Real-time CNN coin classification with OpenCV data acquisition

Hannah Stone, Nguyen Khanh Vu and Thimo Blom

Abstract – **Convolutional neural networks are a powerful and flexible tool for image classification. This flexibility has allowed us to shape a pre-existing model into one that can distinguish between Euro coins at real-time speed. All training data specifically for the coin classification was collected by ourselves using a coin detection program that we also created. The general concept of real-time image classification has endless applications and this project acts as an example of one.**

I. Introduction

THE project initially did not begin with the decision to use a CNN for classification. The original method to be used was a Haar Feature-based Cascade Classifier. This was a classifier built of many 'basic' classifiers which are applied subsequently. When testing this method, the accuracy in coin differentiation was poor and with further research it was determined that while it was good at identifying what objects were coins, it was not as good at identifying what *type* of coin it was. This led us to change classification methods and we decided on using a CNN which seemed to have better performance in more specific object type classification. The specifics on what CNN we used and how we adapted it to our project are discussed later in the report.

Before any classification however, we had to create a way to collect data for our classifier. This is where our data acquisition utilising the OpenCV library came in use. It allowed us to create a program which could identify circular objects in the image and capture a cropped version of each object to be saved as training data. This process was done at the press of a button so less time had to be taken to physically collect the data. The techniques used were similar to that of the last lab assignment.

II. Concept Of Operations

The general working overview of the system is to receive image input and to then display real-time feedback of what the system thinks each coin is. This runs for every frame of image input making it real-time.

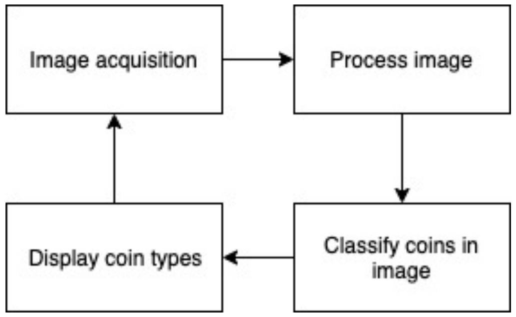


Fig. 1. Block diagram showing a high-level overview of the system's function

This classification system will act as a prototype / model for a commercial application. An example of one specifically relating to coin classification will be the use in banks for counting large quantities of money. Our system will be able to classify as many coins that can fit in one image, which could potentially increase counting efficiency in comparison to existing physical systems that already exist. Another example could be the use in an application for visually impaired users where they could potentially use their phone camera to conveniently count how much money they perhaps had in their hand instantly. Due to the expandability of this system, with enough training data, it could classify between hundreds of different coins if needed.

Therefore, our system is applicable for not one specific application. The broad application is for anything that requires coin classification. This could be for counting purposes or just distinguishing between them. It is difficult to specify particular stakeholders in our project as our end user is not limited. In the bank application example, stakeholders would include the bank company itself, the employees using this system and the customers who would benefit from a potentially accurate and efficient system. In the visual aid example, stakeholders would include the visually impaired user. Ultimately, the system's use is broad and applicable to many uses.

III. System Requirements

The system can be divided into two parts – data acquisition and classification. These will have separate functional and non-functional requirements. Due to the complexity of development, there are not many notable requirements. An increase in system requirements would have not been realistic given the short time constraint. Additionally, we desired to put our main focus on specifically the goal of creating a real-time classifier for coins without a specific implementation or user interface. With more time, these requirements could be expanded to implementing this classifier into a mobile application or user-friendly program for example. While the requirements seem few, the underlying process of completing some of these requirements were complex and or time consuming.

A. *Data Acquisition*

TABLE I
DATA ACQUISITION REQUIREMENTS

Type	Requirement
Functional	Identify circular objects
	Create visual edges on the circular objects for user interpretation
	Save cropped images of each object in respective folder
Non-functional	Find centre of each circular object for use in edge visualisation and image cropping
	Identify circular objects with > 90% accuracy
	Program will work at real-time speed

B. *Coin Classification*

TABLE II
CLASSIFICATION REQUIREMENTS

Type	Requirement
Functional	Receive image input from camera
	Output coin type predictions
	Visually label coins
	Display probability for each prediction
Non-functional	Predict coin types at a >90% accuracy
	Continuously receive image input into the neural network for processing
	Function quick enough to provide real-time feedback

IV. SYSTEM DESIGN

As mentioned before in the system requirements, the system can be divided into two main parts that were worked on separately but later combined to create the final system.

