SWINBURNE VIETNAM

HO CHI MINH CAMPUS

CLASS: **COS30082 – Applied Machine Learning**

Project Report

Topic:

Face Attendance System

FACILITATOR: **Dr. Minh Hoang**

STUDENT NAME:

1. Mai Ngoc Anh Doan - 104220776

TUTORIAL CLASS: Wednesday 1:00PM

HO CHI MINH CITY – NOVEMBER, 2024

Methodology

1. Preprocessing data

```
import os
from PIL import Image
import pandas as pd
from torch.utils.data import Dataset, DataLoader
import torchvision.transforms as transforms

transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=10),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

At the beginning, there are some augmentation would be stated here. These transformations result in extending the database. Moreover, using augmentation can help model to learn under many difficult conditions, fromt that, the accuracy can be improved.

```
# Path to classification data
classification_train_dir = '/kaggle/input/11-785-fall-20-homework-2-part-2/classification_data/train_data'
classification_val_dir = '/kaggle/input/11-785-fall-20-homework-2-part-2/classification_data/val_data'
classification_test_dir = '/kaggle/input/11-785-fall-20-homework-2-part-2/classification_data/test_data'


# Create classification datasets
train_dataset = ImageFolder(root=classification_train_dir, transform=transform)
val_dataset = ImageFolder(root=classification_val_dir, transform=transform)
test_dataset = ImageFolder(root=classification_test_dir, transform=transform)

# Create classification dataloaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=4,prefetch_factor=2)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False, num_workers=4,prefetch_factor=2)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False, num_workers=4, prefetch_factor=2)


data_iter = iter(train_loader)
images, labels = next(data_iter)
print(images.shape, labels.shape)
```

```
torch.Size([32, 3, 128, 128]) torch.Size([32])
```

These are ways of my model read the datasets. Since choosing PyTorch as the framework of this project, the model uses ImageFolder, and DataLoader to load the dataset into the model. In this course, most of the time, the main framework that has been taught and demonstrated is Tensorflow.

Because of that, I have to admit that the choice of using PyTorch is somehow challenging and I have to research on how to work with it.

```python
class VerificationDataset(Dataset):
    def __init__(self, pairs_file, images_dir, transform=None):
        """
        Args:
            pairs_file (str): Path to the pairs text file.
            images_dir (str): Directory with all images.
            transform (callable, optional): Optional transform to be applied on a sample.
        """
        self.pairs = pd.read_csv(pairs_file, sep=" ", header=None, names=["img1", "img2", "label"])
        self.images_dir = images_dir
        self.transform = transform

    def __len__(self):
        return len(self.pairs)

    def __getitem__(self, idx):
        img1_rel_path = self.pairs.iloc[idx, 0]
        img2_rel_path = self.pairs.iloc[idx, 1]
        label = self.pairs.iloc[idx, 2]

        # Construct the absolute path by joining the root images directory with each relative path
        img1_path = os.path.join(self.images_dir, img1_rel_path)
        img2_path = os.path.join(self.images_dir, img2_rel_path)

        # Load images
        try:
            img1 = Image.open(img1_path).convert('RGB')
            img2 = Image.open(img2_path).convert('RGB')
        except FileNotFoundError:
            print(f"File not found: {img1_path} or {img2_path}")
            raise

        # Apply transformations
        if self.transform:
            img1 = self.transform(img1)
            img2 = self.transform(img2)

        return img1, img2, label
```

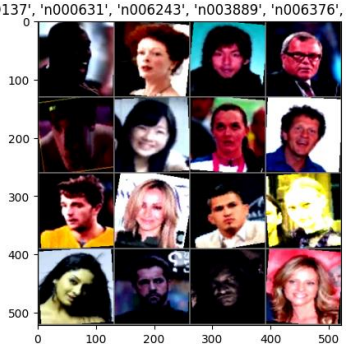
```python
# Paths
verification_dir = '/kaggle/input/11-785-fall-20-homework-2-part-2'  # Root directory for verification data
verification_pairs_file = '/kaggle/input/11-785-fall-20-homework-2-part-2/verification_pairs_val.txt'

# Create verification dataset and dataloader
verification_dataset = VerificationDataset(pairs_file=verification_pairs_file, images_dir=verification_dir, transform=transform)
verification_loader = DataLoader(verification_dataset, batch_size=32, shuffle=False, num_workers=4)

verification_iter = iter(verification_loader)
img1_batch, img2_batch, labels = next(verification_iter)
print(img1_batch.shape, img2_batch.shape, labels.shape)
```

```
torch.Size([32, 3, 128, 128]) torch.Size([32, 3, 128, 128]) torch.Size([32])
```

Apart from the classification dataset, the verification dataset is different. It comes with the .txt file, having the 2 images with pair label {1:same, 0:not same}. The verification dataset is used to validate (test) the metric learning model.

2. Visualizing data

['n001386', 'n002844', 'n009140', 'n005740', 'n001628', 'n009137', 'n000631', 'n006243', 'n003889', 'n006376', 'n000695', 'n007803', 'n008900', 'n002915', 'n008724', 'n004027']



This is visualization of the train dataset. These images look weird because those have been transformed to provide more noise for better training output.

These pairs of images are from the verification dataset. The first pair shows that these are 2 images of 1 person, with label: Same . On the other hand, the second one shows these are 2 different person, with label: Different.

3. Classification model (classification-model.ipynb)

