Team 15: Real or AI image

蔡師睿 林書愷 劉又輔

Motivation

- With the advancement of AI, it’s easy to synthesis images from scratch. This technology indeed benefit humankind in some ways yet it also makes people worried about the credibilty of images. In view of this, our project’s main goal is to classify whether the image is synthesised by AI. The approach we implement includes cnn , GAN, Efficientnets to train models with best learning performance. The result can help us classify image received is genuine resource.

Introduction

- To find out the best-performing method we tried different methods and compared the difference of methods

Convolutional Neural Network

- The most simple way of image classification
- Implementation of convolution layer, maxpooling layer, dense layer will be a great fundamental for futher study
- selected as our baseline

Transfer Learning

- Importing pre-trained efficientnet model to intensify learning rate
- mainly using EfficientNetV2(discriminator) as body and with some extra layers added(not pre-trained)
- selected as our main approach

RelatedWork

- Referring to the paper “CIFAKE: Image Classification and Explainable Identification of AI-Generated Synthetic Images”
- CNN in the paper we tried as baseline
- For further studies, we augmented dataset and implemented transfer learning based on EfficientNetV2 as the main approach
- Use DCGAN to generate more dataset to experiment the models under different fake image sources

Dataset

- **Resource** :“CIFAKE: Real and AI-Generated Synthetic Images”
- **Content** : Test and Train folders, both include Real and Fake folders. Real is genuine images and fake is AI synthesised images by stable diffusion.
- **Image information**: size: 32*32, test : 10000 real and 10000 fake, train: 50000 real and 50000 fake.
- **Usage** : train is used for training models and test is for evaluate accuracy of model predictions.

Dataset : data preprocessing

- In baseline and main approach, the data will be preprocessed to ensure the is cleansed, formalized and credibilty

1. **selection of dataset** : we will randomly select a portion of the images(25000) and the images' label will be memorized based on their credibility (real or fake)

2. **image augmentation** : the selected image will be augmented to increase the features of dataset and the moves including 90 degree rotation and horizontal flip.

3. **data formalization** : outcome of augmentation should be formalized to ensure it satisfies: RGB colored , pixel values normalization(/ 255) , 32*32 sized and the outcome will be shuffled to avoid data cluster

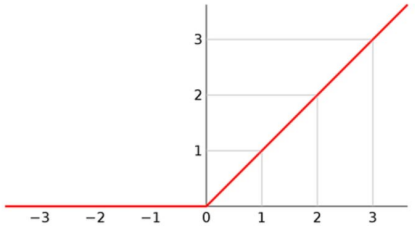
Baseline : Convolutional Neural Network

- **Structure of cnn:**
  1. **Hidden layers** : 4 layer , 2 convolutioning layers(convolution and maxpooling), 2 dense layers
  2. **Activation selection** : Relu(Rectified Linear Unit)
  3. **Output layer** : the last dense layer using **Sigmoid**
  4. **Train set** :
     i. optimizer-> adam
     ii. loss function-> binary_crossentropy
     iii. mertrics-> accuracy

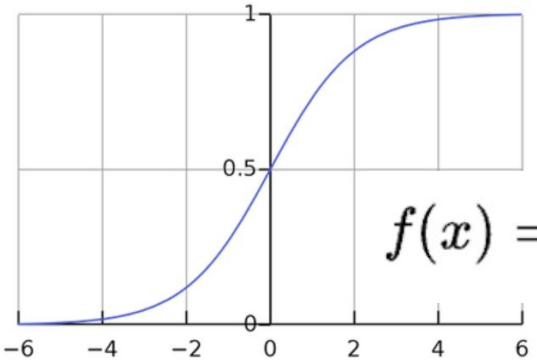
Baseline : Convolutional Neural Network

```python
def CNN():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
    model.add(MaxPooling2D((2, 2)))

    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))

    model.add(Flatten())

    model.add(Dense(64, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

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

    return model
```

CNN source code



Relu $f(x) = \max(0, x)$



Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$

Baseline : Convolutional Neural Network

- **Model training:**
  1. **20 epochs**
  2. **batch size : 200 for model fitting and evaluation**

```
train_ac, train_loss = [], []
test_ac, test_loss = [], []
f1_sc = []

for epoch in range(1, 21):
    history = model.fit(train_x, train_y, epochs=1, batch_size=200)
    loss, ac = model.evaluate(test_x, test_y, batch_size=200)
    predict = model.predict(test_x)
    f1_sc.append(f1(test_y,predict))
    train_loss.append(history.history['loss'])
    train_ac.append(history.history['accuracy'])
    test_loss.append(loss)
    test_ac.append(ac)
```

