



NUS
National University
of Singapore



Master of Technology in Intelligent Systems

ISY5002 Pattern Recognition Systems

Practice Module

Group Report

ISS Project – AI Image Detection

By Two.AI

Team Members

Wee De Li, Darren

Seow Teck Han, Michael

Sankalp

Fong Zhi En, Kelvin

Contents

Contents	2
1. Executive Summary	4
2. Introduction	5
Problem Statement	5
Proposed Solution	6
3. GitHub Location and Demo website.....	7
4. Video Presentation	7
5. System Design and Architecture	8
Use Case Diagram	8
Activity Diagram	9
System Architecture Diagram	10
Frontend	11
Cardinal Requirements.....	11
Assumptions made	11
Metrics	11
Proposed solution.....	11
Backend	12
Cardinal Requirements.....	12
Assumptions made	12
Metrics	12
Proposed solution.....	12
6. Data Analysis	13
Initial Dataset	13
Training the Model	14
1. SVM.....	14
2. Convolution Neural Network.....	18
3. Initial Testing	21
Trying with New Dataset	24
1. SVM.....	25
2. Convolution Neural Network.....	26
3. ResNet Transfer Learning	30
4. Ensemble	32
5. Evaluation	33
7. System Implementation and Setup.....	35
Installation Instructions	35

Troubleshooting	35
Frontend Implementation	36
Backend Implementation	38
8. Conclusion	39
Appendix.....	39
1. Project Proposal	39
2. References.....	42

1. Executive Summary

With the rapid advancement of generative Artificial Intelligence models like ChatGPT and Midjourney, the easy generation of synthetic data presents ethical issues which can deceive users with misleading and/or fake news. In particular, the creation of realistic synthetic images has become increasingly prevalent by spreading misinformation through the uploading of deepfakes. These images, generated by algorithms, can often be indistinguishable from photographs taken by physical cameras or created by human artists. This phenomenon has raised concerns regarding the potential for misinformation and manipulation in visual media.

The project seeks to address this issue by developing a machine learning model capable of accurately discerning between images generated by AI (Artificial Intelligence) algorithms and those of real images taken by a camera. This distinction is crucial for ensuring the authenticity and integrity of visual content in various domains, including journalism, forensics, and content moderation.

The objective of this project is to create an intuitive front-end user interface enabling users to upload images for classification. The interface will connect seamlessly with a backend machine learning model, trained on a comprehensive dataset, to accurately determine whether the uploaded image is artificially generated or a real image. This tool aims to provide a user-friendly solution for verifying the authenticity of visual content, addressing concerns related to synthetic imagery in the digital landscape.

2. Introduction

Problem Statement

With the rapid advancement of generative Artificial Intelligence models like ChatGPT, Midjourney and Stable Diffusion, the quick and easy generation of synthetic image data presents ethical issues which can deceive users with misleading and/or fake news.

These images, generated by algorithms, can often be indistinguishable from real photos, and have raised concerns regarding the potential for misinformation and manipulation in visual media.

There are currently real-life examples of these images going undetected, until someone confessed to using. An example of this is the AI-generated image titled “The Electrician” that won first in the creative category in the open competition of Sony World Photography Awards 2023. The German photographer, Boris Eldagsen, refused to accept the prize for the competition, admitting that his work was AI-generated and claiming that it was just a test to see whether photography competitions were prepared to handle AI-generated image.

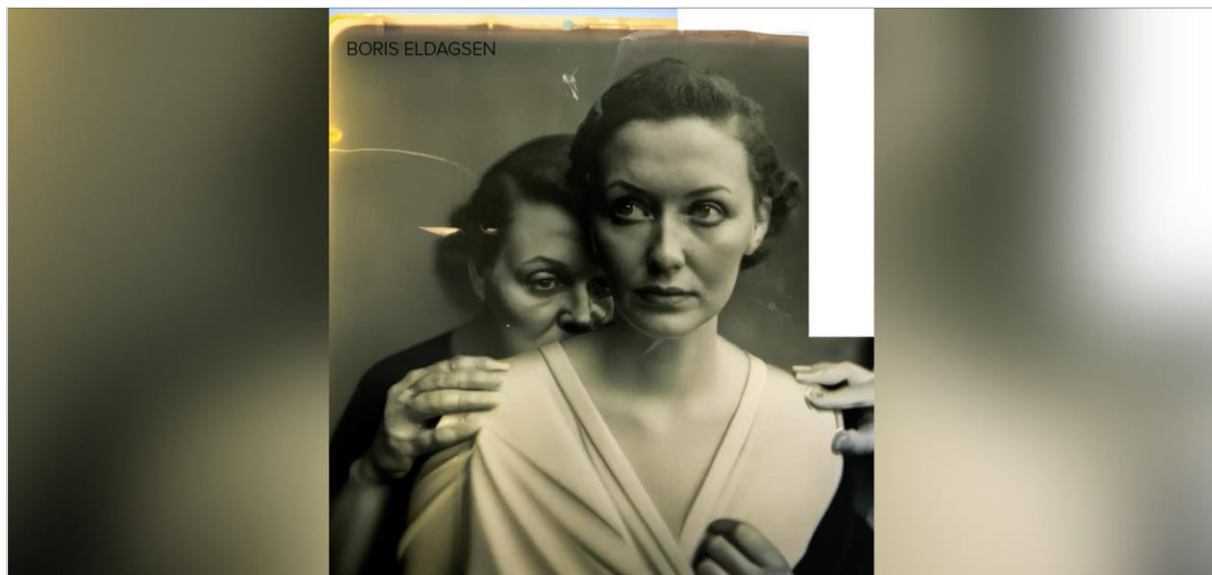


Figure 1 “The Electrician” AI-generated image wins at Sony World Photography Awards; Photo: Boris Eldagsen

These AI-Generated images, when shared without context, can also inadvertently influence political views and opinion,



Figure 2 Political Image Example

Proposed Solution

The proposed solution is to develop a machine learning model capable of discerning between images generated by AI algorithms, and those taken by a real camera.

An intuitive front-end user interface will be developed to allow users to upload images for classification.

One key point to take note of is that the system is designed to detect whether an image is AI-generated, and not whether an image has been digitally altered or photoshopped.

3. GitHub Location and Demo website

The GitHub repository can be found at:

<https://github.com/sink257/PRS-PM-2023-ISK5002>

The demo website for the system can be found at:

<http://54.79.114.148/>

4. Video Presentation

The video presentation for the system can be found at:

<https://youtu.be/UKxGmrfKkME>



5. System Design and Architecture

Use Case Diagram

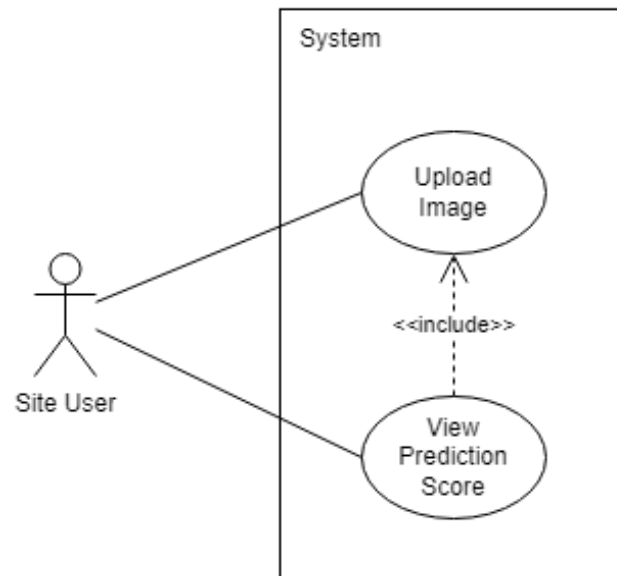


Figure 3: Use Case Diagram

The use case diagram shows the 2 use cases for the site, where the user will need to be able to:

