WHAT'S IN MY POCKET?

A Graph-cutting & Neural Network-based Coin Detector

Computer Vision, Waseda University

2025-08-05

Kevin Kokalari
3225A907-0
kevin.kokalari@fuji.waseda.jp | kokalari@kth.se

Table of Contents

1. Introduction

Everyone has been in this situation before. You are standing in line at the cash register ready to pay, it's now your turn and you take out your wallet wanting to pay with cash. You open your wallet and see the complete mess of coins in the wallet's coin pocket; you try to take some of them out to count them and drop some of them. You pick them back up and after a couple minutes of counting you realize you are just short in coins. In the end, you pay with card making yourself look bad and making everyone else wait unnecessarily in the queue.

This is the reality for me and many others' living and traveling to mostly cash based countries from a heavily digitalized country. Coming from Sweden, a country claimed as the world's first cashless country, cash payments only count for 10% of all transactions (Bryant, 2025), with many people like me not seeing a bill in over a decade. This makes cash payments like in Japan and other countries inconvenient and annoying as the routine of remembering how much cash and coins you are carrying is lost. To combat this problem and the fact that counting coins manually takes time, I will in this computer vision project propose a method for quickly counting coins placed on a surface using a graph cutting approach for segmentation and a CNN for coin classification.

2. Method and Implementation

2.1 Segmentation

2.1.1 Problem Definition

Let's start off with the segmentation part of the implemented program. To understand the choice of segmentation model, we first need to understand what scenarios we might come up against regarding the segmentation. In an ideal world, we would want the coins to be placed on a well-lit mat and uniform surface which contrasts well with the coins in question and where the camera is at a constant distance perpendicular to this surface when taking the picture.

Figure 1a: Three different examples of ideal photos of different Japanese coins (Numista, n.d.; uCoin.net, n.d.; Coin Collecting Wiki, n.d.).

In realty, this will never be the case, and we can expect the picture of the coins to be from an angle, in bad lighting, with a rough background surface and where the zoom or cameras distance from the coins to be a variable. This means that any typical

Figure 1b: Three different examples of real-world scenarios of coin placements in an image (Numista, n.d.; Numista, n.d.).

segmentation methods using only pixel ratios or Hough circles alone won't be enough to reliably segment out the coins in the picture with good performance. This means that we will be using a graph cut based method, more specifically a variation of graph cutting called GrabCut, as we are advised to avoid using deep learning as best as possible.

2.1.2 Assumptions

Before going through how GrabCut works and the rest of the image processing done in this project, the assumptions made here must be presented. There are two important assumptions made here that need to be addressed:

1. *Coins cannot overlap*.
2. *Coins cannot touch each other*.

These assumptions might seem unrealistic for a real-world scenario, and they clearly are. After intensive testing and multiple different tries and different method used, it was concluded that these assumptions had be made for there to be any reliable result. Traditional, non-neural network-based segmentation methods were to unreliable or to poorly performing to be able to handle these scenarios in a satisfactory way.

2.1.3 GrabCut

GrabCut starts off by first receiving a predetermined bounding box around the object to be segmented out. The method then profiles and estimates the color distribution of the object targeted to be segmented out and the background. A Markov random field is then constructed from these distributions using an energy function which has a bias towards regions connected with the same label. Then it optimizes using graph cuts to infer the random field's and energy function's values. This is done iteratively until convergence as the new estimate is assumed to be more accurate than the original ones. The method also corrects the results so that edges don't disappear (Rother, Kolmogorov, & Blake, 2004).

2.1.4 Preprocessing

The segmentation part of the program starts off by first taking the green channel of the original image and then slightly blurring it using a Gaussian blur. Typically, an input image is first converted into a black-and-white monochrome image before blurring and doing other types of processing. This was initially the case also in this project, however after testing, the overall performance of the segmentation seemed to increase when choosing a color channel instead of a gray scale.

The reason behind this could be that the later parts of the segmentation has difficulty in detecting features and contrast/color changes. This is most likely because some of them being lost in the color space conversion due to the conversion function's implementation or due to the new color space does not making the fore- and background contrast as much. All these reasons are just theoretical explanations; the exact reason of the performance drop was not explored further.

After selecting the green color channel, the program does a localized contrast normalization using Contrast-Limited Adaptive Histogram Equalization. The algorithm tiles the input image with a given size by the user. Then for every tile it generates, it calculates a contrast histogram for that tile and cuts of the bin tops which are above a

predefined contrast limit, called a clip limit, given as an input. Without this clip limit, the algorithm would “over-contrast” general noise in the given tile (Mathworks, n.d.).

