PAUL BOULENGER, SEBASTIEN CHAHOUD, MARINE MOUTARLIER
GROUP 9
NORMALBGS

31 MAI 2024

COINS RETRIVAL PROJECT

OVER-VIEW

ON THE TRAIN SET:

- BACKGROUND DETECTION
- NEUTRAL COIN EXTRACTION
- NOISY COIN EXTRACTION
- HAND COIN EXTRACTION
- COINS LABELLING APP
- COINS CLASSIFICATION

ON THE TEST SET:

- INFERENCE OF THE TEST SET

BACKGROUND CLASSIFICATION

We will use the following features:
- **Mean color** the average color of the image
- **Standard deviation of the color:** the standard deviation of the color of the image
- **Skewness of the color**: the skewness of the color of the image

We will train a SVM classifier to detect the background of the image based on the color features.

```python
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Split the dataset into training and test sets with stratification to keep the same class distribution
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train the classifier
classifier = SVC()
classifier.fit(X_train, y_train)
train_accuracy = classifier.score(X_train, y_train)
print(f'Accuracy on the training set: {train_accuracy:%}')

# Evaluate the classifier
eval_accuracy = classifier.score(X_test, y_test)
print(f'Accuracy on the test set: {eval_accuracy:%}')
```

Accuracy on the training set: 100%
Accuracy on the test set: 100%

COIN EXTRACTION

Separated for noisy, neutral and hand

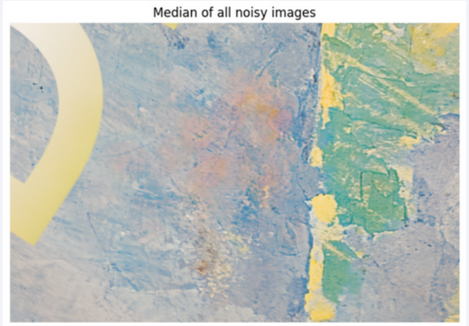
NEUTRAL

- Computing average and median of all neutral
- Subtract the background
- Do a color threshold.
- Apply a few transformations.
- Do Hough circle detection on the final image

Median of the neutral

NOISY

- Computing average and median of all noisy
- Subtract the background
- Do a color threshold.
- Apply a few transformations.
- Do Hough circle detection on the final image



Median of the noisy

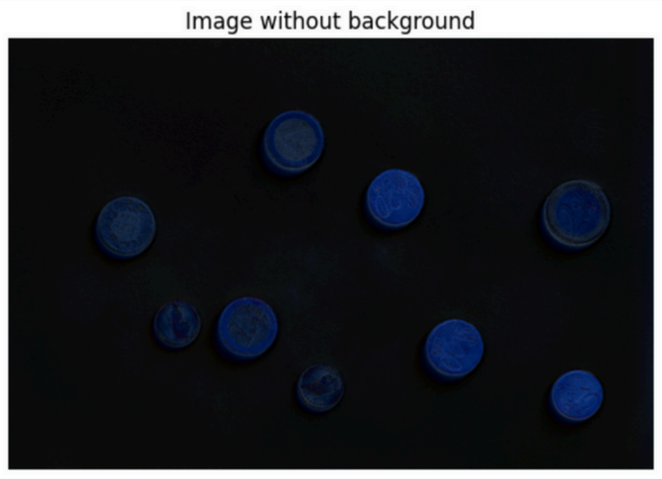
HAND

- Apply a few transformations.
- Do Hough circle detection on the final imagee

→ no average or median as it looked like this

NEUTRAL COIN EXTRACTION

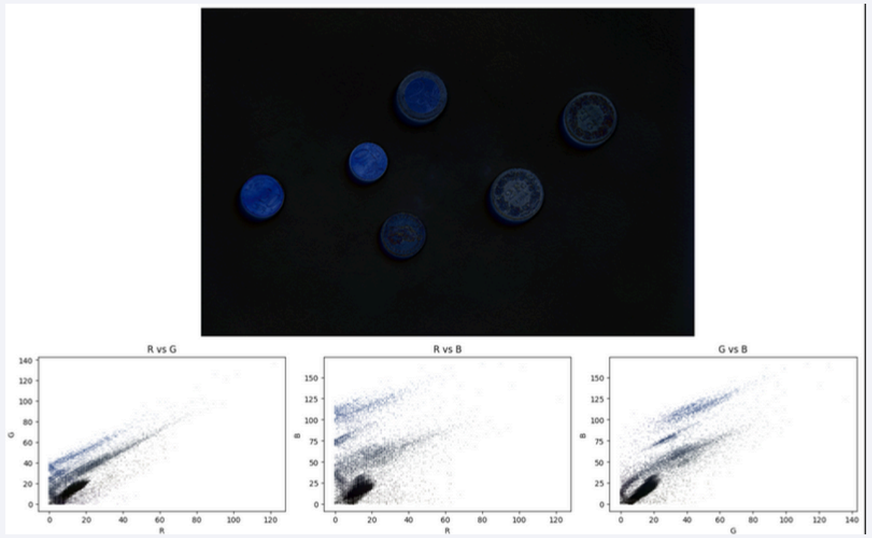
- Computing average and median of all neutral
- Subtract the background
- Do a color threshold.
- Apply a few transformations.
- Do Hough circle detection on the final image



