Bangkok University

Internship
Rapport

Bakery recognition by neural network

Student :
Arthur Gautier

Teacher :
Audrey Minghelli

Mentor :
Pakorn Ubolkosold

August 8, 2023

Contents

Recommendation

I express my gratitude to Bangkok University for welcoming me to their premises during my internship. I extend my thanks to Dr. Pakorn Ubolkosold, my mentor, for providing me with insightful guidance and clear explanations about my project. I am also thankful to Audrey Minghelli for her availability and attentive responses to my inquiries.

Furthermore, I am truly appreciative of the warm reception, patience, and assistance offered by the colleagues I had the pleasure of interacting with throughout this assignment. The various interactions and the work undertaken during my time at Bangkok University have provided me with all the essential elements needed to compose this report.

1 Introduction

Bangkok University (BU) is one of the oldest and largest private universities in Thailand. It was founded in 1962. BU has a strong reputation for academic excellence and offers a wide range of undergraduate and graduate programs. For this internship I was in the engineering Department of the university. It's one of the most prestigious in Thailand. It offers a variety of programs, including civil engineering, mechanical engineering, electrical engineering, and computer engineering. The department is well-equipped with modern laboratories and facilities, and its faculty are highly experienced and qualified.

Figure 1: Bangkok university campus

The objective of my internship was to develop an autonomous artificial intelligence capable of effectively scanning and processing customer orders, thereby facilitating a self-sustained payment service tailored to the needs of local bakeries in Bangkok. I proceed with several steps, encompassing a comprehensive understanding of neural networks and convolutional neural networks (CNNs), the establishment of a robust database, training of the AI using data, and the creation of a user-friendly application and integrated device for the final solution. This report presents a detailed account of the entire development process, including the theoretical foundations, practical challenges encountered, methodologies employed, until the realization of the project purpose.

2 MNIST dataset

2.1 Neural network

To begin, I needed to acquaint myself with neural network techniques. Instead of using my own dataset initially for testing, I opted to start with the MNIST Handwritten Digit dataset. It's a collection of a large number of handwritten digits (0-9) that have been labeled for classification. All the images are in a same size 28 by 28 pixels. The goal is to develop a neural network model that can accurately classify these digits based on their images.

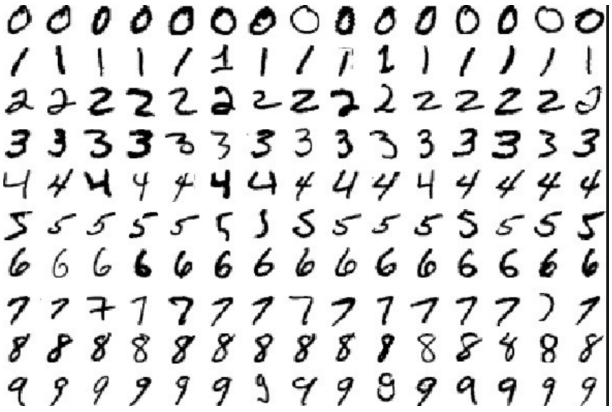

Figure 2: Examples of digit images from the MNIST dataset

2.1.1 Importations

In order to be able to build our model, we need to import all the required modules.

- **Numpy** to be able to compute matrix
- **Matplotlib** to print figures
- **Keras** to build our neural network

I also imported **os** to correct a bug I had with my computer. Our imports then take the form:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from keras.layers import Dense, Flatten
4 from keras.models import Sequential
5 from keras.utils import to_categorical
6 from keras.datasets import mnist
7 import os
8 os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"
```

Then, we need to split our dataset into a training and a test part in order to build the network. The train and test data contain respectively 60000 and 10000 image.

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In the code above, X is the image and y the label of the image.

2.1.2 Data pre-processing

All the labels are represented as a single number or it's not what is expected for a Neural Network : we need to turn each label into one-hot representation to be able to train our model. To do that we'll use *to_ categorial()* function from Keras module. This function is going to create an 1D array fill with 0 with the number of label length except at the index of the label. For example :

```
to_categorical(3, num_classes=10)
```

will create an array of length 10 fill with 0 except on index 3. This representation is called the one-hot encoding. Thus, we have to do this on every label of our train and test data.

```
# Convert y_train into one-hot format
temp = []
for i in range(len(y_train)):
    temp.append(to_categorical(y_train[i], num_classes=10))
y_train = np.array(temp)
# Convert y_test into one-hot format
temp = []
for i in range(len(y_test)):
    temp.append(to_categorical(y_test[i], num_classes=10))
y_test = np.array(temp)
```

2.1.3 Building the model

Now we can start to create the Neural Network with Keras. We start by initialize a sequential model with a flatten layer : we need to reshape our 28 by 28 pixels images (2-dimensions) into a $28 \times 28 = 784$ values (1-dimension).The Sequential model type is probably the easiest way to build a model in Keras as it enables building a model layer by layer.

Then, we connect these values into, for example, 5 neurons with sigmoid ativation function. Defined as below :

$$\forall x \in \mathbb{R}, f(x) = \frac{1}{1 + e^{-x}}$$

We use this function because it's derivable and its codomain is $[0, 1]$ which provide values similar to probability.

To finish, we add another layer which acts as our output. We use the Softmax activation function here. Softmax makes the output sum up to 1 so the output can be interpreted as probabilities. We use 10 neurons in this last layer because our classification task have 10 classes.

```
# Create simple Neural Network model
model = Sequential()
model.add(Flatten(input_shape=(28,28)))
model.add(Dense(5, activation='sigmoid'))
model.add(Dense(10, activation='softmax'))
```

We can now observe the architecture we created with the *model.summary()* function.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 5)                 3925

 dense_1 (Dense)             (None, 10)                60

=================================================================
Total params: 3,985
Trainable params: 3,985
Non-trainable params: 0
_________________________________________________________________
```

Figure 3: Model summary

2.1.4 Compiling the model

Now we just need to compile our model :

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['acc'])
```

The loss function used here is *categorical_ crossentropy* it's one of the best in multiclass classification problem. Same thing with Adam for the optimizer ; the optimizer controls the learning rate and adjusts it throughout training. Lastly, we have accuracy to be passed in mertrics argument in order to measure the performance of our classifier. Now that our model is built, we can now train it.

2.1.5 Training the model

```
model.fit(X_train, y_train, epochs=5, batch_size=16,
      validation_data=(X_test,y_test))
```

```
Epoch 1/5
1875/1875 [==============================] - 3s 2ms/step - loss: 1.6664 - acc: 0.5099 - val_loss: 1.2542 - val_acc: 0.7271
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 1.1286 - acc: 0.7074 - val_loss: 1.0142 - val_acc: 0.7520
Epoch 3/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.9548 - acc: 0.7522 - val_loss: 0.9583 - val_acc: 0.7106
Epoch 4/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.8837 - acc: 0.7669 - val_loss: 0.8647 - val_acc: 0.7792
Epoch 5/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.8447 - acc: 0.7709 - val_loss: 0.7953 - val_acc: 0.7861
313/313 [==============================] - 1s 1ms/step
```

Figure 4: Training

As we can see, after each epoch the model gets more and more precise with a final result of 77% of accuracy (for 5 epoch ; The epoch is a hyperparameter of gradient descent that controls the number of complete passes through the training dataset.). We can also play with the batch size in our parameters : it's a hyperparameter of gradient descent that controls the number of training samples to work through before the model's internal parameters are updated. We can then try to perform predictions on several images stored in our X_test variable.

Let's draw the accuracy and loss curves during the training

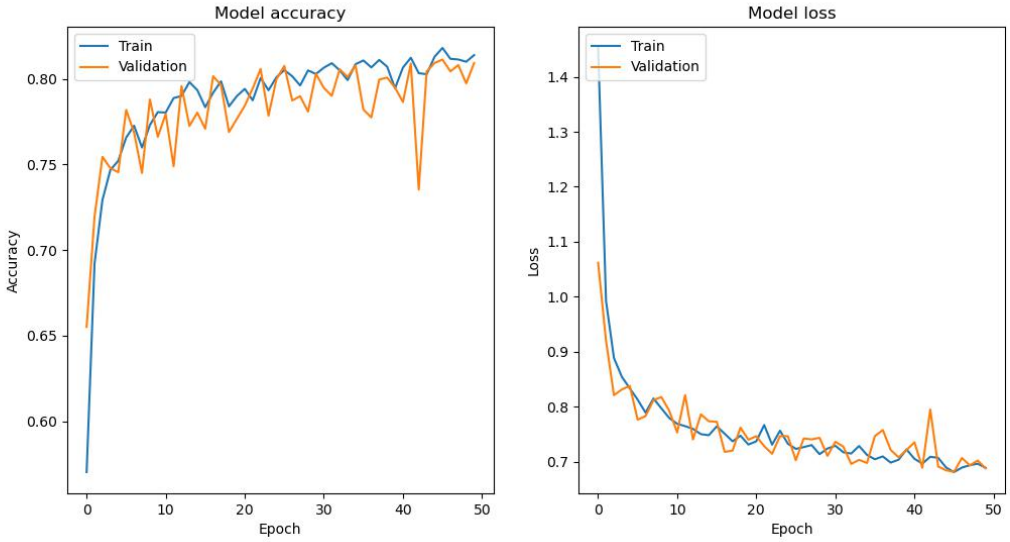

Figure 5: Accuracy and Loss curves for 50 epochs

As we can see, the loss curve is a bit high, it can be explained by the few neurons used in our layers.

2.1.6 Using our model to make predictions

