INF573 Report - Film Restoration

Auguste Crabeil and Guillaume Février

December 2023

1 Introduction

We are both cinema enthusiasts, with a strong interest for everything related to image and movies. This is the reason we took this course, because we thought it was a great way of getting a better understanding of images and the techniques to manipulate them. This project was the opportunity to tackle concrete issues faced by old movies: they are in black-and-white, often "grainy" and noisy, with a low number of frames per second. Our goal was to make them more similar to films that are produced today. Throughout the project, we use a sample video for our tests. It is an extract of *Easy Street*, a film directed by Charlie Chaplin that was released in 1917, more than 100 years ago. Using several computer vision techniques, We worked on video stabilization, denoising, frame interpolation and colorization. We present each of these topics separately as they are rather independent. In each section, we discuss our sources, the methods we implemented and their results, as well as their limits and ways that they could be improved.

2 Stabilization

To simplify our work, we decided to operate with a fixed shot. However, we believe that our work can be adapted to a video with camera movement. Despite our choice of a fixed shot, we can observe camera shake in our original video. This can pose issues for other restoration efforts, as we will explain in later sections. Therefore, our initial goal was to stabilize our video.

After researching the topic [1], we understood that it was possible to adapt the algorithms from Lab 3 (on panoramas) for stabilization purposes. Indeed, we extract key points from each frame and match them with the key points from the first frame. Subsequently, we obtain the homography and align all frames with the first one.

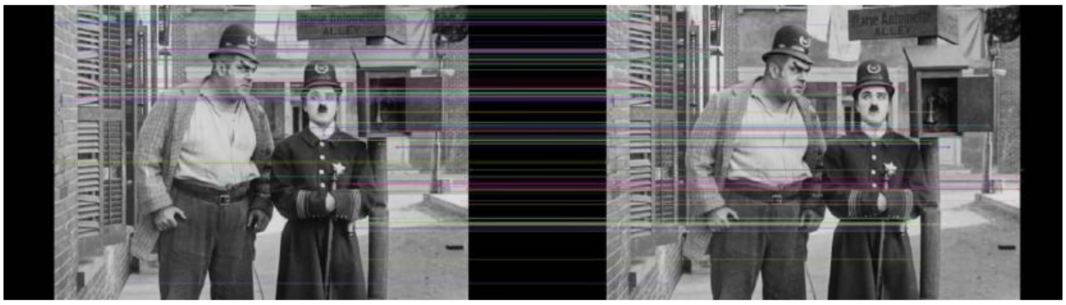


Figure 1: Best matches between frame 0 and 1

This results in a stabilized video. The difference is challenging to discern between the two videos, but improvements become apparent when displaying the difference between multiple frames. By examining the difference between the tenth frame and the first frame, we can observe that the shutters on the sides of the image have "moved". This is not the case in the stabilized video, where only the appearance of video noise between the two frames can be observed.

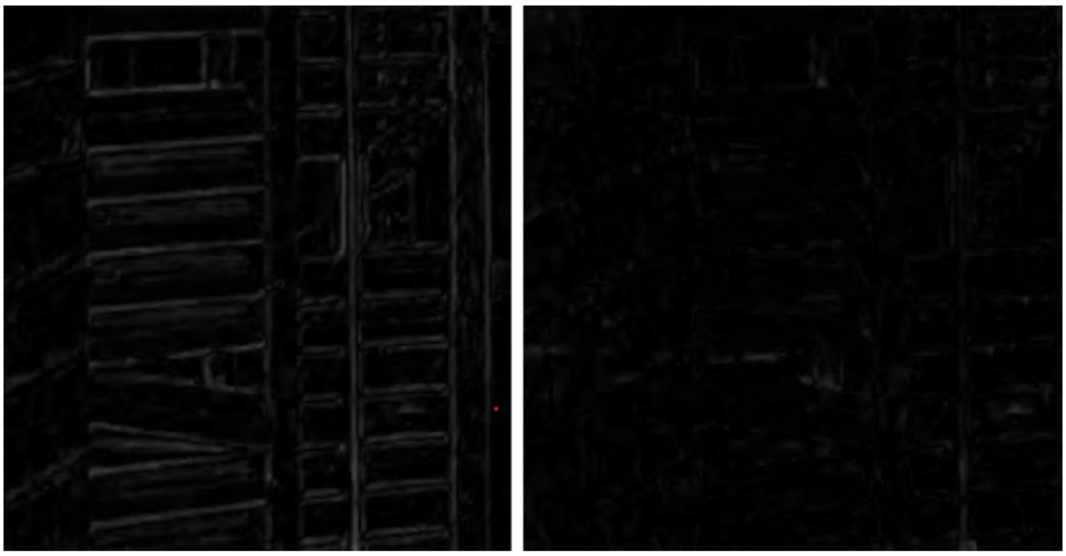


Figure 2: Difference between the tenth frame and the first frame. Left : original video we can see the shutters. Right : stabilized video without the shutters.

3 Denoising

Now that we have a stabilized video, we can focus on more substantial objectives. Here, we will present our work on reducing the visual noise in the video.

To grasp the difficulty of this task, it's crucial to understand that the visual noise in an old film is not merely the addition of typical random noise. This noise consists of white spots and vertical white bands. Some spots have diameters spanning multiple pixels, and the white bands can extend up to a hundred pixels in length. This noise present in old films is commonly referred to as "film grain". This effect is a result of the graininess of the film stock. Due to the unique nature of the noise in our video, many simple methods perform poorly.

We will now explore two different approaches to the problem, the first being relatively conventional, and the second being the subject of recent research.

3.1 Image Filtering

The first approach involves applying filters to our images. We began with a classic approach using a Gaussian blur, which is essentially the application of convolution with a Gaussian kernel. Following that, we experimented with a median blur. This operation involves taking the median value of all the pixels within the kernel area, and the central element is then replaced with this median value.

Both of these approaches yielded similar results; however, since the noise is too regular and prominent, applying large filters is necessary to partially eliminate the noise. Consequently, this leads to a blurred image.

