Wake up in a virtual campus

Liu Yuzhou(21100602d)

Responsable of the experiment or other information

1. Abstract

Nowadays, generative AI has become a hot topic, and it is sometimes hard to distinguish real images and AI-generated images. In a misuse scenario, the fake image can even be used to cheat people. In this report, we will focus on how to find a way to distinguish real images from AI-generated ones. Furthermore, if the image is classified as an AI-generated image, are we able to see which class it belongs to?
So, we have two tasks: the first is to identify whether the image is generated by an AI generator, and the second is to classify the style of the AI-generated image.

Keywords-AI-generated image, machine learning, computer vision

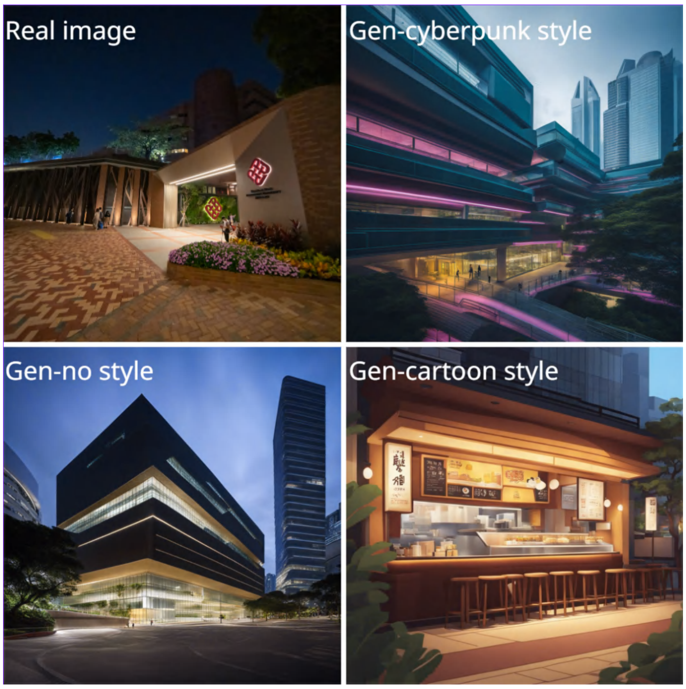

Figure 1. *Different types of image*

In this report, I will introduce:
(1)How I define the problem and how the dataset is collected
(2)what kind of method have I tried to build the model
(3)how the model is trained, validated and tested, and how it performs in a real campus scenario
(4)what problems I have encountered and how I solve them.

2. Setting up

2.1. Problem definition

Our task is to develop a classification model which can do binary classification between the real word image of the PolyU campus and the AI-generated one (**Task 1**). Furthermore, the model will also be able to tell the style of the generated image (**Task 2**).

To address it as a computer vision task, we should leverage the features of the image and classify them. We could have multiple methods to extract the features of the images, either by using some processing technique or applying some deep learning method. The **convolutional neural network(CNN)** could be most appropriate for this task, as convolution operation can efficiently detect edge information, and it could be helpful in distinguishing the difference between AI-generated images and real images. We can see that the edges of AI-generated images are always much clearer than the real images. In this case, this paper implements based on different kinds of architectures that contain a set of CNN layers, either built by myself or by pre-trained models.

Computer vision should be suitable for this task. Because it is powerful for analysing and understanding different types of images. By combining different sets of CV techniques, we could complete this task successfully; from the **model** perspective, this report will explore the power of VGG net and use the Siamese network for fine-tuning our model; From the **image processing** perspective, this report will also try to apply single feed or multi-feed technique in order to improve the performance of the model. I believe we could achieve a decent result on this image classification task by using these techniques.

2.2. Dataset

The dataset is collected by myself. There are two types of images; one is real images taken by myself from the PolyU campus (Some are from the official gallery), and the other is Generated images by PolyU GenAI. Furthermore, the image generated by PolyU GenAI is divided into three subclasses: Cyberpunk style, No style, and cartoon style. For samples from different sets of images, you can refer to **Figure 1**. I have 30 real images (Later collected 16 indoor images) and 14 for each subset of AI-generated images.