These cut tops are then redistributed across all the bins in the histogram. A cumulative distribution function is then defined with this adjusted histogram, and all the bins are then adjusted to this function. At last, all the tiles are then stitched together, resulting in a contrast adjusted image of the original (Mathworks, n.d.). In this project, the tile size was set to 8×8 and the clip limit to 1.7 as those values resulted in the best overall performance of the whole program.

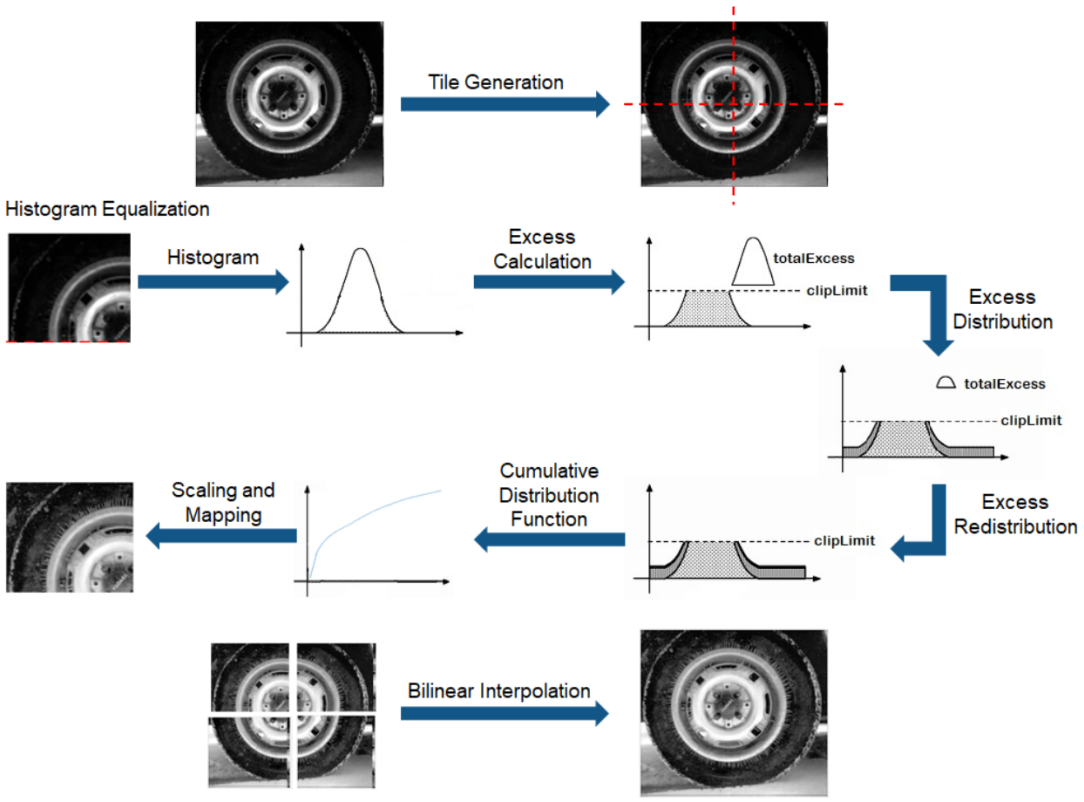

Figure 2: A figure of how the CLAHE-algorithm works taken from MathWorks (n.d.)

After the contrast normalization, a Gaussian blur was then applied to the image using a kernel size of 11×11 with a standard deviation of 2.2. This is done to remove noise from the image as well as to reduce unnecessary detail of the picture, making it easier for later parts to detect coin shapes.

2.1.5 Circle Detection

The program then does a pre-detection round using Hough Circles Transform to find a mask containing a collection of potential circular shapes (OpenCV, n.d.). These potential circles found in the picture are then used to define a potential foreground of the picture. If no circles are found by the Hough circles algorithm, then an area around the center point of the picture will be considered as the potential foreground. This is because a user taking a photo of the coins will most likely place the coins in the center of the picture frame anyways.

After detecting all potential circular objects, a Grab Cut seed mask is defined by initializing it with the whole picture as a sure background. Then, for every circular object found previously, we take its center point and radius, r, and draw two discs from this center point. The first disc with radius r which we label as the foreground, and the second disc with radius $1.3r$ which we label as a probable foreground. This is done to compensate for the fact that Hough circle is not very precise. Grab Cut is expected to make the precise coin cutout and for that, only an approximate area of the coins on the surface is needed. In summary, the algorithm only needs to give Grab Cut good guesses for the coins.

