

Yan Zhang 的 Graduation-Design-Project

Author name: Yan Zhang

Student ID: 22010384

Supervisor: Jasper Zheng

Creative Computing Institute: Creative computing

WebLog

1. Project Description.

The core of the project is to train a clothing recognition model based on the DeepxFashion2 dataset using the Yolov8 algorithm. In order to improve the processing ability and efficiency of complex image data, we integrated MHSA (Multi-Head Self-Attention) and CBAM into YOLOv8 for optimisation, and finally chose to use MHSA as the attention mechanism of C2f to train the model by comparing the model training results. The system can accurately identify clothing items in images in complex environments and recommend fashion items that match the user's current style, thus providing tailored personalised dressing recommendations.

2. Algorithm Reference

For Yolov8, refer to the official Github: <https://github.com/ultralytics/ultralytics>.

For the prototype of CABM, please refer to :

<https://github.com/Jongchan/attention-module>

For MHSA reference: <https://github.com/tranleanh/yolov4-mhsha>

3. Source of dataset

The dataset used in this project is from DeepFashion2:

<https://github.com/switchablenorms/DeepFashion2>

4. Model evaluation metrics

mAP, Precision, Recall, FPS, IoU, all data are saved under result path for analysis after training. Reference blog: <https://blog.csdn.net/java1314777/article/details/134154676>

June 20

Met with Supervisor Jasper for the first time. Discussed the requirements for the graduation project and developed a plan for the summer.

June 20

Met with Supervisor Jasper for the first time. Discussed the requirements for the

graduation project and developed a plan for the summer.

June 27

Selected a few potential topics for the graduation project based on our previous discussions and met with Jasper to discuss the feasibility of these thesis directions.

June 27 - July 15

After reading literature and consulting with my supervisor, I finally decided on my thesis topic: an intelligent system related to fashion.

July 15 - August 10

Read and organized articles related to the thesis topic. Understood the current trends and technologies in fashion recommendation systems.

August 15

After reading literature and comparative studies, decided to use the DeepFashion2 dataset to train my model because it has collected a large number of fashion images and annotations.

August 15 - August 25

Started preprocessing the dataset to facilitate later training.

