MCTA 4363 Deep Learning: Quiz 2 (30 marks)

Name: AHMAD IRFAN BIN KAHAR Matric No:2027127

Fig. 1

Fig. 1 shows a photo of participants in a workshop. As a deep learning engineer, you are tasked with developing an object detection system to detect the faces and count the number of male and female participants in the photo using YOLO v8 and OpenCV.

Marks Distribution:

a) Dataset collection (10 marks)
b) Model selection and training (10 marks)
c) Code for model inference and counting (10 marks)

Instructions for Submission:

1. Submit a single PDF file through the provided link in Microsoft Teams.
2. In the PDF, include:
   - the code snippet,
   - output of the code and
   - the GitHub link containing the code, model and dataset used for training.

Github Link: 2027127_Quiz 2 Repository

a) Dataset Collection

The question requires us to do object detection and count male and female faces in fig. 1. Thus, it requires us to search for datasets online for male and female faces. From Kaggle website we found dataset "Gender Classification Dataset". The dataset consists of 1747 female and 1744 male images for training dataset while their test dataset consists of 100 images for each gender. Due to Fig. 1 image include women wearing hijab, the dataset is combined with another dataset called "Women faces with hijab (scientific use only)". From these combined datasets, a new dataset is created consisting of 400 training images for each gender and 100 images for testing purposes. The datasets split up can be visualized from the table below.

	Training	Testing
Male	400	100
Female	350 (non-Hijab)	50 (non-Hijab)
	50 (Hijab)	50 (Hijab)

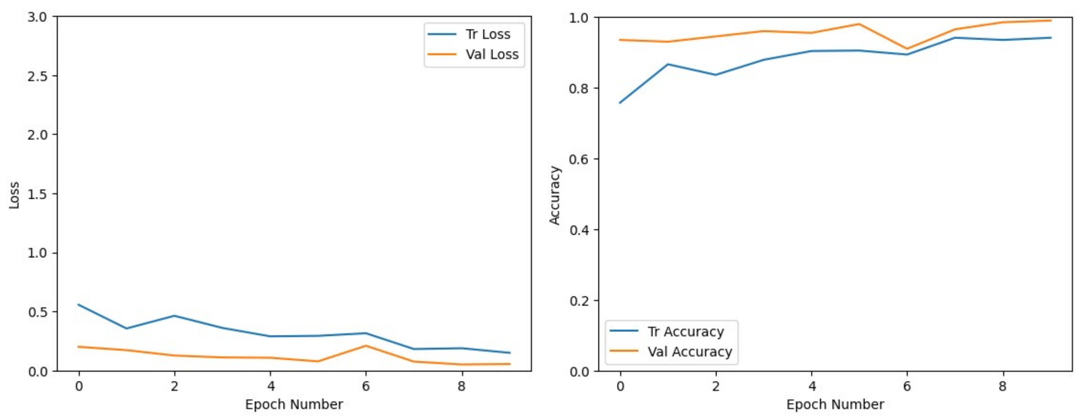
b) Model Selection and Training

After the data collection, we moved to selecting the suitable model and do training to get the best model for this dataset. The model that is chosen are ResNet18 model and transfer learning are done on the dataset collected in section (a). After loading ResNet18 model, the loss function and optimizer is initialized. The cross-entropy loss and Stochastic Gradient Descent (SGD) are used for this model training. The datasets are trained for 10 epochs and the best model out of all of them is saved as "Gender.pt" model. The code snippet of the model training is shown below.

```python
model = models.resnet18(pretrained=True)
num_ftrs = model.fc.in_features

model.fc = nn.Linear(num_ftrs, 2)

# LOSS AND OPTIMIZER
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)


# move the model to GPU
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
model.to(device)
```

```python
# Train the model for 10 epochs

num_epochs = 10
trained_model, history = train_and_validate(model, loss_fn, optimizer, trainloader, testloader, num_epochs)
```

After training the model for 10 epochs, the loss and accuracy curve are plotted and shown below.



c) Code for Model Inference and Counting

After training the model and save it into a "Gender.pt" file. The file is exported out of Google Colab and used in OpenCV or VS code for object detection and doing model inference. The snippets of the important parts of the code for model inference and counting are shown below.

```python
# Load the pretrained PyTorch model
model = torch.load('Gender.pt', map_location=device)
```

```
model.eval()
model.to(device)
```