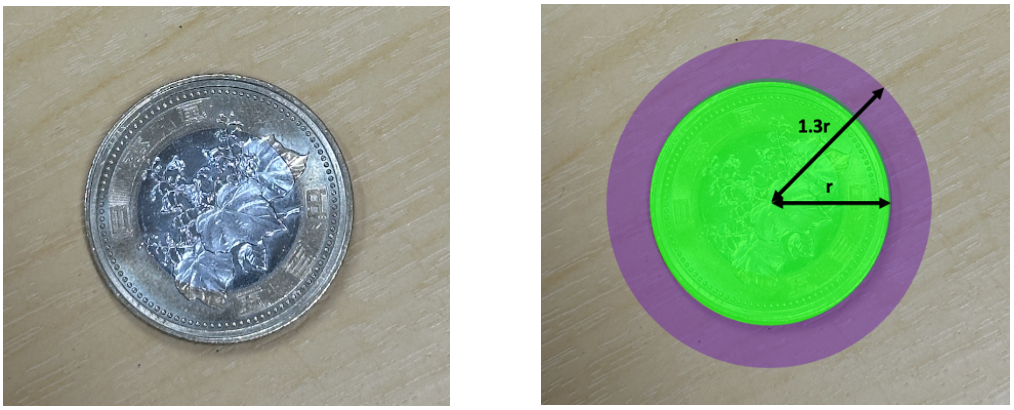

Figure 3: The original picture of the coin to the left, the two drawn discs on top of the coin to the right

Then Grab Cut was used twice, the first time on these disks to segment out the exact coin disk from the picture, the second time starting from the seed mask to improve the overall performance. Grab Cut will then label every found segment either as a sure or probable fore- or background. The results are then combined so that the sure and probable foreground labels merged into one foreground mask.

Lastly, the program iterates over every found foreground object and then use area-based noise filtering. It removes foregrounds which are smaller than 0.04% of the total area of the image and larger than 25% of the full image area. This is done as foregrounds smaller than 0.04% of the total area can be considered as general noise while foregrounds larger than 25% is most likely not a coin. A bounding box is then created with 10% larger sides than the coins radius to prevent later steps of the program from trimming away the coins' rim.

2.2 Face Detection

2.2.1 Problem Definition

Before going into depth over how the detection of the coins' faces were made, let's first talk about the different hurdles which exists when detecting coins denomination. A coin typically has a distinct metallic color and luster which sets it apart from other coins with other denominations. However, this shine significantly degrades overtime as the coin is

used and circulated which turns the coins bright and shiny color into a darker and more matte finish. If you look at the two examples found from Google Images below of 10-yen and 5-yen coins, it's clear that this wear significantly changes the characteristics of the coins' look, making detection harder.

Figure 4a: The face of a well circulated 10-yen coin on the left (Gumnut Antiques, n.d), and a brand-new 10-coin to the right (Hobby of kings, n.d.).

Figure 4b: The face of a well circulated 5-yen coin on the left (uCoin.net, n.d.) and a brand-new 5-coin to the right (Coin Collecting Wiki, n.d.).

A traditional image embedding and feature detection functions would typically consider the color and luster distribution when vectorizing the input images. The question here would then be how one reliably would be able to detect these color- and luster shifts without generating false positives? This isn't the only problem with detecting the face value of a coin either. A single denomination could also have multiple versions of itself which is the case with the Japanese 500-yen coin as an example. Below, these three versions are common and all legal tenders in Japan (Ministry of Finance Japan, n.d.).

Figure 5: Three versions in circulation of the 500-yen coin in Japan (Ministry of Finance Japan, n.d.).

2.2.2 Classifier

These are some of the difficulties faced when detecting the face value of each coin. During initial testing in the beginning of the project, these difficulties proved too problematic for the more traditional and classic machine learning algorithms to handle, such as Decision Trees, nearest neighbor methods, Bayesian methods and ensembles of these.

In the end their performance resulted in an accuracy of up to approximately 65%, which is better than random selection but still not satisfactory results which forced us to use a deep learning-based detection approach. More specifically, the model used is MobileNet v2 which is a lightweight CNN, specifically created for mobile and embedded computer vision tasks (Sandler, Howard, Zhu, Zhmoginov, & Chen, 2018).

The CNN is a model with a backbone and body pretrained on ImageNet, a large open-source database of images manually annotated objects, used for visual object classification and detection in research (Deng, Dong, Socher, Li, Li, & Fei-Fei, 2009). The CNN model's 50 topmost layers in the backbone was also finetuned with a small learning-rate as to not ruin the pre-trained weight. The final weights are then saved for later use.

2.2.3 Training

The head of the neural network is trained using 45 unique pictures for every type of coin existing in the Japanese currency, while 5 unique pictures are used for validation for every denomination. The heads number of neurons was arbitrarily set to 384 using the ReLU activation function. Another layer was then added to the head with 6 neurons with soft-max as an activation function. This final layer represents one neuron for every class and can be seen as the classifier part of the network.

The training data is extended through on-the-fly augmentation by for example rescaling the image, rotating it and shifting the zoom, brightness, width range, and height range, while the validation data is only extended using rescaling. Each image is labeled using its parent folder name and shaped with one-hot encoding.