A. *Components*

- USB webcam
- Laptop

As we are only working with image processing, no other physical components are required. During development, several different tools and pieces of software were required to run and build the final program.

B. *Tools*

- OpenCV
- VGG16 CNN Architecture
- Keras
- TensorFlow
- Google Colab
- GitHub

For the data acquisition, the OpenCV library will be used to apply the image processing functions and blob detection. It will be written in Python as this language was what we had most experience in compared to using this library in C++. For the classifier, we selected the VGG16 CNN architecture [1] due to its high accuracy performance and relatively easy implementation [2] for which we utilised the Keras and TensorFlow library. These are both machine learning specific libraries for Python. In order to train the neural network, we had to use Google Colab as our own computational power was not enough. With Google Colab, we could use their GPUs for training the network, saving us time. All data needed to be shared between us all so we used GitHub to upload and import data where needed whether it was for training or testing the neural network.

V. CODE OVERVIEW AND IMPLEMENTATION

A. *Data Acquisition*

In order to collect cropped images of coins, we needed to process the raw image so sufficient blob detection can occur. We utilised a few techniques learnt from the pervasive computing textbook [3] and also others from existing similar projects. However, first the program must be calibrated so that an accurate background foreground distinction can occur. This is done by setting the newBg parameter to true in the main function. When this occurs, the program will take the first frame of the video input from the camera and save it as the reference background image. This means when calibrating, initially there needs to be no coins in the image frame. After the first frame is saved, all subsequent frames will be processed but not saved.

The background image and current image is first converted from RGB to HSV. We chose to convert it to this colour space because it provided the best distinction between coin and background for thresholding. Also, the absolute difference is found between the two HSV images in order to perform background subtraction which will be used next to create a foreground mask.

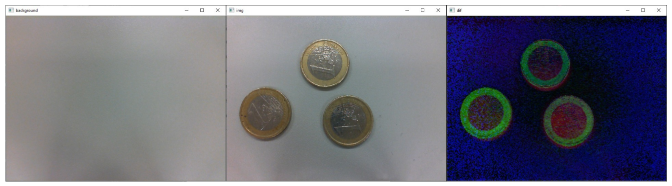
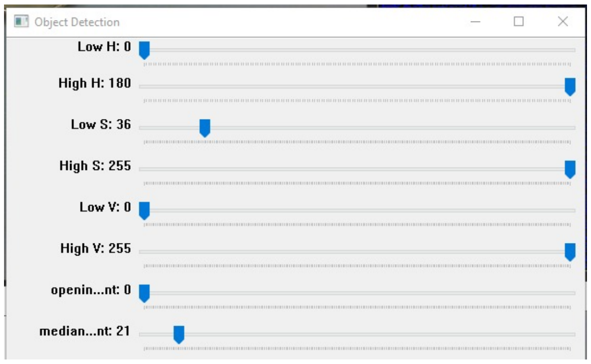


Fig. 2. RGB background image, RGB coin image, HSV coin image

In order to create a mask to separate the foreground from background, we needed several threshold values to select against. When testing existing thresholding functions, it did not select correct values accurately so we chose to adjust these values ourselves. To make this process easier, we created an extra window to have sliders so that we could adjust these values in real-time and receive immediate visual feedback on the accuracy of the thresholding. The threshold values needed to only be adjusted when setting up the system. If the environment changes significantly in the image, these values



Low H: 0
High H: 180
Low S: 36
High S: 255
Low V: 0
High V: 255
openin...nt: 0
median...nt: 21

Fig. 3. Window with several sliders for value adjustment

will need to be adjusted again. For us, it was sufficient enough to have this adjustment process be manual as we were using the system in a relatively controlled environment.

The thresholding values consisted of a lower and upper bound for each pixel's hue, saturation and value (brightness). Therefore, if a pixel's HSV value is within the threshold, it will be considered part of the foreground and vice versa. This produces a binary image. To also improve the foreground mask, we applied two morphological operations – opening and median blur. The opening operation first erodes and then dilates the image using the same size structuring element. We use this to help reduce noise in the mask image but preserve the shape and size of the coins. Median blur is also used to reduce noise in the mask image. The central element is replaced by the median of all the pixels in the kernel area. It also helps to create more distinct edges. The size of the structuring element / kernel for both of these operations is also adjusted in the slider window.

