SIT744 Assignment 2: Deep Learning Project

Due: **8:00pm 29 May 2022** (Monday)

This is an **individual** assignment. It contributes **40%** to your final mark. Read the assignment instruction carefully.

What to submit

This assignment is to be completed individually and submitted to CloudDeakin. **By the due date, you are required to submit the following files to the corresponding Assignment (Dropbox) in CloudDeakin**:

1. **[YourID]_assignment2_solution.ipynb**: This is your Python notebook solution source file.
2. **[YourID]_assingment2_output.html** or **[YourID]_assingment2_output.pdf**: This is the output of your Python notebook solution exported in HTML or PDF format.
3. Extra files needed to complete your assignment, if any (e.g., images used in your answers).

For example, if your student ID is: 123456, you will then need to submit the following files:

- 123456_assignment2_solution.ipynb
- 123456_assignment2_output.html

Warning

Some components of this assignment may involve heavy computation that runs for a long duration. Please start early to avoid missing the assignment due date.

Marking criteria

Your submission will be assessed based on the overall impact of your effort. A useful model (or application) should be your focus. But as in Assignment 1, we will also consider the following criteria at the same time.

- Showing good effort through completed tasks.
- Applying deep learning theory to design suitable deep learning solutions for the tasks.
- Critically evaluating and reflecting on the pros and cons of various design decisions.
- Demonstrating creativity and resourcefulness in providing unique individual solutions.
- Showing attention to detail through a good quality assignment report.

Assignment objective

This assignment is to feedback on your learning in deep learning theory and its application to data analytics or artificial intelligence problems.

It builds on Assignment 1 but requires a higher level of mastery of deep learning theory and programming/engineering skills. In particular, you will practice making design decisions yourself. You are also likely to encounter practical issues that will help you consolidate textbook learning.

```
%pip install tensorflow_addons vit-keras opencv-python kaggle tensorboard-plugin-profile
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting tensorflow_addons
  Downloading tensorflow_addons-0.20.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (591 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 591.0/591.0 kB 18.5 MB/s eta 0:00:00
Collecting vit-keras
  Downloading vit_keras-0.1.2-py3-none-any.whl (24 kB)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages (4.7.0.72)
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.5.13)
Collecting tensorboard-plugin-profile
  Downloading tensorboard_plugin_profile-2.13.0-py3-none-any.whl (5.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.4/5.4 MB 67.5 MB/s eta 0:00:00
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow_addons) (23.1)
Collecting typeguard<3.0.0,>=2.7 (from tensorflow_addons)
  Downloading typeguard-2.13.3-py3-none-any.whl (17 kB)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from vit-keras) (1.10.1)
Collecting validators (from vit-keras)
  Downloading validators-0.20.0.tar.gz (30 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from opencv-python) (1.22.4)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from kaggle) (2022.12.7)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.27.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.65.0)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.1)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.26.15)
Collecting gviz-api>=1.9.0 (from tensorboard-plugin-profile)
  Downloading gviz_api-1.10.0-py2.py3-none-any.whl (13 kB)
Requirement already satisfied: protobuf>=3.19.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard-plugin-prof
```

```
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard-plugin-pr
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.10/dist-packages (from tensorboard-plugin-pro
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=0.11.15->ten
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kagg
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests->kagg
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.4)
Requirement already satisfied: decorator>=3.4.0 in /usr/local/lib/python3.10/dist-packages (from validators->vit-keras)
Building wheels for collected packages: validators
  Building wheel for validators (setup.py) ... done
  Created wheel for validators: filename=validators-0.20.0-py3-none-any.whl size=19579 sha256=3991a0915bd7b7103fcab5ba3d
  Stored in directory: /root/.cache/pip/wheels/f2/ed/dd/d3a556ad245ef9dc570c6bcd2f22886d17b0b408dd3bbb9ac3
Successfully built validators
Installing collected packages: validators, typeguard, gviz-api, vit-keras, tensorflow_addons, tensorboard-plugin-profile
Successfully installed gviz-api-1.10.0 tensorboard-plugin-profile-2.13.0 tensorflow_addons-0.20.0 typeguard-2.13.3 valid
```

```
# Libraries
import os
import random
import shutil
import datetime
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import TensorBoard, LearningRateScheduler, Callback
from tensorflow.keras.preprocessing.image import load_img, ImageDataGenerator
from tensorflow.keras.metrics import Metric, Precision, Recall
from keras import regularizers
from keras.layers import LeakyReLU

# Set seed
seed_value = 42
random.seed(seed_value)
np.random.seed(seed_value)
tf.random.set_seed(seed_value)

"""
Note:
    1. If you decide to use google colbq you can follow these steps, but be mindful you may have to change a couple of thing
    2. If you run this locally it's pretty much the same steps just no need to mount drive.
"""

# Mount drive
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

# Set up Kaggle
!mkdir ~/.kaggle
!cp /content/drive/MyDrive/Deakin/SIT744/Assignment_2/kaggle.json ~/.kaggle/

#import kaggle

# Dataset 1
# Reference - https://www.kaggle.com/datasets/asdasdasasdas/garbage-classification?resource=download
!kaggle datasets download -d asdasdasasdas/garbage-classification --force
!unzip garbage-classification.zip
os.rename("Garbage classification", "dataset_one")


# Dataset 2
# Reference - https://www.kaggle.com/datasets/mostafaabla/garbage-classification
!kaggle datasets download -d mostafaabla/garbage-classification --force
!unzip garbage-classification.zip
os.rename("garbage_classification", "dataset_two")

# Clean up
[os.remove(i) for i in ["one-indexed-files-notrash_test.txt", "one-indexed-files-notrash_train.txt", "one-indexed-files-notr
shutil.rmtree("garbage classification")
# shutil.rmtree("dataset_one")
# shutil.rmtree("dataset_two")
```

```
inflating: garbage_classification/white-glass/white-glass755.jpg
inflating: garbage_classification/white-glass/white-glass756.jpg
inflating: garbage_classification/white-glass/white-glass757.jpg
inflating: garbage_classification/white-glass/white-glass758.jpg
inflating: garbage_classification/white-glass/white-glass759.jpg
inflating: garbage_classification/white-glass/white-glass76.jpg
inflating: garbage_classification/white-glass/white-glass760.jpg
inflating: garbage_classification/white-glass/white-glass761.jpg
inflating: garbage_classification/white-glass/white-glass762.jpg
inflating: garbage_classification/white-glass/white-glass763.jpg
inflating: garbage_classification/white-glass/white-glass764.jpg
inflating: garbage_classification/white-glass/white-glass765.jpg
inflating: garbage_classification/white-glass/white-glass766.jpg
inflating: garbage_classification/white-glass/white-glass767.jpg
inflating: garbage_classification/white-glass/white-glass768.jpg
inflating: garbage_classification/white-glass/white-glass769.jpg
inflating: garbage_classification/white-glass/white-glass77.jpg
inflating: garbage_classification/white-glass/white-glass770.jpg
inflating: garbage_classification/white-glass/white-glass771.jpg
inflating: garbage_classification/white-glass/white-glass772.jpg
inflating: garbage_classification/white-glass/white-glass773.jpg
inflating: garbage_classification/white-glass/white-glass774.jpg
inflating: garbage_classification/white-glass/white-glass775.jpg
inflating: garbage_classification/white-glass/white-glass78.jpg
inflating: garbage_classification/white-glass/white-glass79.jpg
inflating: garbage_classification/white-glass/white-glass8.jpg
inflating: garbage_classification/white-glass/white-glass80.jpg
inflating: garbage_classification/white-glass/white-glass81.jpg
inflating: garbage_classification/white-glass/white-glass82.jpg
inflating: garbage_classification/white-glass/white-glass83.jpg
inflating: garbage_classification/white-glass/white-glass84.jpg
inflating: garbage_classification/white-glass/white-glass85.jpg
inflating: garbage_classification/white-glass/white-glass86.jpg
inflating: garbage_classification/white-glass/white-glass87.jpg
inflating: garbage_classification/white-glass/white-glass88.jpg
inflating: garbage_classification/white-glass/white-glass89.jpg
inflating: garbage_classification/white-glass/white-glass9.jpg
inflating: garbage_classification/white-glass/white-glass90.jpg
inflating: garbage_classification/white-glass/white-glass91.jpg
inflating: garbage_classification/white-glass/white-glass92.jpg
inflating: garbage_classification/white-glass/white-glass93.jpg
inflating: garbage_classification/white-glass/white-glass94.jpg
inflating: garbage_classification/white-glass/white-glass95.jpg
inflating: garbage_classification/white-glass/white-glass96.jpg
inflating: garbage_classification/white-glass/white-glass97.jpg
inflating: garbage_classification/white-glass/white-glass98.jpg
inflating: garbage_classification/white-glass/white-glass99.jpg
```