2.2.4 Detecting New Instances

For every coin the segmentation part of the program found in the image, we add a safety padding around all sides the found coin limited by the size of the original image. The coin is then cropped out from the original source image and the color space is then later changed from BGR to YUV. This is done so that the program can enhance the contrast of the coin without introducing color distortions.

Like before, contrast normalization using CLAHE was then used but this time on the Y-channel to locally boost contrast without amplifying extreme noise. Lastly, the coin was resized to 224×224, the largest resolution allowed by the CNN, and then converting the image back into BGR. The preprocessed coin is then classified by the CNN, and its denomination is then returned only if the CNN is more than 40% sure of the label being

correct. This number was chosen as it resulted in the best overall performance. The bounding box of the coin and its value is drawn on the original image and its value is added to the total sum. This process is done for every coin found in the image until no more coins are left.

2.3 Evaluation

2.3.1 Confusion Matrix

For the evaluation of the denomination classification, a traditional multiclass confusion matrix was used, where every coin denomination is a separate class in the matrix. There is also an extra class label in the matrix called "None". This was added for all the occurrences when the segmentation failed to find all the real coins in the image and thus not sending a segmented coin to the classifier for labeling. Any coin missed to be classified completely will be marked as "None". The true value for None is assumed to always be zero.

The confusion matrix will then be used to calculate different performance metrics where the labels TP, FN and FP stand for all *True Positive*, *False Negative,* and *False Positive* predictions respectively for all images while the noncapitalized ones are per image basis. These are more closely defined in *Table 1* below:

Term	**Definition**
True Positive (TP)	Coins counted and classified correctly.
False Positive (FP)	Extra coins hallucinated by the model.
False Negative (FN)	Real coins the model missed in the image.
True Negative (TN)	**???**

Table 1: The definitions of every classification scenario and what they practically mean

As it is clearly in *Table 1*, the *True Negative's* definition is missing. The reason for this is simply what would a True Negative even be or mean in this case? In binary classification tasks that question is simple, the classified datapoint is either positive or negative and a sample which is both predicted as and truly is negative would be a True Negative sample.

In this case regarding coin classification, every coin in the dataset is a positive sample and everything else which is not a true coin is just some undefined background. For every object in an input image that is not a coin, the model would in reality predict nothing, meaning that the True Negative term would converge towards the image resolution.

Let's say that the True Negative term was included, by for example counting every equal true denominator with a zero-count and a "predicted" zero count as True Negative. This would mean for example that an Accuracy-metric would converge towards or be close

to 1 even though most of the coins were mislabeled, making it misleading and inaccurate at best.

2.3.2 Precision

The precision metric was used to detect how much the overall program hallucinates coins which doesn't exist in the original image as it directly penalizes these instances. In general, high precision ensures high trust as every classified coin will most likely be a real coin. This means that the risk of the total cash value in the picture being overinflated is low in cases with high precision (scikit-learn Developers, n.d.; Powers, 2011). The average precision per denomination is defined as:

$$Precision\ (\%) = \frac{TP}{TP + FP}$$

2.3.3 Recall

Recall as a metric was used here to detect when the overall program misses coins in the given image. Similarly to the Precision metric, the Recall will penalize any coins missed by the program as missing coins in the image is assumed to be as bad as hallucinating extra coins. It indicates if the segmentation and/or detector is missing or mislabeling coins (scikit-learn Developers, n.d.; Powers, 2011). The average recall per denomination is defined here as:

$$Recall\ (\%) = \frac{TP}{TP + FN}$$

Recall together with precision tell us in summary why the model fails, i.e., if the model is too aggressive or too careful.

2.3.4 Multi-Class Accuracy

Multi-class accuracy is used as a metric which combines booth detection correctness as well as classification into one metric. It's simple, intuitive and easy to understand and use to compare and benchmark other future coin detection and classification models with the one presented in this report. The metric can be understood as for every prediction made, what fraction of these predictions had the exact denomination correct (scikit-learn Developers, n.d.; Powers, 2011). In this report, the Multiclass Accuracy is defined as:

$$Multiclass\ Accuracy\ (\%) = \frac{TP}{All\ Predicitons\ Made}$$

2.3.5 Detection Rate

The Detection Rate measures and balances both hallucinations and detection miss while ignoring the true negative label. It stays honest this situation where true negative is not considered and mimics the closest to how a human assessor would grade the result. Basically, it considers all model mistakes and then calculates what fraction of coins are handled perfectly by the model (scikit-learn Developers, n.d.; Powers, 2011). The average detection rate per image is here defined as:

$$Detection\ Rate\ (\%) = \frac{tp}{tp + fp + fn}$$

3. Result and Discussion

3.1 Qualitative Results

The results below are some of many examples of different scenarios and how the algorithm performs. As it can be seen, the algorithm overall, performs well in different lighting conditions, viewing angles, zoom levels, surfaces and backgrounds, different coin counts, as well as when the coin isn't fully segmented out. For reference, some of the images have some of their backgrounds cropped out to better fit the report formatting. In reality, most of the examples below have larges resolutions and sizes.