```
ClassificationModel(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(64, 64, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1))
  (bn3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout1): Dropout(p=0.2, inplace=False)
  (conv4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv5): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv6): Conv2d(128, 128, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1))
  (bn6): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout2): Dropout(p=0.2, inplace=False)
  (conv7): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout3): Dropout(p=0.2, inplace=False)
  (fc1): Linear(in_features=57600, out_features=256, bias=True)
  (bn_fc1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc2): Linear(in_features=256, out_features=128, bias=True)
  (bn_fc2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc3): Linear(in_features=128, out_features=4000, bias=True)
)
```

This is my classification model. When reading the requirements of this project, I got confused so I do not know if pre-trained model is acceptable or not. For that reason, I build the whole CNN model with 3 blocks of layers. The mechanism behind this model is that when receiving an image, the CNN will extract features from the image then predict which classes of train dataset that it belongs. This mechanism had been used once with the bird species assignment, although I used transfer learning in that task.

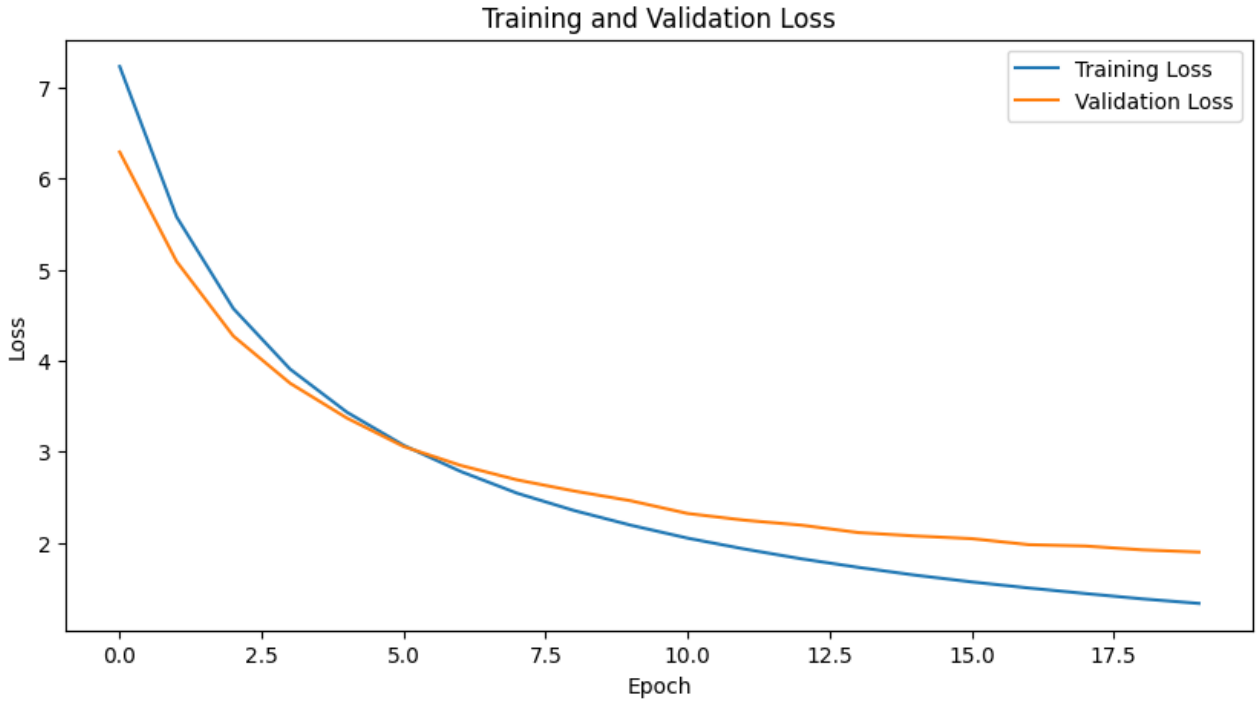
```python
import os

# Path to the saved model
model_path = '/kaggle/working/model_checkpoint.pth'

# Check if a saved model exists and load it if available
if os.path.exists(model_path):
    classification_model.load_state_dict(torch.load(model_path))
    classification_model.eval()  # Set to evaluation mode
    print("Model loaded successfully. Skipping training.")
else:
    print("No saved model found. Proceeding to train the model.")
    # Place your training code here if no checkpoint exists
```

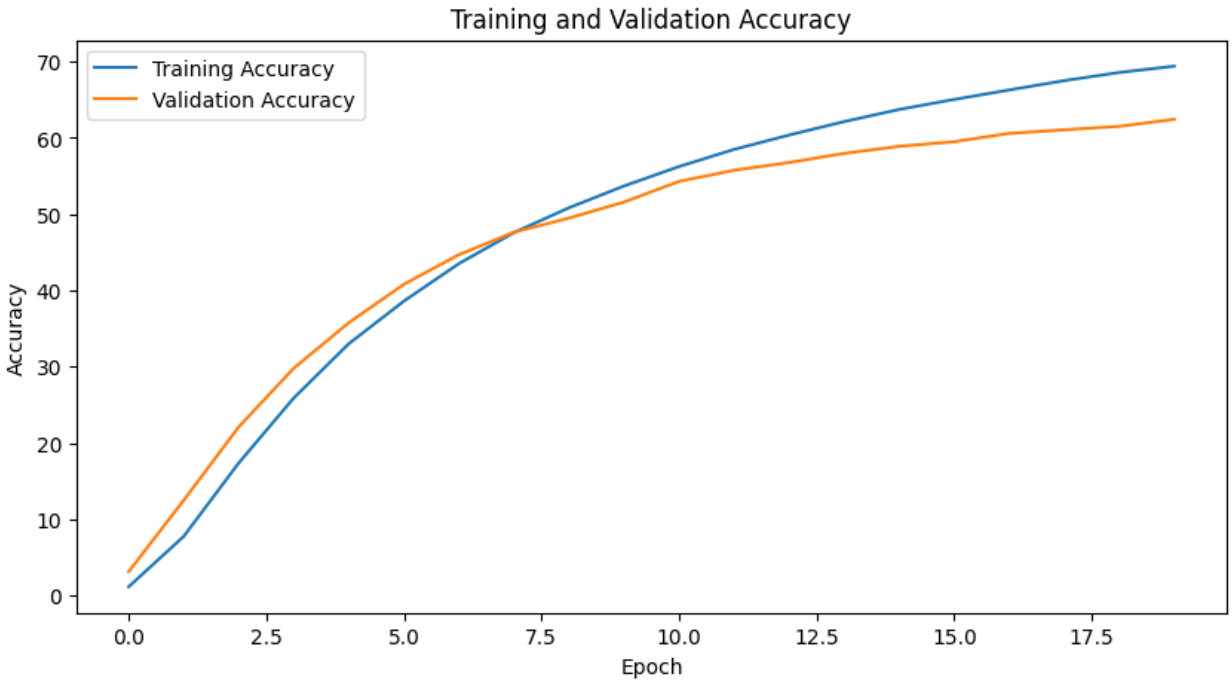
```
No saved model found. Proceeding to train the model.
```

In this project, the biggest challenge is the training time. Because of that, I have to make sure that I have saved model that has finished training. In order not to cause conflict between saved models,

when training the model, it is necessary to check whether the model has been loaded or not. If not, the model begins training, otherwise, it will skip training phase and begin evaluating.