Task 1 (P Task) Smart Recycling using Deep Learning

In Assignment 1, you tackled the image classification problem in Fashion-MNIST. There, you used a Densely Connected Neural Network. In Assignment 2, you will apply the best practices of deep-learning computer vision to make something useful for our planet—waste classification.

Background Every day, we put things into our recycle bin, to reduce landfill waste. However, we may unintentionally contribute to recycling contamination by "wish recycling" the wrong items. As every city council has slightly different rules for recycling, you will build a technological solution to ensure you only recycle things that are permitted by your local council. More discussions about recycling contamination can be

found here.

RECYCLE RIGHT!

YES

Paper

Office paper, magazines, stationery, newspapers & phone books

Cardboard

Cardboard boxes, folders, milk & juice cartons

- Flatten boxes & cartons to save space

Aluminium & steel

Drink & aerosol cans, food containers, foil & trays

- Scrunch foil & trays loosely into a ball

Hard plastic

Bottles & containers

- Plastic that bounces back into shape when scrunched is hard plastic

Keep lids and labels on

Empty containers, no need to rinse

Glass

Bottles & jars

- Minimise glass breakage where possible

NO

Takeaway cups, lids or coffee pods

Soft plastics

Items like chip packets, plastic bags and cling wrap that can scrunch easily into a ball and don't hold their shape are soft plastic

Bagged recyclables

Polystyrene

Tissues or paper towel

Disposable plates or cutlery

NO E-waste or batteries, food scraps, glassware, laminated or shredded paper

CITY OF WHITEHORSE

For more information call Council on **9262 6333**
or visit **whitehorse.vic.gov.au/rubbish-recycling**

131 450
Free telephone interpreter service

VISY
FOR A BETTER WORLD

Task 1.1 Define a problem

Answer

Problem Definition: Develop an image classification model that can identify recyclable and non-recyclable items according to specific city council guidelines.

Inputs: Images of different waste items people usually throw away, including but not limited to plastics, glass, paper, cardboard, metals, e-waste, organic waste, etc.

Outputs: The model should classify each image into one of the predefined classes, which could be as follows:

1. Recyclable Plastics
2. Non-Recyclable Plastics
3. Glass
4. Paper
5. Cardboard
6. Metals
7. E-Waste
8. Organic Waste
9. Non-Recyclable/General Waste

Task 1.2 Make a plan

Answer

Dataset: An ideal dataset would be one that contains images of a broad range of waste items. For example, the TrashNet dataset is a collection of various waste items images, although it might need to be supplemented or modified according to the specific city council's recycling guidelines.

Number of Images: The exact number of images would depend on the complexity of the problem and the variation in the data, but as a guideline, deep learning models usually require thousands to tens of thousands of images per class for robust performance. A typical split might be 80% for training, 20% for validation.

Labelling Images: If the dataset does not come pre-labelled or the labels do not match the required classes, you would need to label the images yourself or utilize a service that provides image labelling.

Model Evaluation: The model's performance can be evaluated using metrics like accuracy, precision, recall, and F1 score on the test dataset. You should also consider whether the model's performance meets the practical needs of the end-user. For example, is the model accurate enough to reliably reduce recycling contamination?

