SIT744 Assignment 2: Deep Learning Project

+ Code + Text

Assignment objective

This assignment is to provide 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.

Task 1 (P Task) Smart Recycling using Deep Learning

In Assignment 1, you tackled a classification problem. 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 slight ly 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

Define an image classification problem that may help people better recycle, particularly by reducing contamination.

Describe the desired inputs and outputs, including the target classes.

Task 1.2 Make a plan

What dataset can you use to develop a deep learning solution?

How many images do you need? How many for training? How many for testing?

Do you need to label the images yourself?

How do you determine if your model is good enough?

Task 1.3 Implement a solution

Collect relevant data. Develop a deep learning model. Report the model performance against the success criteria that you define.

Task 1.1 Define a problem

The image classification problem I propose is a multiclass classification for a dataset of images of many types of garbages so that the model can identify what object is in the image (paper, cardboard, hard plastic,...) and it can detect whether is can be recycled or not. Therefore, it can help reduce contamination. The inputs are images of garbage and outputs are classes of garbages that are defined in the "Your Household Recycling and Waste Services Guide 2024" of WhiteHorse City Council:

1. Recycled

- Paper
- Cardboard
- Glass jars and bottle
- Aluminium and steel
- Plastic bottles and containers

2. Not Recycled

- Plastic bags, soft plastic packaging, food wrappers and clear film
- Takeaway cups and coffee pods
- Electronic items
- Batteries
- Foam and polystyrene
- Tissues, paper towels and napkins
- Nappies
- Plastic toys
- Drinking glass
- Clothing and textiles
- Building waste
- Garden waste
- Food scraps

Task 1.2 Make a plan

I scrape images by keyword from DuckDuckGo for each target class (100 - 200 for each class) so in total the dataset has nearly 1900 images and I choose ratio 60:40 for training, validation (training around 1106 images, validation around 737 images). The data got from DuckDuckGo by keywords has labels by themselves. The efficiency of the model is assessed by its accuracy level of the validation data and the model should not overfitting.

Task 1.3 Implement a solution

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Develop the model

Read data and develop the model

```
import tensorflow as tf
data_train, data_validation = tf.keras.preprocessing.image_dataset_from_directory(
    '/content/drive/MyDrive/Assignment2_Dataset/Part2_WasteImages',
    labels='inferred',
    label_mode='categorical',
    color_mode='rgb',
    batch_size=32,
    image_size=(256, 256),
    shuffle=True,
    seed=11,
    validation_split=0.3,  # Split the dataset into training and validation
    subset= 'both',
    interpolation='bilinear',
    follow_links=False,
    crop_to_aspect_ratio=False,

)
```

```
Found 1843 files belonging to 18 classes.
Using 1106 files for training.
Using 737 files for validation.
```

```
# Rescale pixel values to [0, 1]
data_train = data_train.map(lambda x, y: (x / 255.0, y))
data_validation = data_validation.map(lambda x, y: (x / 255.0, y))
```

Because the size of dataset is small so I will use a pre-trained model because the learned parameters of pre-trained models tend to be statistically stable and it would have “seen” more diverse inputs and generalises better. For this dataset, I will use the VGG16 model from keras.application. The model consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. VGG-16 is renowned for its simplicity and effectiveness, as well as its ability to achieve strong performance on various computer vision tasks, including image classification and object recognition. To use VGG16, I remove the original densely connected classifier and then add a new one (with weights random initialised). Using the new data, I train only the added classifier and freeze the convolution base to get consistent features. I choose the optimizer RMSprop learning_rate=2e-5 for the model to converge faster

```
from tensorflow.keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(256, 256, 3))
conv_base.trainable = False

conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 256, 256, 3)]     0

 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792

 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928

 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0

 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856

 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584

 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0

 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168

 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080
```

```
 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080

 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0

 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160

 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0

 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 0 (0.00 Byte)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import optimizers

def make_model(input_shape=(256, 256, 3), num_classes=18):
    # Define the base model (conv_base) or replace it with your desired pre-trained model
    # Replace this with your desired pre-trained model


    model = models.Sequential()
    model.add(conv_base)
    model.add(layers.Flatten())
    model.add(layers.Dense(256, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(num_classes, activation='softmax'))  # Use 'softmax' for multiclass classification

    model.compile(optimizer=optimizers.RMSprop(learning_rate=2e-5),
                  loss='categorical_crossentropy',  # Use 'categorical_crossentropy' for multiclass classification
                  metrics=['acc'])

    return model

model = make_model()
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 8, 8, 512)         14714688

 flatten_1 (Flatten)         (None, 32768)             0

 dense_2 (Dense)             (None, 256)               8388864

 dropout_1 (Dropout)         (None, 256)               0

 dense_3 (Dense)             (None, 18)                4626

=================================================================
Total params: 23108178 (88.15 MB)
Trainable params: 8393490 (32.02 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
data_train
```

```
<_MapDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None, 18),
dtype=tf.float32, name=None))>
```

```
#Fit the model
history = model.fit(
      data_train,
```

```
    epochs=30,
    validation_data=data_validation,

    )
```

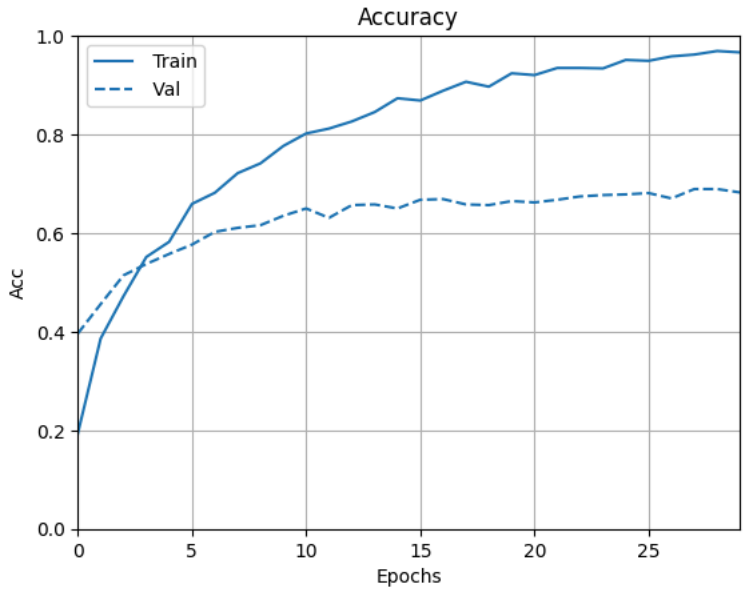
Show hidden output

```
pip install -q git+https://github.com/tensorflow/docs
```

```
  Preparing metadata (setup.py) ... done
  Building wheel for tensorflow-docs (setup.py) ... done
```

```
#Plot the accuracy
import tensorflow_docs as tfdocs
import tensorflow_docs.plots
import matplotlib.pyplot as plt

plotter = tfdocs.plots.HistoryPlotter()
plotter.plot({"": history}, metric = "acc")
plt.title("Accuracy")
plt.ylim([0,1])
```

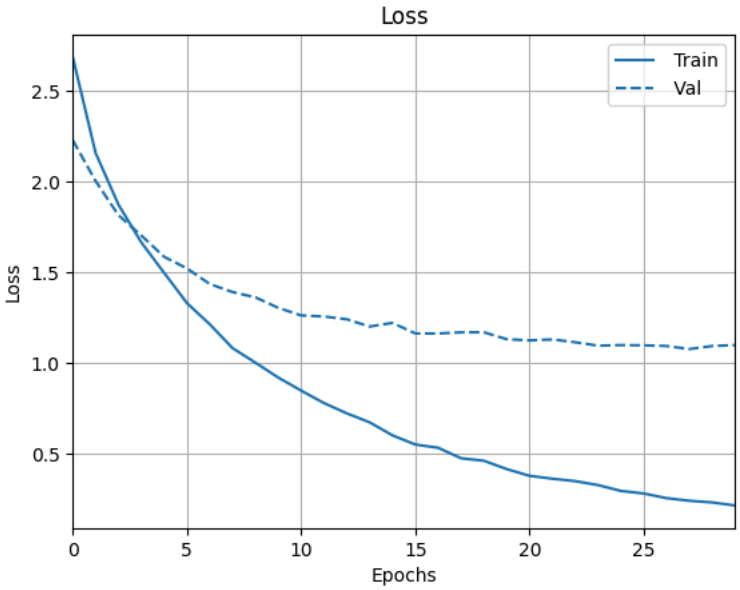
```
(0.0, 1.0)
```



Double-click (or enter) to edit

```
#Plot the loss
import tensorflow_docs

plotter = tfdocs.plots.HistoryPlotter()
plotter.plot({"": history}, metric = "loss")
plt.title("Loss")
```

```
Text(0.5, 1.0, 'Loss')
```



The accuracy of the validation data increases significantly during 30 epochs and reaches about 68% at the final one. This is a fairly good result. The loss and accuracy of the validation data tend to improve, likely indicating of no overfitting problem.

```
#save model
model.save('/content/drive/MyDrive/Models/model_VGG16.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `m
  saving_api.save_model(
```

Double-click (or enter) to edit

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.*)

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.