Without background image

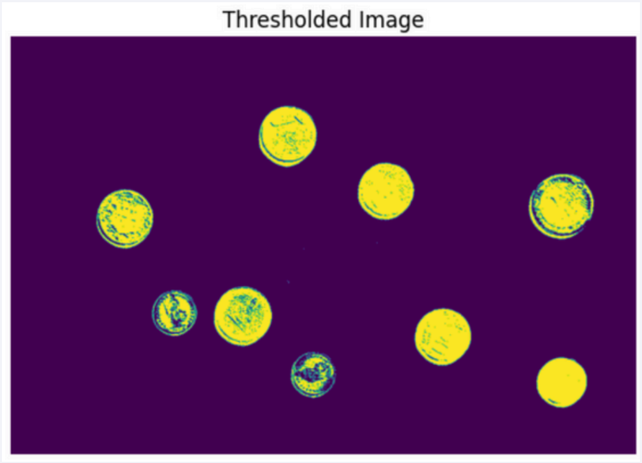
Color threshold

Since the coins have a differrent color than the neutral background, we can easily do a color threshold. We plot the rgb ditribution here to determine manually this threshold.




```
negative_rgb_neutral_threshold = ThresholdRGB(
    min_red=0,
    max_red=255,
    min_green=0,
    max_green=255,
    min_blue=0,
    max_blue=45,
    type="-"
    )
```

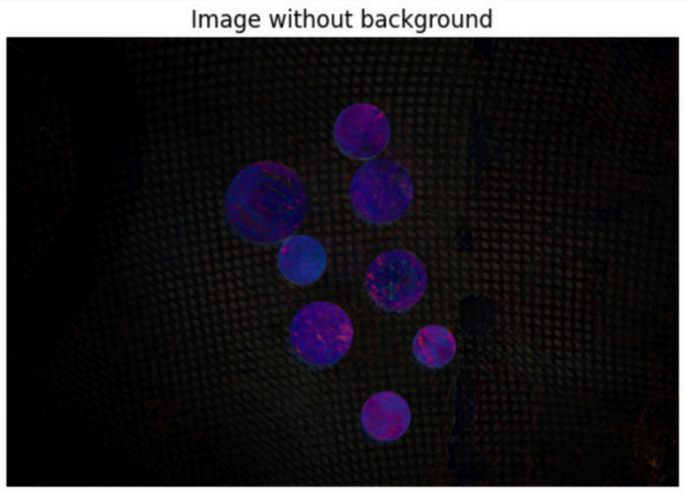
Color threshold



From this we can very well apply the hough circle

NOiSY COIN EXTRACTION

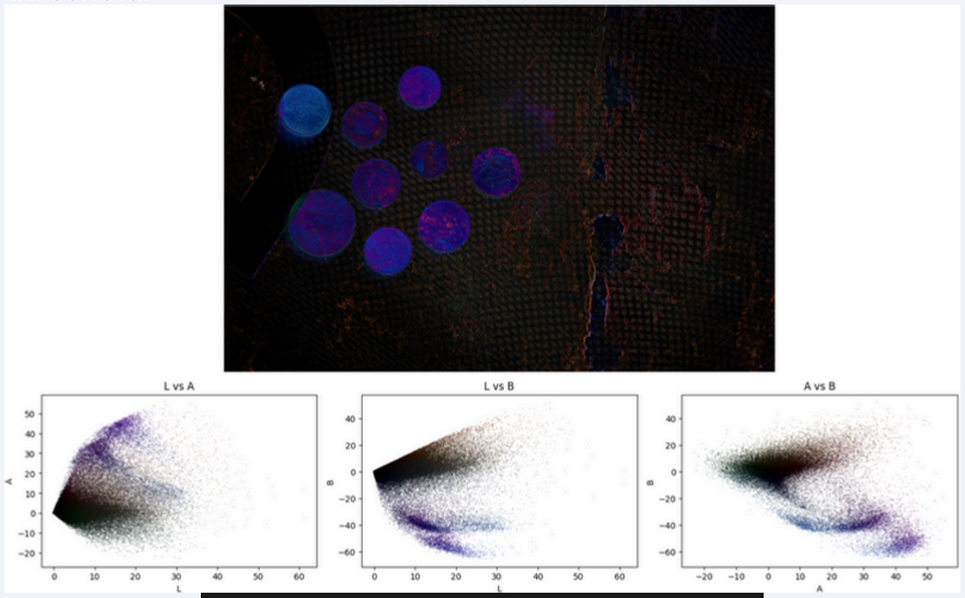
- Computing average and median of all neutral
- Subtract the background
- Do a color threshold.
- Apply a few transformations.
- Do Hough circle detection on the final image



