MOTION ESTIMATION USING CONVOLUTIONAL NEURAL NETWORKS

Mustafa Ozan Tezcan

Boston University
Department of Electrical and Computer Engineering
8 Saint Mary's Street
Boston, MA 02215
www.bu.edu/ece

Dec. 19, 2107

Technical Report No. ECE-2017-04

Contents

List of Figures

List of Tables

Diagram Shortcuts

Since convolutional neural networks include too many repetitions, I used some shortcuts for the common operations. Figures 1 - 4 shows the shortcuts.

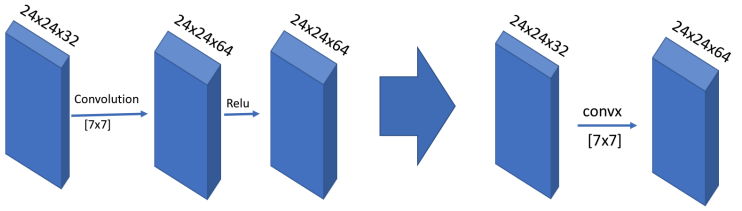

Figure 1: Convolution + ReLU

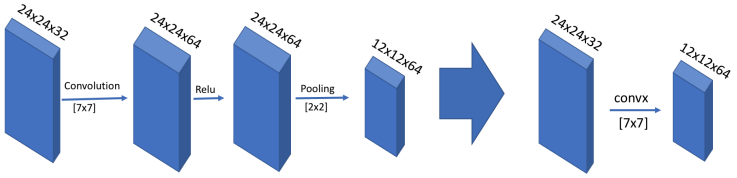

Figure 2: Convolution + ReLU + Pooling

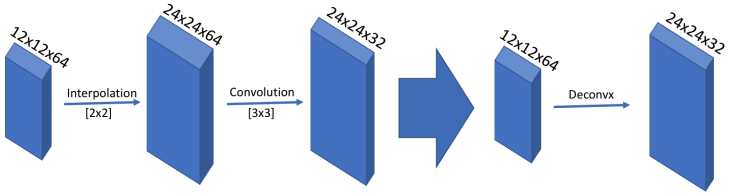

Figure 3: Deconvolution. Note that all deconvolution layers use 2×2 interpolation and 3×3 kernel in convolution.

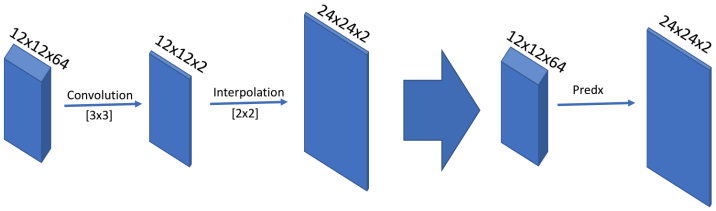

Figure 4: Optical Flow Prediction.

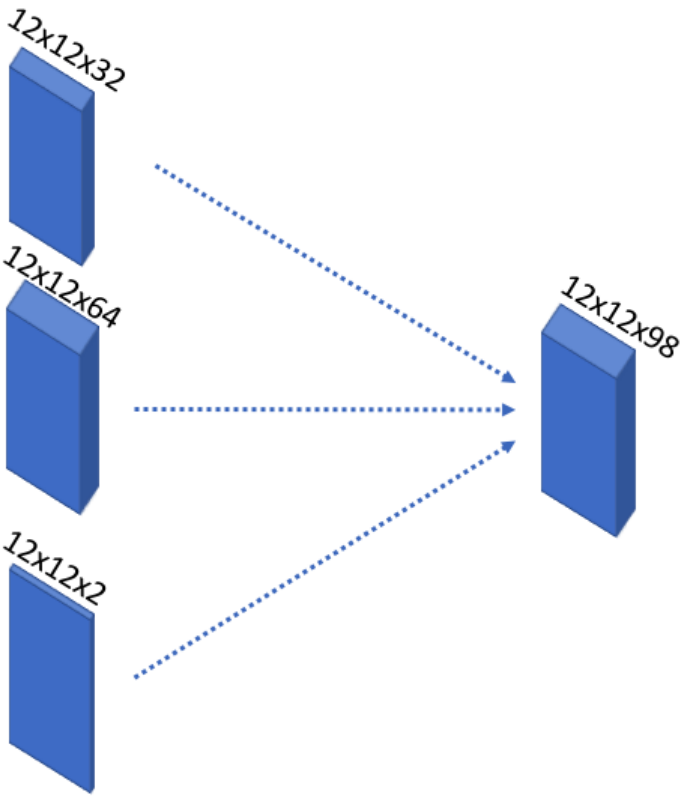

Figure 5: Concatenation

1 Introduction

In recent years, deep learning has been applied to many image processing problems such as image classification, object recognition, semantic segmentation etc. and achieved state-of-the-art results. However, most of those methods are single image based. When it comes to video processing, deep learning based methods are not so popular, yet. In this project, I have used deep learning to design a motion estimation network between two frames. This problem is being addressed in several deep learning papers. One of the best ones is Flownet [1, 2]. Flownet is a convolutional neural network which takes an image pair as an input and tries to predict optical flow which is given as ground truth. However, there is no ground truth optical flow for real images and the amount of the synthetic training data is not sufficient. So, the authors designed a new dataset which is called flying chairs [1] and consists of 22.872 image pairs with their optical flows. In the dataset, they used real images as backgrounds and synthetic chairs as moving objects. The network was trained on this dataset and then fine-tuned on Sintel [3] and KITTI [4] datasets.

2 Literature Review

Model-based approaches dominated the motion estimation problem for years. Some of the most popular motion estimation algorithms are block matching, phase correlation, Lucas Kanade and Horn Schunk (HS) methods [5]. These methods use different mathematical formulations to model the motion. Even though they are very successful for some cases, their performance is limited by model limitations. For example, most of the model-based approaches start by modeling motion in the continuous domain, so they assume small motion between the input frames. For overcoming these model limitations, deep learning based algorithms have been proposed [1, 6, 7]. Even though these methods don't have model limitations, they suffer from data limitations. In this project, I investigated FlowNet [1] and compared it with HS.

3 Problem Definition

I defined motion estimation as a supervised learning problem. Given a set of frame pairs and their optical flows, we can design a learning-based method for motion estimation. Figure 6 illustrates this approach.

Let $\{\mathbf{F1}_k, \mathbf{F2}_k \in \mathbb{R}^{w\times h\times 3}, \mathbf{OF}_k \in \mathbb{R}^{w\times h\times 2}\}_{k=1}^{N}$ represent our dataset, where $\mathbf{F1}_k$ and $\mathbf{F2}_k$ represent the frame pair and $\mathbf{OF}_k$ is the optical flow from $\mathbf{F1}_k$ to $\mathbf{F2}_k$. We aim to design an algorithm $\mathbf{H}(\mathbf{F1}_k, \mathbf{F2}_k) = \widehat{\mathbf{OF}}_k$ to predict the optical flow of an unseen frame pair. For measuring the performance of the algorithm, we can use end point error (EPE) or motion compensated prediction error (MCPE) which are defined below.

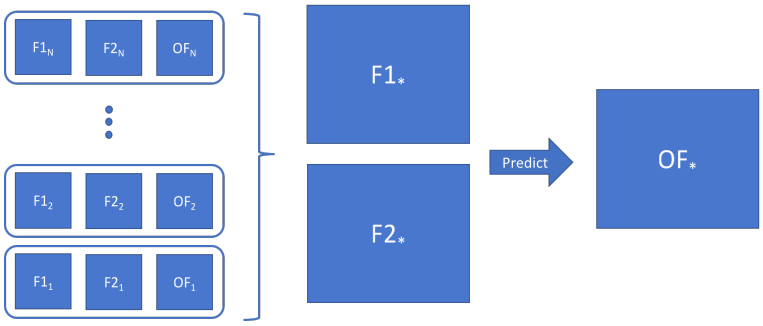

Figure 6: Motion estimation via supervised learning

$$EPE = \sum_{k=1}^{N} |\widehat{\mathbf{OF}}_k - \mathbf{OF}_k|_F^2 \tag{1}$$

where $|\mathbf{X}|_F$ is the Frobenius norm defined by the square root of the sum of the squared elements of $\mathbf{X}$.

$$MCPE = \sum_{k=1}^{N} |warp(\mathbf{F1}_k, \widehat{\mathbf{OF}}_k) - \mathbf{F2}_k|_F^2 \tag{2}$$

here $warp(\mathbf{F1}_k, \widehat{\mathbf{OF}}_k)$ returns the warped version of $\mathbf{F1}_k$ with respect to the motion vectors in $\widehat{\mathbf{OF}}_k$ using linear interpolation.

4 Proposed Solution: FlowNet

In this project, I replicated the simpler version of the work described in [1]. The authors made two contributions. Their first contribution is to design of a new dataset called "Flying Chairs" and their second contribution is the implementation of two different convolutional neural networks (CNN) for motion estimation.