training stage source code

Main approach : Transfer Learning

- **Method : Layer Transfer**
- **Model building :** Get EfficientNet layers from tensorflow.keras and add layers to form new model
- **Including:**
  - **batchnormalization**
  - **dense**
  - **dropout**

```python
def Built_Model():
    base_model = keras.applications.EfficientNetV2B3(
        include_top=False,
        weights="imagenet",
        input_shape=(32, 32, 3),
        pooling='max'
    )
    base_model.trainable = True

    x = base_model.output
    x = BatchNormalization()(x)
    x = Dense(
        256,
        kernel_regularizer = keras.regularizers.l2(l=0.0095),
        activity_regularizer = keras.regularizers.l1(0.005),
        bias_regularizer = keras.regularizers.l1(0.005),
        activation='relu'
    )(x)
    x = Dropout(rate=0.4, seed=123)(x)

    output = Dense(2, activation='softmax')(x)

    model = keras.models.Model(inputs=base_model.input, outputs=output)

    model.compile(keras.optimizers.Adamax(), loss='categorical_crossentropy', metrics=['accuracy'])
    model.summary()

    return model
```

transfer learning model

Main approach : Transfer Learning

- **More details of added layers:**
  - **batchnormalization: stablelize output of each layers and accelerate learning rate**
  - **dense: import regularizers to regular neurals and eventually avoid overfitting**
  - **activation: relu(before dropout to avoid overfitting), softmax(layer before output, multiple features)**
  - **train set:**
    - **optimizer-> adamax**
    - **loss-> categorical_crossentropy**
    - **metrics-> accuracy**

Main approach : Transfer Learning

- **About EfficientNetV2:**
  - use EfficientNetV2B3(in tensorflow application)
    - batch = 200
    - epoch = 20
  - rescaling of cnn, corresponding to baseline
  - multiple stage(formed by different types of cnn) constructed
  - selecting B3 due to it has largest depth in V2
  - should have best performance due to network size

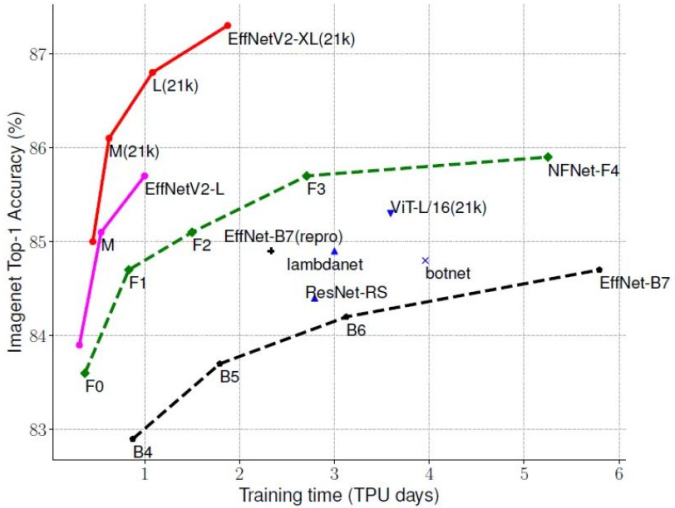
Main approach : Transfer Learning

- **EfficientNet Archietecture:**

Table 4. EfficientNetV2-S architecture – MBConv and Fused-MBConv blocks are described in Figure 2.

Stage	Operator	Stride	#Channels	#Layers
0	Conv3x3	2	24	1
1	Fused-MBConv1, k3x3	1	24	2
2	Fused-MBConv4, k3x3	2	48	4
3	Fused-MBConv4, k3x3	2	64	4
4	MBConv4, k3x3, SE0.25	2	128	6
5	MBConv6, k3x3, SE0.25	1	160	9
6	MBConv6, k3x3, SE0.25	2	272	15
7	Conv1x1 & Pooling & FC	-	1792	1

EfficientNet V2架構 (資料來源)



(a) Training efficiency.

