CSCI-E25: Computer Vision

Final Project: Plant disease identification by Swarm Robotics in Agriculture

Harvard University
Spring 2024
Instructor: Stephen Elston
Submitted by: Bethun Bhowmik

Abstract

The report presents the development of an diaease identifiucation in plants via ground robots and agent-based model (ABM) simulation of Particle Swarm Optimization (PSO) algorithm to enhance agricultural practices through swarm robotics.

The primary goal of this project is to improve crop yields by effectively utilizing Computer Vision and PSO to manage plant disease identification.

Dataset used: cassava from Tensorflow datasets

Models used:

- Model 1: EfficientNetB3
- Model 2: InceptionResNetV2
- Model 3: EfficientNetV2S
- Model 4: MobileNetV2

Detailed comparative analysis on the models have been done. We have also run the PSO algorithm ro show swarm robotics characteristics on path plannning, coordination and communication between the robots

Introduction

In precision agriculture, the integration of technology has revolutionized traditional farming methods, paving the way for increased efficiency and sustainability. Among these technological advancements, the application of swarm robotics for plant disease identification and weeed detection has emerged as a particularly promising tool to enhance crop management and yield.

Infopulse estimates the market size of robotics in agriculture to be around $15 Billion (2022) which is projected to reach$40 Billion by 2028. Swarm robotics in agriculture leverages the collective behaviour of multiple robots to perform tasks more efficiently than individual robots could achieve alone.

By simulating the interactions between ground robots, this project aims to refine strategies for efficient plant disease identification while focussing on path planning, coordination, and real-time communication. The primary objective is to explore various models for plan disease identification and comparec them, thereby maximizing crop yield and reducing unnecessary labour and resource expenditure.

Project Goal:

- **Goal 1:** Classify different types of plant diseases and do necessary data pre processing
- **Goal 2:** Compare four models - EfficientNetB3, InceptionResNetV2, EfficientNetV2S, MobileNetV2, on accuracy, precision and recall for correct classification including top-3-accuracy
- **Goal 3:** Build a Particle Swarm Optimization (PSO) algorithm to enable path planning, optimization and coordination among ground robots

I have successfully incorporated all three goals in this project

Import necessary libraries, modules and models

We import the neccessary, modules and models required for the project.
The image size is set to 224 and batch size to 64.

```
pip install tensorflow-addons
```

```
    Requirement already satisfied: tensorflow-addons in /usr/local/lib/python3.10/dist-packages (0.23.0)
    Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow-addons) (24.0)
    Requirement already satisfied: typeguard<3.0.0,>=2.7 in /usr/local/lib/python3.10/dist-packages (from tensorflow-addons)
```

```
import numpy as np
import tensorflow_datasets as tfds
import tensorflow as tf
import matplotlib.pyplot as plt
import keras
from keras import layers
import pandas as pd
from time import time
from datetime import datetime
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, TensorBoard, CSVLogger
from keras.optimizers import Adam
from keras.metrics import TopKCategoricalAccuracy
import csv
from keras.models import Model, load_model
import numpy as np
from keras.layers import Lambda
import sklearn.metrics as metrics
from sklearn.metrics import confusion_matrix, classification_report
from keras import backend as K
from skimage.io import imread
from skimage.transform import resize
from keras.applications import EfficientNetB0
from keras.applications import EfficientNetB3
import tensorflow_addons as tfa
from tensorflow.keras.applications import EfficientNetV2S
from keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications import MobileNetV2
from keras.layers import Dense, GlobalAveragePooling2D
import requests

IMG_SIZE = 224
BATCH_SIZE = 64
```

```
/usr/local/lib/python3.10/dist-packages/tensorflow_addons/utils/tfa_eol_msg.py:23: UserWarning:

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(
```

Load the Cassava Dataset from Tensorflow Datasets

The Cassava dataset consists of leaf images for the cassava plant depicting **healthy** and four (4) disease conditions; ,

- Cassava Mosaic Disease (CMD)
- Cassava Bacterial Blight (CBB)
- Cassava Greem Mite (CGM)
- Cassava Brown Streak Disease (CBSD)

