Frame Booster

Kamil Barszczak

Frame interpolation

First image

Image to predict

Second image

First image

Image to predict

Second image

First image

Image to predict

Second image

The dataset used during the training

- Vimeo90K - triplet (~ 51000 samples, 448x256x3, 35 GB)

First image

Image to predict

Second image

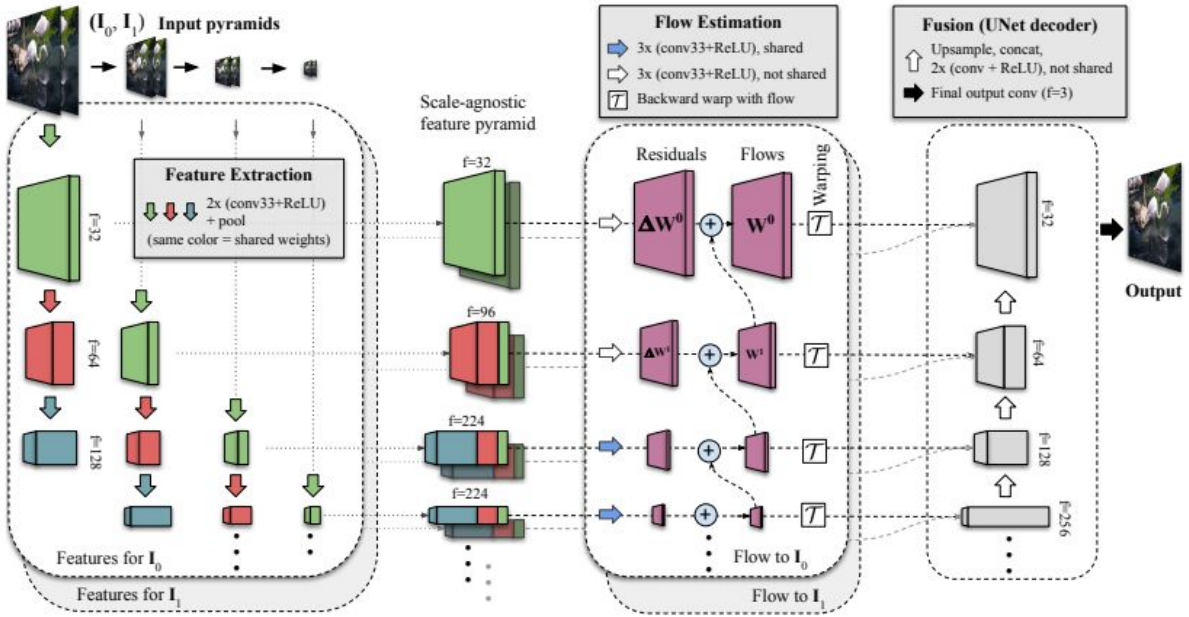
- Custom data (~ 1000 samples, 256x144x3, 1.3 GB)