4.1 Dataset

There are several existing dataset for evaluation of motion estimation algorithms [3, 4], however they don't have too many input frames. For optimizing the weights of a CNN, one needs a large dataset. One of the options is to create a dataset with optical flow ground truth, but this a very complicated problem. So the authors decided to design a synthetic dataset called "Flying Chairs" [1]. They used real background images and added synthetic chair images on top of those backgrounds. Then, they applied randomly generated affine morion to the backgrounds and 3D affine motion to the individual chair objects for creating the optical flow field and the frame pairs. Using this approach, they created 22.782 frame pairs with their optical flow ground

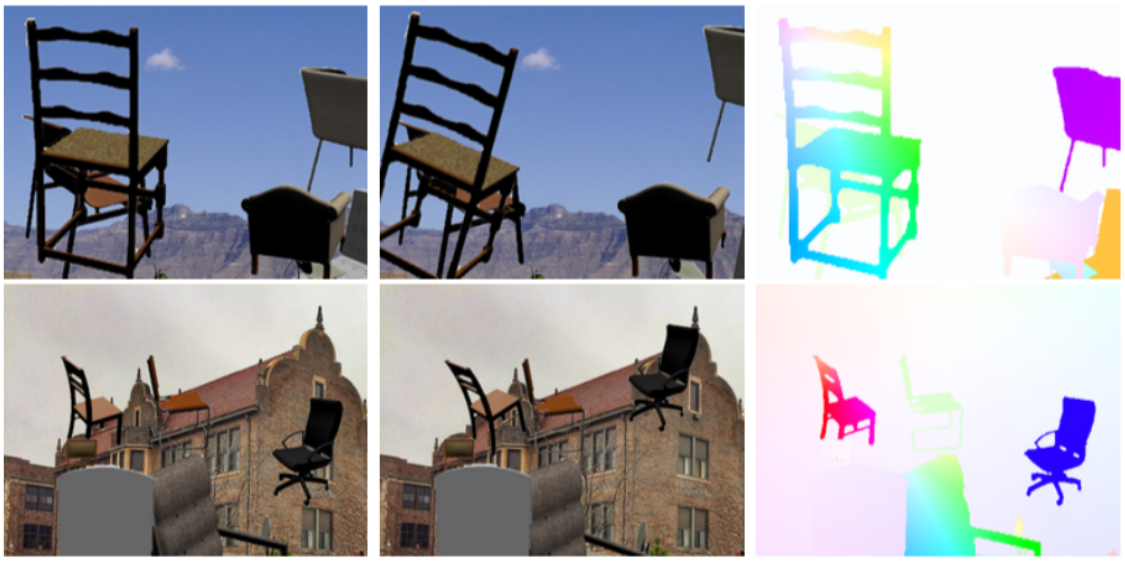

Figure 7: Flying Chairs Dataset. First two columns show frame pairs and the last column shows the color coded optical flows (Taken from [1])

truths. The size of the frames is 384x512. Figure 7 shows some examples from the dataset.

For visualization of the optical flow, I used color coding used in [3]. The direction of the motion is being represented by different colors, and the magnitude is represented by the intensity of that color. Figure 8 shows the color coding.

4.2 Network Architectures

FlowNet has two different variants. The first one is called FlowNet-Simple (FNS). It takes the concatenation of two frames as an input and predicts the optical flow matrix. So for 3-channel frames (i.e. RGB), it takes $W \times L \times 6$ dimensional array as an input and produces $W \times L \times 2$ dimensional output for the vertical and horizontal coordinates of the optical flow. Figure 9 shows the overview of FNS network. The second variant is called FlowNet-Correlation (FNC). It takes two frames as two separate inputs of size $W \times L \times 3$ and concatenates their embeddings in the middle layers using a correlation function. In my project, I did not use the original networks since they were too deep and it was taking too long to train an algorithm using them. I decimated the input frames to 192×256 and took 48×48 sized random crops from the decimated frames. So my networks are trained with $48 \times 48 \times 3$ dimensional frame pairs and their $48 \times 48 \times 2$ optical flow matrices.

Figure 8: Color coding of the motion vectors (Taken from [1])

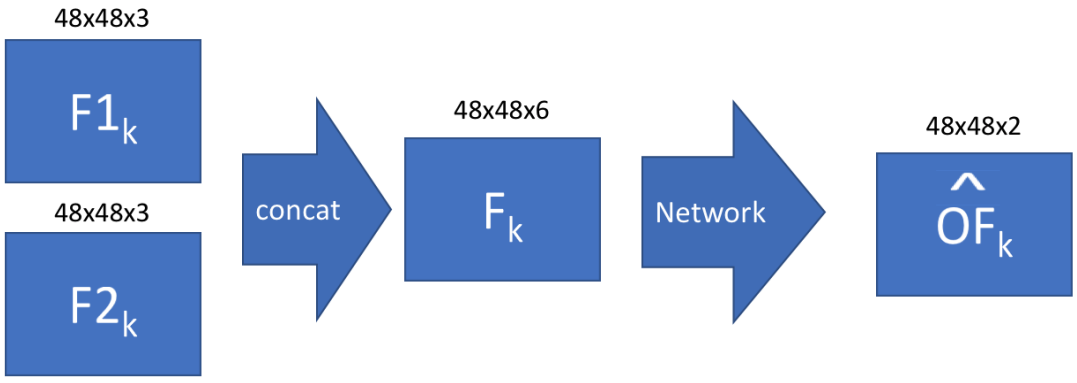

Figure 9: Overview of FNS

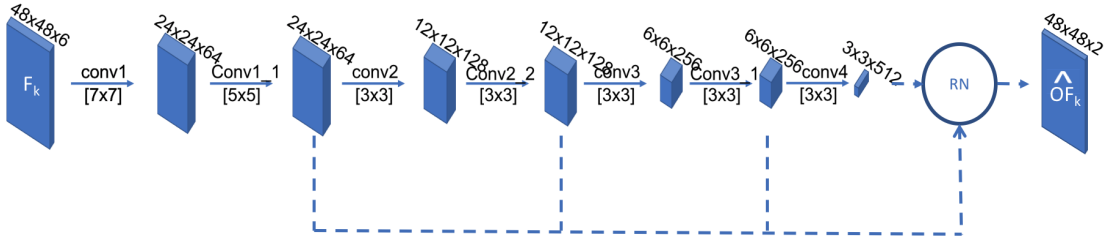

Figure 10: Details of FNS

4.2.1 FlowNet-Simple (FNS)

Figure 10 shows the details of FNS. The architecture is very similar to an autoencoder. The idea is to first extract some low dimensional features from the image by using a fully convolutional network (no fully connected layers) and then predicting the optical flow fields from these features by using a fully deconvolutional network. I will call the convolutional part Feature Extractor Network (FEN) and the deconvolutional part Refinement Network (RN). FEN is very similar to classification networks as it uses convolution, pooling, batch normalization and nonlinear activation layers. The size of the convolution filters starts from 7×7 and decreases to 3×3.

RN uses the compressed information from FEN to get a motion prediction. Since the spatial dimensions decreased in FEN, RN uses deconvolutional networks to increase the spatial dimensions. Figure 11 shows a more detailed version of FNS including the refinement network. In the early research of deep learning, it was observed that the initial layers of the networks learn the very fundamental features related to images such as edge representations and those kinds of features are used frequently in many computer vision tasks including motion estimation. So, it is a good idea to use these feature representations in the RN part of the network to get a better motion estimation. Note that this approach is not usable in the classification-based CNNs since the sizes of the last layers are smaller than the first ones. Every layer of RN is being concatenated with a layer of the same size from FEN and also with the optical flow estimation from the previous layer. For example, let's look at the layer after *Deconv2* operation. *Deconv2* produces an output with 64 channels and this output is being concatenated with the output of *Conv2_1* which has 128 channels and also with the output of *Pred3* which has 2 channels. So, overall the size of this layer is 194. As a result, RN produces several predictions of the optical flow in different sizes. Predictions are computed at $(48 \times 2^{-n}) \times (48 \times 2^{-n})$ resolutions where $n = 0, 1, 2, 3$. So, in a way RN uses a similar approach to pyramid networks.

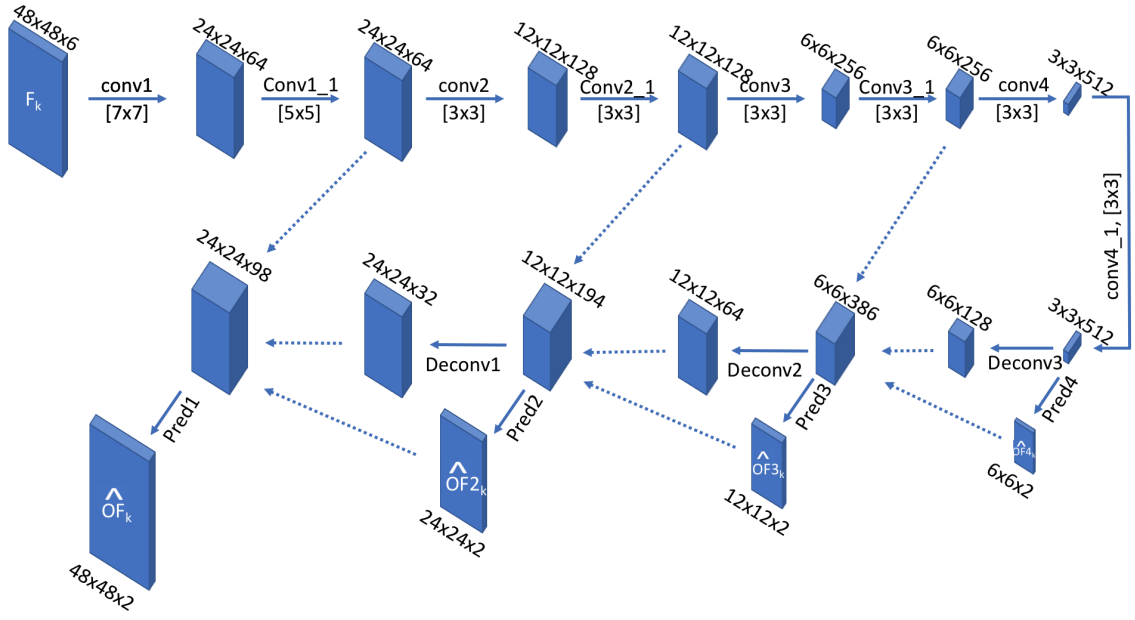

Figure 11: FNS with detailed refinement network (RN)

4.2.2 FlowNet-Correlation (FNC)

In FNC, authors split the FEN part into two siamese networks (sharing the same weights) for two frames and used a correlation layer to combine these two networks. This procedure is shown in Figure 12. The correlation layer compares different patches from feature representation of the input frames.

