceHorses, BQSquare, BlowingBubbles and BasketballPass, none of which was used for training. Every second frame starting from the 4th till the 48th frame was discarded and interpolated with the NN. For evaluation a 10 pixel margin was removed to avoid border artifacts. The PSNR was used to compare the algorithms. For testing the network was trained for 200000 iterations, which took about 10 hours on a GTX1070 graphic card.

In Tab. 3.1 the results are shown. The results are compared against PAMC which serves as reference. Also the three inner candidates are evaluated. Unfortunately none of the results achieved with CNNs reach PAMC. However, for the RaceHorses sequence it was possible to exceed all candidates when run with three candidates. Fig. 3.2 shows the single frame PSNR. Given the figure and the other results, a viable assumption is, that the CNN is good in improving badly estimated frames (the results suggest around 30dB as threshold) and gets worse the better the original results get. Visual inspection shows that the main problem is a decrease in sharpness as if by a slight low pass filtering. However it is also capable of removing some artifacts as shown in Fig. 3.2. Note the ghosting artifact at the back of the horse which is considerably weaker after the CNN.
Table 3.1: This table shows the results for FRUC with CNN. PAMC serves as reference. Additionally the candidates MCLA, MC-F, MC-B, the last of which are the forward and backward estimated motion compensated frames with a distance of 1 are evaluated.

<table>
<thead>
<tr>
<th></th>
<th>PAMC</th>
<th>MCLA</th>
<th>MC-F</th>
<th>MC-B</th>
<th>No Mask</th>
<th>Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td># Candidates</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BlowingBubbles</td>
<td>29.87</td>
<td>29.42</td>
<td>29.33</td>
<td>29.44</td>
<td>28.78</td>
<td>28.77</td>
</tr>
<tr>
<td>BasketballPass</td>
<td>39.01</td>
<td>37.92</td>
<td>38.08</td>
<td>37.99</td>
<td>35.80</td>
<td>35.85</td>
</tr>
<tr>
<td>BQSquare</td>
<td>37.72</td>
<td>34.86</td>
<td>34.54</td>
<td>34.90</td>
<td>29.54</td>
<td>29.51</td>
</tr>
<tr>
<td>Mean</td>
<td>33.89</td>
<td>32.71</td>
<td>32.60</td>
<td>32.73</td>
<td>30.74</td>
<td>30.73</td>
</tr>
</tbody>
</table>

Figure 3.2: Single frame PSNR of RaceHorses after interpolation with MCLA (the best candidate) and after processing with a 3 candidate CNN without mask.
Figure 3.3: Visual comparison for the *RaceHorses* sequence. The most distinctive difference is marked.
Chapter 4

Summary

In this work two approaches for FRUC were evaluated. Volume alignment showed significant improvements for non motion compensated interpolation on non regular sampling grids. However for motion compensated interpolation the improvement decreased, but it was possible to improve the results by a motion based hybridization scheme. Also it was not possible to combine VA with a multi layer consistency check. In future works another kind of volume alignment can be tested, in which the weighting function in 3D-FSE is aligned instead of the support volume. This may help to stabilize the results especially in areas of small movement, since it is a softer approach and requires no interpolation of image data in the actual alignment step, which should increase the resulting sharpness. Another topic to research is finding a better hybridization scheme also based on other features rather than just the motion vectors. It might even be possible to apply machine learning techniques. This was not done in this work, since it requires a lot of training data, which are very time consuming to create with 3D-FSE.

The other approach was to perform FRUC with the help of CNNs. This approach showed rather bad results. However, the capabilities of CNNs were not fully covered but it was shown, that in case of the RaceHorses sequence improvement was possible. In future work different network structures and different features may be tested. It might also be possible to increase the sharpness again by deconvolution algorithms.
Bibliography


