Two Novel Approaches to the Emoji-Hunt Computer Vision Challenge: A Deep Dive Analysis of Colour, Culture, and Transformations

Finn Morin
Software Engineering
University of Victoria
Victoria, Canada
finnmorin@gmail.com

Owen Jaques
Software Engineering
University of Victoria
Victoria, Canada
owjaques@gmail.com

***Abstract*—We develop two novel approaches to the emoji-hunt problem. The first approach is based on SLIC, and the second is based on a convolutional neural network. Both share a common post-processing pipeline. Using the second approach we see a final score of 25, making no mistakes in ~80% of cases. We analyze the results from both methods and propose improvements for remaining outlier cases.**

Keywords—emoji-hunt, image processing, computer vision, machine learning, CNN, segmentation, SLIC

I. Introduction

The Emoji-Hunt Computer Vision Challenge is a computer vision challenge (really) wherein an emoji is hidden in a 512x512 image a random number of times (between 1 and 10) with random transformations applied. These can vary from simple brightness adjustment to coarse dropout rendering the emoji almost invisible. For example, there are 10 in Fig. 1. Can you find them all? The challenge is to write a Python program using whatever means available that takes a target emoji and the search image as input, and outputs the centroids of each emoji.

A. Scoring

The score for the prediction algorithm is defined by the mean of the euclidean distances between the predicted centroids and the true centroids. In the case of overprediction or underprediction there is a mismatch between the points, and the distance for these elements is set to the maximum of 512. The official test for the program takes the mean score across 50 runs.

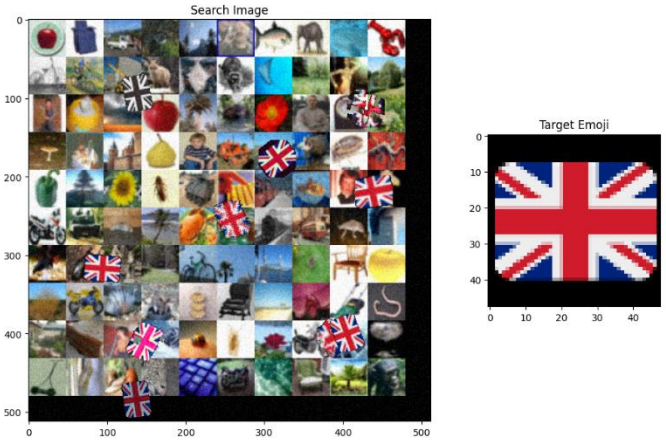

Fig. 1. Emoji-Hunt sample input. An example of the image that the emoji-hunt challenge gives the algorithm to look for emojis in.

The best possible score for an image is 0 and the worst is 512. A purely random predictor scores an average of 290 across 50 runs which we use to check if our new ideas are somehow worse than random chance.

II. Implemented Approach

A. Classical Approach: Colour Matching

1) Motivation: As suggested by the teaching team, we started with designing an algorithm that used only ‘classical’ (ie: not deep learning), computer vision methods. Based on what we had learned in class we implemented an algorithm that would look for clusters of pixels in the search image with similar colors to that in the emoji, then output a binary segmentation mask, from which the locations of the emojis could then be determined.

2) Algorithm: First, both the image and the target emoji are segmented into superpixels using SLIC (Fig. 2), a popular superpixel algorithm that creates superpixels based on the five-dimensional color and image plane space [1]. What paticulary appealed to us about SLIC was that one of its hyperparameters allows the user to configure its distance measure based on whether the algorithm should emphasize spacial proximity or color proximity [1]. Its also worth noting that the image of the target emoji is only segmented on the emoji itself, the background of the emoji’s image is not taken into account.