1. Upload an image
2. View the predicted score

Activity Diagram

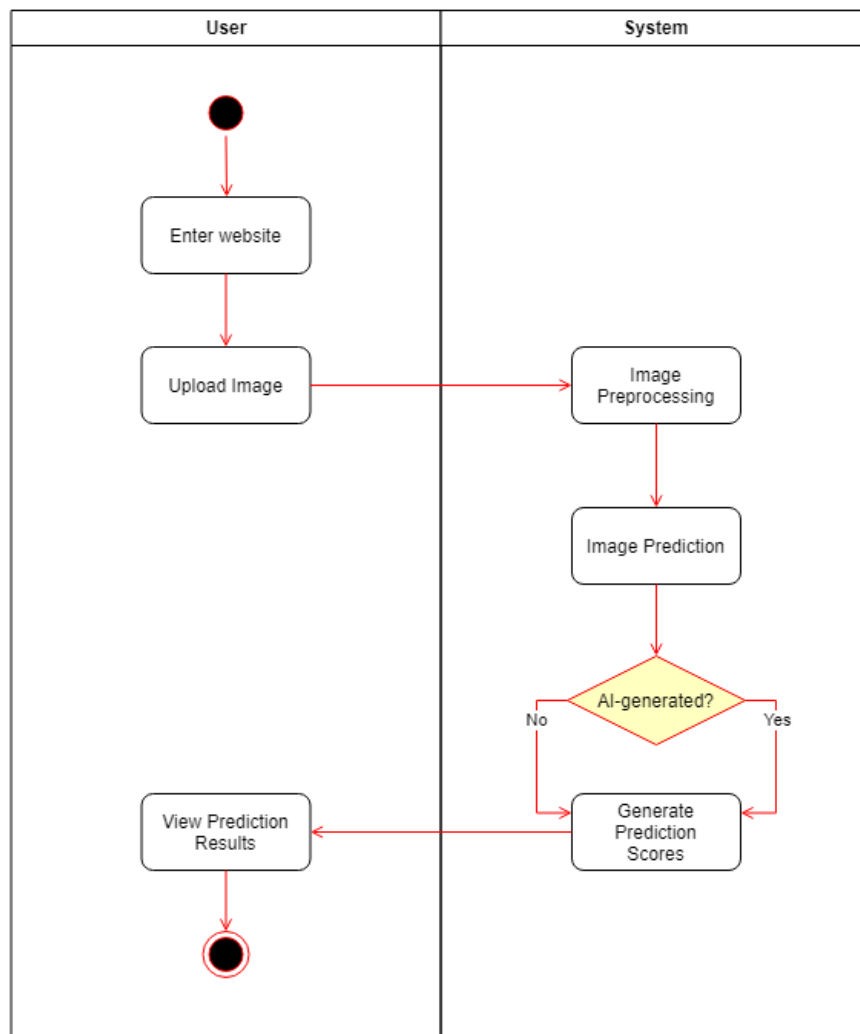


Figure 4: Activity Diagram

The Activity diagram shows the expected flow of activities in the system, from the user entering the website, to uploading an image, to the image being processed, to the AI predicting if the image is AI generated and returning with the result.

System Architecture Diagram

The system will be made up of a frontend and a backend service, with each service being hosted in a container. They will allow them to be easily hosted in any container management services, such as Docker.

The architecture of the system can be seen in the diagram below.

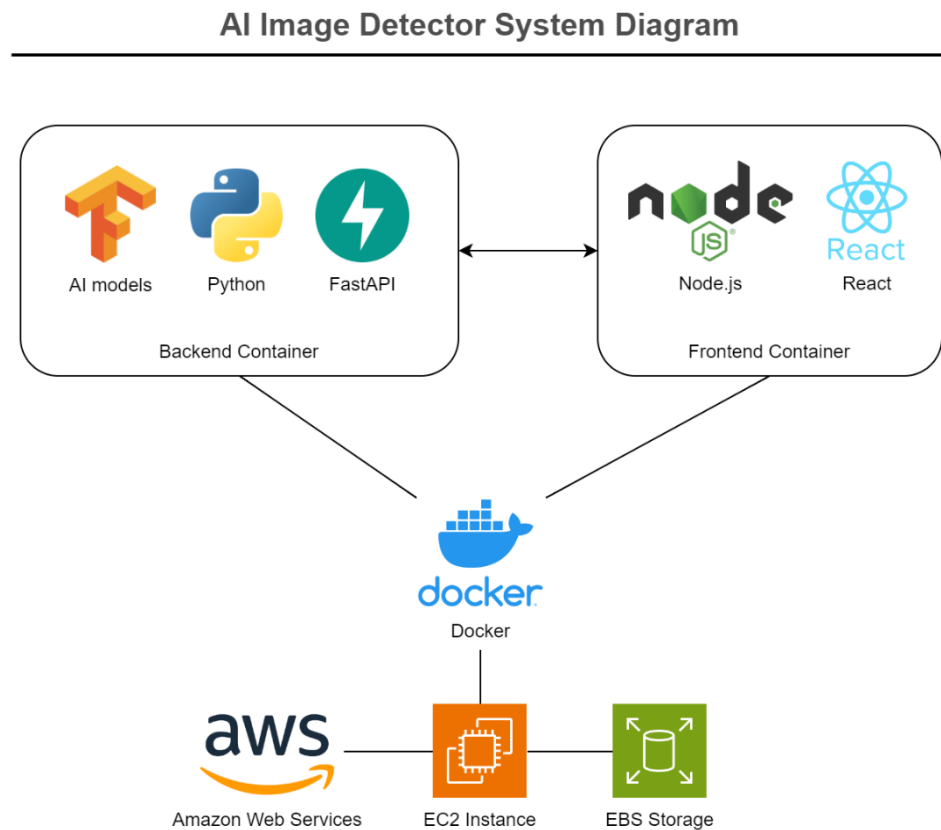


Figure 5: System Architecture

The system is hosted on Amazon Web Services, comprising of one EC2 instance running Amazon Linux OS with an attached EBS storage volume. Docker is installed in the EC2 instance and there are 2 docker containers, one for the backend and one for the frontend. The backend container is running Python and handles the image processing, AI models and inferencing, and application API. The frontend container is running Node.js, hosting a React based frontend, and it connects to the backend via API call.

Frontend

Cardinal Requirements

The solution should provide an easy-to-use frontend for users to upload an image, on either mobile or desktop.

Assumptions made

Users are accessing from either a mobile device or a desktop computer

Metrics

The following [metrics](#) are commonly used for UX design

- Time-on-task
This KPI describes the time (in minutes and seconds) that a user needs to complete a task successfully.
 $\text{Time on Task} = \text{Time taken to access the website} + \text{Time taken upload an image and get back the results.}$
- User Error Rate
This refers to the number of times a user makes a wrong entry.

We will only use the time-on-task as a metric to decide the front end to be used.

Apart from quantifying metrics solely on the solution, we should also consider metrics of the platform used.

- Commonly used platforms
This is a crucial factor because human machine interface has strong reliance on the platform they used. For example, a teenager would typically use mobile phone more than an adult. Our system will cater to both mobile and web-based users.

Proposed solution

The frontend will be developed using the React framework. Part of the reason React was chosen, was because of the readily available number of templates to use, that will help with rendering the website for both mobile and desktop devices, while appearing cohesive.

The frontend will communicate with the backend with a REST API call, using base64 encoding and JSON to send the image information over.

Backend

Cardinal Requirements

The solution should provide a simple query API to

- Listen for incoming queries from users.
- Retrieves responses from Image Prediction System.
- Provide a response to front-end.

Assumptions made

This backend will not have hundreds or thousands of requests per second. The metric used should just be a simple latency.

Metrics

The following metrics are commonly used for accessing web endpoints

- Average and Max Latency
Round trip time from accessing a web query to a web response.

Apart from latency, there are some other qualifying conditions to access the use of the endpoint.

- Ease of maintenance.

Proposed solution

The backend will be developed using FastAPI, as it is fast and easy to use, with the trained model easily integrated into it.

The backend will get a REST API call from the frontend, run the image information through the trained model, and return the result containing whether the image is AI generated, together with a probability percentage score, for the user to see.

6. Data Analysis

Initial Dataset

The initial dataset is sourced from [Kaggle](#), which contains 60,000 synthetically generated images using the stable-diffusion model, and 60,000 real images from the CIFAR-10 dataset.

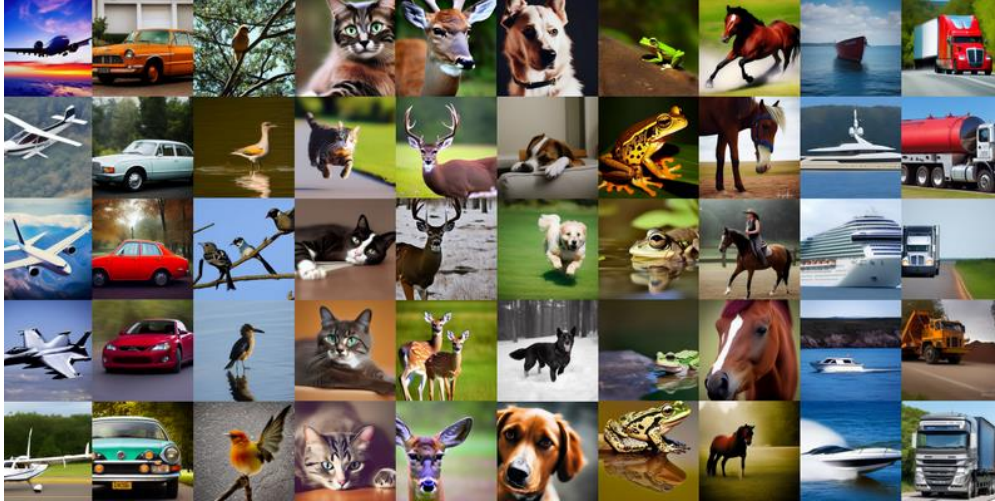


Figure 6: Image Samples from Dataset

The dataset was separated into a training dataset of 100,000 images and a testing dataset of 20,000 images, with equal class.