Dataset consists of a total of 9430 labelled images. The 9430 labelled images are split into a training set (5656), a test set(1885) and a validation set (1889). The number of images per class are unbalanced with the two disease classes CMD and CBSD having 72% of the images. I have used additional data augmentation techniques to acciunt for the imbalance.

```
dataset_name = "cassava"
(ds_train, ds_test), ds_info = tfds.load(
    dataset_name, split=["train", "test"], with_info=True, as_supervised=True
)
NUM_CLASSES = ds_info.features["label"].num_classes
```

```
Downloading and preparing dataset 1.26 GiB (download: 1.26 GiB, generated: Unl
```

```
Dl Completed...: 100%  1/1 [00:32<00:00, 10.71s/ url]

Dl Size...: 100%  1291/1291 [00:32<00:00, 123.77 MiB/s]

Extraction completed...: 100%  9430/9430 [00:32<00:00, 400.61 file/s]
```

```
IOPub message rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_msg_rate_limit`.

Current values:
NotebookApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
NotebookApp.rate_limit_window=3.0 (secs)

IOPub message rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_msg_rate_limit`.

Current values:
NotebookApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
NotebookApp.rate_limit_window=3.0 (secs)

Dataset cassava downloaded and prepared to /root/tensorflow_datasets/cassava/(
```

Data preprocessing - Resize the images

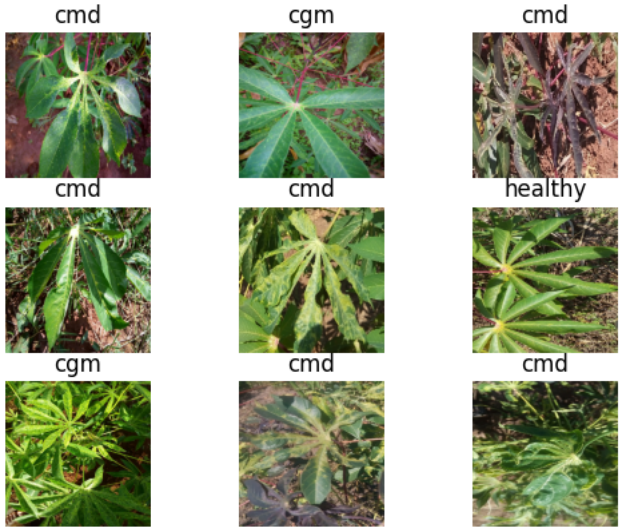
Resize the images in both the training and test datasets to a specified size (224, 224), as a preprocessing step to ensure uniformity in input dimensions.

```
size = (IMG_SIZE, IMG_SIZE)
ds_train = ds_train.map(lambda image, label: (tf.image.resize(image, size), label))
ds_test = ds_test.map(lambda image, label: (tf.image.resize(image, size), label))
```

Visualizing the data

Below we show the first 9 images in the dataset with their labels.

```
def format_label(label):
    string_label = label_info.int2str(label)
    parts = string_label.split("-")
    if len(parts) > 1:
        return parts[1]
    else:
        return parts[0]


label_info = ds_info.features["label"]
for i, (image, label) in enumerate(ds_train.take(9)):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(image.numpy().astype("uint8"))
    plt.title("{}".format(format_label(label)))
    plt.axis("off")
```



Data preprocessing - Check count of images in Train dataset

Below code shows the count of images for each of the five labels **cmd, cgm, healthy, cbsd, cbb**. We observe that the dataset is skewed heavily towards cmd

```
label_counts = {}

for image, label in ds_train:
    label_str = format_label(label)
    if label_str in label_counts:
        label_counts[label_str] += 1
    else:
        label_counts[label_str] = 1

# Print the counts for each label
for label, count in label_counts.items():
    print("Label: {}, Count: {}".format(label, count))
```

```
Label: cmd, Count: 2658
Label: cgm, Count: 773
Label: healthy, Count: 316
Label: cbsd, Count: 1443
Label: cbb, Count: 466
```

Data Preprocessing - Data augmentation

We define an image augmentation pipeline to enhance performance of models and generalization capabilities by:

Random gamma adjustment function to generate a random gamma value adding variability to image contrast

Image augmentation layers:

- Random rotation, translation, and flipping for geometric transformations
- Random contrast and brightness adjustments
- Random gamma adjustment, introduced through a Lambda layer