First image

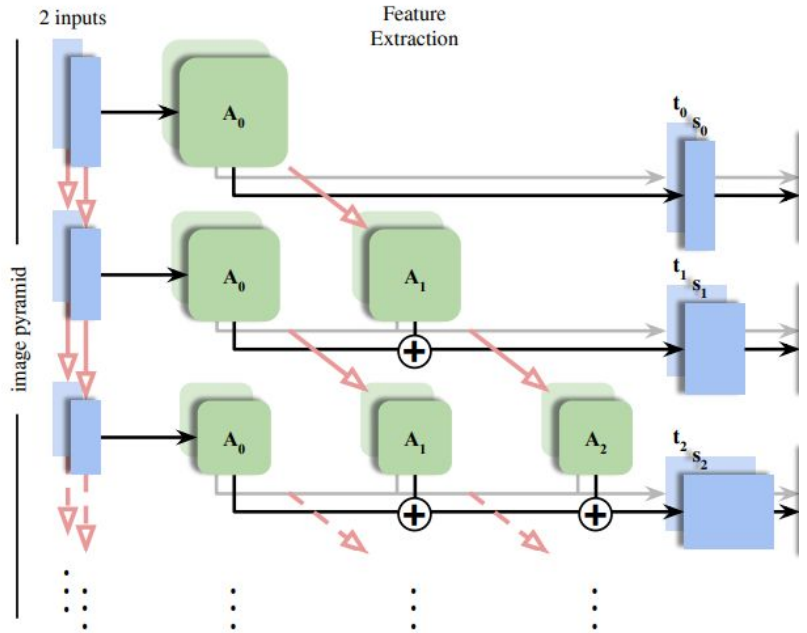
Image to predict

Second image

FBNet

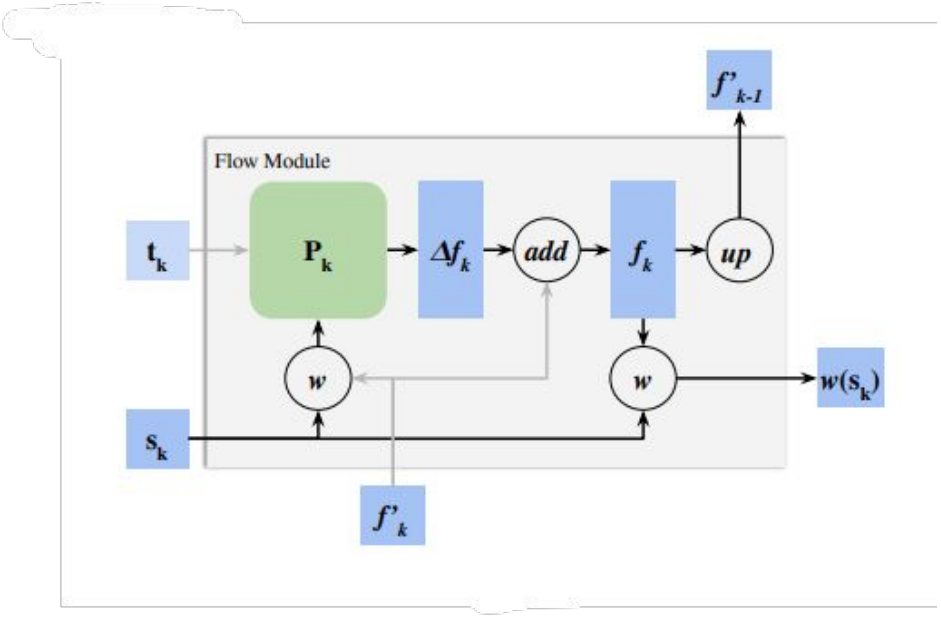

(I_0, I_1) Input pyramids
Feature Extraction
2x (conv33+ReLU) + pool
(same color = shared weights)
f=32
f=64
f=128
Features for I_0
Features for I_1
Scale-agnostic feature pyramid
f=32
f=96
f=224
f=224
Flow Estimation
3x (conv33+ReLU), shared
3x (conv33+ReLU), not shared
Backward warp with flow
Residuals
Flows
Warping
ΔW^0
W^0
ΔW^1
W^1
Flow to I_0
Flow to I_1
Fusion (UNet decoder)
Upsample, concat, 2x (conv + ReLU), not shared
Final output conv (f=3)
f=32
f=64
f=128
f=256
Output