Training the Model

1. SVM

Data ingestion

```
SVM.ipynb

from PIL import Image
import numpy as np
import os


def load_images_from_folder(folder):
    images = []
    labels = []
    for subfolder in os.listdir(folder):
        subfolder_path = os.path.join(folder, subfolder)
        if os.path.isdir(subfolder_path):
            label = 1 if subfolder == 'REAL' else 0
            for filename in os.listdir(subfolder_path):
                img_path = os.path.join(subfolder_path, filename)
                print(filename)
                img = Image.open(img_path)
                img = np.array(img) / 255.0 # Normalize pixel values to [0, 1]
                images.append(img)
                labels.append(label)
    return np.array(images), np.array(labels)

# Load training and testing data
train_images, train_labels = load_images_from_folder('train')
test_images, test_labels = load_images_from_folder('test')

# Flatten images
train_features = train_images.reshape(len(train_images), -1)
test_features = test_images.reshape(len(test_images), -1)
```

We load all the images using the ‘pillow’ package, converting them into a NumPy array. Since the images were all in 32 by 32 pixels, there was no need to standardize and resize the images. In addition, we normalize the pixel values from 0 to 1, which would help in our model training, making sure that no certain features will be prioritized. After which, we flatten the NumPy arrays so that they will be in a 1D array.

Training the model with GridSearchCV



```
SVM.ipynb

from sklearn import svm
from sklearn.model_selection import GridSearchCV
import pickle

# Defining the parameters grid for GridSearchCV
param_grid={'C':[0.1,1,10,100],
            'gamma':[0.0001,0.001,0.1,1],
            'kernel':['rbf','poly']}

# Creating a support vector classifier
svc=svm.SVC(verbose=True, probability= True)

# Creating a model using GridSearchCV with the parameters grid
classifier=GridSearchCV(svc,param_grid, verbose = True)

# Train the Classifier
classifier.fit(train_features, train_labels)
classifier.fit(subset_train_features, subset_train_labels)

# Save the classifier to a file
with open('svm_classifier_updated_gridsearch.pkl', 'wb') as f:
    pickle.dump(classifier, f)
```

We use GridSearchCV to do the training of the dataset, helping in the automation of the hyperparameter tuning, instead of manually trying out different combinations and evaluating them. In addition, it performs cross validation during the search process, splitting the data into multiple subsets and evaluating the model on each combination. This will also help us to prevent overfitting on the image's features, since the model is evaluated on the validation set, rather than on a single training dataset.

Issues that we faced

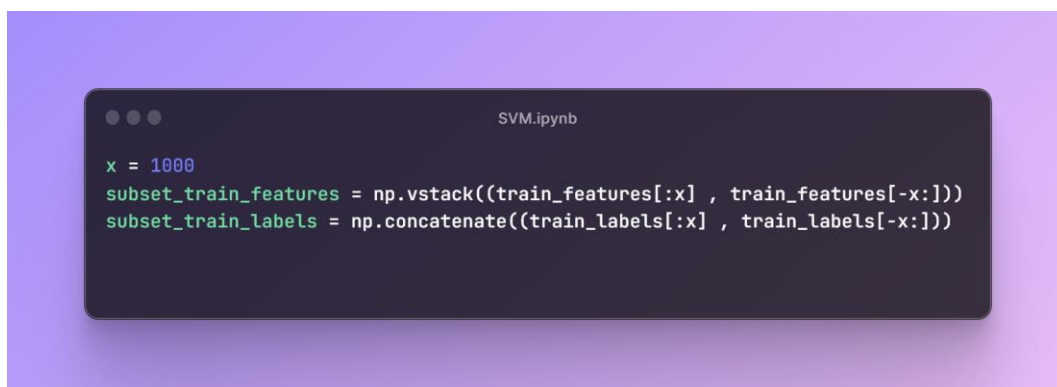
After running the code above, the training of the model was not done even after more than 24 hours. We concluded to **stop the training**, since

1. Time-Consuming: Training an SVM on a large dataset is computationally expensive and time-consuming. With 100,000 images, the training process could take a substantial amount of time, potentially weeks or even longer, depending on the complexity of the model and available computing resources.
2. Grid Search Complexity: Hyperparameter tuning, as shown in our code, becomes even more complex with a larger dataset. The search space for hyperparameters increases, which can lead to longer computation times.
3. Constraints of training on CPU with sklearn: Since sklearn do not support training on the GPU, CPUs are slower for parallelizable tasks like matrix operations, which are crucial in SVM training.

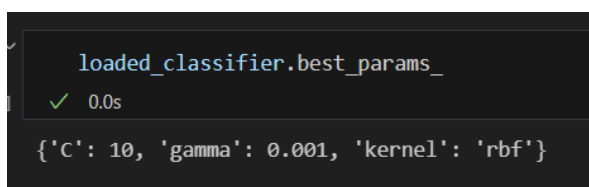
Workaround

After evaluation and research, to get a SVM model out for our project, we decided to use a subset of the training data (2000 images), a practical approach to address the computational challenges, hoping that training will be significantly faster.

However, we do note and recognize the limitations of doing such, with risks of the model overfitting and not capturing the underlying patterns of the data.



We take 1000 images from both classes (REAL & FAKE) and rerun our GridsearchCV. This ran for 2 hours with the following parameters:



Evaluation

Using the loaded classifier that we have trained; we assess and evaluate the performance of the model:

	precision	recall	f1-score	support
0	0.75	0.72	0.73	10000
1	0.73	0.76	0.74	10000
accuracy			0.74	20000
macro avg	0.74	0.74	0.74	20000
weighted avg	0.74	0.74	0.74	20000

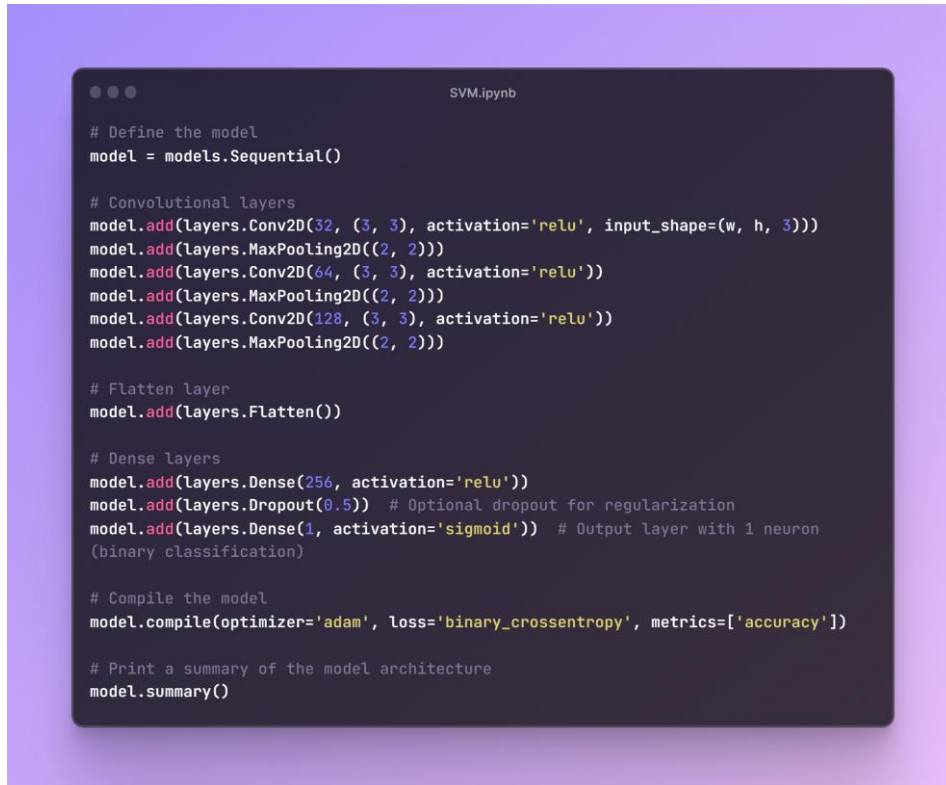
	Predicted Fake	Predicted Real
Actual Fake	7183	2817
Actual Real	2447	7553

The baseline model managed to achieve an accuracy rate of **73.68%** on the test dataset, which is decent for the small subset training dataset that we had.