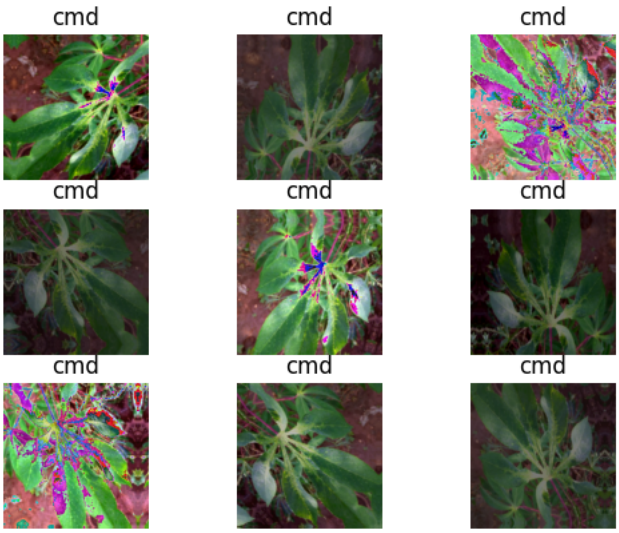
Image Augmentation Function to enhance the training dataset's diversity and model robustness.

```
def random_gamma(image):
    gamma = tf.random.uniform([], 0.8, 1.2)
    return tf.image.adjust_gamma(image, gamma=gamma)

img_augmentation_layers = [
    layers.RandomRotation(factor=0.15),
    layers.RandomTranslation(height_factor=0.1, width_factor=0.1),
    layers.RandomFlip(mode="horizontal_and_vertical"),
    layers.RandomContrast(factor=0.1),
    layers.RandomBrightness(factor=0.1),
    Lambda(random_gamma)
]

def img_augmentation(images):
    for layer in img_augmentation_layers:
        images = layer(images)
    return images
```

Visualize the effects of data augmentation

We observe that the 9 images below have been augmented and fine-tuned

```
for image, label in ds_train.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        aug_img = img_augmentation(np.expand_dims(image.numpy(), axis=0))
        aug_img = np.array(aug_img)
        plt.imshow(aug_img[0].astype("uint8"))
        plt.title("{}".format(format_label(label)))
        plt.axis("off")
```



Data Preprocessing - One Hot Encoding

Deep neural networks primarily operate on numerical data. Categorical labels need to be transformed into a format that the model can understand and learn from. So one-hot encoding ensures that the model treats classes as discrete categories rather than continuous values, preventing bias in the model's predictions based on the ordering of class labels.

```
# One-hot / categorical encoding
def input_preprocess_train(image, label):
    image = img_augmentation(image)
    label = tf.one_hot(label, NUM_CLASSES)
    return image, label


def input_preprocess_test(image, label):
    label = tf.one_hot(label, NUM_CLASSES)
    return image, label


ds_train = ds_train.map(input_preprocess_train, num_parallel_calls=tf.data.AUTOTUNE)
ds_train = ds_train.batch(batch_size=BATCH_SIZE, drop_remainder=True)
ds_train = ds_train.prefetch(tf.data.AUTOTUNE)

ds_test = ds_test.map(input_preprocess_test, num_parallel_calls=tf.data.AUTOTUNE)
ds_test = ds_test.batch(batch_size=BATCH_SIZE, drop_remainder=True)
```

Training a model from scratch

We build an EfficientNetB0, that is initialized from scratch. We will later use Transfer learning on this model

```
model = EfficientNetB0(
    include_top=True,
    weights=None,
    classes=NUM_CLASSES,
    input_shape=(IMG_SIZE, IMG_SIZE, 3),
)
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])

model.summary()
```

```
 predictions (Dense)         (None, 5)                 6405      ['top_dropout[0][0]']

==================================================================================================
Total params: 4055976 (15.47 MB)
Trainable params: 4013953 (15.31 MB)
Non-trainable params: 42023 (164.16 KB)
__________________________________________________________________________________________________
```