Task 2.3 Identifying model strengths and weaknesses

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?

Task 2.1 Build an input pipeline for data augmentation

```
#Develope the model
from tensorflow.keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
```

```
                  include_top=False,
                  input_shape=(256, 256, 3))
conv_base.trainable = False

conv_base.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.
58889256/58889256 [==============================] - 0s 0us/step
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0

 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792

 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928

 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0

 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856

 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584

 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0

 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168

 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080

 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0

 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160

 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0

 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 0 (0.00 Byte)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import optimizers

def make_model(input_shape=(256, 256, 3), num_classes=18):
    # Define the base model (conv_base) or replace it with your desired pre-trained model
    # Replace this with your desired pre-trained model


    model = models.Sequential()
    model.add(conv_base)
    model.add(layers.Flatten())
    model.add(layers.Dense(256, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(num_classes, activation='softmax'))  # Use 'softmax' for multiclass classification

    model.compile(optimizer=optimizers.RMSprop(lr=2e-5),
                  loss='categorical_crossentropy',  # Use 'categorical_crossentropy' for multiclass classification
                  metrics=['acc'])

    return model
```

```
model_augment = make_model()
model_augment.summary()
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.leg
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 8, 8, 512)         14714688

 flatten (Flatten)           (None, 32768)             0

 dense (Dense)               (None, 256)               8388864

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 18)                4626

=================================================================
Total params: 23108178 (88.15 MB)
Trainable params: 8393490 (32.02 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
#Create data augmentation flow with ImageDataGenerator
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=20,
                                   horizontal_flip=True,
                                   validation_split=0.2,
                                  )
```

```
val_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
```

```
train_data = train_datagen.flow_from_directory( '/content/drive/MyDrive/images',
                                               seed=10,
                                               target_size=(256, 256),
                                               color_mode='rgb',
                                               batch_size=32,
                                               class_mode='categorical',
                                               shuffle=True,
                                               subset = 'training')
```

```
Found 1480 images belonging to 18 classes.
```

```
train_data
```

```
<keras.src.preprocessing.image.DirectoryIterator at 0x78aeddd39fc0>
```

```
val_data = val_datagen.flow_from_directory('/content/drive/MyDrive/images',
                                           seed=10,
                                          target_size=(256, 256),
                                          color_mode='rgb',
                                          batch_size=32,
                                          class_mode='categorical',
                                          shuffle=False,
                                          subset = 'validation')
```

```
Found 363 images belonging to 18 classes.
```

```
# Training with augmented data
batch_size=32
import tensorflow as tf
!rm -rf ./logs/

from datetime import datetime
import os


root_logdir = "logs"
run_id = datetime.now().strftime("%Y%m%d-%H%M%S")
logdir = os.path.join(root_logdir, run_id)
```

```
# Profile from batches 1 to 30
tb_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir,histogram_freq = 1,profile_batch = '1,30')

# Train the model and use the TensorBoard Keras callback to collect
# performance profiling data
history_data_aug=model_augment.fit(train_data,
          steps_per_epoch = train_data.samples // batch_size,
          epochs=30,
         validation_data=val_data,
          callbacks=[tb_callback])
```

Show hidden output

```
model_augment.save(('/content/drive/MyDrive/Models/model_augment.h5'))
```

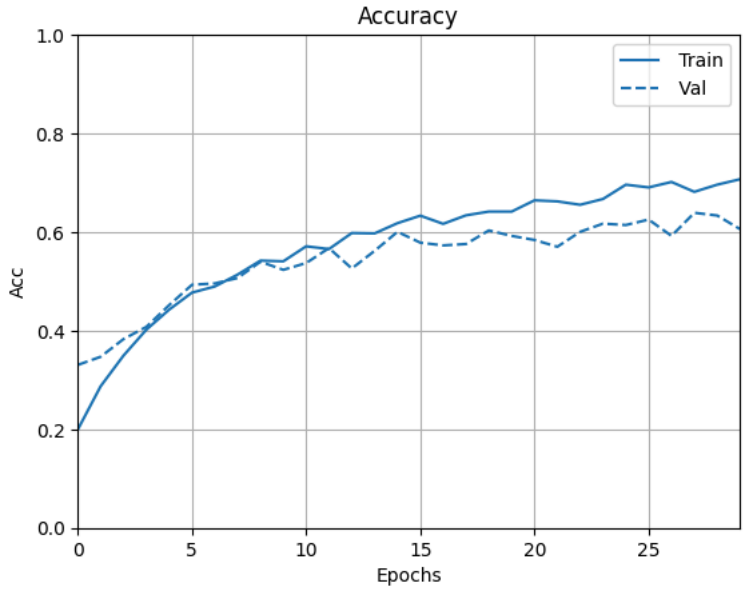
```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `m
  saving_api.save_model(
```

```
%pip install git+https://github.com/tensorflow/docs
```

```
Collecting git+https://github.com/tensorflow/docs
  Cloning https://github.com/tensorflow/docs to /tmp/pip-req-build-wh7y1ymp
  Running command git clone --filter=blob:none --quiet https://github.com/tensorflow/docs /tmp/pip-req-build-wh7y1ymp
  Resolved https://github.com/tensorflow/docs to commit eea8c521822d188d136083d9af4c58e9c381b756
  Preparing metadata (setup.py) ... done
Collecting astor (from tensorflow-docs==2024.5.3.31743)
  Downloading astor-0.8.1-py2.py3-none-any.whl (27 kB)
Requirement already satisfied: absl-py in /usr/local/lib/python3.10/dist-packages (from tensorflow-docs==2024.5.3.31743) (1.4.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from tensorflow-docs==2024.5.3.31743) (3.1.4)
Requirement already satisfied: nbformat in /usr/local/lib/python3.10/dist-packages (from tensorflow-docs==2024.5.3.31743) (5.10.4)
Requirement already satisfied: protobuf>=3.12 in /usr/local/lib/python3.10/dist-packages (from tensorflow-docs==2024.5.3.31743) (3.20.3)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from tensorflow-docs==2024.5.3.31743) (6.0.1)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->tensorflow-docs==2024.5.3.31743)
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.10/dist-packages (from nbformat->tensorflow-docs==2024.5.3
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/dist-packages (from nbformat->tensorflow-docs==2024.5.3.3174
Requirement already satisfied: jupyter-core!=5.0.*,>=4.12 in /usr/local/lib/python3.10/dist-packages (from nbformat->tensorflow-docs==20
Requirement already satisfied: traitlets>=5.1 in /usr/local/lib/python3.10/dist-packages (from nbformat->tensorflow-docs==2024.5.3.31743
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->tensorflow-docs
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nb
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->tensorflo
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->tensorflow-doc
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-packages (from jupyter-core!=5.0.*,>=4.12->nbformat->
Building wheels for collected packages: tensorflow-docs
  Building wheel for tensorflow-docs (setup.py) ... done
  Created wheel for tensorflow-docs: filename=tensorflow_docs-2024.5.3.31743-py3-none-any.whl size=182531 sha256=d48962664fe957b64529807
  Stored in directory: /tmp/pip-ephem-wheel-cache-7ibc_8w5/wheels/86/0f/1e/3b62293c8ffd0fd5a49508e6871cdb7554abe9c62afd35ec53
Successfully built tensorflow-docs
Installing collected packages: astor, tensorflow-docs
Successfully installed astor-0.8.1 tensorflow-docs-2024.5.3.31743
```

