COMP1827
Introduction to Artificial Intelligence

Visual Recognition in Deep Learning: CARTOONIFY AN IMAGE

Han Ha[1]

Other group members:
Joannah Mae M Guarin Meija[2]**, Nisha Mondal**[3]**,**
Sai Sree Sravan Kumar Thota[4]

Abstract. *In an age of advanced and ever-evolving technology, it only takes a few minutes to find numerous photo editing applications on the internet, which can correct colors, change details, or even turn a real-life photo into an animated picture. However, to perform that conversion, there are many ways from simple to complex, from a few mobile operations to complicated algorithms behind each application program, which produces different results of varying quality. Therefore, this research work will represent one of the techniques using OpenCV image processing library as well as applying a chain of methods such as Canny edge detection, Gaussian filter, thresholding and bilateral method, K-means clustering algorithm, etc. to scale and transform images.*

The above is also our group's chosen topic - cartoonify an image - and our objective for this is to analyze methods, compare their parameters and derive their effects on extracted research subjects (input real-life to output cartoonist photos) and followingly, to draw our conclusion and prediction about future developments.

1 Introduction

In the past decades, the recognition and classification of an object is getting increasingly popular and becoming an emerging research field in Artificial Intelligence (AI) area, specifiedly in deep learning. Image processing applications, therefore, play a crucial role in the study of accurate edge detection and color quantization. Today's new advancement offers people several ways to convert captured images of all types (scenery, city views, people, food, plants, etc.) to cartoons using Adobe's Photoshop - Illustrator, paint.net, B612, SnapChat filter repository and many more. However, regarding this matter, the limited memory and CPU resources could not adapt to high computational capacities and low processing power for image storage and intermediate calculations [1]. Fortunately, this challenging task is resolvable with the support of a new application environment, which provides available operating tools to run on hardware with complex image processing libraries. By that, our "Cartoonify An Image" project is written in Python and runs on Google Colab with supporting libraries: Numpy and Open Source Computer Vision Library (OpenCv) and a sequence of built-in functions provided by this library as below:

- Canny edge detector: Detect clear edges and emphasize them;
- Threshold method & Gaussian filter: Flatten regions of the image, adapt RGB images to grayscale ones, then convert them to binary images;
- Median blur application: Reduce image noise;
- Bilateral filter: Sharpen edges after median blur post-effect, create more solid and homogeneous colored images;
- Bitwise operation: Merge the edge mask and processed color image into one;
- K-means technique of clustering: Used in color quantization function.

With all of the listed, the prime aim of this paper is to discuss in detail how implementation and enhancement of these operations to be made in Sect 4 and followed by presenting extensive simulation findings, review and output image quality in Sect 5. The remainder of this paper is organized as follows: Sect 1: The introduction of the project; Sect 2: Background; Sect 3: Need of project and related-context definition and finally Sect 6: Consist of conclusion and future work suggested.

2. Background and definition of the related concepts

Cartoonization: Animation (also known as cartoon) is a widely adopted art-form that can be applied on various contexts and diverse purposes nowadays. Some people tweaked their photos out of curiosity, some use them for mass advertising. On a larger scale, people use animation effects to create exhibited works of pop art or invest in filmmaking, which is a rising trend led by giant studios like Disney, Ghibli, etc. According to **[2]**, artists have many choices of approach and methods to turn real-life photography into usable cartoon products, whose process is known as image cartoonization. There is no official definition for cartoonization, however, my team suppose that it is the replication of human eyes' complex vision in a simplified, brighter and homogeneous form.

Computer vision (CV): Is a field of mentioned-above replicating computation. To support computers to understand, process and return

[1] School of Computing and Mathematical Sciences, University of Greenwich, London SE10 9LS, UK, email: gh0816w@gre.ac.uk
[2] School of Computing and Mathematical Sciences, University of Greenwich, London SE10 9LS, UK, email: jg8813r@gre.ac.uk
[3] School of Computing and Mathematical Sciences, University of Greenwich, London SE10 9LS, UK, email: nm2420z@gre.ac.uk
[4] School of Computing and Mathematical Sciences, University of Greenwich, London SE10 9LS, UK, email: st6988u@gre.ac.uk

adequate outputs, different algorithms and arithmetic calculations are performed. Furthermore, CV and Machine Learning are closely related areas. Machine learning helps enhance the vision of recognizing and tracking by offering powerful methods. CV, in particular, helps widen the spectrum of machine learning [3]. Nevertheless, a large number of problems remain unsolved due to both narrow knowledge of biological vision and complexity of vision perception [4], which is also the constraints of our project performance.