Next, both the image and the target emoji are converted to the LAB color space, considered to be perceptually uniform [1], which is an important charateristic for the next steps in our algorithm. The algorithm then computes the average color of each superpixel in both the image and the target emoji. A new

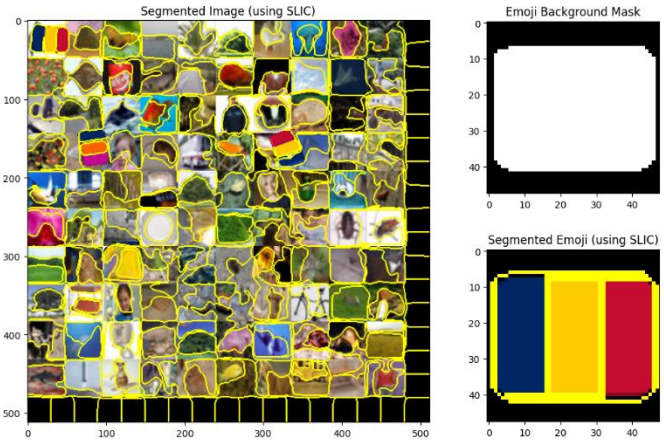

Fig. 2. Segmented image and target emoji. On the left is the original image given by emoji-hunt with the borders of the segments outlined. The top right is the binary segmentation mask which is given to SLIC to tell it where to compute the segmentation of the emoji. On the bottom right is the segmented emoji.

image is then created with the color of each pixel set to the average color of its corresponding superpixel (Fig. 3).

After the creation of this new image, for each of its pixels similarity scores are computed for each of the average colors in the target emoji's superpixels. The similarity scores are calculated using the euclidean LAB distance (1). A binary matrix representing the similarity scores is then computed by thresholding the similarity scores (1 will represent a match while 0 will represent no match). The value on which to threshold is further discused in section 3.6. This matrix is three-dimensional, image width (x) by image height (y) by number of superpixels in the target emoji (c).

$$d_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (a - a_i)^2} \qquad (1)$$

Finally, a sliding window of size 32x32 slides over the binary matrix along the x and y axes with a stride of 4. The window is then summed along the c axis. If 80% of the pixels in the summed window are greater than zero, all the pixels in the target image which correspond to the pixels greater than zero in the summed window are marked. This will then output a binary segmentation mask (Fig. 3), which with some preprocessing described in section 2.C, can be translated to the x-y coordinates of the emojis.

B. *"Modern" Approach: Throw a CNN at it*

1) Motivation: While the classical method was in development we were searching for something that would be more robust to the transformations that significantly alter shape and colour (the main downfall of our classical approach to be discussed). Having very little experience with neural networks, we decided that would be the best avenue to explore. In the worst case we could always fall back to the classical method and write a report going into lengthy detail about how terrible it is.

2) Network Architecture: Initially we tried some in-house architectures based on VGG-16's idea of Conv→Conv→Pool→Repeat *[2]* with 20 output neurons to predict the locations of each centroid (2 outputs per prediction). Training seemed promising at first but ultimately all these attempts failed to converge on anything useful and were scrapped. What did work though was implementing U-Net *[3]*.

We chose U-Net for a few reasons, first and foremost we saw the challenge as a segmentation problem, and U-Net has been shown to perform well on many segmentation tasks beyond medical image processing [4]. It is also relatively simple to implement in our ML library of choice, Keras [5] and many examples exist online.

As seen in Fig. 4, U-Net is a CNN with two main parts: a contracting path, which captures the context of the input image, and an expanding path, which enables precise localization. During the contracting path, the image's spatial resolution is gradually reduced while increasing the number of feature maps. In contrast, the expanding path upsamples the feature maps and restores the original image size. To help preserve spatial information there are also skip connections that directly connect the corresponding contracting and expanding paths. For a more