```
Epoch 1/20 11895/11895 [==============================] - 1499s 126ms/step - loss: 7.2284 - accuracy: 1.1297% - val_loss: 6.2896 - val_accuracy: 3.1125%
Epoch 2/20 11895/11895 [==============================] - 1285s 108ms/step - loss: 5.5791 - accuracy: 7.7402% - val_loss: 5.0893 - val_accuracy: 12.4750%
Epoch 3/20 11895/11895 [==============================] - 1256s 106ms/step - loss: 4.5712 - accuracy: 17.3677% - val_loss: 4.2693 - val_accuracy: 22.1000%
Epoch 4/20 11895/11895 [==============================] - 1278s 107ms/step - loss: 3.9056 - accuracy: 25.8968% - val_loss: 3.7508 - val_accuracy: 29.8000%
Epoch 5/20 11895/11895 [==============================] - 1302s 109ms/step - loss: 3.4328 - accuracy: 33.0098% - val_loss: 3.3683 - val_accuracy: 35.7625%
Epoch 6/20 11895/11895 [==============================] - 1259s 106ms/step - loss: 3.0690 - accuracy: 38.5957% - val_loss: 3.0556 - val_accuracy: 40.8000%
Epoch 7/20 11895/11895 [==============================] - 1289s 108ms/step - loss: 2.7828 - accuracy: 43.5143% - val_loss: 2.8490 - val_accuracy: 44.6875%
Epoch 8/20 11895/11895 [==============================] - 1286s 108ms/step - loss: 2.5426 - accuracy: 47.5827% - val_loss: 2.6892 - val_accuracy: 47.6375%
Epoch 9/20 11895/11895 [==============================] - 1264s 106ms/step - loss: 2.3531 - accuracy: 50.8404% - val_loss: 2.5675 - val_accuracy: 49.4875%
Epoch 10/20 11895/11895 [==============================] - 1280s 108ms/step - loss: 2.1918 - accuracy: 53.6964% - val_loss: 2.4599 - val_accuracy: 51.5875%
Epoch 11/20 11895/11895 [==============================] - 1250s 105ms/step - loss: 2.0497 - accuracy: 56.2477% - val_loss: 2.3205 - val_accuracy: 54.3000%
Epoch 12/20 11895/11895 [==============================] - 1251s 105ms/step - loss: 1.9304 - accuracy: 58.4968% - val_loss: 2.2469 - val_accuracy: 55.7625%
Epoch 13/20 11895/11895 [==============================] - 1255s 106ms/step - loss: 1.8230 - accuracy: 60.3826% - val_loss: 2.1926 - val_accuracy: 56.7750%
Epoch 14/20 11895/11895 [==============================] - 1258s 106ms/step - loss: 1.7294 - accuracy: 62.1457% - val_loss: 2.1113 - val_accuracy: 57.9750%
Epoch 15/20 11895/11895 [==============================] - 1256s 106ms/step - loss: 1.6440 - accuracy: 63.7382% - val_loss: 2.0740 - val_accuracy: 58.9000%
Epoch 16/20 11895/11895 [==============================] - 1253s 105ms/step - loss: 1.5692 - accuracy: 65.0545% - val_loss: 2.0444 - val_accuracy: 59.5125%
Epoch 17/20 11895/11895 [==============================] - 1253s 105ms/step - loss: 1.5029 - accuracy: 66.2976% - val_loss: 1.9785 - val_accuracy: 60.6000%
Epoch 18/20 11895/11895 [==============================] - 1251s 105ms/step - loss: 1.4418 - accuracy: 67.5187% - val_loss: 1.9632 - val_accuracy: 61.0750%
Epoch 19/20 11895/11895 [==============================] - 1270s 107ms/step - loss: 1.3834 - accuracy: 68.5935% - val_loss: 1.9214 - val_accuracy: 61.5250%
Epoch 20/20 11895/11895 [==============================] - 1266s 106ms/step - loss: 1.3332 - accuracy: 69.4158% - val_loss: 1.8970 - val_accuracy: 62.4500%
```





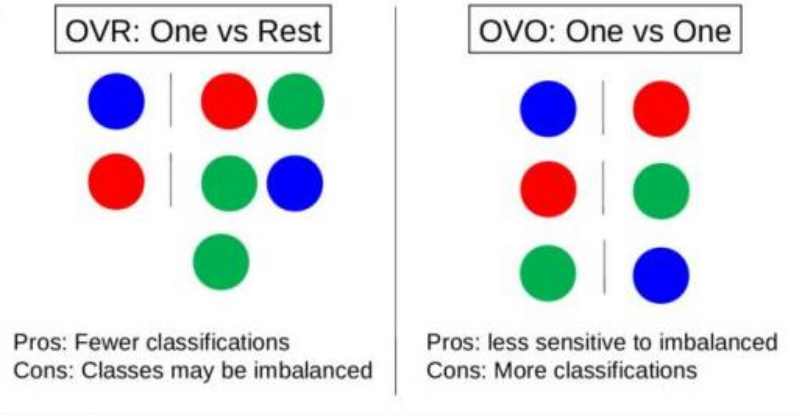
After training the model for 20 epochs, the result comes in with 69% train accuracy and 62% validation accuracy. Beside that, above are 2 plots of loss and accuracy across 20 epochs when the model training. With my point of view, these plots looking good, if anything happen, it may result in overfitting.

```python
# After training, evaluate on the test set
classification_model.eval()  # Set model to evaluation mode
test_running_loss = 0.0
test_correct = 0
test_total = 0

with torch.no_grad():  # Disable gradient calculation for faster computation
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)  # Move data to GPU

        # Mixed precision test pass
        with autocast():
            outputs = classification_model(images)
            loss = criterion(outputs, labels)

        test_running_loss += loss.item() * images.size(0)
        _, predicted = torch.max(outputs, 1)
        test_total += labels.size(0)
        test_correct += (predicted == labels).sum().item()

# Calculate test metrics
test_loss = test_running_loss / len(test_loader.dataset)
test_accuracy = 100 * test_correct / test_total