Main approach(Extra) : ResNet50

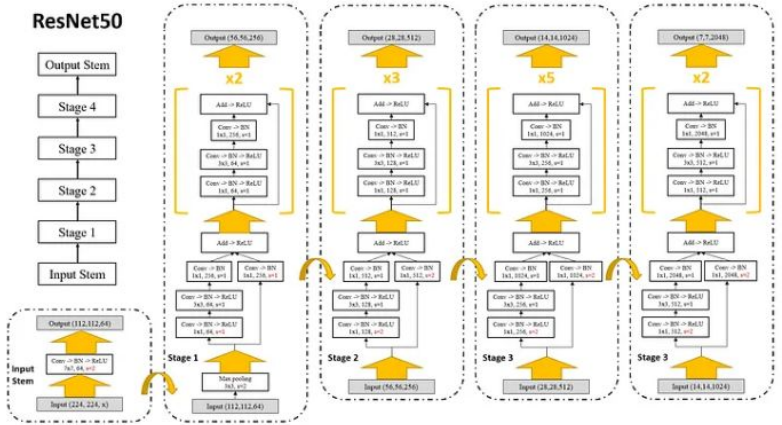
- **ResNet50 Archietecture:**
  - **Method : similar to EfficientV2**
  - **use ResNet instead**
  - **only added dense and Dropout layers**
  - **train set:**
    - **optimizer->Adam,learning rate=0.0007**
    - **loss->categorical crossentropy**
    - **metrics->accuracy**

```
def Built_Model():
    base_model = keras.applications.ResNet50(
        include_top=False,
        weights="imagenet",
        input_shape=(32, 32, 3),
        pooling='max'
    )
    base_model.trainable = True

    x = base_model.output
    x = Dense(256, activation='relu')(x)
    x = Dropout(rate=0.6)(x)
    output = Dense(2, activation='softmax')(x)

    model = keras.models.Model(inputs=base_model.input, outputs=output)

    model.compile(keras.optimizers.Adam(0.0007), loss='categorical_crossentropy', metrics=['accuracy'])
    model.summary()

    return model
```

Main approach(Extra) : ResNet50

- **ResNet50 Archietecture:**
  - **performance is lower than EfficientNetV2**
  - **further study will not use this**
  - **quick note:**
    - **input stem: use convolution as cnn and lower resolution**
    - **stage block: similar to EfficientNet, 4**
    - **output stem: we built our own**

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9



Main approach(Extra) : ResNet50

- **ResNet50 Archietecture:**
  - **train session:**
    - **batch = 200**
    - **epoch = 20**

```
history = model.fit(train_data, steps_per_epoch=len(train_data), epochs=20, validation_data=test_data)
```

Advanced Experiment : DCGAN

- **Besides stable diffusion synthesised dataset , we created a DCGAN model to test our CNN and EfficientNetV2 model**
- **CNN and EfficientNetV2 models will be trained by stable diffusion dataset**
- **Goal is to verify our models can classify fake images in different features**

DCGAN

- **Structure of GAN: Generator and Discriminator**
- **Generator :**
  - **Hidden layer : 7 layers**
  - **will generate new images based on test images from cifake dataset**
  - **use conv2dTranspose and batchnormalization to restore features**
  - **use leakyRelu as activation function to preserved negative features while lowering their weights**

generator source code

```
def Generator():
    model = Sequential()
    model.add(Dense(8*8*256, use_bias=False, input_shape=(100,)))
    model.add(BatchNormalization())
    model.add(LeakyReLU())

    model.add(Reshape((8, 8, 256)))

    model.add(Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
    model.add(BatchNormalization())
    model.add(LeakyReLU())

    model.add(Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    model.add(BatchNormalization())
    model.add(LeakyReLU())

    model.add(Conv2DTranspose(3, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))

    model.summary()
    return model
```

DCGAN

- **Discriminator:**
  - **Hidden layer : 5 layers**
  - **trained with cifake dataset and generator synthesis images**
  - **use conv2d to extract features**
  - **use leakyRelu as activation function to preserved negative features while lowering their weights**
  - **use dropout to accelerate training and avoid overfitting**

discriminator source code

```
def Discriminator():
    model = Sequential()
    model.add(Conv2D(64, (5, 5), strides=(2, 2), padding='same', input_shape=(32, 32, 3)))
    model.add(LeakyReLU())
    model.add(Dropout(0.3))

    model.add(Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(LeakyReLU())
    model.add(Dropout(0.3))

    model.add(Flatten())
    model.add(Dense(1))

    model.summary()
    return model
```

DCGAN

- **loss function:**
  - **discriminator : will get losses from real and fake out and the total loss will be calculated by cross entropy**
  - **generator : based on the fake losses cross entropy**

```python
def discriminator_loss(real_output, fake_output):
    cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss

def generator_loss(fake_output):
    cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
    return cross_entropy(tf.ones_like(fake_output), fake_output)
```