```
predictions = model.predict(X_test)
predictions = np.argmax(predictions, axis=1)
print(predictions)
```

```
fig, axes = plt.subplots(ncols=10, sharex=False,
                         sharey=True, figsize=(20, 4))
for i in range(10):
    axes[i].set_title(predictions[i])
    axes[i].imshow(X_test[i], cmap='gray')
    axes[i].get_xaxis().set_visible(False)
    axes[i].get_yaxis().set_visible(False)
plt.show()
```

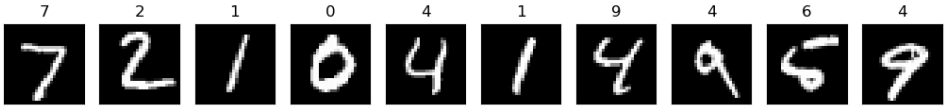

Figure 6: Test result

As we can see, the predictions is pretty great, the model gave a false result for some figures. To have a deeper view of the quality of our model we can import the module *confusion_matrix* from *sklearn.metrics* and use the function as follow :

```
# Obtain the confusion matrix
cm = confusion_matrix(y_test.argmax(axis=1), predictions)
print(cm)
```

We get this matrix as a result :

```
[[ 908    0    5    7    6   39    0    1   14    0]
 [   0 1097    1    5    1    0    8    0   23    0]
 [ 144   45  727   21   13    2   27    2   45    6]
 [  41   37  100  772    0   14   26    4    7    9]
 [   2    4   12    0  798    3   25    7   11  120]
 [  67    5    6   85   26  535   25    9  132    2]
 [  30    5   63    5   10    6  706    0  132    1]
 [   8   52    9   24   17    1    2  800    9  106]
 [  70   49   24   10   13   22   66    5  694   21]
 [  24   14    2    7   62    3    3   64    6  824]]
```

Figure 7: Confusion matrix

In the confusion matrix, each row represents the true class of the data samples, while each column represents the predicted class by the model. The diagonal elements of the matrix correspond to the correctly predicted samples, showing high values in this case due to the strong diagonal pattern. This indicates that the model's predictions align closely with the actual ground truth, demonstrating good accuracy across different classes. As we can see, the 9 are often recognize as a 4 which is relevant on our example above.

2.2 Deep learning

Now that we understand how a neural network works and how to build it, let's look at deep learning networks. To experiment deep learning we are going to build a convolutions neural network (CNN) in order to classify images.

2.2.1 Importations

As previously, in order to be able to build our model, we need to import all the required modules.

- **Numpy** to be able to compute matrix
- **Matplotlib** to print figures
- **Keras** to build our neural network
- **Sklearn.metrics** to have the confusion matrix of our training model

We will use the MNIST database imported from Keras as before.

2.2.2 Data pre-processing

To start we need to format our input data $(X_{\text{train}}, X_{\text{test}})$ to the shape expected by our model. Thus, after importing the data as previously, we need to write :

```
#reshape data to fit model
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)
```

We indicate here that X_{train} and X_{test} have respectively 60000 and 10000 28 by 28 images in grey scale (specified by the 1 at the end). Then, we must ensure that the labels are in the one-hot representation form using the *to_ categorical()* function from Keras.

```
#one-hot encode target column
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

2.2.3 Building our model

We will use again the Sequential model type. Our first two layers are convolutional layers. They are going to deal with our input images seens as 2-dimensional matrices. The 64 and 32 refers to the number cof nodes in each layer. The Karnel size refers to the dimension of our convolution matrix for the filtering. So here convolution matrix is dimension 3. The activation function used in these two first layers is the ReLU function, defined as below :

$$\forall x \in \mathbb{R}, f(x) = max(0, x)$$

In order to connect our convolution layers to our dense, output layer, we need to add a flatten layer between these two. The final dense layer needs 10 nodes as before, one for each possible outcome (0,9) - we continue to use the softmax function.

```
#create model
model = Sequential()
#add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 64)        640

 conv2d_1 (Conv2D)           (None, 24, 24, 32)        18464

 flatten (Flatten)           (None, 18432)             0

 dense (Dense)               (None, 10)                184330

=================================================================
```

Figure 8: Model summary

2.2.4 Compiling the model

Now we just need to compile our model. We use the same parameters as before. Check the previous section for more explanations.

```
#Compile model using accuracy to measure model performance
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

2.2.5 Training the model

We have now our model ready to be trained. We processed as before with 3 epochs and a batch size of 16.