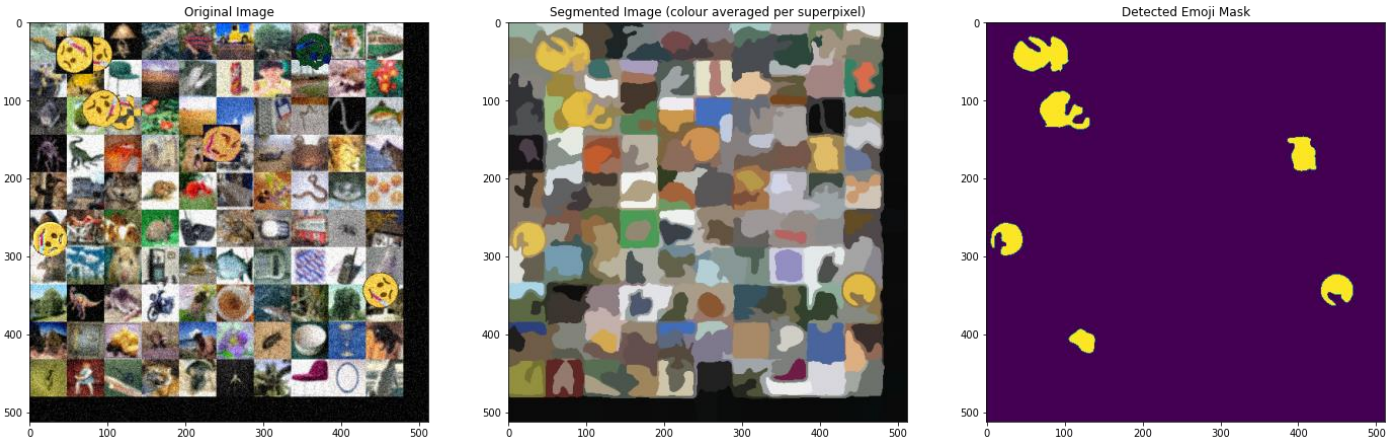

Fig. 3. Classical Approach Pipeline. The first image shown is the original image given by the emoji-hunt challenge. The second is the image once it has been segmented by SLIC and the colors of each super pixel have been averaged. The third image is the segmentation mask that the algorithm outputs.

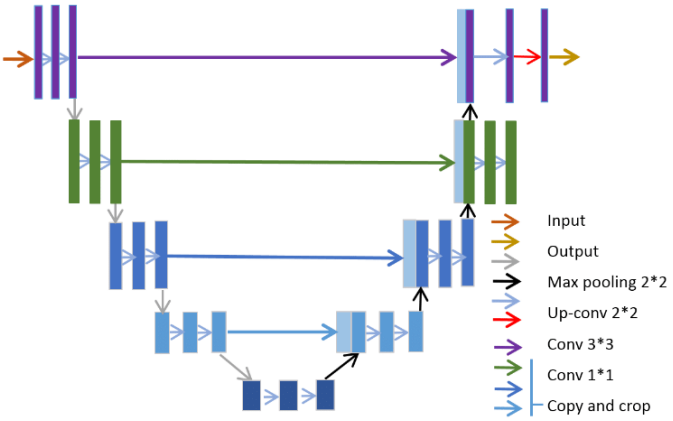

Fig. 4. U-Net Network Architecture [6]

complete explanation please see the original paper. Note we use the original, unaltered architecture as well as the original loss function.

3) Training: To train the model we create a Keras "Generator" object which lets us pass the network a new, unique sample for each step. Originally we had anticipated needing to generate training and validation sets in advance and then dump them to disk but they generate fast enough on the fly to avoid this (0.2s/image). Since the network will also (theoretically) never see the same sample twice the chances of overfitting are extremely slim. This is why we chose to not use a validation set. In retrospect this probably wasn't the best choice since the network could still overfit to the training set however, we did not notice this in practice. We did employ a test set though (we're not complete animals); the official scoring function and some rigorous visual inspections were used to compare between models and gauge performance.

With a batch size of 8 (the maximum on our RTX 3080 w/ 10GB of VRAM), 150 batches per epoch, a learning rate of 10^{-5}, and the usual training callbacks of ReduceLrOnPlateau and EarlyStopping. The loss (sparse categorical crossentropy), accuracy, and intersection over union (IoU) stopped improving after ~45 epochs when the network had fully converged. Over training the network saw ~55,000 images.

C. Post-processing & Prediction

Since both of our approaches output a binary segmentation, some post-processing is required to get the coordinates of the emojis from them. First, each connected region in the binary segmentation mask is labelled. Using this labelled mask, the centroid, area, and perimeter of each labelled region is computed using scikit-image's regionprops module [7]. Naively, the centroids of each region could represent the detected emojis, however, in reality there are many other factors which must be considered.