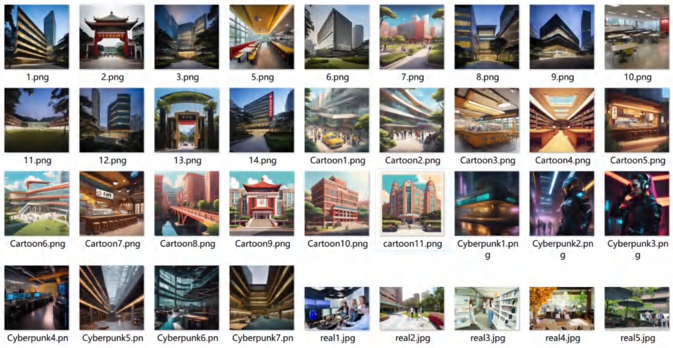

Figure 2. *Part of collected dataset*

Then, we try to view the distribution of different kinds of images. We can observe from **Figure 3** that the colour distribution for different types of images is quite different, especially between real images and AI-generated ones.

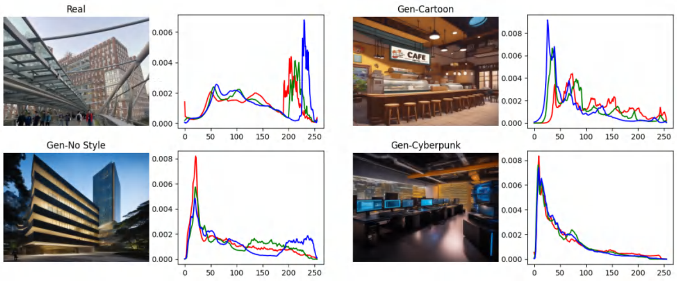

Figure 3. *Color distribution for different kind of images*

Hopefully, we can make full use of this information in our training process.

3. Methodology

3.1. Overall

In this section, we will first try to test the power of **CNN** on a simple model, then try to use more complicated models such as **VGG-net** to improve our performance. Finally, we will tune our model by Siamese network and use the idea of **VLAD** to do the prediction.
We will evaluate the performance separately for:
Task 1: Distinguish between real images and AI-generated ones.
Task 2: Distinguish between different styles of images.

3.2. Train with a simple CNN

3.2.1. Training & Performance.
I first attempted to try to test the power of CNN on feature extraction. So I just simply built a simple CNN model, which contains 2 CNN layers and 3 linear layers. The performance will be evaluated by 2 from two perspectives. The first is whether it can distinguish AI-generated images and whether the classification between different styles is correct.

Model/Classifier	Accuracy
CNN(Task 1)	60%
CNN(Task 2)	53.33%

Table 1. *Simple CNN performance*

3.2.2. Difficulties & trials on improvement.
1. (**Converge slowly**) The loss of the model converges slowly, as we need to train over 100 epochs in order to minimise the loss close to 0.
2. (**Improve the performance**) Because we have limited training data, so in order to achieve a better performance, we need to prevent the model from overfitting; here, I applied the dropout strategy to let the prediction not over-dependent on the part of neurons. And the accuracy for both tasks has improved a lot (**Task 1**: from 50% to 60%,**Task 2**: from 30% to 53.33%).

3.2.3. Reflection.
As the model converges slowly, and the performance does not meet our expectations. I will try to use some pre-trained models in the next method, as the models have already been trained based on some of the image datasets (The volume of data is larger). And as the parameters are pre-trained (May converge quicker).

3.3. Train with VGG+Self designed architecture

3.3.1. Model explanation.
Since VGG-net is trained based on a large image dataset for classification tasks and has stacked a bunch of CNN layers. I believe it can extract the feature nicely for the images we have. Here, I will remove the classification layer of the VGG net. Our training will only based on the feature extraction layers of VGG net. Hopefully, we can utilize the strong feature map extracted by VGG net to perform our downstream task.