```
#Train the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test),
                    epochs=3,
                    batch_size=16)
```

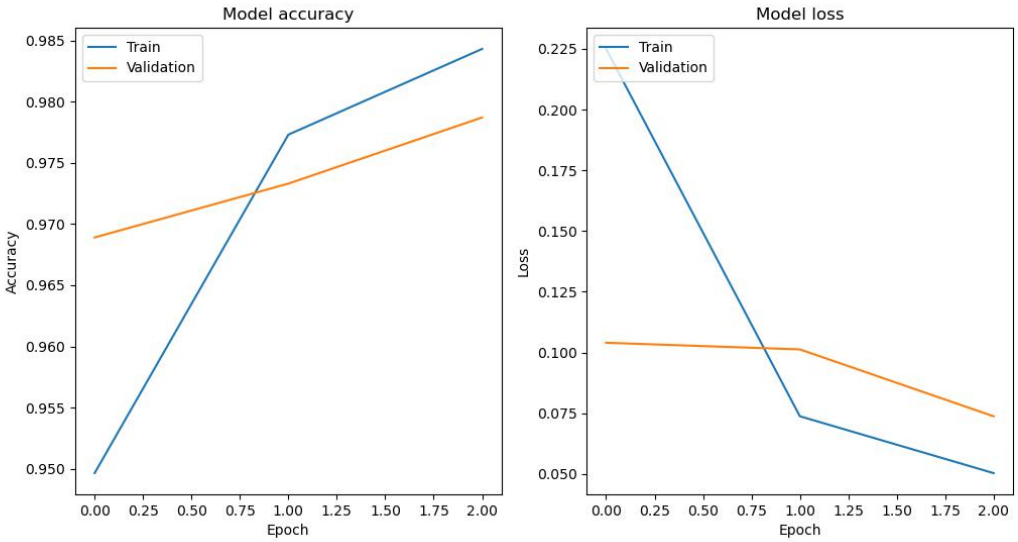

Figure 9: Accuracy and loss curves for 3 epochs

As we can see we reach a good accuracy with only 3 epochs, an accuracy around 98%. I also tried to train the model with a more epochs, we reach quickly the over-fitting with this model : the loss curve based on validation set increase after each epochs.

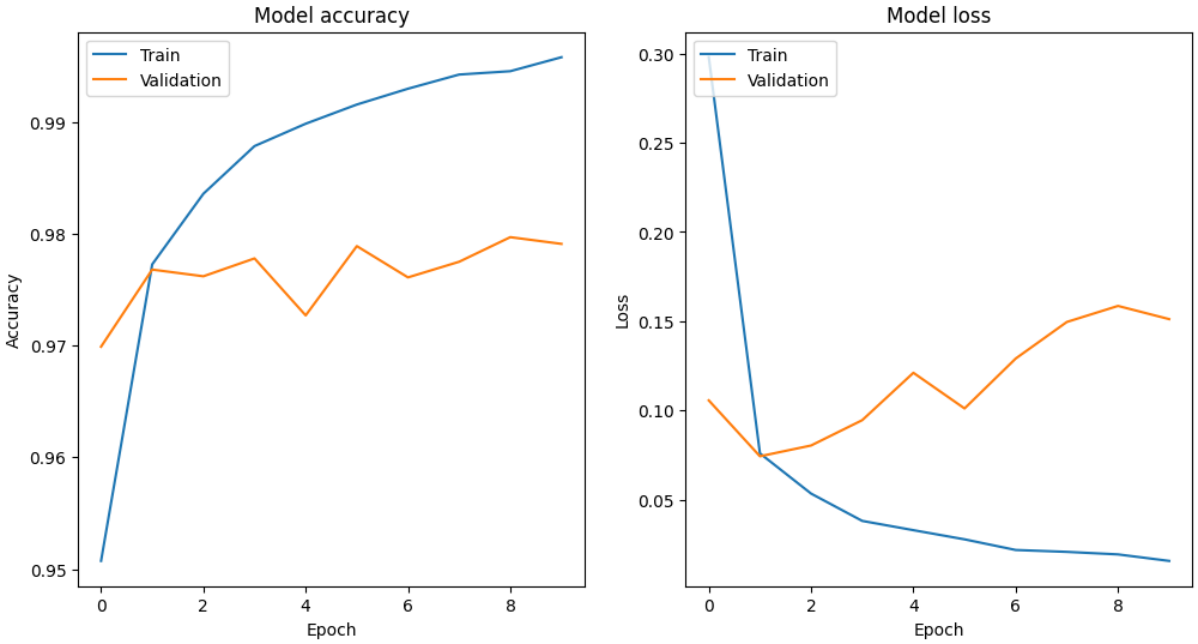

Figure 10: Accuracy and loss curves for 10 epochs

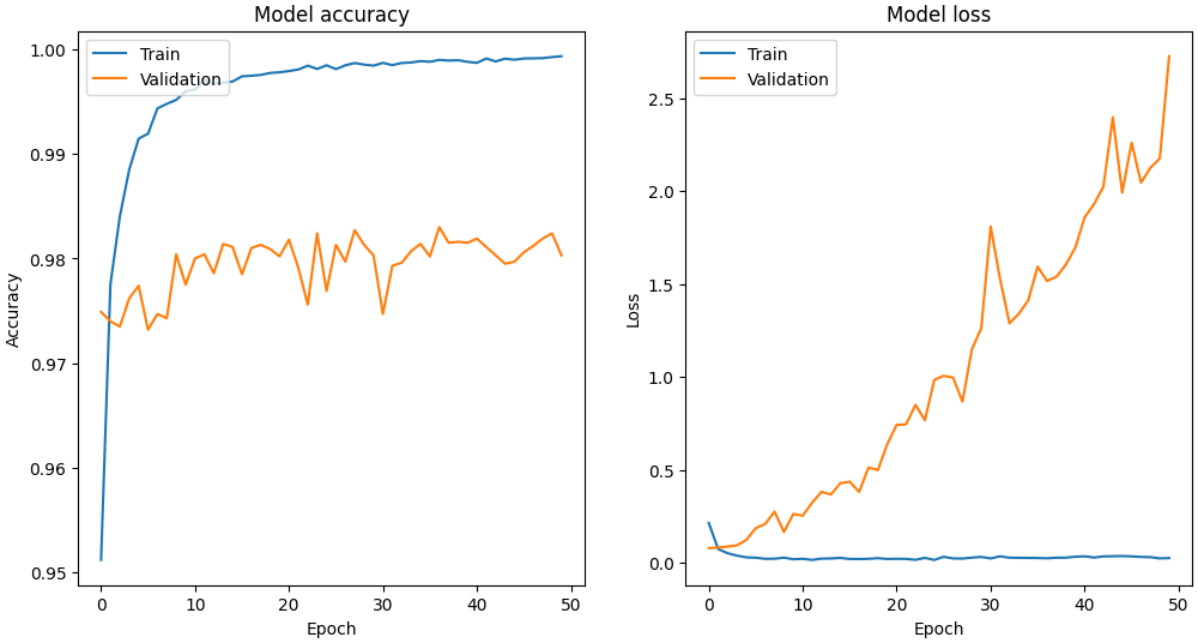

Figure 11: Accuracy and loss curves for 50 epochs

2.2.6 Using our model to make predictions

We can now try to use our trained model with images of our test dataset. Let's use the same pictures as the previous section. As we can see on these 10 images, the model makes no mistakes.

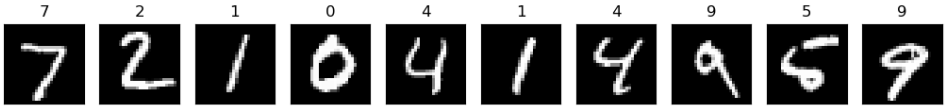

Figure 12: Test result

For more serious results we can look at the confusion matrix below, based on the test set.

```
[[ 972    0    3    0    0    0    2    1    2    0]
 [   2 1116    2    5    0    1    3    0    6    0]
 [   2    1 1010    3    2    0    1    8    5    0]
 [   0    0    0 1002    0    4    0    0    3    1]
 [   1    2    0    0  967    0    5    1    0    6]
 [   2    0    0   14    1  863    3    0    8    1]
 [   5    3    0    0    2    0  941    0    7    0]
 [   0    0   10    5    4    0    0  994    3   12]
 [   3    3    5    2    2    5    1    1  948    4]
 [   2    5    0    6    6    5    0    0   11  974]]
```

Figure 13: Confusion matrix

It can therefore be seen that deep networks are much more efficient than simple networks. They require less epochs and give better results. They are nevertheless more complex in their architecture and integrate many more parameters than conventional neural networks.

2.3 Transfer learning

Now, we are going to explore another method for image classification. This method is called Transfer Learning; it involves training a model on a new task based on previously learned tasks (as shown in the right picture below). Using this method, the learning process can be faster, more accurate, and may require less training data. In contrast, with traditional machine learning (as shown in the left picture), the knowledge is not retained or accumulated. The learning process is performed without considering past learned knowledge from other tasks.

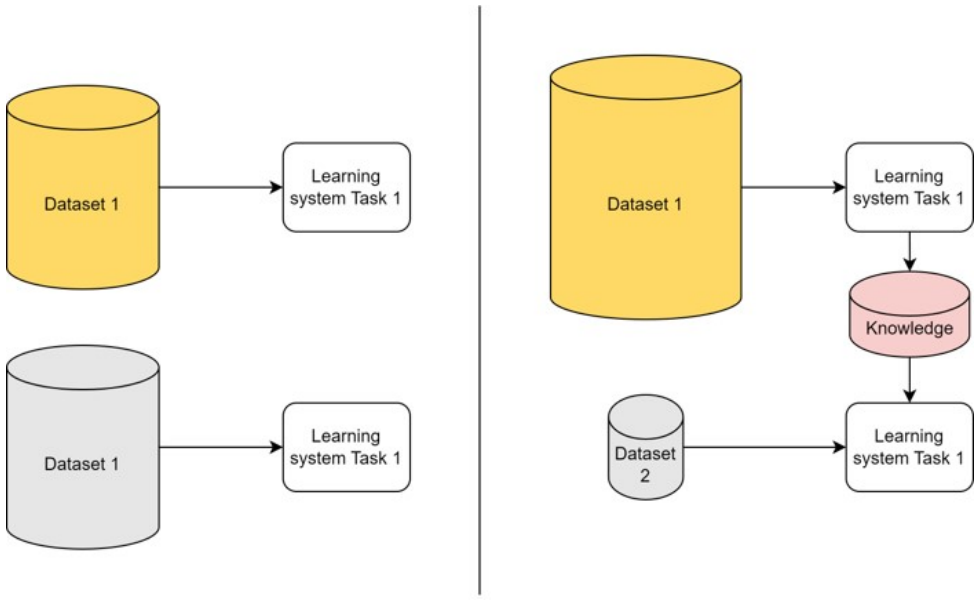

Figure 14: Transfer learning diagram

2.3.1 Importations

In order to be able to build our model, we need to import all the required modules.

- **Numpy** to be able to compute matrix
- **Matplotlib** to print figures
- **Keras** to build our neural network
- **Sklearn.metrics** to have the confusion matrix of our training model

We will use the MNIST database imported from Keras as before to start. To apply transfer learning techniques, the first step is to select a pre-trained model to use for the transfer. I chose VGG16 as my starting point.