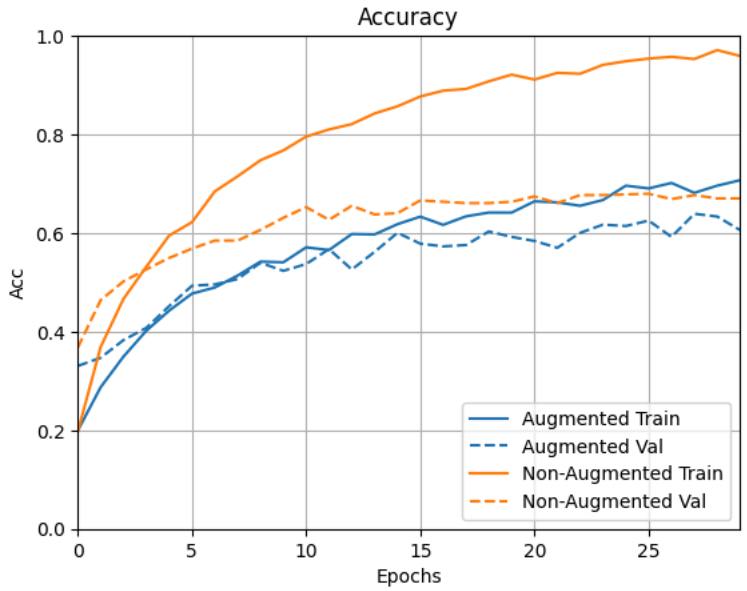
```
#Plot accuracy of train and validation data
import tensorflow_docs as tfdocs
import tensorflow_docs.plots
import matplotlib.pyplot as plt


plotter = tfdocs.plots.HistoryPlotter()
plotter.plot({"": history_data_aug}, metric = "acc")
plt.title("Accuracy")
plt.ylim([0,1])
```

```
(0.0, 1.0)
```



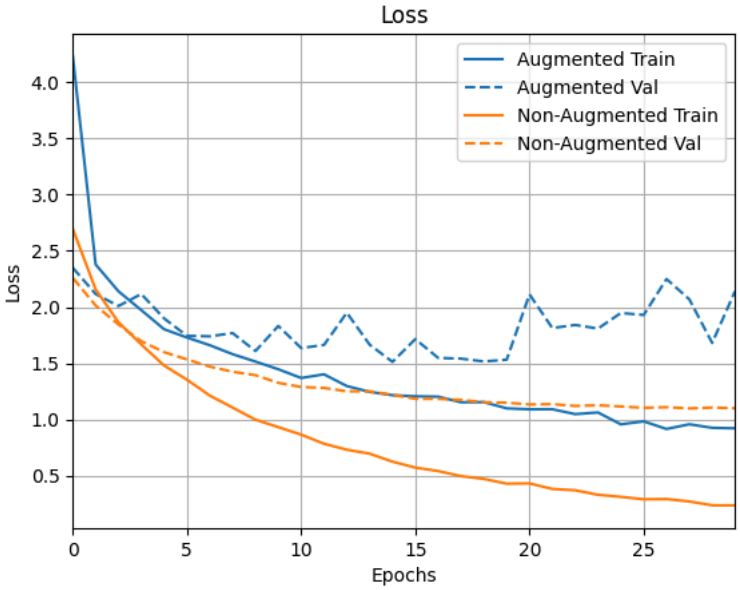

```
import tensorflow_docs as tfdocs
import tensorflow_docs.plots
plotter = tfdocs.plots.HistoryPlotter()
plotter.plot({ "Augmented": history_data_aug, "Non-Augmented": history}, metric = "acc")
plt.title("Accuracy")
plt.ylim([0,1])
```

```
(0.0, 1.0)
```




```
plotter = tfdocs.plots.HistoryPlotter()
plotter.plot({ "Augmented": history_data_aug, "Non-Augmented": history}, metric = "loss")
plt.title("Loss")
```

```
Text(0.5, 1.0, 'Loss')
```



As we can see, the augmented validation does not improve significantly compared to non-augmented validation data.

Double-click (or enter) to edit

Profile your input pipeline

```
pip install -U tensorboard_plugin_profile
```

```
Collecting tensorboard_plugin_profile
  Downloading tensorboard_plugin_profile-2.15.1-py3-none-any.whl (5.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.6/5.6 MB 22.7 MB/s eta 0:00:00
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<5.0.0dev,>=3.19.6 in /usr/local/lib/python3.10/dist-packages (from tensorboard_plugin_profile) (
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard_plugin_profile) (67.7.2)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard_plugin_profile) (1.16.0)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.10/dist-packages (from tensorboard_plugin_profile) (3.0.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=0.11.15->tensorboard_plugin_
Installing collected packages: gviz-api, tensorboard_plugin_profile
Successfully installed gviz-api-1.10.0 tensorboard_plugin_profile-2.15.1
```

```
import tensorboard

print(tensorboard.__version__)
```

```
2.15.2
```

```
Start coding or generate with AI.
```

```
batch_size=32
import tensorflow as tf
!rm -rf ./logs/

from datetime import datetime
import os


root_logdir = "logs"
run_id = datetime.now().strftime("%Y%m%d-%H%M%S")
logdir = os.path.join(root_logdir, run_id)


# Profile from batches 1 to 30
tb_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir,histogram_freq = 1,profile_batch = '1,30')
```

```
# Train the model and use the TensorBoard Keras callback to collect
# performance profiling data
model.fit(train_data,
          steps_per_epoch = train_data.samples // batch_size,
          epochs=30,
          callbacks=[tb_callback])
```

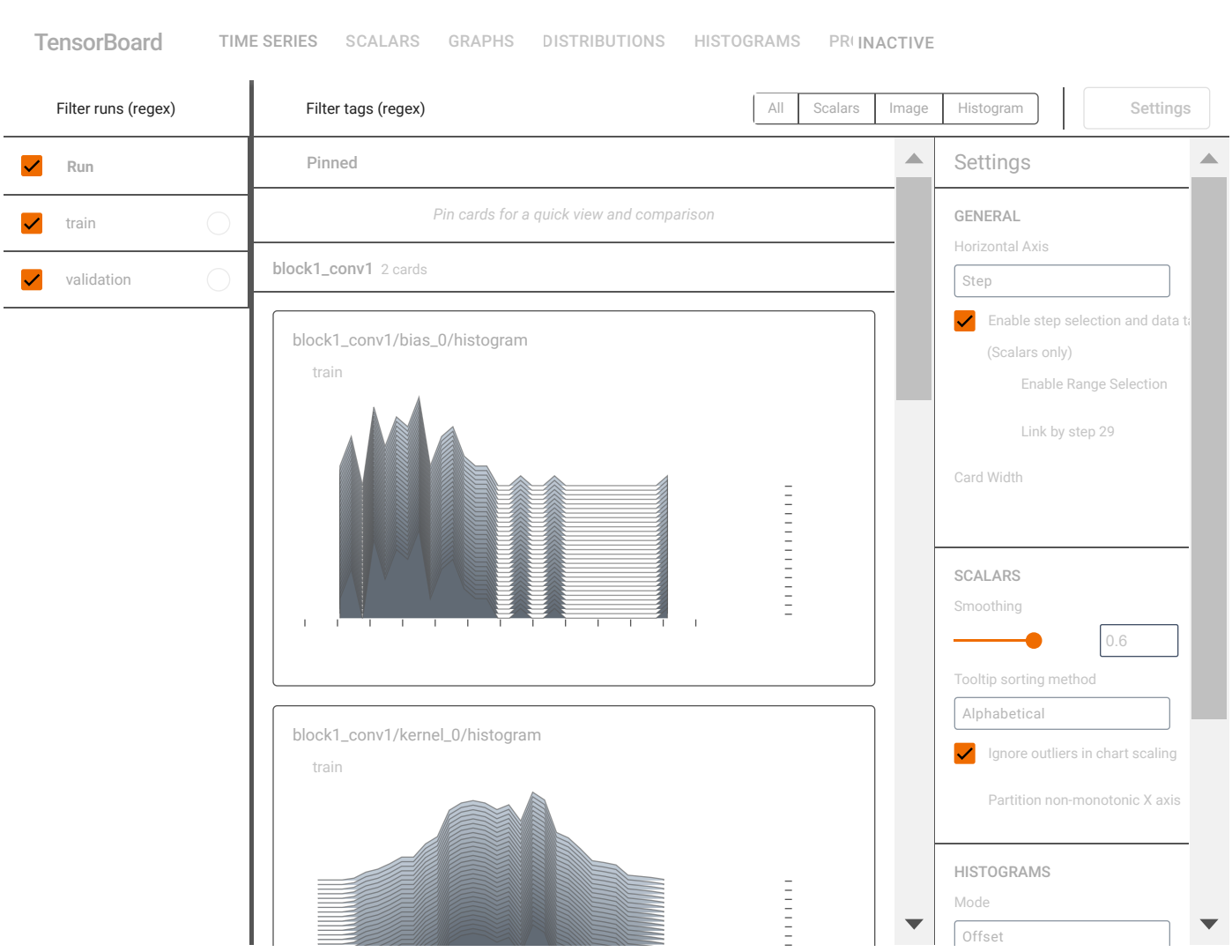
```
Epoch 1/30
10/46 [=====>.....................] - ETA: 5:14 - loss: 2.9174 - acc: 0.1318
```

```
%load_ext tensorboard
```

```
pip install protobuf==3.20.3
```

```
Requirement already satisfied: protobuf==3.20.3 in /usr/local/lib/python3.10/dist-packages (3.20.3)
```

```
%tensorboard --logdir /content/logs/20240512-022327
```