Though originally, the VGG net was designed for object classification, I think these two tasks should share strong similarities, like colour and edge..., so this VGG model should be rather appropriate.

Two different architectures:
(1)Model 1: The first architecture I tried simply used **three linear layers**, and there will be four channels for output, and the channel obtained the highest score will be considered as the class label predicted.
(2)Model 2: The second architecture is to get the intermediate output from the linear layer (As shown in **Figure 4**). Because the output is histograms, so I just simply consider it as a Bag of visual words (**BOW**) and try to use clustering algorithms to obtain the result. Hopefully, the histograms can be considered as a strong representation of the image.

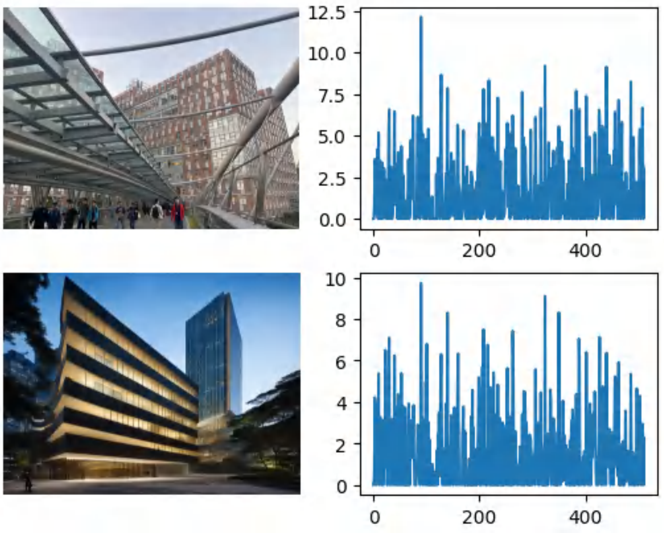

Figure 4. *Intermediate output for different images*

3.3.2. Training.
Model 1:
We can view from **Figure 5** that the loss training set dropped rapidly, nearly approaching 0 around the third epoch, and the accuracy for the validation set keeps rising after this moment. This means the best training result can be obtained at the third epoch. That means we could stop our training earlier before the result gets worse. The result can be viewed from **Table 2**, and we can see it achieved 80% accuracy on AI-generated image distinguish.

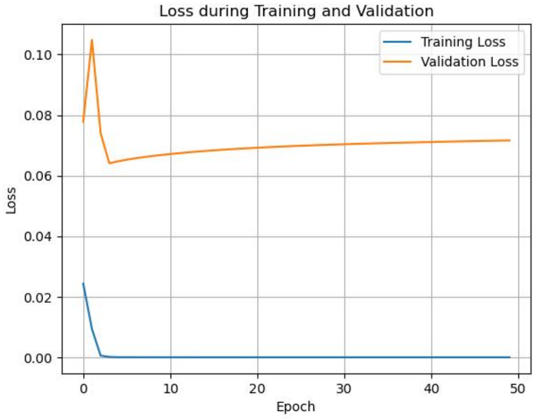

Figure 5. *Change of loss*

Model 2:
We used different kinds of models to train based on the histograms extracted by the linear layers. And there are 512 columns for each histogram so that we can regard them as different visual words. We observe from **Table 2** that the SVM model has achieved the highest accuracy for AI-image distinguish, which is 93.33%(**Task1**). As the validation set has 15 images, there is only one image misclassified!!! And the classification of different styles of images is also quite decent, which is 80%(**Task2**).