```
1  # Library for Transfer Learning
2  from keras.applications import VGG16
3  from keras.applications.vgg16 import preprocess_input
4  from keras.utils import img_to_array
5  from keras.utils import array_to_img
```

VGG is indeed integrated into the Keras library, and it represents a powerful Convolutional Neural Network (CNN) architecture developed by the Visual Geometry Group (VGG) at the University of Oxford in 2014. This model consists of 16 layers, primarily utilizing 3x3 convolutional filters, along with three fully connected layers for classification purposes. It's important to note that VGG was trained on the ImageNet dataset, which comprises over one million images.

2.3.2 Transfer Learning

Once the dataset has been split, as explained in the previous section, the transfer learning process can be initiated automatically using the following command:

```
history = model.fit(X_train,y_train,epochs=20,batch_size=128,verbose=True,validation
```

I didn't change the parameters of the previous sections (loss : categorical_crossentropy, optimizer : adam, metrics :accuracy) to compile my model. We reach these results :

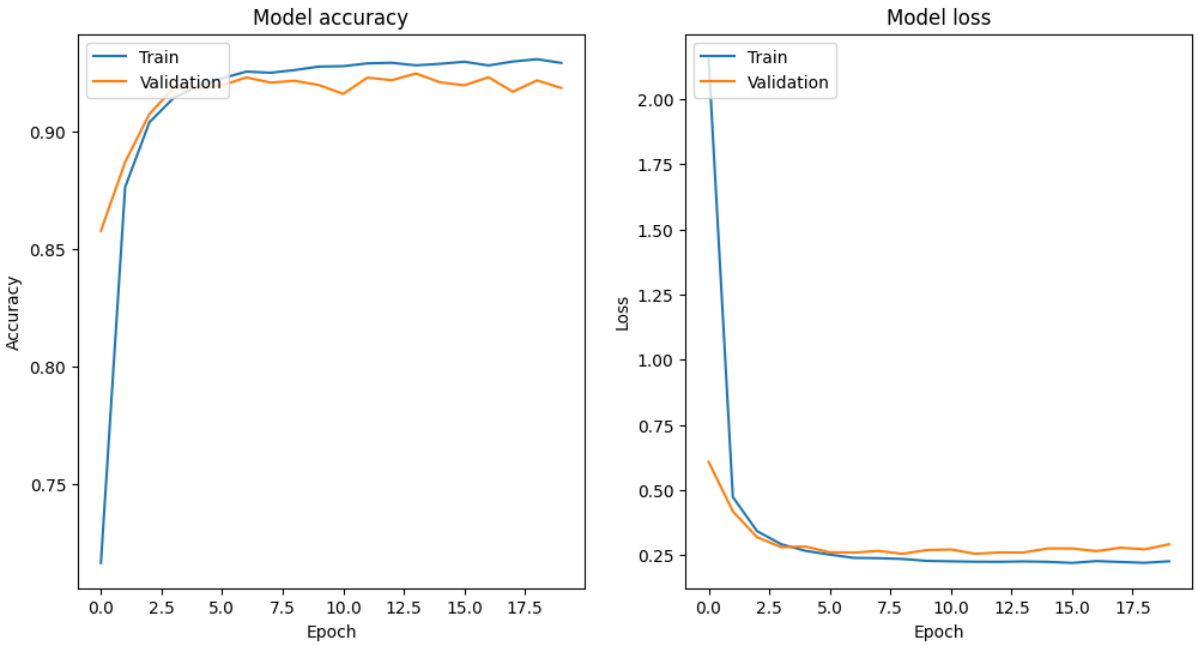

Figure 15: Accuracy and Loss curves with 20 epochs

We get an accuracy of 92% with 20 epochs which is pretty fast.

2.3.3 Using out model to make prediction

To see the results of the transfer we can look at the confusion matrix below, based on the test set :

```
[[1167    1    3    0    5    0   16    0    5    1]
 [   0 1275    6    1   24    3    5    7    1    1]
 [   0    5 1050   12   13   35   21   10   23    2]
 [   1    0   35 1028    4  128    6    3   38    6]
 [   2    0    1    0 1150    1    5    4    2   10]
 [   3    2   17   41    5  913   15    3   14    1]
 [   4    1    4    1    9    4 1181    0    0    1]
 [   2    6   17    8   44   10    1 1163    2   47]
 [   4    4   29    8   26   46   15    6 1014   30]
 [   6    0    5    5   43    9    8   12   12 1083]]
```

Figure 16: Confusion matrix

After applying transfer learning and evaluating the model's performance, we obtain a confusion matrix that illustrates the accuracy observed in the previous curve, revealing a strong diagonal pattern.

2.4 Application test

Now that all our model are trained we can now try to use them on an app with images different from the dataset. The idea here of the app here is to let the user draw a figure with the mouse on the computer and let the model to recognize what figure has been drawn. To do this application we are going to use the framework *streamlit*.

2.4.1 Drawing part

Before to implement the model into the application we need to have the possibility to draw on the app. To do this we going to use a program shared on GitHub called *streamlit-drawable-canvas*. Thus, without too much effort, we are able to draw and export figures on the app.

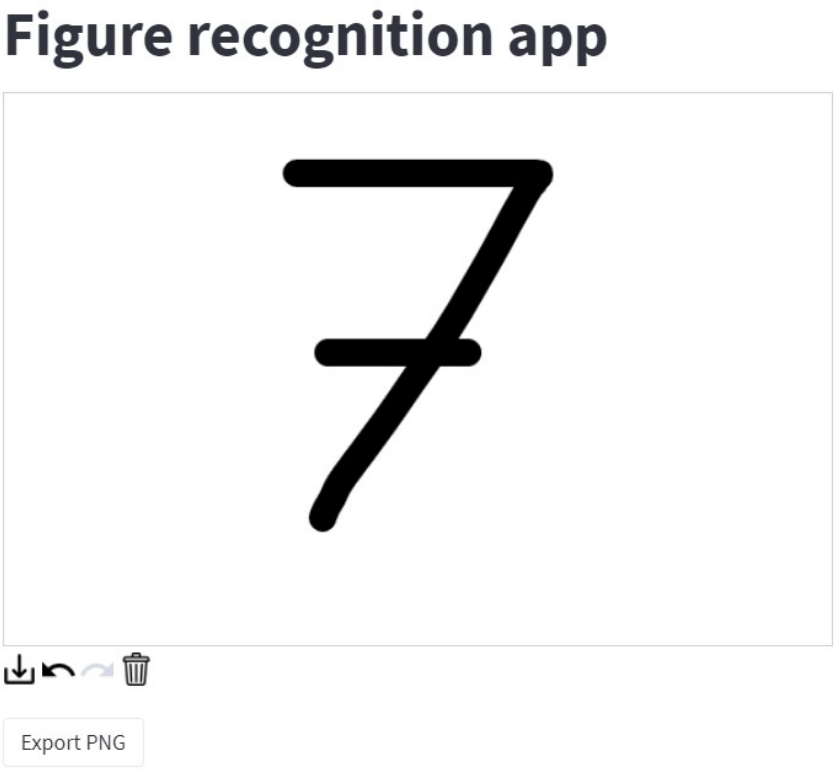

Figure 17: drawing widget on streamlit

3 Custom dataset

Now that we have gained a deeper understanding of the method of training with transfer learning, we can begin the process of building our own dataset, which is directly linked to

the purpose of our project. Once the dataset is built, we will proceed as before, following several steps as explained in the diagram below:

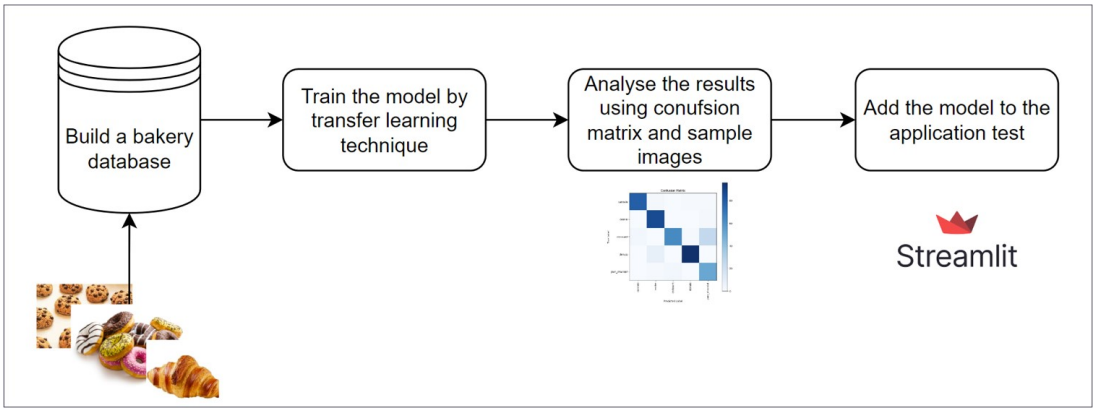

Figure 18: Process

These steps encompass preparing the data, splitting it into training and validation sets, selecting an appropriate pre-trained model for transfer learning, fine-tuning the model on our dataset, and evaluating its performance. Finally we will build an application with streamlit in order to print the result of the detection.

3.1 Build a dataset

To build the dataset for classification images, I used a library called *bing-image-downloader*. Thanks to this library we'll be able to download plenty of images from Bing images. To use it, we just have to create a folder and specitfy in entry the name of the class (for example "croissant"). We can also vary the number of images to download, we'll increase this number to the maximum (300).

