ELEC4630 ASSIGNMENT 2

HENGJI ZHAO

47184521

Question 1: The Fingerprint Recognition

The basic recognition part code was provided by the Professor's Github sources, which were contained in the fingerprint repository. After I read and run the code, to simplify the process of the comparsion, I write the key part into one function which is "compare_function" is the following part. Then I will introduce my code following in details.

In [287…
```
"""
Check whether the given file 'utils.py' is in the current directory.
The file "utils.py" contains many useful function for showing the
status of fingerprint recognition.
Though the question do not require us to do the fingerprint show,
it can help us if we have problems of the written code
"""
from os import path
if not path.exists('utils.py'):
    # If running on colab: the first time download and unzip additional files
    !wget https://biolab.csr.unibo.it/samples/fr/files.zip
    !unzip files.zip
```

In [288…
```
# Import the required libraries for this question.
# Run utils.py for helper functions
import utils
import math
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
from utils import *
from ipywidgets import interact
```

In [289…
```
# Also, import the required libraries, such as "tkinter"
# which is necessary for the GUI building.
import utils

from tkinter import *
from utils import *
import numpy as np
import cv2 as cv
import math
import matplotlib.pyplot as plt
from ipywidgets import interact
import os

import time
import json
from PIL import ImageTk, Image
from tkinter import filedialog, messagebox
```

In [290…
```
"""
We define a read image function, in this way we can read
images easier.
```

```
"""
def read_image(address):
    fingerprint = cv.imread(address, cv.IMREAD_GRAYSCALE)
    return fingerprint
```

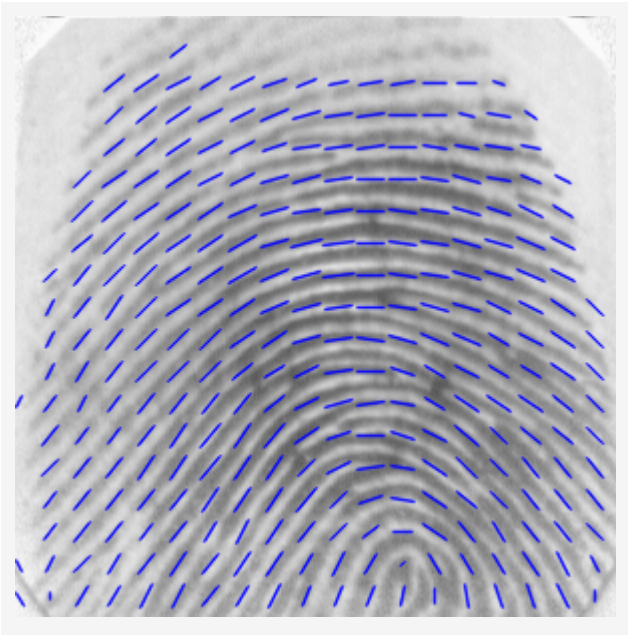
Estimate of local ridge orientation

In [291…

```
"""
In this function, we combine all the code in example about
how to estimate the orientation about the local ridge orientation.

I hope we can get the result we want from the address of the image
directly, this can make our work easier.
"""
def estimate_ridge_orientation(address, thr_value=0.25):

    # Read the image by using the address.
    fingerprint = read_image(address)
    gx = cv.Sobel(fingerprint, cv.CV_32F, 1, 0)
    gy = cv.Sobel(fingerprint, cv.CV_32F, 0, 1)

    gx2, gy2 = gx**2, gy**2
    gm = np.sqrt(gx2 + gy2)

    w = (25, 25)
    sum_gm = cv.boxFilter(gm, -1, w, normalize=False)

    thr = sum_gm.max() * thr_value
    mask = cv.threshold(sum_gm, thr, 255, cv.THRESH_BINARY)[1].astype(np.uint8)

    W = (23, 23)
    gxx = cv.boxFilter(gx2, -1, W, normalize = False)
    gyy = cv.boxFilter(gy2, -1, W, normalize = False)
    gxy = cv.boxFilter(gx * gy, -1, W, normalize = False)
    gxx_gyy = gxx - gyy
    gxy2 = 2 * gxy

    # '-' to adjust for y axis direction
    orientations = (cv.phase(gxx_gyy, -gxy2) + np.pi) / 2
    sum_gxx_gyy = gxx + gyy

    strengths = np.divide(cv.sqrt((gxx_gyy**2 + gxy2**2)),
                          sum_gxx_gyy,
                          out=np.zeros_like(gxx),
                          where=sum_gxx_gyy!=0)

    # We return the corresponding fingerprint image, the
    # orientations, strengths and masks for the following code.
    return fingerprint, orientations, strengths, mask


# We take the fingerprint 101_1.tif as an example to
# check whether the function can work well.
# We take the 101_1.tif in the folder as the template
# for the result checking.
add1 = 'DB1_B/101_1.tif'
fingerprint, a2, a3, a4 = estimate_ridge_orientation(add1)
```

```
show(draw_orientations(fingerprint, a2, a3, a4, 1, 16), 'Orientation image')
```

Orientation image



Estimation of local ridge frequency

In the example, we selected the specific place to do the frequency analysis, in our question code, to make the answer more accuracy, we can do the analysis for the whole image.

In [292…

```
# We put all the code for estimate the ridge frequency
# together to make our comparsion easier, now we can get the
# answer we want from its address directly.
def estimate_ridge_frequency(address):

    fingerprint = read_image(address)

    region = fingerprint

    # Smooth the region to reduce the noise.
    smoothed = cv.blur(region, (5,5), -1)
    xs = np.sum(smoothed, 1)

    # Find the indices of the x-signature local maxima
    local_maxima = np.nonzero(np.r_[False, xs[1:] > xs[:-1]]
                              & np.r_[xs[:-1] >= xs[1:], False])[0]

    # Calculate all the distances between consecutive peaks
    distances = local_maxima[1:] - local_maxima[:-1]

    # Estimate the ridge line period as the average of the above distances
    ridge_period = np.average(distances)

    # Return the ridge line period.
    # Then we can get the ridge line period from the address of image.
    return ridge_period
```

```
# We try the function and test it.
value = estimate_ridge_frequency(add1)
print(value)
```

```
12.428571428571429
```

Fingerprint enhancement

Also, we want to get the enhanced image directly from the address, all the things is that we want to simplify this process of comparsion.

In [293…

```
def finger_enhancement(address, or_count=8):

    fingerprint, orientations, _, mask = estimate_ridge_orientation(address)

    ridge_period = estimate_ridge_frequency(address)

    # Create the filter bank.
    gabor_bank = [gabor_kernel(ridge_period, o)
                  for o in np.arange(0, np.pi, np.pi/or_count)]

    # Filter the whole image with each filter
    # Note that the negative image is actually used,
    # to have white ridges on a black background as a result
    nf = 255-fingerprint
    all_filtered = np.array([cv.filter2D(nf, cv.CV_32F, f)
                             for f in gabor_bank])

    y_coords, x_coords = np.indices(fingerprint.shape)

    # For each pixel, find the index of the closest
    # orientation in the gabor bank
    orientation_idx = np.round(((orientations % np.pi) / np.pi)
                               * or_count).astype(np.int32) % or_count

    # Take the corresponding convolution result for each pixel,
    # to assemble the final result
    filtered = all_filtered[orientation_idx, y_coords, x_coords]

    # Convert to gray scale and apply the mask
    enhanced = mask & np.clip(filtered, 0, 255).astype(np.uint8)

    return enhanced

# Check whether the function can work normally.
# we find it can work well.
enhanced = finger_enhancement(add1)
show(fingerprint, enhanced)
```

Detection of minutiae positions

For this section, it is nort easy to put all the code together, so we need to divide them into different functions and then combine together.

In [294…

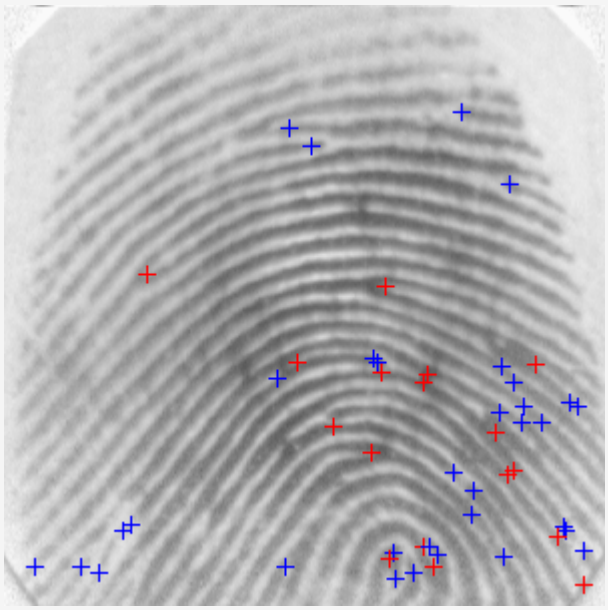
```
def thinning_process(address):
    fingerprint = read_image(address)
    enhanced = finger_enhancement(address)

    # Binarization
    _, ridge_lines = cv.threshold(enhanced, 32, 255, cv.THRESH_BINARY)

    # Thinning
    skeleton=cv.ximgproc.thinning(ridge_lines,
                                  thinningType=cv.ximgproc.THINNING_GUOHALL)

    return skeleton

ske = thinning_process(add1)
show(ske)
```

In [295…
```python
def compute_crossing_number(values):
    return np.count_nonzero(values < np.roll(values, -1))
```

In [296…
```python
# Create a filter that converts any 8-neighborhood into the
# corresponding byte value [0,255]
cn_filter = np.array([[  1,  2,  4],
                      [128,  0,  8],
                      [ 64, 32, 16]
                     ])

# Create a lookup table that maps each byte value to the
# corresponding crossing number
all_8_neighborhoods = [
    np.array([int(d) for d in f'{x:08b}'])[::-1]
    for x in range(256)]

cn_lut = np.array(
    [compute_crossing_number(x)
     for x in all_8_neighborhoods]).astype(np.uint8)
```

In [297…
```python
def detect_minutiae_position(address):

    global cn_lut

    # Skeleton: from 0/255 to 0/1 values
    skeleton = thinning_process(address)
    skeleton01 = np.where(skeleton!=0, 1, 0).astype(np.uint8)

    # Apply the filter to encode the 8-neighborhood
    # of each pixel into a byte [0,255]
    neighborhood_values = cv.filter2D(skeleton01,
                                      -1,
                                      cn_filter,
                                      borderType = cv.BORDER_CONSTANT)

    # Apply the lookup table to obtain the crossing
    # number of each pixel from the byte value of
    # its neighborhood
    cn = cv.LUT(neighborhood_values, cn_lut)
```

```
    # Keep only crossing numbers on the skeleton
    cn[skeleton==0] = 0

    # crossing number == 1 --> Termination,
    # crossing number == 3 --> Bifurcation
    minutiae = [(x,y,cn[y,x]==1)
                for y, x in zip(*np.where(np.isin(cn, [1,3])))]

    # The virable of mask is required for this question,
    # Then we use the code we written previously to
    # do it.
    _, _, _, mask = estimate_ridge_orientation(address)

    # A 1-pixel background border is added to the mask
    # before computing the distance transform
    mask_distance = cv.distanceTransform(
        cv.copyMakeBorder(mask, 1, 1, 1, 1, cv.BORDER_CONSTANT),
        cv.DIST_C, 3)[1:-1, 1:-1]

    filtered_minutiae = list(
        filter(lambda m: mask_distance[m[1], m[0]]>10, minutiae))

    return filtered_minutiae, neighborhood_values, cn

# Chenk whether our function can work well.
minu, _, _ = detect_minutiae_position(add1)
show(draw_minutiae(fingerprint, minu))
```



Estimation of minutiae directions

In [298…

```
def compute_next_ridge_following_directions(previous_direction, values):
    next_positions = np.argwhere(values!=0).ravel().tolist()
    if len(next_positions) > 0 and previous_direction != 8:

        # There is a previous direction: return all the next directions,
        # sorted according to the distance from it,
        # except the direction, if any, that corresponds to the
```

```
        # previous position
        next_positions.sort(key=lambda d: 4-abs(abs(d-previous_direction)-4))

        # the direction of the previous position is the opposite one
        if next_positions[-1] == (previous_direction + 4) % 8:
            next_positions = next_positions[:-1] # removes it
    return next_positions
```

In [299…

```
r2 = 2**0.5 # sqrt(2)

# The eight possible (x, y) offsets with each
# corresponding Euclidean distance
xy_steps = [(-1,-1,r2),( 0,-1,1),( 1,-1,r2),
            ( 1, 0,1),( 1, 1,r2),( 0, 1,1),
            (-1, 1,r2),(-1, 0,1)]

# LUT: for each 8-neighborhood and each previous direction [0,8],
# where 8 means "none", provides the list of possible directions
nd_lut = [[compute_next_ridge_following_directions(pd, x)
           for pd in range(9)]
           for x in all_8_neighborhoods]
```

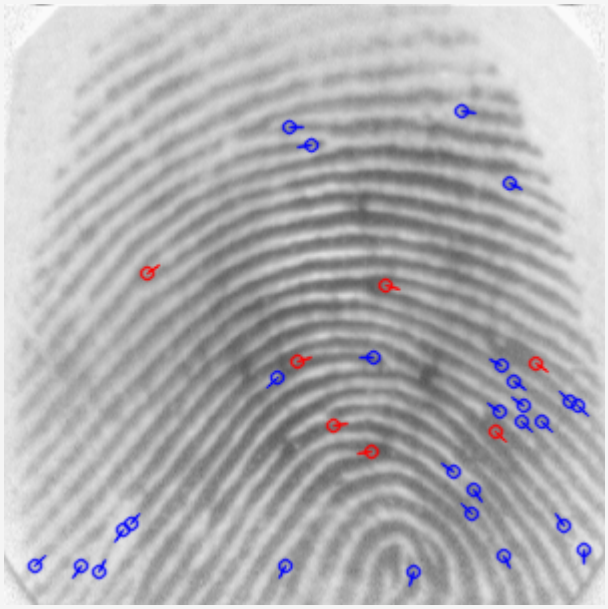
In [300…

```
def follow_ridge_and_compute_angle(neighborhood_values, cn, x, y, d = 8):
    px, py = x, y
    length = 0.0
    while length < 20: # max length followed
        next_directions = nd_lut[neighborhood_values[py,px]][d]
        if len(next_directions) == 0:
            break
        # Need to check ALL possible next directions
        if (any(cn[py + xy_steps[nd][1],
                   px + xy_steps[nd][0]] != 2
                   for nd in next_directions)):
            break # another minutia found: we stop here
        # Only the first direction has to be followed
        d = next_directions[0]
        ox, oy, l = xy_steps[d]
        px += ox ; py += oy ; length += l

    # check if the minimum length for a valid direction
    # has been reached
    return math.atan2(-py+y, px-x) if length >= 10 else None
```

In [301…

```
def detect_minutiae_direction(address):

    global nd_lut

    valid_minutiae = []

    # The the previously built function for necessary
    # parameters used in this function.
    filtered_minutiae,neighborhood_values,cn=detect_minutiae_position(address)

    for x, y, term in filtered_minutiae:
        d = None

        # termination: simply follow and compute the direction
        if term:
            d = follow_ridge_and_compute_angle(neighborhood_values, cn, x, y)
```

```
        # bifurcation: follow each of the three branches
        else:
            dirs = nd_lut[neighborhood_values[y,x]][8]
            if len(dirs)==3:

                angles = [follow_ridge_and_compute_angle(
                    neighborhood_values,
                    cn, x+xy_steps[d][0],
                    y+xy_steps[d][1], d)
                    for d in dirs]

                if all(a is not None for a in angles):
                    a1, a2 = min(((angles[i], angles[(i+1)%3])
                                  for i in range(3)),
                                  key=lambda t: angle_abs_difference(t[0],t[1]))
                    d = angle_mean(a1, a2)
        if d is not None:
            valid_minutiae.append( (x, y, term, d) )

    return valid_minutiae

# Check the result of our work for building function
# to do minutiae direction
va = detect_minutiae_direction(add1)
show(draw_minutiae(fingerprint, va))
```



In [302…

```
mcc_sigma_s = 7.0
mcc_tau_psi = 400.0
mcc_mu_psi = 1e-2

def Gs(t_sqr):
    """
    Gaussian function with zero mean and
    mcc_sigma_s standard deviation,
    see eq. (7) in MCC paper
    """
    return np.exp(-0.5 * t_sqr /
                  (mcc_sigma_s**2)) / (math.tau**0.5 * mcc_sigma_s)
```

```
def Psi(v):
    """
    Sigmoid function that limits the contribution
    of dense minutiae clusters,
    see eq. (4)-(5) in MCC paper
    """
    return 1. / (1. + np.exp(-mcc_tau_psi * (v - mcc_mu_psi)))
```

Creation of local structure.