Based on the Profiling, the most time-consuming operation is input time because 97.3 % off the total step time sampled is waiting for input. To overcome this problem, we can come up with solutions:

- Enqueuing data: combine small input data chungks into fewer but larger chunks
- Data preprocessing: increase num_parallel_calls in Dataset map() or preprocess the data offline
- Reading data from files in advance: tune parameters with tf.data API (prefetch size, interleave cycle_length, reader buffer_size)
- Reading data from files on demand: read data in advance using (prefetch, interleave, reader buffer_

Task 2.2 Compare the performance under equal training time

In my case, data augmentation does not improve the model performance significantly. This is probably due to the fact that the quality of original data is not good enough so data augmentation could exacerbate these issues by introducing additional noise or artifacts. However in other case with better quality data, data augmentation is a powerful technique for enhancing the generalization ability of deep learning models and the longer training time would be well spent. To compare the model performance in these two scenarios, I will train the model with and without augmentation in the same time that is 10 minutes and compares their classification accuracy.

```
#Develope the model
from tensorflow.keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(256, 256, 3))
conv_base.trainable = False

conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 256, 256, 3)]     0

 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792

 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928

 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0

 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856

 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584

 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0

 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168

 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080

 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0

 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160

 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0

 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 0 (0.00 Byte)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import optimizers

def make_model(input_shape=(256, 256, 3), num_classes=18):
    # Define the base model (conv_base) or replace it with your desired pre-trained model
    # Replace this with your desired pre-trained model


    model = models.Sequential()
    model.add(conv_base)
    model.add(layers.Flatten())
    model.add(layers.Dense(256, activation='relu'))
    model.add(layers.Dropout(0.5))
```

```
    model.add(layers.Dense(num_classes, activation='softmax'))  # Use 'softmax' for multiclass classification

    model.compile(optimizer=optimizers.RMSprop(learning_rate=2e-5),
                  loss='categorical_crossentropy',  # Use 'categorical_crossentropy' for multiclass classification
                  metrics=['acc'])

    return model

model = make_model()
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 8, 8, 512)         14714688

 flatten_3 (Flatten)         (None, 32768)             0

 dense_6 (Dense)             (None, 256)               8388864

 dropout_3 (Dropout)         (None, 256)               0

 dense_7 (Dense)             (None, 18)                4626

=================================================================
Total params: 23108178 (88.15 MB)
Trainable params: 8393490 (32.02 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
import tensorflow as tf
data_train, data_validation = tf.keras.preprocessing.image_dataset_from_directory(
    '/content/drive/MyDrive/images',
    labels='inferred',
    label_mode='categorical',
    color_mode='rgb',
    batch_size=32,
    image_size=(256, 256),
    shuffle=True,
    seed=11,
    validation_split=0.4,  # Split the dataset into training and validation
    subset= 'both',
    interpolation='bilinear',
    follow_links=False,
    crop_to_aspect_ratio=False,

)
```

```
Found 1843 files belonging to 18 classes.
Using 1106 files for training.
Using 737 files for validation.
```

```
# Rescale pixel values to [0, 1]
data_train = data_train.map(lambda x, y: (x / 255.0, y))
data_validation = data_validation.map(lambda x, y: (x / 255.0, y))
```

The original data

```
import time

# Set the desired training time limit in seconds
desired_training_time = 600  # 10 mins

start_time = time.time()

# Train the model in a loop until the desired time limit is reached
while True:
    # Check if the current time exceeds the desired time limit
    if time.time() - start_time >= desired_training_time:
        break  # Exit the loop if the desired time limit is reached

    # Code for training your model for one epoch or one step
    history=model.fit(data_train, epochs=1, validation_data=data_validation)
```

```
35/35 [==============================] - 23s 576ms/step - loss: 2.6989 - acc: 0.2061 - val_loss: 2.3126 - val_acc: 0.3460
35/35 [==============================] - 22s 552ms/step - loss: 2.2084 - acc: 0.3562 - val_loss: 2.0667 - val_acc: 0.4369
35/35 [==============================] - 23s 545ms/step - loss: 1.9179 - acc: 0.4602 - val_loss: 1.8889 - val_acc: 0.4858
35/35 [==============================] - 20s 533ms/step - loss: 1.7096 - acc: 0.5398 - val_loss: 1.7523 - val_acc: 0.5170
35/35 [==============================] - 22s 547ms/step - loss: 1.5407 - acc: 0.5868 - val_loss: 1.6819 - val_acc: 0.5278
35/35 [==============================] - 21s 554ms/step - loss: 1.3960 - acc: 0.6148 - val_loss: 1.5631 - val_acc: 0.5712
35/35 [==============================] - 18s 463ms/step - loss: 1.2659 - acc: 0.6465 - val_loss: 1.5270 - val_acc: 0.5821
35/35 [==============================] - 19s 489ms/step - loss: 1.1586 - acc: 0.6944 - val_loss: 1.4602 - val_acc: 0.5957
35/35 [==============================] - 21s 543ms/step - loss: 1.0435 - acc: 0.7351 - val_loss: 1.4131 - val_acc: 0.6106
35/35 [==============================] - 18s 463ms/step - loss: 0.9666 - acc: 0.7731 - val_loss: 1.3851 - val_acc: 0.6079
35/35 [==============================] - 19s 476ms/step - loss: 0.8969 - acc: 0.7676 - val_loss: 1.3317 - val_acc: 0.6242
35/35 [==============================] - 23s 559ms/step - loss: 0.8123 - acc: 0.7839 - val_loss: 1.2947 - val_acc: 0.6364
35/35 [==============================] - 22s 551ms/step - loss: 0.7439 - acc: 0.8228 - val_loss: 1.2791 - val_acc: 0.6282
35/35 [==============================] - 23s 608ms/step - loss: 0.6828 - acc: 0.8336 - val_loss: 1.2622 - val_acc: 0.6309
35/35 [==============================] - 22s 594ms/step - loss: 0.6364 - acc: 0.8490 - val_loss: 1.2410 - val_acc: 0.6499
35/35 [==============================] - 19s 485ms/step - loss: 0.6077 - acc: 0.8553 - val_loss: 1.2296 - val_acc: 0.6431
35/35 [==============================] - 18s 456ms/step - loss: 0.5515 - acc: 0.8825 - val_loss: 1.2098 - val_acc: 0.6472
35/35 [==============================] - 25s 660ms/step - loss: 0.5064 - acc: 0.8834 - val_loss: 1.1852 - val_acc: 0.6540
35/35 [==============================] - 25s 646ms/step - loss: 0.4980 - acc: 0.8906 - val_loss: 1.1781 - val_acc: 0.6445
35/35 [==============================] - 18s 470ms/step - loss: 0.4519 - acc: 0.9132 - val_loss: 1.1681 - val_acc: 0.6540
35/35 [==============================] - 19s 487ms/step - loss: 0.4182 - acc: 0.9195 - val_loss: 1.1537 - val_acc: 0.6621
```