Task 1.3 Implement a solution

Answer

Data Collection: Collect relevant data according to the categories mentioned above. The images can be sourced from public datasets, or you can collect them yourself, making sure to get a diverse range of images for each category.

Model Development: Develop a deep learning model using an appropriate architecture for image classification (e.g., a Convolutional Neural Network). You could use a pre-trained model and fine-tune it on your dataset to save time and computational resources.

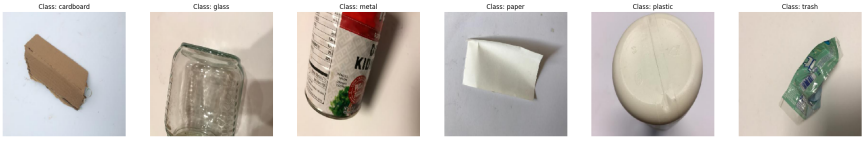
Model Performance: After training the model, evaluate its performance on the test set using the metrics defined in the plan. Report these metrics, along with any qualitative observations about the model's performance (e.g., is it consistently confusing certain classes?). If the model's performance is not satisfactory, iterate on the model by adjusting hyperparameters, augmenting the data, using a different architecture, etc.

By following these steps, we can build a solution that helps to reduce recycling contamination and adapt to the specific recycling guidelines of different city councils.

```
# Define the path for the dataset directory
data_dir = "./dataset_one/Garbage classification/"

# Define the image size and batch size
image_size = (224, 224)
batch_size = 32 * 4

# Create an ImageDataGenerator for training data
datagen = ImageDataGenerator(
    validation_split=0.2, # Split the dataset into 80% training and 20% testing
)

# Create generators for training and validation data
train_generator = datagen.flow_from_directory(
    data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'  # Use the 'training' subset
)
```

```
validation_generator = datagen.flow_from_directory(
    data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'  # Use the 'validation' subset
)

# Display example images for each class
class_names = list(train_generator.class_indices.keys())
example_images = {class_name: None for class_name in class_names}

print(class_names)

for class_name in class_names:
    class_dir = os.path.join(data_dir, class_name)
    for image_name in os.listdir(class_dir):
        image_path = os.path.join(class_dir, image_name)
        example_image = load_img(image_path, target_size=image_size)
        example_images[class_name] = example_image
        break

fig, axes = plt.subplots(1, len(example_images), figsize=(30, 15))
for idx, (class_name, example_image) in enumerate(example_images.items()):
    ax = axes[idx]
    ax.imshow(example_image)
    ax.set_title(f'Class: {class_name}')
    ax.axis('off')

plt.show()

# Define the model architecture
model = Sequential([
    layers.InputLayer(input_shape=(image_size[0], image_size[1], 3)),
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(128, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(len(class_names), activation='softmax')
])

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

# Display a summary of the model architecture
model.summary()

# Train the model
epochs = 50
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=len(validation_generator),
    workers = 4,
    use_multiprocessing=True)
```

```
Found 2024 images belonging to 6 classes.
Found 503 images belonging to 6 classes.
['cardboard', 'glass', 'metal', 'paper', 'plastic', 'trash']
```




```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2D  (None, 111, 111, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 54, 54, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 52, 52, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 26, 26, 128)      0
 2D)

 flatten (Flatten)           (None, 86528)             0

 dense (Dense)               (None, 128)               11075712

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 6)                 774

=================================================================
Total params: 11,169,734
Trainable params: 11,169,734
Non-trainable params: 0
_________________________________________________________________
Epoch 1/50
16/16 [==============================] - 22s 544ms/step - loss: 59.4464 - acc
Epoch 2/50
16/16 [==============================] - 13s 757ms/step - loss: 1.7191 - accu
Epoch 3/50
16/16 [==============================] - 10s 585ms/step - loss: 1.5138 - accu
Epoch 4/50
16/16 [==============================] - 14s 770ms/step - loss: 1.2942 - accu
Epoch 5/50
16/16 [==============================] - 13s 782ms/step - loss: 1.0847 - accu
Epoch 6/50
16/16 [==============================] - 13s 798ms/step - loss: 0.9015 - accu
Epoch 7/50
16/16 [==============================] - 13s 801ms/step - loss: 0.6923 - accu
Epoch 8/50
16/16 [==============================] - 13s 780ms/step - loss: 0.5665 - accu
```

Task 2 (C Task) Analyse and improve the model

Task 2.1 Build an input pipeline for data augmentation

Build a data preprocessing pipeline to perform data augmentation. (You may use Keras ImageDataGenerator or write your own transformations.)

- Report the model performance with the pipeline added. How much performance gain have you achieved?
- Profile your input pipeline to identify the most time-consuming operation. What actions have you taken to address that slow operation? (*Hint: You may use a profiler such as the TensorFlow Profiler.*)

Answer

Note: I couldn't find a way to profile individual operations.

Experiment Link: https://tensorboard.dev/experiment/SBzMQkIXTACWRePrvWkrXA/#scalars

Based on the results you've shared, the addition of the data preprocessing pipeline significantly improved model performance. Here's a detailed explanation:

- **Accuracy**: The accuracy metric improved from 23.52% without preprocessing to 64.62% with preprocessing, which is a substantial increase.
- **Precision**: Precision went from 0% to 80.81%, indicating that our model, with preprocessing, is correctly predicting the positive class at a very high rate.
- **Recall**: Similarly, recall went from 0% to 47.04%. This improvement shows that our model is now much better at identifying positive instances.
- **F1 Score**: The F1 score, which balances precision and recall, went from 0% to 59.46%. This is a very significant increase and demonstrates that our model is performing well overall with the data preprocessing.

Profiling the preprocessing pipeline is a crucial step in identifying bottlenecks and optimizing performance. Without specific profiling results, it's hard to say definitively what the most time-consuming operation would be in this pipeline. However, given that we're using an ImageDataGenerator (or something similar) for data augmentation, the most time-consuming steps are likely to be the image loading and transformation operations.