DCGAN

- **train session:**
  - **epoch = 50**
  - **both optimizers are adam**
  - **step:**
    - **generate images with noise 100(random n)**
    - **train discriminator with real and generated images**
    - **calculate losses and renew gradients and both optimizers**

```python
def train(dataset, epochs, noise, generator, discriminator, generator_opt, discriminator_opt):
    for epoch in range(epochs):
        for image_batch in tqdm(dataset, desc='Batch'):
            train_step(image_batch, epoch, noise, generator, discriminator, generator_opt, discriminator_opt)
```

epoches

step

```python
def train_step(image_batch, epoch, noise, generator, discriminator, generator_opt, discriminator_opt):
    noise = tf.random.normal([200, noise])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)
        real_output = discriminator(image_batch, training=True)
        fake_output = discriminator(generated_images, training=True)
        gen_loss = generator_loss(fake_output)
        disc_loss = discriminator_loss(real_output, fake_output)

    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)
    generator_opt.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
    discriminator_opt.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables))
```

Evalutaion metrics : f1 score and accuracy

- **accuracy:**

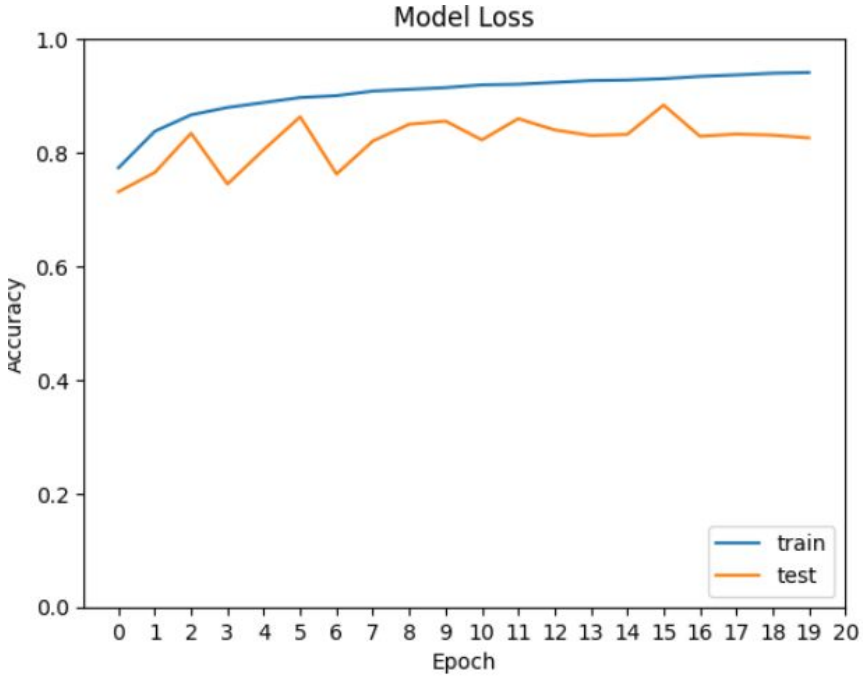
$$accuracy = \frac{1}{n}\sum_{i=1}^{n} f(x) \quad where\ f(x) = \begin{cases} 1 & if\ output = label \\ 0 & otherwise \end{cases}$$

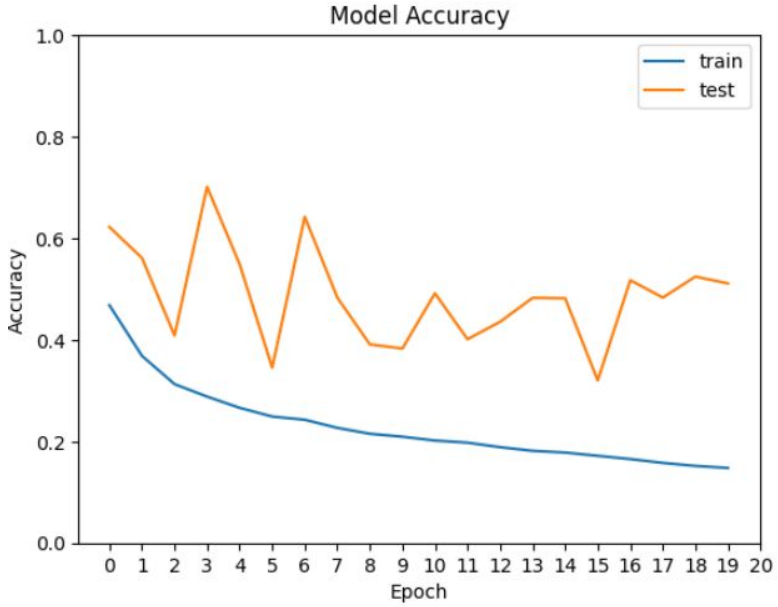
- **F1 score:**

```
Recall(召回率) = TP/(TP+FN)
Precision(準確率) = TP/(TP+FP)
F1-score = 2 * Precision * Recall / (Precision + Recall)
```