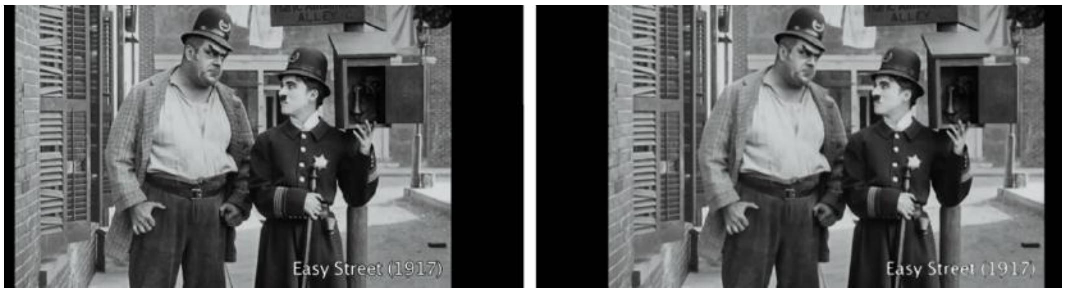


Figure 3: Left : original image, Right : median blur

To address the blurring issue, we experimented with a bilateral filter that preserves edges. The bilateral filter applies a double filter: a Gaussian filter based on spatial distance, as seen before, and another Gaussian filter based on the difference in intensity between pixels. The Gaussian function of space ensures that only nearby pixels are considered for blurring, while the Gaussian function of intensity difference ensures that only pixels with similar intensities to the central pixel are considered for blurring. Consequently, it preserves edges, as pixels at edges tend to have large intensity variations.

$$BF[I]_p = \underbrace{\frac{1}{W_p}}_{\text{Normalization Factor}} \sum_{q \epsilon S} \underbrace{G_{\sigma_s}(||p-q||)}_{\text{Space Weight}} \underbrace{G_{\sigma_r}(|I_p - I_q|)}_{\text{Range Weight}} I_q$$

Figure 4: The formula takes into account both terms, spatial and intensity.

The results of the bilateral filter are also mixed. Indeed, to partially eliminate the noise, a large filter must be applied, resulting in a very unnatural-looking image.

3.2 Convolutional Autoencoder

A simple filter is therefore insufficient. Consequently, we investigated numerous recent methods developed on the subject [2]. The idea we developed was to capture the noise to learn how to eliminate it using Deep Learning. This involves several steps in the reasoning process: attempting to capture the noise in the video, adding this noise to high-quality videos, and ultimately training a model to transform noisy videos into denoised ones. This approach aims to create a method tailored to the specific noise characteristics of our video.

For capturing the noise, we utilized the video stabilization part. Now that the video is stabilized, we can fix the background. By averaging all our frames, we obtain an image where the characters are blurred (due to their movements), while the background is of good quality and denoised (as the background is stabilized).

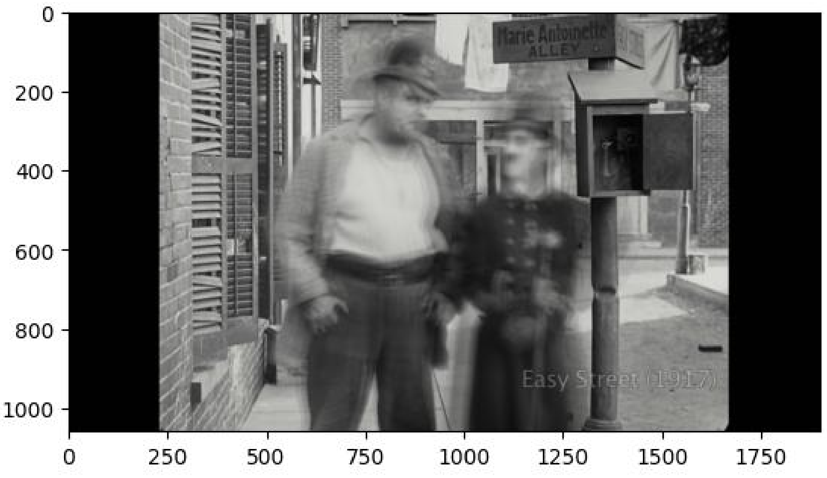


Figure 5: Average image with denoised background.

We can now take the difference between the background of our frames and the background of our averaged image. Since the video is stabilized, this difference simply represents the noise in the video, which we can then save for each frame.

Now that we have captured the noise, we can add it to any black-and-white video. This results in a video with the same style of noise as our original video.

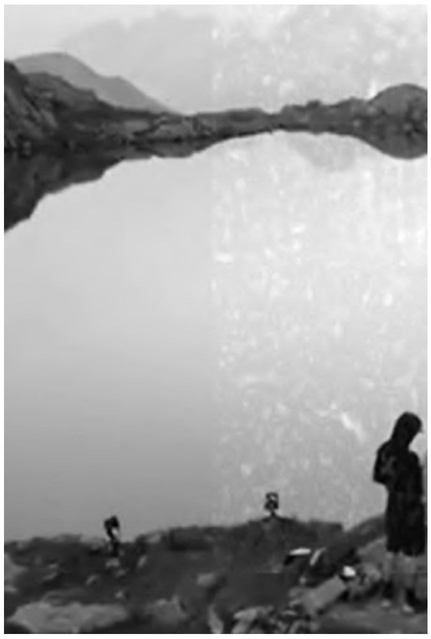


Figure 6: Left : original image. Right : added noise.

We now have the ability to generate noisy versions of any video. Consequently, we can create a training set with the input being the noisy version and the output being the original version without noise. This allows us to train a deep learning model specialized in the noise characteristics of our video. The most commonly used models for this task currently are denoising autoencoders (DAE) [3]. An autoencoder is an artificial neural network used for unsupervised learning. The goal of an autoencoder is to learn a representation (encoding) of a dataset, typically to reduce the dimension of that set and recognize the important parts of the data.