For this line of codes, it is to load the pretrained model from Google Colab into OpenCV to do object detection. The model is sent to the device which will be utilizing the GPU of the device.

```
# Define the preprocessing transformations
preprocess = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225]),])
```

For this line of codes, it is to ensure that the input image is the same as the specification of the trained images. The image is resized to 224x224, transform the image to tensor form, and normalize it using the mean and standard deviation obtained from ImageNet datasets.

```
# Load the face detector (Haar Cascade)
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

# Load the image
image_path = 'Faces.jpg'
image = cv2.imread(image_path)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Detect faces in the image
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5,
minSize=(30, 30))
```

For this line of codes, it is to load a face detector model already available from OpenCV which is called Haar Cascade. This model will do face detection on the image and create a bounding box. The next set of codes are to read the "face.jpg" image which is the image from fig. 1.

```
    # Perform inference
    with torch.no_grad():
        outputs = model(face_tensor)
```

```
        prediction = outputs.argmax(dim=1).item()  # Get the predicted class (0 for female, 1 for male)

    # Update counters
    if prediction == 0:
        label = "Female"
        num_females += 1
    else:
        label = "Male"
        num_males += 1

    # Draw the bounding box and label on the image
    cv2.rectangle(image, (x, y), (x+w, y+h), (255, 0, 0), 2)
    cv2.putText(image, label, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)
```

For this line of codes, it is basically doing model inference where from the Haar cascade face detection, the code will inference the "Gender.pt" model onto the detected faces and do a prediction whether it is a male or a female. Then for every prediction, the counter for male and female increases.

Lastly, the output for this face detection and gender counting can be seen below.

From the output, we can see that the model detected 10 male faces and 11 female faces. Upon further inspection of the output image, we can see several mistakes.

First one is that the model detected some of the certificate as a face for example the woman on the far right holding a certificate identified as a female. The female sitting at the front also holding a certificate detected as a male. Other than that, the model failed to detect the face of a woman wearing niqab or purdah covering her whole face except her eyes. Some faces that are covered partially are also not able to be detected by the model.

This could be caused by several issues. The first one might be due to the small size of the dataset and the variety of it. There are no images of women wearing niqab for the model to train on. The datasets should be enlarged to 1000 images for the model to learn various faces of female and male. Other than that, it could be due to the lack of epochs the model is trained in. Increasing the epochs more could allow the model to learn and perfect a better model in detecting the faces. Furthermore, data augmentation could be done to combat the lack of training data. Either flipping, rotating or center crop the training images to create multiple variation of the existed images in the training dataset.