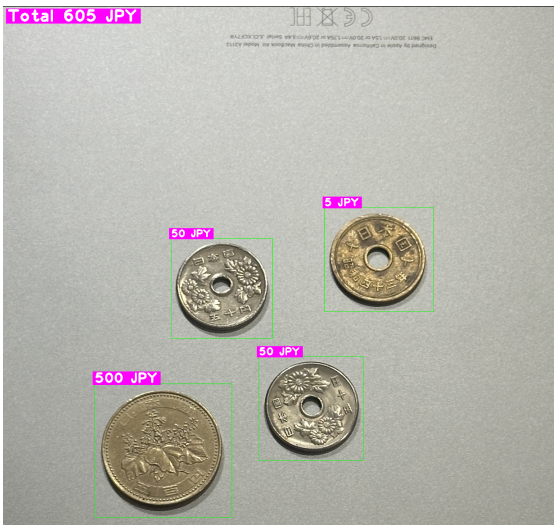

Figure 6: Example Results in different scenarios

3.2 Quantitative Results and Discussion

The resulting confusion matrix after running the test dataset containing 63 unique and previously unseen images by the model; with a coin label distribution of 35 coins per denomination.

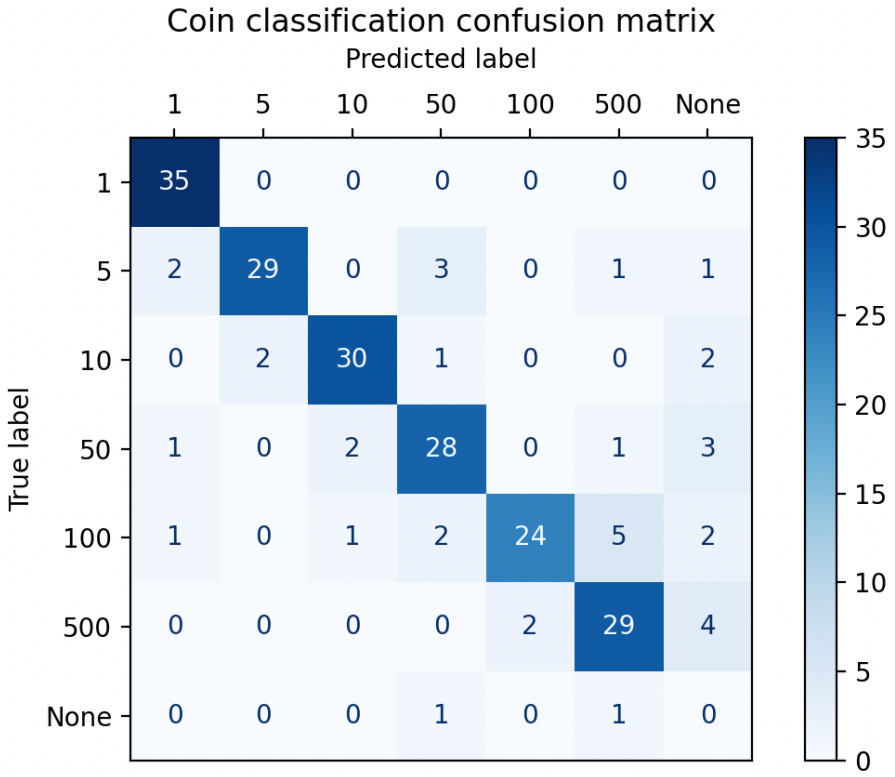

Figure 7: Confusion Matrix after detecting coins from 63 unique images

What can be observed is that the model seems to have the most difficulty in detecting 100-yen coins, most likely because the 100-yen coin's features and surface is relatively weak compared to other coin types as well as it having a similar color and finish to the 50-yen coin. Also, it was observed that the model in general had difficulty in differentiating the backside of the 100-yen coin with the older 500-yen coin in some lighting conditions.

As can also be observed above are the multiple predicted and true "None" values. As previously mentioned, the None-label means that the model hallucinated and predicted something as a coin when it's not, represented by the None-row in the matrix, as well as when the model misses a coin which exists within the image, represented by the "None"-column. The "true positive" for the None-label or equivalently true negative for the confusion matrix, depending on how one sees it, will always be 0 as previously discussed in the Method.

What can be observed is that there seems to be an overweight in times where the model misses to detect coins which exists in the image compared to the model hallucinating. This is an indication of the segmentation part of the model having difficulties in detecting a fully correct foreground with coins in the image something which will be

discussed later. It can also be observed by looking at the None-row that the model rarely hallucinates new coins in the image which do not exist. This is something which also gets confirmed when looking at the table below.