FeatureExtraction



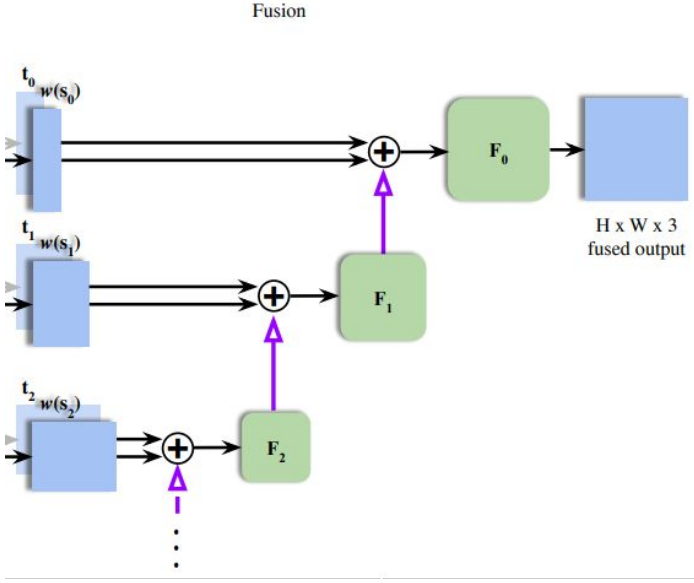

```
"""
PyramidFeatureExtraction is a class that makes the pyramid of the input image of depth 4. Then th
features of the image at each level and concatenates them with the features captured from the pre
This layer output is a 4-dimensional tuple with extracted features from each level. The first ele
are the features from the finest level.
"""
class PyramidFeatureExtraction(layers.Layer):
    def __init__(self, filter_count=[16, 32, 64], filter_size=(3, 3), activation='relu', regulari
        super(PyramidFeatureExtraction, self).__init__(**kwargs)

        # for pyramid creation
        self.downsample_avg = layers.AveragePooling2D((2, 2))

        # for feature extration (those layers are shared)
        self.cnn_1st_level = layers.Conv2D(filter_count[0], filter_size, activation=activation, k
        self.cnn_2nd_level = layers.Conv2D(filter_count[1], filter_size, activation=activation, k
        self.cnn_3rd_level = layers.Conv2D(filter_count[2], filter_size, activation=activation, k

        # concatenation layers
        self.concat_2nd_level = layers.Concatenate()
        self.concat_3rd_level = layers.Concatenate()
        self.concat_4th_level = layers.Concatenate()

        self.filter_count = filter_count
        self.filter_size = filter_size
        self.activation = activation
        self.regularizer = regularizer

    def get_config(self):
        config = super().get_config()
        config.update({
            "filter_count": self.filter_count,
            "filter_size": self.filter_size,
            "activation": self.activation,
            "regularizer": self.regularizer,
        })
        return config

    def call(self, inputs):
        # pyramid
        input_1 = inputs
        input_2 = self.downsample_avg(input_1)
        input_3 = self.downsample_avg(input_2)
        input_4 = self.downsample_avg(input_3)

        # feature extraction for layer 1
        input_1_column_1_row_1 = self.cnn_1st_level(input_1)
        input_2_column_1_row_2 = self.cnn_1st_level(input_2)
        input_3_column_1_row_3 = self.cnn_1st_level(input_3)
        input_4_column_1_row_4 = self.cnn_1st_level(input_4)

        # downsample layer 1
        input_1_column_2_row_2 = self.downsample_avg(input_1_column_1_row_1)
        input_2_column_2_row_3 = self.downsample_avg(input_2_column_1_row_2)
        input_3_column_2_row_4 = self.downsample_avg(input_3_column_1_row_3)

        # feature extraction for layer 2
        input_1_column_2_row_2 = self.cnn_2nd_level(input_1_column_2_row_2)
        input_2_column_2_row_3 = self.cnn_2nd_level(input_2_column_2_row_3)
        input_3_column_2_row_4 = self.cnn_2nd_level(input_3_column_2_row_4)

        # downsample layer 2
        input_1_column_3_row_3 = self.downsample_avg(input_1_column_2_row_2)
        input_2_column_3_row_4 = self.downsample_avg(input_2_column_2_row_3)

        # feature extraction for layer 3
        input_1_column_3_row_3 = self.cnn_3rd_level(input_1_column_3_row_3)
        input_2_column_3_row_4 = self.cnn_3rd_level(input_2_column_3_row_4)

        # concatenate
        concat_1st = input_1_column_1_row_1
        concat_2nd = self.concat_2nd_level([input_2_column_1_row_2, input_1_column_2_row_2])
        concat_3rd = self.concat_3rd_level([input_3_column_1_row_3, input_2_column_2_row_3, input_
        concat_4th = self.concat_4th_level([input_4_column_1_row_4, input_3_column_2_row_4, input_

        return concat_1st, concat_2nd, concat_3rd, concat_4th
```