The augmented data

```
import time

# Set the desired training time limit in seconds
desired_training_time = 600  # 10 mins

start_time = time.time()

# Train the model in a loop until the desired time limit is reached
while True:
    # Check if the current time exceeds the desired time limit
    if time.time() - start_time >= desired_training_time:
        break  # Exit the loop if the desired time limit is reached

    # Code for training your model for one epoch or one step
    history_data_aug= model.fit(
    train_data,
    epochs=1, validation_data=val_data,
    )
```

```
47/47 [==============================] - 46s 953ms/step - loss: 2.6352 - acc: 0.2128 - val_loss: 2.3392 - val_acc: 0.3113
47/47 [==============================] - 46s 957ms/step - loss: 2.1944 - acc: 0.3635 - val_loss: 2.1293 - val_acc: 0.3884
47/47 [==============================] - 44s 932ms/step - loss: 1.9475 - acc: 0.4432 - val_loss: 1.9972 - val_acc: 0.4215
47/47 [==============================] - 49s 1s/step - loss: 1.7705 - acc: 0.5000 - val_loss: 1.8995 - val_acc: 0.4656
47/47 [==============================] - 45s 962ms/step - loss: 1.6092 - acc: 0.5405 - val_loss: 1.8235 - val_acc: 0.4711
47/47 [==============================] - 45s 957ms/step - loss: 1.5195 - acc: 0.5703 - val_loss: 1.7591 - val_acc: 0.5096
47/47 [==============================] - 47s 997ms/step - loss: 1.4275 - acc: 0.5919 - val_loss: 1.6888 - val_acc: 0.5455
47/47 [==============================] - 45s 953ms/step - loss: 1.3487 - acc: 0.6203 - val_loss: 1.6480 - val_acc: 0.5647
47/47 [==============================] - 45s 970ms/step - loss: 1.2728 - acc: 0.6338 - val_loss: 1.6218 - val_acc: 0.5565
47/47 [==============================] - 45s 951ms/step - loss: 1.2157 - acc: 0.6581 - val_loss: 1.5629 - val_acc: 0.5647
```

It can be seen that under equal training time of 10 mins, the model tested on original data gives significantly better accuracy than on augmented data (66% and 56%)

Task 2.3 Identifying model strengths and weaknesses

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?

```
#Load data
import tensorflow as tf
data_train, data_validation = tf.keras.preprocessing.image_dataset_from_directory(
    '/content/drive/MyDrive/images',
    labels='inferred',
    label_mode='categorical',
    color_mode='rgb',
    batch_size=32,
    image_size=(256, 256),
    shuffle=True,
    seed=11,
```

```
    validation_split=0.4,  # Split the dataset into training and validation
    subset= 'both',
    interpolation='bilinear',
    follow_links=False,
    crop_to_aspect_ratio=False,

)
```

```
Found 1843 files belonging to 18 classes.
Using 1106 files for training.
Using 737 files for validation.
```

```
# Rescale pixel values to [0, 1]
data_train = data_train.map(lambda x, y: (x / 255.0, y))
data_validation = data_validation.map(lambda x, y: (x / 255.0, y))


from tensorflow.keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(256, 256, 3))
conv_base.trainable = False

conv_base.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.
58889256/58889256 [==============================] - 0s 0us/step
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0

 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792

 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928

 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0

 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856

 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584

 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0

 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168

 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080

 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0

 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160

 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0

 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 0 (0.00 Byte)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import optimizers
```

```
def make_model(input_shape=(256, 256, 3), num_classes=18):
    # Define the base model (conv_base) or replace it with your desired pre-trained model
    # Replace this with your desired pre-trained model


    model = models.Sequential()
    model.add(conv_base)
    model.add(layers.Flatten())
    model.add(layers.Dense(256, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(num_classes, activation='softmax'))  # Use 'softmax' for multiclass classification

    model.compile(optimizer=optimizers.RMSprop(learning_rate=2e-5),
                  loss='categorical_crossentropy',  # Use 'categorical_crossentropy' for multiclass classification
                  metrics=['acc'])

    return model

model = make_model()
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 8, 8, 512)         14714688

 flatten (Flatten)           (None, 32768)             0

 dense (Dense)               (None, 256)               8388864

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 18)                4626

=================================================================
Total params: 23108178 (88.15 MB)
Trainable params: 8393490 (32.02 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
history= model.fit(
    data_train,
    epochs=30, validation_data=data_validation,
    )
```

Show hidden output

```
import numpy as np

# Get predictions on the test dataset
predictions = model.predict(data_validation)
```

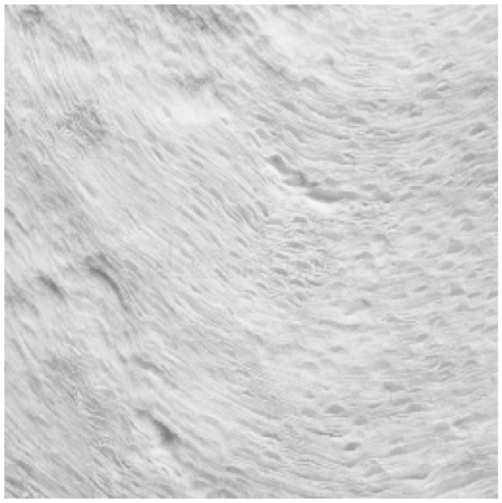
```
24/24 [==============================] - 7s 293ms/step
```

```
# Convert predicted probabilities to class labels
predicted_labels = np.argmax(predictions, axis=1)


 #Create a list of images from data_validation
 images = []
 for batch in data_validation:
      for image in batch[0]:
          images.append(image.numpy())


#Create a list of misclassified images
misclassified_images_dict = {}
j=0
for i in misclassified_indices.tolist():
    misclassified_images_dict[j] = (images[i], true_labels[i])
    j+=1
```

Print misclassified images

```
import numpy as np
import matplotlib.pyplot as plt
for j in range(20):
  image = misclassified_images_dict[j][0]
  label = misclassified_images_dict[j][1]
  plt.imshow(image, cmap='gray')
  plt.title('Label: ' + str(label))
  plt.axis('off')
  plt.show()
```