Model/Classifier	Accuracy
VGG + 3 linear layers(task1)	80%
VGG + 3 linear layers(task2)	73.33%
VGG + SVM(task1)	93.33%
VGG + SVM(task2)	80%
VGG + (Random Forest)RFC(Task1)	86.67%
VGG + RFC(Task2)	73.33%

Table 2. *VGG16-net performance*

3.3.3. Difficulties & trials on improvement.

1. (**Initial poor training result**) When I am working with **Model 1**, the result is poor, but after checking the change of loss between the validation set from **Figure 5** that the loss converges around 3 epoch, then I use the Early stopping strategy to improve my performance (Stop is the loss for validation set has not improved for 5 epoch, store the best model). Finally, our result improved.
2. (**Better result for Model 2**) It Seems that whatever traditional model we are using, the result for **Model 2** is generally better than **Model 1**; maybe here the problem for the neural network is overfitting so that the result is not always best. By combining simple models, we can boost its performance, and the result is really out of my expectations.
3. (**Cannot show cam on image**) I tried hard in order to view the **CAM** method for VGG network, but didn't succeed, because seems the implementation for VGG-CAM is based on the classes it outputted, and we need to manipulate the code in CAM library, which could be too hard. CAM will be helpful for us because it will tell us from which aspect the image will be classified as an AI-generated image or a Real image.
4. (**Potential improvement**) Maybe we could apply the Self-attention layer to let the model focused on more representative part of the image, in order to generate better classification result.
5. (**problem of early stopping**) Sometimes, if we are pretty unlucky, the training may start at a bad point, and the result keeps fluctuant; it is possible that the training stopped too early, but we haven't reached the best point of the training.

3.3.4. Reflection.

We find the model converges fast here (like after eight epochs), which means that different kinds of images all share high similarity (The knowledge is similar). By applying the traditional learning method together with the strong feature extraction power of the deep learning method, we can reach a decent result, which I did not expect before seeing its power.

3.4. Patchify+Siamonse+VGG feature extract(Thought of)

3.4.1. Model explanation.

How could I optimise? as the data collected by humans could always be limited. So, I need to find a way to enlarge the dataset. Is that possible to get an enlarged dataset from what we have?

The answer is definitely yes! We could patchify the images, and we can determine the class of the image by looking into different patches. If the majority of the patches vote for one of the classes, it is more likely that it belongs to such a class. We could even train the patchified images to compare with each other even if originally they belong to the same image. As shown in **Figure 6**, I have divided an image into **nine** different patches. Though theoretically, patchify with overlapping makes our model more powerful, but as the training takes a pretty long time, so we just want to see if the method is helpful in improving our training result.

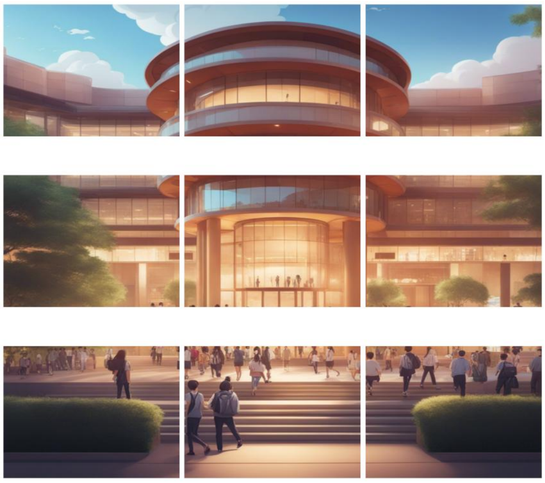

Figure 6. *Intermediate output for different images*

The idea is first to split the image into pieces. Then, we reshape each piece of the image and forward it to the network we trained previously. Next, the job will be done by a Siamese network. We first train the network and then use the trained network to calculate a similarity score for each patch of the image.

I think the next thing is quite innovative. We treat each patch of the image as a **visual word**, and we will give an id to each of the visual words. We also have a label for each visual word(Corresponding to which image it belongs); now, we can create a query to visualise its near neighbours. The result is shown in **Figure 7**.