BidirectionalFlowEsimation



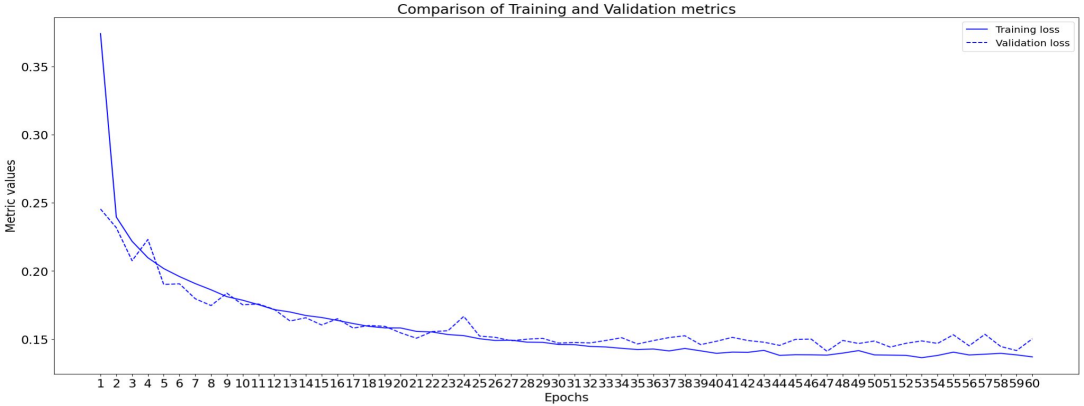

```
"""
BidirectionalFlowEstimation is a layer that warps the features extracted at the given level tryin
to modify them to fit the target image. It predicts the flow between features of both the input_1
and the input_2 and warps them to get the final output. The output shape is the same as the input
"""
class BidirectionalFlowEstimation(layers.Layer):
    def __init__(self, filter_count=[32, 64, 64, 16], filter_size=[(3, 3), (3, 3), (1, 1), (1, 1)
        super(BidirectionalFlowEstimation, self).__init__(**kwargs)

        # flow 1 -> 2
        self.flow_add_1_2 = layers.Add()
        self.flow_upsample_1_2 = layers.UpSampling2D((2, 2), interpolation=interpolation)
        self.flow_1_2_concat = layers.Concatenate(axis=3)
        self.flow_prediction_1_2 =  keras.Sequential([
            layers.Conv2D(filter_count[0], filter_size[0], activation=activation, kernel_regulari
            layers.Conv2D(filter_count[1], filter_size[1], activation=activation, kernel_regulari
            layers.Conv2D(filter_count[2], filter_size[2], activation=activation, kernel_regulari
            layers.Conv2D(filter_count[3], filter_size[3], activation=activation, kernel_regulari
            layers.Conv2D(2, (1, 1), kernel_regularizer=regularizer, padding='same')
        ])

        # flow 2 -> 1
        self.flow_add_2_1 = layers.Add()
        self.flow_upsample_2_1 = layers.UpSampling2D((2, 2), interpolation=interpolation)
        self.flow_2_1_concat = layers.Concatenate(axis=3)
        self.flow_prediction_2_1 = keras.Sequential([
            layers.Conv2D(filter_count[0], filter_size[0], activation=activation, kernel_regulari
            layers.Conv2D(filter_count[1], filter_size[1], activation=activation, kernel_regulari
            layers.Conv2D(filter_count[2], filter_size[2], activation=activation, kernel_regulari
            layers.Conv2D(filter_count[3], filter_size[3], activation=activation, kernel_regulari
            layers.Conv2D(2, (1, 1), kernel_regularizer=regularizer, padding='same')
        ])

        self.filter_count = filter_count
        self.filter_size = filter_size
        self.activation = activation
        self.regularizer = regularizer
        self.interpolation = interpolation

    def get_config(self):
        config = super().get_config()
        config.update({
            "filter_count": self.filter_count,
            "filter_size": self.filter_size,
            "activation": self.activation,
            "regularizer": self.regularizer,
            "interpolation": self.interpolation,
        })
        return config

    def call(self, inputs):
        input_1 = inputs[0]
        input_2 = inputs[1]
        flow_1_2 = inputs[2]
        flow_2_1 = inputs[3]

        # input_1 to input_2 flow prediction
        input_1_warped_1 = tfa.image.dense_image_warp(input_1, flow_1_2)

        flow_change_1_2_concat = self.flow_1_2_concat([input_2, input_1_warped_1])
        flow_change_1_2 = self.flow_prediction_1_2(flow_change_1_2_concat)

        flow_1_2_changed = self.flow_add_1_2([flow_1_2, flow_change_1_2])
        input_1_warped_2 = tfa.image.dense_image_warp(input_1, flow_1_2_changed)
        flow_1_2_changed_upsampled = self.flow_upsample_1_2(flow_1_2_changed)

        # input_2 to input_1 flow prediction
        input_2_warped_1 = tfa.image.dense_image_warp(input_2, flow_2_1)

        flow_change_2_1_concat = self.flow_2_1_concat([input_1, input_2_warped_1])
        flow_change_2_1 = self.flow_prediction_2_1(flow_change_2_1_concat)

        flow_2_1_changed = self.flow_add_2_1([flow_2_1, flow_change_2_1])
        input_2_warped_2 = tfa.image.dense_image_warp(input_2, flow_2_1_changed)
        flow_2_1_changed_upsampled = self.flow_upsample_2_1(flow_2_1_changed)

        return input_1_warped_2, input_2_warped_2, flow_1_2_changed_upsampled, flow_2_1_changed_u
```