Label: 17



Label: 13

Label: 3

Label: 10

Label: 17

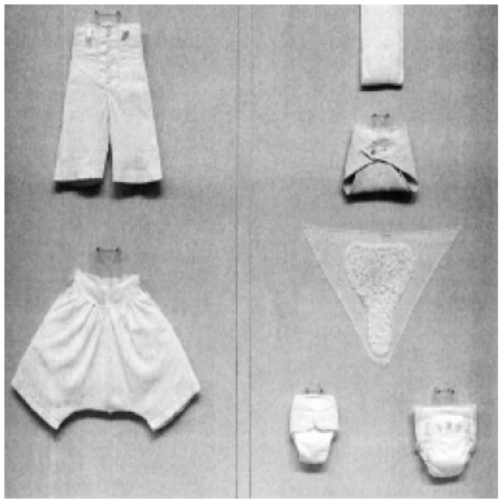
Label: 12

Label: 4

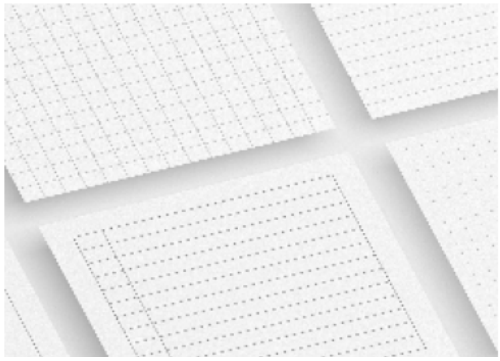
Label: 13

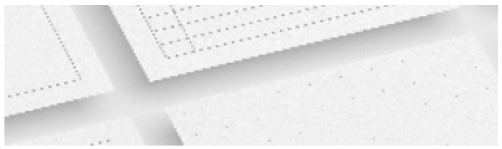
Label: 10

Label: 11



Label: 12

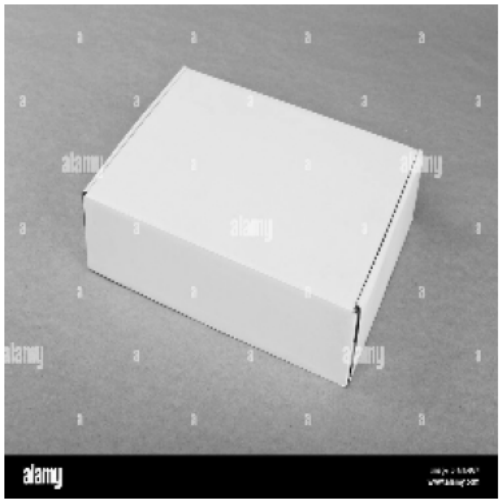




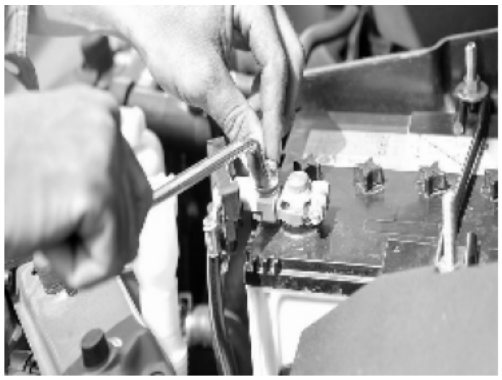
Label: 1

Label: 7

Label: 3

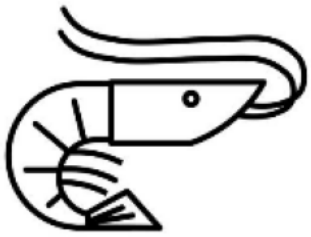


Label: 1



Label: 1

Label: 8



Label: 10

Label: 10

Label: 4

It can be seen from the misclassified images, many objects in the images are not clearly pictured (with enough contrast, lighting for the model to detect it) and they are not captured in a suitable angle so the model can be difficult to capture its most important features. Furthermore, there are pictures with many objects rather than a single one that it also deters the model from predicting exactly.

To improve the model on those images, I need to clean the data input: filter data of good quality (single objects, good lighting, contrast colors, angles), add more data to increase the data in-distribution so the model can generalize better, do data augmentation to improve generalization and data inputs.

Double-click (or enter) to edit

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.

You need to include sufficient analysis to demonstrate that:

- You have obtained a deeper understanding of challenges in model generalisation, through designing experiments and analysing the results.
- You have empirically evaluated different measures to address such challenges and can apply classroom learning to explain why each measure may or may not work.

I use a different dataset from a different source: Trashnet dataset from Stanford that contains 2527 images in different classes:501 glass, 594 paper, 403 cardboard, 482 plastic, 410 metal, 137 trash. I modify the dataset to match with the classes of my dataset and I will use it to test my models.

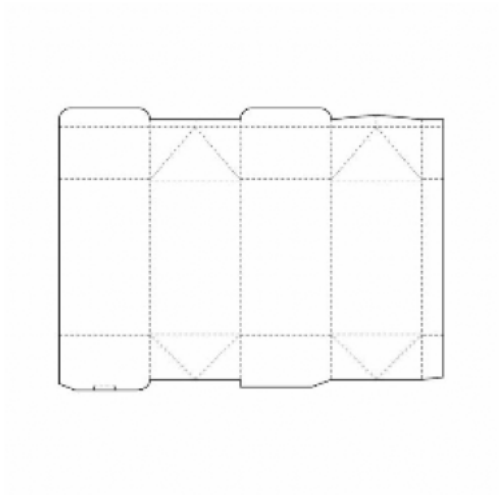
Display images from the original dataset with label "Cardboard"

```
import numpy as np
import matplotlib.pyplot as plt
for j in range(20):
  image = cardboard_images_dict[j][0]
  label = cardboard_images_dict[j][1]
  plt.imshow(image, cmap='gray')
  plt.title('Label: ' + str(label))
  plt.axis('off')
  plt.show()
```