First, regions which represent noise must be removed. To implement this, regions with an area less than 45 pixels are simply removed from the list. We found this worked very well as any noise in the masks was quite small. Next, groups of regions which all belong to a single emoji must be merged into one region (Fig. 5). To implement this, regions which are less than 20 pixels away from other regions are merged together into one region.

Large regions with non-uniform shapes are then considered. A region, r, is considered to be large if (2) or (3) is satisfied, where a represents an area and p represents a perimeter. A region is considered to be a uniform shape if it is approximately a square, which is determined based on whether (4) is satisfied.

$$a_r > mean(a_{regions}) + 1.5 \times std(a_{regions}) \quad (2)$$

$$p_r > median(p_{regions}) + 1.5 \times std(p_{regions}) \quad (3)$$

$$0.9\frac{p_{region}}{4} < \sqrt{a_{region}} < 1.1\frac{p_{region}}{4} \quad (4)$$

These large, non-uniform regions are then assumed to be two emojis and are removed from the list of regions. To determine the two new centroids of these emojis, the k-means clustering algorithm is applied on the region's pixels (k=2, initialized using k-means++). K-means was chosen as it allows the user to configure the amount of clusters and because it works well on data of uniform density [8]. Additionally, as shown in Fig. 5, it works quite well for our use case, with red dots representing the two new computed centroids.

Finally, the centroids of the remaining regions and those computed in the previous two steps are combined into a single list. This list represents the coordinates of every emoji detected and will form the final output given to the official emoji-hunt test function.

III. Experiments & Evaluation

A. Experiments and Solutions

1) Mask Augmentation: As segmentation masks were (suspiciously) not provided natively with the emoji-hunt *[9]* library these needed to be implemented before any cool stuff could happen. Furthermore, some augmentations to these masks were found to improve test performance. To address the afformentioned case of overlapping emojis a 3-wide line of 0s

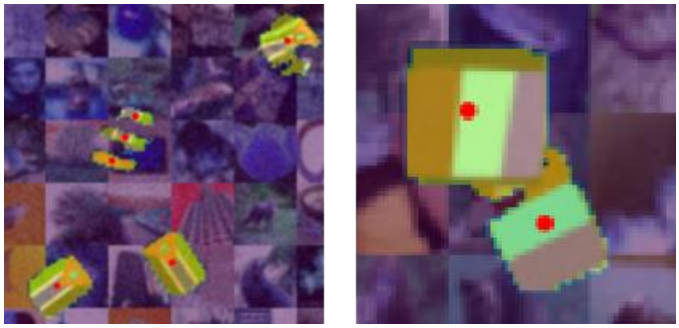

Fig. 5. Post processing edge cases. Clustering small regions into one emoji (left). Clustering large regions into two emojis (right).

was inserted into the segmentation mask along the intersection of 2 or more emojis as shown in Fig. 5. This lowered the official score by 20 points, a huge improvement. Visual inspections confirmed that the network was indeed now making an attempt to individually segment overlapping emojis but didn't always suceed (small bridges remain in some cases as in Fig. 5). Interestingly, experiments with increasing the size of the gap didn't improve the score further.

2) Post-processing: We noticed that the segmentation mask occasionally had a small amount of noise surrounding a detected emoji. As a result the noise was being detected as several distinct emojis, which greatly lowered the score of both models. The solution we implemented was to simply remove any regions that had a small area (under 45) from being further considered.

The next difficult case we found was where severe augmentations like coarse dropout would separate emojis into 2 or more parts (Fig. 5). This was fixed by considering all remaining small regions and grouping them by Euclidean distance, averaging their centroids together. Any small regions within a radius of 20 are grouped as a single emoji. This method can fail when 2 emojis are near to each other and both affected by severe coarse dropout, but we have yet to see this case arise.

Another pressing issue for our emoji-hunt score was overlapping emojis. While the augmented mask did help, additional processing was needed. We noticed that overlapping emojis (almost) always resulted in a large, odd shapes that could be detected as outliers. Originally detection of these was done purely by area (2) before clustering as discussed. This naïve method did detect obvious cases like in the right portion of Fig. 5 but failed on most others. Furthermore, it picked up on false positives when single emojis were simply bigger than most others. Adding perimeter and shape checks (3), (4) address these false positives nicely. The remaining weakness of this approach include the case when more than 2 emojis overlap (rare but has been observed) and the case when all, or almost all, the emojis in an image overlap and we have no reference for a true mean area.