# Print test results
print(f"Test Loss: {test_loss:.4f} - Test Accuracy: {test_accuracy:.4f}%")
```

```
/tmp/ipykernel_23/247065606.py:12: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with autocast():
Test Loss: 1.8586 - Test Accuracy: 63.1250%
```

The last stage of this cycle is validating the model with test dataset. This model does not provide AUC score because ROC does not support multi-class classification. Although AUC still available,

but we have to format to OvO (One vs One) or OvR (One vs Rest). Both ways require complex configuration with lots of resources. At the end, I decided to use accuracy to qualify the model.



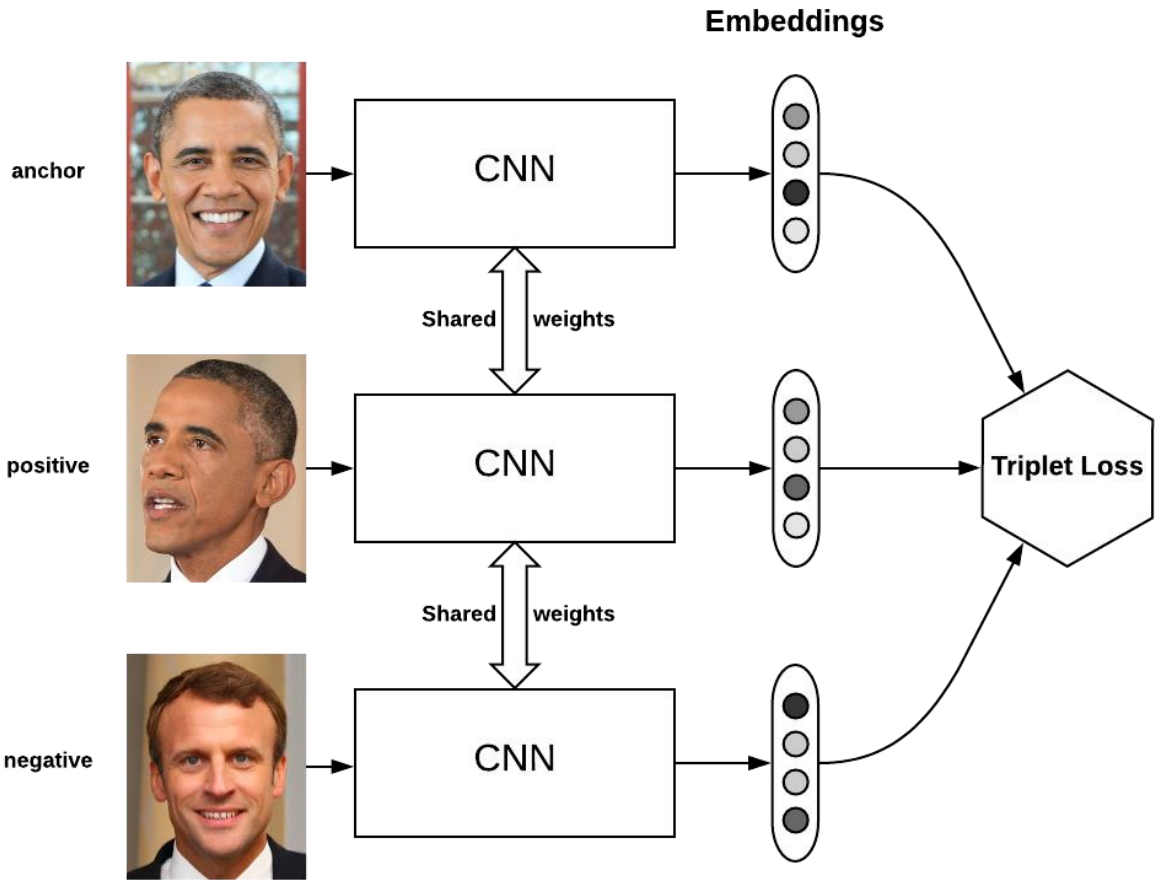
Test Accuracy: 63%

4. Verification model (metric-learning.ipynb)

```python
class EmbeddingNet(nn.Module):
    def __init__(self):
        super(EmbeddingNet, self).__init__()
        # Convolutional layers
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(64)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(128)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)
        self.bn3 = nn.BatchNorm2d(256)
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Fully connected layers
        self.fc1 = nn.Linear(256 * 16 * 16, 256)
        self.bn_fc1 = nn.BatchNorm1d(256)
        self.fc2 = nn.Linear(256, 128)

    def forward(self, x):
        # Convolutional layers
        x = F.relu(self.bn1(self.conv1(x)))
        x = self.pool1(x)

        x = F.relu(self.bn2(self.conv2(x)))
        x = self.pool2(x)

        x = F.relu(self.bn3(self.conv3(x)))
        x = self.pool3(x)

        # Flatten the tensor
        x = x.view(x.size(0), -1)

        # Fully connected layers
        x = F.relu(self.bn_fc1(self.fc1(x)))
        x = self.fc2(x)
        return x
```

This is the architecture of my verification model. In this model, I use the Triplet Network to distinguish between negative, anchor, and positive labels. Because of that, the CNN will play a

rolen as an embedding net that will embedd each images go into the model. Moreover, this CNN's architecture is simpler than the classification because its responsibility is alleviate.



This is the method triplet network that I used in this project.

```python
class TripletNetwork(nn.Module):
    def __init__(self, embedding_net):
        super(TripletNetwork, self).__init__()
        self.embedding_net = embedding_net

    def forward(self, anchor, positive, negative):
        # Generate embeddings for each input
        anchor_embedding = self.embedding_net(anchor)
        positive_embedding = self.embedding_net(positive)
        negative_embedding = self.embedding_net(negative)
        return anchor_embedding, positive_embedding, negative_embedding
```

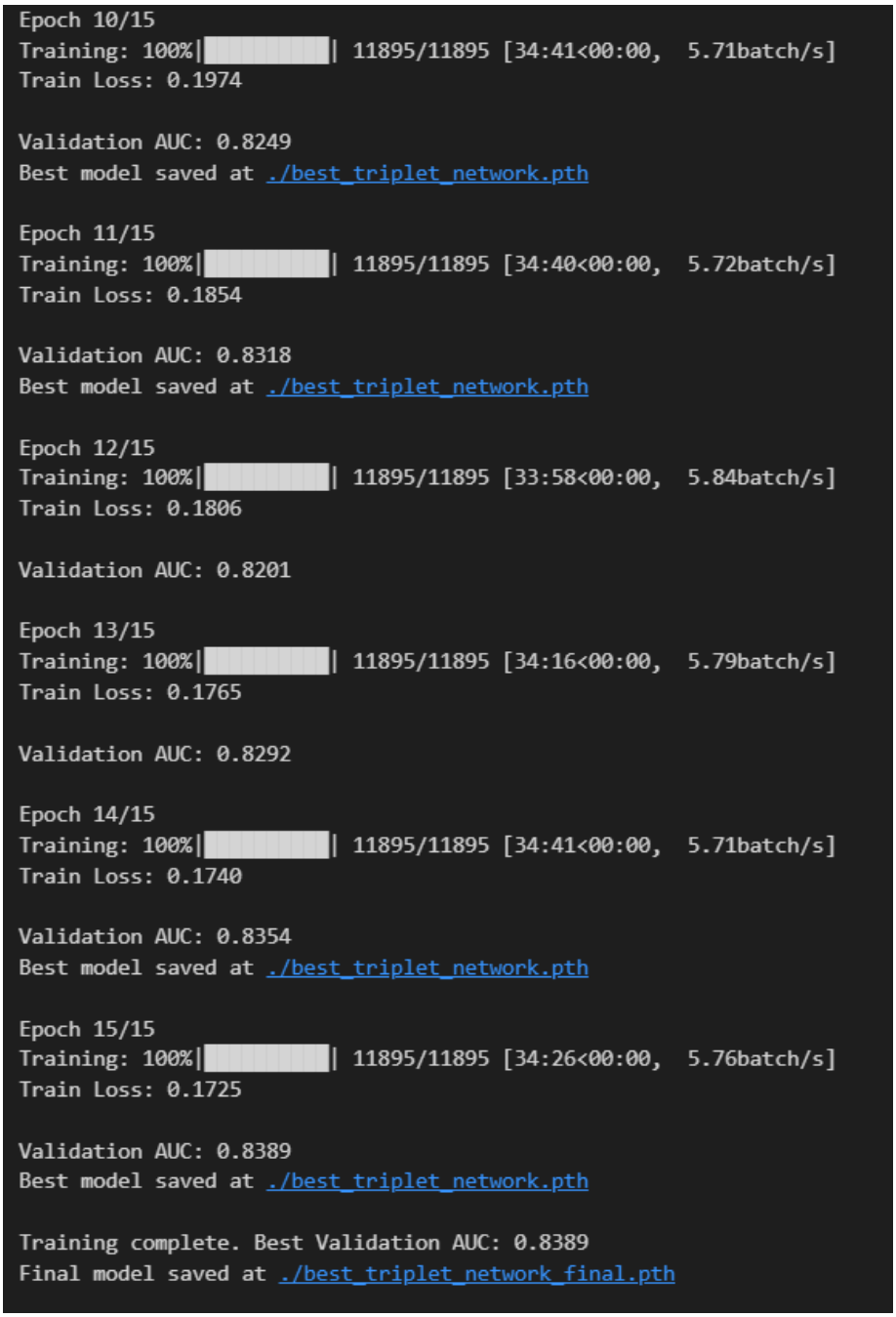
```python
# Define the triplet loss function
def triplet_loss_fn(anchor, positive, negative, margin=1.0):
    positive_dist = F.pairwise_distance(anchor, positive, p=2)
    negative_dist = F.pairwise_distance(anchor, negative, p=2)
    loss = torch.relu(positive_dist - negative_dist + margin)
    return loss.mean()
```

As I said above, this metric learning approach will take advantage of Triplet Network that will output based on the distance between input image with anchor, whether it on the negative or positive side. If output is negative, the input image is not the same in the database. On the other hand, output is positive, leading to the input image has a match with a image in database.