Related work (Background): A work of [5] is based on pre-trained neural networks and 3-Component loss function to train feed-forward network and create high perceptual quality cartoon-like pictures by style transfer and refine loss function over it. Despite the slow running time, the program successfully creates artistic images thanks to a neural algorithm, which uses neural representations to separate and recombine layers of arbitrary images. Based on the same method as the former work of [5] for image transformation, however, the results recorded from [6] are faster and more optimized with approximate 3 orders of magnitude in real-time. Beside loss function method, [7] decide to adjust their method by reducing swapping batch normalization with instance normalization in both training and testing time. What makes it the best approach is that they utilize and reuse those trained records for high-performance architectures training. Not only leading in real running time, this method also generates the most aesthetically appealing cartoon-style rendering images.

3. Need of project/ Proposed methods

OpenCV: The algorithms used to process images in our project is OpenCV, an open-source library written in C and compiled for many embedded platforms, most common is Python. Also, indicated by **[1]**, the library involves logical and arithmetic functions to attain instant recognition and enable it to "produce high computational efficiency with a strong focus on real-time applications."

NumPy: As OpenCV mostly focuses on processing images, after considering SciPy and Matplotlib, this project opts NumPy library for other numerical operations to change OpenCV and array structures to NumPy arrays, allowing numerous input of images. According to **[8]**, A collection of data values is used to calculate the consistency of the sum of variance in the standard deviation. The data points are similar to the mean to deviate from the low standard points of the collection, whilst the high standard deviation extends over a large variety of data points values that are conveniently spaced out. We used the standard deviation in our algorithm by taking the window size of 3x3 into account and replacing the resulting location in the centre. The standard deviation is calculated by using:

$$\sigma = \frac{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})2}}{n-1}$$

Where:
n = number of data points; x_i = Value of data; $\bar{x}$ = Mean of the data

Canny edge detector: For this cartoon-looking rendering task, clear edges play an important position in the overall result. Although experienced with Laplacian and Sobel filters to detect edges, our group did not achieve any satisfactory results until we tried out and opted for Canny Edge Detector, which significantly boosts the output records. In order to fix undesirable results from former methods, as explained by **[9]**, Canny method contains two good strategies:

1. Low pass an image (or a low-pass Gaussian filter) with derivative of Gaussian filter 2D:
$$(I * g) * h_x = I * (g * h_x)$$
Where: h_x derived from x direction; g: Gaussian
2. Apply gradient function (or apply a Gaussian filter) with effect of sigma on derivatives.

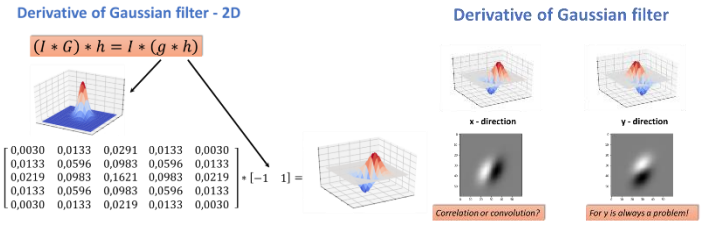


Figure 1. Canny Edge Detector by [9]

Greyscale – Threshold method – Median blur – Gaussian blur: [1] stated in their paper that an image whose each pixel value has an single sample is referred as representation of no chromatic discrepancy, also known as greyscale or BNW. After that, for more satisfying result, Threshold method is applied to convert greyscale images to binary ones. In other words, it will assign pixel value to 255 (white) if original value is above the threshold, otherwise, it will assign to 0 (black).