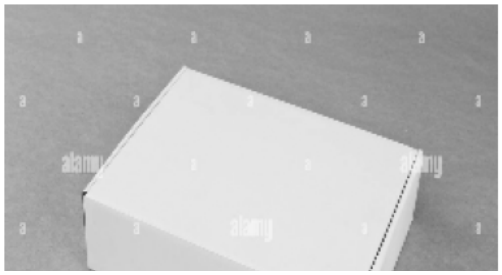
Label: 3

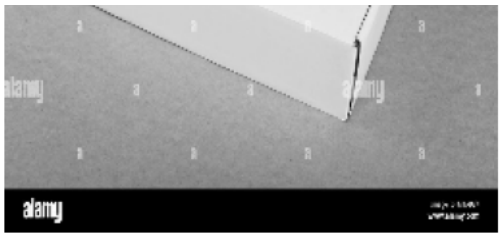
Label: 3

Label: 3



Label: 3



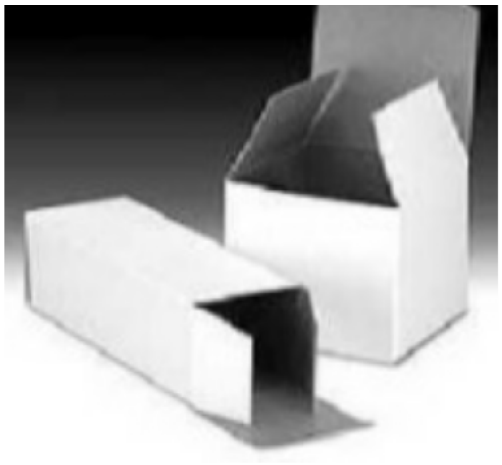


Label: 3

Label: 3

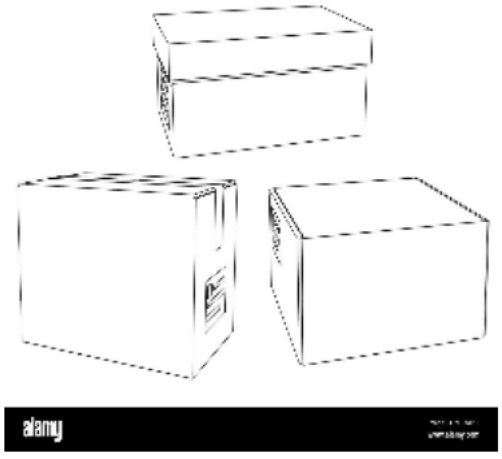
Label: 3

Label: 3



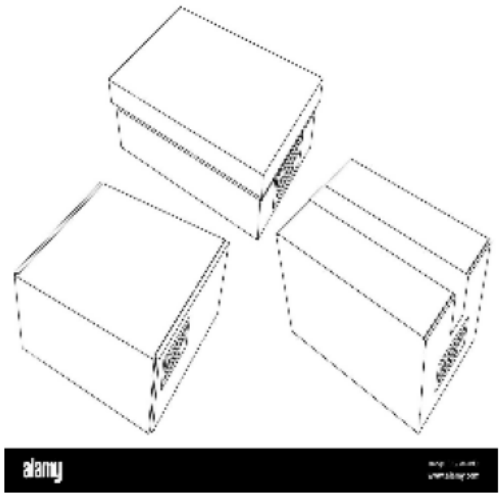
Label: 3

Label: 3



Label: 3

Label: 3



Label: 3

Label: 3

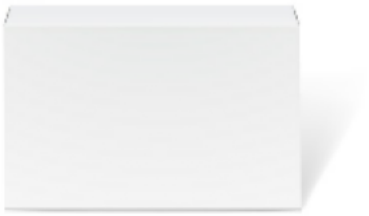
Label: 3

Label: 3

Label: 3

Label: 3

Label: 3

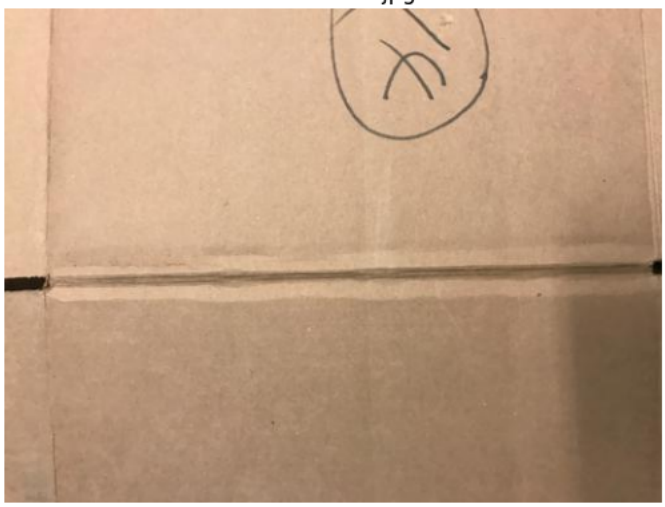


Label: 3

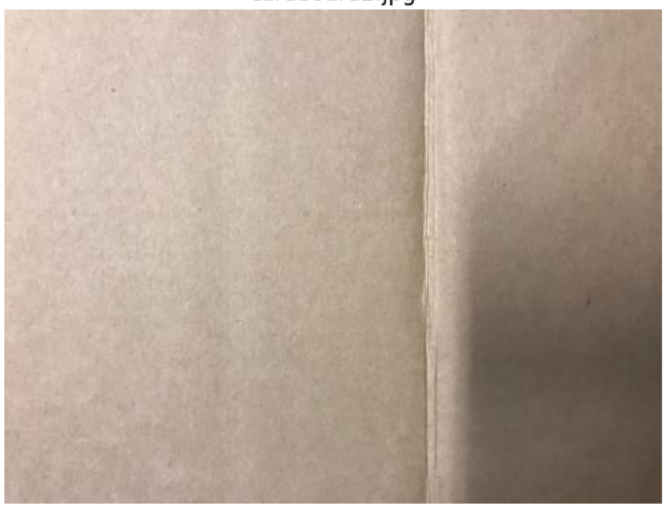
Display images from the new test dataset with label "Cardboard"

```
import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Define the folder path
folder_path = "/content/drive/MyDrive/images_test/cardboard"

# Display the first 10 images in the folder
for i in range(1, 11):
    # Construct the file path for each image
    image_path = os.path.join(folder_path, f"cardboard{i}.jpg")

    # Check if the file exists
    if os.path.exists(image_path):
        # Load the image
        image = mpimg.imread(image_path)

        # Display the image
        plt.imshow(image)
        plt.title(f"cardboard{i}.jpg")
        plt.axis('off')  # Turn off axis
        plt.show()
    else:
        print(f"Image {i} not found.")
```

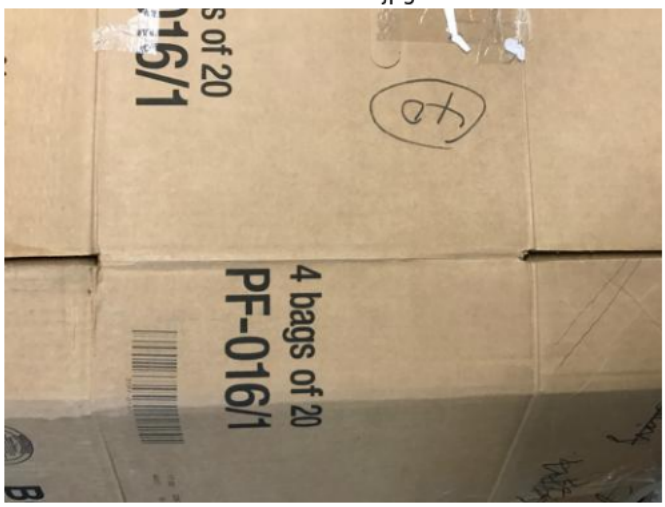
cardboard1.jpg



cardboard2.jpg



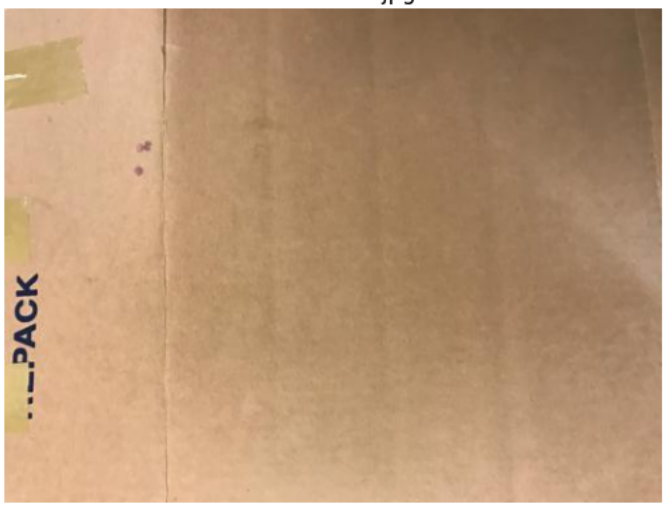
cardboard3.jpg



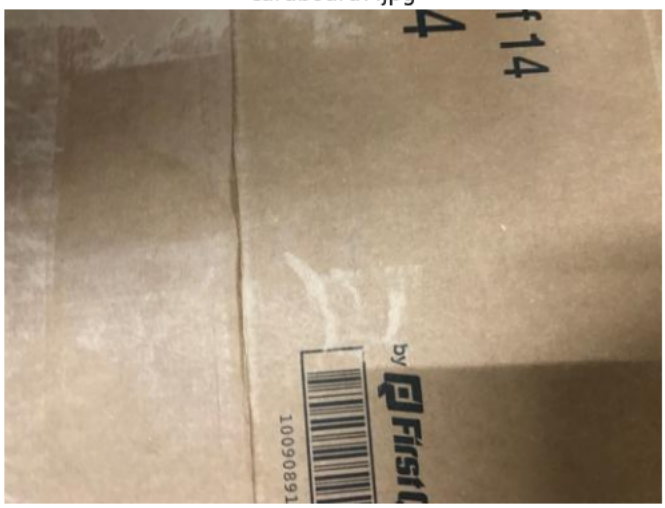
cardboard4.jpg

cardboard5.jpg

cardboard6.jpg



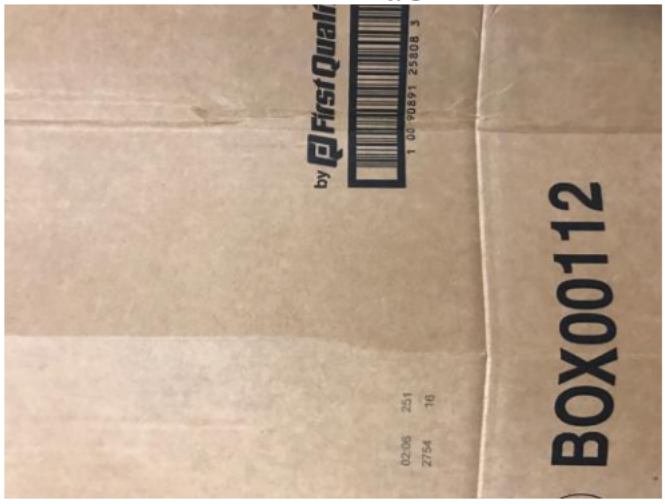
cardboard7.jpg



cardboard8.jpg

cardboard9.jpg

cardboard10.jpg



It can be seen clearly from images of the label 'cardboard' from two different source of dataset, the original dataset has different color, lighting. Many of these images depict entire cardboard objects captured from far distance. In contrast, the new dataset predominantly features close-up shots of cardboard fragments, showcasing intricate details and varying angles of the material. That is to say two different dataset posesses different features of images of the same class. Lets try to fit the model into the new dataset to see whether the model can perform well in the new dataset.