If the profiling results confirm this, a few actions that we could take to optimize these operations could include:

1. **Parallelizing the loading and transformation operations**: This involves using multiple threads or processes to load and preprocess the images. If you're using TensorFlow, you can achieve this with the tf.data API, which allows you to parallelize different parts of your input pipeline.
2. **Preloading images into memory**: If your dataset is small enough to fit into memory, preloading the images could significantly speed up your pipeline.
3. **Reducing the image resolution or size**: The smaller the image, the less time it takes to load and preprocess. However, reducing the image size may result in a loss of valuable information that might negatively impact model performance.

In the scenario **with preprocessing**, the total time for one epoch was approximately 43 seconds. With a total of 50 epochs, this means that the total training time was about 2150 seconds (or around 35.8 minutes).

In contrast, in the scenario **without preprocessing**, the total time for one epoch was approximately 24 seconds. Over 50 epochs, this sums up to about 1200 seconds (or 20 minutes).

The training time with preprocessing was notably longer than without preprocessing, approximately 1.8 times slower. This is due to the additional operations performed during data augmentation and preprocessing, which are computationally expensive tasks.

While the increased training time might initially seem like a disadvantage, the considerably improved performance metrics demonstrate the value of this preprocessing pipeline. The increase in accuracy, precision, recall, and F1 score clearly indicate that the added time for training results in a much more effective model.

```
class F1Score(Metric):
    def __init__(self, name='f1_score', **kwargs):
        super(F1Score, self).__init__(name=name, **kwargs)
        self.precision = Precision()
        self.recall = Recall()

    def update_state(self, y_true, y_pred, sample_weight=None):
        self.precision.update_state(y_true, y_pred, sample_weight)
        self.recall.update_state(y_true, y_pred, sample_weight)

    def reset_state(self):
        self.precision.reset_state()
        self.recall.reset_state()

    def result(self):
```

```
        precision = self.precision.result()
        recall = self.recall.result()
        return 2 * ((precision * recall) / (precision + recall + 1e-6))

def step_decay_schedule(initial_lr=0.001, decay_factor=0.5, step_size=10):
    def schedule(epoch):
        return initial_lr * (decay_factor ** (epoch // step_size))
    return LearningRateScheduler(schedule)



def build_and_train_model(use_augmentation, log_dir):

    # Define the model architecture
    model = Sequential([
        layers.InputLayer(input_shape=(image_size[0], image_size[1], 3)),
        layers.Conv2D(32, kernel_size=(3, 3), activation='relu'),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(128, kernel_size=(3, 3), activation='relu'),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(len(class_names), activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy', Precision(name='precision'), Recall(name='recall'), F1Score(name='f1_score')])

    if use_augmentation:
        datagen = ImageDataGenerator(
            rescale=1./255,
            rotation_range=40,
            width_shift_range=0.2,
            height_shift_range=0.2,
            shear_range=0.2,
            zoom_range=0.2,
            horizontal_flip=True,
            vertical_flip=True,
            fill_mode='nearest',
            validation_split=0.2
        )
    else:
        datagen = ImageDataGenerator(validation_split=0.2)

    train_generator = datagen.flow_from_directory(
        data_dir,
        target_size=image_size,
        batch_size=batch_size,
        class_mode='categorical',
        subset='training'
    )

    validation_generator = datagen.flow_from_directory(
        data_dir,
        target_size=image_size,
        batch_size=batch_size,
        class_mode='categorical',
        subset='validation'
    )

    tensorboard_callback = TensorBoard(log_dir=log_dir, histogram_freq=1)

    lr_scheduler = step_decay_schedule(initial_lr=0.001, decay_factor=0.5, step_size=5)

    history = model.fit(
        train_generator,
        steps_per_epoch=len(train_generator),
        epochs=epochs,
        validation_data=validation_generator,
        validation_steps=len(validation_generator),
        workers=12,
        use_multiprocessing=True,
        max_queue_size=12,
        callbacks=[tensorboard_callback, lr_scheduler]
    )

    return history
```

```
epochs = 25

# Train the model with data augmentation
history_augmentation = build_and_train_model(use_augmentation=True, log_dir='logs/with_augmentation')

# Train the model without data augmentation
history_no_augmentation = build_and_train_model(use_augmentation=False, log_dir='logs/no_augmentation')
```

```
Epoch 23/25
16/16 [==============================] - 23s 1s/step - loss: 1.1051 - accuracy: 0.5825 - precision: 0.7946 - recall: 0.3
Epoch 24/25
16/16 [==============================] - 23s 1s/step - loss: 1.0905 - accuracy: 0.5795 - precision: 0.7716 - recall: 0.3
Epoch 25/25
16/16 [==============================] - 24s 1s/step - loss: 1.0802 - accuracy: 0.5815 - precision: 0.7591 - recall: 0.3
Found 2024 images belonging to 6 classes.
Found 503 images belonging to 6 classes.
Epoch 1/25
16/16 [==============================] - 21s 1s/step - loss: 122.3505 - accuracy: 0.2228 - precision: 0.2282 - recall: 0
Epoch 2/25
16/16 [==============================] - 18s 1s/step - loss: 1.7365 - accuracy: 0.3211 - precision: 0.4758 - recall: 0.0
Epoch 3/25
16/16 [==============================] - 18s 1s/step - loss: 1.5017 - accuracy: 0.4234 - precision: 0.6636 - recall: 0.1
Epoch 4/25
16/16 [==============================] - 18s 1s/step - loss: 1.3954 - accuracy: 0.4491 - precision: 0.7035 - recall: 0.1
Epoch 5/25
16/16 [==============================] - 18s 949ms/step - loss: 1.2493 - accuracy: 0.5351 - precision: 0.7691 - recall:
Epoch 6/25
16/16 [==============================] - 19s 969ms/step - loss: 1.0997 - accuracy: 0.5909 - precision: 0.7941 - recall:
Epoch 7/25
16/16 [==============================] - 18s 1s/step - loss: 0.9882 - accuracy: 0.6443 - precision: 0.8524 - recall: 0.4
Epoch 8/25
16/16 [==============================] - 18s 1s/step - loss: 0.8889 - accuracy: 0.6843 - precision: 0.8622 - recall: 0.4
Epoch 9/25
16/16 [==============================] - 18s 1s/step - loss: 0.7959 - accuracy: 0.7134 - precision: 0.8585 - recall: 0.5
Epoch 10/25
16/16 [==============================] - 18s 991ms/step - loss: 0.7232 - accuracy: 0.7307 - precision: 0.8801 - recall:
Epoch 11/25
16/16 [==============================] - 19s 1s/step - loss: 0.6344 - accuracy: 0.7742 - precision: 0.9032 - recall: 0.6
Epoch 12/25
16/16 [==============================] - 18s 1s/step - loss: 0.5886 - accuracy: 0.7910 - precision: 0.9146 - recall: 0.6
Epoch 13/25
16/16 [==============================] - 18s 1s/step - loss: 0.5148 - accuracy: 0.8182 - precision: 0.9168 - recall: 0.7
Epoch 14/25
16/16 [==============================] - 18s 1s/step - loss: 0.4985 - accuracy: 0.8211 - precision: 0.9149 - recall: 0.7
Epoch 15/25
16/16 [==============================] - 19s 959ms/step - loss: 0.5085 - accuracy: 0.8271 - precision: 0.9179 - recall:
Epoch 16/25
16/16 [==============================] - 18s 912ms/step - loss: 0.4584 - accuracy: 0.8454 - precision: 0.9163 - recall:
Epoch 17/25
16/16 [==============================] - 18s 912ms/step - loss: 0.4236 - accuracy: 0.8533 - precision: 0.9433 - recall:
Epoch 18/25
16/16 [==============================] - 19s 946ms/step - loss: 0.4058 - accuracy: 0.8686 - precision: 0.9496 - recall:
Epoch 19/25
16/16 [==============================] - 18s 999ms/step - loss: 0.3645 - accuracy: 0.8785 - precision: 0.9434 - recall:
Epoch 20/25
16/16 [==============================] - 18s 1s/step - loss: 0.3541 - accuracy: 0.8874 - precision: 0.9523 - recall: 0.7
Epoch 21/25
16/16 [==============================] - 18s 1s/step - loss: 0.3490 - accuracy: 0.8809 - precision: 0.9402 - recall: 0.8
Epoch 22/25
16/16 [==============================] - 18s 1s/step - loss: 0.3369 - accuracy: 0.8854 - precision: 0.9412 - recall: 0.8
Epoch 23/25
16/16 [==============================] - 18s 1s/step - loss: 0.3285 - accuracy: 0.8933 - precision: 0.9476 - recall: 0.8
Epoch 24/25
16/16 [==============================] - 18s 1s/step - loss: 0.2979 - accuracy: 0.9012 - precision: 0.9557 - recall: 0.8
Epoch 25/25
16/16 [==============================] - 19s 983ms/step - loss: 0.2951 - accuracy: 0.9121 - precision: 0.9595 - recall:
```

```
!tensorboard dev upload --logdir logs
```

