ELEN644 Homework 4 Report

Justin Goh

February 13, 2023

Introduction

1 Problem 1: Learning Optical Flow with Convolutional Networks

This Problem asks us to write a summary on the paper FlowNet: Learning Optical Flow with Convolutional Networks. This summary goes over the introduction, network architecture, training data and experiments.

1.1 Introduction

The authors note that convolutional neural networks (CNN) have been widely adopted in the field of computer vision. They propose using a CNN to train an end-to-end to predict optical flow field from image pairs. This model needs to learn an image feature representation and how to match the images at different locations. The authors included a correlation layer in the CNN's architecture to enable image matching abilities. To help find the correspondence based on features, the model is able to learn strong features at different scale levels and abstraction.

It was noted that due to the lack of a large dataset, the authors generated a synthetic dataset called "Flying Chairs" dataset. It is made up of random background images where a segmented image of a chair is overlayed. Despite the use of synthetic data, the CNN was able to generalize well.

1.2 Network architecture

The network is trained to predict the x-y flow fields from images. Pooling makes training the CNN feasible and enables aggregation of information. However pooling also reduces the resolution. To overcome this the network contains an expanding part. It also contains a contracting part.

1.2.1 Contracting

The authors used 2 approaches to build the network called FlowNetSimple and FlowNetCorr. FlowNetSimple is given 2 stacked images as input which are then fed into the network. FlowNetCorr creates 2 separate processing streams for each image and later combines them. This enables the network to learn any meaningful features of the 2 images separately.

A correlation layer is used to perform multiplicative patch comparisons between feature maps. For example, to compute the correlation of 2 patches is given by:
$c(x_1, x_2) = \Sigma < f_1(x_1 + o), f_2(x_2 + o) >$
$o \in [-k, k] \times [-k, k]$

Computing $c(x_1, x_2)$ requires $x \cdot K^2$ multiplications. Computing patch comparisons requires $w^2 \cdot h^2$ multiplications. With this many calculations, it becomes unfeasible to perform a forward and backward pass. The authors overcome this by limiting the maximum displacement for comparisons and include striding. Correlations are computed for a neighborhood size of $D := 2d + 1$.

1.2.2 Expanding

Upconvolutional layers are made up of a convolution and unpooling (extending feature maps). The upconvolution is applied to feature maps and then concatenated with the corresponding feature maps from the contractive part. This preserves high-level information and fine local information. At every step, the resolution is increased by 2. This is done for a total of 4 steps.

1.2.3 Variational refinement

This is an upscaling method which begins at 4 times the downsampled resolution. Then the boundaries of the image are computed. This method is computationally expensive but can obtain smooth and subpixel-accurate flow fields.

1.3 Training Data

The following existing datasets were used:

Middlebury: Has 8 image pairs and has small displacements (¡10 pixels).
KITTI: A larger dataset with 194 trainning image pairs and large displacements. Only contains special motion type.
MPI Sintel: Contains ground truth from rendered artificial scenes and special attention to realistic properties.

Despite having existing datasets, the authors still did not have sufficient trianing data. Hence they created the Flying Chairs dataset. This was done by collecting images from Flickr and publicly available set of 3D chair models. Then an affine transformation is applied to the collected images. 2D affine transformations were randomly sampled to generate motion.

Additionally, data augmentation was performed to prevent overfitting. Techniques like translations, rotations, scaling, adding Gaussian noise, adjusting brightness, gamma, contrast and color were used.

1.4 Experiments

1.4.1 Network and Training Details

The following was employed in the network architecture:

- 9 convolutional layers with stride 2
- Models used ReLU activation function
- No fully connected layers were used
- Convolutional filter sizes decrease from 7x7 (first layer), 5x5 (next 2 layers), 3x3 (begining at the 4th layer)
- Endpoint error was used as the cost function (The Euclidean distance between predicted flow and actual flow averaged over all pixels).
- Adam optimizer $\beta 1 = 0.9$, $\beta 2 = 0.999$
- Learning starting at 0.0004 divided by 2 for every 100k iterations after the first 300k.

For FlowNetCorr, exploding gradients occured at $\lambda = 1e^{-4}$. This was resolved by starting at $\lambda = 1e^{-6}$ and then increasing the learning rate to $\lambda = 1e^{-4}$ after 10k iterations. After that the decribed learning rate schedule was followed.