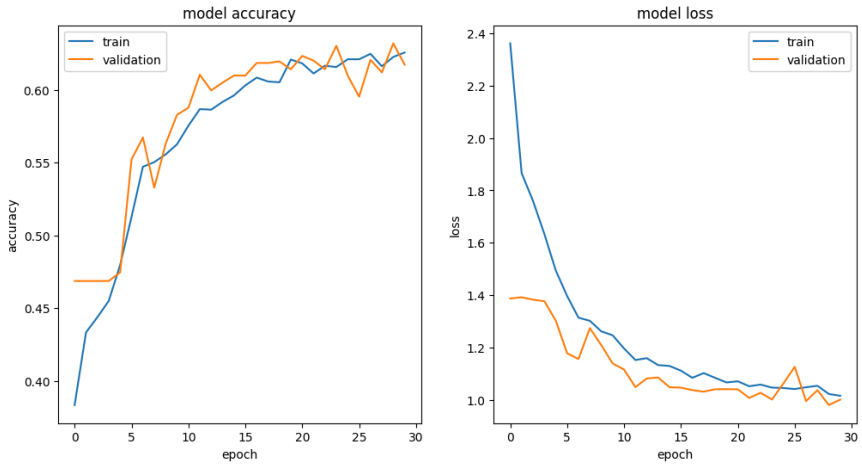
```
epochs = 30
hist = model.fit(ds_train, epochs=epochs, validation_data=ds_test)
```

```
Epoch 2/30
88/88 [==============================] - 267s 3s/step - loss: 1.8663 - accuracy: 0.4334 - val_loss: 1.3916 - val_accurac
Epoch 3/30
88/88 [==============================] - 265s 3s/step - loss: 1.7599 - accuracy: 0.4439 - val_loss: 1.3824 - val_accurac
Epoch 4/30
88/88 [==============================] - 272s 3s/step - loss: 1.6335 - accuracy: 0.4551 - val_loss: 1.3765 - val_accurac
Epoch 5/30
88/88 [==============================] - 273s 3s/step - loss: 1.4942 - accuracy: 0.4798 - val_loss: 1.3028 - val_accurac
Epoch 6/30
88/88 [==============================] - 270s 3s/step - loss: 1.3964 - accuracy: 0.5128 - val_loss: 1.1777 - val_accurac
Epoch 7/30
88/88 [==============================] - 269s 3s/step - loss: 1.3135 - accuracy: 0.5472 - val_loss: 1.1555 - val_accurac
Epoch 8/30
88/88 [==============================] - 270s 3s/step - loss: 1.3016 - accuracy: 0.5504 - val_loss: 1.2737 - val_accurac
Epoch 9/30
88/88 [==============================] - 269s 3s/step - loss: 1.2614 - accuracy: 0.5558 - val_loss: 1.2088 - val_accurac
Epoch 10/30
88/88 [==============================] - 265s 3s/step - loss: 1.2464 - accuracy: 0.5627 - val_loss: 1.1386 - val_accurac
Epoch 11/30
88/88 [==============================] - 267s 3s/step - loss: 1.1961 - accuracy: 0.5755 - val_loss: 1.1155 - val_accurac
Epoch 12/30
88/88 [==============================] - 272s 3s/step - loss: 1.1518 - accuracy: 0.5868 - val_loss: 1.0483 - val_accurac
Epoch 13/30
88/88 [==============================] - 270s 3s/step - loss: 1.1588 - accuracy: 0.5865 - val_loss: 1.0814 - val_accurac
Epoch 14/30
88/88 [==============================] - 269s 3s/step - loss: 1.1324 - accuracy: 0.5918 - val_loss: 1.0851 - val_accurac
Epoch 15/30
88/88 [==============================] - 274s 3s/step - loss: 1.1291 - accuracy: 0.5962 - val_loss: 1.0480 - val_accurac
Epoch 16/30
88/88 [==============================] - 269s 3s/step - loss: 1.1112 - accuracy: 0.6032 - val_loss: 1.0463 - val_accurac
Epoch 17/30
88/88 [==============================] - 272s 3s/step - loss: 1.0839 - accuracy: 0.6085 - val_loss: 1.0368 - val_accurac
Epoch 18/30
88/88 [==============================] - 272s 3s/step - loss: 1.1018 - accuracy: 0.6058 - val_loss: 1.0309 - val_accurac
Epoch 19/30
88/88 [==============================] - 269s 3s/step - loss: 1.0839 - accuracy: 0.6053 - val_loss: 1.0400 - val_accurac
Epoch 20/30
88/88 [==============================] - 275s 3s/step - loss: 1.0662 - accuracy: 0.6209 - val_loss: 1.0405 - val_accurac
Epoch 21/30
88/88 [==============================] - 276s 3s/step - loss: 1.0704 - accuracy: 0.6183 - val_loss: 1.0399 - val_accurac
Epoch 22/30
88/88 [==============================] - 273s 3s/step - loss: 1.0516 - accuracy: 0.6113 - val_loss: 1.0070 - val_accurac
Epoch 23/30
88/88 [==============================] - 273s 3s/step - loss: 1.0580 - accuracy: 0.6167 - val_loss: 1.0266 - val_accurac
Epoch 24/30
88/88 [==============================] - 272s 3s/step - loss: 1.0463 - accuracy: 0.6158 - val_loss: 1.0010 - val_accurac
Epoch 25/30
88/88 [==============================] - 273s 3s/step - loss: 1.0449 - accuracy: 0.6211 - val_loss: 1.0616 - val_accurac
Epoch 26/30
88/88 [==============================] - 270s 3s/step - loss: 1.0409 - accuracy: 0.6211 - val_loss: 1.1259 - val_accurac
Epoch 27/30
88/88 [==============================] - 271s 3s/step - loss: 1.0478 - accuracy: 0.6248 - val_loss: 0.9948 - val_accurac
Epoch 28/30
88/88 [==============================] - 271s 3s/step - loss: 1.0533 - accuracy: 0.6163 - val_loss: 1.0366 - val_accurac
Epoch 29/30
88/88 [==============================] - 278s 3s/step - loss: 1.0221 - accuracy: 0.6227 - val_loss: 0.9798 - val_accurac
Epoch 30/30
88/88 [==============================] - 276s 3s/step - loss: 1.0153 - accuracy: 0.6257 - val_loss: 1.0005 - val_accurac
```