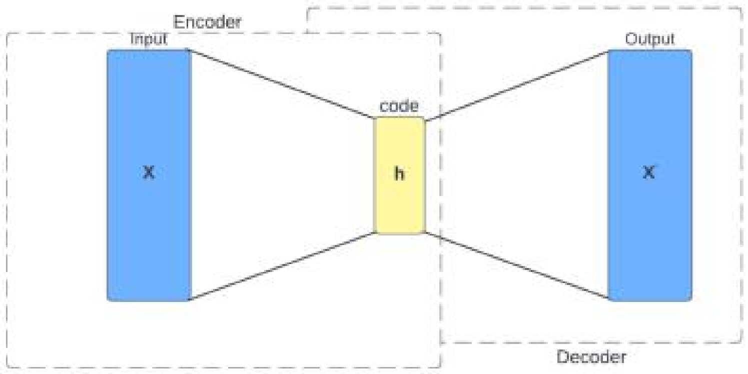
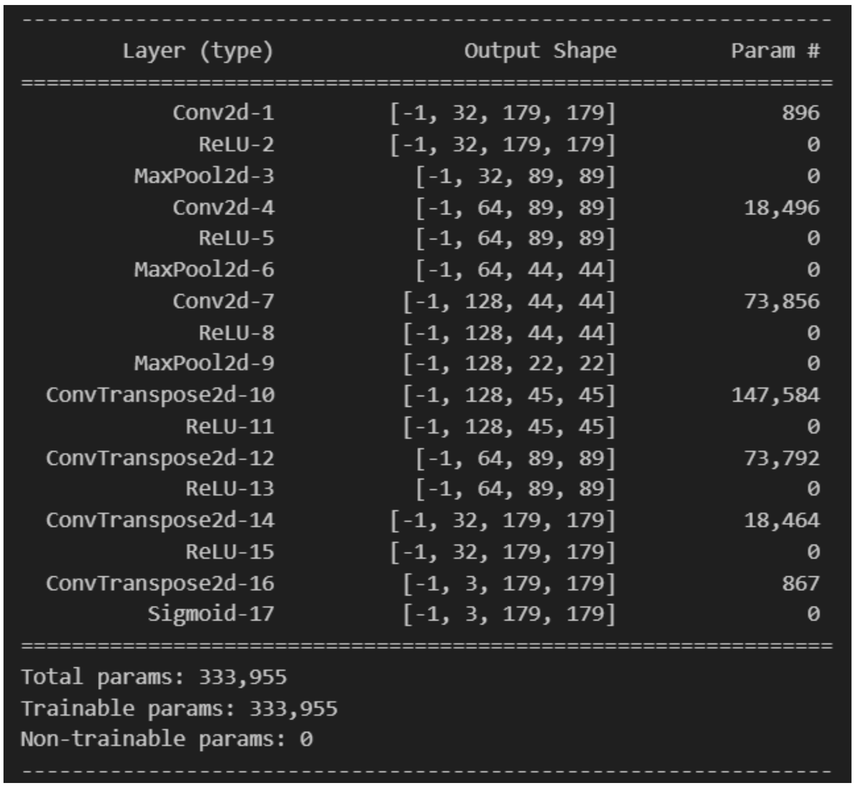


Figure 7: Autoencoder

A DAE is quite similar in architecture to a standard autoencoder, except that it introduces noise to the input images present in the dataset during training and validation. Often, the method used for a DAE involves using convolutional operations for encoding and deconvolution for decoding. The encoder thus reduces the dimension, and the decoder brings it back to the original dimension. With the reduction in dimensionality, the goal is to eliminate noise while retaining all the information in the image across numerous filters. Our model is inspired by the one presented in the article of Y.Zhang [2].