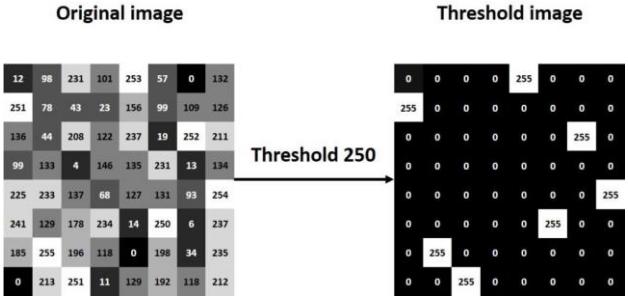


Figure 2. Threshold method by [10]

However, threshold method increased the number of unwanted observed noise from the image that is possibly removed by being blurring.

Bilateral filtering: Is a highly effective edge-preservation and noise-reduction tool **[8]**, which uses enhancement techniques to fix thick or broken edges from Canny and threshold methods **[11]**. The bilateral filter substitutes each pixel value with a weighted average of neighbor ones. However, it distinguishes with Gaussian as it calculates the variance in pixel intensities to retain edges **[10]**. The following equation can explain this further:

$$BF[I]_P = \frac{1}{W_p}\sum_{q \in S} G_{\sigma_S}(||\mathbf{p} - \mathbf{q}||)G_{\sigma_r}(I_p - I_q)I_q$$

Were:

$$W_P = \sum_{q \in S} G_{\sigma_S}(||\mathbf{p} - \mathbf{q}||)G_{\sigma_r(I_p - I_q)}$$

Where:
$\frac{1}{W_p}$: Normalized weighted averaged of neighbor p and q pixels;
σ_S, σ_r: Amount of filtering;
G_{σ_S}: Gaussian function controls distant pixels;
G_{σ_r}: Gaussian function controls pixels with different intensity value from intensity I_p.

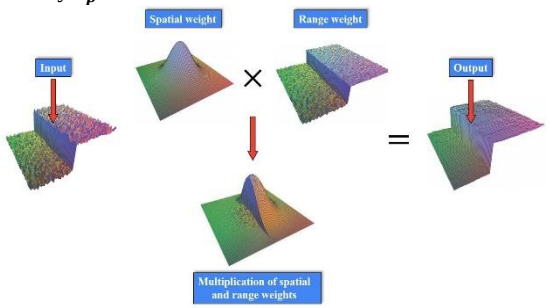


Figure 3. Bilateral filtering by [10]

K-means: The comparison provided below can prove that bilateral filtering needs to work along with k-means clustering to modify and optimize the performance and accuracy of algorithms:

Figure 4. Without (left) and with(right) K-means clusters as partition.

The mean assigned to the n data points is close to each point. Adding k-means yields k to support the initial n data points. This technique is used for dynamic clustering.

$$J(v) = \sum_{i=1}^{c}\sum_{j=1}^{c_i}(\left|\left|x_i - v_j\right|\right|)^2$$

In short: The selection of clustering algorithms, similarity computation methods and centroids only need where is appropriate knowledge of the infrastructure used and the application of the clusters **[10]**.

4. Experiments and results

4.1. Group work comparison tables and comments.

Member	Size	Color	Details	Cartoon-looking
Joannah	✗	✓	✓	✓
Han	✓ K-means	✓	✓	✓
Nisha	✗	✓	✓	✓
Sai Sree	✗	✓	✓	✓

Figure 5. assessment table of satisfaction requirements.

Color and cartoon-looking: All group members used color quantization method to minimize the numbers of color in order to make it more cartoonist with a new simplified and homogeneous range of colors.

Details: Edges in each members' result is different due to the customized parameters of OpenCV functions. Han has the most cartoonist results as using K-means clustering algorithm.

Size: Most of the members have to manually adjust the parameters to keep the length and width proportions of the inputs.

Member	Advantage	Limitation	Running time
Joannah	Easy to understand Meet the requirement	Irresponsive, require manually resize → Affect to ratio of vertical & horizontal edges No advanced/trained code	Fast
Han	Output as expected	No advanced/trained code	Fast
Nisha	Output as expected	Irresponsive, require manually resize → Affect to ratio of vertical & horizontal edges No advanced/trained code	Fast
Sai Sree	Meet the requirement	Irresponsive, require manually resize → Affect to ratio of vertical & horizontal edges No advanced/trained code	Fast

Figure 6. Assessment table of code implementation and performance

Comment: The biggest visible obstacle is the inability to preserve the proportions of the dimensions of the inputs. Also, the level of cartoon-like look is significantly affected by thin / thick / broken edges.