Considering our use case of identifying real vs AI-generated images, where it is more important to avoid misclassifying fake images as real (minimize false negatives), it is crucial to pay attention to the recall for class 0. In this case, the recall for fake images (class 0) is 72%, meaning that the model correctly identifies 72% of the actual fake images.

2. Convolution Neural Network

In addition to using SVM to train our model, we also used a simple Convolutional Neural Network (CNN), well-suited for image classification tasks, and much suitable for our large dataset which we can use GPUs.

A screenshot of a Jupyter Notebook window titled "SVM.ipynb" with a dark background. The code is written in Python and defines a CNN model using Keras. It starts with a Sequential model, followed by three convolutional blocks. Each block consists of a Conv2D layer and a MaxPooling2D layer. The first block has a Conv2D layer with 32 filters and a MaxPooling2D layer with a pool size of (2, 2). The second block has a Conv2D layer with 64 filters and a MaxPooling2D layer with a pool size of (2, 2). The third block has a Conv2D layer with 128 filters and a MaxPooling2D layer with a pool size of (2, 2). After the convolutional blocks, there is a Flatten layer, followed by three dense layers: a Dense layer with 256 units, a Dropout layer with a rate of 0.5, and a final Dense layer with 1 unit using a sigmoid activation. The model is compiled with the Adam optimizer, binary crossentropy loss, and accuracy metric. Finally, the model's architecture is summarized.

```
# Define the model
model = models.Sequential()

# Convolutional layers
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(w, h, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Flatten layer
model.add(layers.Flatten())

# Dense layers
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.5)) # Optional dropout for regularization
model.add(layers.Dense(1, activation='sigmoid')) # Output layer with 1 neuron
(binary classification)

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Print a summary of the model architecture
model.summary()
```

The provided images were three channels representing the Red, Green, and Blue (RGB) components, sized 32 pixels by 32 pixels.

Subsequently, six layers were employed, organized into three sets of Convolution Blocks. Each block consisted of a Convolution layer followed by a Max Pooling Layer. These operations served to extract relevant features and down-sample the feature maps, thereby reducing computational demands.

Following the Convolution Blocks, a Flatten layer was applied to transform the 3D feature maps into a 1D representation. The final three layers comprised of the Dense Layers, with an incorporated Dropout layer for regularization purposes, mitigating the risk of overfitting.

Finally, the network's output layer produced a single value indicating the prediction of whether the given image is generated by artificial intelligence.

The CNN Model used for training can be seen below:

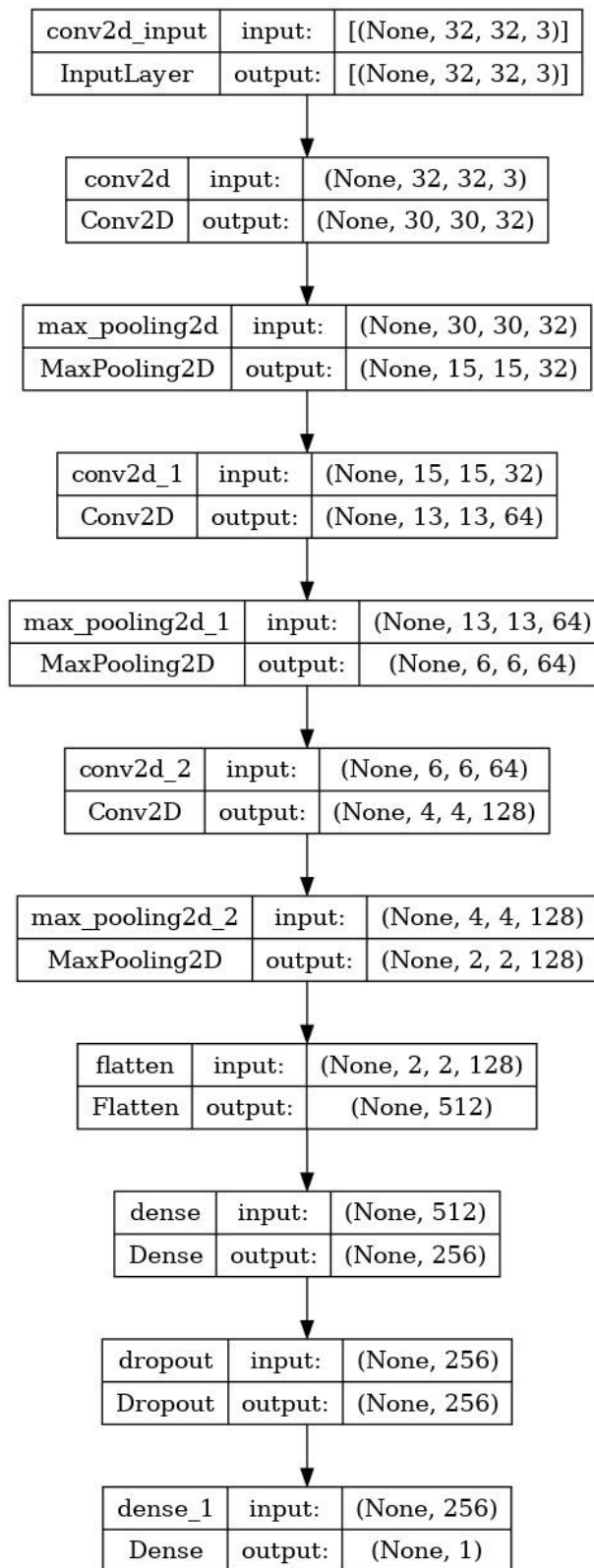


Figure 7: CNN Model

Evaluation

	precision	recall	f1-score	support
0	0.92	0.94	0.93	10000
1	0.94	0.92	0.93	10000
accuracy			0.93	20000
macro avg	0.93	0.93	0.93	20000
weighted avg	0.93	0.93	0.93	20000

	Predicted Fake	Predicted Real
Actual Fake	9423	577
Actual Real	845	9155

The CNN model has demonstrated remarkable performance, achieving an outstanding accuracy rate of 92.89% on the test dataset. This level of accuracy is a strong indicator of the model's ability to correctly classify images into their respective categories.

Furthermore, the emphasis on class 0 recall, which is crucial for minimizing false negatives and ensuring that fake images are not misclassified as real, yields a highly favourable result of 94%. This means that the model successfully identifies 94% of actual fake images, which is a critical aspect of the model's effectiveness.

Overall, the model's exceptional accuracy and high recall for class 0 underscore its proficiency in distinguishing between real and AI-generated images. These results suggest that the CNN model is not only well-constructed but also highly effective in its intended task. This level of performance instils confidence in the model's reliability and suitability for practical applications in identifying real and AI-generated images when tested against our test set. However, this was not the case when doing testing on real life images ...

3. Initial Testing

With the 2 trained models, one from SVM, another from CNN, we started testing the application with new images that we found outside of the original dataset and were not satisfied with the performance. An example of why is shown in the following picture:



CNN Prediction: Image is Real with probability score 99.11%

SVM Prediction: Image is Real with probability score 89.93%

Figure 8: An Obviously Fake Image

An image that is AI-generated getting recognised as real is not a good result. We tried a few more other photos.



CNN Prediction: Image is AI-Generated with probability score 67.17%

SVM Prediction: Image is Real with probability score 84.41%

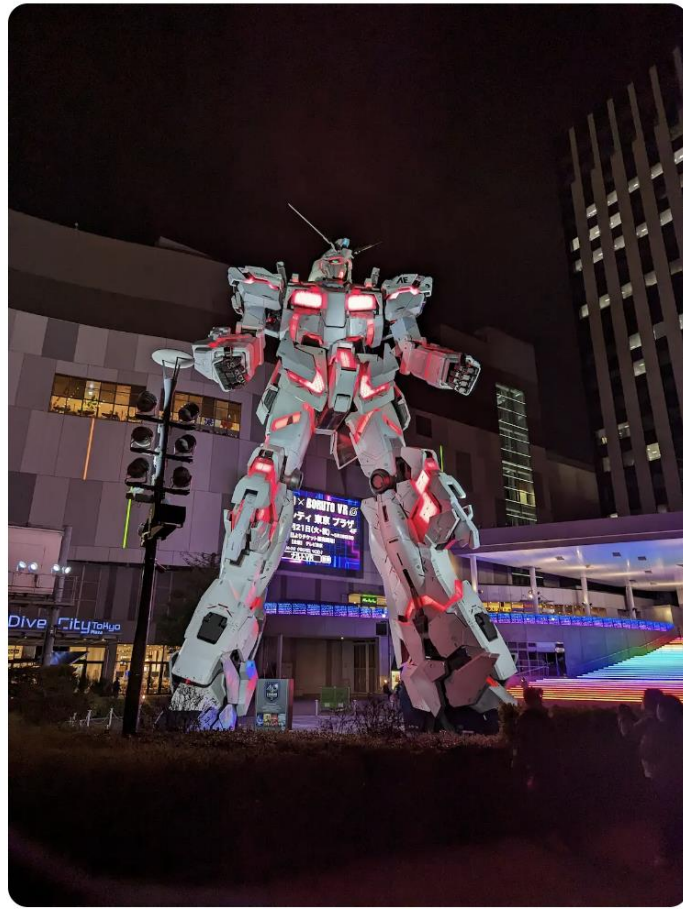
Figure 9: Milo Lover Generated Image



CNN Prediction: Image is Real with probability score 96.75%

SVM Prediction: Image is AI-Generated with probability score 79.45%

Figure 10: St Peter's Basilica



CNN Prediction: Image is AI-Generated with probability score 85.95%
SVM Prediction: Image is AI-Generated with probability score 58.28%