Base model Output and Insights:

- The model undergoes training over 30 epochs, and both training and validation loss decrease steadily over time. This indicates that the model is effectively learning from the training data and generalizing well to unseen validation data.
- The training accuracy steadily increases over epochs. However, the validation accuracy plateaus around 61.75% in the final epoch
- There is a gap between training and validation accuracy, with training accuracy consistently higher than validation accuracy. This suggests that the model may be overfitting to the training data
- The model's performance appears relatively stable over the course of training suggesting that the model architecture and training process are robust and consistent
- The final validation accuracy of 61.75% provides a baseline measure of the model's performance on unseen data. There is room for improvement,which we will explore via transfer learning

```
def plot_hist(hist):
    _,ax = plt.subplots(1,2, figsize = (12,6))
    ax[0].plot(hist.history["accuracy"])
    ax[0].plot(hist.history["val_accuracy"])
    ax[0].set_title("model accuracy")
    ax[0].set_ylabel("accuracy")
    ax[0].set_xlabel("epoch")
    ax[0].legend(["train", "validation"], loc="upper left")
    ax[1].plot(hist.history["loss"])
    ax[1].plot(hist.history["val_loss"])
    ax[1].set_title("model loss")
    ax[1].set_ylabel("loss")
    ax[1].set_xlabel("epoch")
    ax[1].legend(["train", "validation"], loc="upper right")
    plt.show()

plot_hist(hist)
```



Model 1 - Transfer learning (EfficientNetB3) from pre-trained weights

Here we initialize the EfficientNetB3 model with pre-trained ImageNet weights, and we fine-tune it on our own dataset.

- The pretrained weights of the EfficientNetB3 model are used as a starting point. By freezing these weights (model0.trainable = False), the model leverages the knowledge learned from ImageNet without modifying the pretrained features.
- The top layers of the model are customized to adapt the architecture for our classification. This includes adding a global average pooling layer, batch normalization, dropout regularization, and a dense output layer with softmax activation.
- The model is compiled using the Adam optimizer with a learning rate of 1e-2
- The model is configured to compute accuracy and top-3 categorical accuracy metrics during training and evaluation.

```
def build_model0(num_classes):
    inputs = layers.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
    model0 = EfficientNetB3(include_top=False, input_tensor=inputs, weights="imagenet")

    # Freeze the pretrained weights
    model0.trainable = False

    # Rebuild top
    x = layers.GlobalAveragePooling2D(name="avg_pool")(model0.output)
    x = layers.BatchNormalization()(x)

    top_dropout_rate = 0.2
    x = layers.Dropout(top_dropout_rate, name="top_dropout")(x)
    outputs = layers.Dense(num_classes, activation="softmax", name="pred")(x)

    # Compile
    model0 = keras.Model(inputs, outputs, name="EfficientNet")
    optimizer = keras.optimizers.Adam(learning_rate=1e-2)
    top_3_accuracy = TopKCategoricalAccuracy(k=3)
    model0.compile(
```