The Flying Chairs dataset was split into 22,232 training and 640 test samples. Sintel was split into 908 training and 133 validation. The Sintel dataset was also fine-tuned by using a lower learning rate ($\lambda = 1e^{-6}$) for several thousand iterations. After verifying the the optimal number of iterations on the validation set, the whole training set was fined tuned.

1.4.2 Results

The authors made the following observations from their results:
FlowNetC performed better than FlowNetS on the Sintel Clean . The inverse is true for Sintel Final.

FlowNetS performed better than FlowNetC on the KITTI dataset.

On the Flying chairs dataset, FlowNetC performed better than FlowNetS and all other state-of-the-art models. This was also the only dataset where variational refinement did not improve the performance. Overall it was noted that FlowNetC had issues with larger displacements.

The authors concluded that they had demonstrated that it was possible to train a neural network to predict optical flow from a pair of input images. The images themselves need not be realistic.

2 Problem 2

2.1 Questions

Compute and plot the Lucas-Kanade optical flow vectors using the Open CV Flow Library. Study the effects of various parameters.

2.2 Analysis

Open CV enables the calculation of optical flow via the Lucas-Kanade method with the "calcOpticalFlowPyrLK()" function. The following parameters in this function was explored:

maxLevel: One of the assumptions made when calculating optical flow, is that the distance between the point in the current frame is not very far to the new location in the next frame. If there is a large motion between the points, the algorithm will not work properly. By down-scaling the images, large motion will be perceived as smaller motions.

The optical flow algorithm repeatedly down-scales an image to produce lower resolution images. The Lucas-Kanade (LK) algorithm is applied at each level starting from the highest resolution and moving down to lower resolutions. The "maxLevel" parameter sets the highest resolution level to be used in the calculation. Changing the values for "maxLevel" yielded no noticeable difference in the videos I used. This could be due to the fact that in all the videos I used, there were no rapid changes in distance between successive frames. All movement in the chosen videos were gradual.

winSize: The LK algorithm searches for the minimum change in intensity between the current frame and the next frame by using a window around each feature point. "winSize" sets the size of this window in pixels.

Generally, you would use a larger window size if the image is larger and has more indistinct features. The default value in my function is winSize = (15,15). This size was chosen as the default since it yielded optical flow patterns that were most consistent with the movement of objects in the video.

When the winSize was set to (3,3) it was observed that points were moving in an erratic manner inconsistent with the movement of objects in the video.

This could be due to the Aperture problem. This occurs when the window size (or aperture) is too small. If the window size is too small, locally the algorithm can only detect the normal flow. It is unable to determine the actual flow.

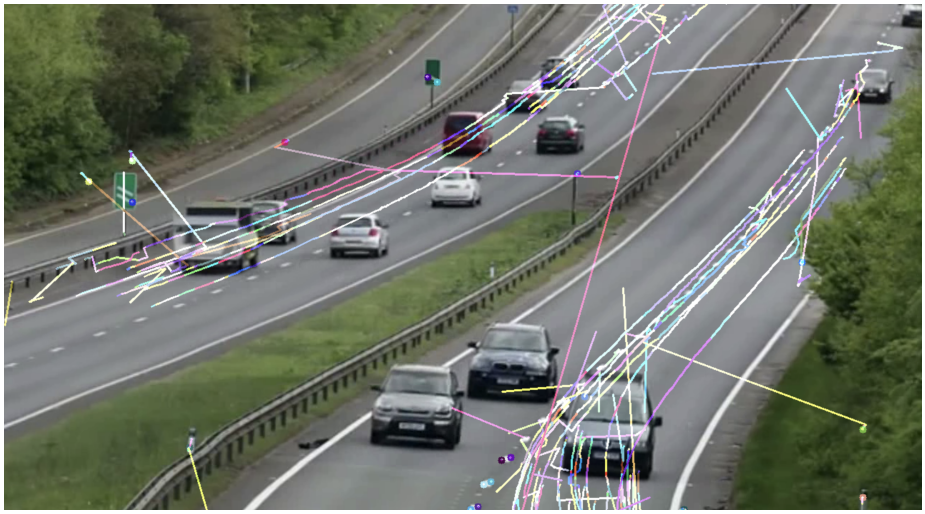

Figure 1: Erratic movement of points when winSize = (3,3)

When the window size was increased (winSize=(39,39)), it was observed that some identified points in the video were moving despite only being associated with a stationary object (or having no motion field) in the video (or having no motion field). This could be due to the fact that optical flow does not equal to the motion field at that those points.