Metric	**Score (%)**
Average Detection Rate per Image	84,46
Average Precision per Denomination	87,48
Average Recall per Denomination	82,95
Multi-Class Accuracy	82,16

Table 2: The average result for key metrics as well as the multi-class accuracy.

When looking at the table above, it can clearly be observed that the average precision is higher than the average recall. This confirms what the confusion matrix showed that the model is less likely to hallucinate extra coins than to miss coins that already exist in the given image. The average detection rate is also higher than the remaining two other metrics which is most likely influenced due to the fact that the precision is noticeably higher than the recall of the model.

Denomination	**Avg. Recall (%)**	**Avg. Precision (%)**
1	100,00	89,74
5	80,56	93,55
10	85,71	90,91
50	80,00	80,00
100	68,57	92,31
500	82,86	78,38

Table 3: The average result for key metrics as well as the multi-class accuracy.

The table above shows the recall and precision when presented on a denomination basis. For the recall column, there are two clear outliers that can be observed, namely the 1-yen coins and the 100-yen coins. In the 1-yen coin case, when the model detects that there is a coin in the image and that coin's true label is 1, then it will always classify it as a 1-yen coin. This could be attributed to the fact that the 1-yen coin is the smallest coin, has the brightest color with a light-gray color with a blue tint, and doesn't seem to have the same luster and color deterioration as other denominations, making them easy to detect compared to other coin denominations.

For the 100-yen coin, the lower recall could be attributed as previously mentioned to coins surface features, i.e. the coins text and patterns, being relatively weak compared to other coins and that the backside also has some similarities with the 500-yen coin making them hard to distinguish. Also, the color and luster of the 100-yen coin is practically the same as the 50-yen coin making it even more difficult to detect.
In the precision column, the result is relatively uniform at around 90% with two exceptions: for 50-yen coins and for 500-yen coins. The reasoning for these two-coin

denominations can be similarly explained as previously. The model was observed to have difficulty in differentiating between the backside of 100- and 500-yen, coins due to its similar backsides in certain light condition, and difficulty in the similar color and luster that the 50-yen and 100-yen coins. It was also observed to have difficulty in distinguishing between the 50-yen and the 5-yen coins in certain lighting conditions and viewing angles due to the middle hole being a dominant feature for both the coins.

3.3 Segmentation Method

The most unexpectedly difficult part of this project was the segmentation algorithm. As previously mentioned in the method, the project assumed that the camera was non-stationary, that there could be multiple coins, coins couldn't touch or overlap, and that different viewing angles other than orthogonal to the coins were allowed.

These assumptions proved the segmentation difficult as the traditional segmentation methods in computer vision, such as Hough Circles, alone couldn't reliably handle all these real-world scenarios. Because deep learning was discouraged to use, the decision was made to base the segmentation on graph cutting which gave satisfactory but likely not as robust results as with a neural network.

Below is a list of different cases presented where this project's graph cutting based segmentation performs in a non-robust or a non-satisfactory way compared to a CNN-based segmentation method. The segmentation method in question used in the example below is Meta AI's Segment Anything segmentation algorithm (Kirillov et al., 2023). The Segment Anything algorithm is the right-hand image while the Graph cutting based one is to the left. Each example below has had some of its backgrounds cropped out to fit the report layout and for clarity. The background in the real images from the dataset is larger.

Figure 8: Example with irregular & uneven background. To the left: Graph cutting; To the right: CNN

Figure 9: Example with a high-reflective background. To the left: Graph cutting; To the right: CNN