Result

- **performance (CNN with cifake dataset only) :**





Result

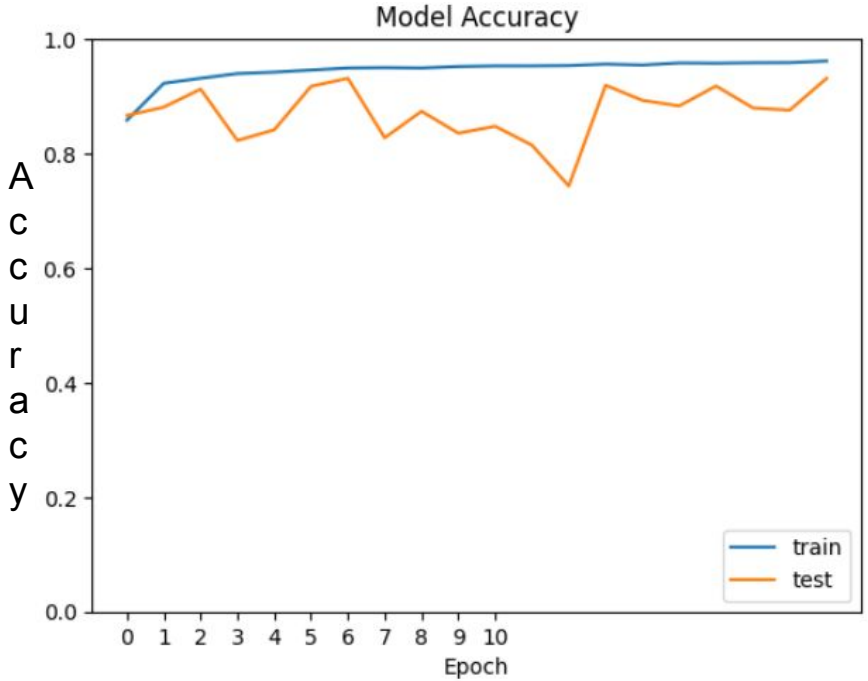
- **performance (CNN with cifake dataset only) :**

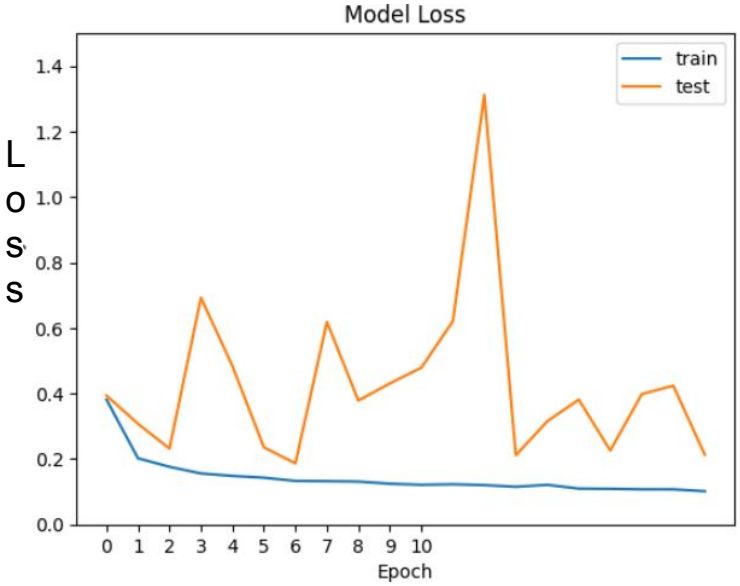
Classification Report:

	precision	recall	f1-score	support
0	0.75	0.99	0.85	10000
1	0.99	0.66	0.79	10000
accuracy			0.83	20000
macro avg	0.87	0.83	0.82	20000
weighted avg	0.87	0.83	0.82	20000