Given two feature maps $\mathbf{f}_1, \mathbf{f}_2 : \mathbb{R}^2 \rightarrow \mathbb{R}^c$ where c is the number of channels and $\mathbf{f}_i(\mathbf{x})$ is a c dimensional feature vector of i^{th} frame's feature representation at location $\mathbf{x} = [x_1, x_2]^T$ (In MATLAB notations $\mathbf{f}_i(x_1, x_2) = \mathbf{F}_i(x_1, x_2, :)$ where $\mathbf{F}_i \in \mathbb{R}^{w \times h \times c}$ is the feature representation of the i_{th} frame),

$$corr[\mathbf{x}, \mathbf{d}] = \sum_{\mathbf{o} \in [-k,k] \times [-k,k]} \langle \mathbf{f}_1(\mathbf{x} + \mathbf{o}), \mathbf{f}_2(\mathbf{x} + \mathbf{o} + \mathbf{d}) \rangle \tag{3}$$

where $\mathbf{d} = [d_1, d_2]^T$ is the displacement, $K = 2k + 1$ is the window size, and $\langle ., . \rangle$ denotes the inner product. In this project, I used $k = 0$ (single pixel correlation).

Note that this computation needs to be done for every $\mathbf{x}$ and $\mathbf{d}$ pair which will result in w^2h^2 elements in total where w and h represents the spatial dimensions of the feature representation. However, it is possible to reduce the number of elements by limiting the amount of motion. So, I calculated Eq. 3 for only $|d_1|, |d_2| \leq 7$. Then I reshaped the output as a $3D$ matrix with $((2 \times 7) + 1)^2 = 225$ channels ($\mathbb{R}^{w \times h \times 15 \times 15} \rightarrow \mathbb{R}^{w \times h \times 225}$).

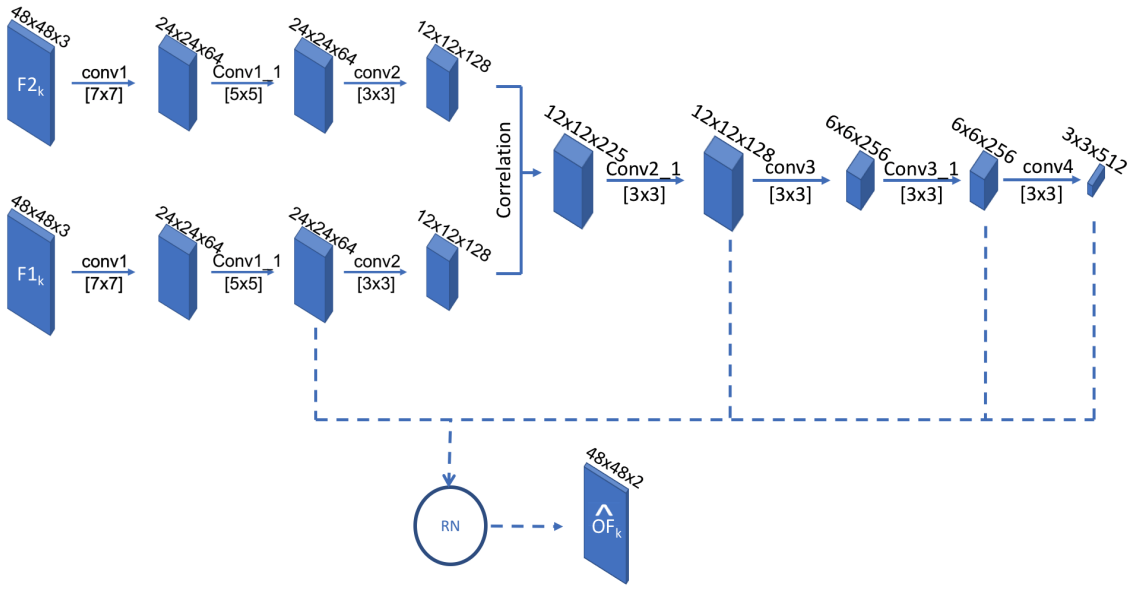

Figure 12: Details of FNC

4.2.3 Loss Function

As explained in Section 4.2.1, the refinement network is used for computing several motion predictions in different spatial dimensions. Of course, only the last one can be used as the overall prediction of motion since it is the only one which has the same size as input frames. However, while defining the loss function we have to use all of the predictions to make sure that they all give a reasonable motion prediction (i.e. account for motion features at different scales). So the overall loss function of the network is formulated as below.

$$\mathcal{L} = \sum_{k=1}^{N} \big(|\widehat{\mathbf{OF}}_k - \mathbf{OF}_k|_F^2 + |\widehat{\mathbf{OF2}}_k - \mathbf{S}_2(\mathbf{OF}_k)|_F^2 + |\widehat{\mathbf{OF3}}_k - \mathbf{S}_4(\mathbf{OF}_k)|_F^2 + |\widehat{\mathbf{OF4}}_k - \mathbf{S}_8(\mathbf{OF}_k)|_F^2 \big) \tag{4}$$

where $\mathbf{S}_m(.)$ means $(m \times m)$ decimation.

5 Experimental Results and Discussion

In my experiments, I used 80% of the resized frame pairs as the training set and the remainder as the test set. Both algorithms are trained with ADAM optimizer with a learning rate of 0.0001, momentum of 0.9, weight decay of 0.0004 and mini-batch size of 32. I applied random data transformations composed of cropping, flipping and rotating the images.

Table 1: Comparison of EPE results using my network with those form the original paper [1].

Method	Test EPE
FNS	3.72
FNC	3.28
FNS-original	2.71
FNC-original	2.19

Table 1 shows the comparison of EPE results for original FNS and FNC (taken from the FlowNet paper) and my shallower implementations. Even though I used a much smaller input size it looks like, I achieved a close performance to the original implementations.

In Figures 13 - 25, I compared the results of my implementation with the Horn Schunk (HS) algorithm. Figures 13 - 16 show MCPE and EPE results for two different frame pairs from the test set. In terms of the EPE performance, both FNS and FNC outperforms HS which was expected since the HS algorithm does not know anything about the ground truth. However, in terms of the MCPE performance, HS performed better than FNS and slightly worse than FNC. This result shows that there is no direct relationship between MCPE and EPE. Even though HS performed poorly in terms of the EPE, it still performs reasonably well according to the MCPE. One can realize that, even for the ground truth, MCPEs are close 1000. The main reason for this big ground truth MCPE is the significant amount of occlusion regions caused by the large motion between frames. In some sense, the ground truth MCPEs are the lower bounds for FNS and FNC since they are trying to achieve the ground truth. Finally, in the color-coded motion visualizations, it can be observed that FNS and FNC predict more patch-wise constant motion field than HS. So, FNS and FNC predictions may be used in motion based image segmentation.

For examining the generalization performance of FNS and FNC, I also tested the algorithms on *Miss America* frames. Figures 17 - 20 show the comparisons for neighboring frames and 4 frames apart. For neighboring frames, FNC performed poorly and HS outperformed both of FNS and FNC. It is very surprising. One can observe that in the flying chairs examples, the initial MCPE between two frames is on the order of 1000, however for *Miss America* with one frame difference, the initial MCPE is only 30. I investigated most of the examples in flying chairs and it seems like most of those examples have a significant amount of motion (much bigger than for neighboring frames at 30 FPS). So, FNS and FNC don't know how to predict small motion and fail to predict it. Another interesting fact is that FNS performed better than FNC for small motion. It can be explained by memorization. Probably, in some of the layers, FNC learned to predict motion vectors with big magnitudes. When I increased the frame spacing from 1 to 4, FNS and FNC outperformed HS. After seeing theses results, I decided to compare the algorithm's MCPE performance with respect to increasing motion.

I downloaded a $360 \times 640, 30$ FPS video from a talk show. I chose this video because it is somewhat similar to the flying chairs dataset. It has a moving background, moving person in the middle of the screen and also moving audience. Figures 21 - 25 show the performance comparison for differently spaced frames. It can be observed that the performance of FNS and FNC increased with increasing frame spacing and FNC outperformed HS when the frame spacing is bigger than 5, however HS outperformed FNS in nearly all of these examples. This fact shows the importance of adding a model-based information (correlation layer) to a learning-based method. One interesting region in these frames is the channel logo on the upper-left (also the channel name in the upper right). Even though the background is moving, the channel logo remains in the same position. HS estimates the motion in the logo perfectly (since it is zero motion), however FNC fails to predict it. I guess, FNC tries to predict a patch-wise constant motion field and so it fails for small details.

6 Challenges and Future Work

For me, the biggest challenge of this project was the network design. In the beginning it seemed like simple CNN architecture however it has lots of details which makes it harder to design the network using deep learning libraries. Some of the ideas for the future work are listed below.

- FlowNet architectures try to optimize the weights for EPE. However, one can try to optimize it for prediction error as well. This will result in unsupervised learning problem where we don't have any ground truth.
- Recurrent neural networks can be used to estimate the motion in a video.
- FNS and FNC can be combined to get a better motion prediction
- The networks can be fine-tuned on real optical flow datasets like KITTI [4].
- We can add more frame pairs with small motion to training data for increasing the performance of the algorithm for small motion.
- We can try to use RNNs for finding an iterative solution. For example, in the first step, we can give 3 inputs to the network: 2 frames and an estimation for the optical flow which can be calculated by classical methods such as block matching or Horn and Schunck. In the following steps, the network will take the first frame, the warped version of the second frame and the motion estimation from the previous iteration. Actually, I was planning to try this improvement, but replicating the original paper took more time than I expected.