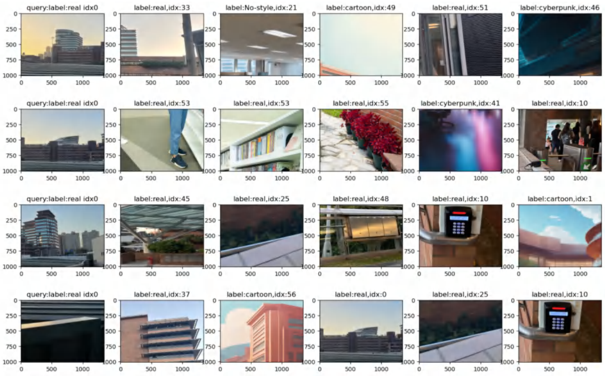

Figure 7. *Query result for different patch(idx of the image)*

Here, I will treat all the sub-images in the training dataset as our database. And use the images in the validation set to form queries. Next, we are going to create a histogram for each image. Here, I could come up with two methods to do that: **(1)** based on the six nearest images of each sub-image; if there is a hit, add the similarity to the corresponding position of the query image. **(2)** Build a histogram for each patch independently, and finally, make the decision for the whole image based on the voting of different sub-patches.

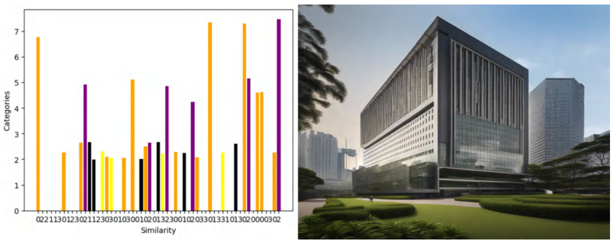

Figure 8. *Reason for not choosing Way1*
***black**: cyberpunk, **orange**: real, **yellow**: no style, **purple**: cartoon*

I am not going to work with method **(1)**, as we can observe in **Figure 8** (Which shows the similarity of the image to each image in the training set, and color shows the class of the corresponding training image), the classification result is kind of unsatisfied for **(1)** as originally the image should be classified as class 3 (Gen-Nostyle), but it shows that it is more likely to be classified as class 0 (Real image), I think the potential reason is that some patches of the image may be misclassified, so it can mislead the classification of the whole image (for example if we got a piece of sky, maybe it is difficult to distinguish it between AI-generated image, but the misclassified result for this part can influence the whole image).

Then, I am going to work with method **(2)**. To show more detail of way 2, we still can have two ways in order to make the prediction. The **first way** is to obtain the score for each patch based on all the training images and then use an SVC model to make the prediction. The **second way** is just to do hard voting for each patch to obtain its class. Then, based on the classification of different patches, we will do a voting prediction for the whole image.

Model/Classifier	Accuracy
SVM+score on all train image (task1)	55.3%
Hard voting (task1)	63%

Table 3. *Result for two ways of method 2*

We can see from Table 3 that the result is quite unsatisfactory. I think it could caused by the poor representative of the score obtained by the **Siamese Network**, so I decided to give a trial on just simply using the feature vector of each patch directly. Hopefully, it will improve the results we obtained previously.

Surprisingly, the result is really decent, as we have achieved decent accuracy for both **task 1** and **task 2**. Although for some of the patches, the classification result is quite poor, when we apply the voting strategy to obtain the result of the whole image, it is quite decent. We can observe the result from **Table 4**, which shows the classification result and the whole image separately.

Model/Classifier	Accuracy
SVM + patch_classification (task1)	72.3%
SVM + patch_classification (task2)	62.2%
SVM + whole image_classification (task1)	100%
SVM + whole image_classification (task2)	93.3%

Table 4. *Result for two ways of method 2*

We can notice that the classification result for different patches is kind poor, but the result is decent for the whole image from **Figure 9**. The image should belong to the class Cyberpunk. We can see that for some of the patches, the classification result is poor, but the majority of the patches are correct, so the overall classification result is correct.