Data preprocessing and format conversion: the DeepFashion2Yolo_thread.py (Figure 2) script was used to convert the DeepFashion2 dataset annotations into a format compatible with the YoloV8 model. This conversion includes converting bounding box coordinates and class labels.

```

1 # Import the modules we need
2 import sys
3 import math
4 import random
5 import time
6
7 # A function to produce a batch of images and their labels
8 def get_batch(batch_size, num_classes, image_paths, labels, num_epochs):
9     # Create a list of image paths and labels
10     image_paths = image_paths[:batch_size]
11     labels = labels[:batch_size]
12
13     # Shuffle the image paths and labels
14     random.shuffle(image_paths)
15     random.shuffle(labels)
16
17     # Create a batch of images and labels
18     images = []
19     labels = []
20     for i in range(batch_size):
21         image_path = image_paths[i]
22         label = labels[i]
23         image = load_image(image_path)
24         images.append(image)
25         labels.append(label)
26
27     return images, labels
28
29 # A function to load an image from a file
30 def load_image(image_path):
31     # Open the image file
32     image = open(image_path, 'rb')
33     # Load the image data
34     image_data = image.read()
35     # Close the image file
36     image.close()
37
38     # Convert the image data to a NumPy array
39     image_array = numpy.frombuffer(image_data, dtype=numpy.uint8).reshape([1, 28, 28, 3])
40
41     # Return the image array
42     return image_array
43
44 # A function to calculate the loss of a batch of images and labels
45 def calculate_loss(images, labels, num_epochs):
46     # Calculate the loss for each epoch
47     for epoch in range(num_epochs):
48         # Calculate the loss for each batch
49         for batch in range(batch_size):
50             images, labels = get_batch(batch_size, num_classes, image_paths, labels, num_epochs)
51             loss = calculate_loss_batch(images, labels)
52             total_loss += loss
53
54     # Return the total loss
55     return total_loss
56
57 # A function to calculate the loss of a batch of images and labels
58 def calculate_loss_batch(images, labels):
59     # Calculate the loss for each batch
60     for i in range(batch_size):
61         image = images[i]
62         label = labels[i]
63         loss = calculate_loss_image(image, label)
64         total_loss += loss
65
66     # Return the total loss
67     return total_loss
68
69 # A function to calculate the loss of a single image and label
70 def calculate_loss_image(image, label):
71     # Calculate the loss for a single image and label
72     loss = 0
73     for i in range(1, 28):
74         for j in range(1, 28):
75             for k in range(1, 3):
76                 loss += (image[i, j, k] - label[i, j, k]) ** 2
77
78     # Return the loss
79     return loss
80
81 # A function to train a neural network
82 def train_neural_network(image_paths, labels, num_epochs):
83     # Calculate the total loss
84     total_loss = calculate_loss(image_paths, labels, num_epochs)
85
86     # Print the total loss
87     print('Total loss: %f' % total_loss)
88
89     # Return the total loss
90     return total_loss
91
92 # A function to test a neural network
93 def test_neural_network(image_paths, labels, num_epochs):
94     # Calculate the total loss
95     total_loss = calculate_loss(image_paths, labels, num_epochs)
96
97     # Print the total loss
98     print('Total loss: %f' % total_loss)
99
100     # Return the total loss
101     return total_loss
102
103 # A function to save a neural network
104 def save_neural_network(image_paths, labels, num_epochs):
105     # Calculate the total loss
106     total_loss = calculate_loss(image_paths, labels, num_epochs)
107
108     # Print the total loss
109     print('Total loss: %f' % total_loss)
110
111     # Return the total loss
112     return total_loss
113
114 # A function to load a neural network
115 def load_neural_network(image_paths, labels, num_epochs):
116     # Calculate the total loss
117     total_loss = calculate_loss(image_paths, labels, num_epochs)
118
119     # Print the total loss
120     print('Total loss: %f' % total_loss)
121
122     # Return the total loss
123     return total_loss
124
125 # A function to evaluate a neural network
126 def evaluate_neural_network(image_paths, labels, num_epochs):
127     # Calculate the total loss
128     total_loss = calculate_loss(image_paths, labels, num_epochs)
129
130     # Print the total loss
131     print('Total loss: %f' % total_loss)
132
133     # Return the total loss
134     return total_loss
135
136 # A function to visualize a neural network
137 def visualize_neural_network(image_paths, labels, num_epochs):
138     # Calculate the total loss
139     total_loss = calculate_loss(image_paths, labels, num_epochs)
140
141     # Print the total loss
142     print('Total loss: %f' % total_loss)
143
144     # Return the total loss
145     return total_loss
146
147 # A function to compare a neural network
148 def compare_neural_network(image_paths, labels, num_epochs):
149     # Calculate the total loss
150     total_loss = calculate_loss(image_paths, labels, num_epochs)
151
152     # Print the total loss
153     print('Total loss: %f' % total_loss)
154
155     # Return the total loss
156     return total_loss
157
158 # A function to optimize a neural network
159 def optimize_neural_network(image_paths, labels, num_epochs):
160     # Calculate the total loss
161     total_loss = calculate_loss(image_paths, labels, num_epochs)
162
163     # Print the total loss
164     print('Total loss: %f' % total_loss)
165
166     # Return the total loss
167     return total_loss
168
169 # A function to debug a neural network
170 def debug_neural_network(image_paths, labels, num_epochs):
171     # Calculate the total loss
172     total_loss = calculate_loss(image_paths, labels, num_epochs)
173
174     # Print the total loss
175     print('Total loss: %f' % total_loss)
176
177     # Return the total loss
178     return total_loss
179
180 # A function to profile a neural network
181 def profile_neural_network(image_paths, labels, num_epochs):
182     # Calculate the total loss
183     total_loss = calculate_loss(image_paths, labels, num_epochs)
184
185     # Print the total loss
186     print('Total loss: %f' % total_loss)
187
188     # Return the total loss
189     return total_loss
190
191 # A function to benchmark a neural network
192 def benchmark_neural_network(image_paths, labels, num_epochs):
193     # Calculate the total loss
194     total_loss = calculate_loss(image_paths, labels, num_epochs)
195
196     # Print the total loss