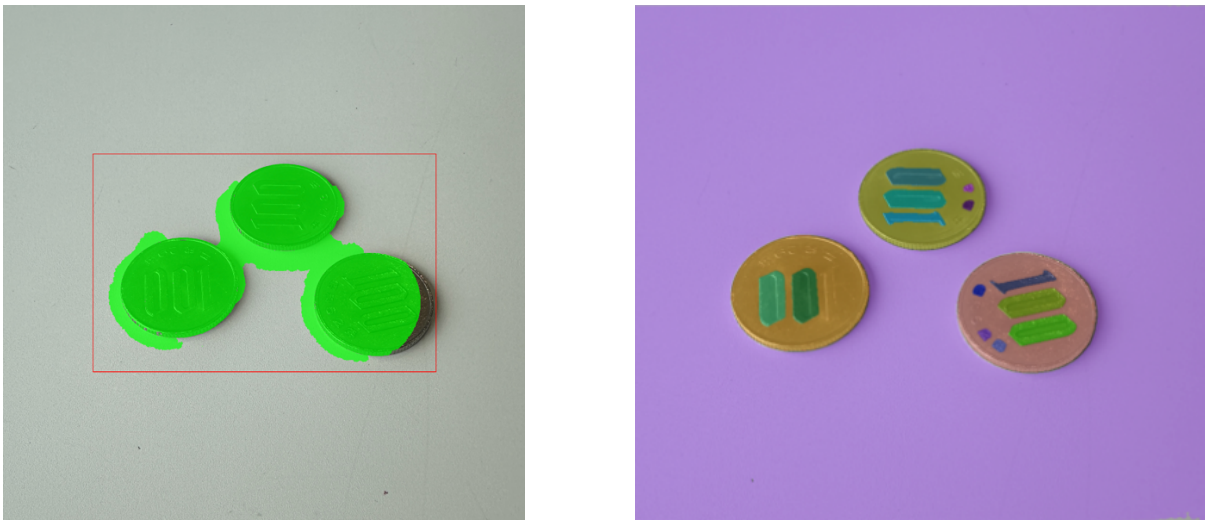

Figure 10: Example with closely placed coins with a significant viewing angle.

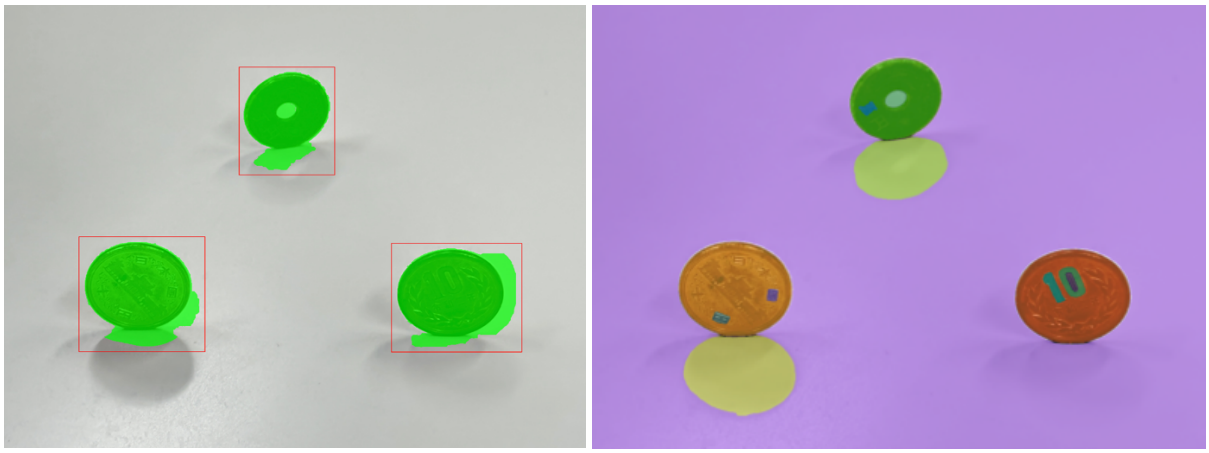

Figure 11: Example with non-ideal lighting conditions and an even more extreme viewing angle.

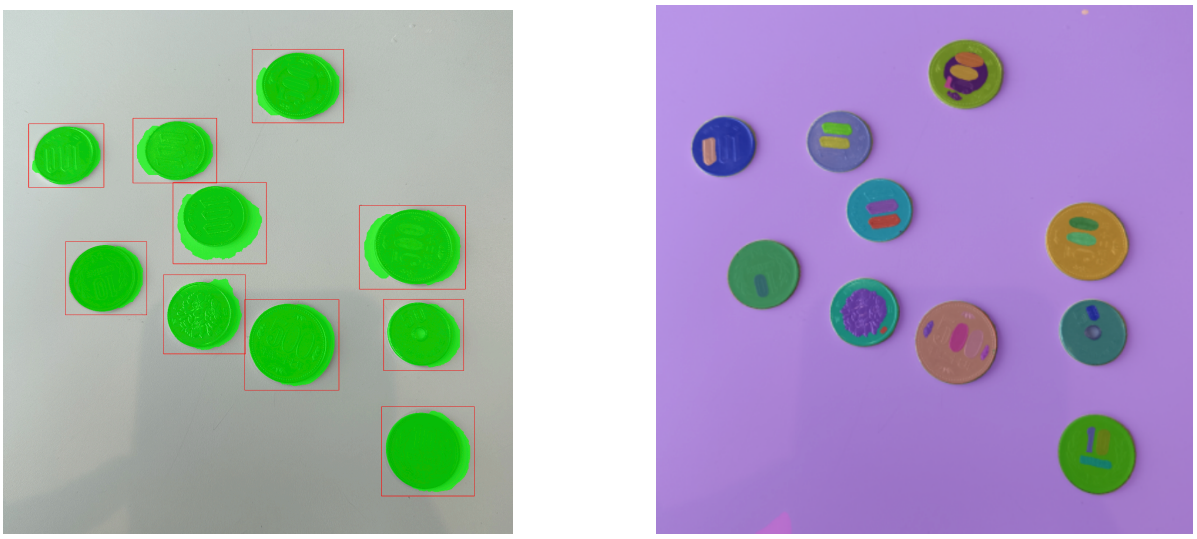

Figure 11: Example with multiple coins in one image. To the left: Graph cutting; To the right: CNN