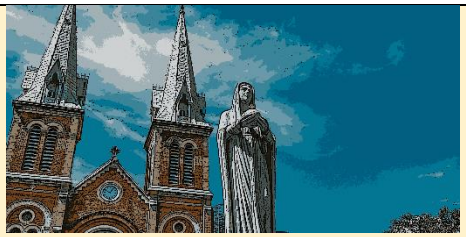
Member	Representative output
Joannah	
Han	
Nisha	
Sai Sree	

Figure 7. Table of output quality comparison.

4.2. Individual work

This section is divided into 2 parts: (A) Enhancement varies across filters applied and (B) aggregates test cases in diverse styles of captured photos.

A. Enhancement varies across each layers of filters applied

Target: Render cartoon-like images from any styles of real-world photos, which need to have at least these two primary characteristics:

- The colours in cartoon should be more homogeneous and pastel-like.
- The edges in cartoon should be sharper, more solid and noticeable.

Figure 8. Real-world photo (left) vs cartoon (right).

B. Synthesis of used cases according to multiple styles of real-life captured photographs

Description: Images are selected based on 6 different group styles, respectively natural scenery, city views, people, food, animals and plants.

Purpose: To easily compare the correlation and contrast of each representative group.

General factors: Based on three cartoon depictions from a study of [12]: surface – structure - texture, the samples of each group will include:

- Bright photos >< Dark photos;
- Close-ups photos >< Telephoto shots;
- Colorful photos >< Almost monochrome photos;
- Detailed photos >< Expansive space photos.

Used cases:

1. Natural scenery

2. City views

3. People

4. Food

5. Animals and plants

5. Code analysis and discussion

To achieve the desired outputs, I divide my program into 3 steps as follows:

1. Detect and emphasize edges
2. Apply filters
3. Cartoonify image

Step 0: Preparation

As mentioned in Sect 3, `cv2 (OpenCV)` and `NumPy` are necessary libraries, respectively to digitally processing input images and enable to loading a series of images (after each layers of filters being applied). Also, function `cv2_imshow` is needed for programming in Google Colab.

```
# Import libraries
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
```

After libraries are ready, image is followingly loaded by function `cv2.imread()` and showed by `cv2_imshow()`.

```
# Import image
img = cv2.imread("/content/saigonNotreDame.jfif")
cv2_imshow(img)
```

Step 1: Detect edges

1.1. Also mentioned in Sect 3, the Canny method has been chosen as the detector best suited between other edge detecting methods. However, it alone is not enough to obtain the desirable return. Hence, to reduce too many details being captured, its input parameters should be adjusted. In that case is 100 and 200.

```
# Detect clear edges with Canny edge detector
# Reduce unneeded details by changing Canny's input parameters (eg. 100 & 200)
edges = cv2.Canny(img, 100, 200)
cv2_imshow(edges)
```

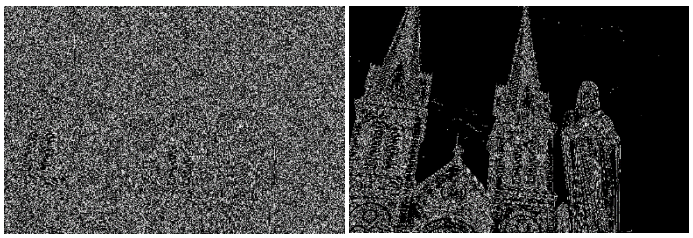


Figure 9. cv2.Canny(img, 0, 0) and cv2.Canny(img, 100, 200)

1.2. Canny already gave a satisfying results, however, better result can be achieved by transforming a greyscale image to a binary using threshold method `cv2.adaptiveThreshold()` to calculate smaller regions of the picture.

Similar to Canny's input parameters could affect to the output results, in this step, to decrease the number of observed edges and remove noise from image, median filter `cv2.medianBlur()` is applied, it only need to take to input parameters, respectively the image itself and the size of filter.