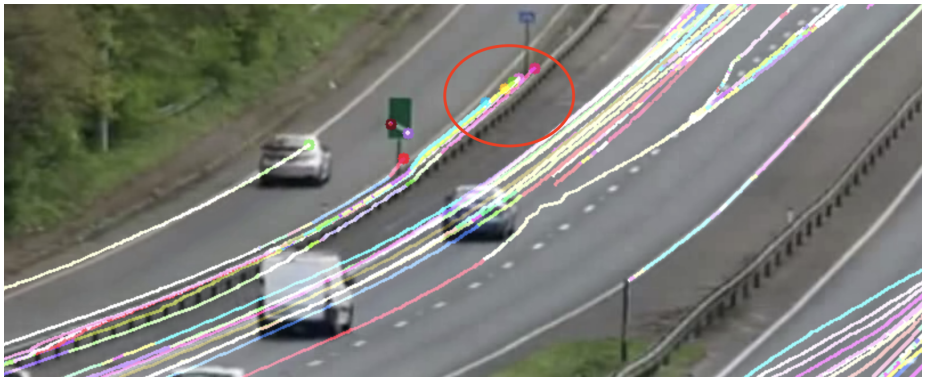

Figure 2: Points along the highway divider was observed to be moving despite being associated with a stationary object.

criteria: This parameter specifies the termination criteria for the algorithm. Once any criteria has been reached, the algorithm will stop iteratively updating the position of each feature point. The 2 criterias specified in my function are maxCount and epsilon. maxCount is the maximum number of iterations the algorithm will perform. epsilon is the minimum change in intensity needed to continue iteration. There was no noticeable differences when either of these parameters were adjusted.

3 Problem 3

3.1 Question

Pick any 10 images from CIFAR 10 data set and calculate any features. Use K-means and Mean-shift to perform image segmentation using those features.

You should plot the image segments for all the 10 images. You should explain clearly your observations. You should identify which segmentation method (K-Means or Mean-Shift) you would prefer for your 10 test images and explain why.

3.2 Analysis

Features I decided to use were RGB values. The following images were chosen from the CIFAR-10 datatset.

Figure 3: Pictures used from the CIFAR-10 dataset

K-Means: When applying the K-means algorithm, I used k values when k=3

and k=20 for comparison. The k value indicates how many clusters the algorithm will form. It was observed that increasing the value of k did not yield any noticeable difference when performing image segmentation.