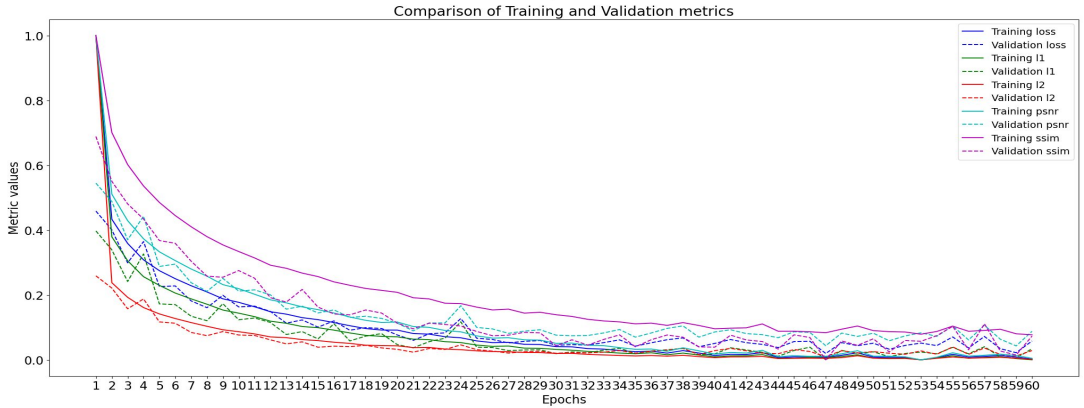
WarpedFeatureFusion




```
"""
arpedFeatureFusion is a layer that concatenates and fuses all the warped features
btained from the FlowEstimator layer. This layer generates the final output.
"""
lass WarpedFeatureFusion(layers.Layer):
    def __init__(self, feature_extractor_filter_count=[16, 32, 64], filter_size=(3, 3), activat
        super(WarpedFeatureFusion, self).__init__(**kwargs)
        fefc = feature_extractor_filter_count

        self.cnn_1st_level_o = layers.Conv2D(3, (1, 1), activation=output_activation, padding='
        self.add_1st_level = layers.Add()

        self.up_2nd_level = layers.UpSampling2D((2, 2), interpolation=interpolation)
        self.cnn_2nd_level_2 = layers.Conv2D(fefc[0], filter_size, activation=activation, kerne
        self.cnn_2nd_level_1 = layers.Conv2D(fefc[0], filter_size, activation=activation, kerne
        self.add_2nd_level = layers.Add()

        self.up_3rd_level = layers.UpSampling2D((2, 2), interpolation=interpolation)
        self.cnn_3rd_level_2 = layers.Conv2D(fefc[0] + fefc[1], filter_size, activation=activat
        self.cnn_3rd_level_1 = layers.Conv2D(fefc[0] + fefc[1], filter_size, activation=activat
        self.add_3rd_level = layers.Add()

        self.up_4th_level = layers.UpSampling2D((2, 2), interpolation=interpolation)
        self.cnn_4th_level_2 = layers.Conv2D(fefc[0] + fefc[1] + fefc[2], filter_size, activati
        self.cnn_4th_level_1 = layers.Conv2D(fefc[0] + fefc[1] + fefc[2], filter_size, activati
        self.add_4th_level = layers.Add()

        self.feature_extractor_filter_count = feature_extractor_filter_count
        self.filter_size = filter_size
        self.activation = activation
        self.regularizer = regularizer
        self.interpolation = interpolation

    def get_config(self):
        config = super().get_config()
        config.update({
            "feature_extractor_filter_count": self.feature_extractor_filter_count,
            "filter_size": self.filter_size,
            "activation": self.activation,
            "regularizer": self.regularizer,
            "interpolation": self.interpolation,
        })
        return config

    def call(self, inputs):
        input_1 = inputs[0]
        input_2 = inputs[1]

        # merge 4th level
        added_4th_level = self.add_4th_level([input_1[3], input_2[3]])
        cnn_4th_1 = self.cnn_4th_level_1(added_4th_level)
        cnn_4th_2 = self.cnn_4th_level_2(cnn_4th_1)
        up_4th = self.up_4th_level(cnn_4th_2)

        # merge 3rd level
        added_3rd_level = self.add_3rd_level([input_1[2], input_2[2], up_4th])
        cnn_3rd_1 = self.cnn_3rd_level_1(added_3rd_level)
        cnn_3rd_2 = self.cnn_3rd_level_2(cnn_3rd_1)
        up_3rd = self.up_3rd_level(cnn_3rd_2)

        # merge 2nd level
        added_2nd_level = self.add_2nd_level([input_1[1], input_2[1], up_3rd])
        cnn_2nd_1 = self.cnn_2nd_level_1(added_2nd_level)
        cnn_2nd_2 = self.cnn_2nd_level_2(cnn_2nd_1)
        up_2nd = self.up_3rd_level(cnn_2nd_2)

        # merge 1st level
        added_1st_level = self.add_1st_level([input_1[0], input_2[0], up_2nd])
```