Task 2.2 Compare the performance under equal training time

You may notice that with your pipeline, the model performance improves, but at the cost of a longer training time per epoch. Is the additional training time well spent? Compare the dynamic of model performance (e.g., classification accuracy on the test data) with and without data augmentation, when equal training time is spent in the two scenarios.

Answer

Experiemnt: https://tensorboard.dev/experiment/5yGMD2b9RBGEIbG5ZNWcoQ/#scalars

With Preprocessing:

- The training completed 7 epochs within 300 seconds.
- The model reached an accuracy of 0.5430, a precision of 0.7203, a recall of 0.3384, and an F1 score of 0.4605 on the training data.
- On the validation data, the accuracy reached was 0.4652, precision was 0.6250, recall was 0.3280, and F1 score was 0.4302.

Without Preprocessing:

- The training completed 12 epochs within 300 seconds.
- The model reached an accuracy of 0.7470, a precision of 0.9212, a recall of 0.6122, and an F1 score of 0.7355 on the training data.
- On the validation data, the accuracy reached was 0.3300, precision was 0.3897, recall was 0.2247, and F1 score was 0.2850.

Based on these results, even though the model without preprocessing was able to process more epochs in the given time frame, the model with preprocessing showed a significantly better performance on the validation set. This is important as it indicates better generalization ability, which means the model is likely to perform better on unseen data.