197     print('Total loss: %f' % total_loss)
198
199     # Return the total loss
200     return total_loss
201
202 # A function to monitor a neural network
203 def monitor_neural_network(image_paths, labels, num_epochs):
204     # Calculate the total loss
205     total_loss = calculate_loss(image_paths, labels, num_epochs)
206
207     # Print the total loss
208     print('Total loss: %f' % total_loss)
209
210     # Return the total loss
211     return total_loss
212
213 # A function to log a neural network
214 def log_neural_network(image_paths, labels, num_epochs):
215     # Calculate the total loss
216     total_loss = calculate_loss(image_paths, labels, num_epochs)
217
218     # Print the total loss
219     print('Total loss: %f' % total_loss)
220
221     # Return the total loss
222     return total_loss
223
224 # A function to save a neural network
225 def save_neural_network(image_paths, labels, num_epochs):
226     # Calculate the total loss
227     total_loss = calculate_loss(image_paths, labels, num_epochs)
228
229     # Print the total loss
230     print('Total loss: %f' % total_loss)
231
232     # Return the total loss
233     return total_loss
234
235 # A function to load a neural network
236 def load_neural_network(image_paths, labels, num_epochs):
237     # Calculate the total loss
238     total_loss = calculate_loss(image_paths, labels, num_epochs)
239
240     # Print the total loss
241     print('Total loss: %f' % total_loss)
242
243     # Return the total loss
244     return total_loss
245
246 # A function to evaluate a neural network
247 def evaluate_neural_network(image_paths, labels, num_epochs):
248     # Calculate the total loss
249     total_loss = calculate_loss(image_paths, labels, num_epochs)
250
251     # Print the total loss
252     print('Total loss: %f' % total_loss)
253
254     # Return the total loss
255     return total_loss
256
257 # A function to visualize a neural network
258 def visualize_neural_network(image_paths, labels, num_epochs):
259     # Calculate the total loss
260     total_loss = calculate_loss(image_paths, labels, num_epochs)
261
262     # Print the total loss
263     print('Total loss: %f' % total_loss)
264
265     # Return the total loss
266     return total_loss
267
268 # A function to compare a neural network
269 def compare_neural_network(image_paths, labels, num_epochs):
270     # Calculate the total loss
271     total_loss = calculate_loss(image_paths, labels, num_epochs)
272
273     # Print the total loss
274     print('Total loss: %f' % total_loss)
275
276     # Return the total loss
277     return total_loss
278
279 # A function to optimize a neural network
280 def optimize_neural_network(image_paths, labels, num_epochs):
281     # Calculate the total loss
282     total_loss = calculate_loss(image_paths, labels, num_epochs)
283
284     # Print the total loss
285     print('Total loss: %f' % total_loss)
286
287     # Return the total loss
288     return total_loss
289
290 # A function to debug a neural network
291 def debug_neural_network(image_paths, labels, num_epochs):
292     # Calculate the total loss
293     total_loss = calculate_loss(image_paths, labels, num_epochs)
294
295     # Print the total loss
296     print('Total loss: %f' % total_loss)
297
298     # Return the total loss
299     return total_loss
300
301 # A function to profile a neural network
302 def profile_neural_network(image_paths, labels, num_epochs):
303     # Calculate the total loss
304     total_loss = calculate_loss(image_paths, labels, num_epochs)
305
306     # Print the total loss
307     print('Total loss: %f' % total_loss)
308
309     # Return the total loss
310     return total_loss
311
312 # A function to benchmark a neural network
313 def benchmark_neural_network(image_paths, labels, num_epochs):
314     # Calculate the total loss
315     total_loss = calculate_loss(image_paths, labels, num_epochs)
316
317     # Print the total loss
318     print('Total loss: %f' % total_loss)
319
320     # Return the total loss
321     return total_loss
322
323 # A function to monitor a neural network
324 def monitor_neural_network(image_paths, labels, num_epochs):
325     # Calculate the total loss
326     total_loss = calculate_loss(image_paths, labels, num_epochs)
327
328     # Print the total loss
329     print('Total loss: %f' % total_loss)
330
331     # Return the total loss
332     return total_loss
333
334 # A function to log a neural network
335 def log_neural_network(image_paths, labels, num_epochs):
336     # Calculate the total loss
337     total_loss = calculate_loss(image_paths, labels, num_epochs)
338
339     # Print the total loss
340     print('Total loss: %f' % total_loss)
341
342     # Return the total loss
343     return total_loss
344
345 # A function to save a neural network
346 def save_neural_network(image_paths, labels, num_epochs):
347     # Calculate the total loss
348     total_loss = calculate_loss(image_paths, labels, num_epochs)
349
350     # Print the total loss
351     print('Total loss: %f' % total_loss)
352
353     # Return the total loss
354     return total_loss
355
356 # A function to load a neural network
357 def load_neural_network(image_paths, labels, num_epochs):
358     # Calculate the total loss
359     total_loss = calculate_loss(image_paths, labels, num_epochs)
360
361     # Print the total loss
362     print('Total loss: %f' % total_loss)
363
364     # Return the total loss
365     return total_loss
366
367 # A function to evaluate a neural network
368 def evaluate_neural_network(image_paths, labels, num_epochs):
369     # Calculate the total loss
370     total_loss = calculate_loss(image_paths, labels, num_epochs)
371
372     # Print the total loss
373     print('Total loss: %f' % total_loss)
374
375     # Return the total loss
376     return total_loss
377
378 # A function to visualize a neural network
379 def visualize_neural_network(image_paths, labels, num_epochs):
380     # Calculate the total loss
381     total_loss = calculate_loss(image_paths, labels, num_epochs)
382
383     # Print the total loss
384     print('Total loss: %f' % total_loss)
385
386     # Return the total loss
387     return total_loss
388
389 # A function to compare a neural network
390 def compare_neural_network(image_paths, labels, num_epochs):
391     # Calculate the total loss
392     total_loss = calculate_loss(image_paths, labels, num_epochs)
393
394     # Print the total loss
395     print('Total loss: %f' % total_loss)
396
397     # Return the total loss
398     return total_loss
399
400 # A function to optimize a neural network
401 def optimize_neural_network(image_paths, labels, num_epochs):
402     # Calculate the total
```