Without background image

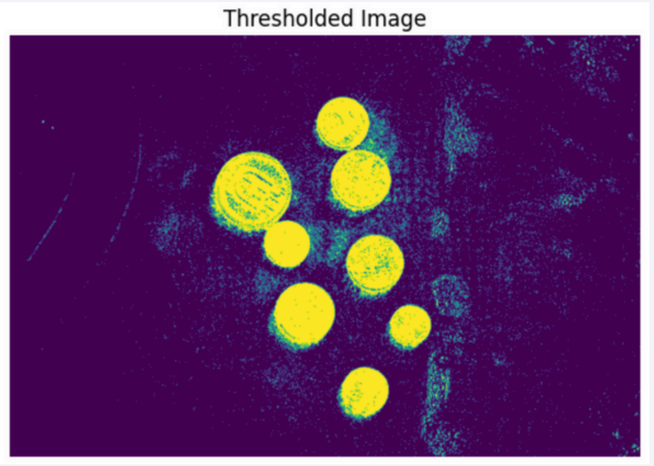
Color threshold

Since the coins have somehow a similar color than the noisy background, we can do a color threshold, but a lab one. We plot the lab distribution here to determine manually this threshold.




```
negative_lab_noisy_threshold = ThresholdLAB(
    min_l = 0,
    max_l = 27,
    min_a = -100,
    max_a = 100,
    min_b = -5,
    max_b = 23,
    type='-'
    )
```

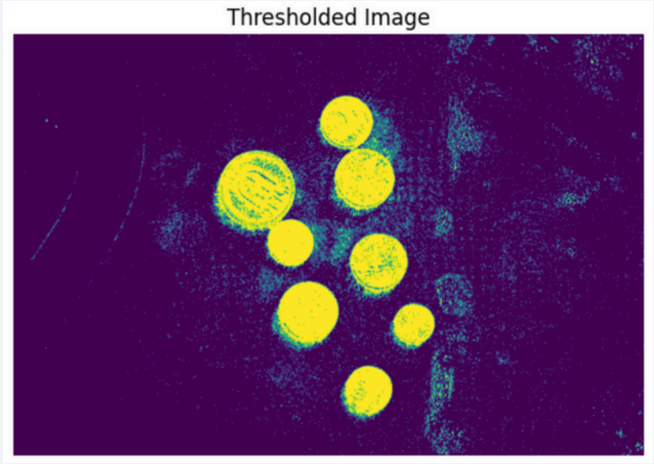
Color threshold



From this we can very well apply the hough circle

NOiSY COIN EXTRACTION

- Computing average and median of all neutral
- Subtract the background
- Do a color threshold.
- Apply a few transformations.
- Do Hough circle detection on the final image



Color threshold

Apply transformation

We apply closing since the coins are not full, as well as a removing of objects since some of the background is picked up. This will help with the further Hough detection on the coins. It might not be perfect if the coins are not fully circular.