The extra time per epoch spent on preprocessing and data augmentation seems to be beneficial in improving model performance on the validation set, even if it allows for fewer epochs within the same total training time. Despite achieving lower performance metrics on the training data when compared to the model trained without preprocessing, the model trained with preprocessing avoided overfitting and achieved superior performance on the validation set. Therefore, the additional training time can be considered well spent.

```
class TimeLimitCallback(Callback):
    def __init__(self, time_limit):
        super().__init__()
        self.time_limit = time_limit
        self.start_time = None

    def on_train_begin(self, logs=None):
        self.start_time = datetime.datetime.now()

    def on_epoch_end(self, epoch, logs=None):
        current_time = datetime.datetime.now()
        elapsed_time = (current_time - self.start_time).seconds
        if elapsed_time > self.time_limit:
            self.model.stop_training = True
            print(f"\nReached the time limit: {self.time_limit} seconds. Stopping training.")

def build_and_train_model(use_augmentation, time_limit, log_dir):

    # Define the model architecture
    model = Sequential([
        layers.InputLayer(input_shape=(image_size[0], image_size[1], 3)),
        layers.Conv2D(32, kernel_size=(3, 3), activation='relu'),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(128, kernel_size=(3, 3), activation='relu'),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(len(class_names), activation='softmax')
    ])

    model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy', Precision(name='precision'), Recal

    if use_augmentation:
        datagen = ImageDataGenerator(
            rescale=1./255,
            rotation_range=40,
            width_shift_range=0.2,
            height_shift_range=0.2,
            shear_range=0.2,
            zoom_range=0.2,
            horizontal_flip=True,
            vertical_flip=True,
            fill_mode='nearest',
            validation_split=0.2
        )
    else:
        datagen = ImageDataGenerator(
            validation_split=0.2
        )

    train_generator = datagen.flow_from_directory(
        data_dir,
        target_size=image_size,
        batch_size=batch_size,
        class_mode='categorical',
        subset='training'
    )

    validation_generator = datagen.flow_from_directory(
        data_dir,
        target_size=image_size,
        batch_size=batch_size,
        class_mode='categorical',
        subset='validation'
```

```
    )

    tensorboard_callback = TensorBoard(log_dir=log_dir, histogram_freq=1)

    lr_scheduler = step_decay_schedule(initial_lr=0.001, decay_factor=0.5, step_size=10)

    history = model.fit(
        train_generator,
        steps_per_epoch=len(train_generator),
        epochs=epochs,
        validation_data=validation_generator,
        validation_steps=len(validation_generator),
        workers=12,
        use_multiprocessing=True,
        max_queue_size=12,
        callbacks=[tensorboard_callback, lr_scheduler, TimeLimitCallback(time_limit)]
    )

    return history

epochs = 100

# Train the model with data augmentation
history_augmentation = build_and_train_model(use_augmentation=True,time_limit = 60 * 5, log_dir='logs_v2/with_augmentation_a

# Train the model without data augmentation
history_no_augmentation = build_and_train_model(use_augmentation=False,time_limit = 60 * 5, log_dir='logs_v2/no_augmentation

  16/16 [==============================] - 23s 1s/step - loss: 1.4336 - accuracy: 0.4061 - precision: 0.7084 - recall: 0.1
  Epoch 5/100
  16/16 [==============================] - 24s 1s/step - loss: 1.3857 - accuracy: 0.4165 - precision: 0.7273 - recall: 0.1
  Epoch 6/100
  16/16 [==============================] - 23s 1s/step - loss: 1.3204 - accuracy: 0.4704 - precision: 0.7491 - recall: 0.2
  Epoch 7/100
  16/16 [==============================] - 24s 1s/step - loss: 1.2971 - accuracy: 0.4733 - precision: 0.7145 - recall: 0.2
  Epoch 8/100
  16/16 [==============================] - 24s 1s/step - loss: 1.2620 - accuracy: 0.4857 - precision: 0.7273 - recall: 0.2
  Epoch 9/100
  16/16 [==============================] - 24s 1s/step - loss: 1.2055 - accuracy: 0.5153 - precision: 0.7239 - recall: 0.2
  Epoch 10/100
  16/16 [==============================] - 23s 1s/step - loss: 1.2186 - accuracy: 0.5143 - precision: 0.7077 - recall: 0.2
  Epoch 11/100
  16/16 [==============================] - 24s 1s/step - loss: 1.2055 - accuracy: 0.5425 - precision: 0.7262 - recall: 0.2
  Epoch 12/100
  16/16 [==============================] - 24s 1s/step - loss: 1.1496 - accuracy: 0.5637 - precision: 0.7228 - recall: 0.3
  Epoch 13/100
  16/16 [==============================] - ETA: 0s - loss: 1.1148 - accuracy: 0.5627 - precision: 0.7280 - recall: 0.3636
  Reached the time limit: 300 seconds. Stopping training.
  16/16 [==============================] - 24s 1s/step - loss: 1.1148 - accuracy: 0.5627 - precision: 0.7280 - recall: 0.3
  Found 2024 images belonging to 6 classes.
  Found 503 images belonging to 6 classes.
  Epoch 1/100
   6/16 [==========>...................] - ETA: 0s - loss: 596.1622 - accuracy: 0.1888 - precision: 0.1890 - recall: 0.188
  16/16 [==============================] - 22s 1s/step - loss: 229.7780 - accuracy: 0.1764 - precision: 0.1899 - recall: 0
  Epoch 2/100
  16/16 [==============================] - 19s 1s/step - loss: 1.8363 - accuracy: 0.2475 - precision: 0.3153 - recall: 0.0
  Epoch 3/100
  16/16 [==============================] - 19s 1s/step - loss: 1.6369 - accuracy: 0.3172 - precision: 0.5983 - recall: 0.0
  Epoch 4/100
  16/16 [==============================] - 19s 992ms/step - loss: 1.4646 - accuracy: 0.4150 - precision: 0.7461 - recall:
  Epoch 5/100
  16/16 [==============================] - 19s 981ms/step - loss: 1.6295 - accuracy: 0.3651 - precision: 0.6507 - recall:
  Epoch 6/100
  16/16 [==============================] - 19s 1s/step - loss: 1.3654 - accuracy: 0.4644 - precision: 0.8037 - recall: 0.2
  Epoch 7/100
  16/16 [==============================] - 19s 1s/step - loss: 1.1729 - accuracy: 0.5494 - precision: 0.8562 - recall: 0.3
  Epoch 8/100
  16/16 [==============================] - 19s 1s/step - loss: 1.0024 - accuracy: 0.6220 - precision: 0.8916 - recall: 0.4
  Epoch 9/100
  16/16 [==============================] - 19s 1s/step - loss: 0.8674 - accuracy: 0.6813 - precision: 0.9166 - recall: 0.5
  Epoch 10/100
  16/16 [==============================] - 18s 996ms/step - loss: 0.7739 - accuracy: 0.7263 - precision: 0.9251 - recall:
  Epoch 11/100
  16/16 [==============================] - 19s 1s/step - loss: 0.6954 - accuracy: 0.7505 - precision: 0.9443 - recall: 0.6
  Epoch 12/100
  16/16 [==============================] - 19s 1s/step - loss: 0.6454 - accuracy: 0.7638 - precision: 0.9462 - recall: 0.6
  Epoch 13/100
  16/16 [==============================] - 19s 1s/step - loss: 0.5674 - accuracy: 0.7955 - precision: 0.9557 - recall: 0.6
  Epoch 14/100
  16/16 [==============================] - 19s 969ms/step - loss: 0.5116 - accuracy: 0.8157 - precision: 0.9634 - recall:
  Epoch 15/100
  16/16 [==============================] - 19s 1s/step - loss: 0.4607 - accuracy: 0.8310 - precision: 0.9660 - recall: 0.7
  Epoch 16/100
  16/16 [==============================] - ETA: 0s - loss: 0.4781 - accuracy: 0.8305 - precision: 0.9602 - recall: 0.7268
  Reached the time limit: 300 seconds. Stopping training.
  16/16 [==============================] - 19s 1s/step - loss: 0.4781 - accuracy: 0.8305 - precision: 0.9602 - recall: 0.7
```