Figure 11: Unicorn Gundam Statue from Japan

Considering the diverse outcomes observed, there is a suspicion of potential overfitting in both trained models. Additionally, we suspect that the significant downsizing of images for processing to a 32 x 32 resolution may have led to a loss of crucial details, potentially contributing to the inaccuracies in the results. Notably, achieving a higher probability of an image being classified as AI-generated was straightforward, particularly when utilizing images that may appear unrealistic at first glance but are, in fact, real. For instance, consider Figure 10 depicting St. Peter's Basilica, which, due to its grandeur, can give the initial impression of being generated by AI. Similarly, Figure 11 portrays a scale model statue, which both models seem to identify as AI-generated. This highlights the nuanced challenges associated with accurately discerning real from AI-generated images.

Furthermore, we posit that the choice of dataset, specifically CIFAR10 for real images, may not have been comprehensive enough to adequately represent the diverse features found in real-world images. The CIFAR10 dataset primarily focuses on a few categories of animals and vehicles, which may not encompass the wide array of characteristics present in everyday scenes and objects. This potential limitation could have impacted the model's ability to discern real images accurately.

As a result, we have opted to search for a more comprehensive dataset that give better real-world results than the current one.

Trying with New Dataset

Later, we found another dataset on [Kaggle](#) that could also be used to differentiate between real and AI Generated images. This was a bigger dataset, with 2,496,738 total images, consisting of 964,989 real images, and 1,531,749 fake images.

The 1,531,749 fakes images were generated with 25 different generators, which includes 13 GANs, 7 Diffusion, and 5 miscellaneous generators. In addition, the real images here consisted of various categories of images, from landscape, animals to faces of celebrities. This seems like a more prominent dataset to help us tackle the issues we faced.

The training images are also bigger than the previous dataset, 200 x 200 instead of 32 x 32, which may help address the loss in detail from downscaling the images previously.

**Note: after downloading the dataset from Kaggle and splitting the images into train and test set: we had 882,752 images for training and 220,694 images for testing.*

The breakdown is as follows for the training dataset:

Fake images: 363,839

Real images: 518,913

which is an imbalanced dataset.

We then did the same that we had run for our initial dataset: SVM and CNN

In addition, we also tested Transfer learning techniques with the ResNet model.

1. SVM

Utilising the same procedure of taking 1000 images randomly from each class from the training dataset, and going through GridSearchCV, the model finished training after 10 hours (in particular, the training takes much longer here because the dimension of the image is now 200x200x3 as compared to 32x32x3).

Evaluation

	precision	recall	f1-score	support
0	0.48	0.68	0.56	90962
1	0.68	0.48	0.56	129732
accuracy			0.56	220694
macro avg	0.58	0.58	0.56	220694
weighted avg	0.60	0.56	0.56	220694

	Predicted Fake	Predicted Real
Actual Fake	61930	29032
Actual Real	67766	61966

The SVM model has demonstrated poor performance, achieving only an accuracy rate of 56% on the test dataset. In addition, the precision of fake images is below 50%. In light of the poor performance, we decided not to continue evaluating different SVM parameters due to the below reasons:

- 1) Due to computational efficiency, it is impossible to train an SVM model on 800k+ images. Utilising 2000 images was not giving up to standard performance on our test dataset, unlike in our previous CIFAKE dataset.
- 2) In addition, our local machines were crashing when trying to load in the 200k+ test images as a numpy array to do our evaluation due to memory. As such, to evaluate, we had to do a for loop on each image and use the classifier to predict one by one. This took 18 hours to evaluate our SVM model, which was not sustainable.

2. Convolution Neural Network

The CNN was retrained again with the new data set, using the same process used above, updating the model to take in take 200x200x3 images instead:

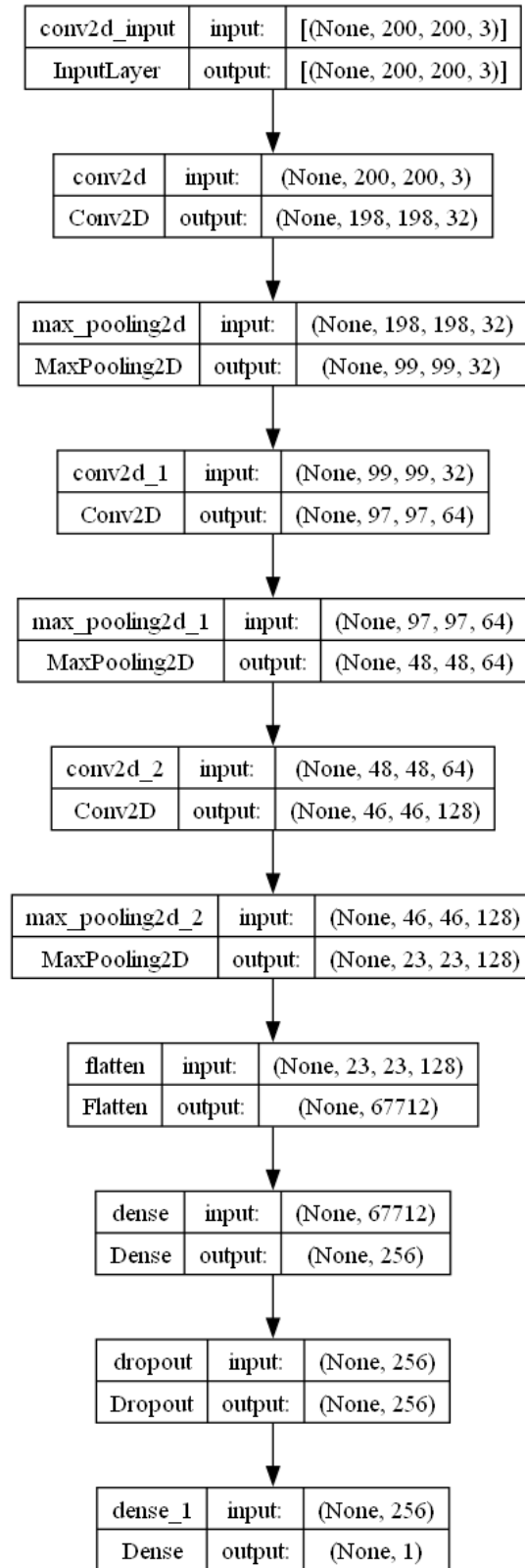


Figure 12: CNN Model

Additional methods compared to initial CNN:

1) Imbalanced classes

It is crucial to address the issue of imbalanced classes within the training dataset. It is imperative that we implement a methodology to prevent our model from exhibiting bias towards the predominant class, which, in this context, pertains to the REAL images. This will help to ensure a more equitable and accurate performance across all classes.

To do so, we implement class weights when training the model. The `weight_for_0` and `weight_for_1` variable is then computed. These weights are inversely proportional to the number of instances in each class. They represent the importance assigned to each class during training. In our case, the fake image class with fewer instances will have higher weights to compensate for their scarcity.

```
fake_number = 363839
real_number = 518913
total = fake_number + real_number

weight_for_0 = (1 / fake_number) * (total / 2.0)
weight_for_1 = (1 / real_number) * (total / 2.0)

class_weight = {0: weight_for_0, 1: weight_for_1}

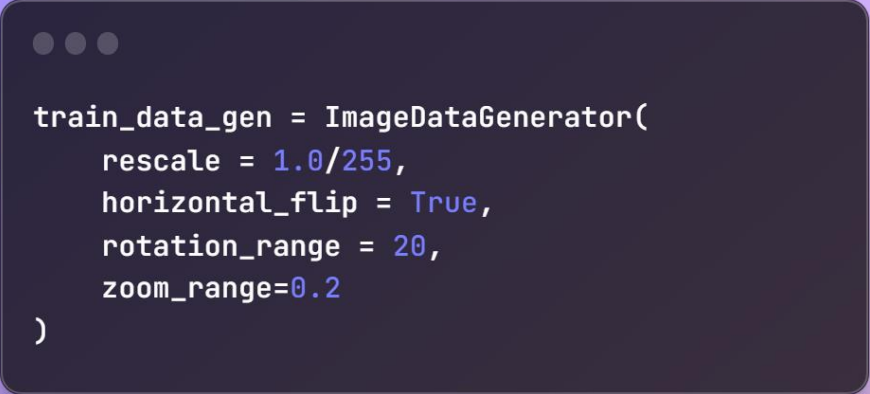
# Train the model
history = model.fit(train_Dataset,
                    epochs=10,
                    class_weight=class_weight)
```

2) Data Augmentation

In the existing configuration, the neural network is exposed to the identical set of images during each epoch of the training process. However, it is imperative to introduce diversity to compel the network to focus on learning the crucial features for classification accurately.