In [303…

```
def create_local_structure(address):

    # Compute the cell coordinates of a generic local structure
    mcc_radius = 70
    mcc_size = 16

    g = 2 * mcc_radius / mcc_size
    x = np.arange(mcc_size)*g - (mcc_size/2)*g + g/2
    y = x[..., np.newaxis]
    iy, ix = np.nonzero(x**2 + y**2 <= mcc_radius**2)
    ref_cell_coords = np.column_stack((x[ix], x[iy]))

    valid_minutiae = detect_minutiae_direction(address)

    # n: number of minutiae
    # c: number of cells in a local structure
    # matrix with all minutiae coordinates and directions (n x 3)
    xyd = np.array([(x,y,d)
                    for x,y,_,d in valid_minutiae])

    d_cos = np.cos(xyd[:,2]).reshape((-1,1,1))
    d_sin = np.sin(xyd[:,2]).reshape((-1,1,1))
    rot = np.block([[d_cos, d_sin], [-d_sin, d_cos]])

    # rot@ref_cell_coords.T : n x 2 x c
    # xy : n x 2
    xy = xyd[:,:2]

    # cell_coords: n x c x 2 (cell coordinates for each local structure)
    cell_coords = np.transpose(rot@ref_cell_coords.T + xy[:,:,np.newaxis],
                               [0,2,1])
    dists = np.sum((cell_coords[:,:,np.newaxis,:] - xy)**2, -1)

    # cs : n x c x n (the spatial contribution of each
    # minutia to each cell of each local structure)
    cs = Gs(dists)
    diag_indices = np.arange(cs.shape[0])

    # remove the contribution of each minutia to its own cells
    cs[diag_indices,:,diag_indices] = 0

    # local_structures : n x c (cell values for each local structure)
    local_structures = Psi(np.sum(cs, -1))

    return local_structures, ref_cell_coords

# Check the result for the function.
loc, ref = create_local_structure(add1)
```

```
@interact(i=(0,len(va)-1))
def test(i=0):
    show(draw_minutiae_and_cylinder(fingerprint, ref, va, loc, i))
```

```
interactive(children=(IntSlider(value=0, description='i', max=33), Output()), _do
m_classes=('widget-interact',…
```

The comparsion part

Then we need to combine all the function as a single one, this function can return all the data we need for the comparsion and just use the address for the input.

The parameters we need for comparsion are the fingerprint, valid minutiae and the structure. Then we can use the functions that can produce these data into one function. We name this function as "quick_answer".

In [304…
```
def quick_answer(address):

    f1, _, _, _ = estimate_ridge_orientation(address)
    m1 = detect_minutiae_direction(address)
    ls1, ref = create_local_structure(address)

    return f1, m1, ls1, ref
```

In [305…
```
f1, m1, ls1, ref_cell_coords = quick_answer(add1)
```

This is the comparsion function. Everytime when we need to input a new fingerprint, we can use the function "quick_answer" to get the corresponding data for comparsion. And then we can do the comparsion and get a score for this comparsion. The score for the comparsion can be regarded as how similar between these two images.

I find that the for the example code, when the example code do the comparsion, it loaded the fingerprint to be compared from a file, and the file contained the valid minutiae and local structure. All these data can be achieved from the "quick_answer" function.

The similarity between two local structures r_1 and r_2 can be simply computed as one minus their normalized Euclidean distance:

$$similarity(r_1, r_2) = 1 - \frac{\|r_1 - r_2\|}{\|r_1\| + \|r_2\|}$$

In [306…
```
# Select the 6 pairs with the smallest distances
# (LSS technique)
def compare_function(address, num_p=6):
    global f1, m1, ls1, ref_cell_coords

    f2, m2, ls2, _, = quick_answer(address)

    # Compute all pairwise normalized Euclidean distances between
    # local structures in v1 and v2
```

```
    # ls1                       : n1 x  c
    # ls1[:,np.newaxis,:]       : n1 x  1 x c
    # ls2                       : (1 x) n2 x c
    # ls1[:,np.newaxis,:] - ls2 : n1 x  n2 x c
    # dists                     : n1 x  n2
    dists = np.linalg.norm(ls1[:,np.newaxis,:] - ls2, axis = -1)
    dists /= np.linalg.norm(ls1,
                            axis = 1)[:,np.newaxis] + np.linalg.norm(ls2,
                                                                     axis = 1)

    pairs = np.unravel_index(np.argpartition(dists, num_p, None)
                             [:num_p],
                             dists.shape)
    score = 1 - np.mean(dists[pairs[0], pairs[1]]) # See eq. (23) in MCC paper

    return f2, m2, ls2, pairs, score

add2 = 'DB1_B/104_2.tif'
print(add1)
f2, m2, ls2, pairs, score = compare_function(add2)

@interact(i = (0,len(pairs[0])-1), show_local_structures = False)
def show_pairs(i=0, show_local_structures = False):
    show(draw_match_pairs(f1, m1, ls1, f2, m2, ls2,
                          ref_cell_coords, pairs, i,
                          show_local_structures))
```

```
DB1_B/101_1.tif
interactive(children=(IntSlider(value=0, description='i', max=5), Checkbox(value=
False, description='show_loca…
```

GUI Building

Then we Enter the step for the GUI building part. We can firstly build set the basic template for the comparsion, this make us can do the comparsion directly after we open the GUI.

In our GUI, we can select any fingerprint from the folder and then do the comparsion with all the fingerprints in the Folder.

In [307…

```
# Set DB1_B/101_1.tif as the template
# The templete image can be chaneged by using the GUI.
template_address = 'DB1_B/101_1.tif'
f1, m1, ls1, ref_cell_coords = quick_answer(template_address)
```

The template fingerprint will be compared to all the fingerprints in the folder.

In [308…

```
# Import some useful libraries for the GUI and the
# involved calculation.
import pickle
from itertools import combinations
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve

# Building the GUI interface and set the title
# and its size.
```

```
face = Tk()
face.geometry("600x400")
face.title("Fingerprint Recognition")

# In this part, we can enter the name assigned for
# the fingerprint we want to save.
File_name = Label(face, text="File name: ")
File_name.place(x=5, y=5)

# We can enter the name in this entry.
name_entry = Entry(face)
name_entry.place(x=75, y=5)

# In this place we can set the threshold for the
# comparsion, if use the returned value "score" as the
# judgement.
threshold_tag = Label(face, text="Threshold: ")
threshold_tag.place(x=300, y=5)
threshold_ent = Entry(face)
threshold_ent.place(x=380, y=5)
threshold_ent.insert(0, "0.66")


"""
Just as we talked, the template fingerprint will be compared
with all the fingerprints in the DB1_B folder, then the fault
ratio will be written in this Entry, the logic for judgeing if
the comparsion is success will be talked later.
"""
fault_ratio = Label(face, text="Fault Ratio: ")
fault_ratio.place(x=300, y=50)
fault_ratio_display = Entry(face)
fault_ratio_display.place(x=380, y=50)


"""
This place we set is to tell the operator of the GUI which
fingerprint image is using as the template for comparsion.
"""
current_template = Label(face, text='Fingerprint:')
current_template.place(x=300, y=95)
current_template_display = Entry(face)
current_template_display.place(x=380, y=95)


"""
Initially, we set the fingerprint image '101_2.tif' in the
DB1_1 folder as the template. So, we insert the current
template information in the Entry.
"""
current_template_display.insert(0, '101_2.tif')


# We take the image 101_1.tif as the template.
template_address = "DB1_B/101_2.tif"
# The rule of naming the fingerprint image is like "10x_y",
# Where the '10x' means if the fingerprint image is belong
# to the same person, and the following parameter 'y' means
# this is the yth image of the person.
temp = "101"
```

```
f1, m1, ls1, ref_cell_coords = quick_answer(template_address)

# This is the initial data for saving function, if we change
# the template information, the data inside the template_data
# will be changed as well.
template_data = {"f1": f1, "m1": m1, "ls1": ls1,
        "ref_cell_coords": ref_cell_coords,
        "temp": temp,
        "temp_name": "101_2.tif"}


"""
Here is the saving function, which will save the fingerprint
data as a ".pkl" file, after we successfully saved the information,
there will show a window to tell operators it finished.
"""
def save_function():

    global name_entry, template_data

    # If we do not set a name for saving, it will remind the operators.
    if not name_entry.get():
        messagebox.showinfo("Fingerprint", "No available Name!")
    else:
        string1 = name_entry.get() + "_fingerprint.pkl"

        with open(string1, "wb") as f:
            pickle.dump(template_data, f)

        messagebox.showinfo("Fingerprint", f"{string1} saved!")

# We create these arraies to record the results of the comparsion
scores = [] # This array used to record the scores for each comparsion
ff_array = [] # Record the result of comparsion, true or false.
thresholds = None # Use for recording thresholds after roc_curve.
currentTemplate = '101_2'
ans = [] # Same as the usage of ff_array, but used in other function.


"""
These two arraries are used to record the thresholds and the fault
ratio for drawing the "Threshold vs Error Rate" curve.
"""
thres_xaxis = []
fault_ratio_yaxis = []

# Used for recording the lowest threshold value after the roc_curve.
# The roc_curve will give a list of thresholds value, we start to do
# the analysis from the lowest one.
record = None

"""
The clear function, every time when we do a new comparsion or analysis,
we need to clear the arries which contains the results of former
comparsion and analysis.
"""
def clear_function():

    global scores, ff_array
```

```
    scores.clear()
    ff_array.clear()

"""
This function is used for calculating the fault ratio
after we do all the analysis, I record all the results
of comparsion in an array, true (1) and false (0).
Use the number of 0 to divide the length of the array
then we can get the fault ratio.
"""
def fault_ratio_calculation(arr):

    total = len(arr)
    zeros = arr.count(0)

    ratio = zeros / total

    # Print our the fault after every comparsion.
    # Because if we want to draw the threshold vs
    # fault ratio, it will take a long time.
    print(f"Fault ratio: {ratio}")
    return ratio


"""
If we want to enroll a new fingerprint to compare all the
fingerprints in the DB1_B. We use this function.
"""
def change_template_fingerprint():

    global f1, m1, ls1, ref_cell_coords, temp

    global current_template_display

    global template_data

    # Select the fingerprint from the DB1_B folder.
    filepath = filedialog.askopenfilename(
        title="Select Template Fingerprint",
        filetypes=[("TIF Image", "*.tif"),
                   ("All Files", "*.*")]
    )
    if not filepath:
        return

    # Get the filename in the form like "10x_y.tif"
    new_temp_name = os.path.basename(filepath)

    # We use this to know this fingerprint belong to
    # who, will get a result like "10x".
    temp = new_temp_name.split('_')[0]

    # Print the new template fingerprint's name on
    # the entry, this will remind the operator which
    # fingerprint is comparsing.
    current_template_display.delete(0, END)
    current_template_display.insert(0, new_temp_name)

    # Use the "quick_answer" function to get the answers
    # for the new fingerprint template.
```

```
    f1, m1, ls1, ref_cell_coords = quick_answer(filepath)

    # Form the new fingerprint "template_data"
    template_data = {"f1": f1, "m1": m1, "ls1": ls1,
                     "ref_cell_coords": ref_cell_coords,
                     "temp": temp,
                     "temp_name": new_temp_name}

    # Remind the operator the system finished the fingerprint
    # template change.
    print("NEW FINGERPRINT ENROLLED!")
    print(f"NEW TEMP: {new_temp_name}")


"""
This function we will do the load and compare function, it will
use the template fingerprint to do the comparsion with all the
images inside the DB1_B folder. Then it will get the fault ration
for this comparsion, and generate the roc curve "FPR VS TPR" and
get the threshold values from the roc_curve function.
"""
def load_function():

    global scores, ff_array, temp

    global threshold_ent, fault_ratio_display

    global thresholds, record

    # Prepare for this comparsion.
    clear_function()

    # Select the folder which contains the images we want to do
    # the comparsion.
    folder_path = filedialog.askdirectory(title="Select A Folder")
    if not folder_path:
        messagebox.showinfo("Fingerprint", "Please Select A Folder!")
        return
    else:
        messagebox.showinfo("Fingerprint", "Start Recognitising!")

    # This is set to remind us if we forget to set the threshold value.
    if not threshold_ent.get():
        messagebox.showinfo("Fingerprint",
                            "Please Set a threshold value!")
        return
    else:
        ent = float(threshold_ent.get())

    # Our images of fingerprint are all in tif form, so we browse
    # all the tif files in the folder.
    tif_files = sorted([f for f in os.listdir(folder_path)
                        if f.endswith(".tif")])

    # Compare the fingerprint images one by one.
    for filename in tif_files:
        filepath = os.path.join(folder_path, filename)
        file_id = filename.split("_")[0]

        # Use the compare function we previouslt built.
```

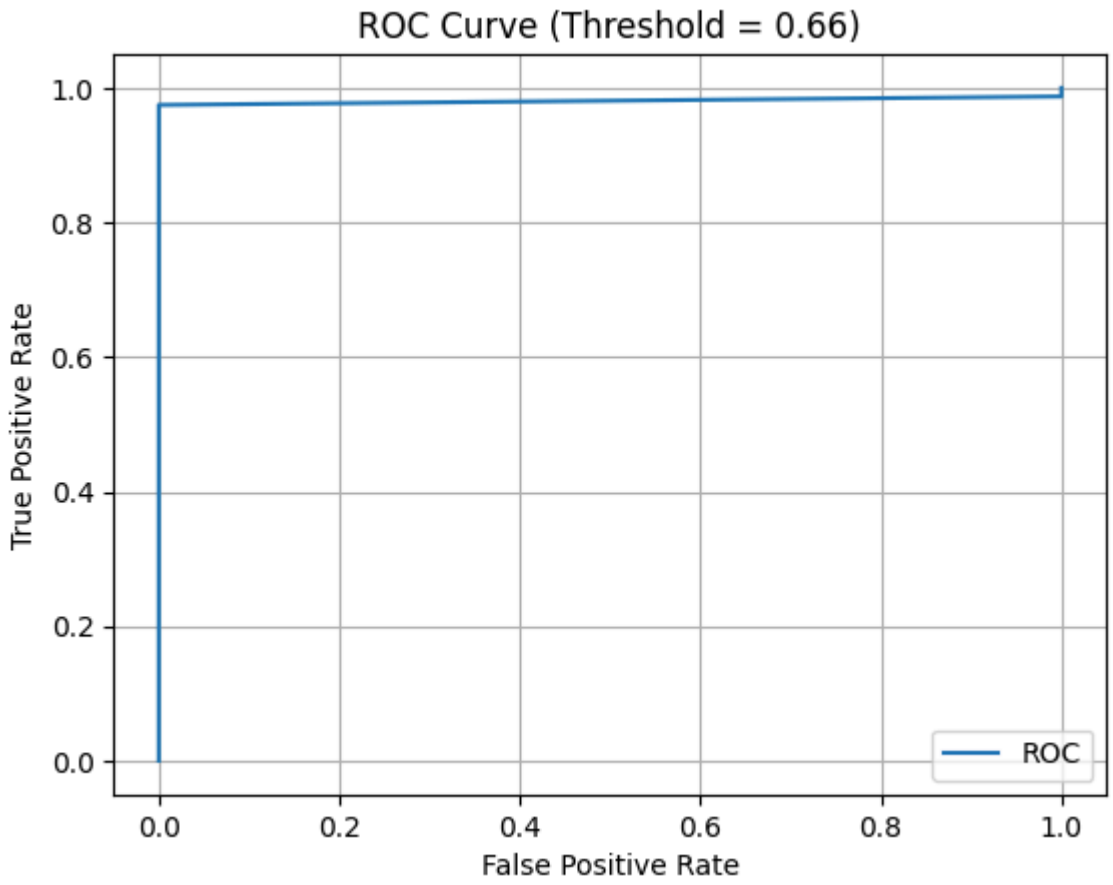
```
        _, _, _, _, score_result = compare_function(filepath)

        score_result = round(score_result, 2)

        # If higher than or equal to the threshold value, we
        # say this are the same person. Record as Positive (1),
        # otherwise record as negative (0).
        if score_result >= ent:
            system_judging = 1
        else:
            system_judging = 0

        # According to the file name, judging whether it is
        # the same person. This is using the temp variable.
        if temp == file_id:
            actual_situation = 1
        else:
            actual_situation = 0

        """
        If judging correctly:
        (Fingerprint belong to same person     -> positive)
        (Fingerprint not belong to same person -> negative)
        we regard these two situations as 1 in our model.
        otherwise set as 0.
        """
        if system_judging == actual_situation:
            ff_array.append(1)
        else:
            ff_array.append(0)

        scores.append(score_result)

    # Calculating the fault ratio.
    fault = fault_ratio_calculation(ff_array)
    fault = fault * 100
    fault_string = str(fault) + "%"

    # Insert the fault ratio to the entry we previously
    # created. If the fault ratio entry already has content inside.
    # We delete the content and then inset.
    if fault_ratio_display.get():
        fault_ratio_display.delete(0, END)
    fault_ratio_display.insert(0, fault_string)

    # Transform the list to array, this is because the function
    # roc_curve need the input data in np.array type.
    ff1 = np.array(ff_array)
    s1 = np.array(scores)

    # Do the roc_curve function for generating the fpr, tpr value
    # for generating the roc curve.
    fpr, tpr, thresholds = roc_curve(ff1, s1)
    print(thresholds)

    # The "record" variable is used to record lowest threshold value
    # of the roc_curve result.
    record = thresholds[-1]

    # After we finished the data calculating, we can start to draw
```

```
    # the figure for us, the details will be shown in the operation
    # session.
    plt.plot(fpr, tpr, label="ROC")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")

    roc = "ROC Curve (Threshold = " + str(ent) + ")"
    plt.title(roc)
    plt.grid(True)
    plt.legend()
    plt.show()


"""
This function can be used to draw the thresholds VS Error Rate.

IMPORTANT: Before we run this function, we need to run the load_funtion
           That is: type the button "LOAD FOLDER & COMPARE" every time
           before we want to run this function.

           The aim of typing "LOAD FOLDER & COMPARE" is to get the
           thresholds array to let our system know from which threshold
           to do the calculation.
"""
def threshold_vs_error_rate():
    global thresholds, temp, ans, record
    global thres_xaxis, fault_ratio_yaxis

    # These two functions are used to record the x-axis (threshold)
    # and the y-axis (fault ratio) data.
    thres_xaxis.clear()
    fault_ratio_yaxis.clear()

    # Our threshold VS fault ratio curve starts from the lowest value
    # of the thresholds array generated by roc_curve function in
    # load_function we built.

    # This is to calculate how many thresholds values we need to
    # calulate.
    number = 100 * (1 - record) + 1

    folder_path = filedialog.askdirectory(title="Select A Folder")
    if not folder_path:
        messagebox.showinfo("Fingerprint", "Please Select A Folder!")
        return
    else:
        messagebox.showinfo("Fingerprint", "Start Recognitising!")

    tif_files = sorted([f for f in os.listdir(folder_path)
                        if f.endswith(".tif")])

    threshold = thresholds[-1] - 0.01

    # Clean the axis arraies before the comparsion.
    thres_xaxis.clear()
    fault_ratio_yaxis.clear()

    # Start the comparsion.
    for i in range(int(number)):
```

```
        # The array to record the result of comparsion.
        ans.clear()

        # Threshold value for current comparsion.
        threshold = threshold + 0.01
        threshold = round(threshold, 2)

        # Record the threshold and add it into the threshold array.
        thres_xaxis.append(threshold)

        # Print the current process, tell operator current threshold.
        print("----------------------------------------")
        print(i, threshold)

        # These part is almost same as the part in load_function.
        for filename in tif_files:
            filepath = os.path.join(folder_path, filename)
            file_id = filename.split("_")[0]

            _, _, _, _, scoree = compare_function(filepath)
            scoree = round(scoree, 2)

            if scoree >= threshold:
                system_judge2 = 1
            else:
                system_judge2 = 0

            if temp == file_id:
                actual = 1
            else:
                actual = 0

            if system_judge2 == actual:
                ans.append(1)
            else:
                ans.append(0)

        # Record the current comparsion's fault ratio.
        faul = fault_ratio_calculation(ans)
        fault_ratio_yaxis.append(faul)

    # Draw the threshold VS fault ratio curve.
    plt.plot(thres_xaxis,
             fault_ratio_yaxis,
             label='Fault Ratio Curve')
    plt.title("Threshold VS Fault Ratio")
    plt.xlabel("Threshold")
    plt.ylabel("Fault Ratio")
    plt.grid(True)
    plt.legend()
    plt.show()


"""
Set the buttons for the GUI.
"""
saveTemplate = Button(face,
                      text="SAVE TEMPLATE",
                      command=save_function)
saveTemplate.place(x=30, y=50)
```

```
loadFolder = Button(face,
                    text="LOAD FOLDER & COMPARE",
                    command=load_function)
loadFolder.place(x=30, y=100)

changeFingerprint = Button(face,
                           text='NEW FINGERPRINT',
                           command=change_template_fingerprint)
changeFingerprint.place(x=30, y=150)

thresholdVersusFault = Button(face,
                              text="THRESHOLD VS ERROR RATE",
                              command=threshold_vs_error_rate)
thresholdVersusFault.place(x=30, y=200)


face.mainloop()
```