Full code for model training

```
import torch, torchvision
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import time
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
import glob
import numpy
import random

from PIL import Image
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torchvision import datasets, models, transforms
from torchsummary import summary


# Setup device-agnostic code
device = "cuda" if torch.cuda.is_available() else "cpu"
device


# Importing zipfile to unzip dataset
import zipfile
from pathlib import Path

# Setup path to data folder
data_path = Path("data/")
image_path = data_path / "Gender" #name of the folder we want to unzip
to

# If the image folder doesn't exist, create new file...
if image_path.is_dir():
    print(f"{image_path} directory exists.")
else:
    print(f"Did not find {image_path} directory, creating one...")
    image_path.mkdir(parents=True, exist_ok=True)

# Unzip Expression data
with zipfile.ZipFile("/content/data/Gender/Gender.zip", "r") as
zip_ref:
        print("Unzipping Expression data...")
        zip_ref.extractall(image_path)
```

```python
image_path = data_path / "Gender"

# Walkthrough the data to check the length of each directories
def walk_through_dir(dir_path):
  """
  Walks through dir_path returning its contents.
  Args:
    dir_path (str or pathlib.Path): target directory

  Returns:
    A print out of:
      number of subdiretories in dir_path
      number of images (files) in each subdirectory
      name of each subdirectory
  """
  for dirpath, dirnames, filenames in os.walk(dir_path):
    print(f"There are {len(dirnames)} directories and {len(filenames)}
images in '{dirpath}'.")

walk_through_dir(image_path)


# Define transformations
data_transform = transforms.Compose(
    [transforms.Resize((224,224)),
     transforms.ToTensor(),
     transforms.CenterCrop(size=224),
     transforms.Normalize([0.485, 0.456, 0.406],
                          [0.229, 0.224, 0.225])])


# Use torchvision.datasets.ImageFolder to create dataset(s)
from torchvision import datasets

# Setup train and testing paths
train_dir = image_path / "/content/data/Gender/train"
test_dir = image_path / "/content/data/Gender/test"


train_data = datasets.ImageFolder(root=train_dir, # target folder of
images
                                  transform=data_transform, #
transforms to perform on data (images)
                                  target_transform=None) # transforms
to perform on labels (if necessary)

test_data = datasets.ImageFolder(root=test_dir,
                                 transform=data_transform)

print(f"Train data:\n{train_data}\nTest data:\n{test_data}")
```

```python
# Turn train and test Datasets into DataLoaders
from torch.utils.data import DataLoader

# Create iterators for the Data loaded using DataLoader module
batchSize = 4

trainloader = DataLoader(dataset=train_data, batch_size=batchSize,
num_workers=1, shuffle=True)
testloader = DataLoader(dataset=test_data, batch_size=batchSize,
num_workers=1, shuffle=False)

train_data_size = len(trainloader.dataset)
test_data_size = len(testloader.dataset)

print(train_data_size)
print(test_data_size)


model = models.resnet18(pretrained=True)
num_ftrs = model.fc.in_features

model.fc = nn.Linear(num_ftrs, 2)

# LOSS AND OPTIMIZER
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)


# move the model to GPU
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
model.to(device)


import time
from tqdm.auto import tqdm

def train_and_validate(model, loss_criterion, optimizer,
train_dataloader, test_dataloader, epochs=25, device='cuda'):
    '''
    Function to train and validate
    Parameters
        :param model: Model to train and validate
        :param loss_criterion: Loss Criterion to minimize
        :param optimizer: Optimizer for computing gradients
        :param train_dataloader: DataLoader for training data
        :param test_dataloader: DataLoader for test/validation data
        :param epochs: Number of epochs (default=25)
        :param device: Device to perform computations ('cuda' or 'cpu')

    Returns
        model: Trained Model with best validation accuracy
```

```
        history: (dict object): Having training loss, accuracy and
validation loss, accuracy
    '''

    start = time.time()
    history = []
    best_acc = 0.0

    # accuracy = torchmetrics.Accuracy(device=device)
    # Initialize the accuracy metric from torchmetrics
    # accuracy =
torchmetrics.classification.Accuracy(task="multiclass",
num_classes=10).to(device)

    for epoch in tqdm(range(epochs)):
        epoch_start = time.time()
        print("Epoch: {}/{}".format(epoch+1, epochs))

        model.train()

        train_loss = 0.0
        train_acc = 0.0

        valid_loss = 0.0
        valid_acc = 0.0

        for i, (inputs, labels) in enumerate(train_dataloader):

            inputs = inputs.to(device)
            labels = labels.to(device)

            # Clean existing gradients
            optimizer.zero_grad()

            # Forward pass - compute outputs on input data using the
model
            outputs = model(inputs)

            # Compute loss
            loss = loss_criterion(outputs, labels)

            # Backpropagate the gradients
            loss.backward()

            # Update the parameters
            optimizer.step()

            # Compute the total loss for the batch and add it to
train_loss
            train_loss += loss.item() * inputs.size(0)
```

```
            # Compute the accuracy
            ret, predictions = torch.max(outputs.data, 1)
            correct_counts =
predictions.eq(labels.data.view_as(predictions))

            # Convert correct_counts to float and then compute the mean
            acc = torch.mean(correct_counts.type(torch.FloatTensor))

            # Compute total accuracy in the whole batch and add to
train_acc
            train_acc += acc.item() * inputs.size(0)

        # Validation - No gradient tracking needed
        with torch.no_grad():

            model.eval()

            # Validation loop
            for j, (inputs, labels) in enumerate(test_dataloader):
                inputs = inputs.to(device)
                labels = labels.to(device)

                # Forward pass - compute outputs on input data using
the model
                outputs = model(inputs)

                # Compute loss
                loss = loss_criterion(outputs, labels)

                # Compute the total loss for the batch and add it to
valid_loss
                valid_loss += loss.item() * inputs.size(0)

                # Calculate validation accuracy
                ret, predictions = torch.max(outputs.data, 1)
                correct_counts =
predictions.eq(labels.data.view_as(predictions))

                # Convert correct_counts to float and then compute the
mean
                acc =
torch.mean(correct_counts.type(torch.FloatTensor))

                # Compute total accuracy in the whole batch and add to
valid_acc
                valid_acc += acc.item() * inputs.size(0)


        # Find average training loss and training accuracy
        avg_train_loss = train_loss / len(train_dataloader.dataset)
        avg_train_acc = train_acc / len(train_dataloader.dataset)
```

```
        # Find average validation loss and training accuracy
        avg_test_loss = valid_loss / len(test_dataloader.dataset)
        avg_test_acc = valid_acc / len(test_dataloader.dataset)

        history.append([avg_train_loss, avg_test_loss, avg_train_acc,
avg_test_acc])

        epoch_end = time.time()

        print("Epoch : {:03d}, Training: Loss: {:.4f}, Accuracy:
{:.4f}%, \n\t\tValidation : Loss : {:.4f}, Accuracy: {:.4f}%, Time:
{:.4f}s".format(epoch, avg_train_loss, avg_train_acc * 100,
avg_test_loss, avg_test_acc * 100, epoch_end - epoch_start))

        # Save if the model has best accuracy till now
        if avg_test_acc > best_acc:
            best_acc = avg_test_acc
            best_model = model
            torch.save(best_model, 'best_model.pt')

    return best_model, history


# Train the model for 10 epochs

num_epochs = 10
trained_model, history = train_and_validate(model, loss_fn, optimizer,
trainloader, testloader, num_epochs)


# 5. Analyze the loss curve

history = np.array(history)
plt.plot(history[:,0:2])
plt.legend(['Tr Loss', 'Val Loss'])
plt.xlabel('Epoch Number')
plt.ylabel('Loss')
plt.ylim(0,3)
# plt.savefig('Gender_loss_curve.png')
plt.show()

# 6. Analyze the accuracy curve

plt.plot(history[:,2:4])
plt.legend(['Tr Accuracy', 'Val Accuracy'])
plt.xlabel('Epoch Number')
plt.ylabel('Accuracy')
plt.ylim(0,1)
# plt.savefig('Gender_accuracy_curve.png')
plt.show()
```