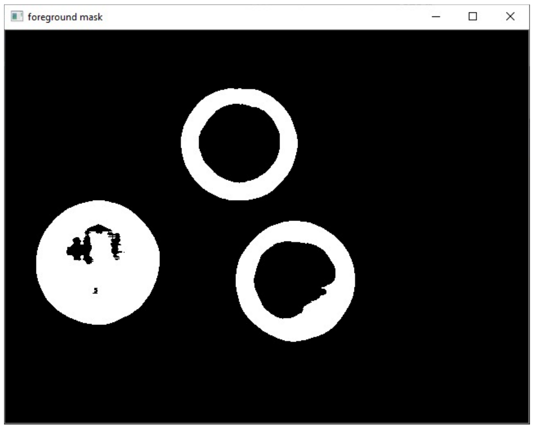


Fig. 4. Foreground mask of three 1 Euro coins

As seen in Fig. 4, for coins such as 1 or 2 euro, the centre part is considered as the background due to it being similar in HSV value as the background. To completely fill the areas containing coins, we needed to find the edges of the coins. To do this we utilised a function in OpenCV called findContours which finds edges in a binary image. This allowed us to fill these edges so that they became solid circular objects as seen in Fig. 5.

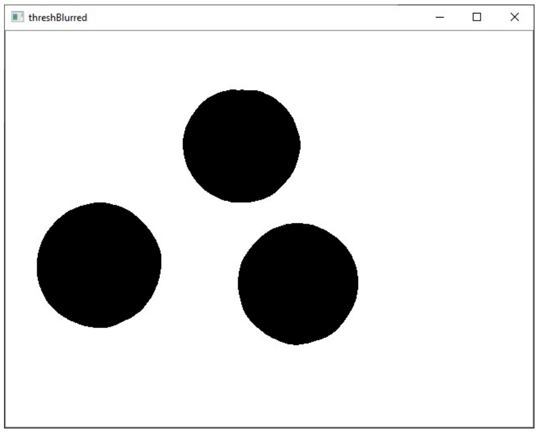


Fig. 5. Filled circles to complete mask

Now that we had clear distinction between background and foreground, we needed to find the centre of the objects so that they can be cropped accordingly for data collection. To do this, we used the SimpleBlobDetector class which not only identified which objects were blobs but also only selected the ones that had a minimum circularity of 0.8. The detect function returned the centres of the blobs which we then used as the centre of the cropped image.

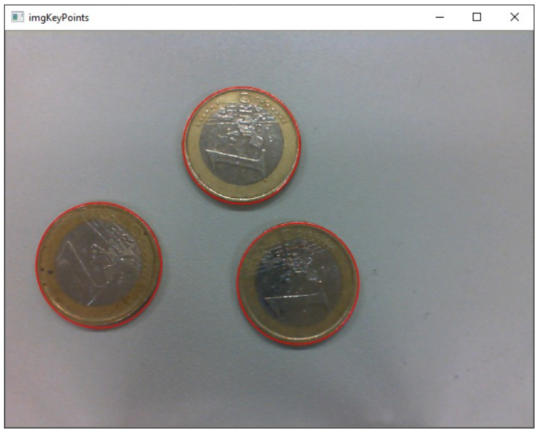


Fig. 6. Red circles indicate that the coins have been identified as circular objects

To now save individual coin images for classification, we created a function called getCoinImages. This uses the centres of the blobs returned previously and first checks if the whole coin can fit in the desired image size. The neural network requires images that are 224 x 224 pixels in size. If the coin is able to fit, all the coordinates of pixels from the raw image within the desired image size will be added to an array. Each cropped image in the array will be saved to the computer using our function saveImages. The program has been designed so that images will be saved when the 'S' key is pressed.

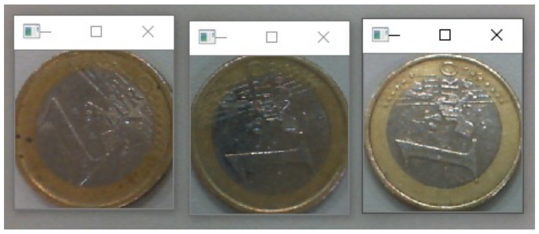


Fig. 7. Final cropped coin images

B. *Classification*

To create our classifier, we chose to create a convolutional neural network based on the VGG16 CNN architecture mentioned earlier in the document. This specific CNN is different to many others in that the kernel size for the filters are very small (3x3). We label the training data using ImageDataGenerator so the images are split into 6 classes (2euro, 1euro, 50cent, 20cent, 10cent, 5cent). Our implementation uses the Sequential method so that the layers of the model will be arranged in sequence. We import the pre-trained VGG16 model to use as our base. This model has been trained on millions of images from the ImageNet data set.