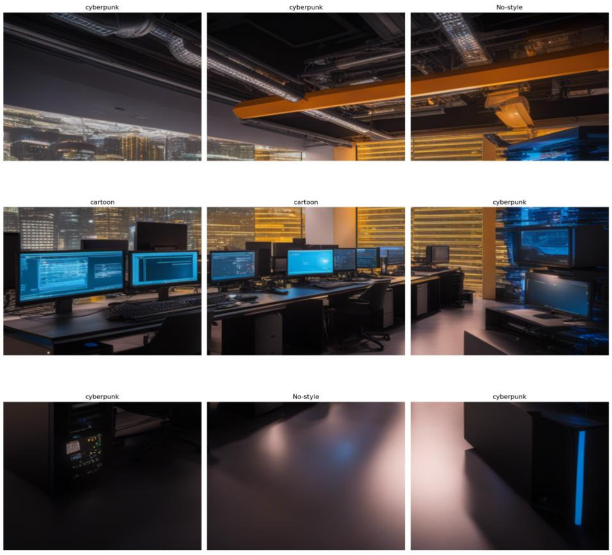

Figure 9. *classification result for different patches (Cyberpunk)*

3.4.2. *Difficulties & trials on improvement.*

1. (**Poor result for Siamese**) The training result is poor for the Siamese network combined with the VGG feats, maybe it is only suitable for detecting similar objects in different images, but not good for capturing the style information for different images. By going through such a network, we missed a lot of information; the final result is poor compared to directly using a **SVM (Linear)** model to do the prediction based on feature vectors.
2. (**Patchify improved performance**) We can see that patchify is a really helpful technique here, because the result boosts for both **task 1** and **task 2**, as our model can detect whether it is an AI-generated image or not only not, but also accurately detect the style of an AI-generated image. The **potential reason** could be **point 1**: we force our model to look into more details in different areas of the image, maybe some of the areas are not representative, so it would easily be misclassified. But as the majority of them will be correct, the overall classification result should be decent. **Point 2**: patchify could enlarge our dataset, as we could take pictures in more detail. For example, originally, we only had 57 training images, but by splitting each one into 9, we could obtain a dataset which contains 513 different images.
3. (**Optimize voting strategy**) Initially, we treat different classes equally and just pick the majority as the voting result, the result is quite poor. But after thinking, I noticed that cyberpunk, cartoon, and no-style are three subtypes of AI-generated images, so I need to consider them as a whole. So I changed the voting strategy to (As we have 9 patches): At least 5 patches are being classified as real, an image can be classified as a real image; if it is not real, we will then do majority voting to determine its generate style.
4. (**Kernel memory leak**) At the starting point, I need to collect the feature vector from the VGG-net trained in **Section 3.3**. However, the kernel memory gets leaked because of the large amount of data (As shown in **Figure 10**); after searching, I found it is caused by the type of tensor; I need to change the require_grad for each of them to false, then I find the storage is not costly.

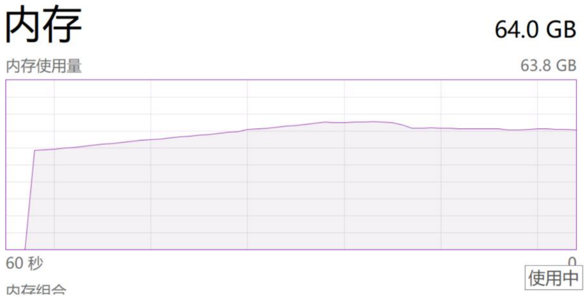

Figure 10. *Memory gets leaked when loading VGG features*

5. (**potential improvement**) We can see here our patchify technique is really simple(Just split into 9 equal size pieces), but we could apply the overlapping skills in order to gain more details in the image. We could also check for each piece; for those pieces that are not representative and could easily be misclassified, we could score them and discard them if the score is low.