```
1 from bing_image_downloader import downloader
2 downloader.download("cookie", limit=300,output_dir="test")
```

After downloading images automatically, we just need to check each images folder and delete all the pictures that doesn't fit with the class.



3.2 Transfer learning

To begin, we construct a model with two classes, "croissant" and "cannelet." The downloaded images are split into two folders: one for training and another for testing. This division is done manually, ensuring a distribution of 70% for training and 30% for testing. In this example, there are 260 and 175 images belonging to the two classes, respectively, in the train and test folders.

After importing VGG16, which is pre-trained on the ImageNet dataset, we freeze all the layers to prevent updates during training. We then add custom layers on top of the pre-trained VGG model, including a Flatten layer to convert the VGG output to a 1D feature vector, followed by Dense layers with ReLU activation functions. The final Dense layer uses the softmax activation function. The model is compiled with a categorical cross-entropy loss function, an Adam optimizer, and the accuracy metric.

Next, we train the custom layers for a specified number of epochs (in this case, 10 epochs). The number of epochs can be adjusted to find the optimal performance.

Subsequently, we enter the fine-tuning process. Certain pre-trained layers are unfrozen, and we retrain the entire network. In this case, the model is retrained for 25 epochs. The model is once again compiled with the same loss function, optimizer, and metrics.

This approach allows us to leverage the knowledge learned by VGG16 on the ImageNet dataset and adapt it to our specific classification problem, enhancing the model's ability to classify "croissant" and "cannelet" images accurately.

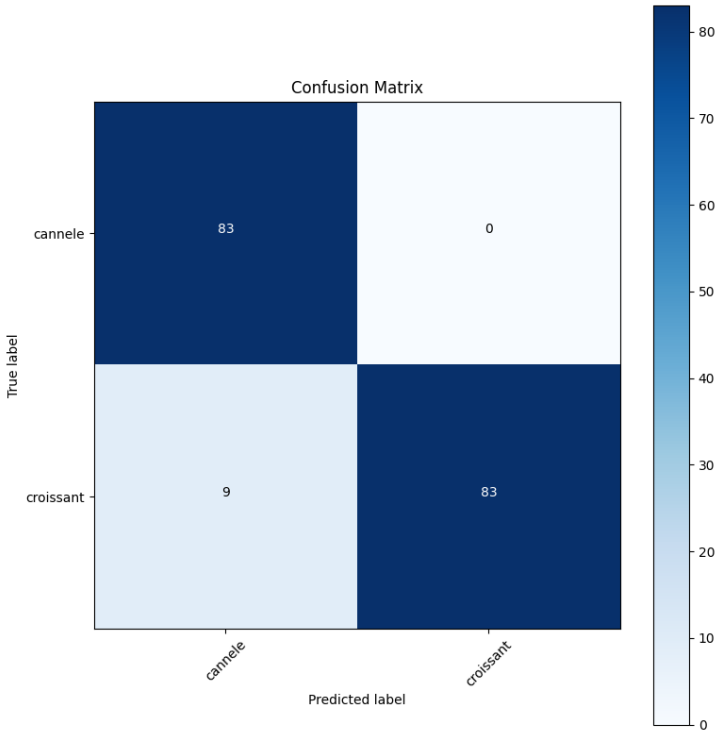

Figure 19: Confusion matrix

As we can see we reach good results.

3.3 Application test

To try the model with custom images, I decided to add the model to the previous streamlit application. I add a section to the application dedicated to the pastry recognition. But this time there is just an image upload tool.

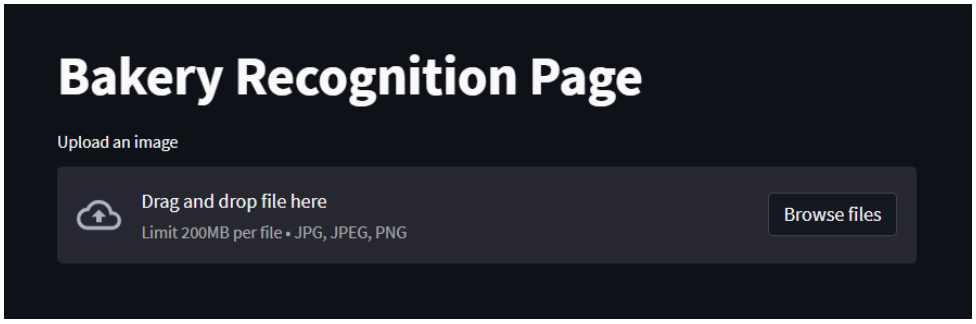

Figure 20: Application test

4 Detection with YoloV8

Up until this point, we have successfully constructed a classification model that is proficient at identifying individual objects within images. This classification process, however, solely pertains to recognizing the object's identity and does not encompass its spatial location within the image. To pass this limitation of single-object detection and get the ability to precisely pinpoint the location of detected objects, we are now poised to integrate YOLOv8 for "You Only Look Once" version 8. It's a state-of-the-art object detection framework. Unlike conventional object detection methods that may require multiple passes over an image, YOLOv8 employs a single pass to both identify object classes and determine their precise spatial coordinates within the image.

4.1 Labeled dataset

Now we are working on a detection problem so we need to work again on the dataset. On every of the precedent images of the dataset, we are going to surround the object to detect. I choose to do this work manually to be more precise and thus get a better accuracy during the detection.

Figure 21: Roboflow

All this work has been done on Roboflow which is a computer vision platform that allows users to build computer vision models faster and more accurately through the

provision of better data collection, preprocessing, and model training techniques. I simply had to add all the images of my dataset on my Roboflow project online and start to label all the images. This process is long but necessary, I had to labeled around 1200 images manually.

Figure 22: Detection box for the training on roboflow