To refine the model for our use, we use a technique called Transfer Learning. The idea of transfer learning is instead of training a deep network from scratch for your task, you take a network trained on a different domain for a different source task and then adapt it for your domain and target task [4]. To apply this technique, we use the pre-trained base model and remove its final layers. This allows us to use the base model to extract features for us but we will next specify what we want our final layers to be. We have chosen to replace the final layers with eight new layers. We also apply transfer learning by fine-tuning the model. This means that we selectively retrain certain layers of the network. For our model, we freeze the all the layers apart from the last seven.

During training, an optimiser needs to be applied. We use the Adam (Adaptive Movement Estimation) optimiser in order to reach the global minima [5]. It is useful if the model gets stuck at local minima as the optimiser should get it out and reach global minima. The learning rate for the optimiser has been set to 0.0001 but this can be adjusted. The learning rate compromises between speed of learning and the accuracy of the result.

The model trains in several stages called epochs. For the initial train, it performs just one epoch. Then all the layers are unfrozen so they can now be adjusted. We have set the second training part to three epochs. While training, the model evaluates its validation accuracy and will only save the model if the validation accuracy improves from the last epoch.

To now use the model, it is exported as a json file to then be imported into the original data acquisition program. During runtime, the same process as used for the data collection will occur but instead of just saving the images to the computer, they will then be used in the neural network's predict function which will return the probabilities for each coin type specifically for each cropped coin image. For visualisation, boxes are drawn around the coins with their predicted type and probability label above.

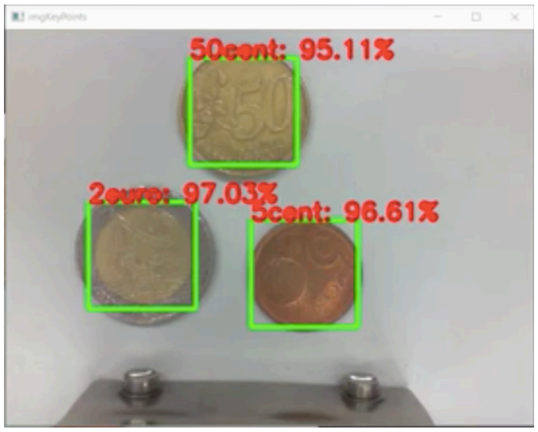


Fig. 8. Final output with visual labelling and predictions

VI. Testing

Testing during data acquisition was done constantly and did not have a specific method. This was due to our ability to visually observe the accuracy of the program. The adjustable value sliders allowed us to optimise the accuracy and we could receive real-time responsive feedback on the effect of adjusting each value. Initially, we had to adjust these values before running the program and had to restart it every time we needed to change one value.

For the CNN, we evaluated its performance by forming a confusion matrix. 30% of our dataset was used for model evaluation while 70% went to the actually training the model.

From this confusion matrix, we can see that the model has

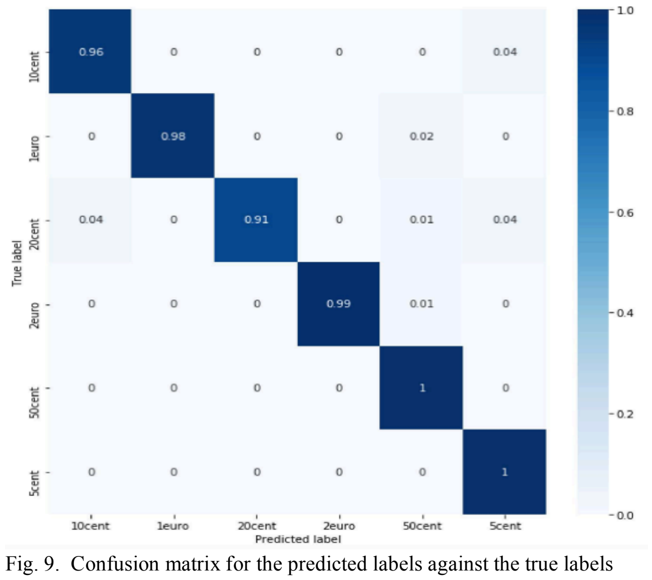


Fig. 9. Confusion matrix for the predicted labels against the true labels

very high accuracy when tested against 30% of the dataset. However, despite it having high accuracy on our dataset, when performing as a real-time classifier, it did not perform as accurately. This is due to not having a large enough data set that represents a range of conditions the camera can be in, for example, high lighting or different background.