Result

- **performance (ResNet with cifake dataset only) :**





Result

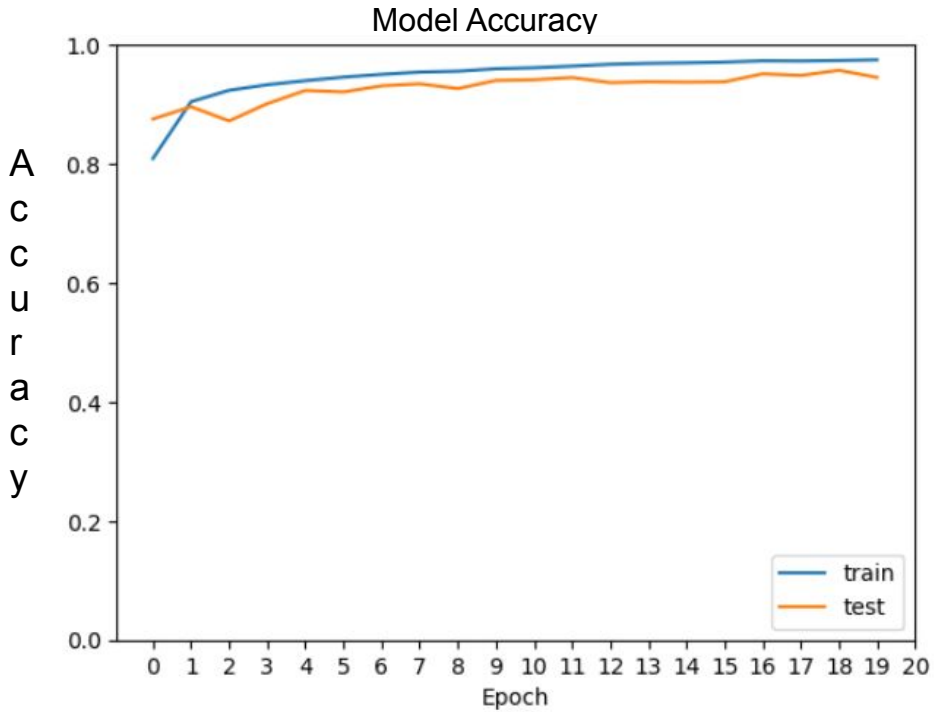
- **performance (ResNet with cifake dataset only) :**

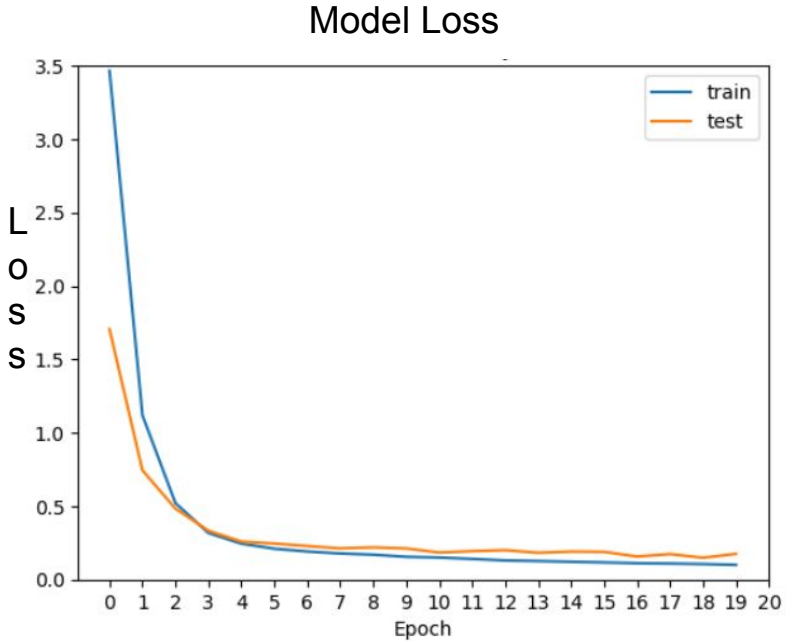
Classification Report:

	precision	recall	f1-score	support
0	0.89	0.98	0.93	10000
1	0.98	0.88	0.93	10000
accuracy			0.93	20000
macro avg	0.94	0.93	0.93	20000
weighted avg	0.94	0.93	0.93	20000