To address we introduced image augmentation for the training of the model. This method involves the creation of randomly altered versions of images at the outset of every epoch. As a result, the network is presented with a varied set of images, ensuring that it does not encounter the exact same images twice. This approach enhances the network's ability to generalize and learn robust features essential for accurate classification.

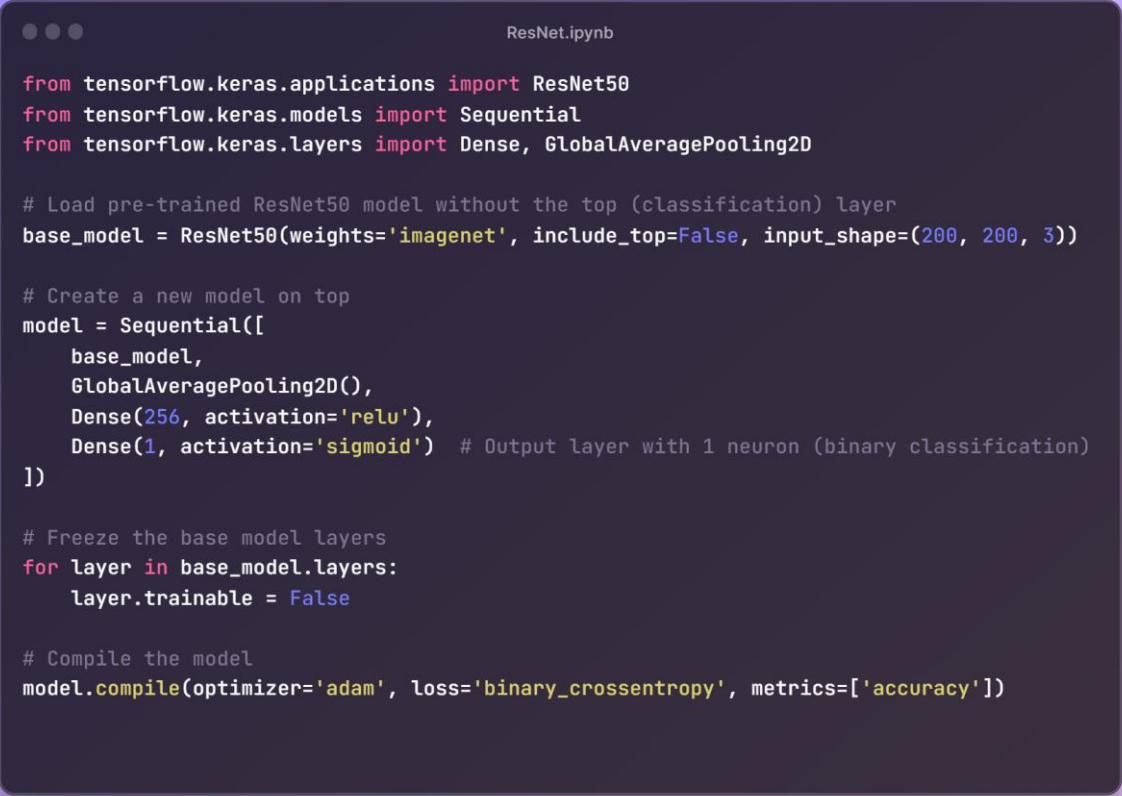
This is done through when declaring our image generator:



```
train_data_gen = ImageDataGenerator(  
    rescale = 1.0/255,  
    horizontal_flip = True,  
    rotation_range = 20,  
    zoom_range=0.2  
)
```

3. ResNet Transfer Learning

We also attempted using the ResNet50 model to see how it would compare to both SVM and CNN that we did.



```
ResNet.ipynb

from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

# Load pre-trained ResNet50 model without the top (classification) layer
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(200, 200, 3))

# Create a new model on top
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(256, activation='relu'),
    Dense(1, activation='sigmoid') # Output layer with 1 neuron (binary classification)
])

# Freeze the base model layers
for layer in base_model.layers:
    layer.trainable = False

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Using the ResNet50 model, we freeze all weights in its layers, allowing the model to retain the features learned from the original ImageNet dataset. After the ResNet layers, we use a global average pooling which is said to be effective in reducing the dimensionality of the feature maps while retaining the most information, leading to reduced complexity and better generalization. We then declare our output layer, which is a dense layer of 1 neuron since we are doing a binary classification.

The final model used is shown below:

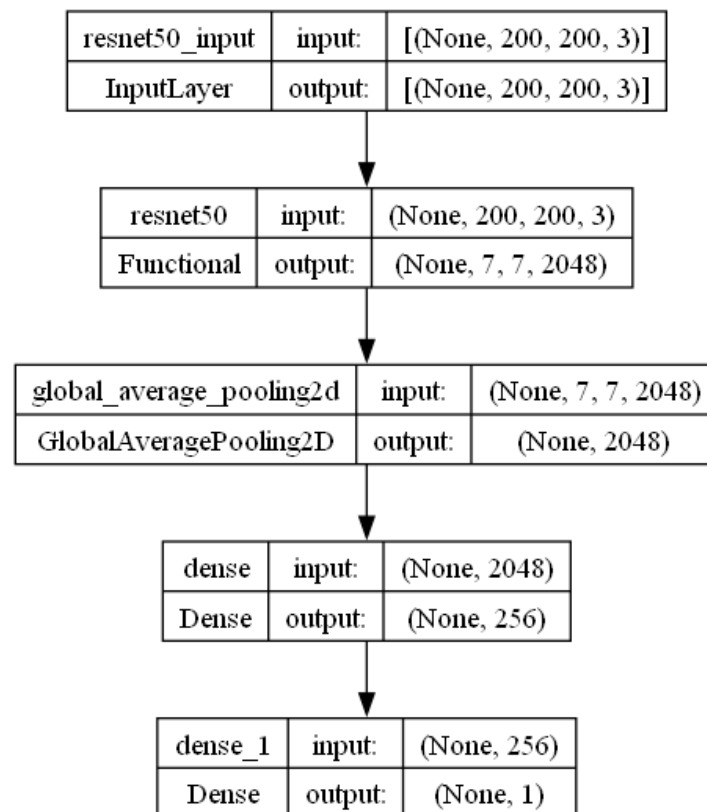


Figure 13: ResNet Model

Additional methods that we tried:

Data Augmentation

Same as the CNN model, we did image augmentation on our training dataset images to introduce diversity to the model.

Unfreezing last 5 layers of ResNet50