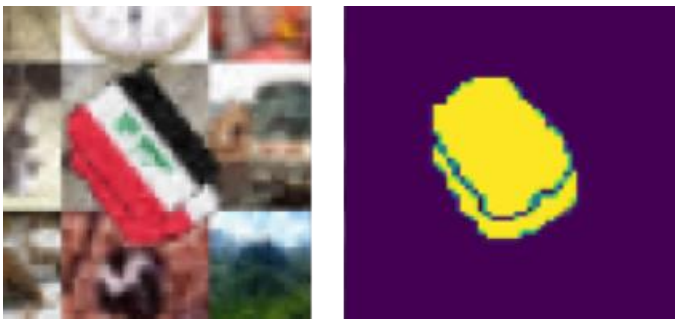

Fig. 6. Example of overlapping input (left) and mask augmentation (right). Note that green pixels in the segmentation mask are display artifacts and do not exist in the actual mask.

3) Colour Space: As a quick experiment we tried changing the input colour space from LAB (leftover from the classical approach) to BGR. This got us an easy 4% improvement in IoU and lowered the official score by 2. Very likely just due to better predictions leading to more accurate centroids in non-edge cases. Nice!

4) Image Resolution: In our training attempts we tried a few different input and output resolutions. At 512x512 (original input size) the network picked up on noise and parts of the background, very likely overfitting to a large feature space. Going down to 128x128 eliminated these signs of overfitting but gave disappointing IoU scores of ~0.8. 256x256 was a happy medium between the two, generalizing well and scoring an IoU of ~0.9. Resampling was performed using OpenCV's INTER_AREA and NEAREST_NEIGHBOUR methods for the image and mask respectively.

5) Matching Multiple Colors: For our classical approach, the main initial disadvantage we initially experienced was its inability to match emojis with several distinct colors, like flags. This is because for our initial implementation we only considered the average color of the entire emoji. This would result in the algorithm looking for clusters of colors that were not at all similar to the real colors in the emoji, which resulted in almost no emoji detections and several false positives. As a solution to this, we started segmenting the emoji using SLIC in addition to the image. We then modified the algorithm to look for groups of clusters that all had similar colors to the clusters as in the emoji using a sliding window method discussed in section 2.A. This did lead to an increase in score from the official emoji-hunt test function but did create a few new problems discussed below.

6) Too many hyperparameters – a classical story: After we finished implementing the classical algorithm we were faced with another problem, this long, complicated algorithm had several hyperparameters which all needed to be tuned. Furthermore, many of the hyperparameters were related, increasing one would mean you would have to decrease another, and vice-versa.

A prime example of this is the threshold used to create the binary similarity matrix. Setting the threshold too high will result in no emojis being detected, whereas a threshold that is too low will result in too many false positives being detected. This is further complicated when the challenge introduces augmentations. Many of the augmentations from the challenge like grayscale, multiply hue and saturation, and change color temperature modify the color of the emojis directly however, some spatial augmentations like coarse dropout end up modifying the color indirectly as well. This is due to how we determine the average color of a super pixel which has been segmented using SLIC. Since we are just taking the average color of all the pixels within the super pixel, noise introduced by spatial augmentations will modify the average color of a

super pixel, forcing the algorithm to use a higher threshold to detect the emojis which have been augmented, which in turn increases the number of false positives. You can see this in Fig. 3, where the average color of all the detected emojis is different from the color of the emojis in the input image. Fig. 3 also displays a good example of false positives being detected, as some super pixels in the background image had a similar color to the color of the target emoji.

In addition to the threshold, the algorithm also requires precise tuning of all of SLIC's hyperparameters for both the image and the target emoji. The 'ideal' parameters were found experimentally to be considerably different for each type of target emoji, which made generalizing the algorithm for all the emojis extremely difficult.