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1        [-1, 32, 179, 179]             896
              ReLU-2        [-1, 32, 179, 179]               0
         MaxPool2d-3          [-1, 32, 89, 89]               0
            Conv2d-4          [-1, 64, 89, 89]          18,496
              ReLU-5          [-1, 64, 89, 89]               0
         MaxPool2d-6          [-1, 64, 44, 44]               0
            Conv2d-7         [-1, 128, 44, 44]          73,856
              ReLU-8         [-1, 128, 44, 44]               0
         MaxPool2d-9         [-1, 128, 22, 22]               0
  ConvTranspose2d-10         [-1, 128, 45, 45]         147,584
             ReLU-11         [-1, 128, 45, 45]               0
  ConvTranspose2d-12          [-1, 64, 89, 89]          73,792
             ReLU-13          [-1, 64, 89, 89]               0
  ConvTranspose2d-14        [-1, 32, 179, 179]          18,464
             ReLU-15        [-1, 32, 179, 179]               0
  ConvTranspose2d-16         [-1, 3, 179, 179]             867
          Sigmoid-17         [-1, 3, 179, 179]               0
================================================================
Total params: 333,955
Trainable params: 333,955
Non-trainable params: 0
----------------------------------------------------------------
```

Figure 8: Description of our model

After numerous training attempts, we are significantly limited by the computing power of our computers. As a result, the outcomes are mixed, and the image still remains quite blurry.

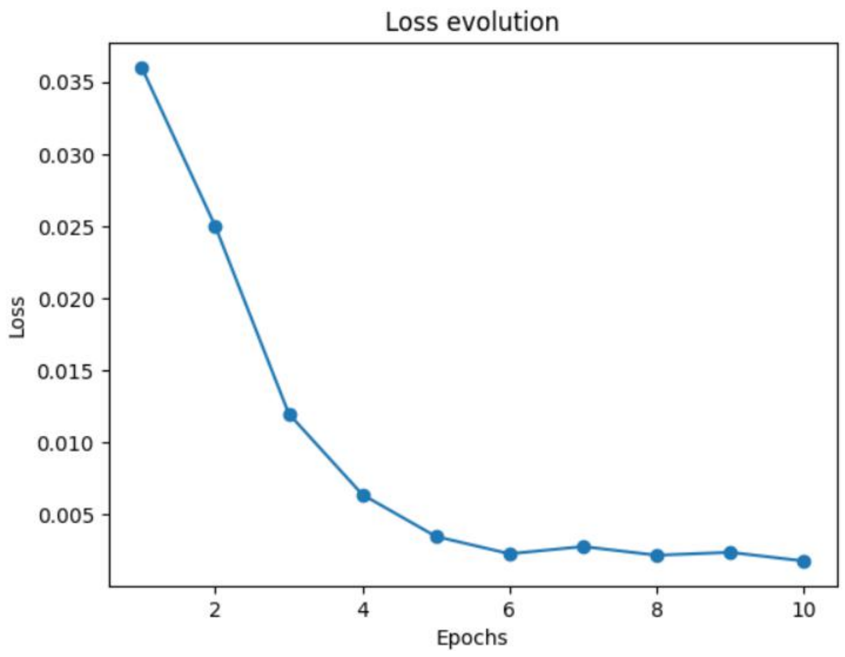


Figure 9: Loss evolution

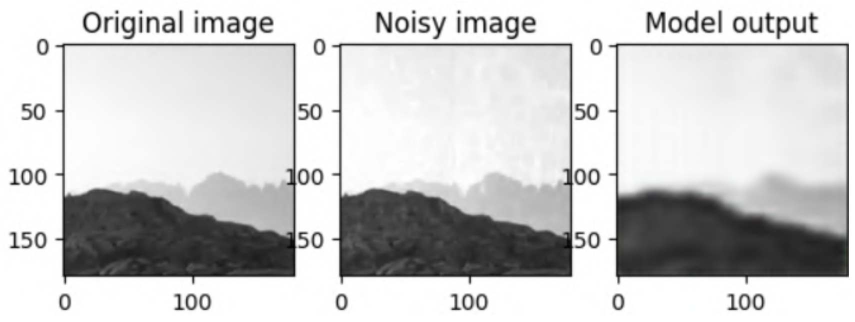


Figure 10: Result of our model

However, based on our research on the subject, we are convinced that this model could be a very good solution for our problem. We found numerous possible improvements to our model.

Firstly, we are constrained by our computers, but in practice, we can engage in data augmentation limitlessly. Any high-quality video can be noised, as explained earlier, and become a training data point. Moreover, it is possible to extend the training duration for a much longer time (increasing the number of epochs), allowing for fine-tuning of the model. We are also aware that our model remains simple, so in a subsequent phase, it may be interesting to add complexity to the model and optimize hyperparameters.

4 Frame interpolation