Result

- **performance (EfficientNetV2 with cifake dataset only) :**





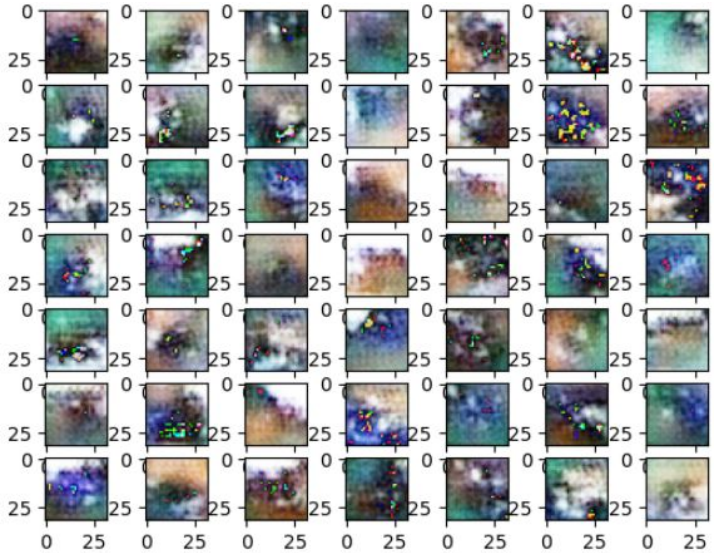
Result

- **performance (EfficientNetV2 with cifake dataset only) :**

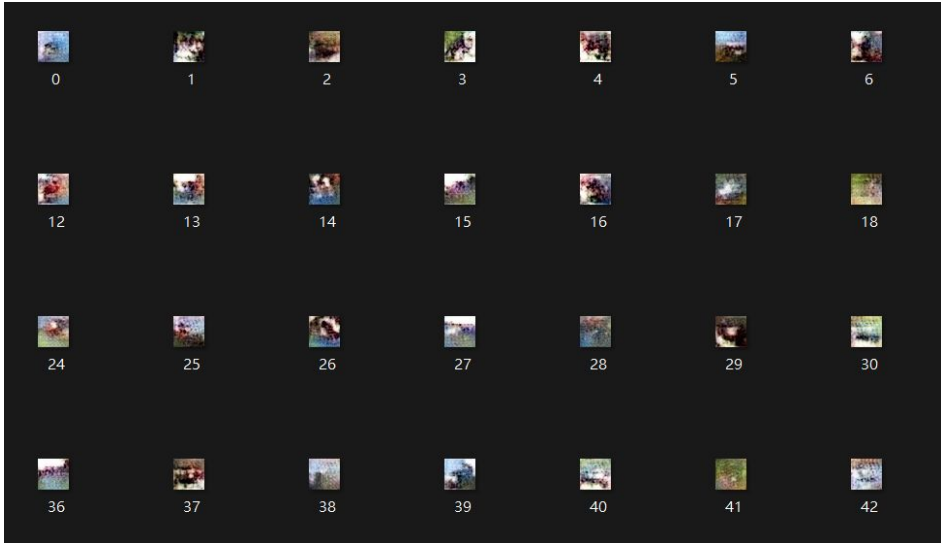
```
Classification Report:
----------------------
              precision    recall  f1-score   support

           0       0.91      0.99      0.95     10000
           1       0.99      0.90      0.94     10000

    accuracy                           0.95     20000
   macro avg       0.95      0.95      0.95     20000
weighted avg       0.95      0.95      0.95     20000
```