3.4.3. *Reflection.*

In this method, we could see that the most helpful technique is **patchify**, as it adds more attention to different parts of the images. That means the recognition of the style of the image highly relies on the detailed information of the images. By rescaling each piece of the image and looking into the detail of each piece, the result is already decent, but I think if we could apply self-attention to also learn the global information of the information, we could complete more difficult classification jobs.

The poor performance of the fine-tuning model Siamese network is what I did not expect; I think maybe removing those parts of images that are not representative should be helpful in improving the performance of this model.

4. Test

4.1. Test on reference dataset

The first trial for our test is on the reference dataset, which is provided by this assignment. The dataset only has two different kinds of images, which are real and Gen-NoStyle. As the images were taken from the PolyU campus at different times, I think the testing result will be comprehensive. And because the result of **VGG feat + Siamese** is already poor during our previous validation process, we won't test this model on the real campus scenario. The following table shows the results we obtained from different models, and we can observe the result for different models for **Task 1** and **Task 2** in **Table 5**.

Model/Classifier	Accuracy
Basic CNN model (task 1)	51.51%
VGG + 3 linear layer (task 1)	66.66%
VGG + RFC (task 1)	70.01%
patchify + VGG + RFC (task 1)	87.8%
patchify + VGG + SVM (task 1)	72.72%
Basic CNN model (task2)	30.3%
VGG + 3 linear layer (task2)	60.06%
VGG + RFC (task 2)	57.57%
patchify + VGG + RFC (task 2)	81.81%
patchify + VGG + SVM (task 2)	63.63%

Table 5. *Testing result of reference dataset*

We can see that the decent result was still achieved by patchify method, though **SVM** achieved the best accuracy in our validation set: RFC works perfectly here. While styles of the images are more likely to be misclassified, I think the result could be acceptable as it is more than 80%. By making a comparison between different results, we can clearly observe the feature extraction power of the **VGG network**, and also notify that patchify is a useful technique to enhance the prediction result. As we can see by applying these methods, our prediction accuracy keeps rising.

4.2. Try to improvement for task1

4.2.1. *Trail 1: Enhance dataset.*

I have collected the images that are misclassified and then tried to find the similarities between them, as we can see from **Figure 11** (Misclassified by SVM model). The majority of the misclassified images are from the inner building; that is because when I was collecting the dataset, I took images from outside of the building but didn't consider taking images from inner buildings. That might make our model lack robustness when doing classification tasks. So, collecting more images from the indoor area could be one aspect of improvement.

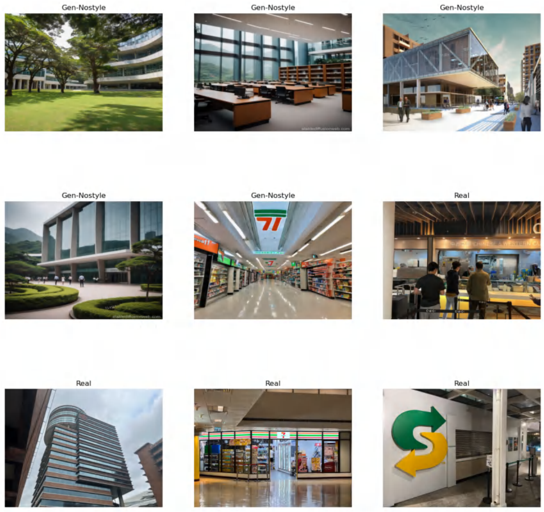

Figure 11. *Misclassified images*

As shown in **Figure 12**, I have collected 16 indoor images from different areas of the PolyU campus and added them to the training set (Choose areas which are not included in the testing dataset). Hopefully, by adding these pictures to let the model learn more knowledge from this aspect, the performance of our model can be improved.