```

ResNet.ipynb

for layer in base_model.layers[:-5]:
    layer.trainable = False
  
```

The later layers in a neural network often learn more abstract and task-specific features. By unfreezing these layers, we allow the model to adapt and refine these features to better suit the specifics of our classification task. The hope is that the model can learn more intricate and specialized patterns specific to our training dataset.

4. Ensemble

Given that we have trained a total of 3 models (SVM, CNN & ResNet transfer learning), we initially decided to use ensemble method of Majority Voting, which might help us to yield more accurate and robust predictions compared to an individual model. Combining predictions from multiple models helps balance out any biases present in individual models, leading to a more reliable final prediction.



CNN Prediction: Image is AI-Generated with probability score 63.05%
ResNet Prediction: Image is AI-Generated with probability score 56.42%
SVM Prediction: Image is AI-Generated with probability score 69.50%

Final Prediction: Image is AI-Generated

Figure 14: A test image showing correct predictions.

However, after the evaluation of our SVM model, which took 18 hours to finish, the performance of the SVM model on new test images was sub-par. As such, we evaluated that including the SVM model in our majority voting might cause our final predictions to be inaccurate.

We decided to only pick one model which gave the best prediction results between our trained CNN & ResNET models, with the evaluation below:

5. Evaluation

For the new dataset, we ran the model training 10 times, with each run taking between 6 hours and 22 hours. The training was done on a Nvidia 4080 GPU locally to speed up the training times.

Run No.	Model Type	Image Size	Epochs	Data Augmentation	Data Balancing	Test Accuracy
1	CNN	32x32	10	No	No	66.68%
2	CNN	200x200	10	No	No	75.10%
3	ResNet	200x200	10	No	No	66.20%
4	CNN	200x200	20	Yes	No	64.10%
5	CNN	200x200	20	No	No	74.77%
6	ResNet Unfrozen	200x200	20	No	No	68.07%
7	CNN	200x200	10	No	Yes	66.10%
8	ResNet Unfrozen	200x200	10	No	Yes	63.10%
9	ResNet Unfrozen	200x200	20	Yes	Yes	64.71%
10	CNN	200x200	20	Yes	Yes	65.58%

In the table above, we can see that some models, such as the standard CNN model in run #2 and run #5 have a high accuracy of around 75% whereas the rest of the accuracies tend to hover around the 65% mark. However, we found that this accuracy did not give expected results in the real world while using our application.

After running the training, we tested the model with a varied number of real-world images, both real and AI generated. This is because even though some models trained with higher test accuracy than others, they did not do as well in real world scenarios due to possible overfitting.

Based on this evaluation, we decided to pick run #9, which is the ResNet model with last five layers unfrozen, trained on 20 epochs and having had Data Augmentation and Data Balancing performed on it.

This particular model was chosen, as one reason we found was that images from newer AI generators which are not part of the training data such as Dalle-3 tended to not work as well with other models. However, our chosen model was generalised enough to return a respectable response when being tested with images outside of its dataset.

The confusion matrix for the chosen model was as follows:

	precision	recall	f1-score	support
0	0.57	0.61	0.59	90962
1	0.71	0.67	0.69	129732
accuracy			0.65	220694
macro avg	0.64	0.64	0.64	220694
weighted avg	0.65	0.65	0.65	220694

	Predicted Fake	Predicted Real
Actual Fake	55852	35110
Actual Real	42750	86982



Prediction: Image is AI-Generated with probability score 53.40%

Figure 115: A sample AI generated image result from a new image generator.

7. System Implementation and Setup

Installation Instructions

System Requirements

- 1) At least 4GB of RAM
- 2) Internet connection
- 3) Docker and Docker Compose to be installed on the system

Installation Procedure

- 1) Install Docker and Docker Compose, if not yet installed.
- 2) git clone the git repository (<https://github.com/sink257/PRS-PM-2023-ISY5002.git>) to a folder.

- 3) Navigate into the SystemCode folder, you should see the following directory structure.

```
[root@ip-172-31-23-18 SystemCode]# ls
00_buildFE.sh  01_buildBE.sh  Dockerfile_BE  Dockerfile_FE  backend  compose.yaml  frontend
```

- 4) `chmod +x 00_buildFE.sh` and `chmod +x 01_buildBE.sh` to allow execution permissions of the build scripts.
- 5) `./00_buildFE.sh` to build the frontend image and wait until it completes.
- 6) `./01_buildBE.sh` to build the backend image and wait until it completes.
- 7) `docker compose up -d` to start both the backend and frontend containers.

After running the above steps, the website should be able to be viewed by accessing the browser and keying in the following address: `http://<IP address>/`

To stop the containers, navigate to the SystemCode folder and run the command:
`docker compose down`

Note: The frontend website is hosted on port 80 and the backend API is hosted on port 8000.

Troubleshooting

Issue: After uploading of image and pressing continue, the prediction result does not show but just stays stuck with a black loading bar.

Possible Cause: The backend API is not responding to the request, either due to a crash or some other error.

Resolution: Check that the backend container is still running and not exited, by typing `docker ps`. If needed, restart the container by navigating to SystemCode folder and typing `docker compose up -d`.

Issue: Image upload fails with a message saying the website has exceeded its free Bytescale limit.

Possible Cause: Bytescale image upload service has hit the free API usage limit.

Resolution: Create a Bytescale account and attach an API key by saving the following line `NEXT_PUBLIC_UPLOAD_API_KEY=<API_KEY>` into a file named `.env` and place it into the frontend folder.

Frontend Implementation

The front end was developed using a React template that fit our use case, with an image upload portion built in.

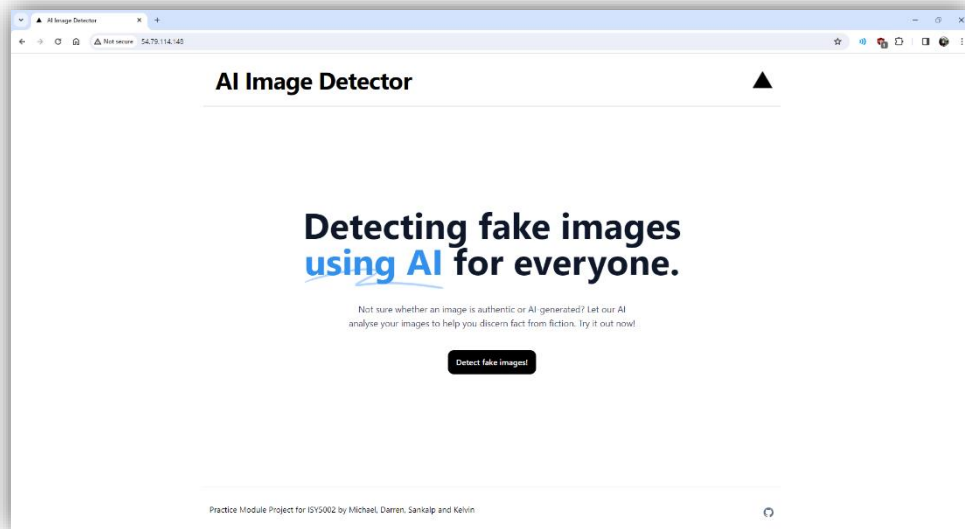


Figure 116: Frontend Main Page

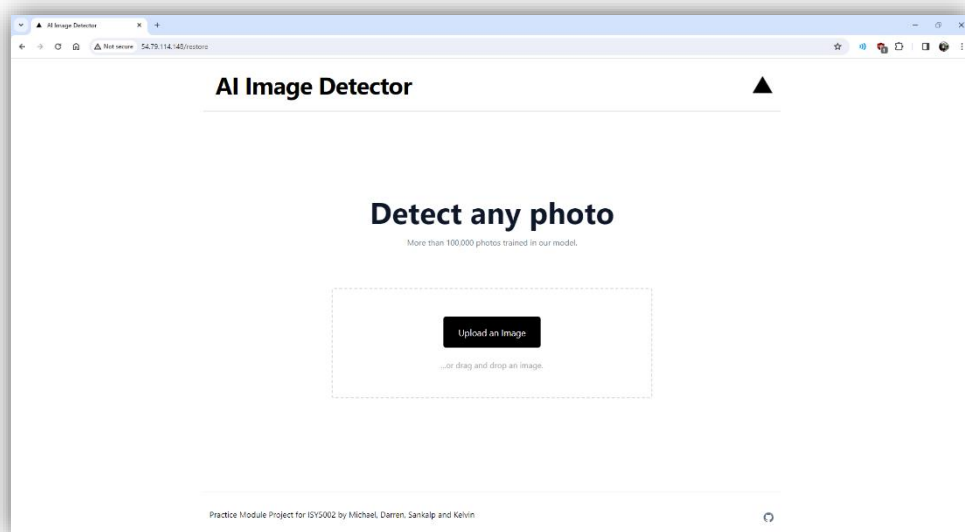


Figure 177: Frontend Upload Page

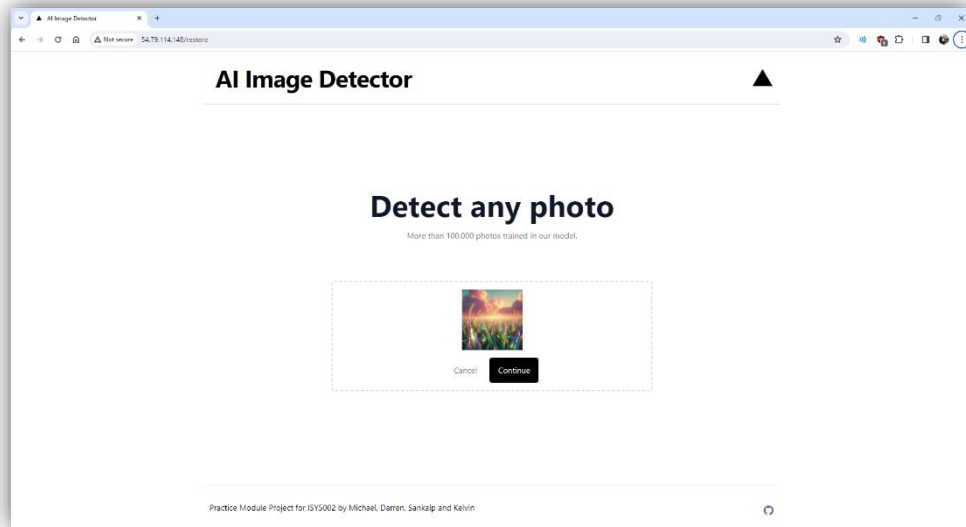


Figure 18: After Upload Image

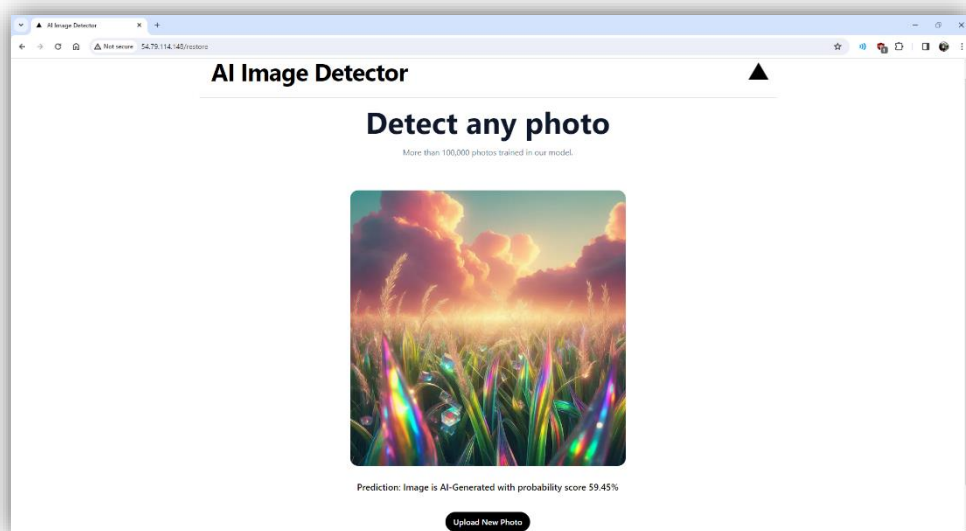


Figure 19: Viewing Prediction Results

Backend Implementation

The backend is done using FastAPI, and is integrated with the trained AI models.

At the initialization of the API, the models to be used are loaded.