1.2 Verify that the dataset has been correctly converted using "showLabel.py".

```

def train_model(model, data_loader, device, num_epochs=100):
    # Training loop
    for epoch in range(num_epochs):
        # Training phase
        model.train()
        for batch_idx, (images, targets) in enumerate(data_loader.train_loader):
            images = images.to(device)
            targets = targets.to(device)

            # Forward pass
            outputs = model(images)

            # Loss function
            loss = criterion(outputs, targets)

            # Backward pass and optimization
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        # Validation phase
        model.eval()
        val_loss = 0
        for batch_idx, (images, targets) in enumerate(data_loader.val_loader):
            images = images.to(device)
            targets = targets.to(device)

            # Forward pass
            outputs = model(images)

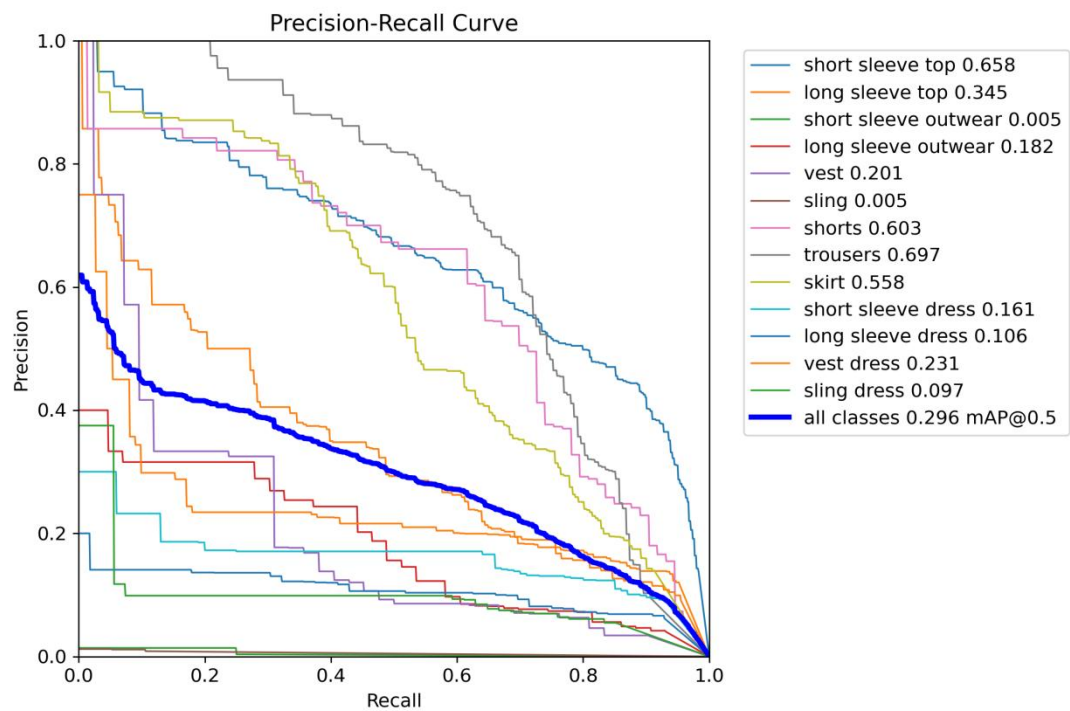
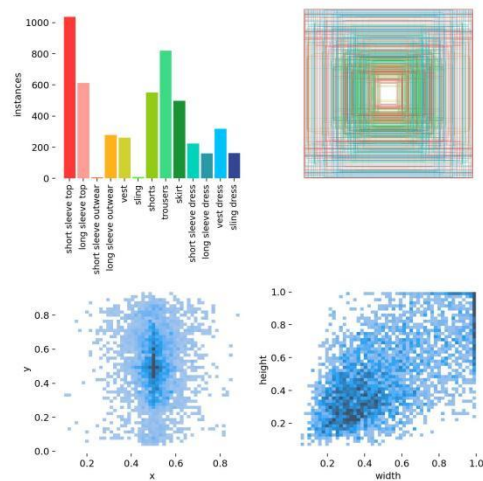