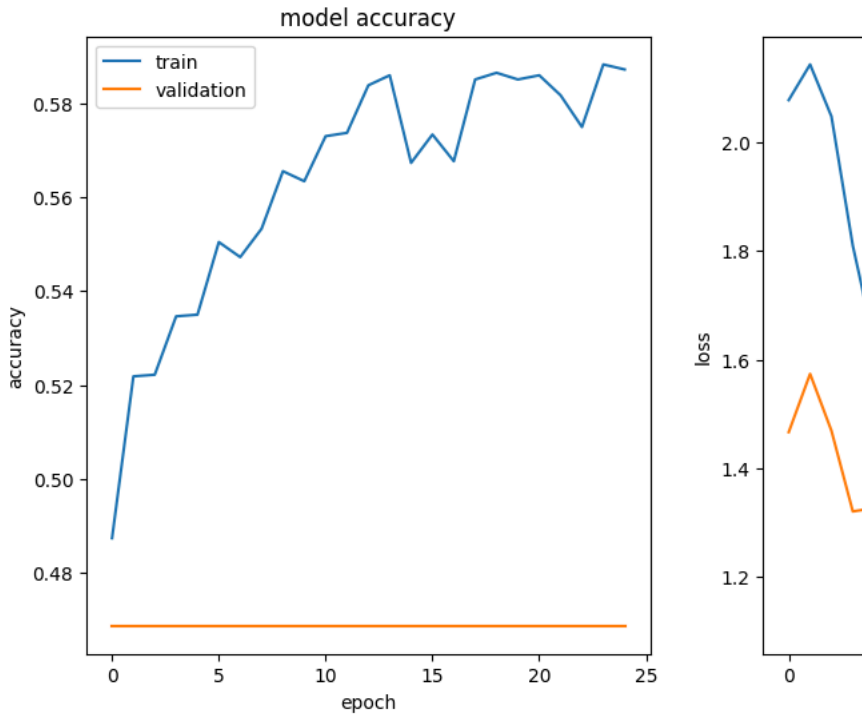
```
        optimizer=optimizer,
        loss="categorical_crossentropy",
        metrics=["accuracy", top_3_accuracy]
    )
    return model0
```

Transfer learning (EfficientNetB3) Output and Insights:

- The training accuracy started at 0.4874 and increased to around 0.5872 by the end of training
- The validation accuracy remained stagnant at 0.4688 throughout the training process, suggesting that the model failed to generalize well to unseen data
- The top-k categorical accuracy for both training and validation sets did not show significant improvement over the epochs
- Both training and validation metrics did not exhibit clear convergence or stability
- The loss values for both training and validation sets fluctuated but did not show a consistent decrease over the epochs

```
model0 = build_model0(num_classes=NUM_CLASSES)

epochs = 25
hist = model0.fit(ds_train, epochs=epochs, validation_data=ds_test)
plot_hist(hist)
```

```
Downloading data from https://storage.googleapis.com/keras-applications/effic
43941136/43941136 [==============================] - 0s 0us/step
Epoch 1/25
88/88 [==============================] - 170s 2s/step - loss: 2.0776 - accura
Epoch 2/25
88/88 [==============================] - 160s 2s/step - loss: 2.1432 - accura
Epoch 3/25
88/88 [==============================] - 159s 2s/step - loss: 2.0475 - accura
Epoch 4/25
88/88 [==============================] - 165s 2s/step - loss: 1.8097 - accura
Epoch 5/25
88/88 [==============================] - 161s 2s/step - loss: 1.6320 - accura
Epoch 6/25
88/88 [==============================] - 161s 2s/step - loss: 1.5706 - accura
Epoch 7/25
88/88 [==============================] - 164s 2s/step - loss: 1.5254 - accura
Epoch 8/25
88/88 [==============================] - 159s 2s/step - loss: 1.4221 - accura
Epoch 9/25
88/88 [==============================] - 163s 2s/step - loss: 1.3546 - accura
Epoch 10/25
88/88 [==============================] - 166s 2s/step - loss: 1.3191 - accura
Epoch 11/25
88/88 [==============================] - 161s 2s/step - loss: 1.2606 - accura
Epoch 12/25
88/88 [==============================] - 165s 2s/step - loss: 1.2269 - accura
Epoch 13/25
88/88 [==============================] - 165s 2s/step - loss: 1.1921 - accura
Epoch 14/25
88/88 [==============================] - 166s 2s/step - loss: 1.1716 - accura
Epoch 15/25
88/88 [==============================] - 161s 2s/step - loss: 1.2015 - accura
Epoch 16/25
88/88 [==============================] - 164s 2s/step - loss: 1.1510 - accura
Epoch 17/25
88/88 [==============================] - 165s 2s/step - loss: 1.1941 - accura
Epoch 18/25
88/88 [==============================] - 167s 2s/step - loss: 1.1332 - accura
Epoch 19/25
88/88 [==============================] - 167s 2s/step - loss: 1.1338 - accura
Epoch 20/25
88/88 [==============================] - 166s 2s/step - loss: 1.1296 - accura
Epoch 21/25
88/88 [==============================] - 173s 2s/step - loss: 1.1107 - accura
Epoch 22/25
88/88 [==============================] - 174s 2s/step - loss: 1.1292 - accura
Epoch 23/25
88/88 [==============================] - 174s 2s/step - loss: 1.1450 - accura
Epoch 24/25
88/88 [==============================] - 172s 2s/step - loss: 1.1337 - accura
Epoch 25/25
88/88 [==============================] - 172s 2s/step - loss: 1.1297 - accura
```



```
one_hot_labels = [x for _,x in  ds_test.unbatch().as_numpy_iterator()]
test_labels = np.argmax(one_hot_labels, axis=1)
test_labels[:100]
```

```
array([0, 3, 2, 1, 3, 3, 3, 3, 2, 2, 3, 3, 3, 1, 3, 0, 2, 3, 3, 3, 1, 1,
       4, 1, 0, 2, 1, 2, 1, 2, 3, 3, 4, 3, 3, 0, 3, 1, 3, 2, 3, 3, 3, 1,
       2, 2, 3, 3, 3, 2, 3, 3, 1, 3, 2, 2, 4, 1, 3, 3, 1, 3, 4, 3, 3, 1,
       3, 1, 3, 2, 4, 3, 2, 2, 1, 3, 3, 3, 3, 1, 1, 1, 3, 3, 1, 1, 2, 1, 3,
       0, 3, 1, 3, 1, 4, 1, 3, 3, 1, 1, 3])
```

```
predictions = model0.predict(ds_test, batch_size=1)
predicted = predictions.argmax(axis=1)
predicted[:100]
```