From this we can very well apply the Hough circle

```python
def noisy_img_transform(img):
    processed_img = img.copy()

    processed_img = remove_objects(processed_img, size = 500)
    processed_img = apply_closing(processed_img, 7)

    return processed_img.astype(np.uint8)
```

HAND COIN EXTRACTION

```python
for img in hand_bg_imgs:
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    blurred = cv.GaussianBlur(gray, (15, 15), 0)
    thresh = cv.adaptiveThreshold(blurred, 255,
                                  cv.ADAPTIVE_THRESH_GAUSSIAN_C,
                                  cv.THRESH_BINARY_INV, 11, 3)
    kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))
    morphed = cv.morphologyEx(thresh, cv.MORPH_CLOSE, kernel, iterations=5)
    edges = cv.Canny(morphed, 50, 150)
```

- Apply a few transformations.
- Do Hough circle detection on the final imagee

Blurred

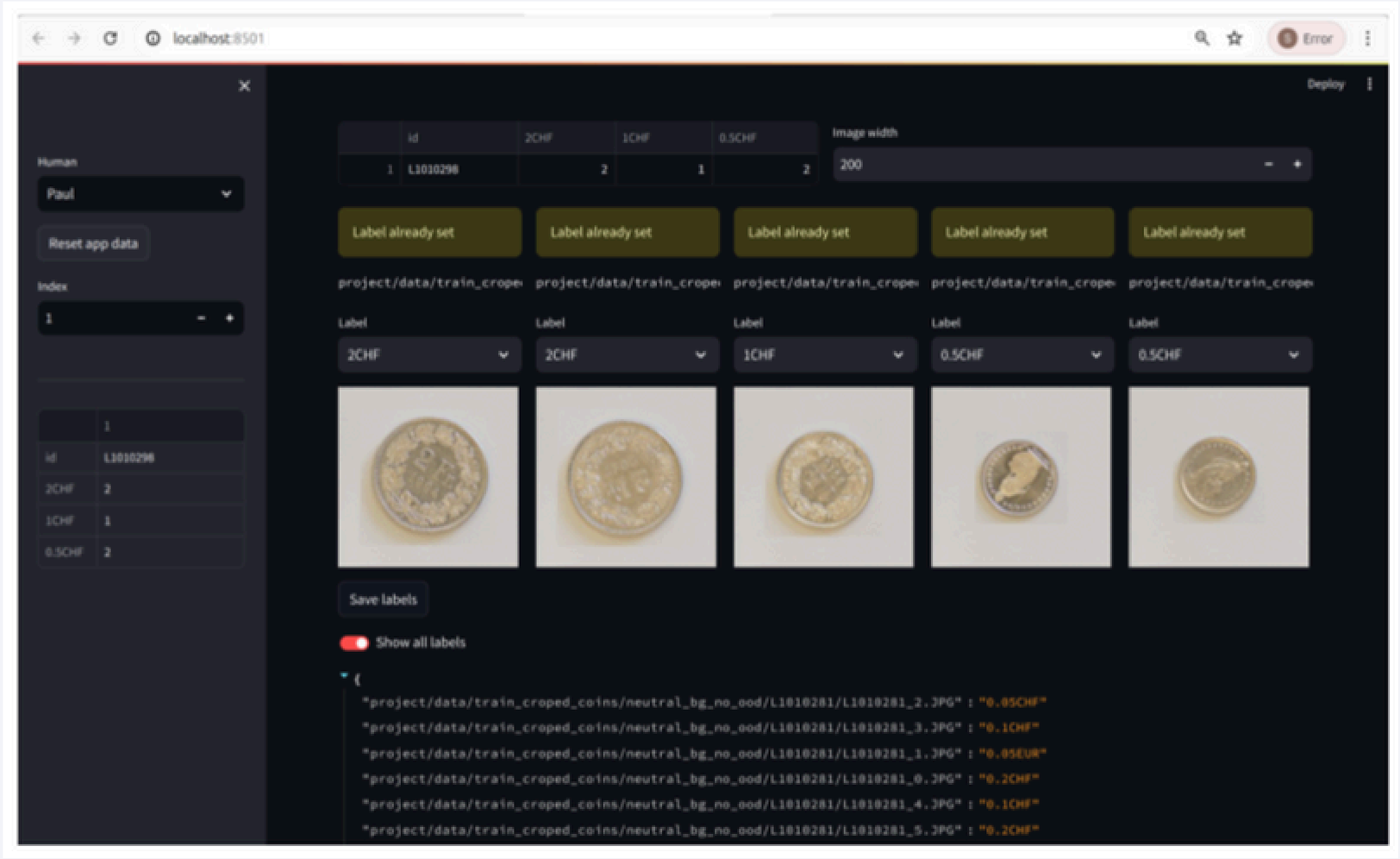
Adaptive threshold

Morphed

From this we can very well apply the Hough circle

COIN LABELING APP

In order to train a classifier to classify the coins, we need to label the coins in the train set. We will create a simple app to label the coins. This app allows for more efficient labelling by splitting the work load among multiple users.