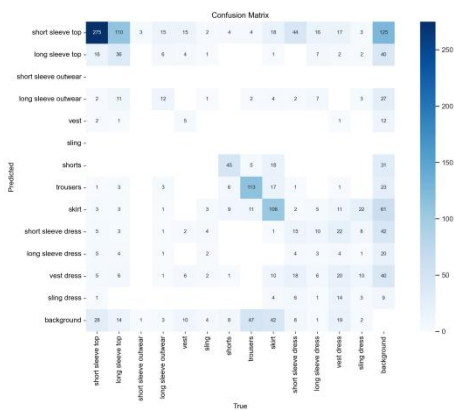
            # Loss function
            val_loss += criterion(outputs, targets)

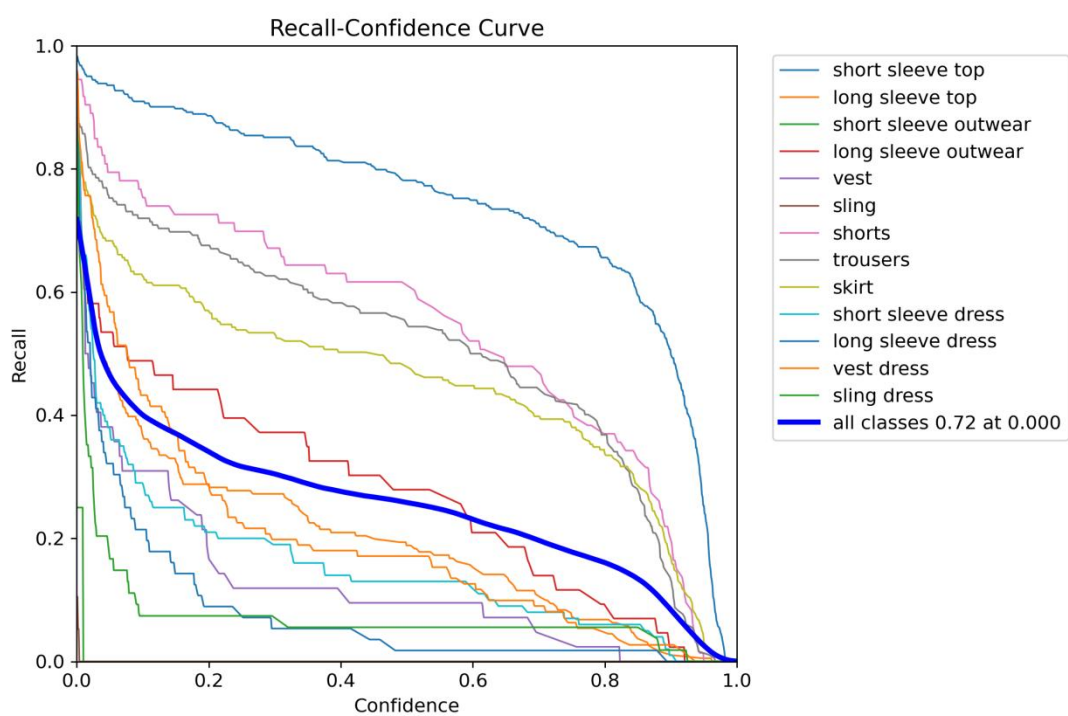
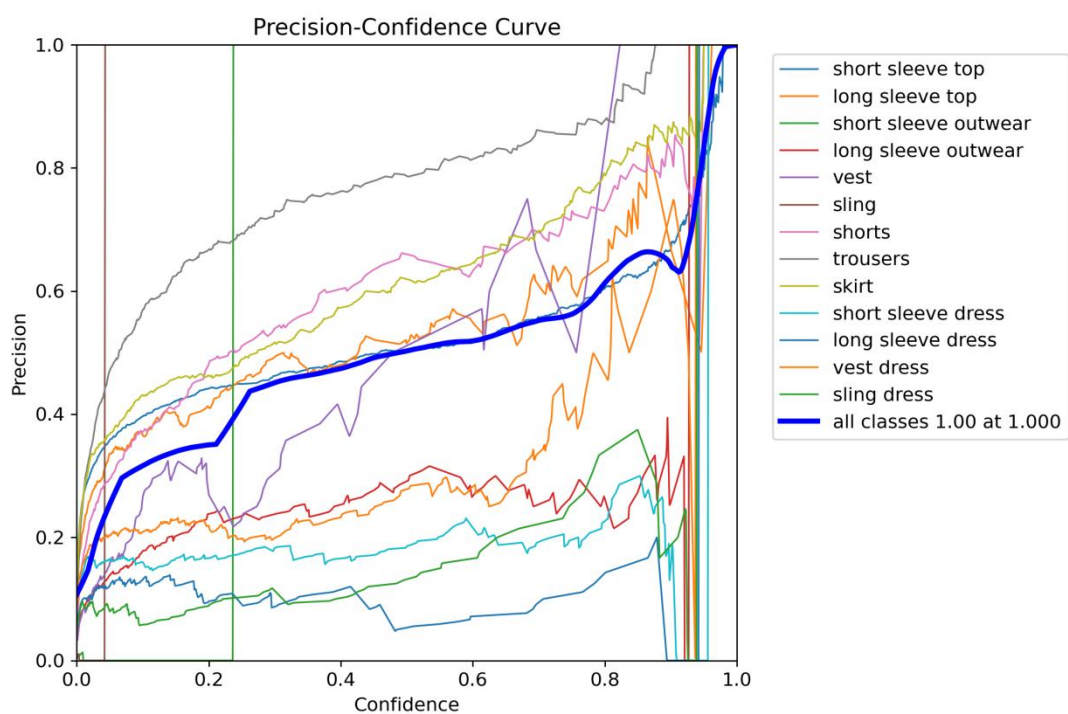
        # Print training results
        print(f'Epoch {epoch+1}: Training Loss: {loss.item():.4f}, Validation Loss: {val_loss/len(data_loader.val_loader):.4f}')

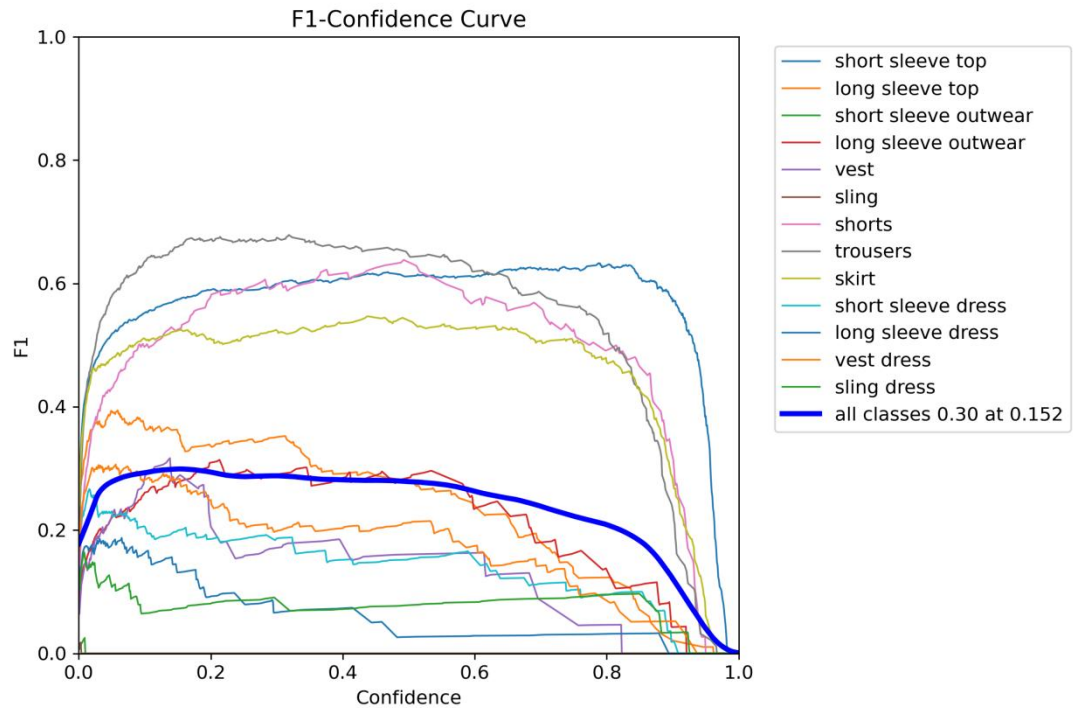
    # Save the trained model
    torch.save(model.state_dict(), 'model.pth')