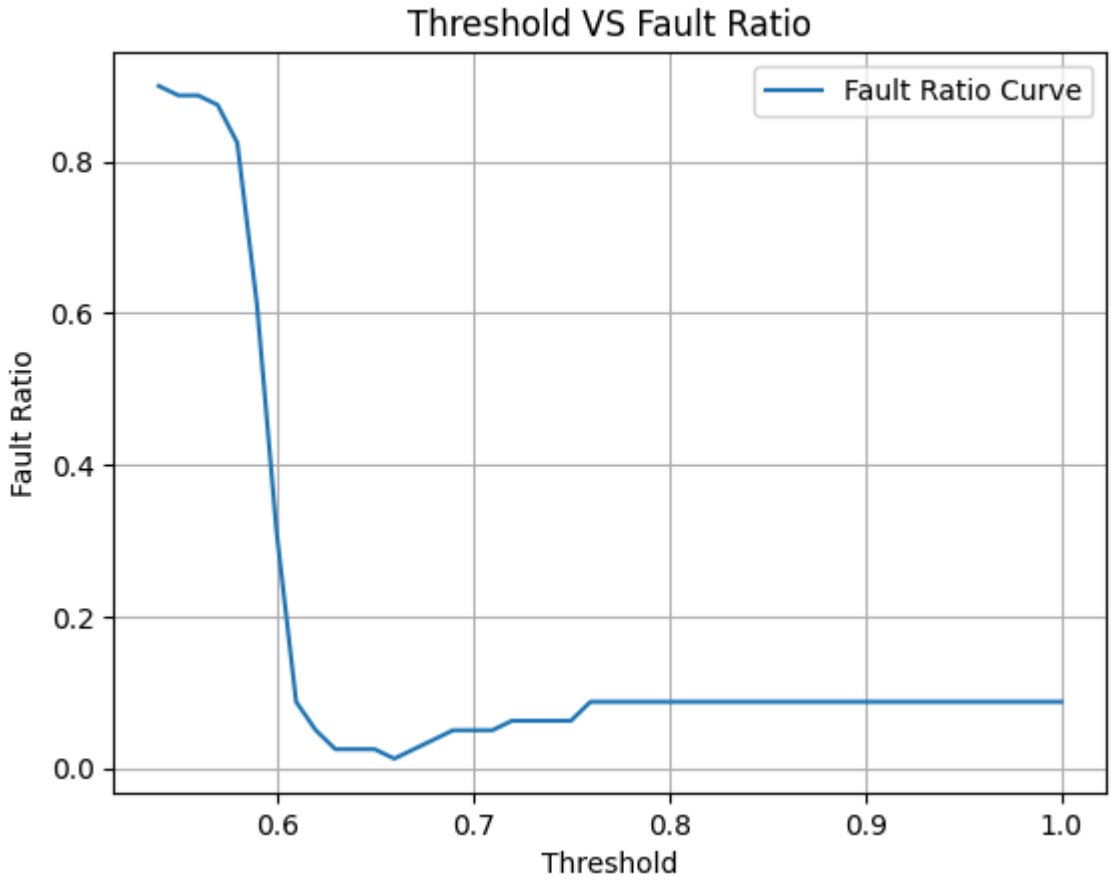
```
Fault ratio: 0.0125
[ inf 1.   0.75 0.65 0.62 0.61 0.6  0.59 0.58 0.57 0.56 0.55 0.54]
```



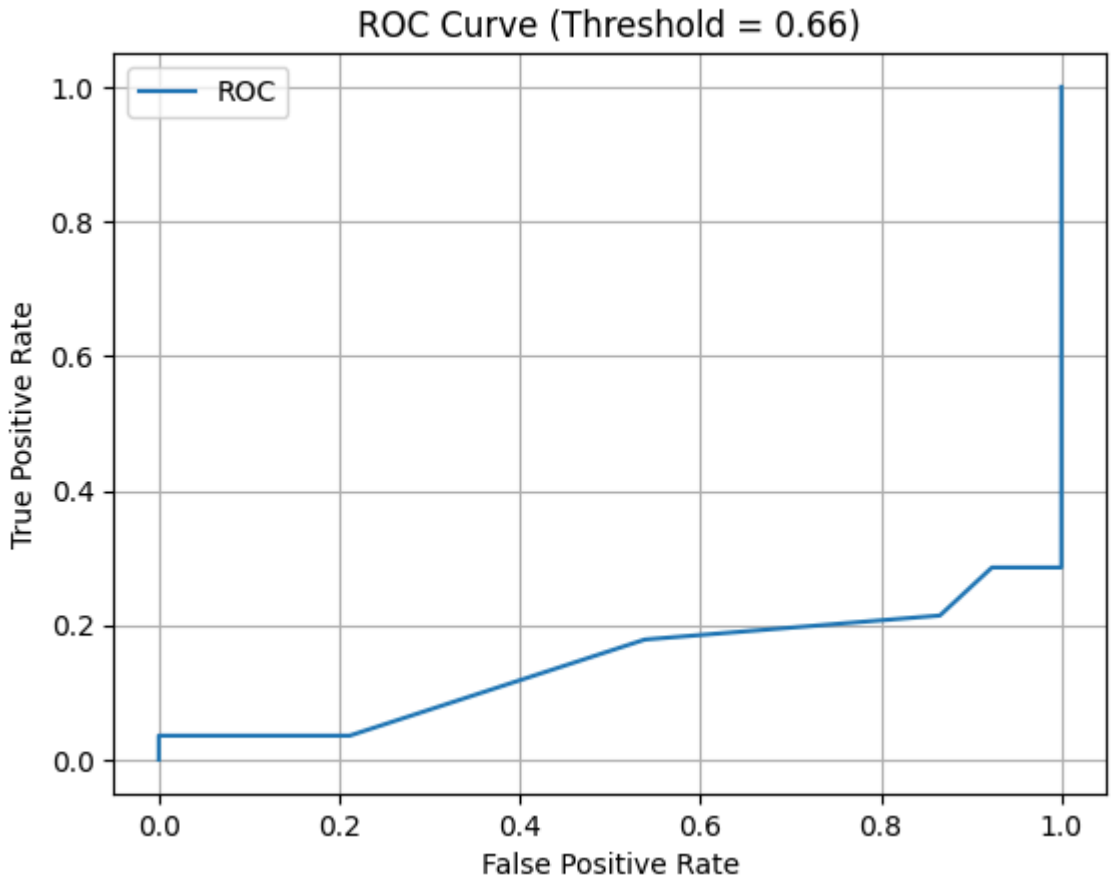
```
----------------------------------------
0 0.54
Fault ratio: 0.9
----------------------------------------
1 0.55
Fault ratio: 0.8875
----------------------------------------
2 0.56
Fault ratio: 0.8875
----------------------------------------
3 0.57
Fault ratio: 0.875
----------------------------------------
4 0.58
Fault ratio: 0.825
----------------------------------------
5 0.59
Fault ratio: 0.6125
----------------------------------------
6 0.6
Fault ratio: 0.3125
----------------------------------------
7 0.61
Fault ratio: 0.0875
----------------------------------------
8 0.62
Fault ratio: 0.05
----------------------------------------
9 0.63
Fault ratio: 0.025
----------------------------------------
10 0.64
Fault ratio: 0.025
----------------------------------------
11 0.65
Fault ratio: 0.025
----------------------------------------
12 0.66
Fault ratio: 0.0125
----------------------------------------
13 0.67
Fault ratio: 0.025
----------------------------------------
14 0.68
Fault ratio: 0.0375
----------------------------------------
15 0.69
Fault ratio: 0.05
----------------------------------------
16 0.7
Fault ratio: 0.05
----------------------------------------
17 0.71
Fault ratio: 0.05
----------------------------------------
18 0.72
Fault ratio: 0.0625
----------------------------------------
19 0.73
Fault ratio: 0.0625
```

```
----------------------------------------
20 0.74
Fault ratio: 0.0625
----------------------------------------
21 0.75
Fault ratio: 0.0625
----------------------------------------
22 0.76
Fault ratio: 0.0875
----------------------------------------
23 0.77
Fault ratio: 0.0875
----------------------------------------
24 0.78
Fault ratio: 0.0875
----------------------------------------
25 0.79
Fault ratio: 0.0875
----------------------------------------
26 0.8
Fault ratio: 0.0875
----------------------------------------
27 0.81
Fault ratio: 0.0875
----------------------------------------
28 0.82
Fault ratio: 0.0875
----------------------------------------
29 0.83
Fault ratio: 0.0875
----------------------------------------
30 0.84
Fault ratio: 0.0875
----------------------------------------
31 0.85
Fault ratio: 0.0875
----------------------------------------
32 0.86
Fault ratio: 0.0875
----------------------------------------
33 0.87
Fault ratio: 0.0875
----------------------------------------
34 0.88
Fault ratio: 0.0875
----------------------------------------
35 0.89
Fault ratio: 0.0875
----------------------------------------
36 0.9
Fault ratio: 0.0875
----------------------------------------
37 0.91
Fault ratio: 0.0875
----------------------------------------
38 0.92
Fault ratio: 0.0875
----------------------------------------
39 0.93
Fault ratio: 0.0875
```

```
----------------------------------------
40 0.94
Fault ratio: 0.0875
----------------------------------------
41 0.95
Fault ratio: 0.0875
----------------------------------------
42 0.96
Fault ratio: 0.0875
----------------------------------------
43 0.97
Fault ratio: 0.0875
----------------------------------------
44 0.98
Fault ratio: 0.0875
----------------------------------------
45 0.99
Fault ratio: 0.0875
----------------------------------------
46 1.0
Fault ratio: 0.0875
```



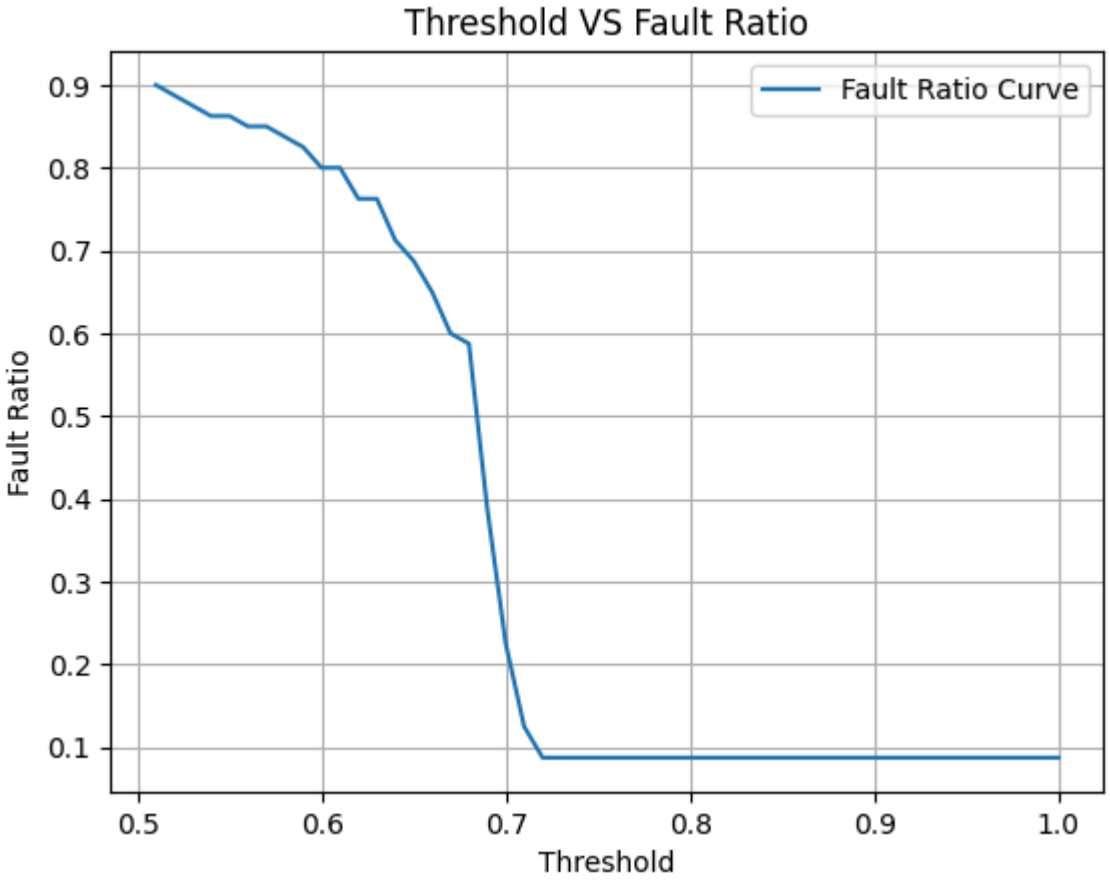

```
NEW FINGERPRINT ENROLLED!
NEW TEMP: 105_2.tif
Fault ratio: 0.65
[ inf 1.   0.71 0.7  0.69 0.68 0.67 0.66 0.65 0.64 0.63 0.61 0.59 0.51]
```


ROC Curve (Threshold = 0.66)
ROC
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

```
----------------------------------------
0 0.51
Fault ratio: 0.9
----------------------------------------
1 0.52
Fault ratio: 0.8875
----------------------------------------
2 0.53
Fault ratio: 0.875
----------------------------------------
3 0.54
Fault ratio: 0.8625
----------------------------------------
4 0.55
Fault ratio: 0.8625
----------------------------------------
5 0.56
Fault ratio: 0.85
----------------------------------------
6 0.57
Fault ratio: 0.85
----------------------------------------
7 0.58
Fault ratio: 0.8375
----------------------------------------
8 0.59
Fault ratio: 0.825
----------------------------------------
9 0.6
Fault ratio: 0.8
----------------------------------------
10 0.61
Fault ratio: 0.8
----------------------------------------
11 0.62
Fault ratio: 0.7625
----------------------------------------
12 0.63
Fault ratio: 0.7625
----------------------------------------
13 0.64
Fault ratio: 0.7125
----------------------------------------
14 0.65
Fault ratio: 0.6875
----------------------------------------
15 0.66
Fault ratio: 0.65
----------------------------------------
16 0.67
Fault ratio: 0.6
----------------------------------------
17 0.68
Fault ratio: 0.5875
----------------------------------------
18 0.69
Fault ratio: 0.3875
----------------------------------------
19 0.7
Fault ratio: 0.225
```

```
----------------------------------------
20 0.71
Fault ratio: 0.125
----------------------------------------
21 0.72
Fault ratio: 0.0875
----------------------------------------
22 0.73
Fault ratio: 0.0875
----------------------------------------
23 0.74
Fault ratio: 0.0875
----------------------------------------
24 0.75
Fault ratio: 0.0875
----------------------------------------
25 0.76
Fault ratio: 0.0875
----------------------------------------
26 0.77
Fault ratio: 0.0875
----------------------------------------
27 0.78
Fault ratio: 0.0875
----------------------------------------
28 0.79
Fault ratio: 0.0875
----------------------------------------
29 0.8
Fault ratio: 0.0875
----------------------------------------
30 0.81
Fault ratio: 0.0875
----------------------------------------
31 0.82
Fault ratio: 0.0875
----------------------------------------
32 0.83
Fault ratio: 0.0875
----------------------------------------
33 0.84
Fault ratio: 0.0875
----------------------------------------
34 0.85
Fault ratio: 0.0875
----------------------------------------
35 0.86
Fault ratio: 0.0875
----------------------------------------
36 0.87
Fault ratio: 0.0875
----------------------------------------
37 0.88
Fault ratio: 0.0875
----------------------------------------
38 0.89
Fault ratio: 0.0875
----------------------------------------
39 0.9
Fault ratio: 0.0875
```

```
-----------------------------------------
40 0.91
Fault ratio: 0.0875
-----------------------------------------
41 0.92
Fault ratio: 0.0875
-----------------------------------------
42 0.93
Fault ratio: 0.0875
-----------------------------------------
43 0.94
Fault ratio: 0.0875
-----------------------------------------
44 0.95
Fault ratio: 0.0875
-----------------------------------------
45 0.96
Fault ratio: 0.0875
-----------------------------------------
46 0.97
Fault ratio: 0.0875
-----------------------------------------
47 0.98
Fault ratio: 0.0875
-----------------------------------------
48 0.99
Fault ratio: 0.0875
-----------------------------------------
49 1.0
Fault ratio: 0.0875
```



Analysis to fingerprint recognition:

According to the figure upon, we can easily find that the result of the recognition is highly related to the threshold value setting and the fingerprint image selecting.