Result

- **performance (DCGAN with cifake dataset only) :**



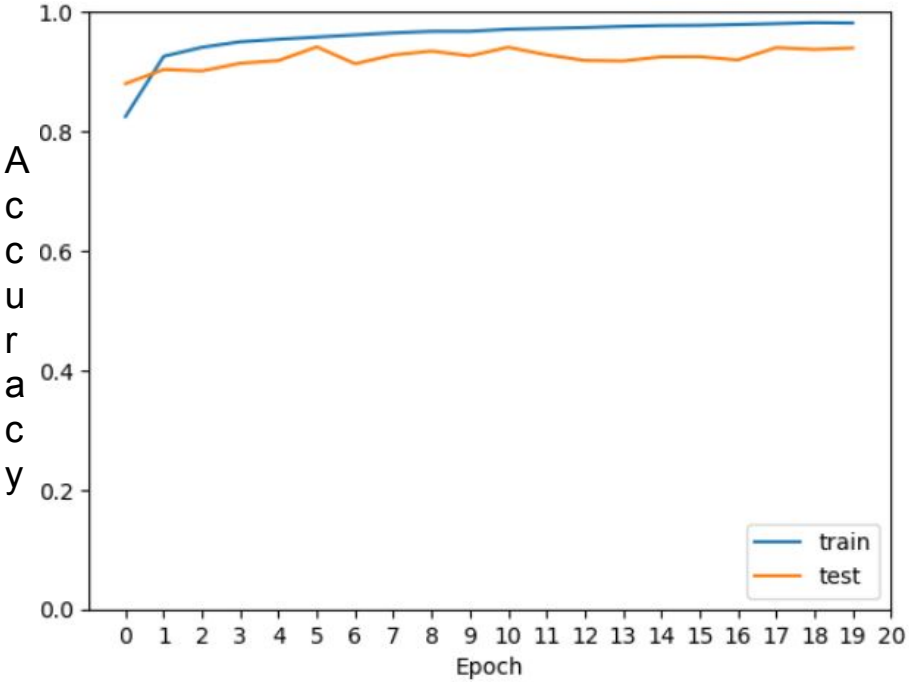
DCGAN generated images based on cifake



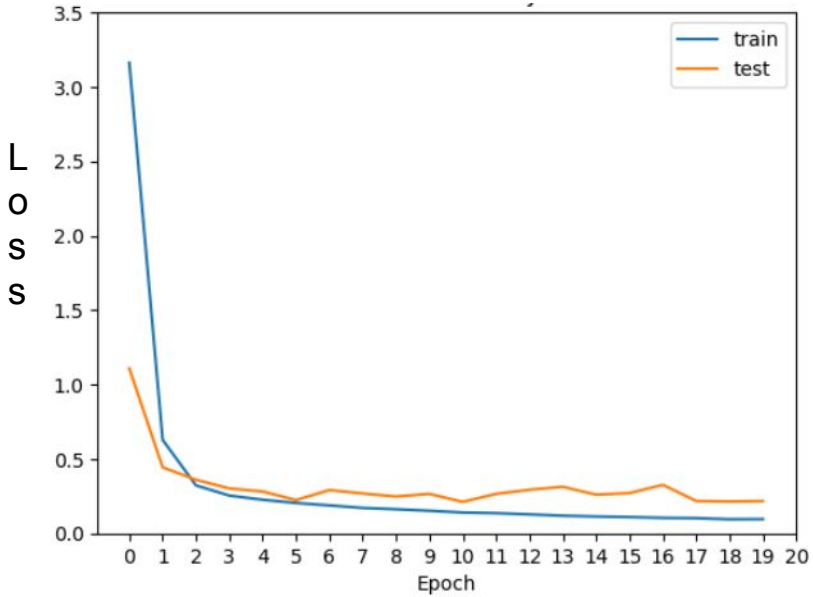
Result

- **performance (EfficientNetV2 with DCGAN and cifake as dataset) :**

Model Accuracy



Model Loss



Result

- **performance (EfficientNetV2 with DCGAN and cifake as dataset) :**

Classification Report:

	precision	recall	f1-score	support
0	0.92	0.99	0.95	10000
1	1.00	0.89	0.94	5000
2	0.93	0.92	0.92	10000
accuracy			0.94	25000
macro avg	0.95	0.93	0.94	25000
weighted avg	0.94	0.94	0.94	25000

Result

- **performance (CNN and EfficientNetV2 and EfficientNetV2_improved test with DCGAN) :**

```
Evaluate the DCGAN generated images with CNN model.
32/32 [==============================] - 0s 3ms/step - loss: 2.6799 - accuracy: 0.4100

Evaluate the DCGAN generated images with EfficientNet model.
32/32 [==============================] - 1s 18ms/step - loss: 6.2624 - accuracy: 0.0020

Evaluate the DCGAN generated images with improved EfficientNet model.
32/32 [==============================] - 1s 16ms/step - loss: 0.2167 - accuracy: 0.9430
```

Analysis

- **Discuss :**
  - **CNN and Transfer learning both has great performance in classify stable diffusion dataset(cifake) while trained by stable diffusion dataset and transfer learning is higher**
  - **However if it is trained by stable diffusion dataset, both has difficulty classify DCGAN generated images**
  - **only if we add DCGAN generated images into training dataset can enhance the performance**
  - **In conclusion, image classification will be limited by its inputting source, if the target dataset possess different features, the model's accuracy will decrease significantly**

Analysis

- **Solution :**
  - **Add some portion of the target dataset into training session**
  - **frozen top layers of current model and to further transfer learning(ex. transfer layer)**
- **In our solution :**
  - **we adopt first solution and the performance has increased significantly**
  - **yet adding more feature will lower the ability of classifying former task**
  - **In sum, can’t have it both way**
  - **but in this case our method works**

Analysis

- **Experiment : mention before**
- **limitation : classification models can only classify images that possess similar features to its training dataset**
- **Application :**
  - **deepfake detection : judge images is real people or fake ones**
  - **image authentication : verified the image's value and authenticity , which will be very helpful in realm of art**
  - **security : differentiate between ai synthesised images can prevent identity theft and fraud**

Reference

- Bird, J.J., Lotfi, A. (2023). CIFAKE: Image Classification and Explainable Identification of AI-Generated Synthetic Images. arXiv preprint arXiv:2303.14126
- CIFAKE: Real and AI-Generated Synthetic Images
- https://medium.com/aiguys/review-efficientnetv2-smaller-models-and-faster-training-47d4215dcdfb
- https://medium.com/swlh/dcgan-deep-convolutional-generative-adversarial-network-1a2e55c35133
- https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4

Link

- GitHub Link : https://github.com/Ray-1026/Real-or-AI-Image
- YouTube presentation link: https://youtu.be/XWz7CX_dI-Q

Contribution of each member

蔡師睿：EfficientNet、DCGAN、GitHub management

林書愷：ResNet50、video、slides

劉又輔：CNN、slides