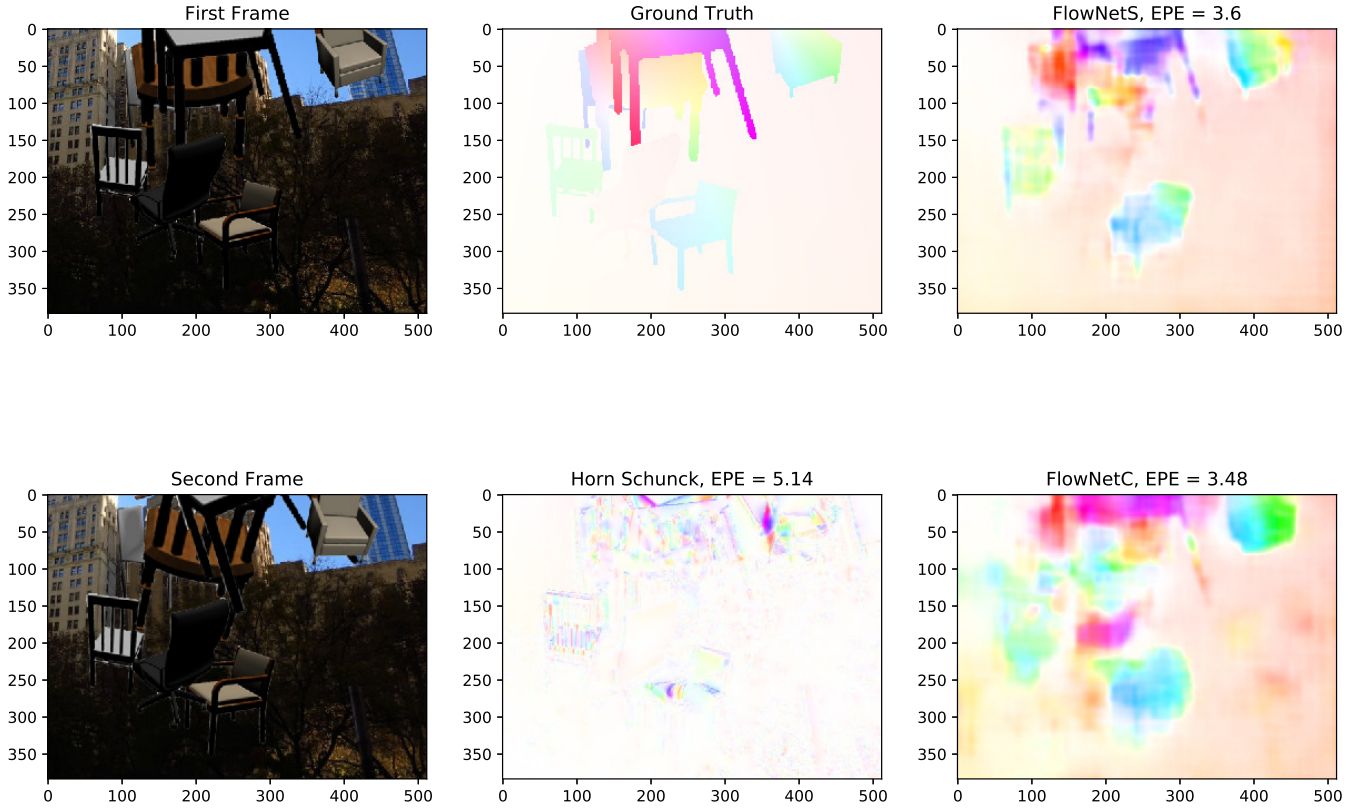

Figure 13: Motion estimation results for an example test image

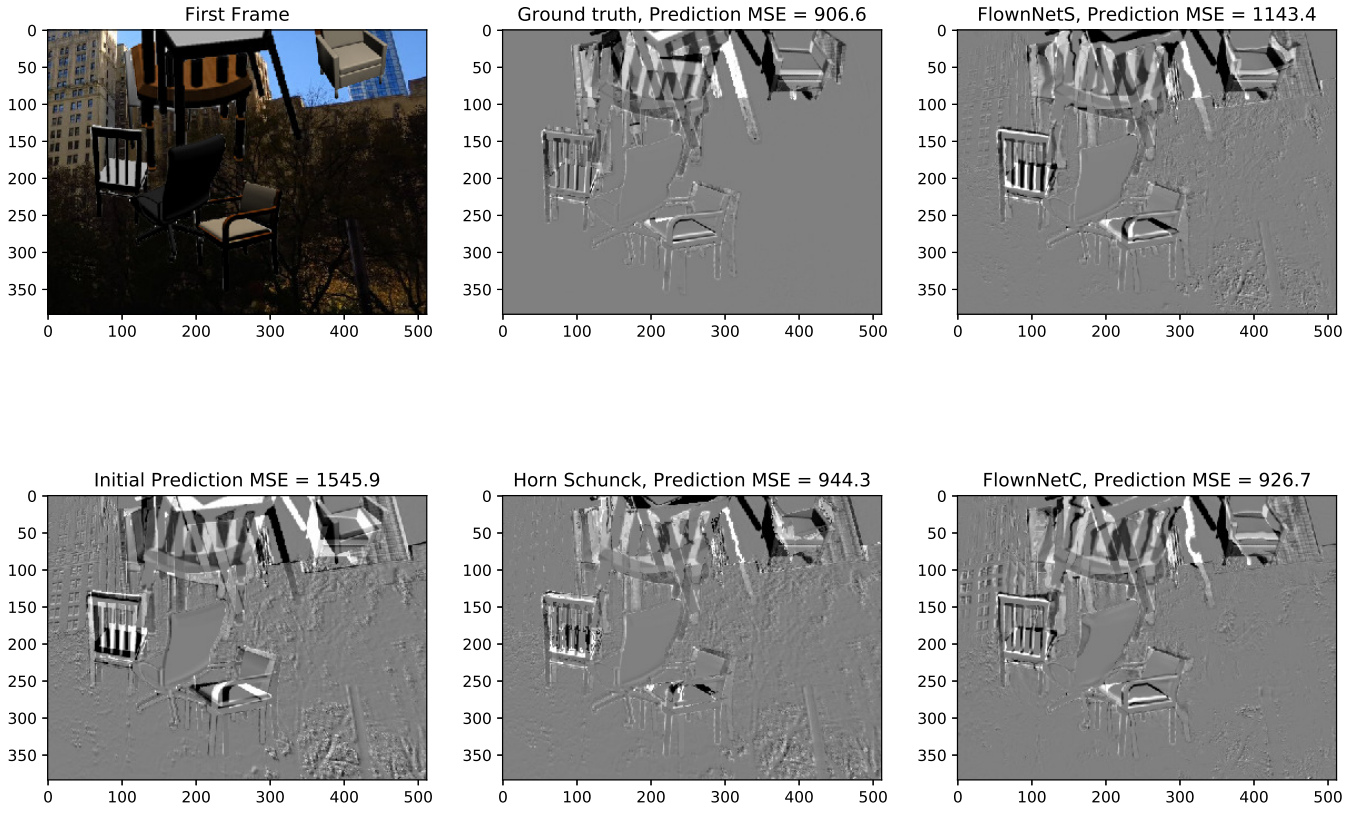

Figure 14: MCPE results for an example test image

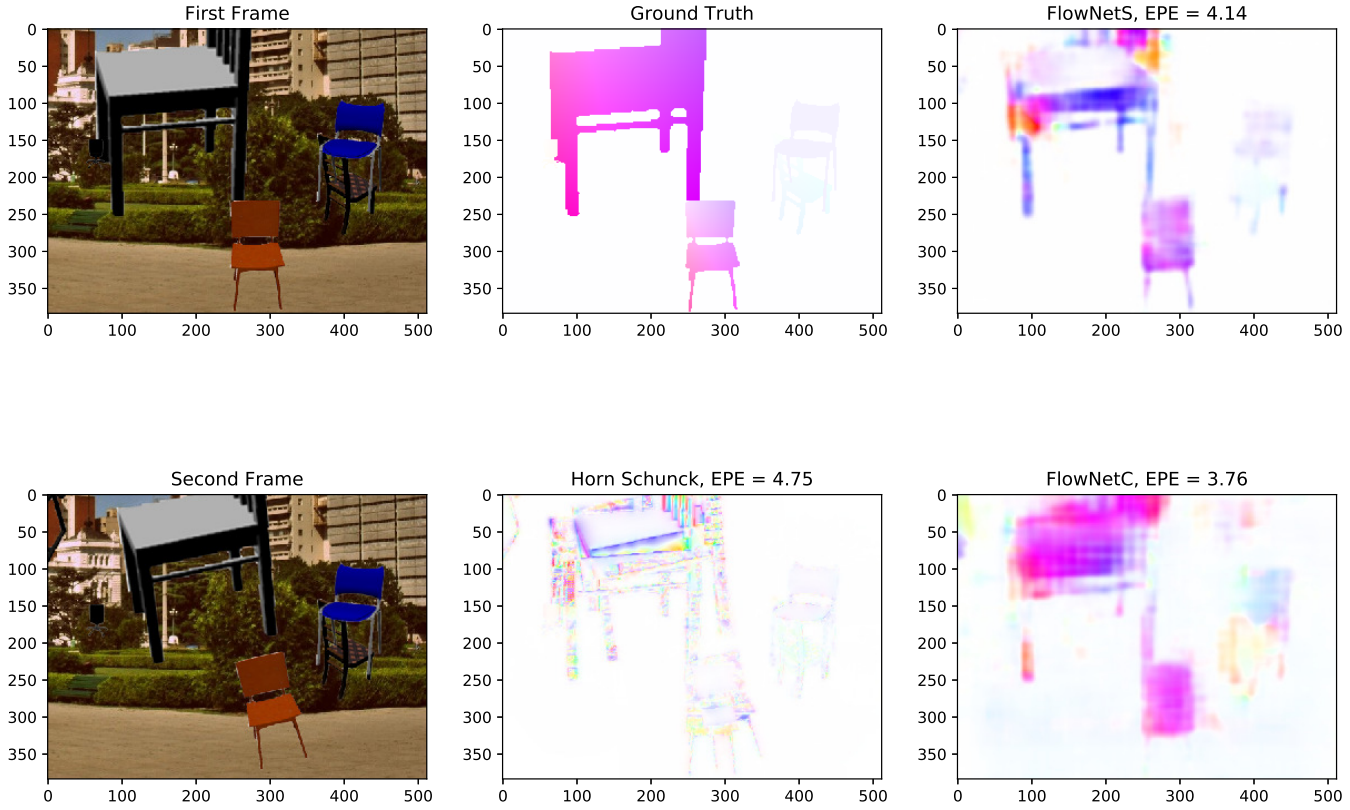

Figure 15: Motion estimation results for an example test image

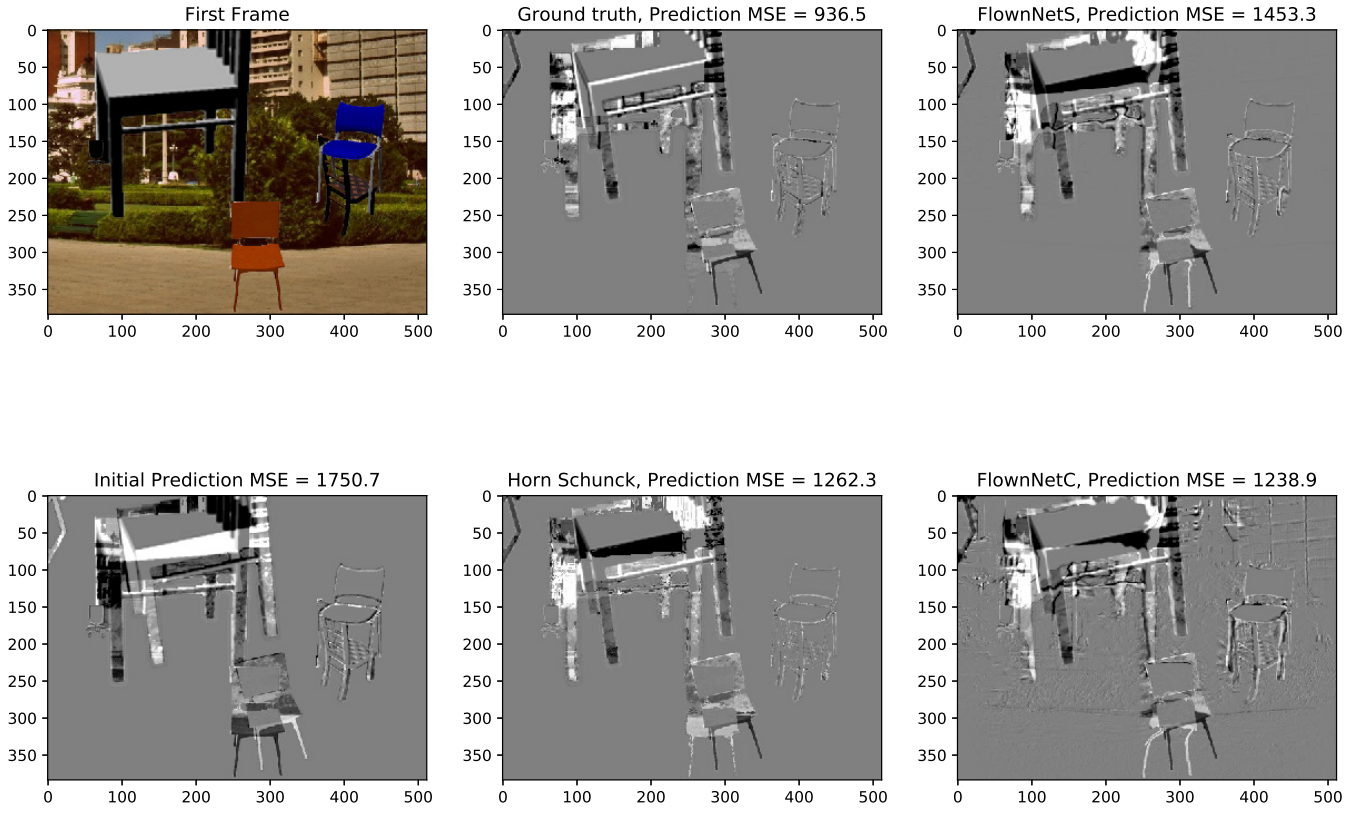

Figure 16: MCPE results for an example test image

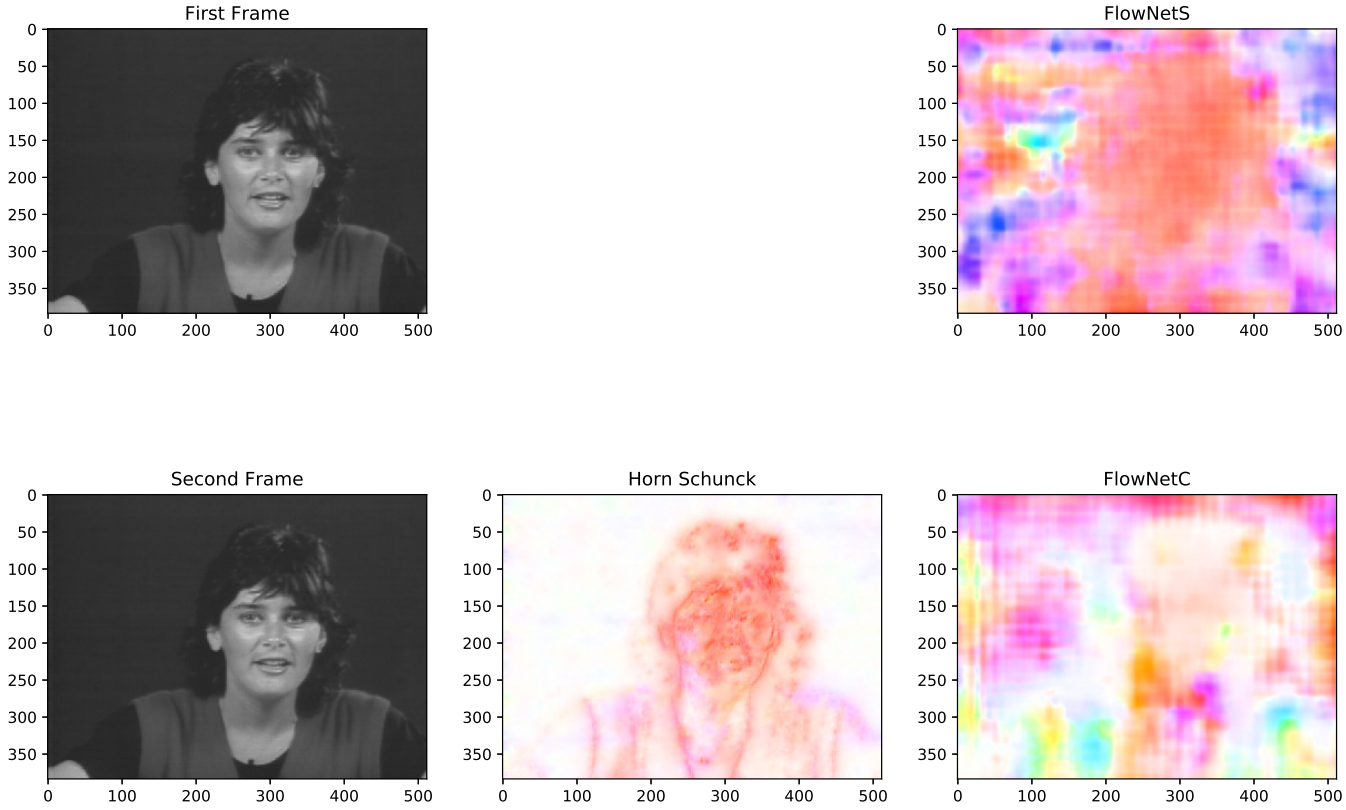