From the previous analysis, we used two fingerprints as the template, they are 101_2.tif and 105_2.tif respectively. To the image 101_2.tif, we find that the ROC curve of tpr VS fpr performs very well, the curve is quite close to the left up side of the graph, this means our model is a good model, and the fault ratio for this model is only **1.25%**, there are about 80 images in this DB1_B folder, 1.25% fault ratio means that there is only one image recognized wrong among all the images. And then we observe the thresholds VS error ratio graph, we know that the lowest error ratio only happens when we set the threshold to 0.66 and the template fingerprint as 0.0125. From the thresholds VS error rate, after the 0.66 threshold value, the error rate starts to increase and finally stop at 0.0875. This is because the threshold value is too high it set all the images as negative, but luckily, most of the fingerprint images do not belong to 101.

Then we do the analysis to the 105_2, if we still use the 0.66 as the threshold value, it returns a very high error ratio, which is not acceptable, but we can do the thresholds VS error ratio analysis. For 105_2, when the threshold value reaches 0.72, the error ratio reached the lowest value, 0.0875, but this value is just same as the situation we previously said, it might because the threshold is too high so that it can not be regraded as positive. So if we set the 105_2.tif as the template fingerprint, it is hard for us to select a suitable to make the system perfect.

So in conclusion, for different fingerprint images, we should set different threshold values so that can make the system reach the best performance. Currently, the best performance we reached is select 101_2.tif as template and set the threshold as 0.66. Figure 1.1 shows the image of the GUI we built.

NOTE:

Before we click the “THRESHOLD AND ERROR RATE” to get the “Thresholds VS Error Rate” curve, we should click the “LOAD FOLDER & COMPARE” Button.

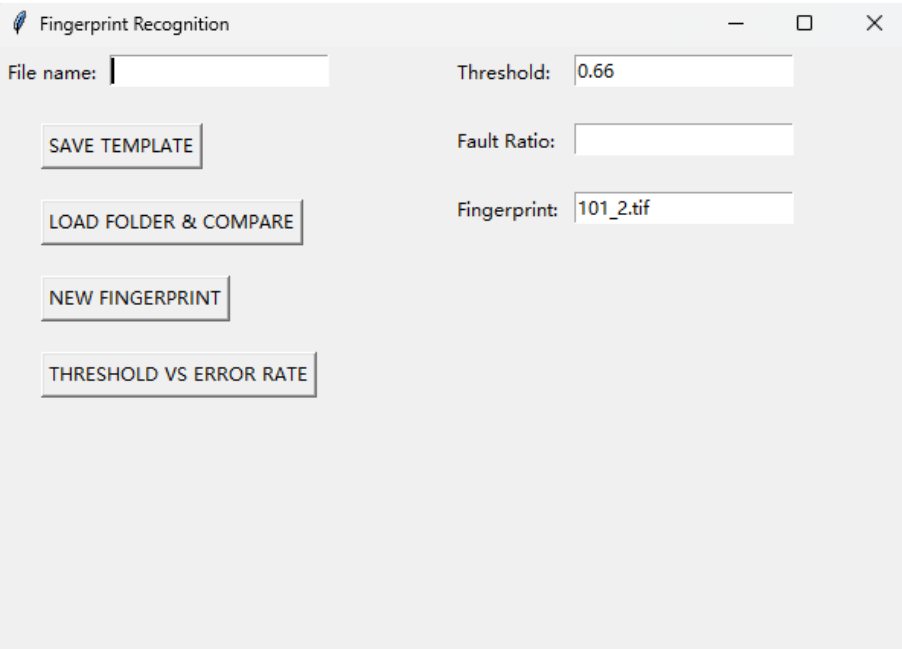


Figure 1.1: The GUI interface.

We type the name we want assigned for the current fingerprint in “File name” entry then click the “SAVE TEMPLATE” button, then the data for the current fingerprint can be saved.

Deep Learning Blogs

This is the address for my GitHub post:

https://github.com/HenryZhao1412/hengji.github.io/tree/master/_posts

The page should be like Figure 2.1:

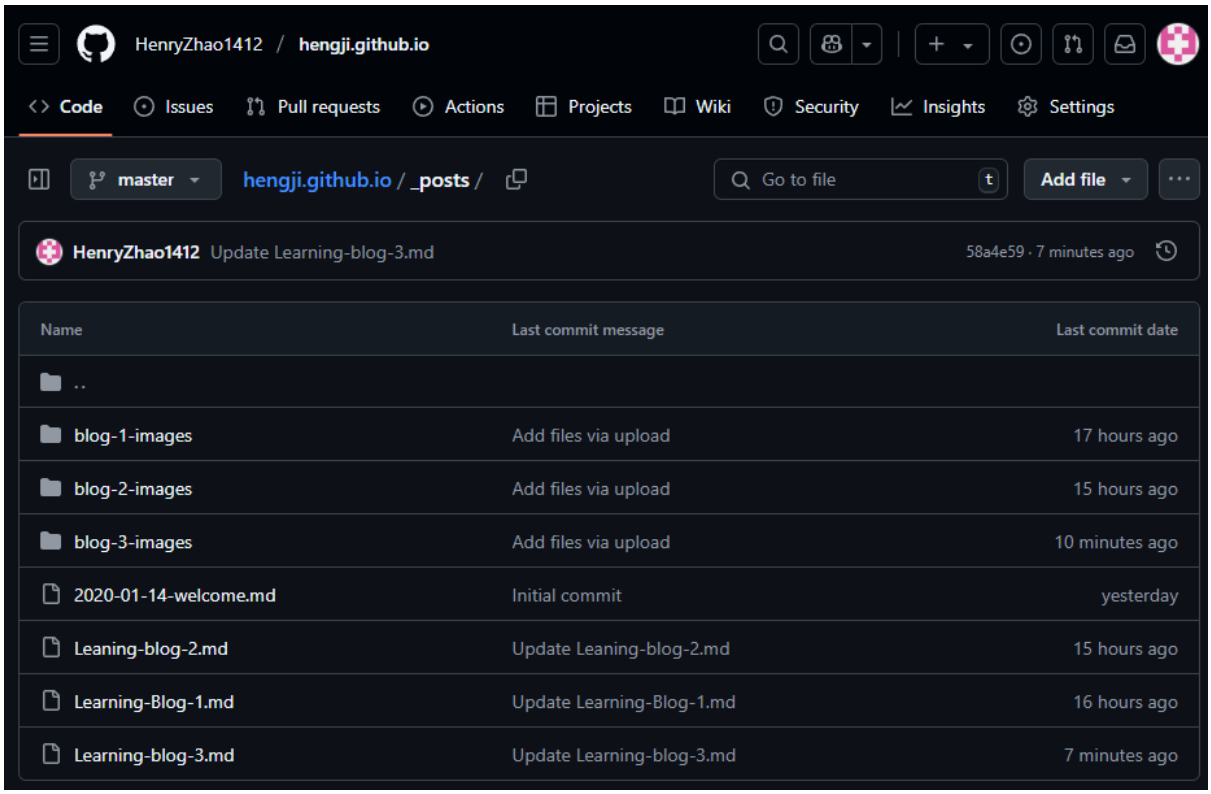


Figure 2.1: My GitHub blogs page.

The file "Learning-Blog-1.md", "Learning-blog-2.md" and "Learning-blog-3.md" are posts I written about my learning to fastbook, NVIDIA setup and cat & woodland classifier respectively.

Here are some example pages of my posts. These images contain all my blog content, but for a better reading experience, we should go the GitHub pages to watch it.

Blog 1:

Hengji's learning blog about Deep Learning & Fastai

The learning after reading the fastbook "01_intro.ipynb", this contains my understand to the 01_intro.ipynb, and I will explain it in my words.

Maching Learning

Traditional Programs

Normally, when the coders are planing to write a program to solve some problems, they just need to think about what inputs they need and what acts should the code do, and then write the code. For example when we are writing a function in Python, we need to think about what input the function require and what steps it should do and what varaible it will return. Like the Python below.

```
def two_number_addition(number1, number2):
    value = number1 + number2
    return value
```

To the code upon, it takes two numbers as input and do the addition operation. We use the following graph to represent this step.

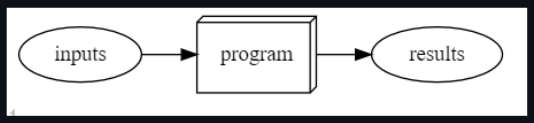


Figure 1: Traditional Program

Weight

Weights are variables, the notebook take a example of the dog classifier, it takes images pixels as inputs, the weights are values define how the program will operate. The structure is shown in the following figure.

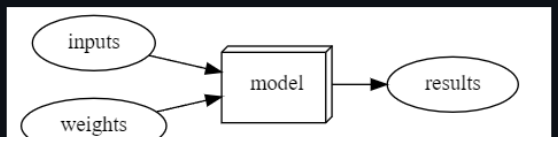


Figure 2.2: Example Page.

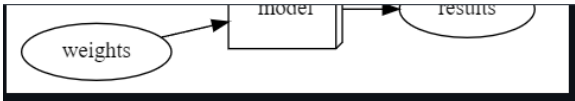


Figure 2: Weights considered

Weight Update

During the machine learning model process, weight should be able to be adjusted every time according to the results of training. It should be sent to the model together with the input data, then the model will give out the result of the training, the model will compare the results with the real label, then evaluate the performance of the model, if the performance do not good, the weight will be updated and then send to the model with input data again, after several iterations, make the performance of the model be perfect. The process should be like the following figure.

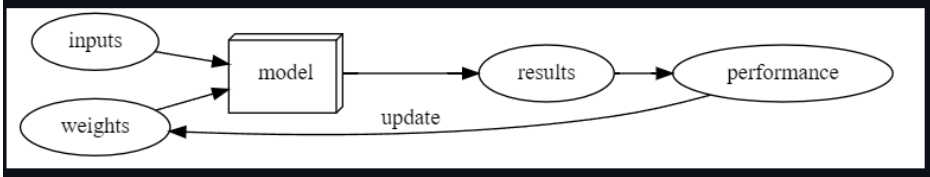


Figure 3: Update weight after the evaluation of performance.

After the training process

After we finished the training of the model, the model should be able to perform the actions we required it to do. We give the model input, then the model give us output. This process is shown in the following figure.

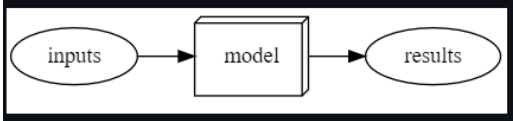
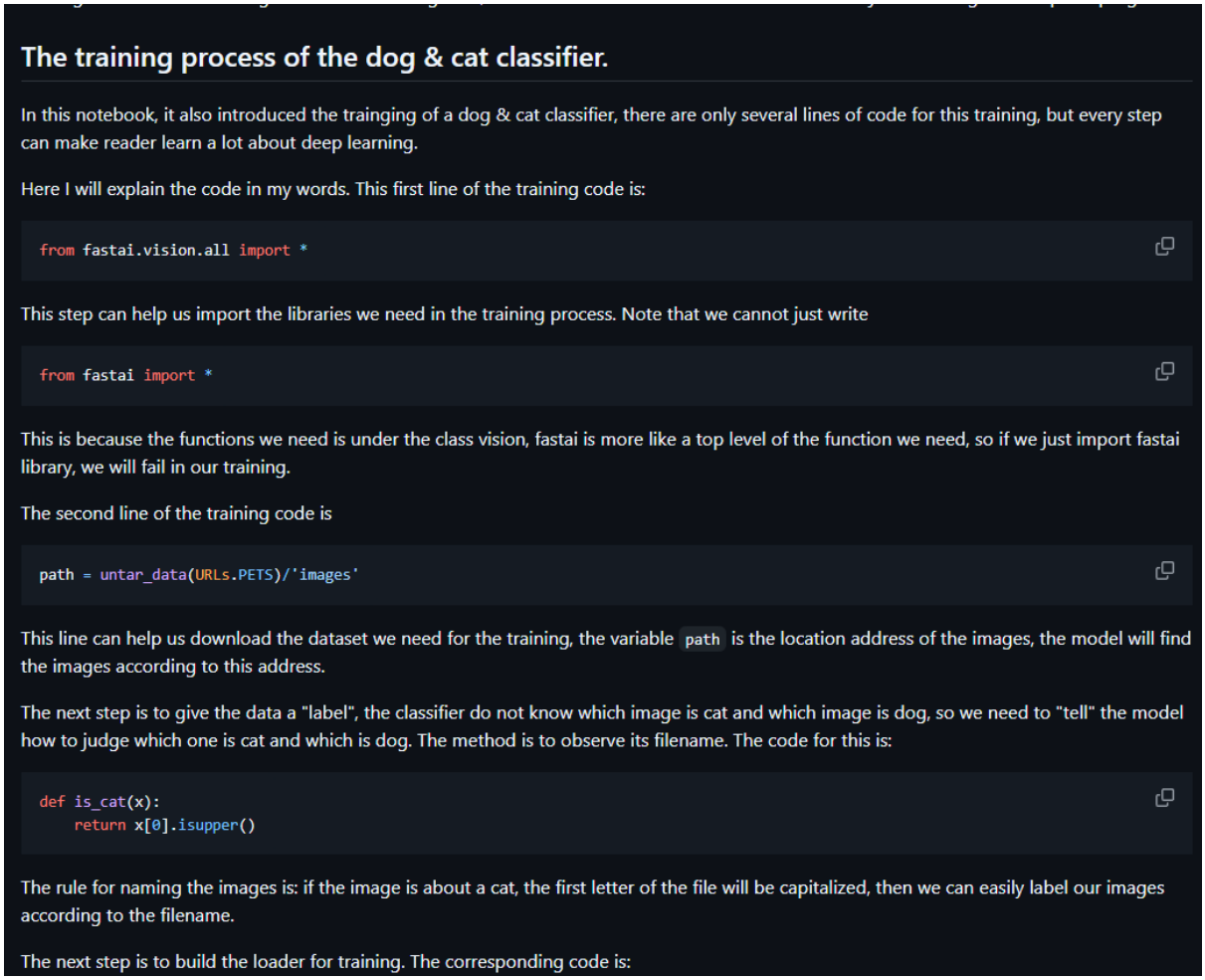


Figure 4: Apply the model finished training.

We might find that this the Figure 4 is similar to Figure 1, this is because a trained model can be treated just like a regular computer program.

Figure 2.3: Example Page



The training process of the dog & cat classifier.

In this notebook, it also introduced the trainging of a dog & cat classifier, there are only several lines of code for this training, but every step can make reader learn a lot about deep learning.

Here I will explain the code in my words. This first line of the training code is:

```
from fastai.vision.all import *
```

This step can help us import the libraries we need in the training process. Note that we cannot just write

```
from fastai import *
```

This is because the functions we need is under the class vision, fastai is more like a top level of the function we need, so if we just import fastai library, we will fail in our training.

The second line of the training code is

```
path = untar_data(URLs.PETS)/'images'
```

This line can help us download the dataset we need for the training, the variable `path` is the location address of the images, the model will find the images according to this address.

The next step is to give the data a "label", the classifier do not know which image is cat and which image is dog, so we need to "tell" the model how to judge which one is cat and which is dog. The method is to observe its filename. The code for this is:

```
def is_cat(x):
    return x[0].isupper()
```

The rule for naming the images is: if the image is about a cat, the first letter of the file will be capitalized, then we can easily label our images according to the filename.