Roboflow offers augmentation technique to create new training examples for the model to learn from by generating augmented versions of each image in the training set. I did some augmentation with rotation, shear, gray-scale, saturation modification, blur and mosaic. Using these techniques I more than double the number of images from 1145 to around 2500. After the Roboflow treatment we get two folders (train and test) shaped to train Yolo.

4.2 Training the model

We now going to train the yolo model. We first need to choose a yolov8 version for our training. I choosed the *yolov8n* version. The training will be done by transfer learning technique as explained before. Unlike VGG16, yolov8 is pre-trained on the cocodataset containing 91 different classes. I did the training using Google Colab. It's cloud-based platform provided by Google that enables users to write and execute Python code in a collaborative and interactive environment. It offers free access to powerful hardware resources, including GPUs that I used. The training has been done with 150 epochs, and a batch size of 16. It gave me these results :

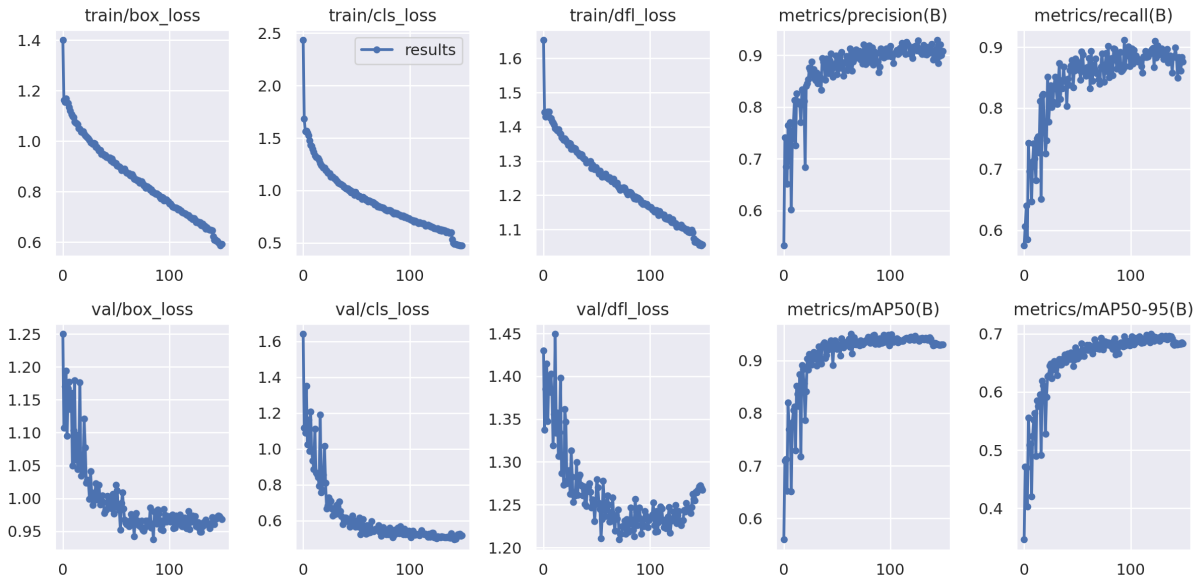

Figure 23: results

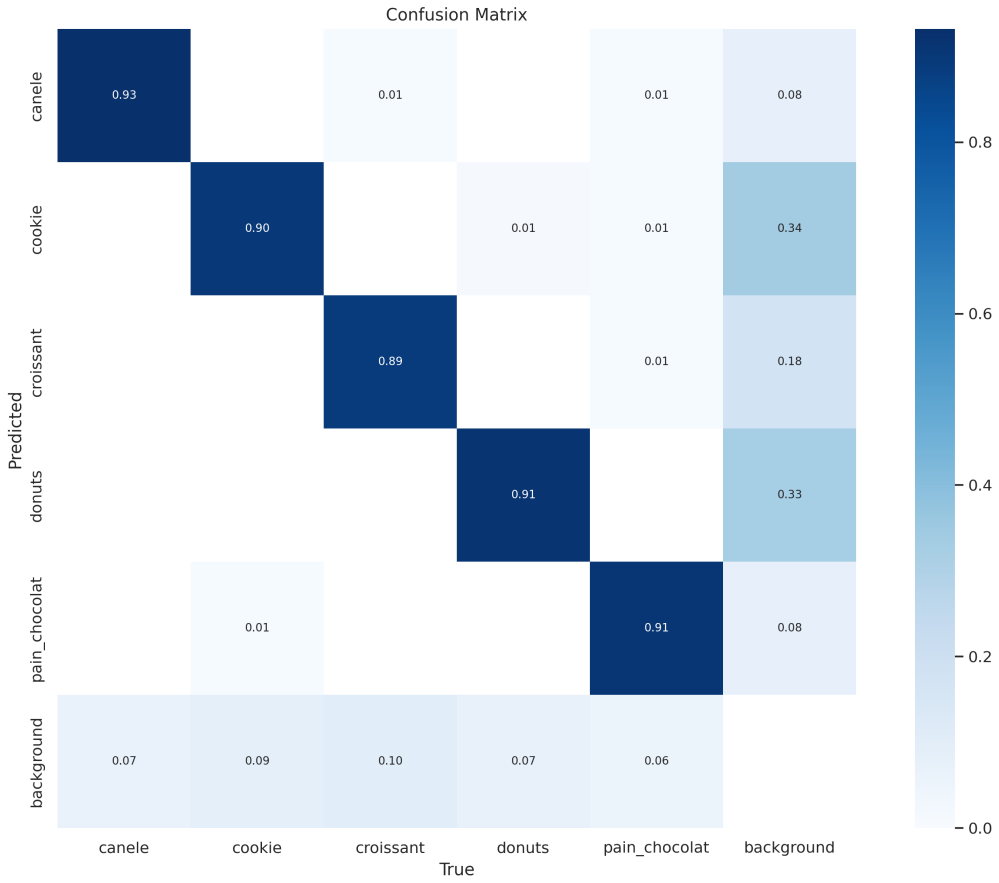

Figure 24: Confusion matrix

As we can see on the results curves, we reach 90% accuracy at the end of the training. If we check the confusion matrix we see that the majority of errors comes from wrong

detection in the background of the images : this problem is called false positive detection. To avoid this problem it's recommended to had 0-10% of background images without any label.

Figure 25: Prediction on samples of the validation dataset

5 Building the prototype

Now that our AI is working, we can build a prototype of the scanner focusing on hardware. In order to do that, we'll use an embedded computing boards (the Jetson Nano), a screen, a keyboard and a mouse. We will then proceed to the assembly of the device.

5.1 Jetson Nano

The Jetson Nano is a compact and powerful computer development board designed by NVIDIA. It is specifically built for AI development and deep learning applications. The Jetson Nano delivers real-time processing for complex AI models. To set up my Jetson Nano for running YOLOv8, I need to format an SD card and install Ubuntu on it. Formatting the SD card ensures that it is compatible with the Jetson Nano and ready for the installation process. Once the SD card is formatted, I can use the Jetson as a computer : Ubuntu is automatically installed during the formatting process. I will need to set up Python and install all the required libraries to run YOLOv8, such as TensorFlow, OpenCV, NumPy, and others. These libraries are essential for running YOLOv8 and

performing real-time object detection. Once everything is set up, I'll be ready to utilize the power of my Jetson Nano and YOLOv8 for my object detection tasks.

Figure 26: Jetson Nano

Figure 27: Camera logitech c920 pro

I used the Logitech C920 Pro to capture the picture for my AI detection prototype.

5.2 Support for the camera

I utilized FreeCAD to design a custom camera holder, and then print it using a 3D printer. I was able to design the camera holder to accommodate my Logitech C920 Pro. Once the design was complete, I transferred the file to a 3D printer, which constructed the physical object layer by layer. The 3D printer's precision ensured that the camera holder was accurately replicated according to my design specifications.

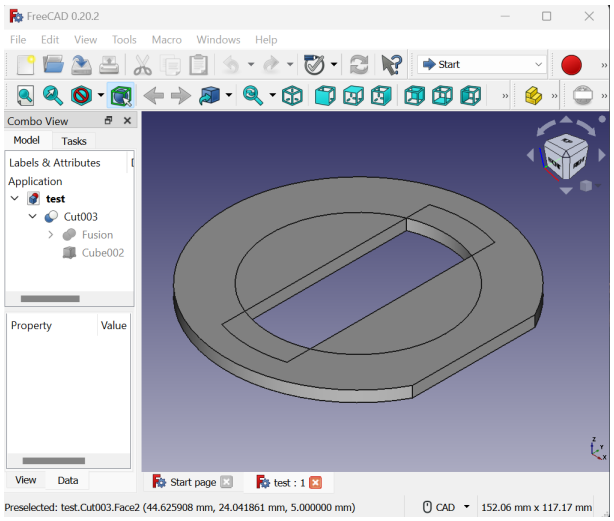

Figure 28: FreeCAD

Figure 29: Camera holder

5.3 Application to run the program

I choosed to add a new section to my streamlit application in order to compile the python program.

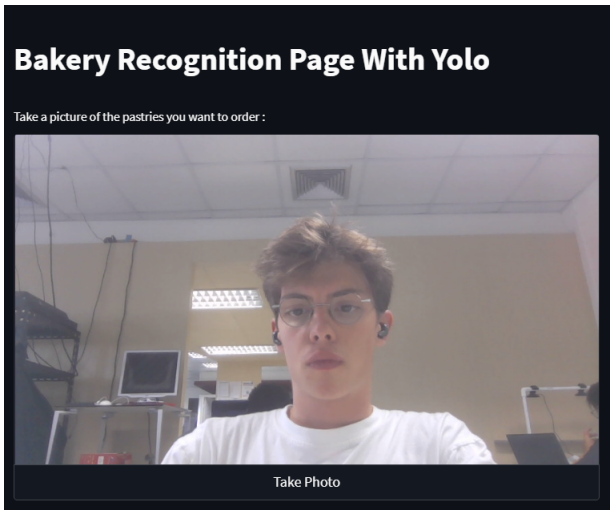

Figure 30: Page for the YOLO recognition

5.3.1 Accounting system

In order to print the price of the customer selection. I coded a function associating a price to each of the classes detected in the image :