Test the model on new dataset

```
import tensorflow as tf
data_train, data_validation = tf.keras.preprocessing.image_dataset_from_directory(
    '/content/drive/MyDrive/images',
    labels='inferred',
    label_mode='categorical',
    color_mode='rgb',
    batch_size=32,
    image_size=(256, 256),
    shuffle=True,
    seed=11,
    validation_split=0.4,  # Split the dataset into training and validation
    subset= 'both',
    interpolation='bilinear',
    follow_links=False,
    crop_to_aspect_ratio=False,

)
class_names = data_train.class_names
```

```
Found 1843 files belonging to 18 classes.
Using 1106 files for training.
Using 737 files for validation.
```

```
Start coding or generate with AI.
```

```
# Load new dataset

import tensorflow as tf
data_test = tf.keras.preprocessing.image_dataset_from_directory(
    '/content/drive/MyDrive/images_test',
    labels='inferred',
    label_mode='categorical',
    color_mode='rgb',

    batch_size=32,
    image_size=(256, 256),
    shuffle=True,
    seed=10,
    validation_split=0,  # Split the dataset into training and validation

    interpolation='bilinear',
    follow_links=False,
    crop_to_aspect_ratio=False,

)
```

```
Found 2392 files belonging to 18 classes.
```

```
# Rescale pixel values to [0, 1]
data_test = data_test.map(lambda x, y: (x / 255.0, y))


#Load model trained on augmented data
model_test = tf.keras.models.load_model('/content/drive/MyDrive/Models/model_augment.h5')


#Test the model on the new dataset
loss, accuracy = model_test.evaluate(data_test)
```

```
75/75 [==============================] - 29s 244ms/step - loss: 7.4311 - acc: 0.0493
```

The test accuracy on the new test data is significantly low (4.93%). This strongly suggests that the model fitted on the orignal dataset causes overfitting and can not generalize to the new dataset.

In fact, generalization is a common challenge in deep learning models. We need to deal with issues like: the quality of dataset (clear images with right objects, right labels), limited training dataset (although large training sets can still produce biased models unable to perform well on new data (Torralba and Efros,2011), recent works revealing the presence of spurious correlations in the popular ImageNet dataset that can cause models to rely on irrelevant and non-generalising decision boundaries (Haynes, S.C., Johnston et al), the risk of biased predictions and generalisation failures (model overfitting).

In my case, the generalisation failures can be caused by following reasons:

- Data: the quality of dataset is not good (many images are not clear, do not capture right objects), the dataset is too small to capture the variety of objects (only nearly 2000 images)
- The model: the model tends to overfit the original dataset. Because it is a pre-trained network so it can fail to predict new classes. Therefore, to resolve this problem, I will come up with following solutions:
  - Fine-tune the model: fine-tune top layers of the model VGG16 and re-train the model
  - Regularize the model to reduce overfitting: use optimizers (use RMSprop to converge faster), use L2 regularization, dropout

Regularize the model: L2 regularization, Dropout, RMSprop optimizers, Data Augmentation

```
import tensorflow as tf
data_train, data_validation = tf.keras.preprocessing.image_dataset_from_directory(
    '/content/drive/MyDrive/images',
    labels='inferred',
    label_mode='categorical',
    color_mode='rgb',
    batch_size=32,
    image_size=(256, 256),
    shuffle=True,
    seed=10,
    validation_split=0.4,  # Split the dataset into training and validation
    subset= 'both',
    interpolation='bilinear',
    follow_links=False,
    crop_to_aspect_ratio=False,

)
class_names=data_train.class_names
```

```
Found 1843 files belonging to 18 classes.
Using 1106 files for training.
Using 737 files for validation.
```

```
class_names
```

```
['aluminium&steel',
 'batteries',
 'buildingwaste',
 'cardboard',
 'clothing&textiles',
 'drinkingglass',
 'electronicitems',
 'foam&polystyrene',
 'foodscraps',
 'glassjars&bottle',
 'housewaste',
 'nappies',
 'paper',
 'plasticbags',
 'plasticbottles&containers',
 'plastictoys',
 'takeawaycups',
 'tissuesandnapkins']
```

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=20,
                                   horizontal_flip=True,
                                   validation_split=0.2)


val_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
```

```
train_data = train_datagen.flow_from_directory( '/content/drive/MyDrive/images',
                                                seed=10,
                                                target_size=(256, 256),
                                                color_mode='rgb',
                                                batch_size=32,
                                                class_mode='categorical',
                                                shuffle=True,
                                                subset = 'training')
```

```
Found 1480 images belonging to 18 classes.
```

```
val_data = val_datagen.flow_from_directory('/content/drive/MyDrive/images',
                                           seed=10,
                                           target_size=(256, 256),
                                           color_mode='rgb',
                                           batch_size=32,
                                           class_mode='categorical',
                                           shuffle=False,
                                           subset = 'validation')
```

```
Found 363 images belonging to 18 classes.
```

```
from tensorflow.keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(256, 256, 3))
conv_base.trainable = False

conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0

 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792

 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928

 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0

 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856

 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584

 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0

 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168

 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080

 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0

 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160

 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0

 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 0 (0.00 Byte)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow.keras import regularizers

def make_model(input_shape=(256, 256, 3), num_classes=18):
    # Define the base model (conv_base) or replace it with your desired pre-trained model
    # Replace this with your desired pre-trained model


    model = models.Sequential()
    model.add(conv_base)
    model.add(layers.Flatten())
    model.add(layers.Dense(256, activation='relu',kernel_regularizer=regularizers.l2(0.01)))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(num_classes, activation='softmax'))  # Use 'softmax' for multiclass classification

    model.compile(optimizer=optimizers.RMSprop(learning_rate=2e-5), #use SGD with learning rate and momentum for better generalization
                  loss='categorical_crossentropy',  # Use 'categorical_crossentropy' for multiclass classification
                  metrics=['acc'])

    return model

model = make_model()
model.summary()
```

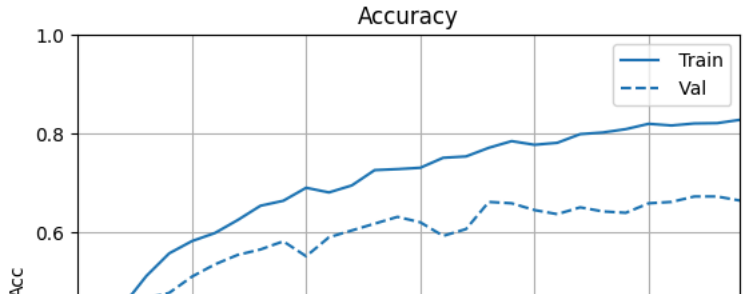
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 8, 8, 512)         14714688

 flatten (Flatten)           (None, 32768)             0

 dense (Dense)               (None, 256)               8388864

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 18)                4626

=================================================================
Total params: 23108178 (88.15 MB)
Trainable params: 8393490 (32.02 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
history= model.fit(
    train_data,
    epochs=30, validation_data=val_data,
    )
```

Show hidden output

```
#Plot accuracy of train and validation data
import tensorflow_docs as tfdocs
import tensorflow_docs.plots
import matplotlib.pyplot as plt


plotter = tfdocs.plots.HistoryPlotter()
plotter.plot({"": history}, metric = "acc")
plt.title("Accuracy")
plt.ylim([0,1])
```

```
(0.0, 1.0)
```




```
#Plot accuracy of train and validation data
import tensorflow_docs as tfdocs
import tensorflow_docs.plots
import matplotlib.pyplot as plt
```