Frame interpolation allows people to create new images in order to increase the number of frames per second of a video. Several techniques exist to perform it. Nowadays, the most advanced ones all include neural networks. Since we wanted to implement more traditional algorithms, using what we saw in class, we decided to interpolate frames using optical flow. An optical flow gives the displacement of pixels between two consecutive frames, thus mapping the movement of the underlying objects.

4.1 Farneback Optical Flow

The Farneback optical flow algorithm, described in Gunnar Farneback's paper [4], computes a dense optical flow. It means each pixel of the first frame is mapped to another one in the second image.

$$next\_frame(y + flow(y,x)[1], x + flow(y,x)[0]) \sim prev\_image(y,x)$$

It is a quite advanced algorithm, often used for frame interpolation. The fact that it is dense is very convenient, because the estimation of the interpolated frame is easy once the flow is computed: each pixel is supposed to move according to $\frac{flow}{2}$.

As the algorithm is complex, the implementation was made with OpenCV's built-in function *cv2.calcOpticalFlowFarneback*. At first, we only computed the forward flow from frame 0 to 1 before creating the interpolated frame with the flow and frame 0. However, to gain stability, we decided to also compute the backward flow (from frame 1 to 0). Then, we blend the two interpolated frames together to get our final image. This double interpolation substantially improved our results. Our original video had 25fps, and the output is a video with 50fps.

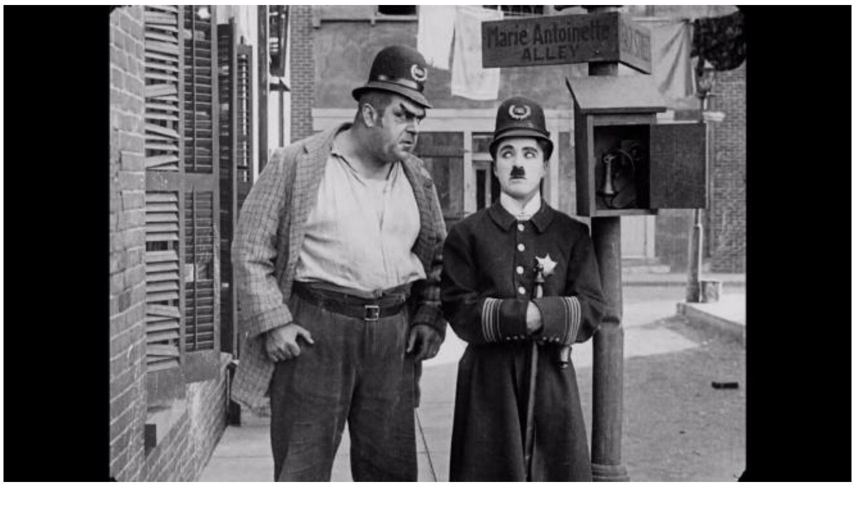


Figure 11: Frame 0

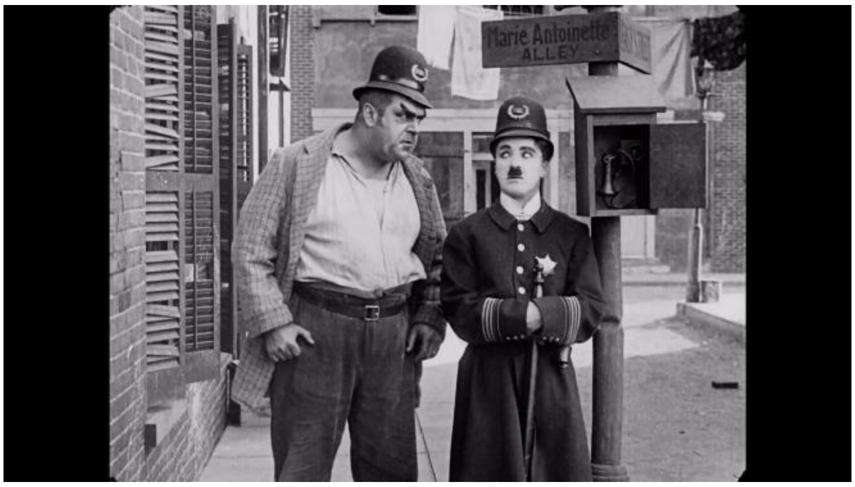


Figure 12: Interpolated frame

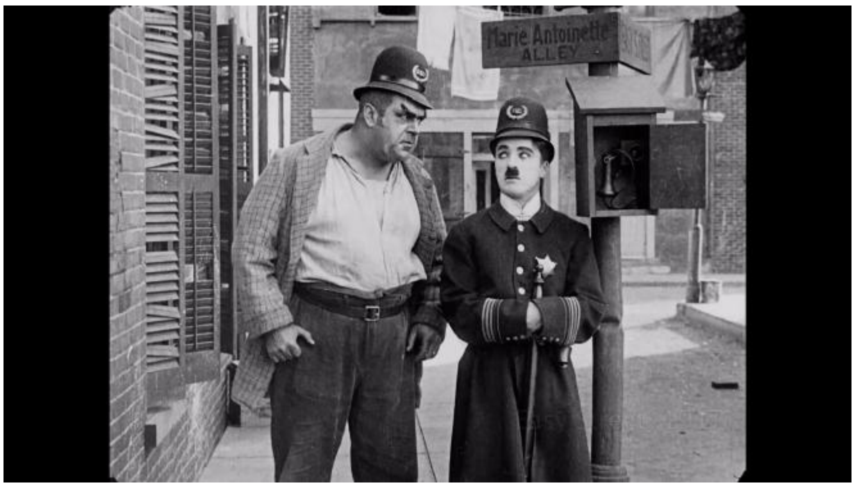


Figure 13: Frame 1

As we can see, we have good results when the movement remains relatively small (Chaplin's head for example). When the objects move over a larger distance, the interpolated frames are less accurate, as seen below with the phone.

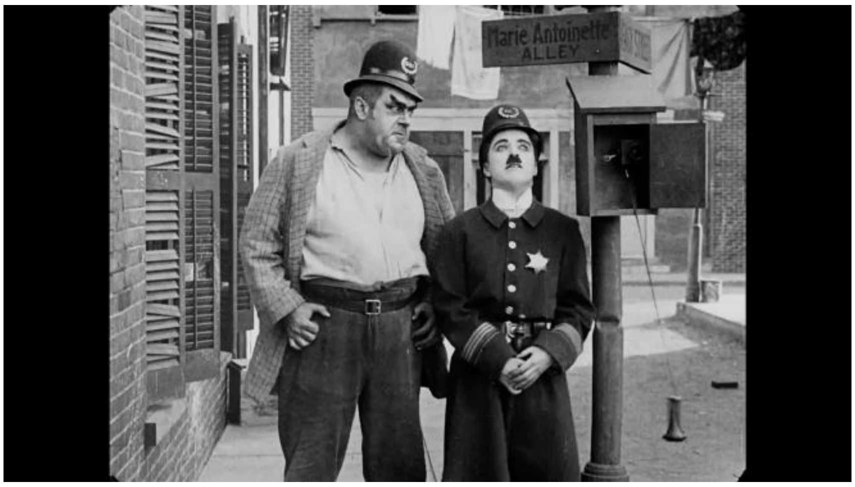


Figure 14: Frame 0

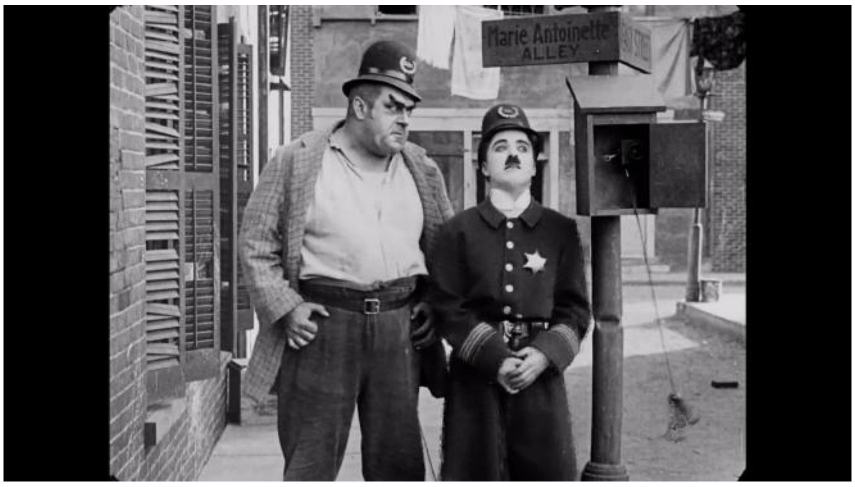


Figure 15: Interpolated frame

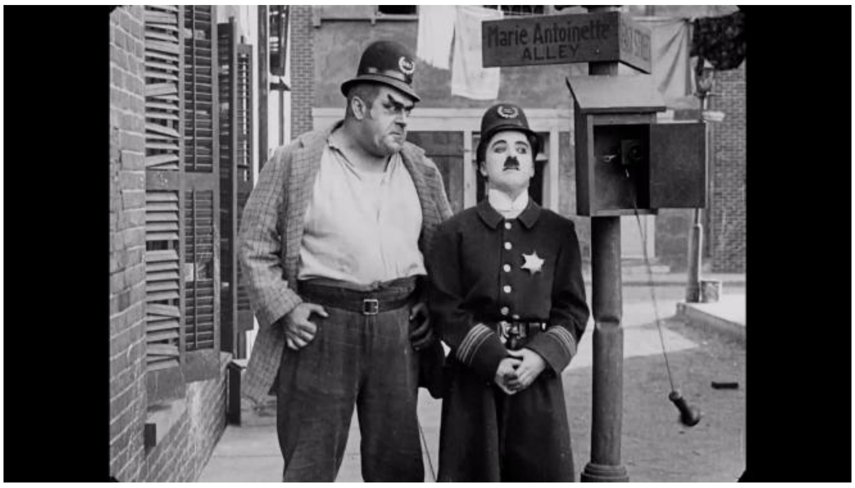


Figure 16: Frame 1

4.2 Lucas-Kanade Optical Flow

We wanted to reimplement an optical flow algorithm ourselves. We chose a simpler one: the Lucas-Kanade method. It computes a sparse optical flow, which means the flow is only computed for specific points. This is advantageous because the runtime is greatly reduced, as only a fraction of the total flow is estimated (supposedly for the most meaningful pixels). The original method is detailed in [5].