Threshold method should be applied after median filter, with its parameters as defined:

- **`max value`**: 255;
- **`cv2.ADAPTIVE_THRESH_MEAN_C`**: mean value of neighborhood area;
- **`cv2.ADAPTIVE_THRESH_GAUSSIAN_C`**: Weighted sum value of neighborhood whose weights are a Gaussian window;
- **`Block size`**: Size of neighborhood area;
- **`C`**: Constant

```
# Convert grayscale to binary image by using a threshold pixel value
# The original pixel value < thresthold value: Assigned to 0
# The original pixel value > thresthold value: Assigned to 255

# Method 1: Origin > Grayscale > Threshold
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
edges = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 9, 5)
cv2_imshow(edges)

# Method 2: Origin > Grayscale > Median fliter > Threshold
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray_1 = cv2.medianBlur(gray, 5)
edges = cv2.adaptiveThreshold(gray_1, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 9, 5)
cv2_imshow(edges)
```

Below figure is the comparison of applied and non-applied median filter:

Figure 11. Differences of non-used (left) and used (right) cv2.medianBlur().

Step 2: Apply filters

Although `cv2.medianBlur()` can be used for a solid, homogeneous and pastel-like effect, but for an optimized performance, bilateral filter `cv2.bilateralFilter()` is the most suitable one

```
# cv2.medianBlur(): Solid homogeneous but unneeded blur img
# cv2.bilateralFilter(): Smoother img with sharper edges
color = cv2.bilateralFilter(img, d=9, sigmaColor=200,sigmaSpace=200)
cv2_imshow(color)
```

There are 3 arguments in this function:

- **`d`**: Diameter of each pixel neighborhood;
- **`sigmaColor`**: Standard deviation of the filter in color space;
- **`sigmaSpace`**: Standard deviation of the filter in coordinate space.

Figure 12. Implementation of cv2.bilateralFilter().

Step 3: Cartoonify image

3.1. This final step is to combine edge mask and processed color image into a single one with the help of `cv2.bitwise_and()` function.

```
# Mix color and edges
cartoon = cv2.bitwise_and(color, color, mask=edges)
cv2_imshow(cartoon)
```

Figure 10. Implementation of cv2.bitwise_and().

3.2. Color quantization method by using K-means clustering algorithm to display output with limited color numbers. *(Picture D)*

```
# Limit the number of colors by using K-means clustering algo
def color_quantization(img, k):
# Defining input data for clustering
  data = np.float32(img).reshape((-1, 3))
# Defining criteria
  criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 20, 1.0)
# Applying cv2.kmeans function
  ret, label, center = cv2.kmeans(data, k, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
  center = np.uint8(center)
  result = center[label.flatten()]
  result = result.reshape(img.shape)
  return result

img_1 = color_quantization(img, 10)
cv2_imshow(img_1)
```

3.3. Create a more homogeneous and pastel-like coloring by applying median filter *(Picture E)*.

```
# Create more homogeneous pastel-like coloring
blurred = cv2.medianBlur(img_1, 3)
cv2_imshow(blurred)
```

3.4. Mix detected edge mask and blurred quantized image, display the final result *(Picture F)*.

```
# Combine detected edges and blurred quantized image
cartoon_1 = cv2.bitwise_and(blurred, blurred, mask=edges)
# Display the final result
cv2_imshow(cartoon_1)
```

Step 4: Comparison of all outputs.

Picture A = Original picture

Picture A = Bilateral + Blur

Picture D = Quantized

Picture C = B + Edges

Picture E = D + Blurred

Picture F = E + Edges

6. Conclusion and future work

Challenge: With the reason of lackage in experience and time constrain, this project is not use trained code or advanced program. The second issue that approached is cartoon-like impact accuracy entirely depend on the type pf image content provide, which is not yet to find a solution except manually adjust input parameters.

Conclusion: This paper has shown the process to transform a real-world photograph into a cartoon, provided examples and test cases as well as demonstrated the hardware and software needed for the conversion. Additionally, the systematic conversion of images to cartoons and the respective algorithms and formulas is shown in a neat diagram in this paper. In this article, we have addressed the need for and scope of material picture cartoons.