Figure 17: Motion estimation results for Miss America - neighboring frames

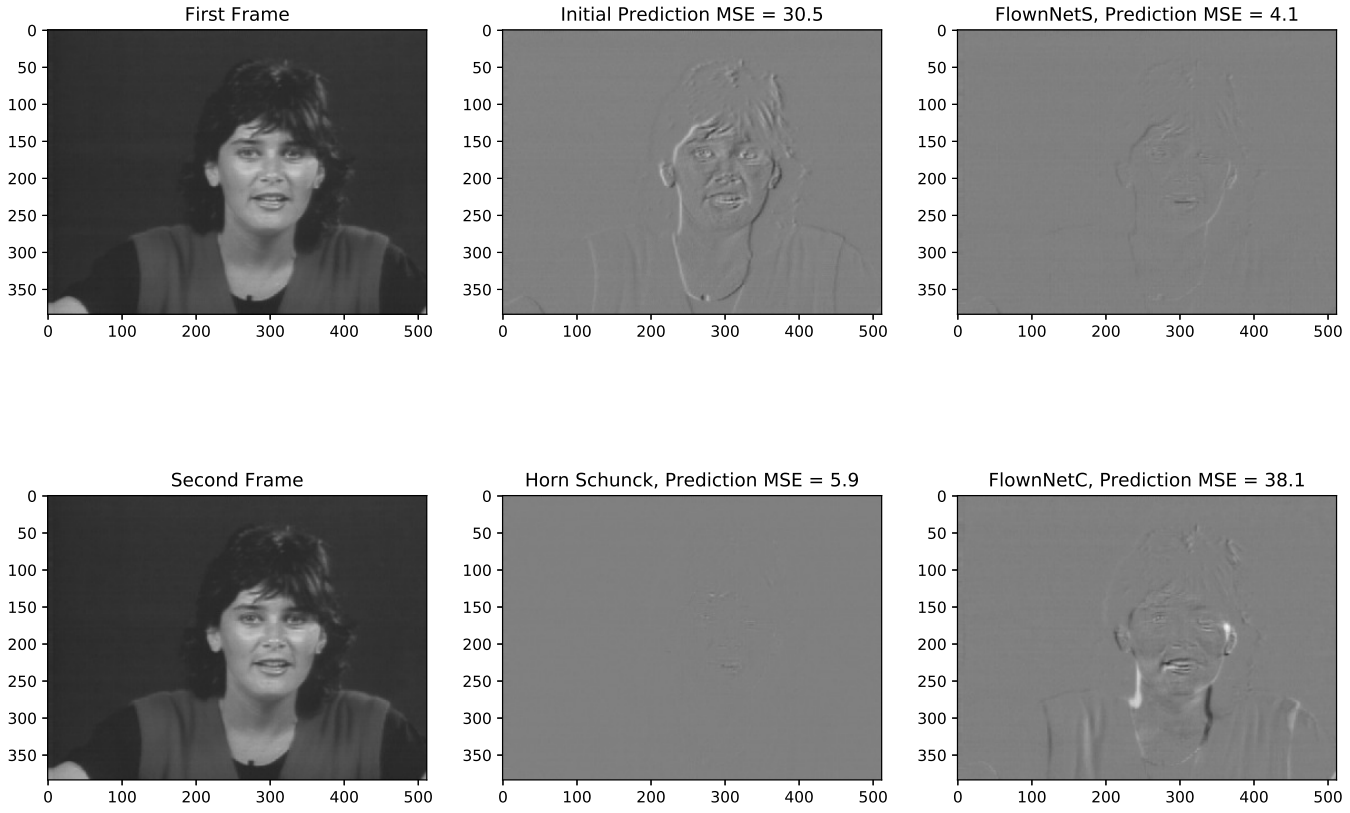

Figure 18: MCPE results for Miss America - neighboring frames

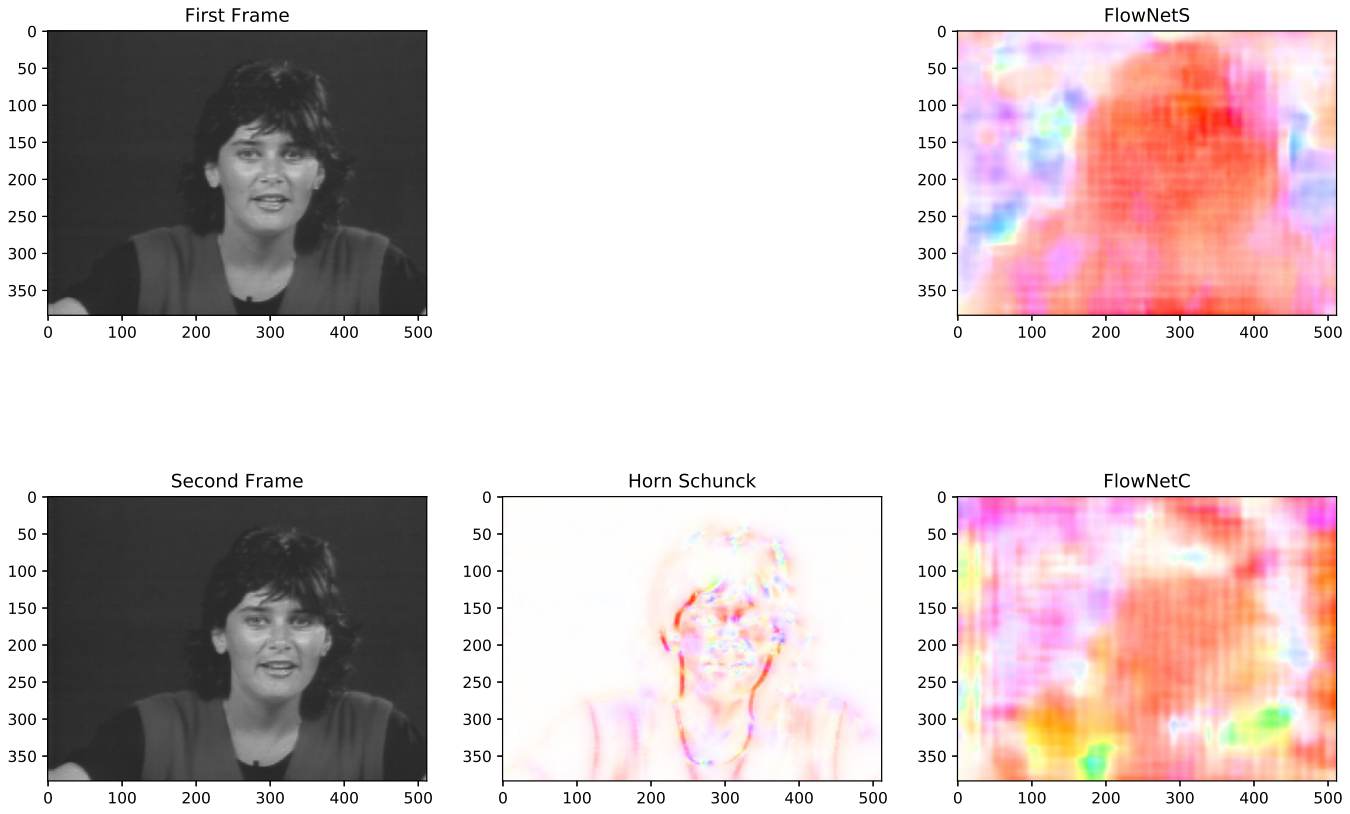

Figure 19: Motion estimation results for Miss America - 4 frames apart

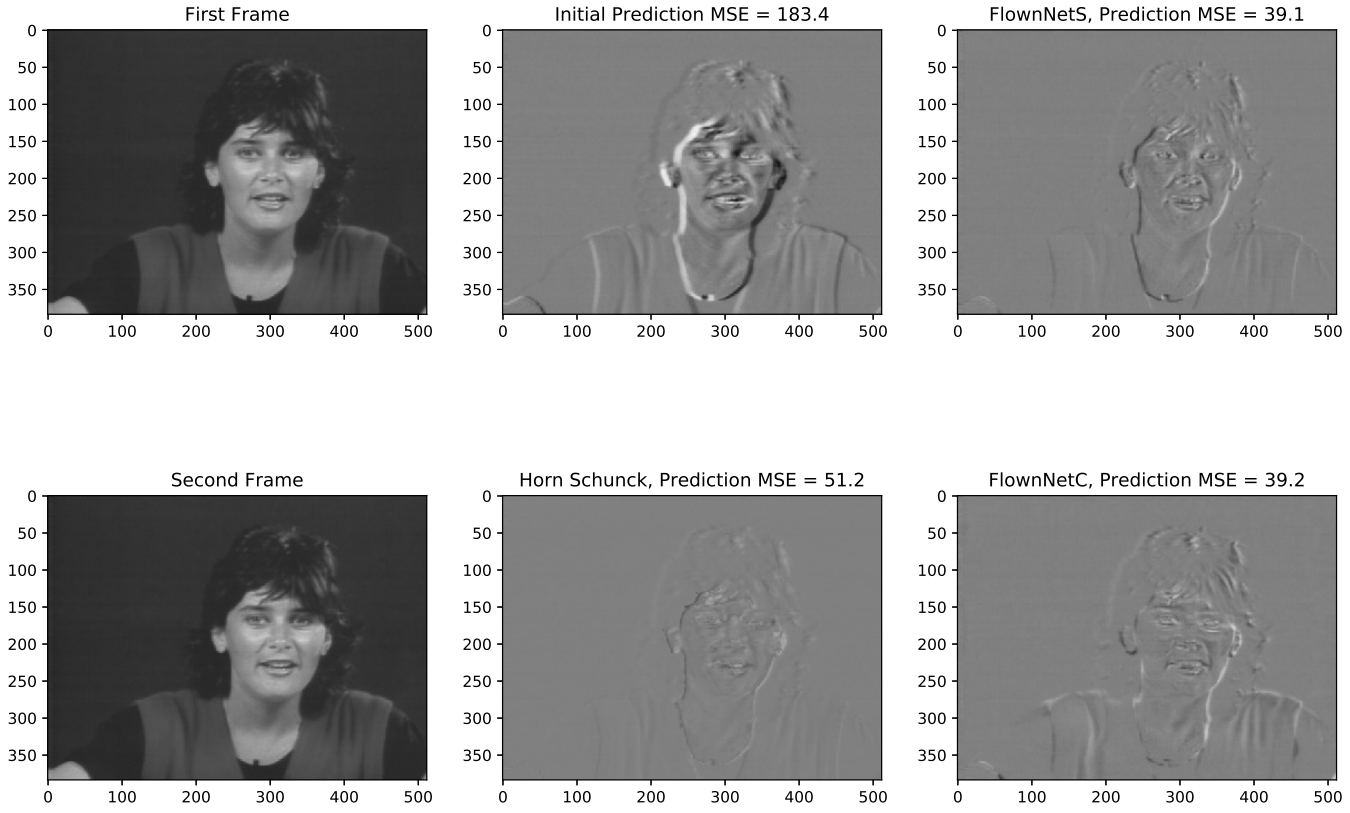

Figure 20: MCPE results for Miss America - 4 frames apart

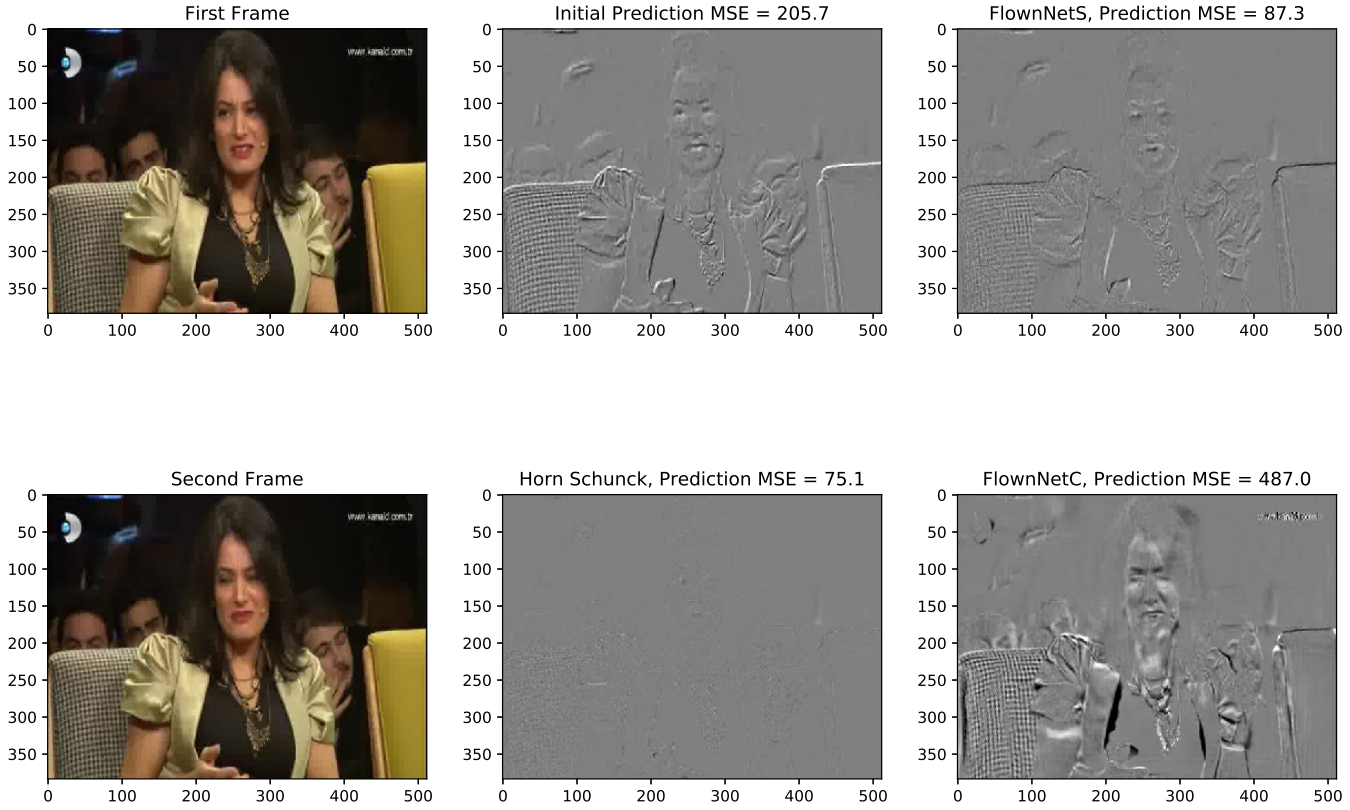

Figure 21: MCPE results for an example video - neighboring frames