```python
# Wrap train_dataset with TripletDataset
triplet_train_dataset = TripletDataset(train_dataset)

# Create DataLoader for triplet dataset
triplet_train_loader = DataLoader(triplet_train_dataset, batch_size=32, shuffle=True, num_workers=4)

# Example usage
triplet_iter = iter(triplet_train_loader)
anchors, positives, negatives, labels = next(triplet_iter)
print(f"Anchor batch shape: {anchors.shape}")
print(f"Positive batch shape: {positives.shape}")
print(f"Negative batch shape: {negatives.shape}")
print(f"Labels batch shape: {labels.shape}")
```

```
Anchor batch shape: torch.Size([32, 3, 128, 128])
Positive batch shape: torch.Size([32, 3, 128, 128])
Negative batch shape: torch.Size([32, 3, 128, 128])
Labels batch shape: torch.Size([32])
```

In order to train a Triplet Network, it is necessary to has 3 images in 1 batch with 3 labels (negative, anchor, positive). There is no dataset that suitable for this format at the beginning. Because of that, I have created triplet dataset from train dataset since it subfolder of the train data folder are 4000 classes, each subfolder is 1 class with approximately 50 images of the same person. For the purpose of this, the triplet dataset will take 2 images from the same class and label anchor and positive. At the same time, 1 image will be taken from another class and labeled negative. By that, we can successfully create a triplet dataset.

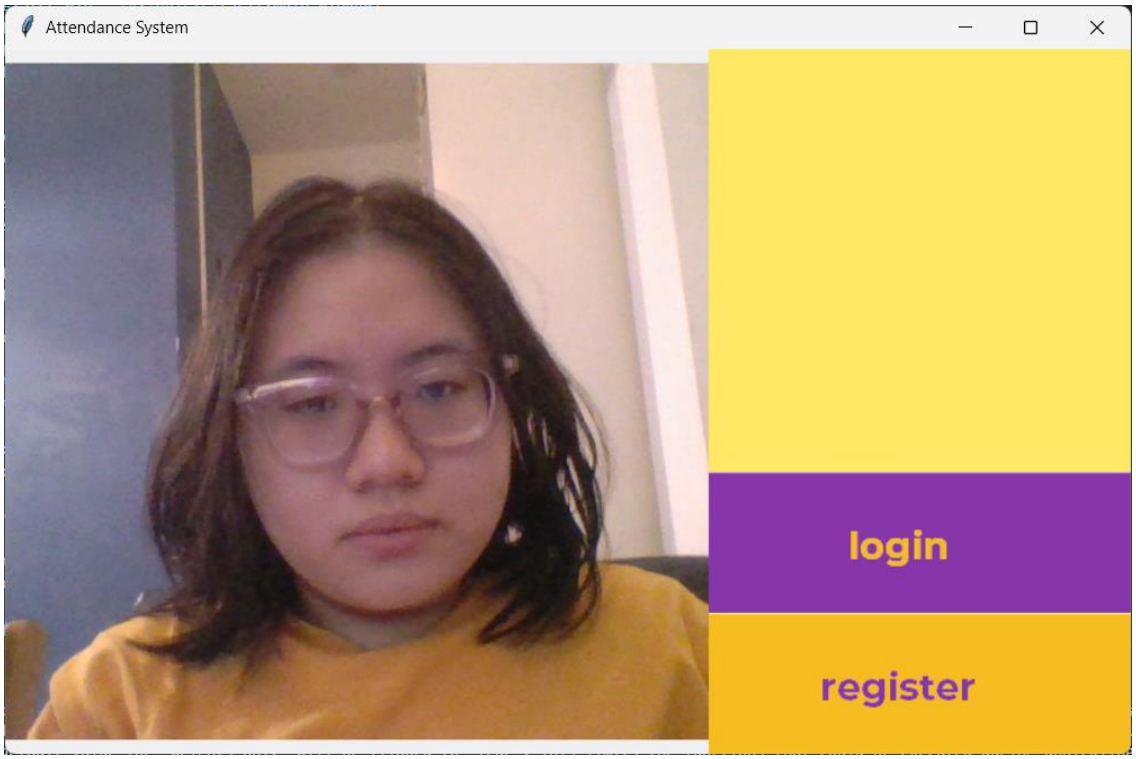


After 15 epochs, the AUC score reached 0.83 and the model is saved for developing a face attendance system. I cannot print the training AUC score after each epoch because ROC continues to not support multi-class classification and the triplet dataset has 3 class, reflecting 3 labels. In order to validate along with training, I have implemented calculating AUC score of verification dataset to not only getting known of the trend in validation AUC score, but also having something to ensure that we on the right track.

Validation AUC Score: 0.83

5. Face Attendance System