Future scope: With the rapid rise of technology advancement, computer vision and visual recognition has been and will be much more popular with new approach and invention support and diverse humans lives. That is my strong belief. As for me, although the project is not as excellent as originally expected, I am still very satisfied with the final results achieved. Although my abilities are not enough, this project gave me the opportunity to challenge and overcome myself, while also sparking an interest in computer vision in general and with image transformation. by AI in particular. I look forward to more development in the future.

ACKNOWLEDGEMENTS

I would like to give my first thanks to our lecturers and tutors who gave us permission to choose this topic, which is the premise giving us a chance to delve deeper into the aspects of AI and know we are suitable for this topic as well as thanks to their help and support, whether through weekly Q&A or emails. The second thank I would like to my teammates from the very beginning until now. Despite the obstacle in geographical distance and different time zones, they always fulfill their responsibilities, together complete group presentations satisfyingly and give feedback to another to improve individual essays. Third, I want to thank myself for not having given up on my studies because of the epidemic situation, thanking my family and people around me for their cheers and positive impacts on me. Lastly, thanks to this coursework, I not only gain more professional knowledge, improve the necessary skills and experience that I am lack of, but also increase self-discipline and develop a great friendship with my groupmates.

REFERENCES

[1] R. S. Deepthi and S. Sankaraiah, "IMPLEMENTATION OF MOBILE PLATFORM USING QT AND OPEN CV FOR IMAGE PROCESSING APPLICATIONS," in *IEEE Conference on Open Systems (ICOS2011)*, Langkawi, Malaysia, 2011.

[2] X. Wang and J. Yu, "Learning to Cartoonize Using White-box Cartoon Representations," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA, 2020.

[3] FullScale, "Machine Learning in Computer Vision," FullScale, 08 05 2019. [Online]. Available: https://fullscale.io/blog/machine-learning-computer-vision/#:~:text=Machine%20learning%20and%20computer%20vision%20are%20two%20fields%20that%20have,are%20used%20in%20computer%20vision.. [Accessed 09 12 2020].

[4] J. Brownlee, "A Gentle Introduction to Computer Vision," machinelearningmastery.com, 19 03 2019. [Online]. Available: https://machinelearningmastery.com/what-is-computer-vision/. [Accessed 09 12 2020].

[5] L. A. Gatys, A. S. Ecker and M. Bethge, "A Neural Algorithm of Artistic Style," *Journal of Vision*, vol. 1, no. 1, pp. 1-16, 2015.

[6] J. Johnson, A. Alahi and L. Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution," in *European Conference on Computer Vision*, Amsterdam, Netherlands, 2016.

[7] D. Ulyanov, A. Vedaldi and V. Lempitsky, "Instance Normalization: The Missing Ingredient for Fast Stylization," 2017.

[8] S. S. S. Reddy and A. Kumar, "Edge Detection and Enhancement of Color Images Based on Bilateral Filtering Method Using K-Means Clustering Algorithm," in *ICT Systems and Sustainability (Proceedings of ICT4SD 2019), Volume 1*, Warsaw, Poland, Springer, 2019, pp. 151-159.

[9] http://datahacker.rs/, "OpenCV #008 Canny Edge Detector," http://datahacker.rs/, 26 06 2019. [Online]. Available: http://datahacker.rs/opencv-canny-edge-detector/. [Accessed 09 12 2020].

[10] datahacker.rs, "#002 OpenCV projects – How to cartoonize an image with OpenCV in Python?," datahacker.rs, 13 09 2020. [Online]. Available: http://datahacker.rs/002-opencv-projects-how-to-cartoonize-an-image-with-opencv-in-python/. [Accessed 09 12 2020].

[11] C. Joshi, D. Jaiswal and A. Patil, "Technical Paper Presentation on Application of Cartoon like Effects to Actual Images," *International Journal of Innovative Science and Research Technology*, vol. 4, no. 3, pp. 451-457, 2019.

[12] X. Wang and J. Yu, "Learning to Cartoonize Using White-box Cartoon Representations," in *IEEE Conference on Computer Vision and Pattern Recognition*, Seattle, USA, 2020.