As can be seen above with these more difficult examples, the CNN-based segmentation seems to perform significantly better. The issue in the CNN examples with the coins where their text and patterns were segmented out can be mitigated by doing relative area filtering as well as adding more blurring.

The above examples should not be interpreted as the model implemented here is performing poorly, it's just that a CNN-based segmentation algorithm will perform better but not perfectly, as can be clearly seen in the last example, in more difficult scenarios. The downside of the Segment Anything algorithm used here is that it cannot be locally run without enterprise-grade hardware. However, the point made here is that a smaller CNN which can be run locally would likely perform somewhere in-between the graph cutting method and the Segment Anything method.

References

1. Bryant, M. (2025, March 16). Back to cash: Life without money in your pocket is not the utopia Sweden hoped. The Guardian. Retrieved from https://www.theguardian.com/technology/2025/mar/16/sweden-cash-digital-payments-electronic-banking-security

2. MathWorks. (n.d.). Contrast limited adaptive histogram equalization. In Vision HDL Toolbox documentation. Retrieved July 20, 2025, from https://www.mathworks.com/help/visionhdl/ug/contrast-adaptive-histogram-equalization.html

3. Ministry of Finance Japan. (n.d.). Currently issued coins. Retrieved July 20, 2025, from https://www.mof.go.jp/english/policy/currency/coin/circulating_coins/list.htm

4. Rother, C., Kolmogorov, V., & Blake, A. (2004). "GrabCut"-Interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics, 23*(3), 309–314. https://doi.org/10.1145/1015706.1015720

5. Yuen, H. K., Princen, J., Illingworth, J., & Kittler, J. (1990). A comparative study of Hough transform methods for circle finding. *Image and Vision Computing, 8*(1), 71–77. https://doi.org/10.1016/0262-8856(90)90059-E

6. OpenCV. (n.d.). *Hough Circle Transform*. OpenCV Documentation. Retrieved July 20, 2025, from https://docs.opencv.org/4.x/da/d53/tutorial_py_houghcircles.html

7. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 4510–4520). https://openaccess.thecvf.com/content_cvpr_2018/papers/Sandler_MobileNetV2_Inverted_Residuals_CVPR_2018_paper.pdf

8. Meta AI Research. (2023). *segment-anything* (GitHub repository). https://github.com/facebookresearch/segment-anything

9. scikit-learn Developers. (n.d.). *precision_recall_fscore_support*. In *scikit-learn* (v1.6) Documentation. Retrieved July 21, 2025, from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html

10. scikit-learn Developers. (n.d.). *accuracy_score*. In *scikit-learn* (v1.6) Documentation. Retrieved July 21, 2025, from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

11. Powers, D. M. W. (2011). Evaluation: From precision, recall and F-measure to ROC, informedness, markedness & correlation. *Journal of Machine Learning Technologies, 2*(1), 37–63. (Author copy). https://bioinfopublication.org/files/articles/2_1_1_JMLT.pdf

12. Gumnut Antiques. (n.d.). *Gumnut Antiques – Antiques on the Central Coast*. Retrieved July 14, 2025, from https://gumnutantiques.com.au/

13. Hobby of Kings. (n.d.). *Japan 10 Yen – Heisei coin Y97 (1989–2019)*. Hobby of Kings. Retrieved July 14, 2025, from https://www.hobbyofkings.com/sv/products/japan-10-yen-heisei-coin-y97-1989-2019

14. uCoin.net. (n.d.). *Japan 5 yen, 2002* [Catalog entry]. Retrieved July 14, 2025, from https://en.ucoin.net/coin/japan-5-yen-2002/?cid=2250

15. *Coin Collecting Wiki.* (n.d.). *JPY 5 Yen*. In *Coin Collecting Wiki*. Retrieved July 14, 2025, from https://coincollecting.fandom.com/wiki/JPY_5_Yen

16. Numista. (n.d.). *5 yen – Heisei (Japan)* [Coin catalog entry]. Numista. Retrieved July 14, 2025, from https://en.numista.com/catalogue/pieces2676.html?mobile=0

17. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 248–255). https://doi.org/10.1109/CVPR.2009.5206848

18. Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., … He, K. (2023). Segment Anything. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. https://arxiv.org/abs/2304.02643