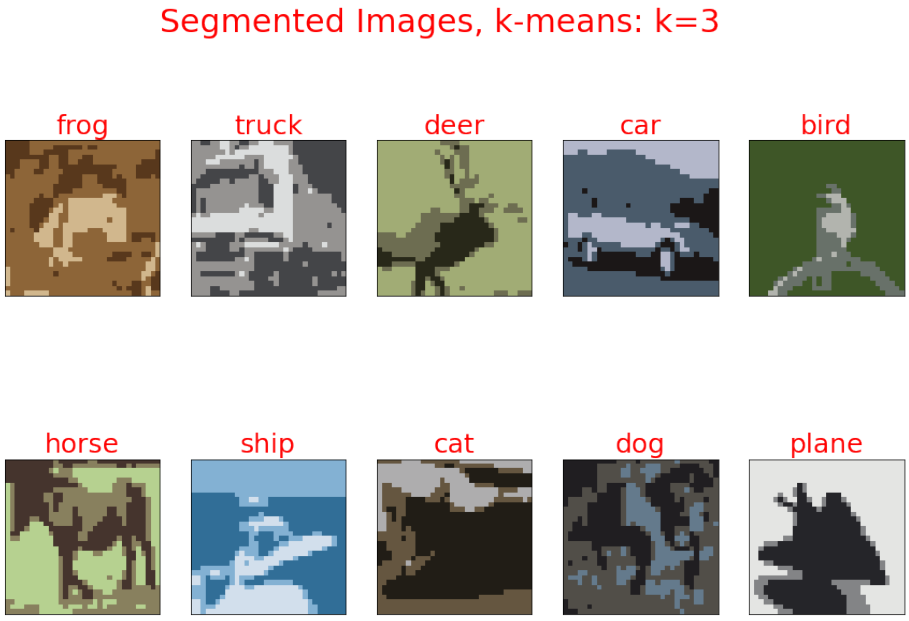

Figure 4: K-means, k=3

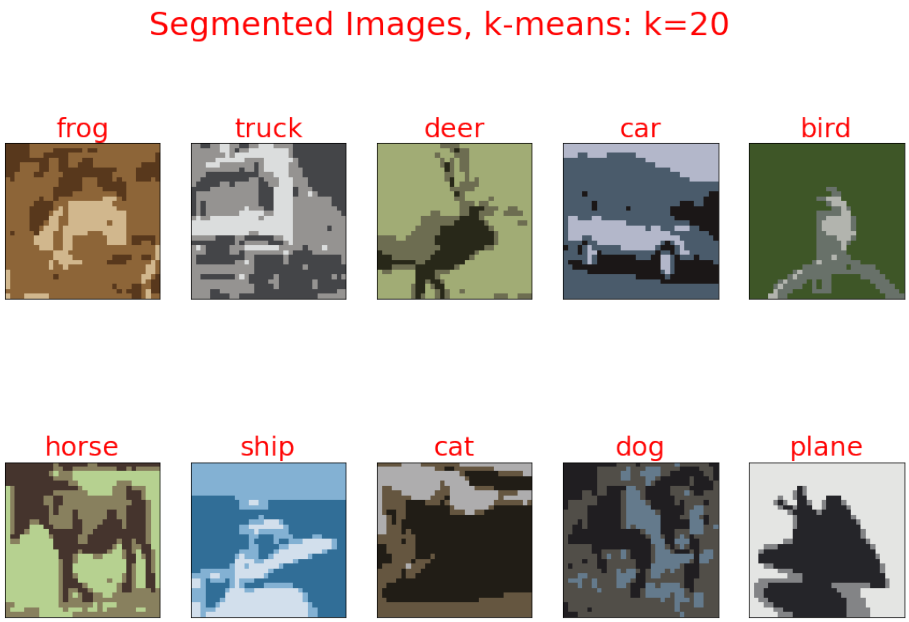

Figure 5: K-means, k=20

Mean-shift: When applying Mean-shift I adjusted the parameter bandwidth to observe how it would affect the output. This parameter is the radius of the window size used to compute the mean. It was observed that decreasing the value of the bandwidth parameter resulted in a segmented image the was nearly identical to the original output image. The values I used were 0.05 and 0.1 as seen below.

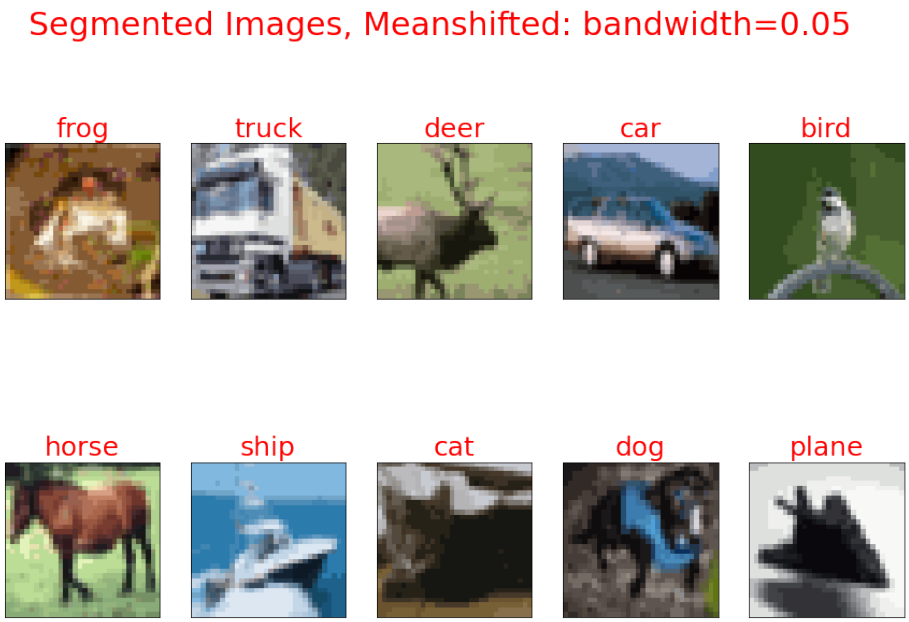

Figure 6: Mean-shift, bandwidth=0.05

Figure 7: Mean-shift, bandwidth=0.1

Conclusion: Between the 2 clustering methods analyzed, I would prefer **Mean-shift**. This is because I have more control over the level of segmentation as opposed to K-means. As I have shown above, changing the k value for K-means did not yield any meaningful change in the output image. Mean-shift on the other hand gave me greater flexibility in adjusting the degree of segmentation in the output image.