The next step is to build the loader for training. The corresponding code is:

Figure 2.4: Example Page

The next step is to build the loader for training. The corresponding code is:

```
dls = ImageDataLoaders.from_name_func(
    path, get_image_file(path), valid_pct=0.2, seed=42,
    label_func=is_cat, item_tfms=Resize(224))
```

The first parameter in this data loader is telling the model where to find the training images.

The second parameter `get_image_files(path)` is helping the model to get the path for every images.

The third parameter `valid_pct=0.2` is meaning that 20% images in the dataset will be used as the validation set, and the rest 80% will be used as the training set. Validation set is necessary because it can be used to monitor the performance of the model. During the training process, the model will remember the training images, so, if we use the training set to test the model, the model will give you 100% correct answer, this is a cheat not what we want. The images in validation set the model didn't seen before, the results of using validation set can help us update the weight of the model.

The fourth parameter `seed=42` will make sure every time we train the model, it can give the model the same training set and the same validation set. This is important because if the model performs not evert well, we can set the same seed to another model and check if this is the problem of images in training set or the validation set.

The fifth parameter is `label_func=is_cat`, this is telling the model what the image in processing contains, this is an image about cat or dog.

The final parameter `item_tfms=Resize(224)` is to resize the size of the image to 224x224 for sending the images into the model.

The next step is to build the model and send the data loader into the model. The code is:

```
learn = vision_learner(dls, resnet34, metrics=error_rate)
```

The first parameter `dls` is the data loader we just built, the images, training set and validation are all prepared well for the model training.

The second parameter in the function is `resnet34` is a pretrained model, the number 34 means there are 34 layers in this architecture. `resnet34` was trained across 1.3 million photos, it can be used to identify the image features.

The third parameter `metrics=error_rate` is the evaluation indicator, this will tell you the performance of the trained model is good or not.

After we finished building the model, we should start our training, the code is:

Figure 2.5: Example Page.

```
learn.fine_tune(1)
```

This is the training process, the parameter inside the function '1' means that the model will be trained for 1 epoch. After the training, the system will tell you the performance of the model. The results of the training is shown in Figure 5.

epoch	train_loss	valid_loss	error_rate	time
0	0.169096	0.027093	0.012179	01:04

epoch	train_loss	valid_loss	error_rate	time
0	0.059723	0.017380	0.003383	01:07

Figure 5: The training results.

The GPU used in my laptop is "NVIDIA GeForce RTX 3060 Laptop", and the time spent on training the model is around 1 minute, actually I trained the model by using my CPU which is "AMD Ryzen 7 5800H with Radeon Graphics", the time spent on training is around 6~7 minutes for 1 epoch, so when we are needed to training a model, GPU is a better choice.

We find that the performance of our model is pretty good, the `error rate` even lower than 0.01, the train loss and valid loss mean how many images in training set and validation set are not enrolled into the model training.

Figure 2.6: Example Page.

Blog 2:

GPU setup learning

This is Hengji's learning of how to setup the NVIDIA GPU for Deep Learning.

Why I need to write this blog? The professor has already provided powerful GPU for our learning.

The first reason is that when I tried to connect to the `Dev Container`, it always failed and show me Figure 1, which makes me feel confused.

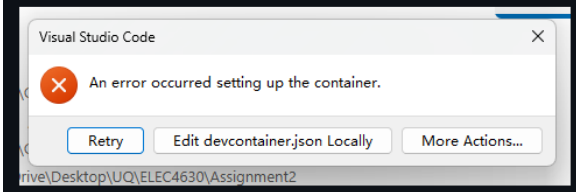


Figure 1: The Error message I come across when connecting the `Dev Container`.

Thanks for tutors' help, I can successfully connect to professor's GPU now.

The second reason is the GPU I cannot use for a long time, now I enrolled in ELEC4630, I can use the powerful GPU provided by professor, but in the future, I cannot use the professor's GPU for a lifetime.

Also, in the future the advanced GPU will be produced, so learning to setup the GPU for Deep Learning is necessary, it is not in the lecture slides but this is important for learning Deep Learning, that's what I think.

These learning resources are from Github, the address is here:

- Setup-NVIDIA-GPU-for-Deep-Learning

This is a very good resource easy to read and understand, so it collects 375 stars until now!

Then I will talk my learning process of setup the GPU.

GPU setup steps.

Figure 2.7: Example Page

1. NVIDIA Video Driver.

The GPU driver is necessary if we want to use the GPU. The address is here:

- NVIDIA GPU Drive Download

After we opened the link, the website should be like Figure 2.

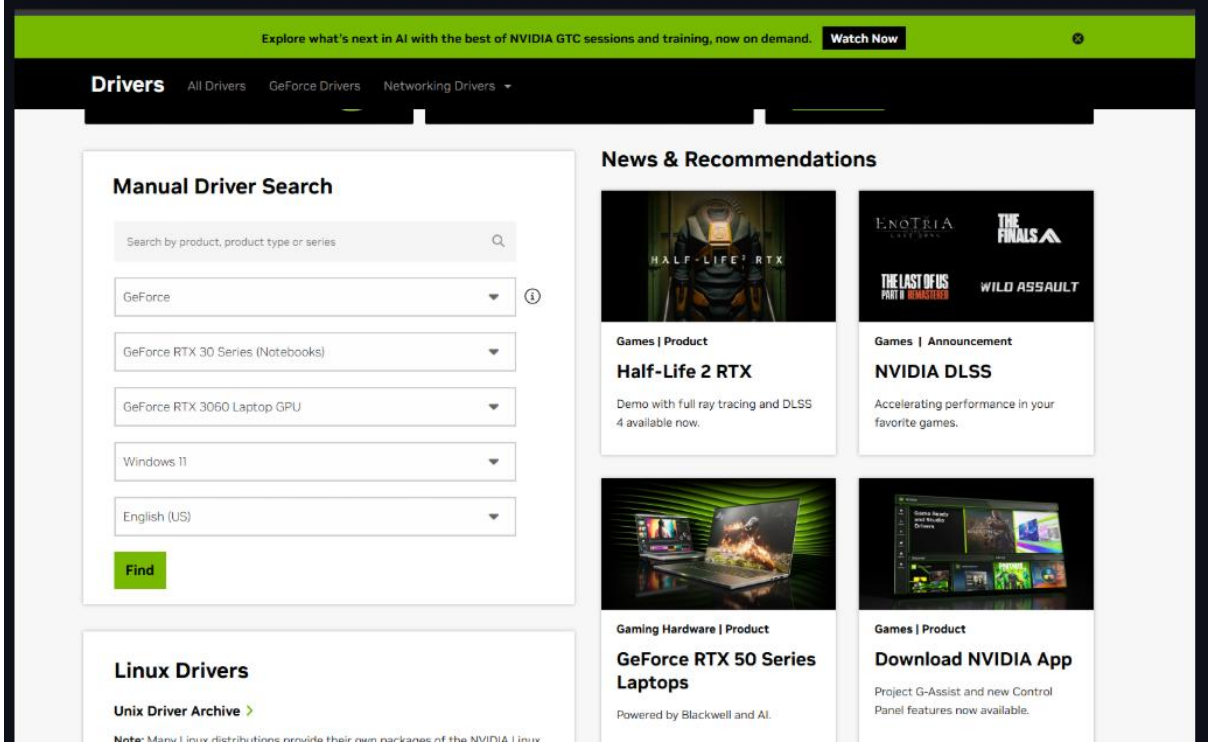


Figure 2.8: Example Page.

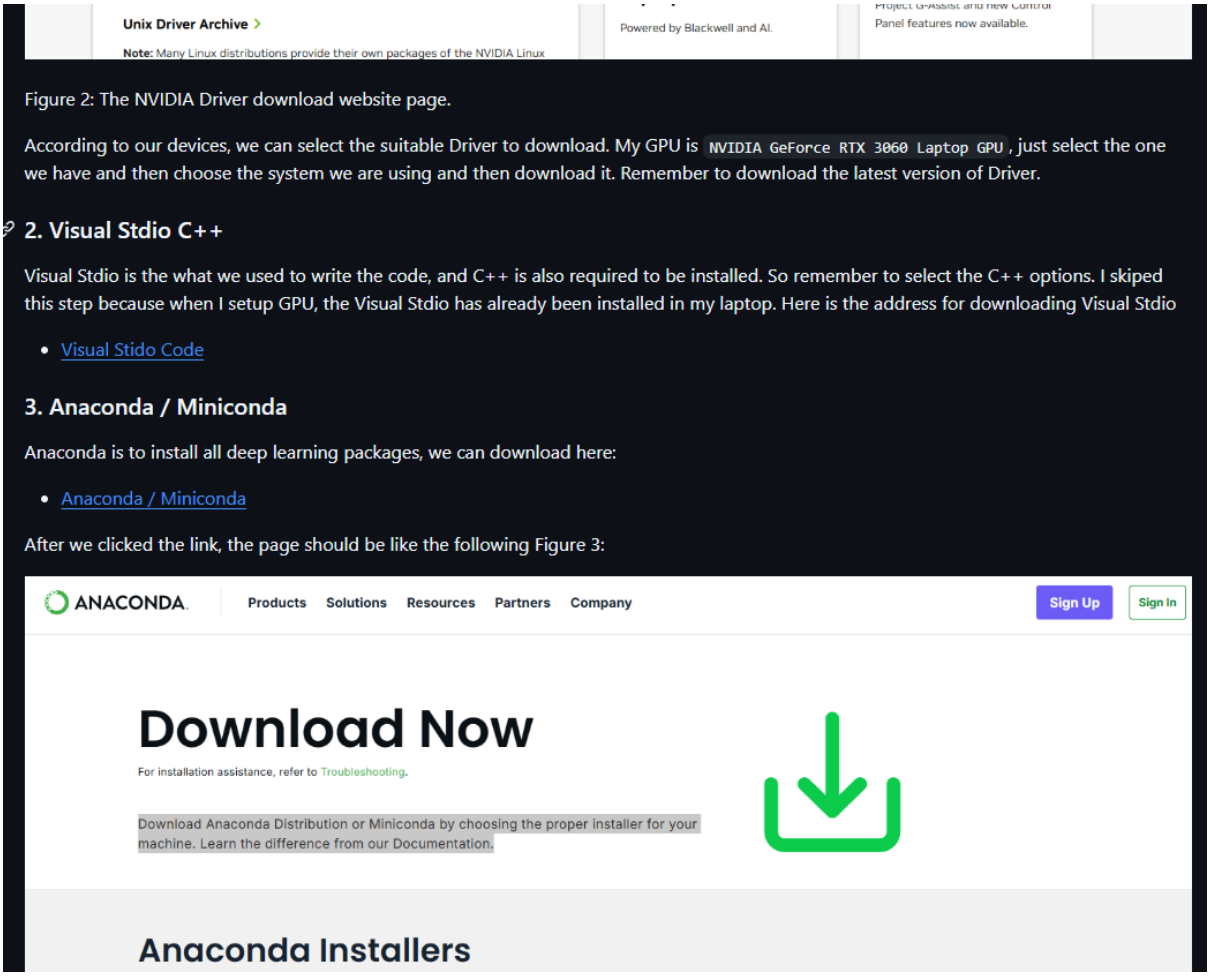


Figure 2.9: Example Page

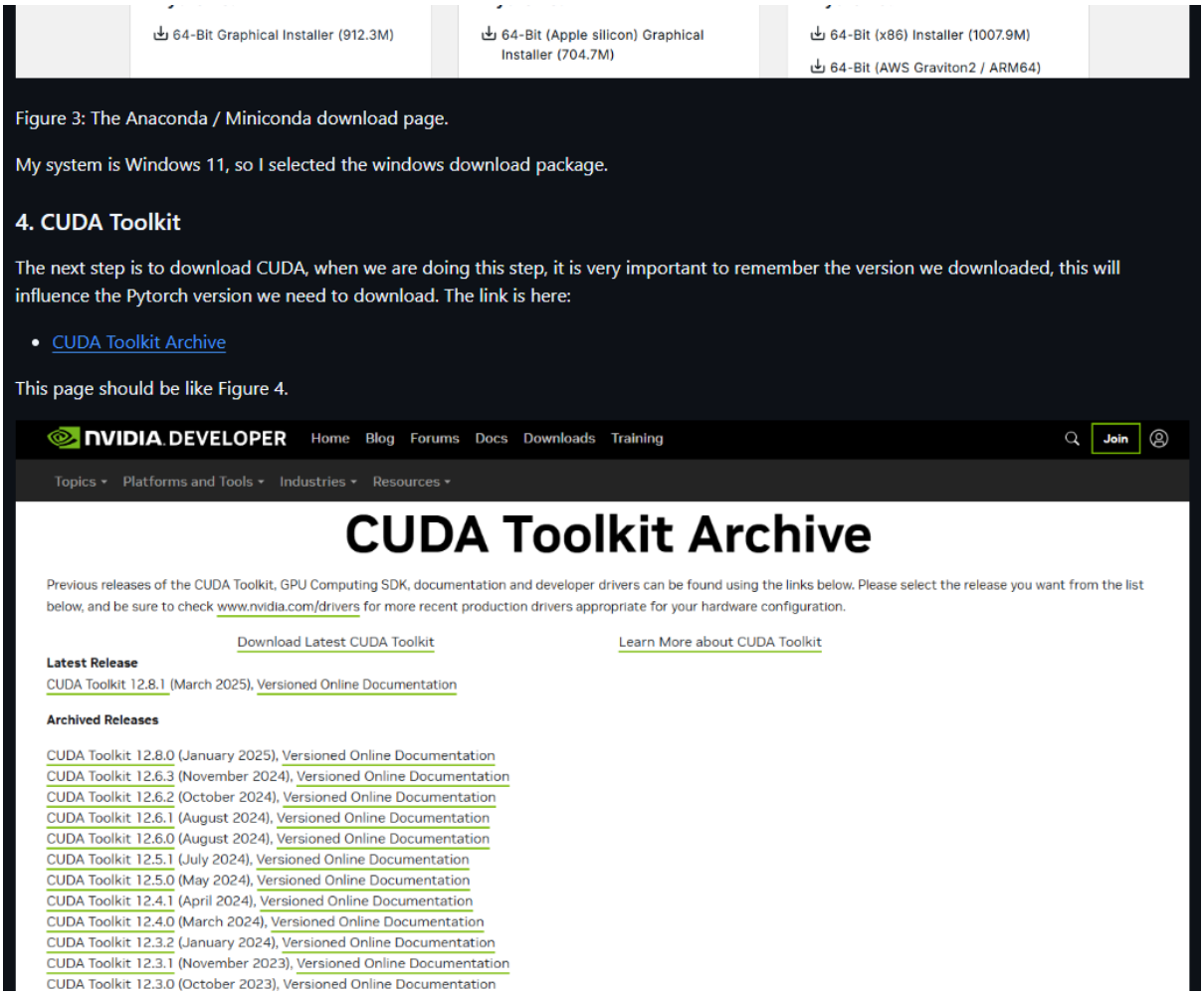


Figure 2.10: Example Page

CUDA Toolkit 11.7.1 (August 2022), Versioned Online Documentation
CUDA Toolkit 11.7.0 (May 2022), Versioned Online Documentation
CUDA Toolkit 11.6.2 (March 2022), Versioned Online Documentation

Figure 4: The CUDA Toolkit Download page.

5. cuDNN

The next step is to download the cuDNN archive, this is a GPU-accelerated library of primitives for deep neural networks. The link is here:

- cuDNN Archive

After we opened this link, the page should be like Figure 5.

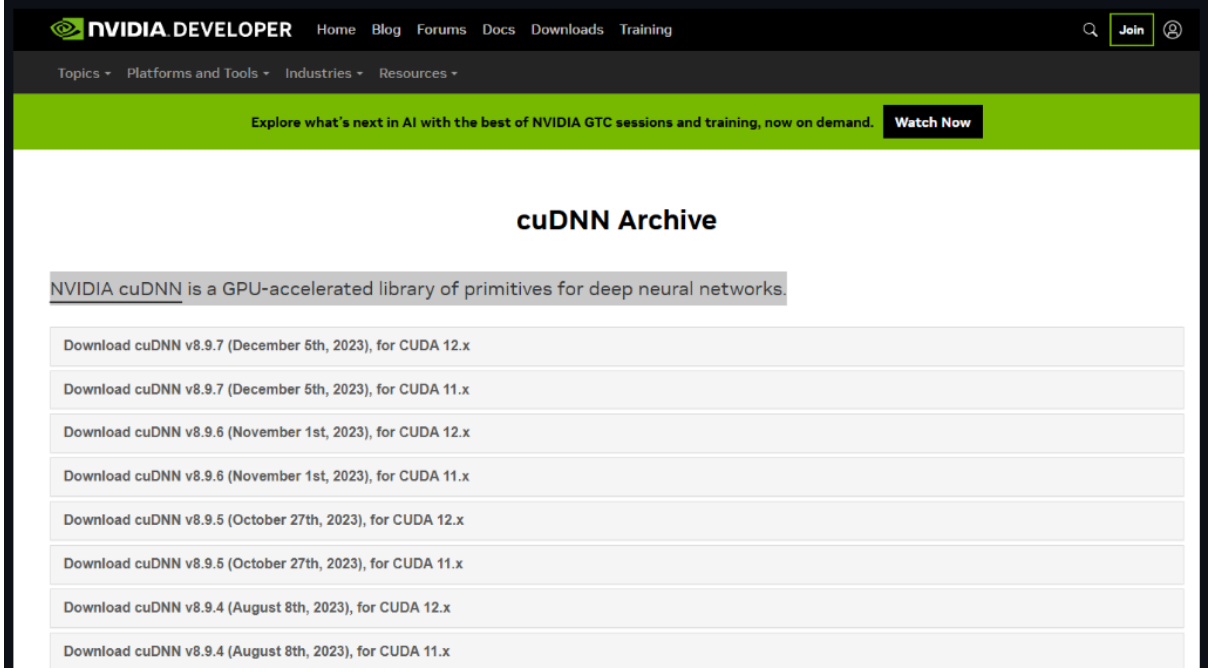


Figure 2.11: Example Page

Figure 5: The page of cuDNN

6. Install Pytorch

The link for installing Pytroch is here:

- Pytorch Download

After we clicked the link, the page should be like Figure 6.