```
class App:
    def __init__(self):
        self.main_window = tk.Tk()
        self.main_window.title("Attendance System")
        self.main_window.geometry("800x500+400+150")
        self.main_window.configure(background="#FFE863")

        self.login_btn = util.get_button(self.main_window, 'login', '#8736AA', self.login, fg="#F5BD1F")
        self.login_btn.place(x=450, y=300)

        self.register_btn = util.get_button(self.main_window, 'register', '#F5BD1F', self.register, fg="#8736AA")
        self.register_btn.place(x=450, y=400)

        self.webcam_label = util.get_img_label(self.main_window)
        self.webcam_label.place(x=0, y=0, width=500, height=500)

        self.add_webcam(self.webcam_label)

        self.db_dir = 'known_faces'
        if not os.path.exists(self.db_dir):
            os.mkdir(self.db_dir)

        # Model loading
        MODEL_PATH = 'best_triplet_network_final.pth'
        self.triplet_net = torch.load(MODEL_PATH, map_location=torch.device('cpu'))  # Load the full model
        self.triplet_net.eval()  # Set to evaluation mode
        self.embedding_net = self.triplet_net.embedding_net  # Extract the embedding network

        self.log_path = 'attendance.txt'
```



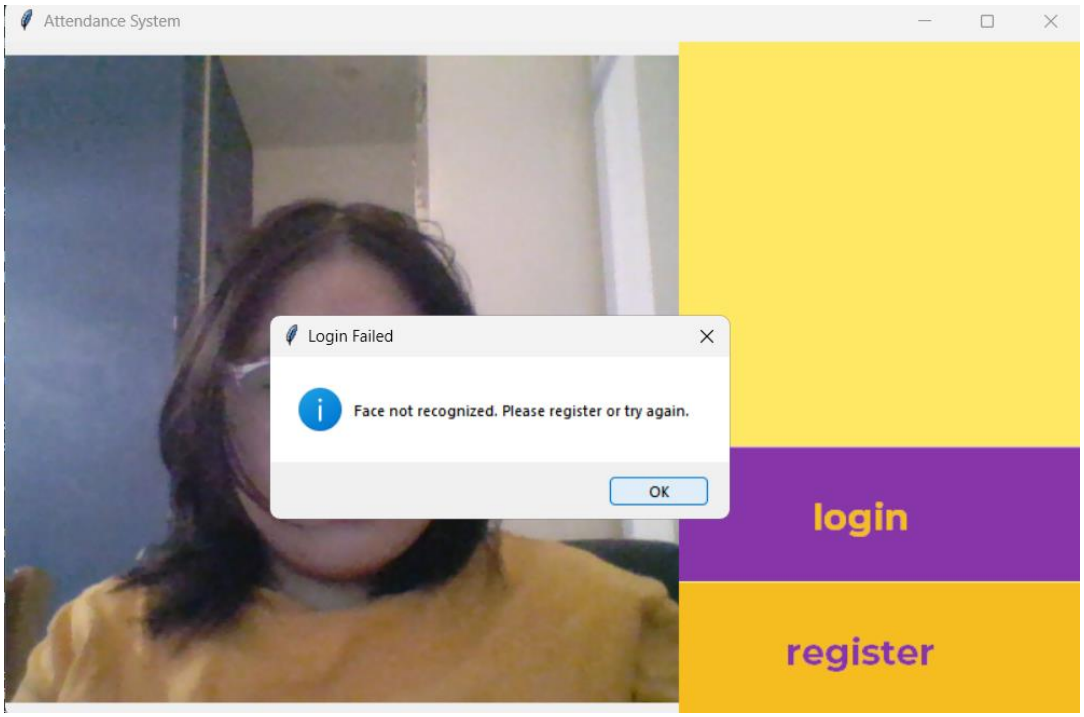
In this application, I use Tkinter framework for creating interface for this attendance system. Some simple design elements are stated in initialize process. The mechanism of this is that when the users register their face through the interface, the image will be stored in the known_faces folder. The last section in this __init__ is model loading. In this part, my model path is stated. With the path, the model is loaded and changed to evaluation mode. Last but not least, the log_path is the file to log whenever the person successfully login.

```
def add_webcam(self, label):
    if 'cap' not in self.__dict__:
        self.cap = cv2.VideoCapture(0)

    self._label = label
    self.process_webcam()

def process_webcam(self):
    ret, frame = self.cap.read()

    if not ret:
        print("Failed to capture frame from webcam. Check if the webcam is properly connected.")
        return

    self.most_recent_capture_arr = frame

    img_ = cv2.cvtColor(self.most_recent_capture_arr, cv2.COLOR_BGR2RGB)
    self.most_recent_capture_pil = Image.fromarray(img_)

    imgTk = ImageTk.PhotoImage(image=self.most_recent_capture_pil)

    self._label.imgTk = imgTk
    self._label.configure(image=imgTk)

    self._label.after(20, self.process_webcam)
```

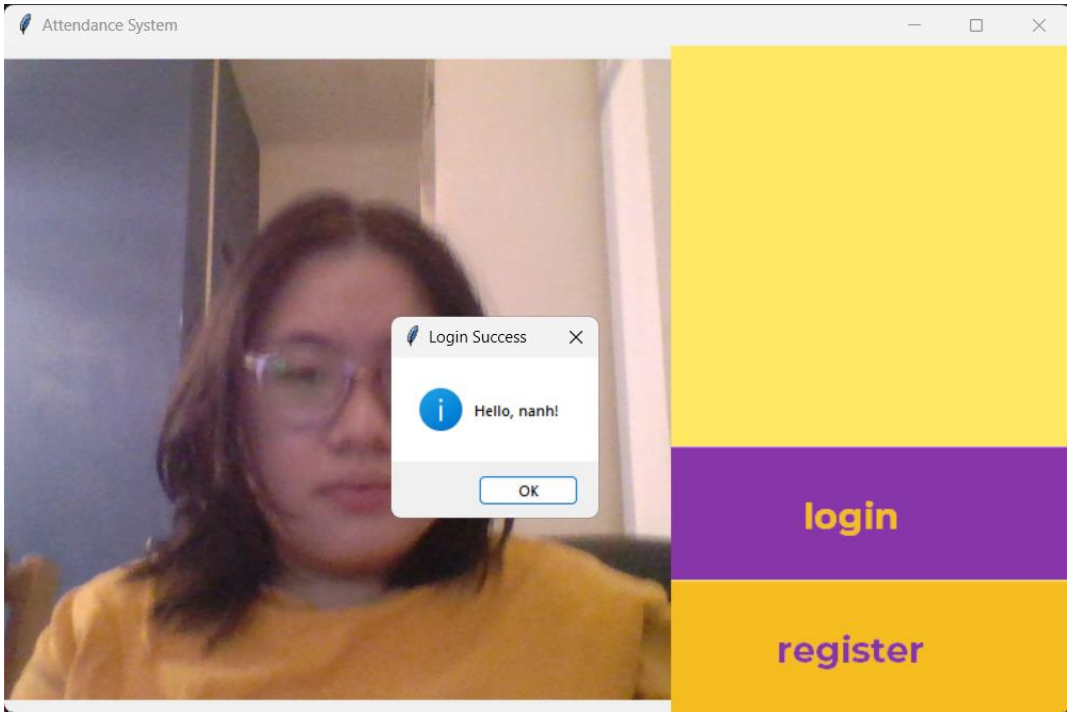
The camera is set up in these 2 functions above, together they can handle the camera functions efficiently on the Tkinter application interface. The add_webcam function helps checking if the camera is available or not, along with assigning Tkinter label for adding camera to the app in the future step. The process_webcam is called to process all necessary steps of the frame, including checking for capture errors, saving images, converting color, changing format to the suitable one. Finally, this function updates the tkinter label and scheduel for next frame update after 20 miliseconds.