The first step is to determine key-points for which the flow will be computed. We followed [6] to implement a Shi-Tomasi corner detector. It is similar to the Harris corner detector discussed in class, except the quality of the corner is given by the minimal eigenvalue of the H matrix. We added parameters to improve corner selection and removed key-points that aren't local maxima (with respect to λ_{min}).

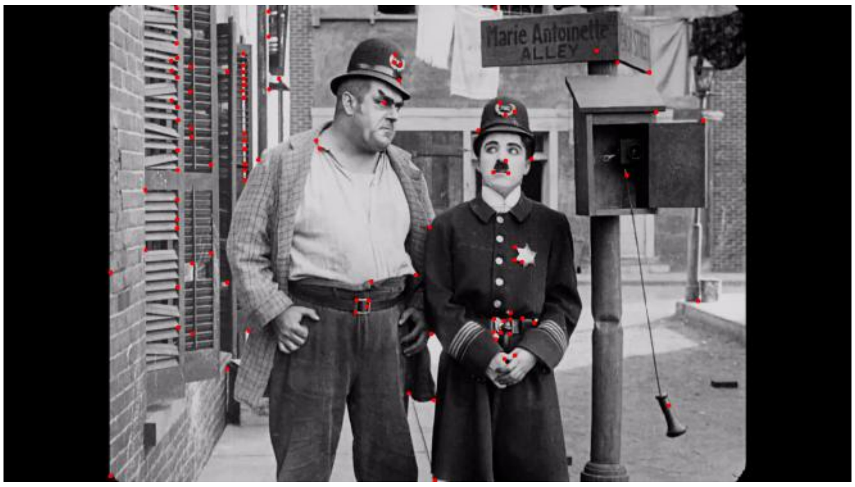


Figure 17: 100 best corners

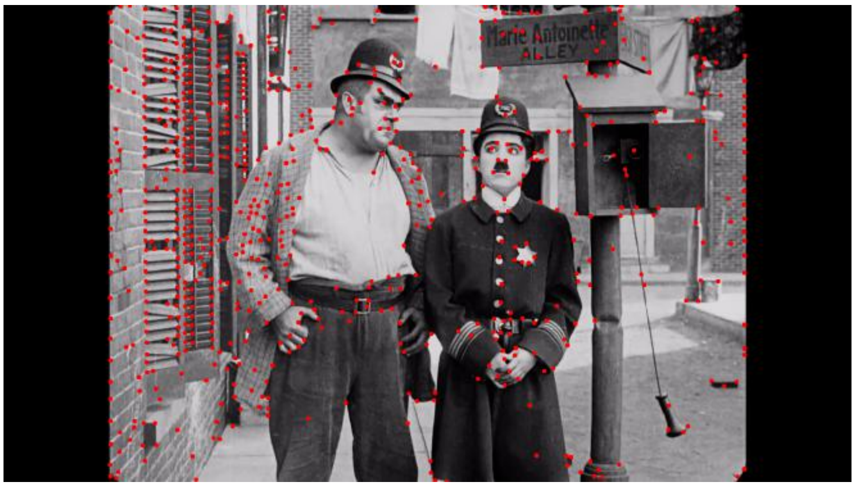


Figure 18: Corners with $\lambda_{min} > 0.05 max(\lambda_{min})$

Then, the LK method derives from the optical flow equation:

$$I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t)$$

where I is the image intensity. After simplification, it yields:

$$\partial_x I V_x + \partial_y I V_y = -\partial_t I$$

where V is the velocity vector we want to compute. We use a weighted window (with a gaussian kernel) to solve for each corner the associated least square equation, giving V:

$$HV = -\begin{bmatrix} \sum_i w_i \partial_x I(p_i) \partial_t I(p_i) \\ \sum_i w_i \partial_y I(p_i) \partial_t I(p_i) \end{bmatrix}$$

where H is the weighted matrix presented in lecture 2 and w are the weights.

In the original method, image pyramids are used to improve the optical flow, which is built iteratively. We first implemented a simplified version, without constructing any pyramids.

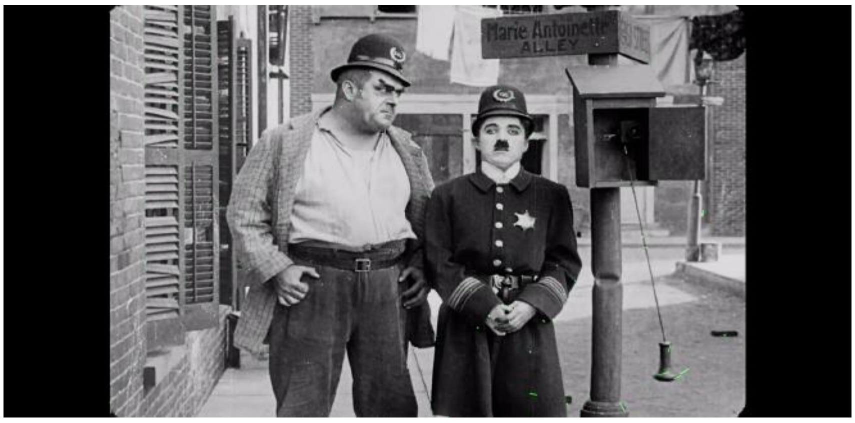


Figure 19: LK Optical Flow computed with *cv2.calcOpticalFlowPyrLK*

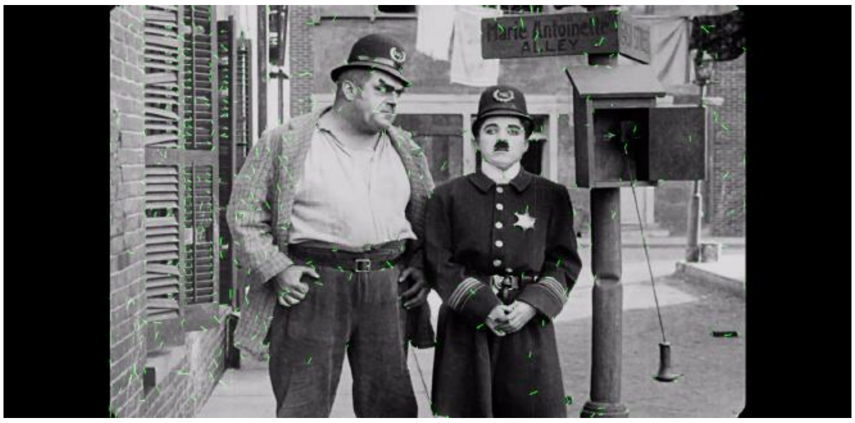


Figure 20: LK Optical Flow computed with our least square equation resolution

As the results were not satisfying, we tried to improve our algorithm by adding image pyramids, as described in [7]. However, it was quite complex and we didn't have time to finish the enhancement.

Interpolating with a sparse flow causes some issues. Indeed, we cannot simply map pixels like we did with the Farneback flow. We tried two methods. The first one uses OpenCV's *remap* function to perform a geometrical transformation of frame 0 based on the flow. The second one estimates an affine transformation between the two images with the sparse flow before warping the first image. However, it seems that the results are always extremely close to frame 0, which results in poor interpolation.

In [8], a method enabling better frame interpolation with sparse optical flow is introduced. It relies on constructing occlusion filters to determine the pixels that disappear when going from frame 0 to 1 and the other way around. Unfortunately, it is rather complex and we did not have time to implement it. This would be the next step.