```

Training results:







From the results, it can be seen that there is still much room for optimisation of the model.

August 26

After various comparisons and literature analysis, decided to use the YOLO series model for clothing detection and recognition.

August 26 - September 10

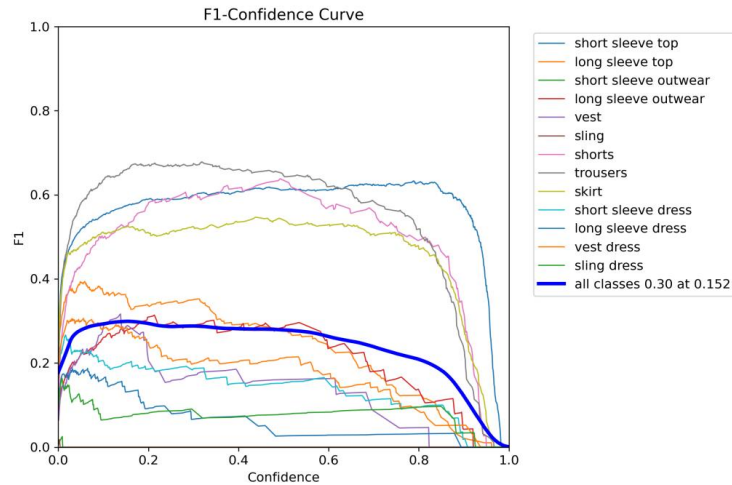
Initially chose the YOLOv4 model but found its accuracy to be low. Finally selected YOLOv8.

September 10 - October 1

Trained the YOLOv8 model using the DeepFashion2 dataset. Due to equipment limitations, only the first 3000 images of the dataset were used for training.

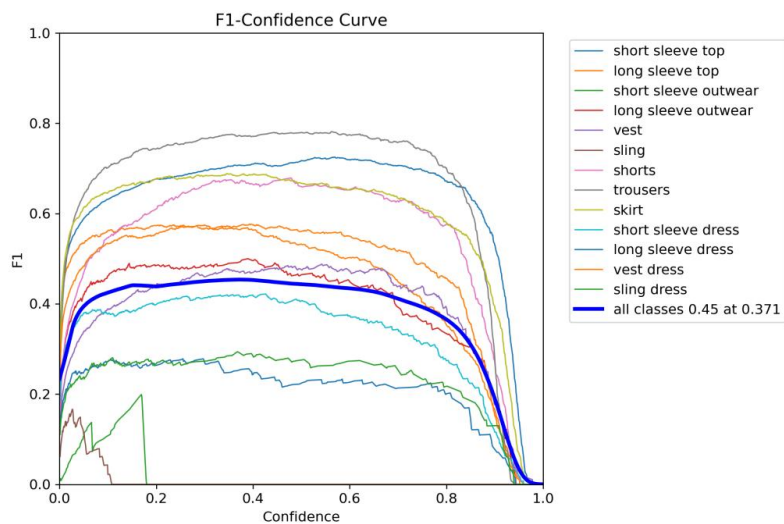
October 1 - October 10

Integrated CBAM into YOLOv8 for optimization, but there was not much improvement in the model training results.



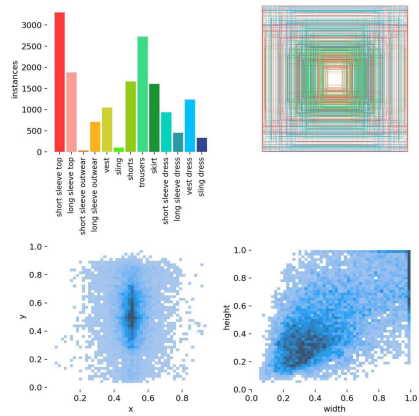
October 10 - October 25

Integrated MHSA into YOLOv8 for optimization. After training model comparison, finally chose to use MHSA as the attention mechanism for C2f in the model training.