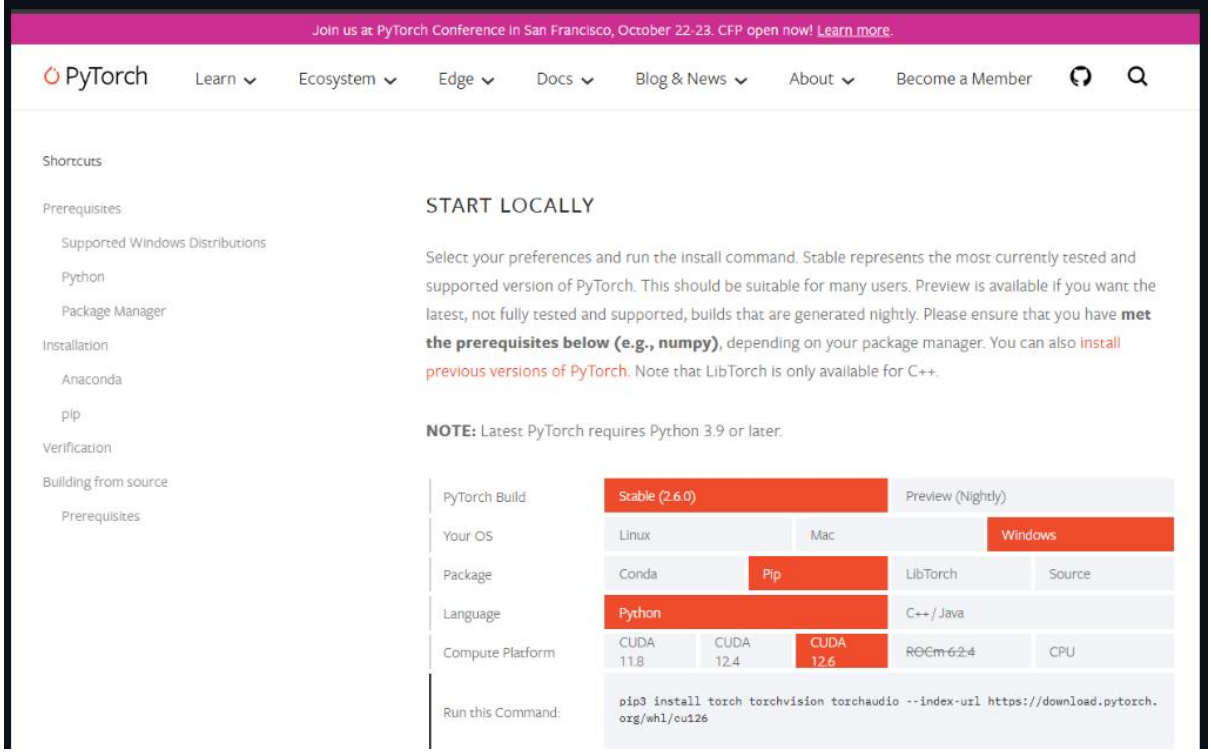


Figure 2.12: Example Page

Run this Command:	pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu126

Figure 6: The Pytorch Download Page

We need to select the correct version of Pytorch, the version should be the same as the Toolkit we just downloaded, the version I selected was 12.8, it is not available in the `Stable (2.6.0)` choice in `PyTorch Build` sublist. Then we should choose the `Preview (Nightly)` choice. Just as Figure 7.

PyTorch Build	Stable (2.6.0)		Preview (Nightly)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
Compute Platform	CUDA 11.8	CUDA 12.6	CUDA 12.8	~~ROCm 6.3~~	CPU
Run this Command:	pip3 install --pre torch torchvision torchaudio --index-url https://download.pytorch.org/whl/nightly/cu128			

Figure 7: Choose the correct version.

Lastly, we can use the following Python code to check whether our GPU can work normally.

```
import torch

print("Number of GPU: ", torch.cuda.device_count())
print("GPU Name: ", torch.cuda.get_device_name())

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print('Using device:', device)
```

Figure 8 is the result of my laptop runing the code, the GPU type is the same as I said before.

Figure 2.12: Example Page

Figure 8 is the result of my laptop runing the code, the GPU type is the same as I said before.

```
import torch

print("Number of GPU: ", torch.cuda.device_count())
print("GPU Name: ", torch.cuda.get_device_name())


device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print('Using device:', device)
```

✓ 13.3s

```
Number of GPU:  1
GPU Name:  NVIDIA GeForce RTX 3060 Laptop GPU
Using device: cuda
```

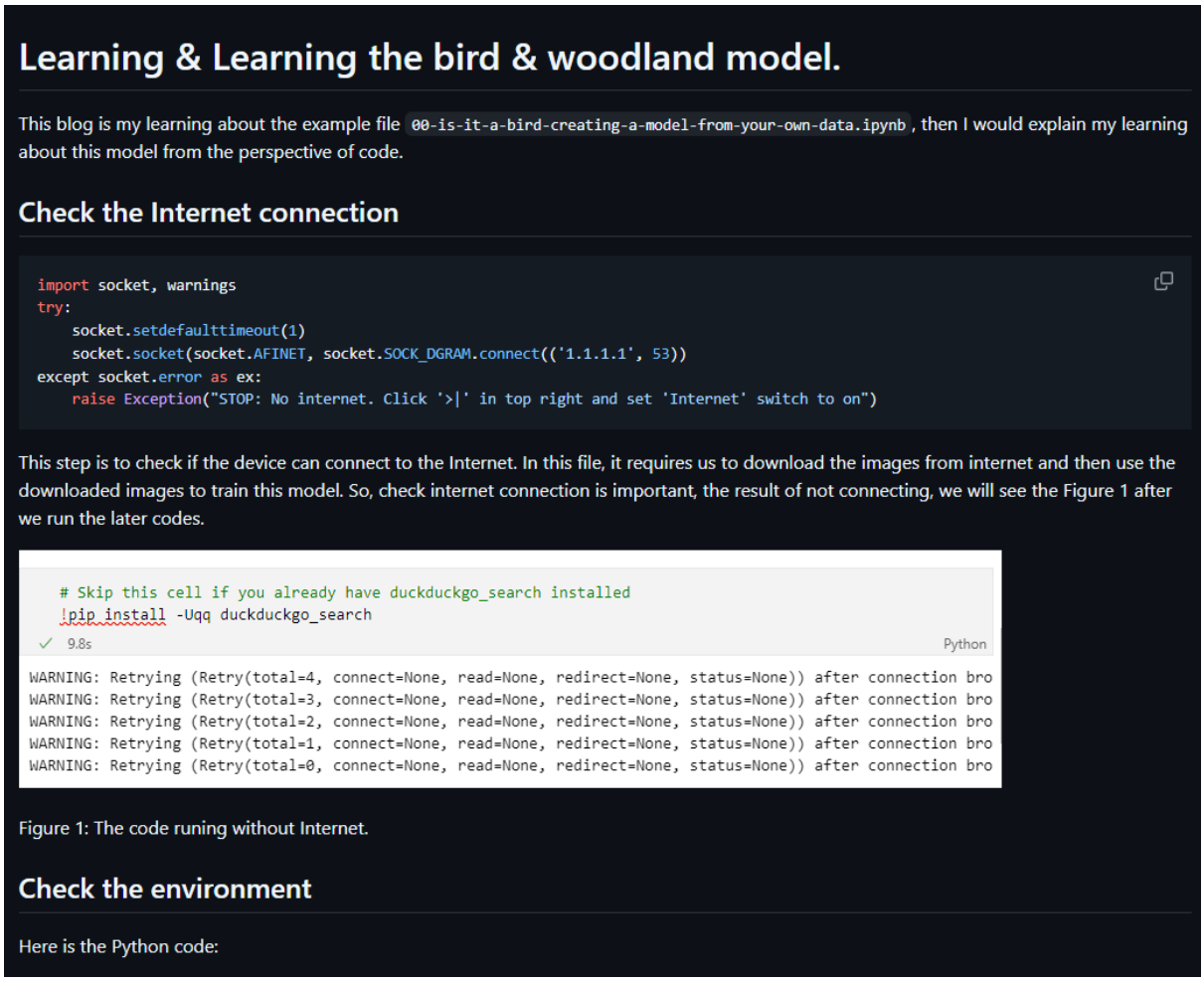
Figure 8: The results of running the Python code.

We find that the GPU can work for training our model in Visual Stdio Code now.

Previously, training 2 epoches for `01-intro.ipynb` dog & cat classifier spent me about 15 minutes on training by using CPU, but now only spend around 2 minitus and 30 seconds for me to train the same code. It can save plenty of time.

Figure 2.13: Example Page.

Blog 3:



Learning & Learning the bird & woodland model.

This blog is my learning about the example file `00-is-it-a-bird-creating-a-model-from-your-own-data.ipynb` , then I would explain my learning about this model from the perspective of code.

Check the Internet connection

```
import socket, warnings
try:
    socket.setdefaulttimeout(1)
    socket.socket(socket.AFINET, socket.SOCK_DGRAM.connect(('1.1.1.1', 53))
except socket.error as ex:
    raise Exception("STOP: No internet. Click '>|' in top right and set 'Internet' switch to on")
```

This step is to check if the device can connect to the Internet. In this file, it requires us to download the images from internet and then use the downloaded images to train this model. So, check internet connection is important, the result of not connecting, we will see the Figure 1 after we run the later codes.

```
# Skip this cell if you already have duckduckgo_search installed
!pip install -Uqq duckduckgo_search
```

✓ 9.8s Python

```
WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None, status=None)) after connection bro
WARNING: Retrying (Retry(total=3, connect=None, read=None, redirect=None, status=None)) after connection bro
WARNING: Retrying (Retry(total=2, connect=None, read=None, redirect=None, status=None)) after connection bro
WARNING: Retrying (Retry(total=1, connect=None, read=None, redirect=None, status=None)) after connection bro
WARNING: Retrying (Retry(total=0, connect=None, read=None, redirect=None, status=None)) after connection bro
```

Figure 1: The code runing without Internet.

Check the environment

Here is the Python code:

Figure 2.14: Example Page

Check the environment

Here is the Python code:

```
import os
iskaggle = os.environ.get('KAGGLE_KERNEL_RUN_TYPE', '')

if iskaggle:
    !pip install -Uqq fastai
```

This step is checking if the code is running under the Kaggle environment, if it is, the code will intall the latest fastai library automatically, the parameter `-U` means intall the latest version. If the fastai is not the latest version, this code will also update the fastai to the latest version as well.

Download images from internet by using `duckduckgo_search`

Install the `duckduckgo_serach` library

```
!pip install -Uqq duckduckgo_search
```

This code can install the `duckduckgo_search` library if this library was not installed in the larptop, if your computer already has this library, skip this line of code.

Define the searching function

```
from duckduckgo_search import DDGS
from fastcore.all import *

ddgs = DDGS()

def search_images(term, max_images=200):
    return L(ddgs.images(term, max_results=max_images)).itemgot('image')
```

Figure 2.15: Example Page

This code is defining the function that used to search the images for model training. It will return an array that contains the URLs of the images, we can set the number of images he searches for according to our needs, and if we do not give a specific number of images, this function will search 200 images in default.

Here is the example of using this function and we asked this function to search 1 image.

```
urls = search_images('bird photos', max_images=1)
urls[0]
```

The Figure 2 shows the result of this code performed in my laptop.

```
urls = search_images('bird photos', max_images=1)
urls[0]
```

✓ 2.0s Python

'https://images.pexels.com/photos/97533/pexels-photo-97533.jpeg?cs=srgb&dl=animal-avian-bird-97533.jpg&fm=jpg'

Figure 2: The result of running function `search_images`.

According to Figure 2, we find that result is an url of image, we can not read any information for this link, then we need to show this image, it will use the following code.

```
from fastdownload import download_url
dest = 'bird.jpg'
download_url(urls[0], dest, show_progress=False)

from fastai.vision.all import *
im = Image.open(dest)
im.to_thumb(256,256)
```

Then the result of running these code is shown in Figure 3.

Figure 2.16: Example Page

Then the result of running these code is shown in Figure 3.

```
from fastdownload import download_url
dest = 'bird.jpg'
download_url(urls[0], dest, show_progress=False)

from fastai.vision.all import *
im = Image.open(dest)
im.to_thumb(256,256)
```

✓ 5.9s Python

Figure 3: The downloaded image of `search_image` example.

In this part of code, we imported a function named `download_url` from the `fastdownload` library. This Function can download the images according to the urls to our own folders, the second parameter `dest` is the filename of the downloaded image. If the image is in the same directory of the current code file, we can just use the filename we assigned for this image to open it, just like the sixth line of the Python code we just plotted.

Similarly, we can download the image of woodlands as well, the code is:

```
download_url(search_images('woodlands photos', max_images=1)[0], 'woodlands.jpg', show_progress=False)
Image.open('woodlands.jpg').to_thumb(256,256)
```

And the result of running this code is shown in Figure 4.

Figure 2.17: Example Page

```
download_url(search_images('woodlands photos', max_images=1)[0], 'woodlands.jpg', show_progress=False)
Image.open('woodlands.jpg').to_thumb(256,256)
```

✓ 2.2s Python

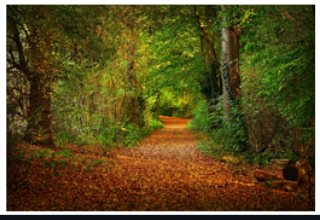


Figure 4: Seach 1 image of woodland.

Build the dataset for training.

Training set is necessary for Python training, in this file we are required to download the images by ourselves, so we need to use the function `search_images` to search images for training. Here is the Python code.

Figure 2.18: Example Page

Build the dataset for training.

Training set is necessary for Python training, in this file we are required to download the images by ourselves, so we need to use the function `search_images` to search images for training. Here is the Python code.

```
searches = 'woodlands','bird'
path = Path('bird_or_not')
from time import sleep
import os
from glob import glob

for o in searches:
    dest = (path/o)
    dest.mkdir(exist_ok=True, parents=True)
    download_images(dest, urls=search_images(f'{o} photo'))
    sleep(10)  # Pause between searches to avoid over-loading server
    download_images(dest, urls=search_images(f'{o} sun photo'))
    sleep(10)
    download_images(dest, urls=search_images(f'{o} shade photo'))
    sleep(10)
    for file in glob(f"{dest}/*.fpx"): # remove fpx files which cause problems with resize_images
        os.unlink(file)
    resize_images(path/o, max_size=400, dest=path/o)
```

The variable `searches` is the target object we want to search, this model should be able to determine an object is bird or woodland, then we define the `searches` variable to let model search images automatically.

`path` is folder's name, this folder should contains the images downloaded from internet, if there is not folder named `bird_or_not` in current directory, the system will create one.

Then the code will start to download the images, it will create 2 sub folders under the `bird_or_not` folder named `woodlands` and `bird` respectively. And then the code will download 200 images each with the specific item. Also, if we download the `.fpx` images, the system will delete the file automatically because our system can not resize the image. Finally, the image will be resize to 400x400.

This step might take some time, my laptop spent 4 minutes and 21.4 seconds on this step.

Train the model

Figure 2.19: Example Page

Train the model

Before we start our training, we should delete the failed images in our image folders. The code is here:

```
failed = verify_images(get_image_files(path))
failed.map(Path.unlink)
len(failed)
```

This step will help us delete the failed images and how many failed images deleted.

The next step is to build the data loader, the code is here:

Figure 2.20: Example Page

```
dls = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    splitter=RandomSplitter(valid_pct=0.2, seed=42),
    get_y=parent_label,
    item_tfms=[Resize(192, method='squish')]
).dataloaders(path)

dls.show_batch(max_n=6)
```

These code used the `seed=42` do divide the dataset into validation set and training set, 20% of the dataset images belong to validation set and the rest belong to the training set.

The `squish` method is stretch or compressimage to a target size (like 192x192 in this code) without preserving the original aspect ratio. The Figure 5 shows some images in the data loader.

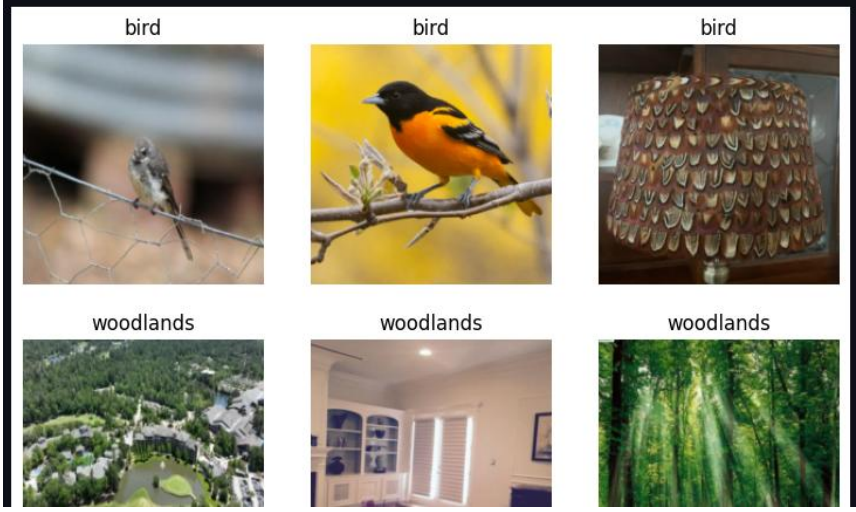


Figure 2.21: Example Page

Figure 5: The images in the dataloader we built.