4.3 Optical Flow with feature extraction and matching

As our custom optical flows lacked accuracy with the LK method, we tried another approach to get sparse optical flows. In class, feature description and matching was mentioned, so we decided to delve deeper into it. Our idea was to identify keypoints in two consecutive frames, generate descriptors for them and then match them, therefore reconstructing the associated optical flow.

To detect corners, we used our Shi-Tomasi detector presented above. Once we had our features, we used a custom SIFT algorithm to generate 128-dimensional descriptors, as described in [9]. Our version is slightly modified: as the corners are detected without building any image pyramids, it is not scale-invariant (not a huge problem in our case since objects remain at the same distance from the camera between consecutive frames). In addition, we used the Sobel operator to compute the image gradients and didn't perform the various smoothing operations described in the original paper. Other than this, our code should be rather close to the real algorithm, calculating each key-point's orientation before constructing its descriptor. The final matching is done with OpenCV's Brute Force Matcher, using L_1 norm. We compared our results to those obtained with OpenCV's SIFT implementation.

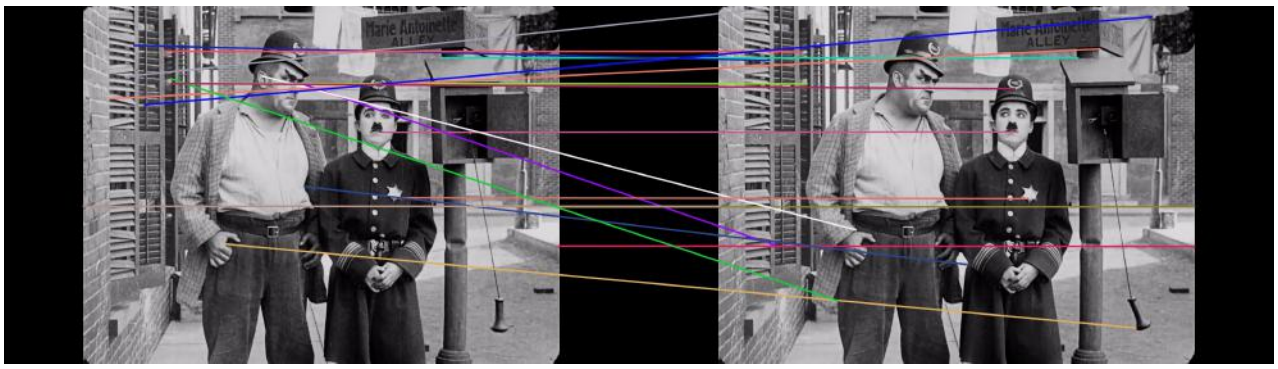


Figure 21: Key-points matching with custom corner detection and SIFT

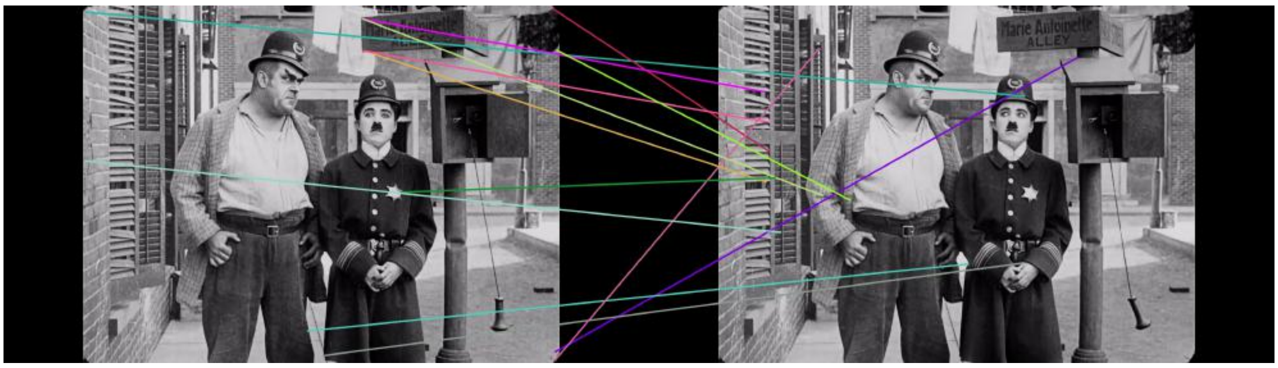


Figure 22: Key-points matching with OpenCV's SIFT implementation

Unfortunately, both perform rather poorly. It seems that this type of images represents a challenge for SIFT feature description and matching.

5 Colorization

To be able to colorize the video, we used a pre-trained neural network. It was designed by Zhang, Isola and Efros in [10]. Relying on a CCN, it is considered state-of-the-art, providing colorful images from gray-scale photographs. The network uses the LAB color space: L stands for lightness, and A (B) encodes color on the green-magenta (blue-yellow) axis. It predicts AB from L. After downloading the network, the implementation is quite straightforward when keeping the recommended parameters.

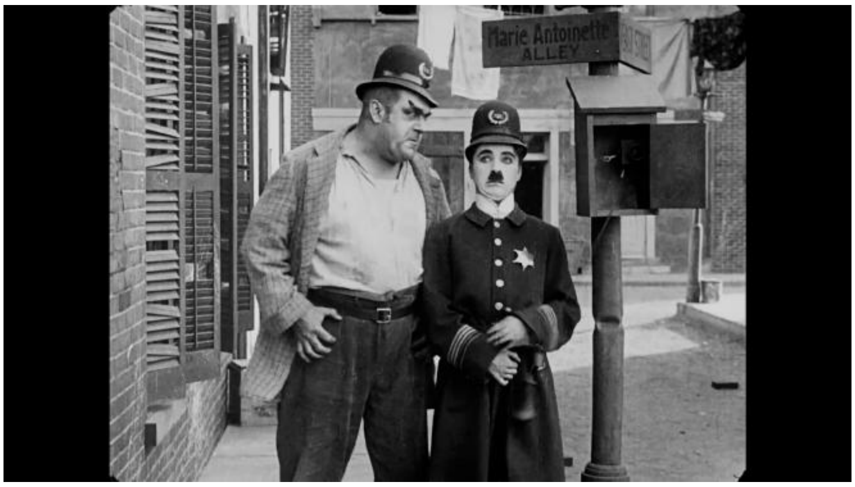


Figure 23: Original frame

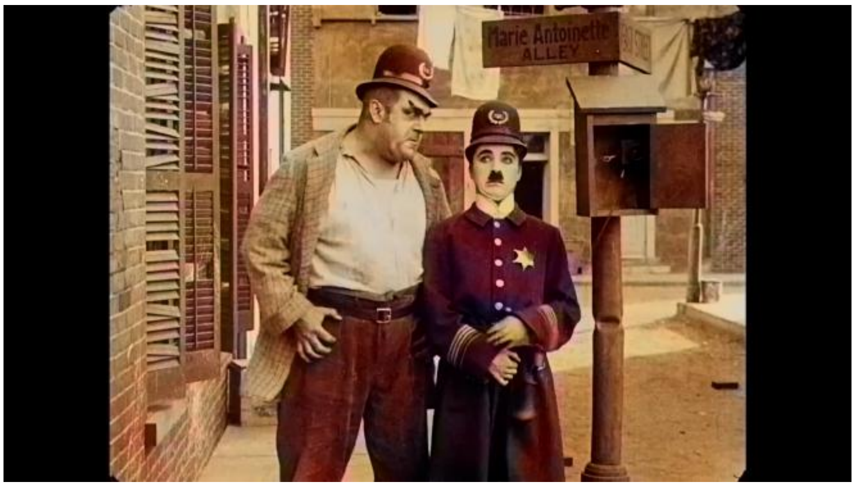


Figure 24: Colorized frame