Initial model tests





Initial model tests

First frame

Predicted and Ground-truth difference

First frame

Predicted and Ground-truth difference

Predicted frame

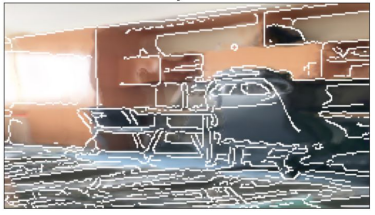
Ground-truth frame

Predicted frame

Ground-truth frame

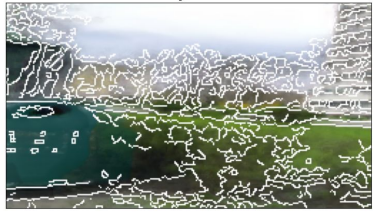
Second frame

Edge shift



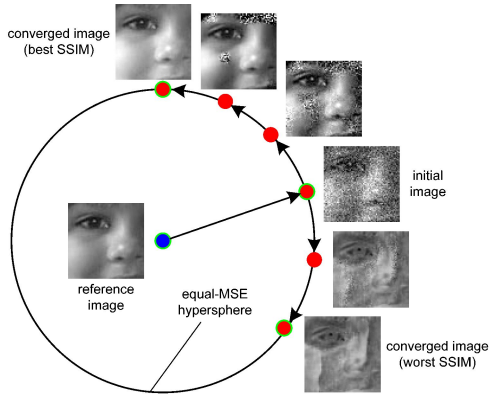
Second frame

Edge shift

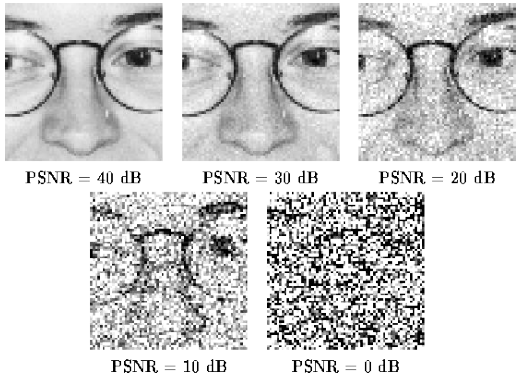


Loss function

SSIM:



PSNR:



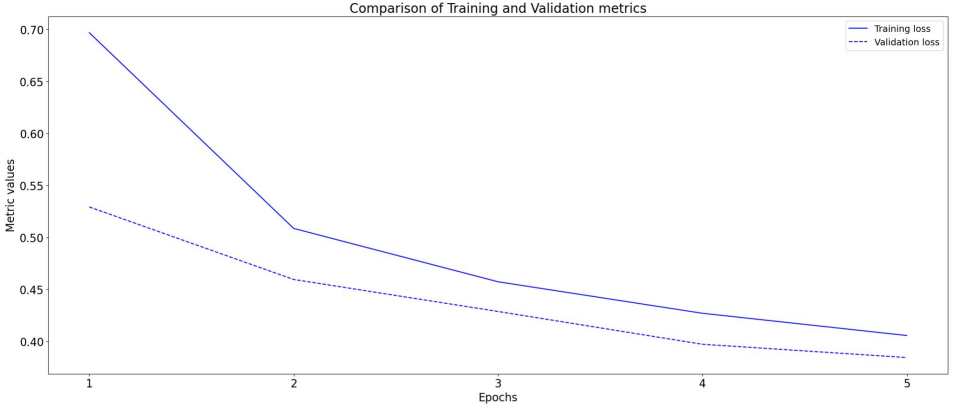
L1: $$|\mathbf{x}|_1 = \sum_{r=1}^{n} |x_r|.$$

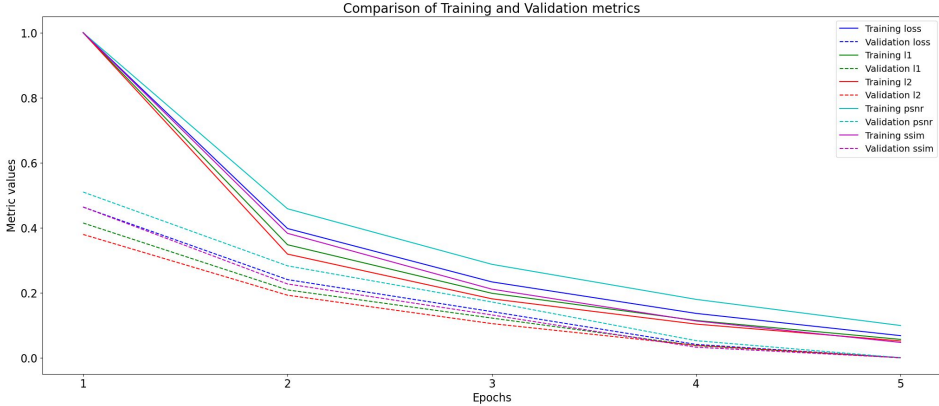
L2: $$|\mathbf{x}| = \sqrt{\sum_{k=1}^{n} |x_k|^2},$$