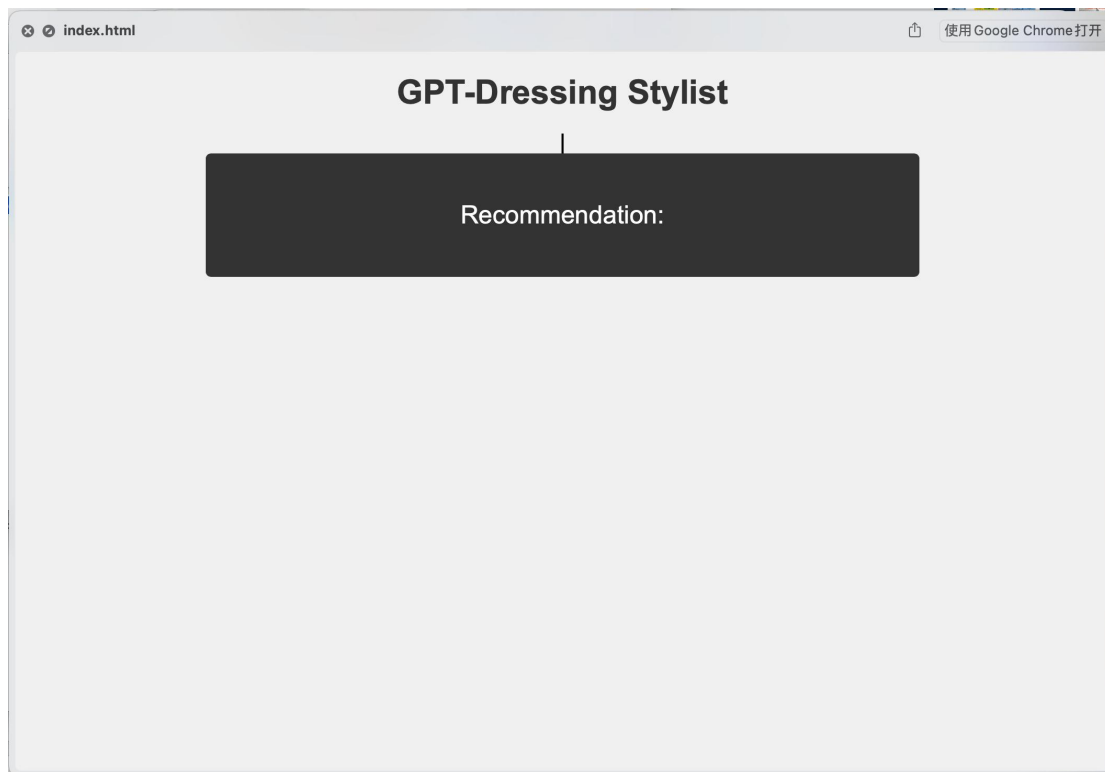
October 25 - November 10

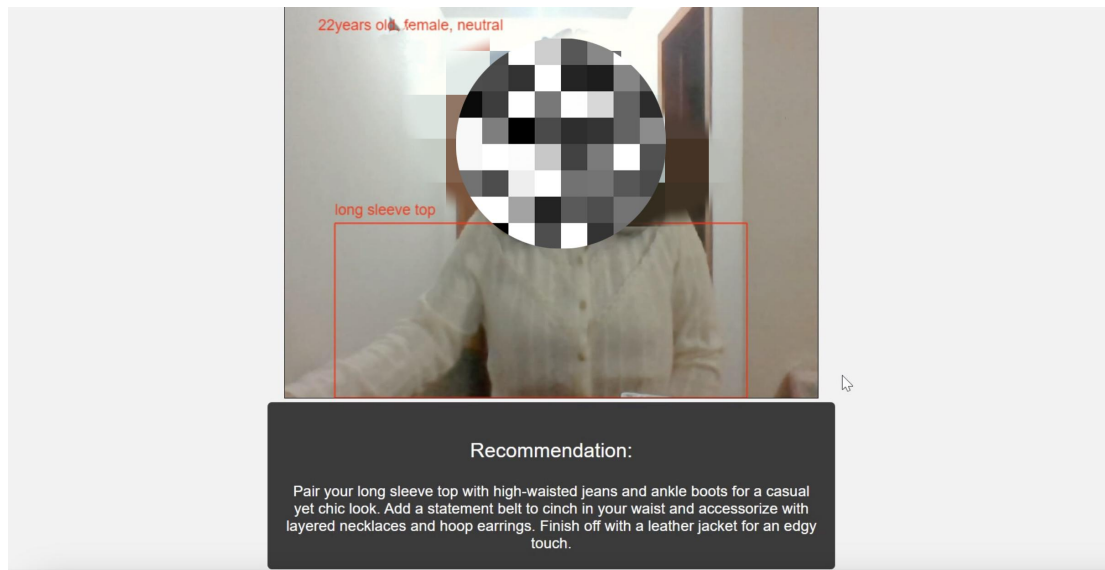
Expanded the training set to the first 10,000 images of the dataset and retrained the model to improve its accuracy and generality.



November 10 - November 15

Worked on system integration and the development of a visual interface.





November 15 - November 22

Organized the literature learned during the process and completed the thesis.