Then we start to train our model, the corresponding code is here:

```
learn = vision_learner(dls, resnet18, metrics=error_rate)
learn.fine_tune(3)
```

We trained this model for 3 epoches, and use the `error_rate` method to monitor the performance of the model, Figure 6 shows the exact process of the training.

epoch	train_loss	valid_loss	error_rate	time
0	0.435453	0.189929	0.064993	00:13

epoch	train_loss	valid_loss	error_rate	time
0	0.123176	0.079741	0.026588	00:14
1	0.060904	0.094793	0.017725	00:13
2	0.026333	0.086640	0.017725	00:14

Figure 6: The training process.

We used the pretrained model `resnet18` for training and finally the `error_rate` reduced from 0.026588 to 0.017725 through the 3 epoches.

Use the model

```
is_bird,_,probs = learn.predict(PILImage.create('bird.jpg'))
print(f"This is a: {is_bird}.")
print(f"Probability it's a bird: {probs[0]:.4f}")
```

These code is to use the bird image we just downloaded to test the model performance, the result is shown in Figure 7.

Figure 2.22: Example Page

These code is to use the bird image we just downloaded to test the model performance, the result is shown in Figure 7.

```
is_bird,_,probs = learn.predict(PILImage.create('bird.jpg'))
print(f"This is a: {is_bird}.")
print(f"Probability it's a bird: {probs[0]:.4f}")
```

✓ 0.1s Python

```
This is a: bird.
Probability it's a bird: 1.0000
```

Figure 7: The performance of our model

We find the performance of our model is good.

Figure 2.23: Example Page

That's all for our blogs.

Question 3: GPU or CPU

(a) Test the GPU under different batch size (16, 32, 64, 128, 256).

Batch size is how many samples is per batch to load into the training. The fastai DataLoader block has a parameter named ‘bs’, which is to define the batch size, we should notice that there is also a parameter named ‘batch_size’ . To change the batch size, we only need to change the parameter ‘bs’, because the ‘batch_size’ is for the compatibility of PyTorch.

The Figure 3.1~3.5 illustrate every time we change batch size under the GPU environment, it records the time spent on training and shows the corresponding batch size.

We have to notice that when we are doing this training, we should comment the code of downloading images, if we do not comment that, it will make our dataset larger, it is impossible to accurately control the variables

```
dls = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    splitter=RandomSplitter(valid_pct=0.2, seed=42),
    get_y=parent_label,
    item_tfms=[Resize(192, method='squish')]
).dataloaders(path, bs=16)

dls.show_batch(max_n=6)

learn = vision_learner(dls, resnet18, metrics=error_rate)
learn.fine_tune(3)
```

✓ 48.7s

epoch	train_loss	valid_loss	error_rate	time
0	0.481057	0.265078	0.086749	00:10

epoch	train_loss	valid_loss	error_rate	time
0	0.237383	0.137426	0.040038	00:12
1	0.139974	0.108140	0.032412	00:12
2	0.053649	0.087587	0.027645	00:12

Figure 3.1: The batch size of 16.

```
dls = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    splitter=RandomSplitter(valid_pct=0.2, seed=42),
    get_y=parent_label,
    item_tfms=[Resize(192, method='squish')]
).dataloaders(path, bs=32)

dls.show_batch(max_n=6)

learn = vision_learner(dls, resnet18, metrics=error_rate)
learn.fine_tune(3)
```

✓ 36.4s

epoch	train_loss	valid_loss	error_rate	time
0	0.462112	0.290967	0.083889	00:07

epoch	train_loss	valid_loss	error_rate	time
0	0.203858	0.131976	0.035272	00:09
1	0.072631	0.105343	0.023832	00:09
2	0.023052	0.102468	0.022879	00:09

Figure 3.2: The batch size of 32.

```
dls = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    splitter=RandomSplitter(valid_pct=0.2, seed=42),
    get_y=parent_label,
    item_tfms=[Resize(192, method='squish')]
).dataloaders(path, bs=64)

dls.show_batch(max_n=6)

learn = vision_learner(dls, resnet18, metrics=error_rate)
learn.fine_tune(3)
```

✓ 32.3s

epoch	train_loss	valid_loss	error_rate	time
0	0.536471	0.234940	0.077216	00:06

epoch	train_loss	valid_loss	error_rate	time
0	0.193915	0.105307	0.029552	00:08
1	0.088788	0.098517	0.021926	00:08
2	0.035234	0.098342	0.025739	00:07

Figure 3.3: The batch size of 64.

```
dls = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    splitter=RandomSplitter(valid_pct=0.2, seed=42),
    get_y=parent_label,
    item_tfms=[Resize(192, method='squish')]
).dataloaders(path, bs=128)

dls.show_batch(max_n=6)

learn = vision_learner(dls, resnet18, metrics=error_rate)
learn.fine_tune(3)
```

✓ 31.0s

epoch	train_loss	valid_loss	error_rate	time
0	0.621161	0.321481	0.081983	00:06

epoch	train_loss	valid_loss	error_rate	time
0	0.250642	0.112742	0.034318	00:07
1	0.121053	0.103700	0.029552	00:07
2	0.066384	0.104849	0.023832	00:07

Figure 3.4: The batch size of 128

```
dls = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    splitter=RandomSplitter(valid_pct=0.2, seed=42),
    get_y=parent_label,
    item_tfms=[Resize(192, method='squish')]
).dataloaders(path, bs=256)

dls.show_batch(max_n=6)

learn = vision_learner(dls, resnet18, metrics=error_rate)
learn.fine_tune(3)
```

✓ 33.3s

epoch	train_loss	valid_loss	error_rate	time
0	0.800658	0.371134	0.074357	00:07

epoch	train_loss	valid_loss	error_rate	time
0	0.295810	0.134337	0.043851	00:08
1	0.173075	0.104424	0.023832	00:08
2	0.109760	0.096941	0.021926	00:08

Figure 3.5: The batch size of 256.

According to these figures, we find that the fastest batch size is 128.

Originally, I thought the fastest should be 256, because it can send the most images to the model every time, however, its actual behaviour is not so good as batch size 128.

(b) Test GPU & CPU

Firstly, we test GPU with the default, we restart the kernel and test it. The result is shown in Figure 3.6.

```
dls = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    splitter=RandomSplitter(valid_pct=0.2, seed=42),
    get_y=parent_label,
    item_tfms=[Resize(192, method='squish')]
).dataloaders(path)

dls.show_batch(max_n=6)

learn = vision_learner(dls, resnet18, metrics=error_rate)
learn.fine_tune(3)
```

✓ 30.3s

epoch	train_loss	valid_loss	error_rate	time
0	0.519295	0.200138	0.080076	00:06

epoch	train_loss	valid_loss	error_rate	time
0	0.199858	0.114353	0.040038	00:07
1	0.094385	0.122033	0.028599	00:07
2	0.036678	0.118368	0.027645	00:07

Figure 3.6: GPU training.

```
dls = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    splitter=RandomSplitter(valid_pct=0.2, seed=42),
    get_y=parent_label,
    item_tfms=[Resize(192, method='squish')]
).dataloaders(path)

dls.show_batch(max_n=6)

learn = vision_learner(dls, resnet18, metrics=error_rate)
learn.fine_tune(3)
```

✓ 9m 8.4s

```
/home/vscode/.local/lib/python3.10/site-packages/fastai/torch_core.py:263: UserWarning: 'has_mps' is deprecated, please use 'torch.backends.mps.is_built()'
  return getattr(torch, 'has_mps', False)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /home/vscode/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
100%|██████████| 44.7M/44.7M [00:01<00:00, 39.9MB/s]
```

epoch	train_loss	valid_loss	error_rate	time
0	0.517403	0.232561	0.066730	01:47

epoch	train_loss	valid_loss	error_rate	time
0	0.205292	0.122241	0.031459	02:23
1	0.099859	0.137408	0.032412	02:26
2	0.040225	0.122116	0.024786	02:28

Figure 3.7: CPU training.

According to the Figure 3.6 and Figure 3.7, we find that the GPU training only spent several seconds for one epoch, but the CPU takes more than two minutes. The GPU can save much time so that can make our training more efficient, therefore, GPU is the best choice.

(c) Use the 'nvtop' tool to check the GPU activities.

Here we run the code under different batch size, each graph I marked its corresponding batch size for that.

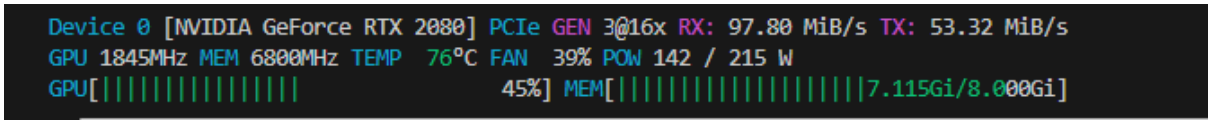


Figure 3.8: The GPU resources usage with a batch size of 16.

According to Figure 3.8, we observed that the GPU usage is about 45% and the power consumption is 142 W,

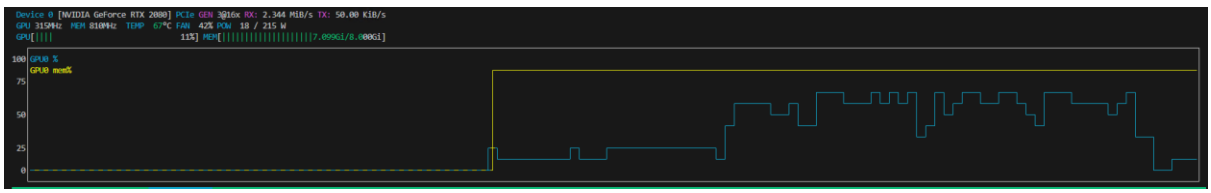


Figure 3.9: The GPU usage graph after training (batch size = 16)

According to Figure 3.9, we find that the curve fluctuated 4 times, considering the Figure 3.1

to Figure 3.5, we know that there are 4 training epochs every time we run the code.

Then Figure 3.10 and Figure 3.11 shows the nvtop monitoring the training with a batch size of 32.

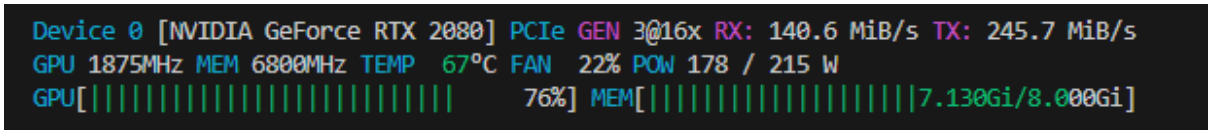


Figure 3.10: The GPU resources usage with a batch size of 32.

According to the Figure 3.10, we find that the GPU usage increased compared the batch size of 16, the GPU usage accounted for 76% and the power consumption increased to 178 W.

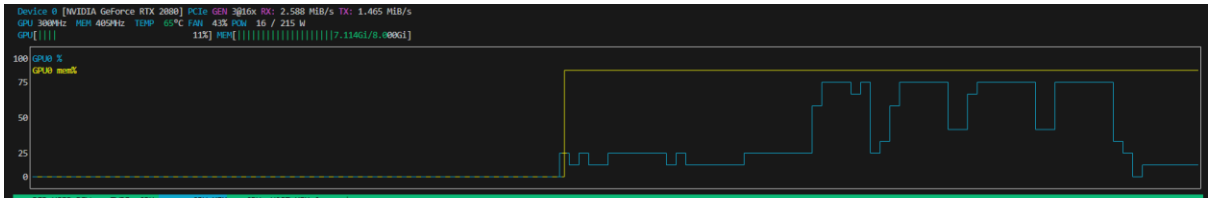


Figure 3.11: The GPU usage graph after training (batch size = 32)

By observing the image Figure 3.9 and Figure 3.11, we find that the memory usage is all most same (just a little increasement), so we can take the memory curve (the yellow one) as a criterion, we find that the peak of the 4 fluctuations increased. That means the GPU usage increased.

Then Figure 3.12 and Figure 3.13 illustrate the situation with batch size = 64

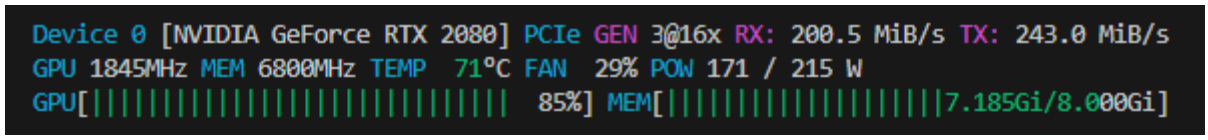


Figure 3.12: The GPU resources usage with a batch size of 64.

According to the Figure 3.12, we find that the GPU usage is 85%, the power consumption decreased because the power consumption is not a constant value, it was fluctuated, when I was taking the screenshot, the power consumption shows 171 W.

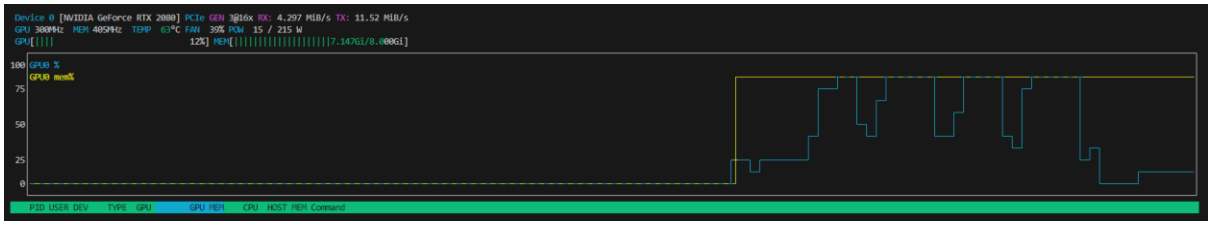


Figure 3.13: The GPU usage graph after training (batch size = 64)

We find that during the training process, the GPU usage curve increased compared to batch size = 32, this time the GPU usage curve meet the memory usage curve.

Then Figure 3.14 and Figure 3.15 illustrate the situation with batch size = 128.

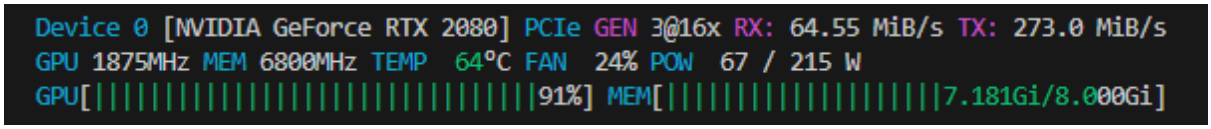


Figure 3.14: The GPU resources usage with a batch size of 128.

The GPU usage increased again compared to batch size = 64, this time accounting for 91% during the training period.

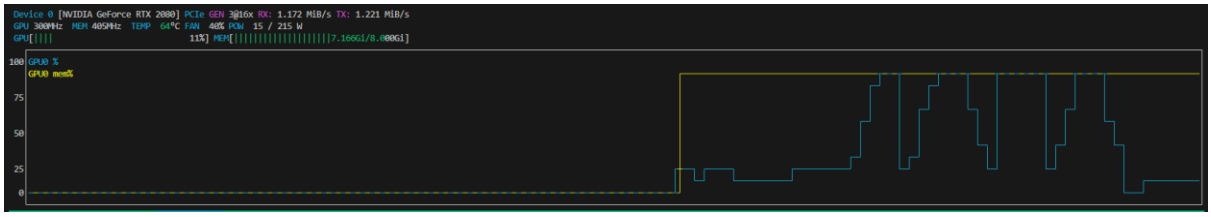


Figure 3.15: The GPU usage graph after training (batch size = 128)

We find that the width of the 4 fluctuations decreased compared to the former training processes, this is because the time spent on training reduced by increasing the batch size, and the GPU usage curve increased a little bit.

The Figure 3.16 and Figure 3.17 illustrate the situation with batch size = 256.

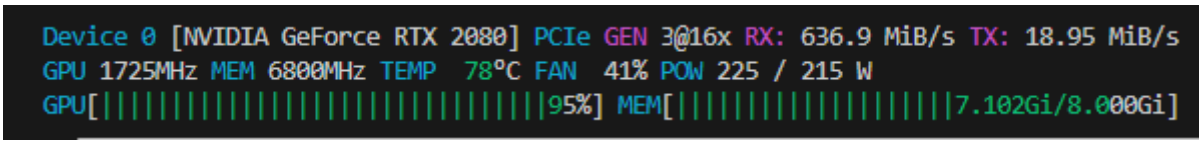


Figure 3.16: The GPU resources usage with a batch size of 256.

We find that the GPU usage increased again as we increased the batch size to 256, and the power consumption also increased to 225 W, even over the 215 W limitation.