COIN CLASSIFICATION

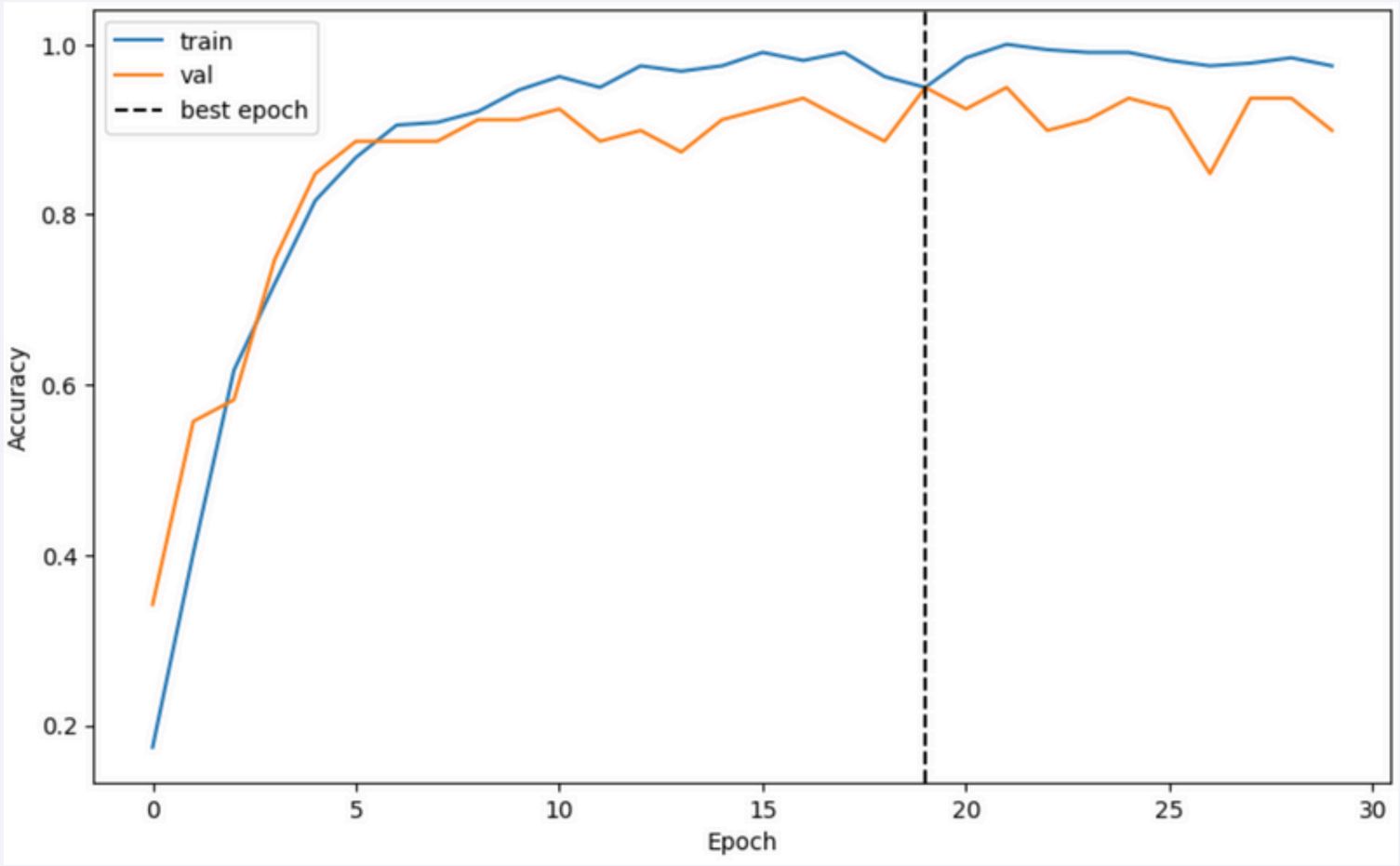
For the classifier training, we will augment the data by applying simple transformations to the images. This will help the classifier to generalize better to new data. The transformations we will apply are:
- **Rotation** we will rotate the images by a random angle between -180 and 180 degrees
- **Jitter**: we will add a small amount of noise to the images

We split the data into a training and validation set with startified sampling to ensure that the classes are balanced in both sets.

```
train_transform = transforms.Compose([
    transforms.Resize(224),  # Resize to a smaller size for faster processing
    transforms.RandomRotation(180),  # Data augmentation
    transforms.ColorJitter(brightness=0.5, contrast=0.5, saturation=0.4, hue=0.3),
    transforms.ToTensor(),           # Convert image to PyTorch tensor
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  # Normalize
])
```

COIN CLASSIFICATION

To classify the coins, we will use an **efficientnet** model. Efficientnet models are state-of-the-art models that are efficient and accurate. We will use the efficientnet-v2-s model, which is a small model that is suitable for this task.
We start from the pre-trained weights obtained from pre-training on the imagenet-1k dataset, which contains a large number of images from different classes. We will fine-tune the model on the coin dataset.