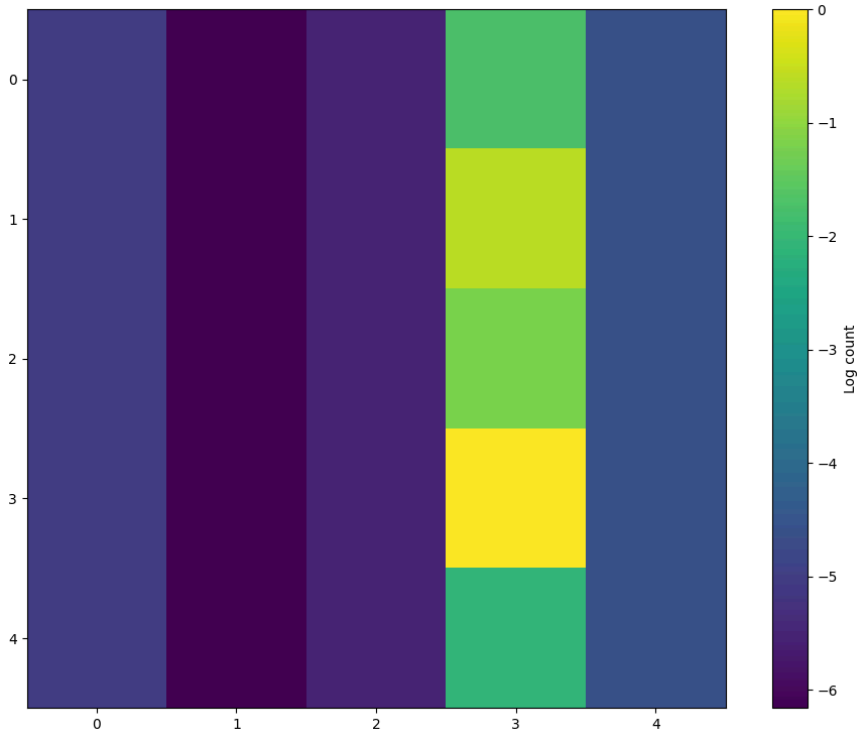
```
29/29 [==============================] - 42s 1s/step
array([3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3])
```

```
print('Overall accuracy of EfficientNetB3 model = ' + str(round(metrics.accuracy_score(test_labels, predicted), 4)))

unique_labels, label_counts = np.unique(test_labels, return_counts=True)
class_precision = metrics.precision_score(test_labels, predicted, labels=unique_labels, average=None)
class_recall = metrics.recall_score(test_labels, predicted, labels=unique_labels, average=None)

sum_label_counts = np.sum(label_counts)
weighted_average = lambda x: round(np.sum(np.divide(x * label_counts, sum_label_counts)), 4)
print('Average precision of EfficientNetB3 model = ' + str(weighted_average(class_precision)))
print('Average recall of EfficientNetB3 model = ' + str(weighted_average(class_recall)))
```

```
Overall accuracy of EfficientNetB3 model = 0.4688
Average precision of EfficientNetB3 model = 0.2197
Average recall of EfficientNetB3 model = 0.4688
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is il
  _warn_prf(average, modifier, msg_start, len(result))
```