Loss = SSIM + PSNR + 5.0*L1 + 10.0*L2

Trening

- TFRecords for processing 19000 training samples (25.5 GB)
- Nadam optimizer (lr = 0.0001, clipvalue=1.0, clipnorm=1.0)
- Epoch time: 22 minutes (1xGTX970 4GB, i5 4690K)
- Model parameters: 1 199 971

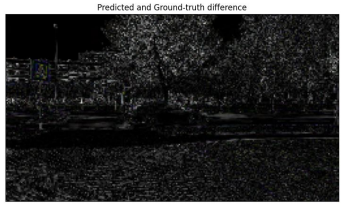


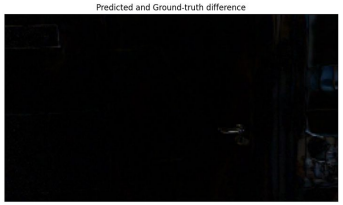


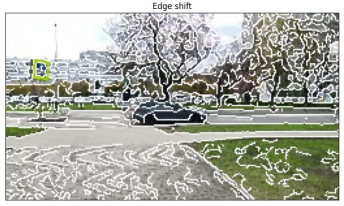
Results

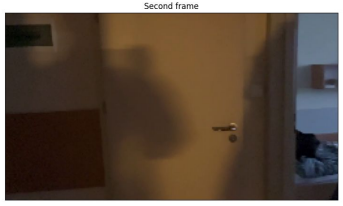
Results for a test set of Vimeo90K in resolution 144 x 256 px

- PSNR: 30.44 (SOTA: 36.76)
- SSIM: 0.9096 (SOTA: 0.9800)