```
class App:
    def login(self):

        for file_name in os.listdir(self.db_dir):
            if file_name.endswith('.jpg'):
                file_path = os.path.join(self.db_dir, file_name)
                registered_img = cv2.imread(file_path)
                registered_tensor = preprocess(cv2.cvtColor(registered_img, cv2.COLOR_BGR2RGB)).unsqueeze(0)

                with torch.no_grad():
                    registered_embedding = self.embedding_net(registered_tensor)

                # Normalize registered embedding
                registered_embedding = normalize(registered_embedding, p=2, dim=1)

                # Calculate distance
                distance = torch.norm(detected_embedding - registered_embedding).item()

                if distance < min_distance:
                    min_distance = distance
                    matched_name = os.path.splitext(file_name)[0]

        if min_distance < 0.7:
            util.msg_box('Login Success', f'Hello, {matched_name}!')
            with open(self.log_path, 'a') as f:
                f.write('{}, {}\n'.format(matched_name, datetime.datetime.now()))
                f.close()
        else:
            util.msg_box('Login Failed', 'Face not recognized. Please register or try again.')
```

Before register:



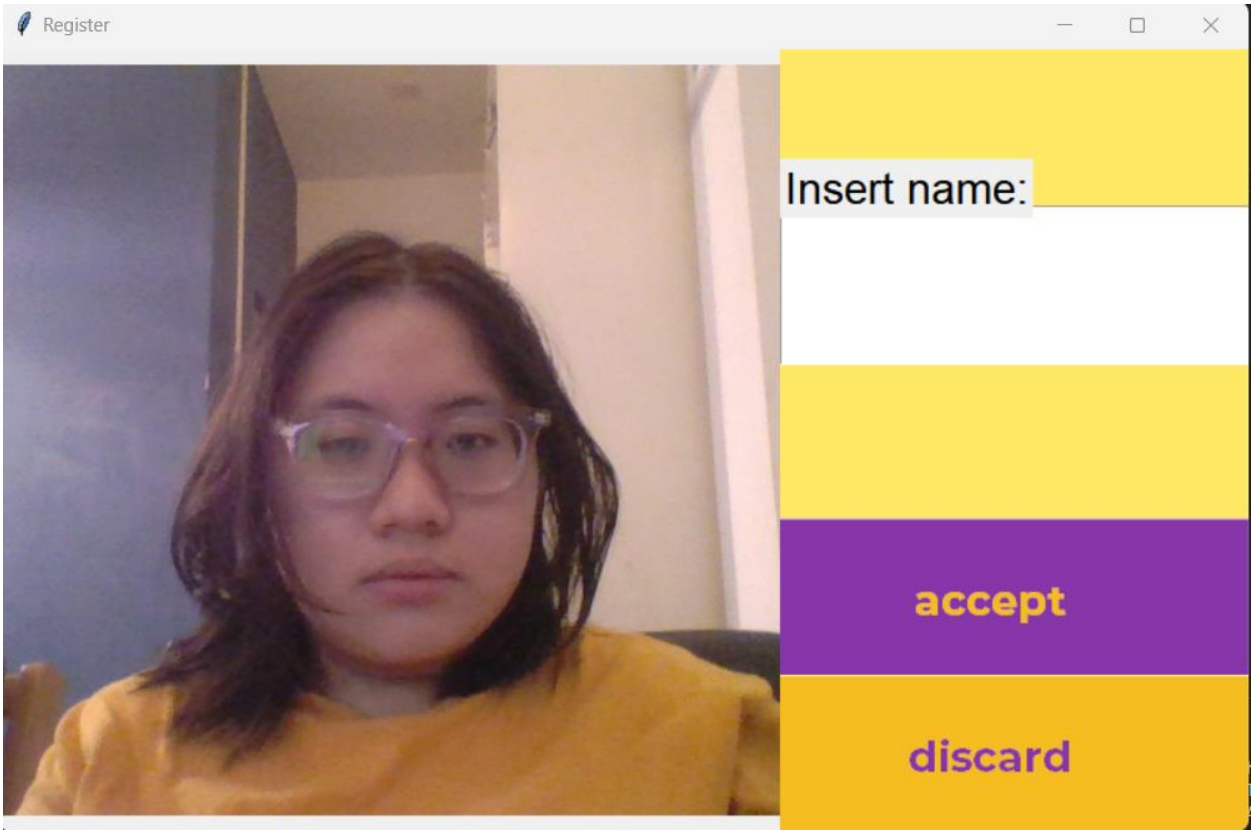
After registering as nanh:

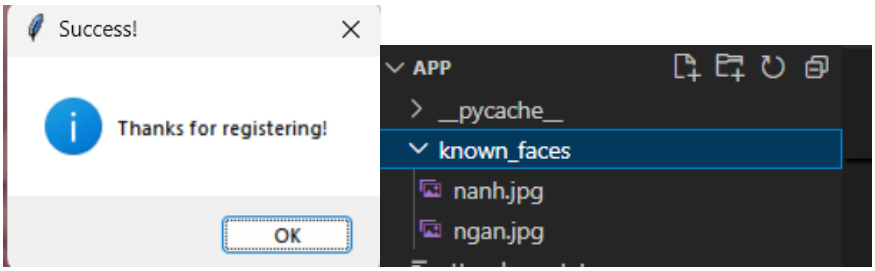


In login fuction, the distance between embedding images and embedding current frame, if the min distance between 2 elements is under the threshold (here is 0.7), the system will pop up the message

box showing that the login is success. On the other hand, the login failed aand the system will show message box saying that face not recognized.