B. *Evaluation*

1) Performance: If you want the big flashy number, our score from the official tester is 25. We have some cool analysis if you keep reading though. As seen in Fig. 7 the modern approach scores under 5 in the grand majority of cases, however our approach is still prone to under-predicting the true number of emojis resulting in a massive score hit. As discussed, we were able to eliminate a lot of these failure cases but some still sadly remain. From a visual inspection of we see failures mainly on images with only 2 emojis that both overlap, which is not detectable by our approach as discussed. We are extremely happy with the performance of this approach though as it "perfectly" finds the exact centers of all emojis 80% of the time, and the worst possible score observed was "only" 156.

As shown in Fig. 8, the best score out classical method managed to achieve was 299. However, this score comes with some reservations. Since the requirements of this challenge were only to design one solution to the problem, once we saw that the modern approach was consistently outperforming the classical approach, we abandoned further hyperparameter tuning and development of the classical method. Given more time, it would have been interesting to further develop this method to see the most optimal score we could have achieved.

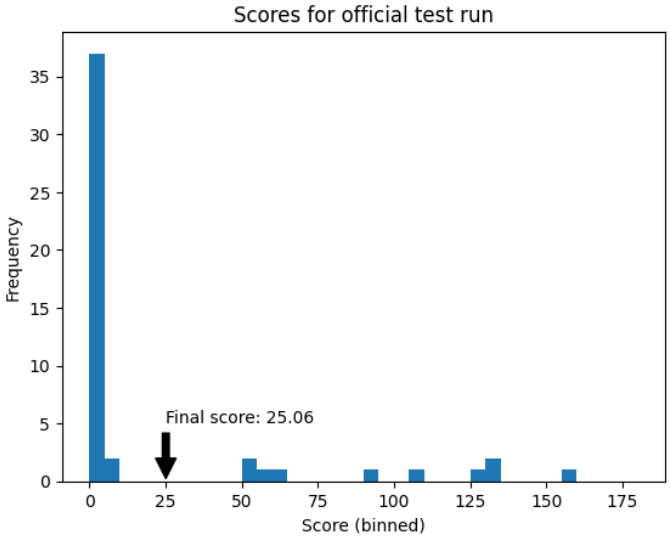

Fig. 7. Official scores for the modern approach, with bin size of 5

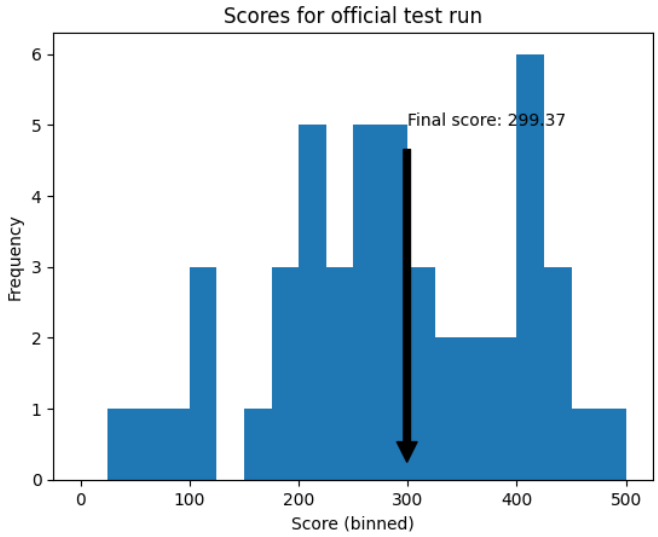

Fig. 8. Official scores for the classical approach, with bin size of 15

2) Proposed Improvements: Mask cleanup, clustering, and post-processing can only go so far, and we believe we are pushing U-Net to its limits for separating overlapping and multi-segment emojis. The authors *somehow* missed the release of Broad-UNet [10] which claims to learn more complex patterns while using fewer parameters. This could be very promising for our emoji hunting endeavours as a network that properly learns the gap between overlapping emojis would be ideal.

Another possible improvement would be to switch to a bounding-box predictor like YOLO [11]. This would possibly model the problem more accurately as in the end we are just predicting centroids and don't need a segmentation mask as an intermediary. This could even be combined with our current approach in an ensemble method to find even the sneakiest hidden, stacked, and corrupted emojis.