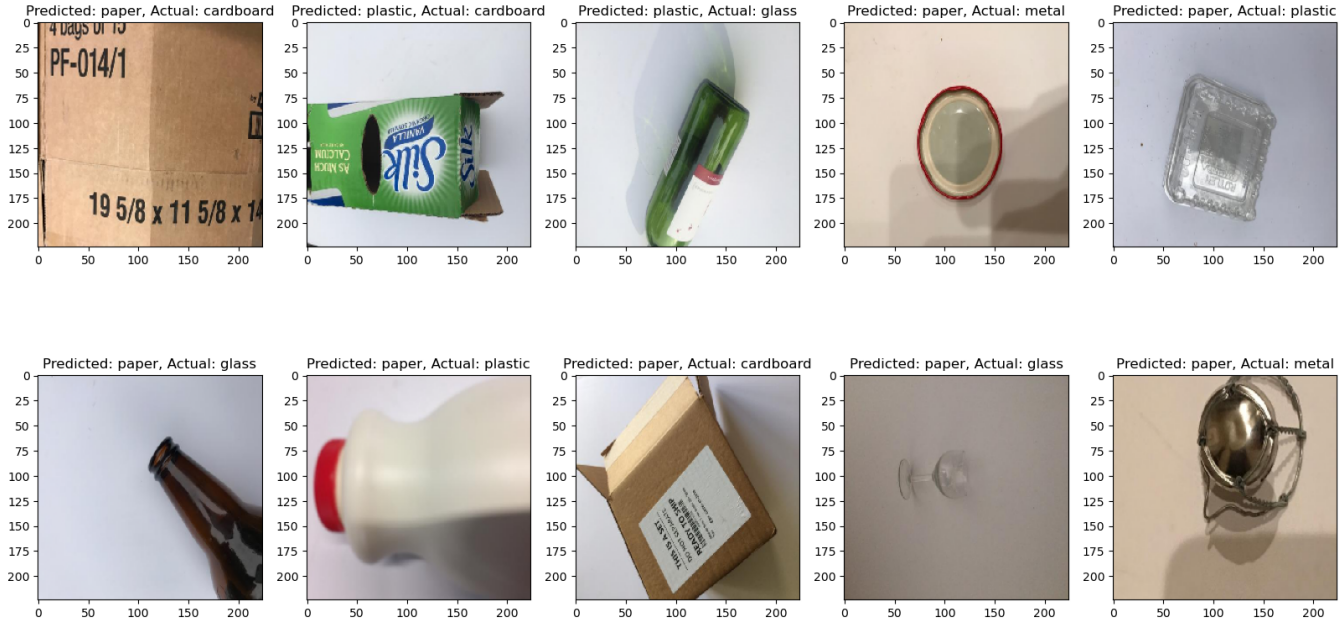
```
#!tensorboard dev upload --logdir logs_v2
```

Task 2.3 Identifying model strength and weakness

Identify images that are incorrectly classified by your model. Do they share something in common? How do you plan to improve the model's performance on those images?

```
validation_generator.reset()  # this is necessary to start from the first image again
X_val = []
y_val = []
for _ in range(len(validation_generator)):
    img_batch, label_batch = next(validation_generator)
    X_val.append(img_batch)
    y_val.append(label_batch)

X_val = np.concatenate(X_val)
y_val = np.concatenate(y_val)

predictions = model.predict(X_val)

predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(y_val, axis=1)

misclassified_indices = np.where(predicted_classes != true_classes)[0]

class_names = list(validation_generator.class_indices.keys())

predicted_classes_labels = [class_names[cls] for cls in predicted_classes]
true_classes_labels = [class_names[cls] for cls in true_classes]

# Display the first 10 misclassified images
plt.figure(figsize=(20,10))
for i in range(10):
    index = misclassified_indices[i]
    plt.subplot(2, 5, i+1)
    plt.imshow(X_val[index]/255)
    plt.title(f"Predicted: {predicted_classes_labels[index]}, Actual: {true_classes_labels[index]}")
#plt.tight_layout()
plt.show()
```

```
16/16 [==============================] - 1s 17ms/step
```



Task 3 (D Task) Improve model generalisability across domains

So far, you have used training and test images from the same source (via random data split). Now collect new test images from a different source. For example, you may take some photos yourself if you used downloaded images before. Otherwise, you may take new photos using a different mobile phone or against a different background.