```
def price(cakes):
    amount = 0
    for cake in cakes:
        if cake == 0:
            amount = amount + 100  # cannele
        elif cake == 1:
            amount = amount + 103  # cookie
        elif cake == 2:
            amount = amount + 101  # croissant
        elif cake == 3:
            amount = amount + 108  # donuts
        elif cake == 4:
            amount = amount + 109  # pain au chocolat
    return amount
```

As a way of improvement, we could try to create a user interface allowing to automatically add a price to the new classes added or even to allow to modify the price of each of the classes in the interface.

5.4 Result

The resulting prototype below incorporates the aforementioned elements, along with a box equipped with an LED light to capture photos under consistent conditions.

Figure 31: Sample image from the camera fixed on the prototype

Figure 32: Prototype : detection box

5.5 Using the detection box to increase the dataset

In this phase of the project, I captured approximately 500 images of authentic pastries using the camera of the detection box. To ensure the diversity of the dataset, I varied the composition of each image—experimenting with cake positions through layering and incorporating cakes of different sizes. This deliberate approach aimed to enhance the dataset's variety and subsequently optimize the accuracy of object detection. As in the previous stages, I manually labeled all the newly captured images. Upon completing this labeling process, I integrated these images into the existing dataset. Through augmentation techniques the dataset ssurpassing a total of 3000 images. This augmented dataset stands to significantly bolster the performance and robustness of the object detection model.

5.5.1 Training result with the expend dataset

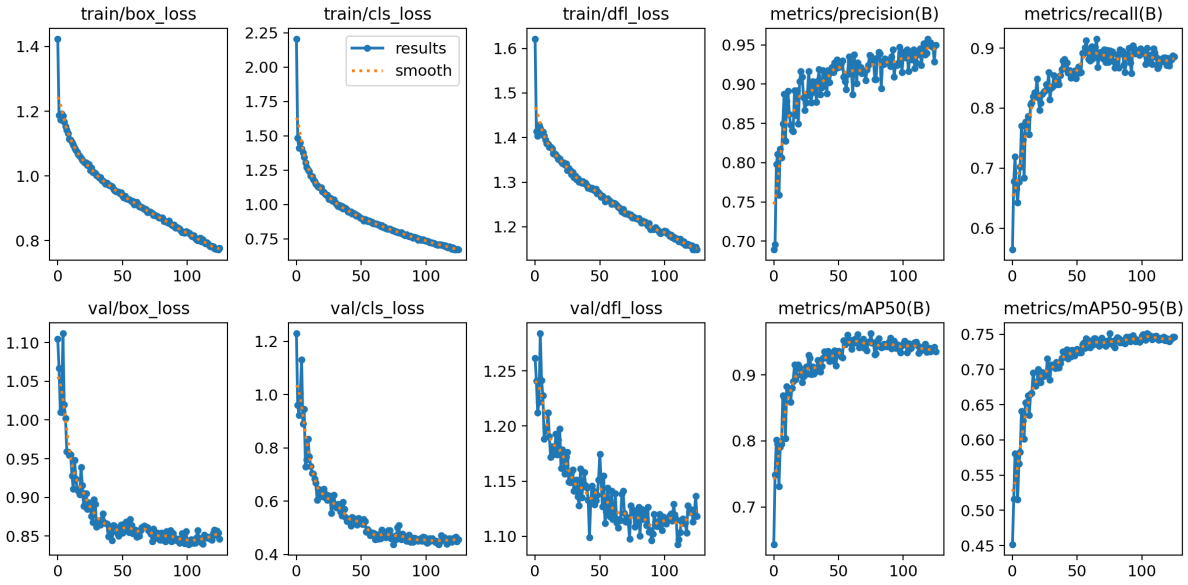

Figure 33: results

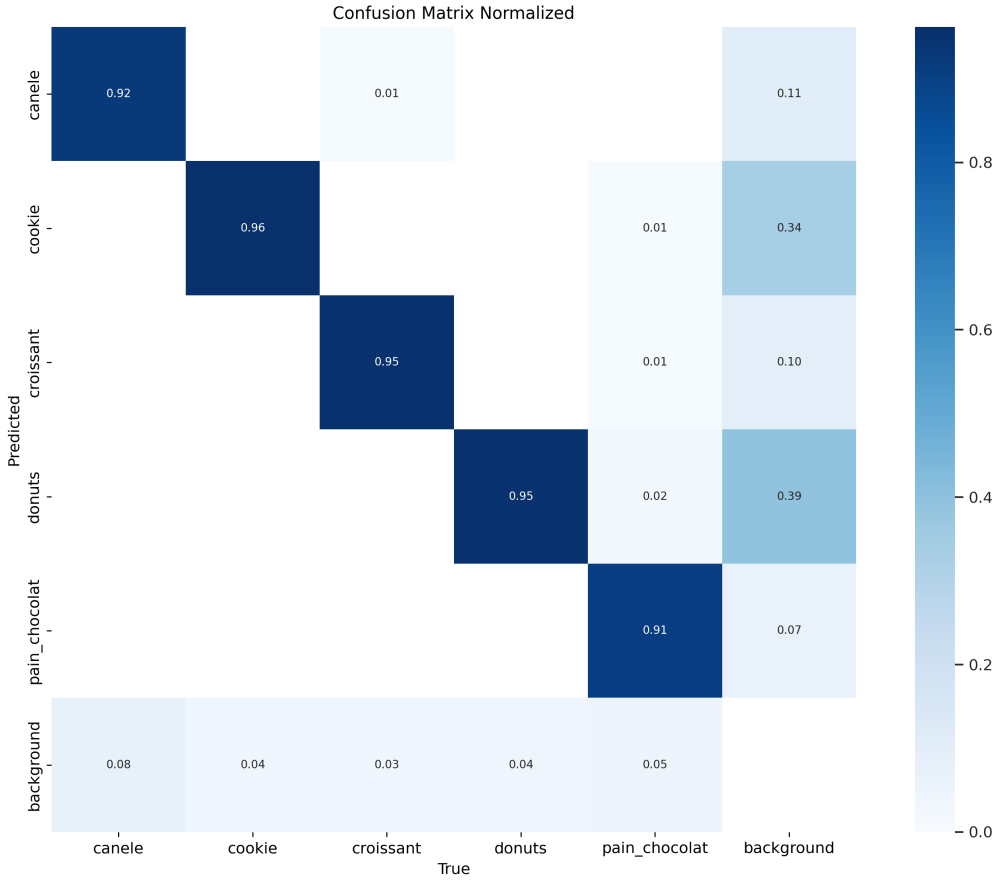

Figure 34: Confusion matrix

As we can see on the results curves, we reach 95% accuracy at the end of the training. If we check the confusion matrix we see as before that the majority of errors comes from wrong detection in the background of the images : this problem is called false positive detection. To avoid this problem it's recommended to had 0-10% of background images without any label.

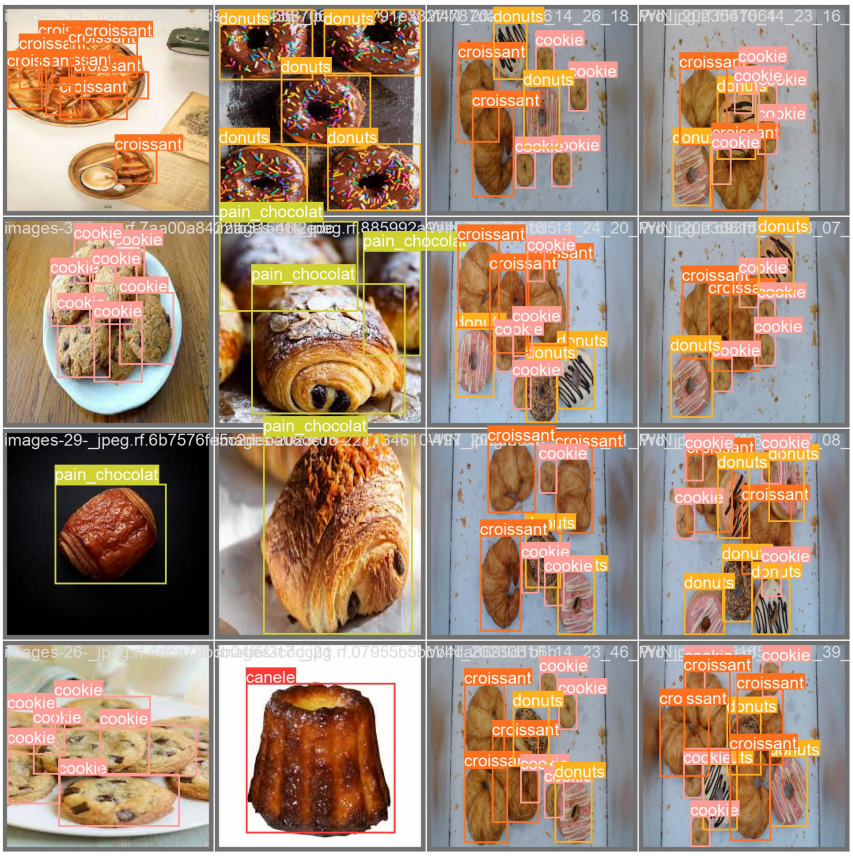

Figure 35: Prediction on samples of the validation dataset

This model is the final version I compiled. It took around 4 hours to get it.

6 Learning automation

The objective of this part is to augment the capabilities of a cake recognition software utilizing the YOLOV8 framework. The initial dataset comprises a substantial number of images of five distinct types of cakes, all captured under consistent lighting conditions and camera angles. The primary goal is to expand the recognition capabilities of the software by introducing a new class of cakes, thereby increasing the number of classes from five to six and ultimately allow you to add as many classes as you want automatically.