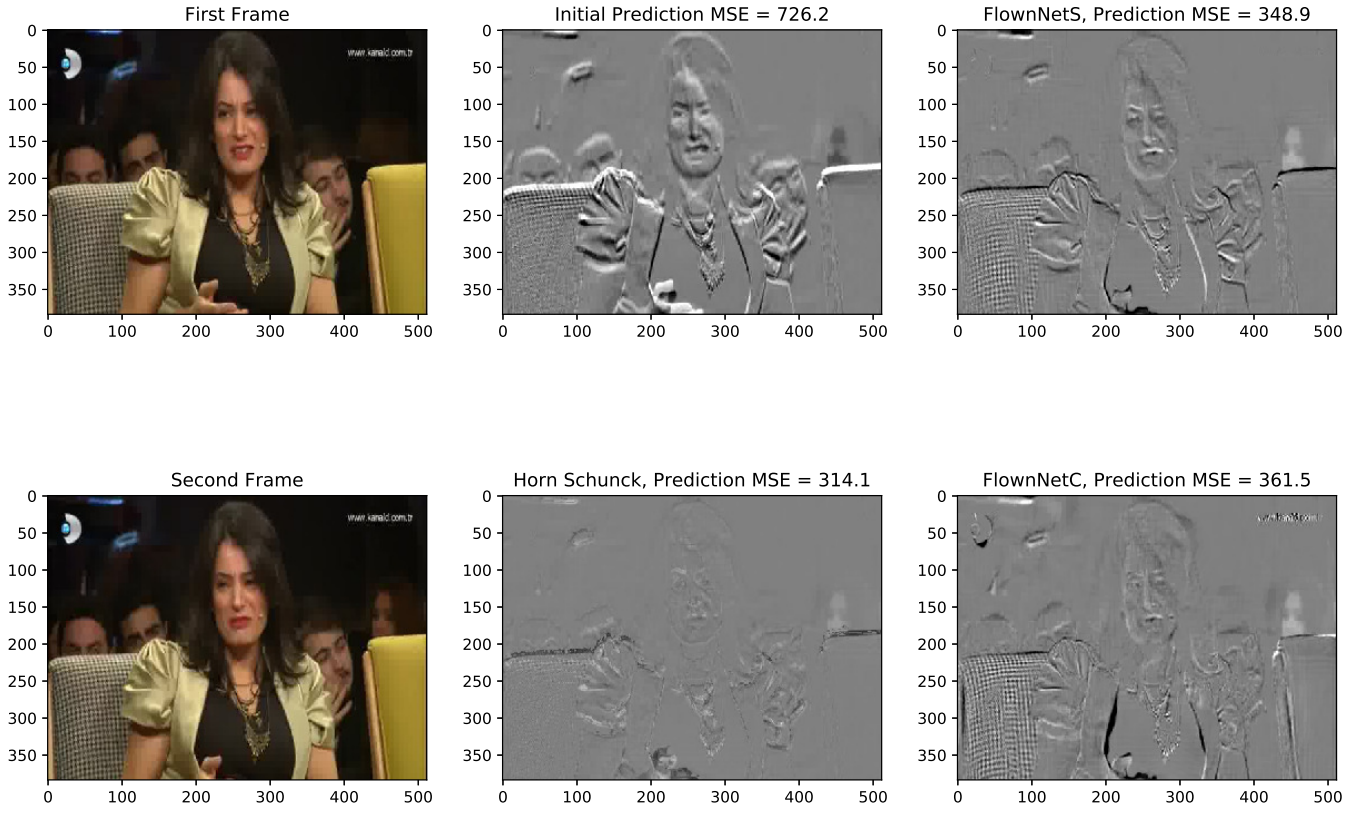

Figure 22: MCPE results for an example video - 5 frames apart

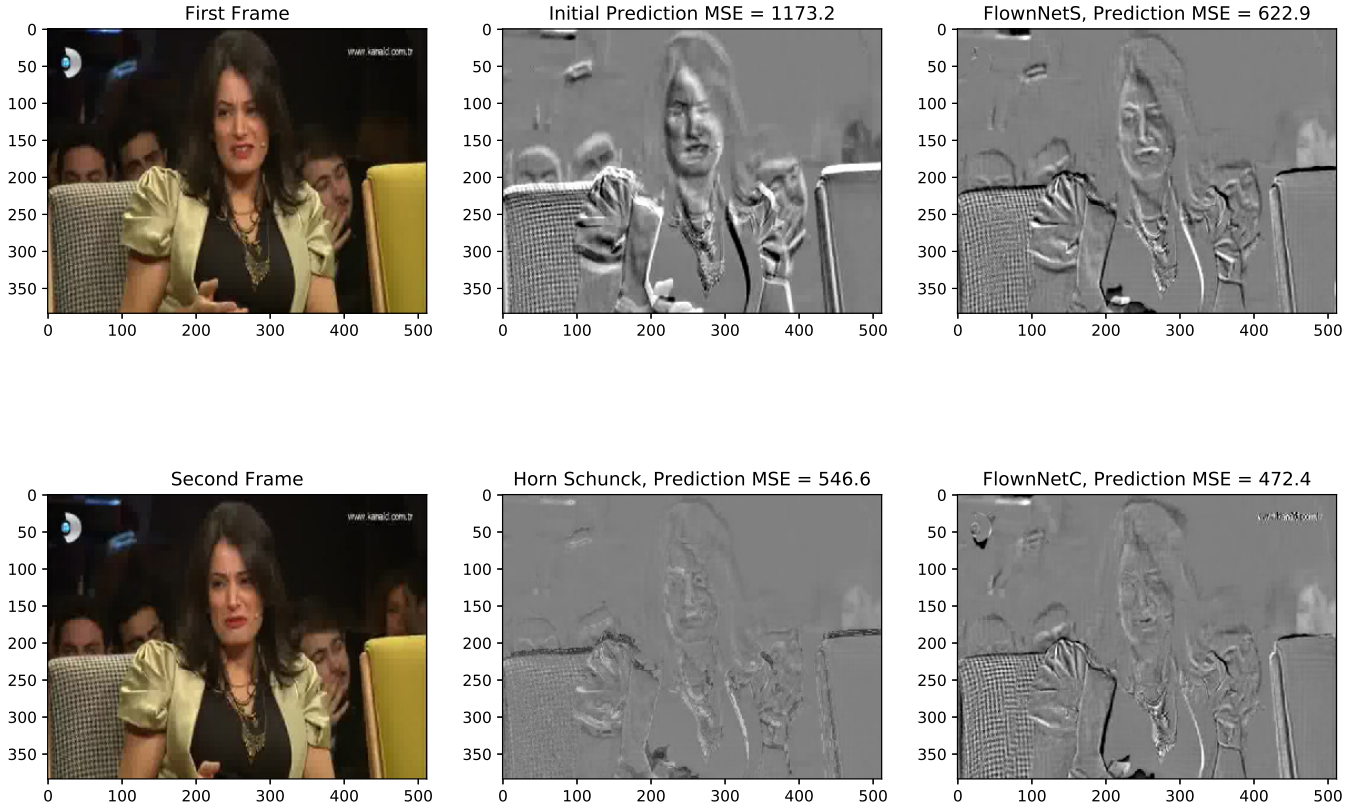

Figure 23: MCPE results for an example video - 10 frames apart

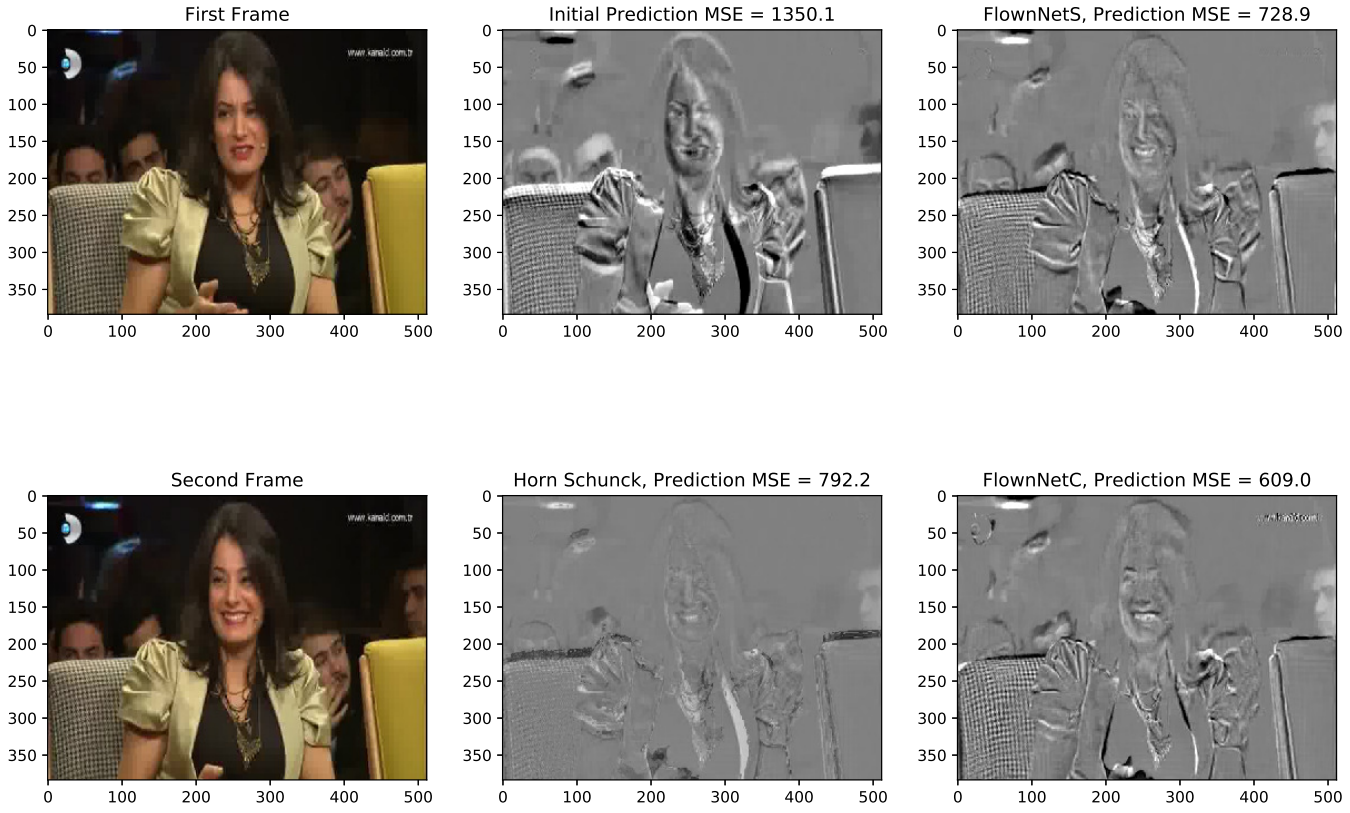

Figure 24: MCPE results for an example video - 30 frames apart

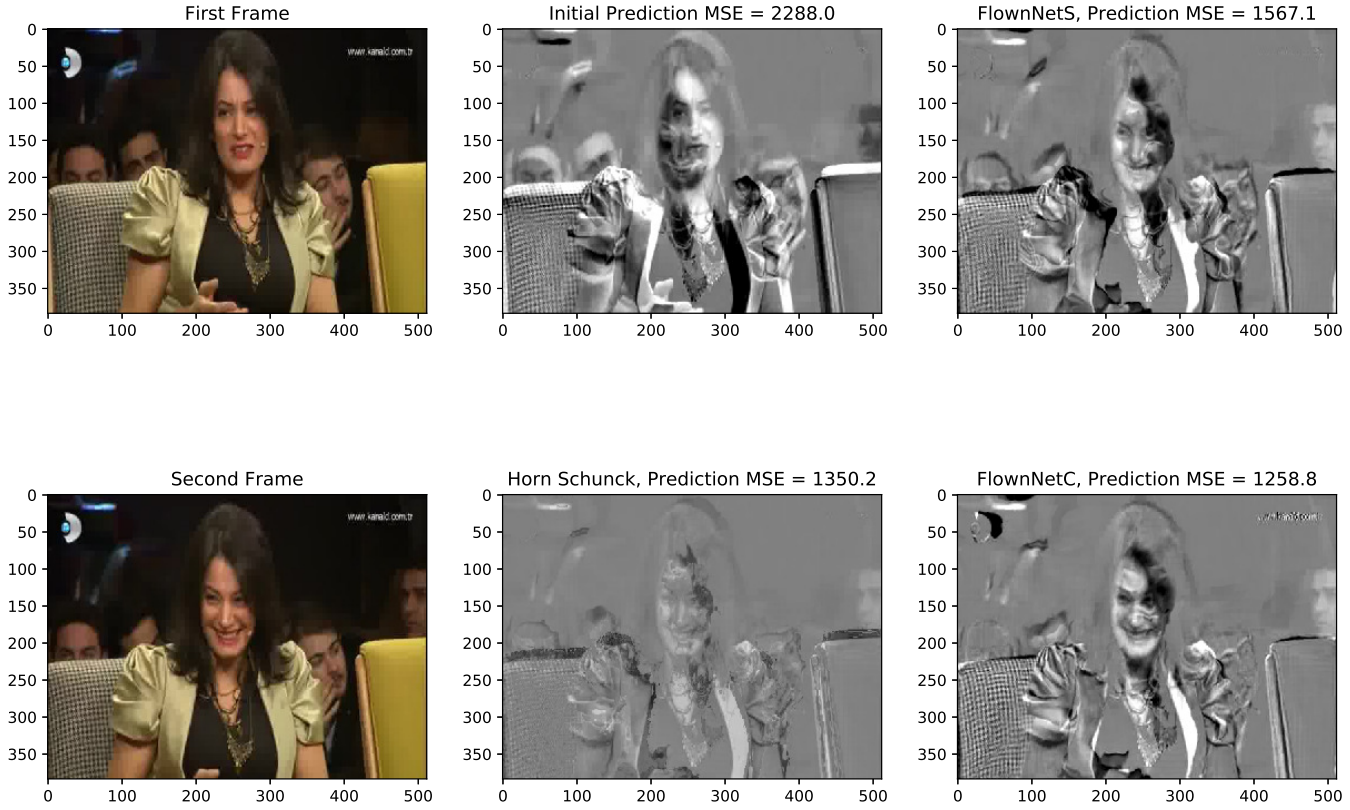

Figure 25: MCPE results for an example video - 50 frames apart

References

[1] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2758–2766, 2015.

[2] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," *arXiv preprint arXiv:1612.01925*, 2016.

[3] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," in *European Conf. on Computer Vision (ECCV)* (A. Fitzgibbon et al. (Eds.), ed.), Part IV, LNCS 7577, pp. 611–625, Springer-Verlag, Oct. 2012.

[4] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[5] A. M. Tekalp, *Digital video processing*. Prentice Hall Press, 2015.

[6] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid, "Deepflow: Large displacement optical flow with deep matching," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1385–1392, 2013.

[7] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid, "Epicflow: Edge-preserving interpolation of correspondences for optical flow," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1164–1172, 2015.