The predicted images are quite yellowish, and we didn't manage to get the network to predict more realistic colors. We think this might be linked to the fact that the video is very old and not as sharp as a black-and-white movie shot today would be. However, it is important to note that the colors of the frames stay coherent throughout the video, which makes it rather pleasing to watch.

6 Conclusion

To conclude, we worked on various aspects of film restoration, including stabilization, denoising, frame interpolation and colorization. We obtained at least partial results for each of these. To tackle these issues, we used filtering, corner detection and matching, optical flows and neural networks. The next steps for our project would be to delve deeper into deep learning for image denoising and implement a high-performance algorithm for frame interpolation with sparse optical flows. Throughout this project, we learned a lot on computer vision, getting a better understanding of some concepts presented in the course. It was a really entertaining way to practice and develop our image analysis skills.

7 References

[1] S. Kulkarni, D. Bormane, and S. Nalbalwar, "Video stabilization using feature point matching," *Journal of Physics: Conference Series*, vol. 787, no. 1, p. 012 017, Jan. 2017. DOI: 10.1088/1742-6596/787/1/012017. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/787/1/012017.

[2] Y. Zhang, "A better autoencoder for image: Convolutional autoencoder," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:209442203.

[3] P. Venkataraman, *Image denoising using convolutional autoencoder*, 2022. arXiv: 2207.11771 [cs.CV].

[4] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," vol. 2749, Jun. 2003, pp. 363–370, ISBN: 978-3-540-40601-3. DOI: 10.1007/3-540-45103-X_50.

[5] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'81, Vancouver, BC, Canada: Morgan Kaufmann Publishers Inc., 1981, pp. 674–679.

[6] J. Shi and Tomasi, "Good features to track," in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600. DOI: 10.1109/CVPR.1994.323794.

[7] J.-Y. Bouguet, "Pyramidal implementation of the lucas kanade feature tracker," 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:9350588.

[8] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," *International Journal of Computer Vision*, vol. 92, no. 1, pp. 1–31, Mar. 2011, ISSN: 1573-1405. DOI: 10.1007/s11263-010-0390-2. [Online]. Available: https://doi.org/10.1007/s11263-010-0390-2.

[9] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

[10] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," *CoRR*, vol. abs/1603.08511, 2016. arXiv: 1603.08511. [Online]. Available: http://arxiv.org/abs/1603.08511.