COIN CLASSIFICATION

We evaluate the model on the validation set.

```
Accuracy of the network on the validation set: 94 %
```

	accuracy	class_proportion
0.01EUR	100.0	4.55%
0.02EUR	100.0	5.88%
Not a coin	100.0	3.21%
5CHF	100.0	6.42%
2EUR	100.0	8.56%
2CHF	100.0	5.35%
1EUR	100.0	2.67%
0.5EUR	100.0	7.49%
OOD	100.0	13.10%
0.2EUR	100.0	7.75%
0.2CHF	100.0	2.94%
0.1EUR	100.0	5.61%
0.05EUR	100.0	6.95%
0.05CHF	100.0	3.48%
0.1CHF	75.0	4.81%
0.5CHF	75.0	5.35%
1CHF	50.0	5.88%

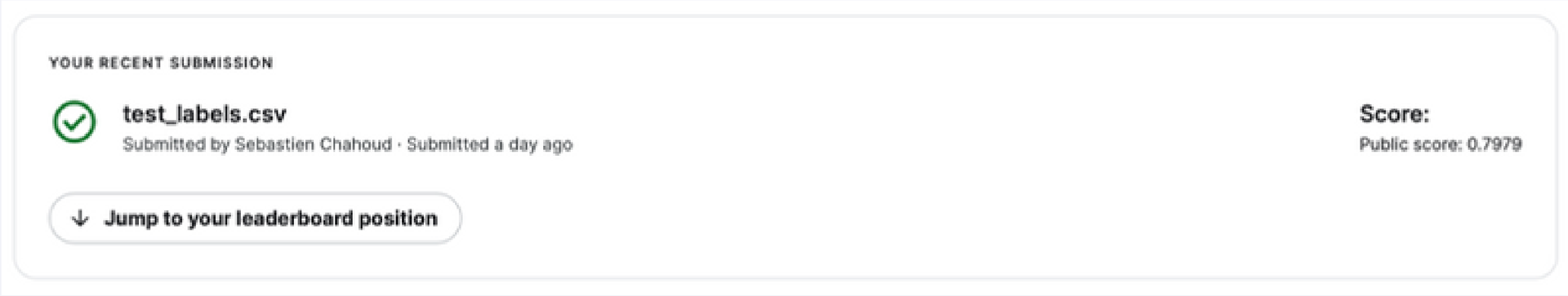
We observe that the model has a high accuracy on the validation set (94%), which indicates that it is able to generalize well to new data.
Even tough our classes are unballanced, we do not observe a performance drop on minority classes.
Hence we decided not to apply any class balancing techniques.

TEST SET INFERENCE

- To detect the coin on each image, we will apply a segmentation process. This segmentation process is distinct for each background type. Hence we first need to detect the background of each image.
- We store the results into different folder for the coin extraction
- We perform the corresponding coin extraction based on the background
- We store the results into different folder for the coin classification
- We perform the classification on those 17 classes:

{'2CHF': 0, '2EUR': 1, '0.5CHF': 2, '0.5EUR': 3, '0.2EUR': 4, '0.1EUR': 5, '0.02EUR': 6, 'Not a coin': 7, '1CHF': 8, '5CHF': 9, '0.01EUR': 10, '0.05CHF': 11, '0.1CHF': 12, '0.2CHF': 13, '0.05EUR': 14, 'OOD': 15, '1EUR': 16} Number of classes: 17

- Store the result into a csv file
- Submit to Kaggle!



CONCLUSION

We successfully developed a robust system capable of accurately identifying and classifying coins.

This project showed the importance of combining theoretical knowledge with practical application. The ability to accurately detect and classify coins is a daily helpful task.

Team Collaboration:
We had a great team! Each member contributed their expertise to solve various challenges.

Future Directions:
- **Enhancing Robustness** Incorporating additional features to improve the model's robustness against varying lighting conditions and coin wear.
- **Extended Applications**: Exploring the application of our coin detection framework to other areas such as counterfeit detection and foreign currency classification.