To achieve this, a supplementary dataset of approximately 100 real individual images of the new cake class has been collected. All the taken images of this dataset has to be took under the same condition of the condition detection : thus, pictures has to be took inside the detection box. The intention is to employ these images to generate a substantial set of 1000 artificial images for this new class. By combining the new cake images with the existing dataset, a diverse and comprehensive collection of artificially generated images

can be created. For the image placement, I used an Archimedean spiral. I randomly placed the images on the spiral, which covers the entire center of the detection box. This way, the images are gathered at the center of the detection box, making it appear as if it were a real image.

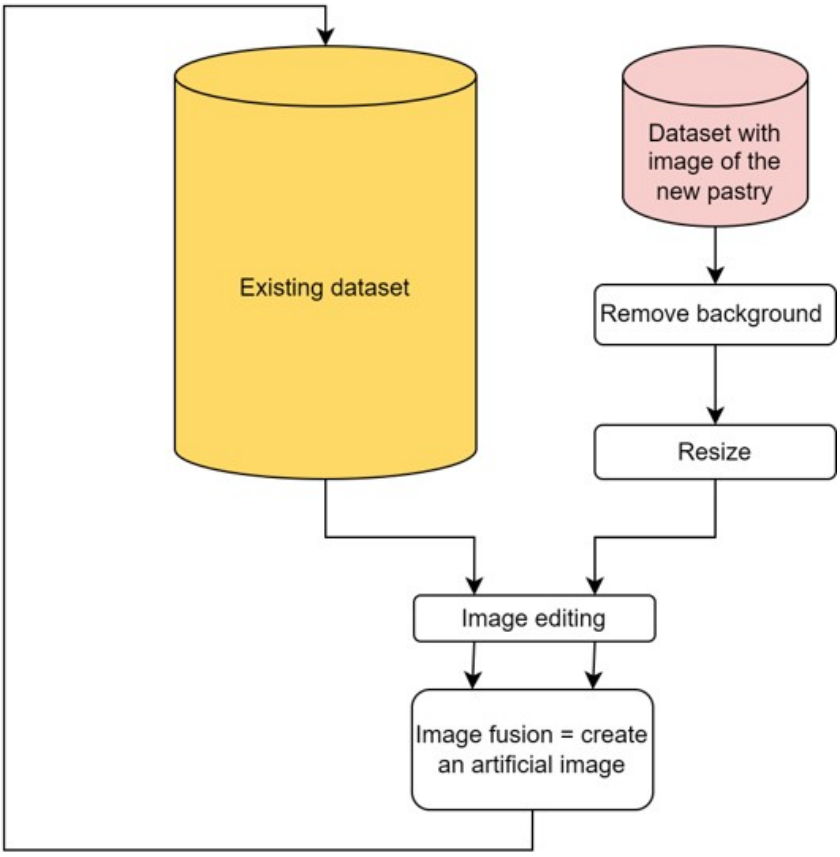

Figure 36: Augmentation process

The motivation behind this software is to enhance the model's ability to accurately identify and classify the new class of cakes. By introducing a wide range of artificial images, the YOLOV8 model can be trained to effectively recognize the new cake class, ultimately elevating the overall performance of the cake recognition system. The successful implementation of this project will not only improve the recognition accuracy of the software but also open up possibilities for incorporating additional classes in the future.

Figure 37: Artificial image generated

Behind each generated image, a label file is created containing the location of the cake on the image along with the class type. In the label file provided below, there are only donuts detected, which is why all lines start with the number 6.

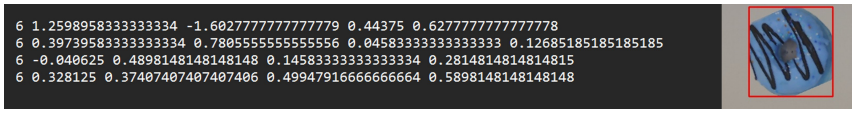

Figure 38: Location file associated to each image

7 Improvement

As my internship draws to a close and the project remains partially complete, the focus shifts towards enhancing its performance. A team of three young Thai interns has taken over the project's reins. Their primary concentration will be on bolstering the database by substantially increasing the volume of images for each class. They will refine the image augmentation system that I initiated. Furthermore, in order to enhance detection accuracy, they will explore diverse solutions, such as increasing the number of cameras for detection and cross-referencing the captured data. Real-time detection implementation will also be considered.

This transition marks the next phase in the project's evolution, and I'm confident that the new team's efforts will further refine and advance the outcomes achieved so far.

References

[1] Muhammad Ardi (2020) *Simple Neural Network on MNIST Handwritten Digit Dataset*, becominghuman.ai.

[2] Jason Brownlee (2022) *Difference Between a Batch and an Epoch in a Neural Network*, machinelearningmastery.com.

[3] Jaz Allibhai (2018) *Building a Convolutional Neural Network (CNN) in Keras*, towardsdatascience.com.

[4] Virat Kothari (2020) *Image Classification of MNIST using VGG16*, kaggle.com.