Full code for model Inference and Counting

```
import torch
import cv2
import numpy as np
from PIL import Image
from torchvision import transforms

# Check if CUDA is available and set device accordingly
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Load the pretrained PyTorch model
model = torch.load('Gender.pt', map_location=device)
model.eval()
model.to(device)

# Define the preprocessing transformations
preprocess = transforms.Compose([
    transforms.Resize((224, 224)),  # Adjust according to your model's input
size
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225]),
])

# Load the face detector (Haar Cascade)
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

# Load the image
image_path = 'Faces.jpg'
image = cv2.imread(image_path)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Detect faces in the image
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5,
minSize=(30, 30))

num_males = 0
num_females = 0

for (x, y, w, h) in faces:
    # Extract the face region
    face = image[y:y+h, x:x+w]
    face_rgb = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)

    # Convert the face to PIL Image
    face_pil = Image.fromarray(face_rgb)

    # Preprocess the face
    face_tensor = preprocess(face_pil)
```

```
    face_tensor = face_tensor.unsqueeze(0)  # Add batch dimension
    face_tensor = face_tensor.to(device)  # Move tensor to the correct device

    # Perform inference
    with torch.no_grad():
        outputs = model(face_tensor)
        prediction = outputs.argmax(dim=1).item()  # Get the predicted class
(0 for male, 1 for female)

    # Update counters
    if prediction == 0:
        label = "Female"
        num_females += 1
    else:
        label = "Male"
        num_males += 1

    # Draw the bounding box and label on the image
    cv2.rectangle(image, (x, y), (x+w, y+h), (255, 0, 0), 2)
    cv2.putText(image, label, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255,
0, 0), 2)

# Display the counter on the top left corner
cv2.putText(image, f'Males: {num_males}  Females: {num_females}', (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

# Save the output image
output_image_path = 'output_image.jpg'
cv2.imwrite(output_image_path, image)

# Resize the window to fit the image
cv2.namedWindow('Image', cv2.WINDOW_NORMAL)
cv2.imshow('Image', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```