```
resnet_loaded_classifier = tf.keras.models.load_model('my_model_resnet_databal.keras')
```

The API will wait for a POST request containing a base64-encoded image from the front end, get the predictions from the model, and send the results back to the front end for display.

```
@app.post("/")
async def process_request(request: Request):
    logging.info('Processing a request.')
    data = await request.json()
    bytes_data = data['bytes_data']
    np_img_cnn = load_image_cnn(bytes_data)
    proba_resnet, result_resnet = cnn_classifier(np_img_cnn, resnet_loaded_classifier)
    return {'result_resnet': result_resnet, 'proba_resnet': proba_resnet}
```

The backend also does image pre-processing by resizing the images to 200 x 200 pixels and ensuring that it is in RGB format. The prediction results consist of the predicted class result along with a probability score.

```
def load_image_cnn(bytes_data):
    bytes_data = base64.b64decode(bytes_data)
    image = BytesIO(bytes_data)
    img = Image.open(image)
    if img.mode != 'RGB':
        img = img.convert('RGB')
    img = img.resize((200, 200))
    np_img = np.array(img) / 255.0
    return np_img

def cnn_classifier(np_img, loaded_classifier):
    np_img = np.expand_dims(np_img, axis=0)
    # loaded_classifier = tf.keras.models.load_model('my_model_200.keras')
    proba = float(loaded_classifier.predict(np_img)[0][0])
    result = int(proba > 0.5)
    print('cnn', proba, result)
    return proba, result
```

8. Conclusion

With the rate of advancement of AI, it is only matter of time before AI becomes either an integral part of our lives or becomes an invasive force in our lives instead.

And with this advancement of AI, AI image generation has become more realistic, and has become a more readily available tool to the masses. With the masses, come all sort of people, including bad-faith actors. This gives rise to people that may use these AI Image Generation tools to create images to stir public outrage. Having a tool readily available to help identify if an image is real, or AI generated can help to reduce such issues.

What we had in mind was to develop something that was easy to access to use, and so we decided on a website form factor, was it can be easily accessed and used by anyone with internet access.

However, as AI Image Generation is also advancing, with more realistic images each time, it may become harder to detect these images as AI Generated. This will require tools like ours to also evolve along. And it may end up being an arms race of sorts between AI Image Generators and AI Image Detection Tools.

Overall, our tools should be used in concurrence with human judgement, as, like any other AI prediction tool, may not always be accurate.

Appendix

1. Project Proposal

GRADUATE CERTIFICATE: Pattern Recognition Systems (PRS)

PRACTICE MODULE: Project Proposal

Date of proposal: 9 September 2023	
Project Title: ISS Project – Fake Image Detection	
Group ID (As Enrolled in Canvas Class Groups): Group 9	
Group Members (Name, Student ID):	
SANKALP	A0226756W
SEOW TECK HAN, MICHAEL	A0270178B
WEE DE LI, DARREN	A0269370X
FONG ZHI EN, KELVIN	A0269365N

Sponsor/Client:

N/A.

Project is done as a public service and may also be of eventual interest to government agencies if it works well.

Background/Aims/Objectives:**Background**

With the rapid advancement of generative Artificial Intelligence models like ChatGPT and Midjourney, the easy generation of synthetic data presents ethical issues which can deceive users with misleading and/or fake news. In particular, the creation of realistic synthetic images has become increasingly prevalent by spreading misinformation through the uploading of deepfakes. These images, generated by algorithms, can often be indistinguishable from photographs taken by physical cameras or created by human artists. This phenomenon has raised concerns regarding the potential for misinformation and manipulation in visual media.

Aims

The project seeks to address this issue by developing a machine learning model capable of accurately discerning between images generated by AI algorithms and those of real images taken by a camera. This distinction is crucial for ensuring the authenticity and integrity of visual content in various domains, including journalism, forensics, and content moderation.

Objective

The objective of this project is to create an intuitive front-end user interface enabling users to upload images for classification. The interface will connect seamlessly with a backend machine learning model, trained on a comprehensive dataset, to accurately determine whether the uploaded image is artificially generated or a real image. This tool aims to provide a user-friendly solution for verifying the authenticity of visual content, addressing concerns related to synthetic imagery in the digital landscape.

Project Descriptions

Comprehensive Dataset Collection and Annotation:

A diverse dataset will be curated, encompassing both synthetic images generated by various AI algorithms and real-world images, which will be labelled for our machine learning model. All images must be of the same size to ensure consistency for the model, especially for neural networks. Initial dataset will be sourced from Kaggle.

<https://www.kaggle.com/datasets/birdy654/cifake-real-and-ai-generated-synthetic-images>

We will also consider introducing data augmentation of the images to increase dataset size to prevent overfitting, e.g., flip horizontally or vertically, rotation to add diversity to the training images.

In addition, we note that the generated images are from stable-diffusion model, and that using the images for training might be inaccurate when it comes to other generative AI image platforms like Dall-E or Midjourney. Therefore, we may consider adding more training images of our own to the training dataset, to increase the accuracy and introduce variety of generated images from different platforms.

Implementation of an image classification model:

This project aims to develop a powerful image classification model by experimenting and testing on several methods, from traditional machine learning methods to deep learning neural networks. This model will be trained on the annotated dataset to accurately distinguish between synthetic and real images.

Traditional machine learning methods:

- 1) Support Vector Machines (SVM)
- 2) Random Forest

Neural Networks/Deep learning methods:

- 1) Traditional Convolutional Neural Network (CNN)
- 2) Transfer Learning

Thorough Model Evaluation:

Rigorous testing and validation procedures will be conducted to assess the classification model's performance. Metrics such as accuracy, precision, recall, F1-score, Confusion Matrix, and ROC-AUC will be employed to quantify and choose the final model's effectiveness.

User-Friendly Web Interface Development:

To facilitate practical application, the project aims to create an intuitive web user interface. This interface will allow users to easily upload images for classification and view the results, including a confidence score indicating the model's level of certainty. This web application can be accessed by both computer and mobile browsers.

Cloud Hosted Model and Web App:

We will aim to host the completed model and the web application on a cloud platform, so that it is accessible online. Through the use of REST APIs, we will connect the front end to a back end that is hosting the model.

2. References

- [CIFAKE: Real and AI-Generated Synthetic Images](#)
- [ArtiFact: Real and Fake Image Dataset](#)
- <https://www.tomorrowstoday.com/2023/04/24/ai-generated-image-wins-world-renowned-photography-competition/>
- <https://fortune.com/2023/03/23/a-i-generated-photo-video-disinformation-internet/>