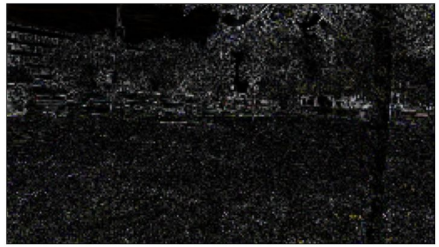
Results

First frame

Predicted and Ground-truth difference

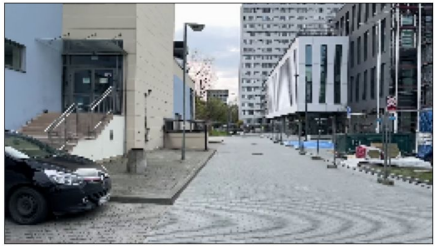
First frame

Predicted and Ground-truth difference



Predicted frame

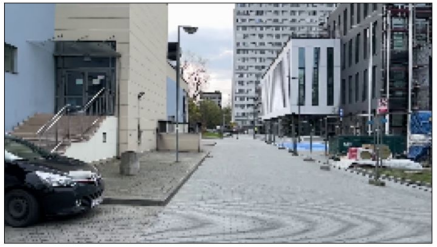
Ground-truth frame



Predicted frame

Ground-truth frame

Second frame



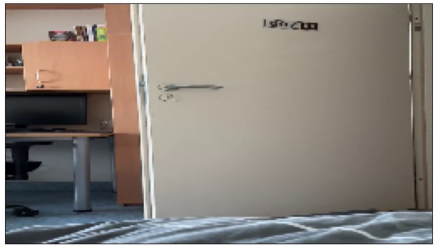
Edge shift

Second frame

Edge shift

Results

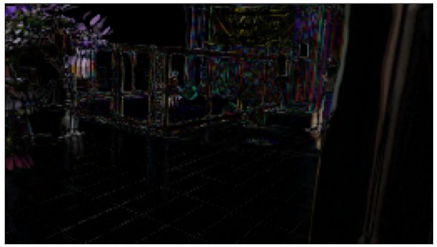
First frame



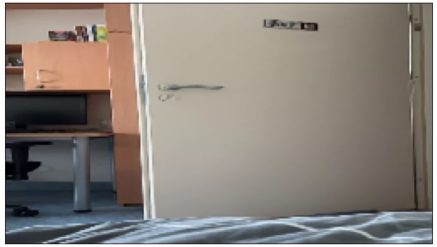
Predicted and Ground-truth difference

First frame

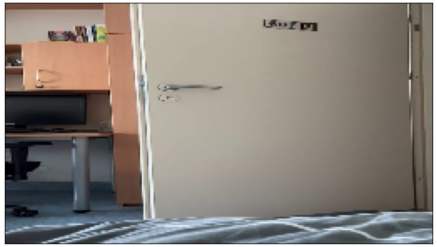
Predicted and Ground-truth difference



Predicted frame



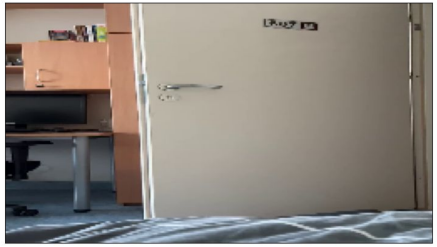
Ground-truth frame



Predicted frame

Ground-truth frame

Second frame



Edge shift

Second frame

Edge shift

Model issues

First frame

Predicted and Ground-truth difference

First frame

Predicted and Ground-truth difference

Predicted frame

Ground-truth frame

Predicted frame

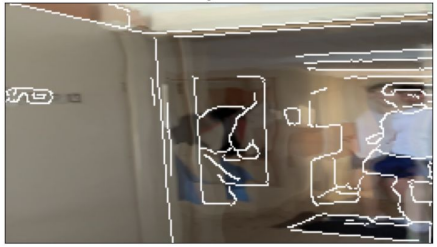
Ground-truth frame

Second frame

Edge shift

Second frame

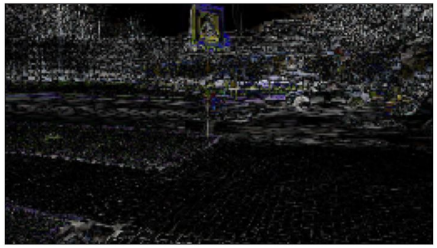
Edge shift



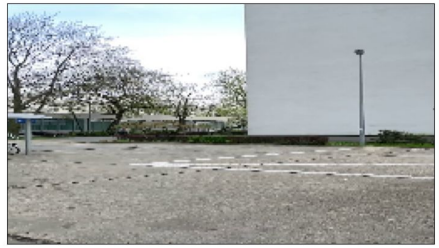
Model issues

First frame

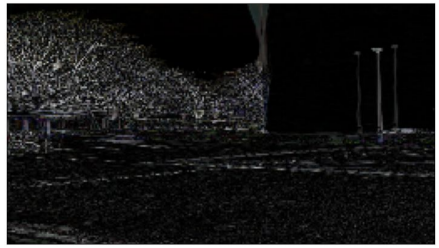
Predicted and Ground-truth difference



First frame

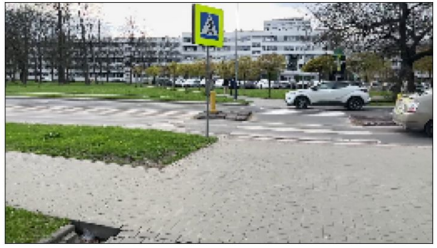


Predicted and Ground-truth difference



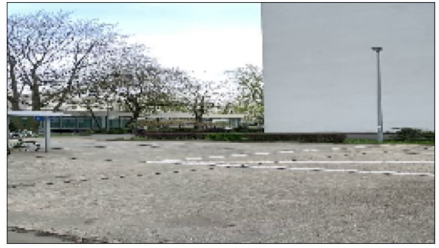
Predicted frame

Ground-truth frame



Predicted frame

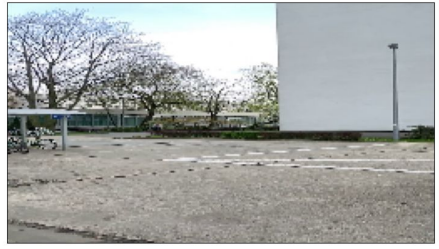
Ground-truth frame



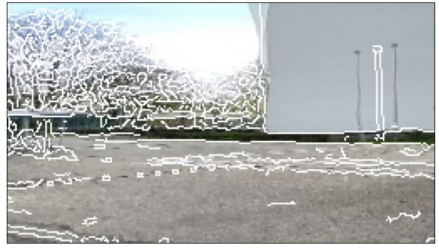
Second frame

Edge shift

Second frame



Edge shift



Project development

- Teach the model on a dataset with a bigger motion
- Fix described model issues
- Create an environment for an end-to-end frame-boosting application

Resources

1. Single Image Super Resolution with deep convolutional neural networks
2. Real-Time Intermediate Flow Estimation for Video Frame Interpolation
3. Depth-Aware Video Frame Interpolation
4. BiFormer: Learning Bilateral Motion Estimation via Bilateral Transformer for 4K Video Frame Interpolation
5. Attention is all you need
6. Video Frame Interpolation via Adaptive Convolution
7. Large Motion Frame Interpolation
8. FILM: Frame Interpolation for Large Motion
9. Multi-view Image Fusion
10. Perceptual Losses for Real-Time Style Transfer and Super-Resolution
11. Image Style Transfer Using Convolutional Neural Networks
12. Exploring Motion Ambiguity and Alignment for High-Quality Video Frame Interpolation
13. PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume