Maching Learing - Final Project report

109611092 李耀凱

1.Environment details

a. Python version: 3.10.12

```
import sys
print(sys.version)
```

```
3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0]
```

b. Framework :
For Deep Learning, I used TensorFlow and Keras as my frame
Also, I use train_test_split() from scikit-learn to cut the train data into training and valid data

```
!pip install Keras-Preprocessing
!pip install tensorflow
!pip install efficientnet
!pip install matplotlib seaborn
!pip install scikit-learn
```

c. Hardware: T4 GPU (On Google Colab)

```
!nvidia-smi
```

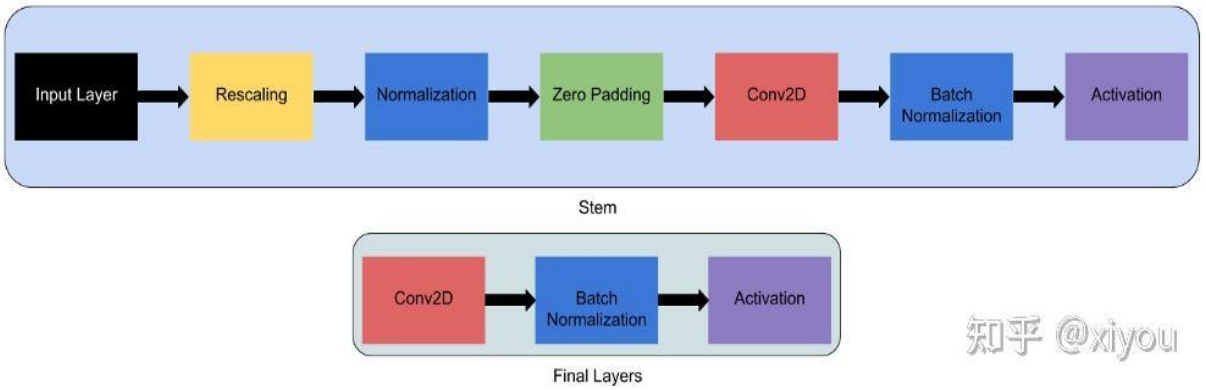
```
Fri Jan  5 05:33:17 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |                    0 |
| N/A   63C    P0              29W /  70W |   2173MiB / 15360MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
+---------------------------------------------------------------------------------------+
```

2.Implementation details

a. Model architecture

EfficientNetB0 is a model introduced by the Google Brain team in 2019. Its main features are efficiency and lightweight design.



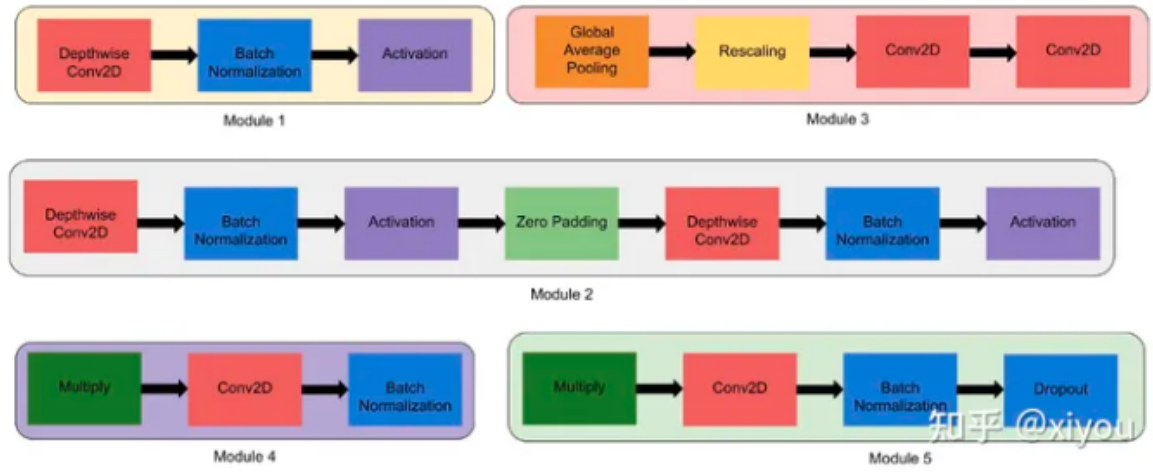
As in the diagram above, the "stem" of this model consists of 7 blocks. Each block is composed of:

I. InputLayer: Accepts the model's input.
II. Rescaling: Rescales the input data to a specific range.
III. Normalization: Normalizes the input data, making its mean 0 and standard deviation 1.
IV. ZeroPadding: Adds zeros around the edges of the input data to increase its dimensions.
V. Conv2D: A 2D convolutional layer used to extract features from the input data.
VI. BatchNormalization: Accelerates the training process by normalizing the activations of the layer.
VII. Activation: Introduces non-linearity to the model through an activation function.

The "final layer" consists of:

Conv2D / BatchNormalization / Activation function.

This model has 237 layers, and all of these layers are composed of the 5 modules and stem below :



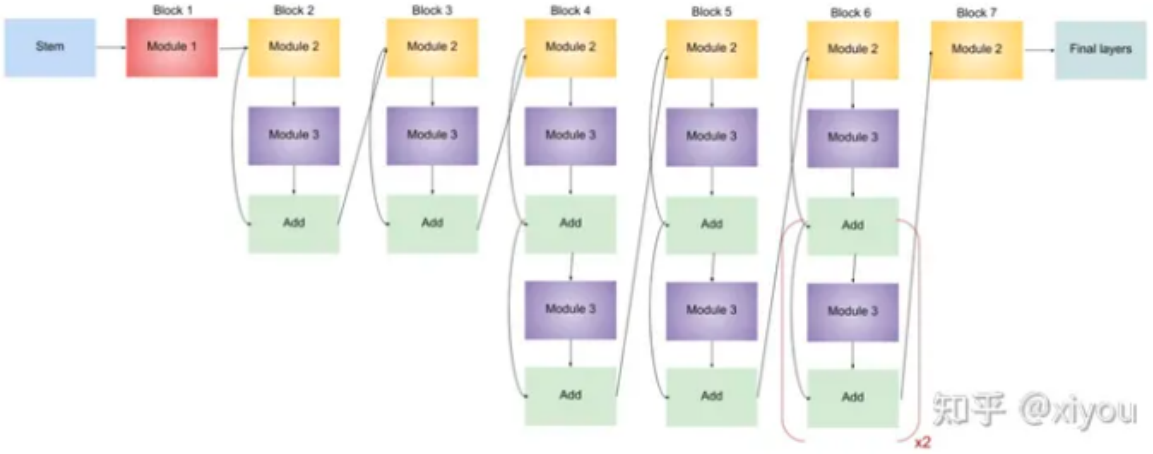
Module 1: This serves as the starting point for the sub-blocks.

Module 2: This is used as the starting point for the first sub-block of all 7 main blocks, except for the first one.

Module 3: This is the skip connection that connects with all sub-blocks.

Module 4: Used to combine the skip connections from the first sub-block.

Module 5: Each sub-block is connected to its preceding sub-block in a skip connection manner, and this module is used to merge them.



The final model architecture is composed of these 5 modules.
reference : https://zhuanlan.zhihu.com/p/366738106

b. Hyperparameters

I. epoch = 70
II. batch_size = 32
III. shuffle = false
IV. randomseed = 42
V. drop out = 0.5

VI. dense = 512
VII. normalization : L2, 0.01
VIII. optimizer : Adam, lr = 0.00015

c. Training strategy

I. Data Preprocessing:
Employed ImageDataGenerator for data augmentation and data generation.

II. Data Segmentation:
Split the original dataset into training and testing data. (0.7 / 0.3)

III. Model Selection:
Used EfficientNetB0 as the base model and froze all its layers.

IV. Model Adjustment:
Added additional fully connected layers on top of the EfficientNet output and modified the structure of the model.

V. Training Configuration:
Utilized the Adam optimizer and categorical_crossentropy as the loss function.
Set up callbacks for early stopping, model checkpointing, and learning rate reduction.

```
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
checkpoint_callback = ModelCheckpoint('model_checkpoint.h5', monitor='val_accuracy', save_best_only=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, min_lr=1e-6)
```

3.Experimental results

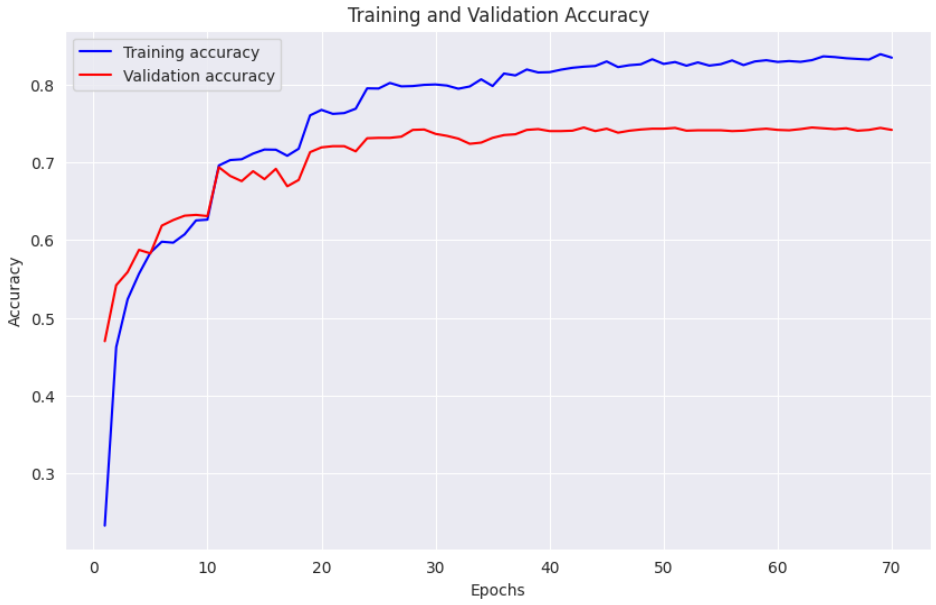
Evaluation metrics

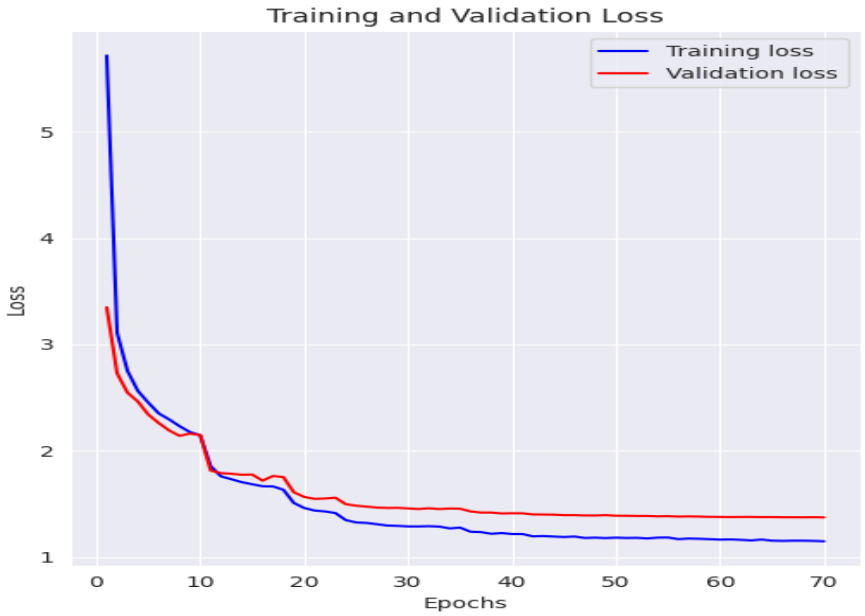
Accuracy : Calculates the number of correct predictions divided by the total number of samples. This is the most common evaluation metric for classification problems.

```
Epoch 1/70
245/245 [==============================] - ETA: 0s - loss: 5.7179 - accuracy: 0.2327 /usr/local/lib/python3.10/dist-packages/keras/src/engine/traini
  saving_api.save_model(
245/245 [==============================] - 3455s 14s/step - loss: 5.7179 - accuracy: 0.2327 - val_loss: 3.3497 - val_accuracy: 0.4699 - lr: 0.0010
Epoch 2/70
245/245 [==============================] - 41s 167ms/step - loss: 3.1099 - accuracy: 0.4622 - val_loss: 2.7290 - val_accuracy: 0.5419 - lr: 0.0010
Epoch 3/70
245/245 [==============================] - 38s 154ms/step - loss: 2.7510 - accuracy: 0.5238 - val_loss: 2.5465 - val_accuracy: 0.5587 - lr: 0.0010
Epoch 4/70
245/245 [==============================] - 38s 155ms/step - loss: 2.5629 - accuracy: 0.5566 - val_loss: 2.4629 - val_accuracy: 0.5873 - lr: 0.0010
Epoch 5/70
245/245 [==============================] - 36s 147ms/step - loss: 2.4526 - accuracy: 0.5838 - val_loss: 2.3409 - val_accuracy: 0.5827 - lr: 0.0010
Epoch 6/70
245/245 [==============================] - 38s 155ms/step - loss: 2.3518 - accuracy: 0.5978 - val_loss: 2.2608 - val_accuracy: 0.6185 - lr: 0.0010
Epoch 7/70
245/245 [==============================] - 39s 160ms/step - loss: 2.2955 - accuracy: 0.5966 - val_loss: 2.1917 - val_accuracy: 0.6256 - lr: 0.0010
Epoch 8/70
245/245 [==============================] - 41s 167ms/step - loss: 2.2316 - accuracy: 0.6073 - val_loss: 2.1400 - val_accuracy: 0.6313 - lr: 0.0010
Epoch 9/70
245/245 [==============================] - 37s 150ms/step - loss: 2.1758 - accuracy: 0.6252 - val_loss: 2.1614 - val_accuracy: 0.6323 - lr: 0.0010
Epoch 10/70
245/245 [==============================] - 37s 152ms/step - loss: 2.1455 - accuracy: 0.6262 - val_loss: 2.1475 - val_accuracy: 0.6307 - lr: 0.0010
```

```
245/245 [==============================] - 37s 153ms/step - loss: 1.1681 - accuracy: 0.8312 - val_loss: 1.3783 - val_accuracy: 0.7431 - lr: 1.5625e-05
Epoch 60/70
245/245 [==============================] - 35s 144ms/step - loss: 1.1642 - accuracy: 0.8290 - val_loss: 1.3773 - val_accuracy: 0.7416 - lr: 1.5625e-05
Epoch 61/70
245/245 [==============================] - 36s 146ms/step - loss: 1.1657 - accuracy: 0.8301 - val_loss: 1.3764 - val_accuracy: 0.7411 - lr: 1.5625e-05
Epoch 62/70
245/245 [==============================] - 36s 146ms/step - loss: 1.1620 - accuracy: 0.8291 - val_loss: 1.3771 - val_accuracy: 0.7426 - lr: 1.5625e-05
Epoch 63/70
245/245 [==============================] - 37s 151ms/step - loss: 1.1562 - accuracy: 0.8313 - val_loss: 1.3774 - val_accuracy: 0.7446 - lr: 1.5625e-05
Epoch 64/70
245/245 [==============================] - 36s 147ms/step - loss: 1.1646 - accuracy: 0.8361 - val_loss: 1.3759 - val_accuracy: 0.7436 - lr: 7.8125e-06
Epoch 65/70
245/245 [==============================] - 36s 148ms/step - loss: 1.1539 - accuracy: 0.8352 - val_loss: 1.3760 - val_accuracy: 0.7426 - lr: 7.8125e-06
Epoch 66/70
245/245 [==============================] - 37s 151ms/step - loss: 1.1520 - accuracy: 0.8337 - val_loss: 1.3746 - val_accuracy: 0.7436 - lr: 7.8125e-06
Epoch 67/70
245/245 [==============================] - 37s 152ms/step - loss: 1.1536 - accuracy: 0.8328 - val_loss: 1.3743 - val_accuracy: 0.7406 - lr: 7.8125e-06
Epoch 68/70
245/245 [==============================] - 37s 149ms/step - loss: 1.1539 - accuracy: 0.8321 - val_loss: 1.3739 - val_accuracy: 0.7416 - lr: 7.8125e-06
Epoch 69/70
245/245 [==============================] - 36s 147ms/step - loss: 1.1518 - accuracy: 0.8390 - val_loss: 1.3748 - val_accuracy: 0.7441 - lr: 7.8125e-06
Epoch 70/70
245/245 [==============================] - 37s 149ms/step - loss: 1.1484 - accuracy: 0.8345 - val_loss: 1.3734 - val_accuracy: 0.7416 - lr: 7.8125e-06
```

Learning curve :





Ablation Study :

In this model, I've tried several super parameters to run it.

for learning rate, I chose 0.01 / 0.001 / 0.0001 / 0.00001 / 0.0015 as the initial value and tune the reduction rate with 0.2 / 0.3 / 0.5
However, for the initial rate >= 0.001 or < 0.0001 , It performed awfully.
So, I didn't save the result.

I also tuned the epoch in order to fulfill the reduction of learning rate to small enough from 30 to 50 to 70

Also, I've trained 4-5 models as well (EfficientNetB7, VGG16, InceptionV3, ResNet50) In theory, EfficientNetB7 and ResNet50 should perform better
However, they didn't.

Here are the result
EfficientNetB7 :

```
Epoch 21/30
245/245 [==============================] - 116s 472ms/step - loss: 1.1867 - accuracy: 0.8107 - val_loss: 1.5636 - val_accuracy: 0.6839 -
lr: 6.2500e-05
Epoch 22/30
245/245 [==============================] - 117s 479ms/step - loss: 1.1817 - accuracy: 0.8119 - val_loss: 1.5619 - val_accuracy: 0.6864 -
lr: 6.2500e-05
Epoch 23/30
245/245 [==============================] - 115s 470ms/step - loss: 1.1753 - accuracy: 0.8129 - val_loss: 1.5621 - val_accuracy: 0.6854 -
lr: 6.2500e-05
Epoch 24/30
245/245 [==============================] - 116s 473ms/step - loss: 1.1672 - accuracy: 0.8183 - val_loss: 1.5547 - val_accuracy: 0.6828 -
lr: 6.2500e-05
Epoch 25/30
245/245 [==============================] - 116s 473ms/step - loss: 1.1621 - accuracy: 0.8193 - val_loss: 1.5524 - val_accuracy: 0.6854 -
lr: 6.2500e-05
Epoch 26/30
245/245 [==============================] - 115s 471ms/step - loss: 1.1594 - accuracy: 0.8166 - val_loss: 1.5598 - val_accuracy: 0.6777 -
lr: 6.2500e-05
Epoch 27/30
245/245 [==============================] - 118s 479ms/step - loss: 1.1579 - accuracy: 0.8134 - val_loss: 1.5503 - val_accuracy: 0.6879 -
lr: 6.2500e-05
Epoch 28/30
245/245 [==============================] - 116s 472ms/step - loss: 1.1589 - accuracy: 0.8107 - val_loss: 1.5392 - val_accuracy: 0.6859 -
lr: 6.2500e-05
Epoch 29/30
245/245 [==============================] - 115s 471ms/step - loss: 1.1538 - accuracy: 0.8135 - val_loss: 1.5451 - val_accuracy: 0.6828 -
lr: 6.2500e-05
Epoch 30/30
245/245 [==============================] - 117s 477ms/step - loss: 1.1591 - accuracy: 0.8149 - val_loss: 1.5434 - val_accuracy: 0.6890 -
lr: 6.2500e-05
```

Resnet50 :

```
244/244 [==============================] - 124s 507ms/step - loss: 0.9428 - accuracy: 0.7339 - val_loss: 1.2213 - val_accuracy: 0.6470
Epoch 20/30
244/244 [==============================] - 127s 519ms/step - loss: 0.9515 - accuracy: 0.7329 - val_loss: 1.2313 - val_accuracy: 0.6440
Epoch 21/30
244/244 [==============================] - 124s 507ms/step - loss: 0.9533 - accuracy: 0.7308 - val_loss: 1.2332 - val_accuracy: 0.6450
Epoch 22/30
244/244 [==============================] - 124s 510ms/step - loss: 0.9472 - accuracy: 0.7340 - val_loss: 1.2345 - val_accuracy: 0.6470
Epoch 23/30
244/244 [==============================] - 124s 506ms/step - loss: 0.9305 - accuracy: 0.7381 - val_loss: 1.2323 - val_accuracy: 0.6481
Epoch 24/30
244/244 [==============================] - 126s 514ms/step - loss: 0.9400 - accuracy: 0.7363 - val_loss: 1.2297 - val_accuracy: 0.6496
Epoch 25/30
244/244 [==============================] - 124s 507ms/step - loss: 0.9529 - accuracy: 0.7290 - val_loss: 1.2202 - val_accuracy: 0.6522
Epoch 26/30
244/244 [==============================] - 124s 509ms/step - loss: 0.9296 - accuracy: 0.7368 - val_loss: 1.2311 - val_accuracy: 0.6486
Epoch 27/30
244/244 [==============================] - 123s 505ms/step - loss: 0.9103 - accuracy: 0.7444 - val_loss: 1.2317 - val_accuracy: 0.6491
Epoch 28/30
244/244 [==============================] - 124s 508ms/step - loss: 0.9309 - accuracy: 0.7426 - val_loss: 1.2282 - val_accuracy: 0.6486
Epoch 29/30
244/244 [==============================] - 125s 510ms/step - loss: 0.9279 - accuracy: 0.7399 - val_loss: 1.2290 - val_accuracy: 0.6465
Epoch 30/30
244/244 [==============================] - 125s 513ms/step - loss: 0.9120 - accuracy: 0.7439 - val_loss: 1.2263 - val_accuracy: 0.6460
```

I spent lots of time fine tuning these models, however, I cannot get higher accuracy
For VGG16 and InceptionV3 I get even lower score
I thought I've tried as much as possible, I have change the preprocessing ways, epochs, learning rate, and the size of train set, but it still cannot surpass 75% : (

**要重現code, 須將dataset
https://drive.google.com/drive/folders/11d-cnnm0A6_t4bHq7dc4FChSMx8r0qBM?usp=sharing
資料夾複製到該帳戶的雲端(My drive)中就可以mount 並訓練使用