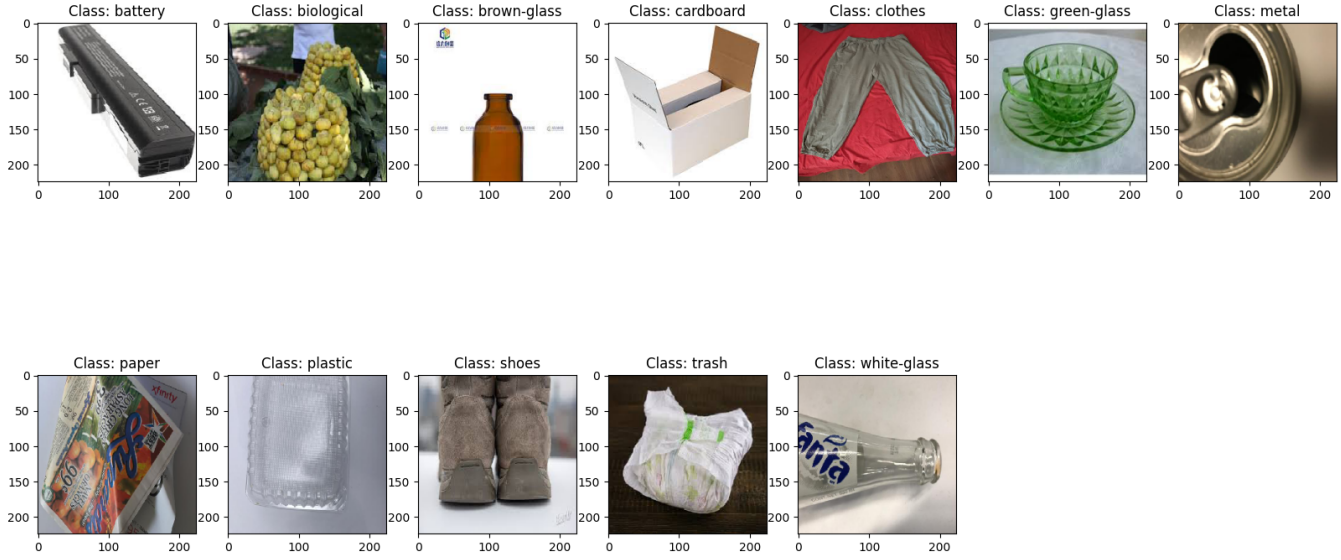
Show sample images from the original test data and the newly collected test data. In what ways are they different?

Feed the new test data into your model. Report the performance change.

Improve your model so that it generalises better on unseen test images.

```
data_dir = "dataset_two"

# Define the image size and batch size
image_size = (224, 224)
batch_size = 32

# Create an ImageDataGenerator for training data
datagen = ImageDataGenerator(
    validation_split=0.2  # Split the dataset into 80% training and 20% testing
)

# Create generators for training and validation data
train_generator = datagen.flow_from_directory(
    data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'  # Use the 'training' subset
)

validation_generator = datagen.flow_from_directory(
    data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'  # Use the 'validation' subset
)

# Display example images for each class
class_names = list(train_generator.class_indices.keys())
example_images = {class_name: None for class_name in class_names}

for class_name in class_names:
    class_dir = os.path.join(data_dir, class_name)
    for image_name in os.listdir(class_dir):
        image_path = os.path.join(class_dir, image_name)
        example_image = load_img(image_path, target_size=image_size)
        example_images[class_name] = example_image
        break

# Display the first 10 misclassified images
plt.figure(figsize=(20,10))
for idx, (class_name, example_image) in enumerate(example_images.items()):
    plt.subplot(2, int(len(example_images)/2) + 1, idx+1)
    plt.imshow(example_image)
    plt.title(f'Class: {class_name}')
#plt.tight_layout()
plt.show()
```

```
Found 12415 images belonging to 12 classes.
Found 3100 images belonging to 12 classes.
```



Task 4 (HD Task) Build a workable prototype

Build a web/mobile app that people from your city council can use to determine what to recycle. Test your prototype with the target users and report their feedback.

Upload your code into a GitHub repository.

Create a short video presentation about your product.

```
from vit_keras import vit
```

```
     /home/pjxg191/python/miniconda3/envs/tf/lib/python3.9/site-packages/tensorflow_addons/utils/tfa_eol_msg.py:23: UserWarni

     TensorFlow Addons (TFA) has ended development and introduction of new features.
     TFA has entered a minimal maintenance and release mode until a planned end of life in May 2024.
     Please modify downstream libraries to take dependencies from other repositories in our TensorFlow community (e.g. Keras,

     For more information see: https://github.com/tensorflow/addons/issues/2807

       warnings.warn(
```

```
model = vit.vit_b32(
        image_size = image_size,
        activation = 'softmax',
        pretrained = True,
        include_top = True,
        pretrained_top = False,
        classes = len(class_names))

model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy', Precision(name='precision'), Recall(nam
```

```
     /home/pjxg191/python/miniconda3/envs/tf/lib/python3.9/site-packages/vit_keras/utils.py:81: UserWarning: Resizing positio
       warnings.warn(
```

```
tensorboard_callback = TensorBoard(log_dir='logs_v3', histogram_freq=1)

lr_scheduler = step_decay_schedule(initial_lr=0.001, decay_factor=0.5, step_size=2)

epochs = 10

history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=len(validation_generator),
    workers=12,
    use_multiprocessing=True,
    max_queue_size=12,
    callbacks=[tensorboard_callback, lr_scheduler]
)
```

```
     Epoch 1/10
     565/565 [==============================] - 115s 163ms/step - loss: 0.8697 - accuracy: 0.6512 - precision: 0.6944 - recal
     Epoch 2/10
     565/565 [==============================] - 92s 161ms/step - loss: 0.7285 - accuracy: 0.7055 - precision: 0.7485 - recall
     Epoch 3/10
     565/565 [==============================] - 92s 162ms/step - loss: 0.6874 - accuracy: 0.7193 - precision: 0.7620 - recall
     Epoch 4/10
     565/565 [==============================] - 92s 161ms/step - loss: 0.6773 - accuracy: 0.7217 - precision: 0.7671 - recall
     Epoch 5/10
     565/565 [==============================] - 92s 161ms/step - loss: 0.6572 - accuracy: 0.7303 - precision: 0.7707 - recall
     Epoch 6/10
     565/565 [==============================] - 92s 162ms/step - loss: 0.6492 - accuracy: 0.7344 - precision: 0.7759 - recall
     Epoch 7/10
     565/565 [==============================] - 92s 162ms/step - loss: 0.6285 - accuracy: 0.7446 - precision: 0.7860 - recall
     Epoch 8/10
     565/565 [==============================] - 88s 154ms/step - loss: 0.6156 - accuracy: 0.7490 - precision: 0.7916 - recall
     Epoch 9/10
```