```
def register(self):
    self.register_win = tk.Toplevel(self.main_window)
    self.register_win.geometry("800x500+410+160")
    self.register_win.configure(background="#FFE863")
    self.register_win.title("Register")


    self.accept_btn = util.get_button(self.register_win, 'accept', '#8736AA', self.accept, fg="#F5BD1F")
    self.accept_btn.place(x=450, y=300)

    self.discard_btn = util.get_button(self.register_win, 'discard', '#F5BD1F', self.discard, fg="#8736AA")
    self.discard_btn.place(x=450, y=400)

    self.capture_label = util.get_img_label(self.register_win)
    self.capture_label.place(x=0, y=0, width=500, height=500)

    self.add_img(self.capture_label)

    self.entry_text = util.get_entry_text(self.register_win)
    self.entry_text.place(x=500, y=100)

    self.text_label = util.get_text_label(self.register_win, 'Insert name:')
    self.text_label.place(x=500, y=70)
```





When users want to register their image, another window will pop-up which capturing the current frame. There are 2 choices with this the captured image, user can accept it then provide their name. If the image is not good enough, users can discard it and try taking picture that suit their preference. The name that user insert is also the name of the image.

```
# Save the embedding
    EMBEDDINGS_FILE = 'embeddings.pkl'
    if os.path.exists(EMBEDDINGS_FILE):
        with open(EMBEDDINGS_FILE, 'rb') as f:
            known_embeddings = pickle.load(f)
    else:
        known_embeddings = {}

    known_embeddings[name] = embedding
    with open(EMBEDDINGS_FILE, 'wb') as f:
        pickle.dump(known_embeddings, f)

    util.msg_box('Success!', 'Thanks for registering!')
    self.register_win.destroy()
```

When registering image, the features of image will be extracted immediately and saved as embedding.pkl. By that when need to comparing between current frame with the database, the reuslt will be outputted faster.

In order to help main.py works accurately, there are 2 files: util.py and model.py

- The util.py file: some functions that helps me in working with Tkinter interface.
- The model.py file: my CNN model and triplet network to work as face attendance system

Result and Discussion

1. Result

According to what I have stated above, the classification model can not be evaluated by the AUC score because this model is multi-class classification. Moreover, when reading the project requirements, I do not understand if I need to evaluate the classification model with the verification pairs or not. Because of that, I have try evaluating classification model with the verification pair and outputting the AUC score. Here is the code that do not appear in the final notebook:

```
import torch
from sklearn.metrics import roc_auc_score, accuracy_score
from torch.nn.functional import cosine_similarity
import numpy as np

device = torch.device("cuda" if torch.cuda.is_available() else
          "cpu")
classification_model.eval()  # Set the model to evaluation mode

# Initialize lists to store similarity scores and labels
similarities = []
labels = []

print(f"Number of batches in verification_loader:
     {len(verification_loader)}")

with torch.no_grad():
    for img1, img2, label in verification_loader:
        img1, img2, label = img1.to(device), img2.to(device),
                            label.to(device)

        # Generate embeddings
        embedding1 = classification_model(img1)
        embedding2 = classification_model(img2)

        # Compute similarity (using cosine similarity here)
        similarity_batch = cosine_similarity(embedding1,
                                embedding2, dim=1)

# Convert to list and store results for AUC calculation
        similarities.extend(similarity_batch.cpu().numpy())
        labels.extend(label.cpu().numpy())

print(f"Number of similarity scores: {len(similarities)}")
print(f"Number of labels: {len(labels)}")

# Calculate AUC score using the similarity scores and true labels
if len(similarities) > 0 and len(labels) > 0:
    auc_score = roc_auc_score(labels, similarities)
    print(f"AUC Score: {auc_score:.4f}")
```

```
    thresholds = np.arange(-1, 1, 0.01)
    best_threshold = 0
    best_accuracy = 0

    for threshold in thresholds:
        predictions = [1 if sim > threshold else 0 for sim in
                       similarities]
        accuracy = accuracy_score(labels, predictions)

        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_threshold = threshold
print(f"Optimal Threshold: {best_threshold}, Best Accuracy:
      {best_accuracy:.4f}")

    predictions = [1 if sim > best_threshold else 0 for sim in
                   similarities]

    # Calculate final metrics
    accuracy = accuracy_score(labels, predictions)
    print(f"Final Accuracy with Threshold {best_threshold}:
          {accuracy:.4f}")
else:
    print("No similarities or labels collected. Check data loading
          or model output.")
```

With these lines of code, the classification model will be evaluated based on the verification dataset. The result of output is under:

```
Number of batches in verification_loader: 276
Number of similarity scores: 8805
Number of labels: 8805
AUC Score: 0.8887
Optimal Threshold: 0.9900000000000018, Best Accuracy: 0.6415
Final Accuracy with Threshold 0.9900000000000018: 0.6415
```

As can be seen, the accuracy of this aligns with the accuracy of the test dataset. By this way, the AUC score is 0.88, a kind of good result.

On the other hand, the AUC score is 0.83 of verification model. This is good but not good as the classification model. I do not know what I think but I use this model instead of the classification one for the face attendance system.

2. Discussion

Although I can make it to the end of this project, there are some challenges that I want to address. If I can solve these problems, the project may result in better outcomes.

2.1 The number of classes:

The train dataset of classification model has many classes, the number can leverage to 4000 classes. Despite my model has been trained and evaluated, it is pain to work with 4000 classes. The model takes nearly 8 hours to train and evaluates. In addition, whenever load the data into the notebook on Kaggle, it took 15 minutes everytime I rerun the notebook. These problems leading to long runtime every time tuning hyperparameters. Moreover, the limit of Kaggle GPU is only 30 hours per week. In order to finish this project, I have created 3 Kaggle accounts to fulfill the requirements.

2.2 Room for improvement:

Because I am not aware of the big dataset or long runtime, I begin working on this project late, leading to this project does not fulfill all the requirements. I do not have time to develop the CNN model for spoofing so that my model does not anti-spoofing. Additionally, there are no innovation in my project. Beside all of that disadvantage, I am happy and proud of working on this project in short time. If I have a chance to improve this project, I will need to divide the time more efficiently to develop a better face attendance system.

2.3 Lightning:

After experimenting my application with many persons, I recognized that my application had troubles in verifying images with different lightning conditions. With the same lightning, my application can easily outputs the suitable result. Conversely, person cannot be recognized if the lightning is different. In my opinion, face attendance system usually does not work remotely so I think that not the big problem here.

Last but not least, I want to refer to a github repository that has been the savior in introducing Tkinter to me. ➢ face-attendance-system