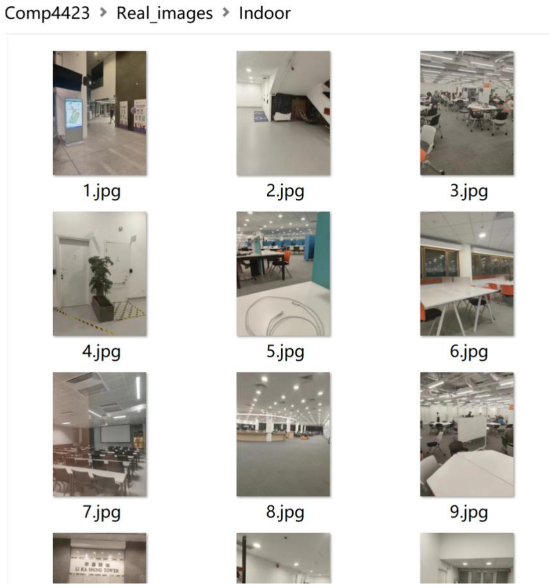

Figure 12. *Collected indoor images*

4.2.2. *Trail 2: K-fold validation*.

Maybe during the training process, we did not obtain a good split (Like if we are supper unlucky, we missed some important pictures in our training dataset, and the images fall into the validation dataset). So, in order to avoid such a thing, I will try to apply the K-fold validation to find the "Best" split (Select the fold which can obtain the highest validation score). We can observe from **Figure 13**, it shows the accuracy of the validation set when we are doing patch classification, and the result is quite fluctuant. As the 8^{th} fold attains the highest accuracy score, I am going to choose it to perform testing.

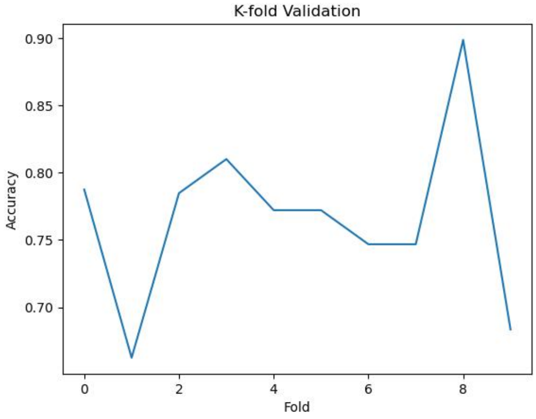

Figure 13. *Performance for different fold*

4.2.3. *Improved result*.

We can observe from **table 6** that the result is improving by taking these two strategies.

Model/Classifier	Accuracy
patchify + VGG + RFC (task 1)	90.9%
patchify + VGG + SVM (task 1)	87.8%
patchify + VGG + RFC (task 2)	84.9%
patchify + VGG + SVM (task 2)	72.7%

Table 6. *Improved testing result of reference dataset*

5. General limitation

(1) Our model doesn't work well for patches of the image, which means that if the user only cuts a piece from that, which means that the model is not good enough when the user input is small and doesn't include a lot of details. I think the model should be improved from this perspective in the future.

(2) We can also observe from the test part that the model somehow lacks flexibility, because we can see that the model cannot recognise those indoor images as AI-generated images if we haven't included indoor images in its training set.

(3) We may also need to include other different types of AI-generated images, in order to let the model work in more complex scenarios.

6. SUMMARY

In this essay, I tried different ways to distinguish AI-generated images and real-world images(**Task 1**). At the same time, try to tell the style of the generated images(**Task 2**). I explored a lot of methods, starting from a simple neural network, toward more complex models. Among all the methods I have attempted, I find that three things are pretty useful: **(1)**: to combining VGG-net with the simple traditional model. **(2)**: use patchify to enhance the prediction result. **(3)**: Use

K-fold to improve the testing result. And one thing failed, **(4)** : Though the model doesn't work well when combined with a Siamese network, I still learned a lot from the experience, as I tried a lot of voting methods to improve the result.

I believe the performance can be improved by applying patchify with overlapping and using the self-attention technique to enhance the relationship between different parts of the image.

References

1. Polyu official gallery:
   `https://www.polyu.edu.hk/en/photo-gallery/`
2. PolyU GenAI
   `https://genai.polyu.edu.hk/Text-To-Image`