A possible improvement to the classical method would be swapping the sliding window approach for a graph-based approach. Effectively, instead of using a sliding window to find regions which match the emoji, each adjacent super pixel would be connected in a graph, and then the algorithm would attempt to find subgraphs which matched the graph of the adjacent super pixels of the target emoji. We envision this being a much more robust method. That being said, the authors strongly feel that this is a problem best suited for a neural network to tackle and would suggest that readers considering the emoji-hunt challenge first start with that before looking into classical methods.

IV. Conclusion

Our study implemented two approaches to tackle the emoji-hunt problem. The U-Net architecture, a modern approach, achieved excellent segmentation accuracy with an IoU score of 0.93. This indicates that U-Net was indeed able to apply to fields outside of medical imaging. Additionally, we applied post-processing techniques to refine the predicted masks from both approaches and predict the centers of each emoji. This greatly improved predictions in challenging cases, underscoring the effectiveness of relying on more than one method in computer vision tasks.

Furthermore, we evaluated the scores of each approach, with the modern one performing extremely well. However, we also identified opportunities for further improvement, particularly in detecting very difficult emojis that might have been missed or over-predicted due to their size, transformations applied, or overlap with others. As future work, we propose exploring newer network architectures and techniques that may potentially raise the segmentation score and remove the need for some post-processing steps. Overall, our study highlights the potential of image segmentation techniques, specifically the U-Net architecture, for solving the emoji-hunt problem and suggests avenues for future research to continue advancing the accuracy and robustness for others looking to locate sneaky emojis.

Interested parties can find our emoji-hunt code at our GitHub organization where you will find other cool projects such as Earthquake Simulator and Sealed Bid Auction. https://github.com/fern-software/EmojiHunt.

Acknowledgment

We'd like to thank Alexandra Branzan Albu and Declan Mcintosh for running ECE471 at the University of Victoria. We really enjoyed the course and this final project. We strongly encourage any readers of this paper to take ECE 471: Computer Vision at the University of Victoria if possible.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "SLIC Superpixels Compared to State-of-the-Art Superpixel Methods," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 11, pp. 2274–2282, Nov. 2012, doi: 10.1109/TPAMI.2012.120.

[2] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition." arXiv, Apr. 10, 2015. Accessed: Apr. 04, 2023. [Online]. Available: http://arxiv.org/abs/1409.1556

[3] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation." arXiv, May 18, 2015. Accessed: Apr. 04, 2023. [Online]. Available: http://arxiv.org/abs/1505.04597

[4] "Practical image segmentation with Unet," *Tuatini's blog*, Oct. 01, 2017. http://tuatini.me/practical-image-segmentation-with-unet/ (accessed Apr. 06, 2023).

[5] F. Chollet and others, "Keras." 2015. [Online]. Available: https://keras.io

[6] "Figure 2. The architecture of Unet.," *ResearchGate*. https://www.researchgate.net/figure/The-architecture-of-Unet_fig2_334287825 (accessed Apr. 06, 2023).

[7] S. van der Walt *et al.*, "scikit-image: Image processing in Python," *PeerJ*, vol. 2, Jul. 2014, doi: 10.7717/peerj.453.

[8] "k-Means Advantages and Disadvantages | Machine Learning," *Google Developers*. https://developers.google.com/machine-learning/clustering/algorithm/advantages-disadvantages (accessed Apr. 06, 2023).

[9] D. Mcintosh, "emojihunt." Accessed: Apr. 05, 2023. [Online]. Available: https://pypi.org/project/emojihunt/

[10] J. G. Fernandez and S. Mehrkanoon, "Broad-UNet: Multi-scale feature learning for nowcasting tasks," *Neural Netw.*, vol. 144, pp. 419–427, Dec. 2021, doi: 10.1016/j.neunet.2021.08.036.

[11] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors." arXiv, Jul. 06, 2022. Accessed: Apr. 04, 2023. [Online]. Available: http://arxiv.org/abs/2207.02696