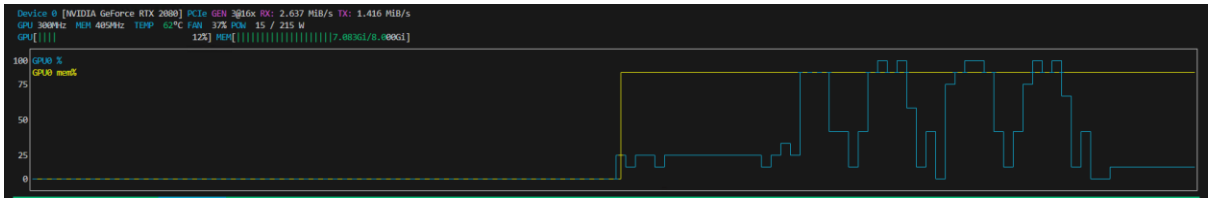


Figure 3.17: The GPU usage graph after training (batch size = 256)

From Figure3.17, we find that the GPU usage curve increased again even over the yellow memory curve.

In consumption, according to these experiences of training model under different conditions, I find that as we increased the batch size, the GPU usage and the power consumption will increase together, however, there is almost no influence on the memory usage.

Through this question, I witnessed the performance of GPU and CPU when we are training the model, the GPU is much more efficient than CPU, therefore, the GPU is a better choice for training the model. Also, when we are training the model, we need to consider the GPU usage when we are setting the parameters, we can not set a very large batch size that make the GPU overloaded, this will make the temperature rising and cause a damage to our GPU.

Q4: Objects Classifier, t-SNE & Confusion matrics

In [ ]:
```
# Doing the Internet connection check.
import socket,warnings
try:
    socket.setdefaulttimeout(1)
    socket.socket(socket.AF_INET,
                  socket.SOCK_DGRAM).connect(('1.1.1.1', 53))
except socket.error as ex:
    raise Exception("STOP: No internet. Click '>|' in top right and set 'Interne
```

In [2]:
```
# Check whether in the Kaggle environment, if in, install the latest fastai
import os
iskaggle = os.environ.get('KAGGLE_KERNEL_RUN_TYPE', '')

if iskaggle:
    !pip install -Uqq fastai
```

Download images of birds and non-birds

In [3]:
```
# Install the library of duckduckgo_search for searching training images.
!pip install -Uqq duckduckgo_search
```

In [4]:
```
from duckduckgo_search import DDGS
from fastcore.all import *

ddgs = DDGS()

# Define the seraching function, this function will return a url.
# I doubled the number of image downloads from the internet
def search_images(term, max_images=200):
    return L(ddgs.images(term, max_results=max_images)).itemgot('image')
```

In [5]:
```
from fastdownload import download_url
from fastai.vision.all import *

# Define the searching classes, these are objects we need to
# let our model learn.
searches = 'airplane', 'automobile', 'bird', 'cat', 'dog'
path = Path('q4_training_data')

from time import sleep
import os
from glob import glob

# Download the images for training.
# Here we start, in each loop we use the searching function for
# 3 times, that means download 1200 images for each object, however
# The download of some images might go wrong, so need to remove
# the failed images later.
```

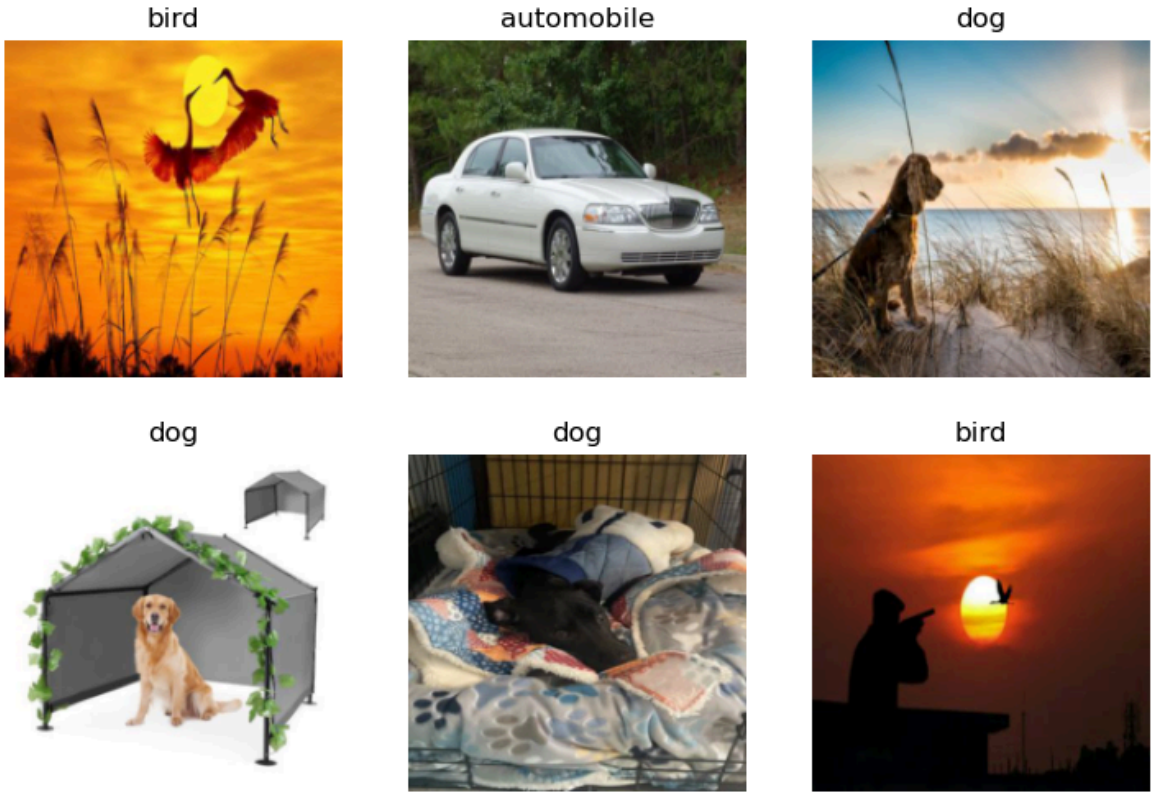
```
for o in searches:
    dest = (path/o)
    dest.mkdir(exist_ok=True, parents=True)
    download_images(dest, urls=search_images(f'{o} photo'))
    sleep(10)
    download_images(dest, urls=search_images(f'{o} sun photo'))
    sleep(10)
    download_images(dest, urls=search_images(f'{o} shade photo'))
    sleep(10)

    # Need to remove the ".fpx" images, cause it can't be resized.
    for file in glob(f"{dest}/*.fpx"):
        os.unlink(file)
    resize_images(path/o, max_size=400, dest=path/o)
```

Train the model

In [6]:
```
# Delete the failed download images.
failed = verify_images(get_image_files(path))
failed.map(Path.unlink)
len(failed)
```

Out[6]: 172

In [7]:
```
# Build the data loader, use 20% of dataset as validation set
# and the rest 80% used as training set. In this training, we
# also choose the seed=42 to divide the dataset.
# Finally, use the 'squish' method to resize the images.
dls = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    splitter=RandomSplitter(valid_pct=0.2, seed=42),
    get_y=parent_label,
    item_tfms=[Resize(192, method='squish')]
).dataloaders(path)

# Show some images inside the data loader.
dls.show_batch(max_n=6)
```



In [8]:
```
# Start to train the model, we train this model for 5 epoches.
learn = vision_learner(dls, resnet18, metrics=error_rate)
learn.fine_tune(5)
```

epoch	train_loss	valid_loss	error_rate	time
0	0.643972	0.399411	0.124057	01:10

epoch	train_loss	valid_loss	error_rate	time
0	0.320522	0.237087	0.064857	01:16
1	0.178848	0.222854	0.062971	01:18
2	0.059797	0.240455	0.055807	01:17
3	0.029115	0.224487	0.050528	01:15
4	0.014281	0.226161	0.050151	01:17

From the results of the model training, we find that the performance (according to the error rate, lower error rate, the better performance of model) is getting better after each epoch.

In [9]:
```
# I downloaded an image of plane from internet and named
# the image of 'pl.jpg' to test the performance of the model
# It looks good.

is_cat, _, probs = learn.predict(PILImage.create('pl.jpg'))
print(is_cat)
print(f'{probs[0]:.4f}')
```

```
airplane
1.0000
```

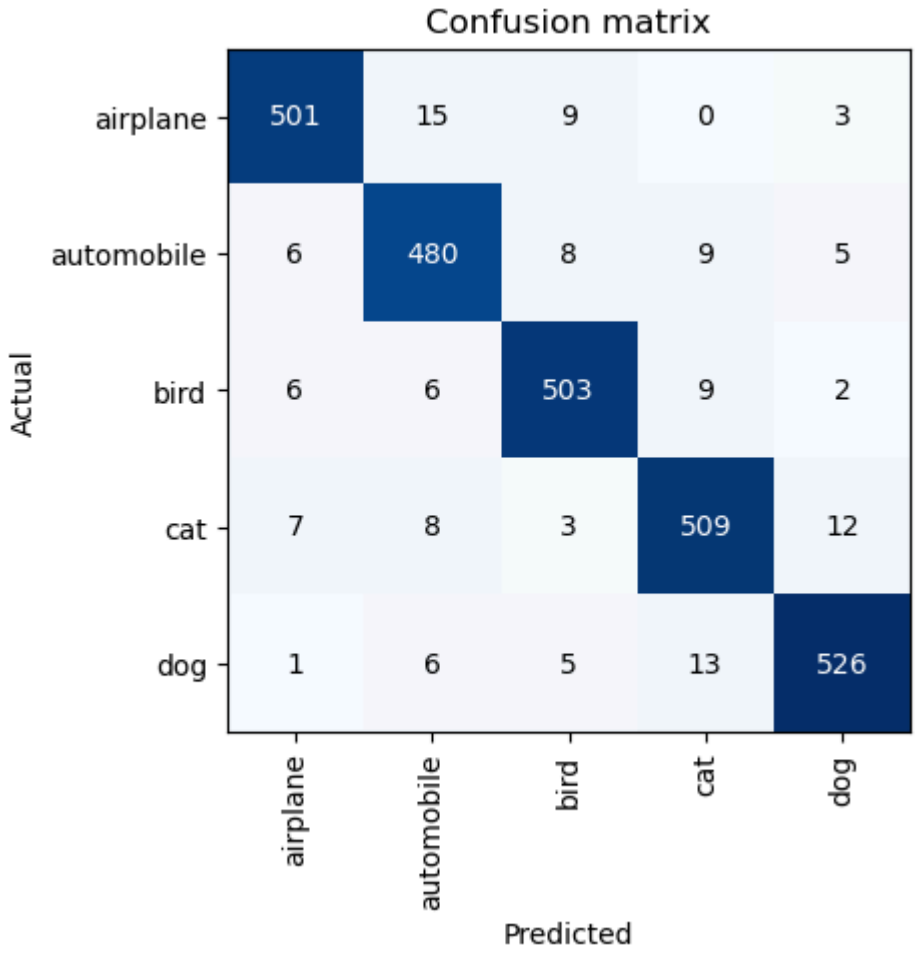
In [10]:
```
# We do not want to train the model every time when we want
# to do the prediction, so it is import to package it into
# a file, the next time if we want to use this model, we can
# load it directly without training.
learn.export()
path = Path()
path.ls(file_exts='.pkl')
```

Out[10]:
```
(#1) [Path('export.pkl')]
```

Confusion matrix

The confusion matrix is using the validation set to test the performance of our model, then plot the results directly on a nxn (n is how many classes we need to determine) matrix, we can easily check the result of this method.

In [11]:
```
# The confusion matrics to do the performance checking.
interp = ClassificationInterpretation.from_learner(learn)
interp.plot_confusion_matrix()
```



According to the confusion matrix, we find most of the images are recognized right. The number of correct recognition is 2519, the number of incorrect recognition is 133, then the over error rate deduced from the Confusion Matrix is [133/(133+2519)] = 0.05015 which is about 5%.

Loss function

For this multi-class classification task, we used the `CrossEntropyLoss`, which is the default loss function used by fastai for classification problems. It is well-suited for scenarios where each input belongs to one of several classes. The loss combines a `LogSoftmax` and `Negative Log Likelihood` to encourage the model to assign high probability to the correct class.

In [12]:
```
print(learn.loss_func)
```

```
FlattenedLoss of CrossEntropyLoss()
```

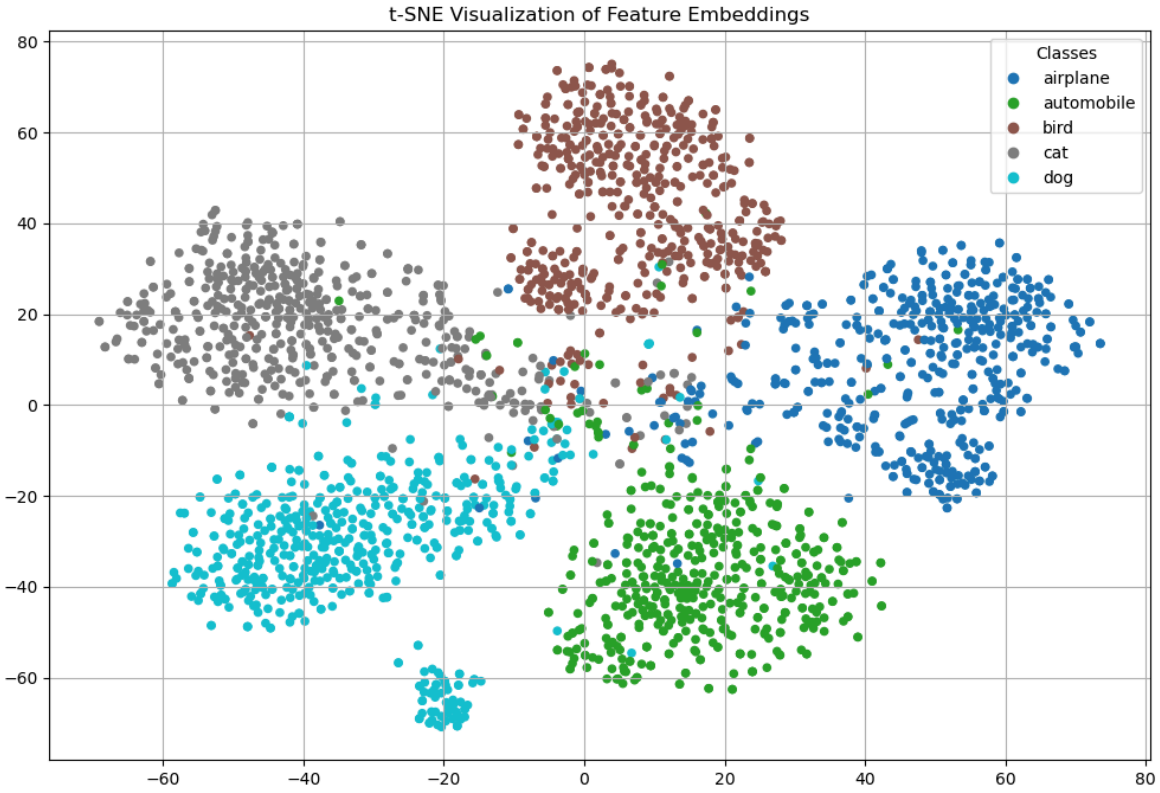
t-SNE Analysis

The t-SNE method is to map the high-dimension data points to a lower-dimensional space, typically we transform them into 2 or 3 dimensions.

In [ ]:
```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import numpy as np
import torch

# Extract the features of the model.
# The model final output has five classes (corresponding
# to the 5 objects we defined), however, we need to capture
# the second last layer's output. Because it is the most
# informative general feature representation extracted
# before the model classification, this can best reflect
# the true relationship between samples.
def extract_features(model, dataloader):

    # Use this array to collect features.
    features = []

    # Save the output of the current layer into features
    # list for subsequent analysis during the forward
    # propagation of the model.
    def grab_data(model, input, output):
        features.append(output.cpu())

    # Grab the output feature of the second last layer.
    data_collect = model[1][-2].register_forward_hook(grab_data)

    # Use the validation set to test the model, and collect
    # their real labels.
    _, real_labels = learn.get_preds(dl=dataloader)

    # Remove the hook to avoid repeated triggering when
    # the model is run later.
    data_collect.remove()

    # Transform the collected data into the form of numpy
    x_subset = torch.cat(features).numpy()
    y_subset = real_labels.numpy()
    return x_subset, y_subset
```

```
# Use the tsne algorithm to reduce the high-dimensional feature
# x to two dimensions and outputs a two-dimensional
# representation x_tsne that is easy to visualize.
def tsne_process(x):
    tsne = TSNE(n_components=2, perplexity=30,
                n_iter=1000, random_state=42)
    x_tsne = tsne.fit_transform(x)
    return x_tsne

# Draw the t-SNE graph.
def plot_tsne(x_tsne, y_subset, labels):

    plt.figure(figsize=(12, 8))
    scatter = plt.scatter(x_tsne[:, 0], x_tsne[:, 1], c=y_subset.astype(int), cm
    plt.legend(handles=scatter.legend_elements()[0], labels=labels, title="Class
    plt.title("t-SNE Analysis")
    plt.grid(True)
    plt.show()

x_subset, y_subset = extract_features(learn.model, dls.valid)
x_tsne = tsne_process(x_subset)
plot_tsne(x_tsne, y_subset, dls.vocab)
```

```
c:\Users\henry\anaconda3\Lib\site-packages\sklearn\manifold\_t_sne.py:1162: Futur
eWarning: 'n_iter' was renamed to 'max_iter' in version 1.5 and will be removed i
n 1.7.
  warnings.warn(
```



From the t-SNE analysis graph, we find that the five classes are classified very obviously.