Classifier development did not require large amounts of adjustment in comparison to the data acquisition part. More time was needed in developing and training the model. If we wanted to make significant adjustments such as layer changes, it would require retraining the whole model again which each time takes around 20 minutes. As this is a workable prototype, we felt that testing, other than accuracy evaluation which is done during training, was not necessary or would not contribute significantly.

The system overall performs as expected and does achieve all initial system requirements. Evaluating whether it has achieved most of these requirements is simply done by running the system and observing for yourself. Because the system is only being used in a controlled environment, we expected to have high accuracy and functionality.

VII. Evaluation

Overall, the system works as intended and designed. However, there are areas for improvement and consideration. There are also areas for potential development if wanting to extend this system into a commercial product.

A. Safety Considerations

An issue to be considered for all neural networks is that they can be fooled into producing incorrect predictions. As we are dealing with images specifically for our system, methods such as placing stickers within the image can alter how it is interpreted. At the moment, this does not pose a large issue for our system. However, if our system was to be implemented in software for a bank to use, as suggested earlier in the document, this could potentially lead to inaccurate counting of money. This would have repercussions for both customer and bank. This could be potentially exploited on a high level if this system was relied on heavily.

B. Possible Improvements

An improvement that could be made is to simply increase the amount of training data for the CNN. Due to time limitations, only a couple thousand images could be collected for training. This is a relatively small amount for neural network training. Ideally, we would have wanted numbers in the tens or hundreds of thousands. As a group this wasn't feasible but if it were possible, it would most likely increase the accuracy of the network. Additionally, collecting varied data such as images with low lighting, high lighting, different backgrounds etc. would also improve the model's performance.

Another improvement would be to extend the system to classify both coin faces. Currently, a limitation of the system is that it is only able to classify 'head' faces of coins. We chose to limit the data collection to only one side as it would decrease the variation in features that the CNN needed to identify. Also, it would increase the amount of data needed to be collected. However, if more time permitted, the system could be extended to allow this.

An improvement in how the user interacts with the system could be made. At the moment, it is not user friendly for the average person. It would only be usable by those with knowledge in programming as there is very limited to no user interface. Many adjustments that need to be made can only be done by modifying the code such as when changing the program to whether or not you calibrate it by capturing a new background image. This could be improved by providing a graphical user interface that implements the system and allows for easier adjustment where needed at user level.

A physical improvement of the components would be to use a higher resolution camera. With a better camera, data collection rate would increase because more coins can fit in one image with sufficient detail. At the moment, the program is capable of only fitting around four coins in frame for them to be accurately detected.

VIII. Conclusion

As a team, we are very satisfied with what we have created in the time given and resources provided. Delving into the field of deep learning was a complex but rewarding. Through a lot of research, we were able to build this with little to no assistance. Also, by combining knowledge gained from the course such as image processing techniques for our data acquisition and knowledge gained from outside the course, we refined and built on existing skills. This project definitely tested our abilities! All files including our dataset can be accessed on GitHub [6].

References

[1] K. Simonyan and A. Zisserman "Very deep convolutional neural network for networks for large-scale image recognition" Visual Geometry Group, Department of Engineering Science, Univeristy of Oxford, arXiv:1409.1556, 2015

[2] M. U. Hassan (2018, November). *VGG16 – Convolutional network for classification and detection* [Online]. Available: https://neurohive.io/en/popular-networks/vgg16/

[3] N. Silvis-Cividjian, *Pervasive Computing Engineering Smart Systems*, 2016, ch. 4

[4] D. Sarkar (2018, November). *A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning* [Online]. Available: https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

[5] P. Skalski (2018, November) *How to train Neural Network faster with optimizers?* [Online]. Available: https://towardsdatascience.com/how-to-train-neural-network-faster-with-optimizers-d297730b3713

[6] T. Blom, Physical Computing repository [Online]. Available: https://github.com/thim0o/CoinDetector