The x-axis and y-axis represent the true labels and the predicted labels. The confusion matrix below tells that the model is unable to correctly classify the labels We observe misclassifications for most of the labels along the diagonals and scope for improvement.

```
confusion_matrix = metrics.confusion_matrix(test_labels, predicted)

plt.figure(figsize = (12,9))
p = plt.imshow(np.log(np.divide(confusion_matrix + 1.0, np.sum(confusion_matrix, axis=1))))
cb = plt.colorbar(p)
_=cb.set_label('Log count')
```

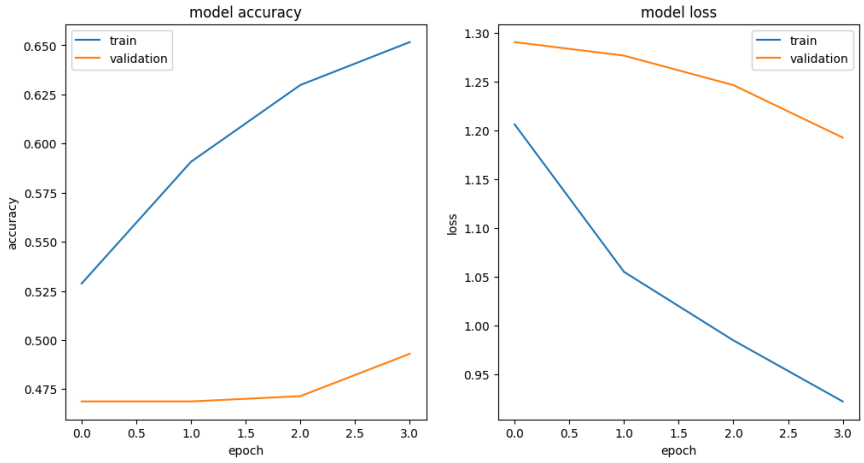


The function, unfreeze_model, unfreezes the InceptionResNetV2 model and then selectively unfreezes the top 20 layers while keeping the BatchNormalization layers frozen. Finally, it compiles the model with Adam optimizer with a new learning_rate=1e-5 to facilitate further training.

```
def unfreeze_model(model):
    # Unfreeze the entire model
    model.trainable = True

    # We unfreeze the top 20 layers while leaving BatchNorm layers frozen
    for layer in model.layers[-20:]:
        if not isinstance(layer, layers.BatchNormalization):
            layer.trainable = True

    optimizer = keras.optimizers.Adam(learning_rate=1e-5)
    model.compile(
        optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"]
    )

# Unfreeze the model
unfreeze_model(model0)
```

```
epochs = 4
top_k_accuracy = TopKCategoricalAccuracy(k=3)
optimizer = keras.optimizers.Adam(learning_rate=1e-5)

# Compile the model with top_k_accuracy as a metric and Adam optimizer
model0.compile(
    optimizer=optimizer,
    loss="categorical_crossentropy",
    metrics=["accuracy", top_k_accuracy]
)

hist = model0.fit(ds_train, epochs=epochs, validation_data=ds_test)
plot_hist(hist)
```

```
Epoch 1/4
88/88 [==============================] - 544s 6s/step - loss: 1.2060 - accura
Epoch 2/4
88/88 [==============================] - 510s 6s/step - loss: 1.0549 - accura
Epoch 3/4
88/88 [==============================] - 518s 6s/step - loss: 0.9848 - accura
Epoch 4/4
88/88 [==============================] - 517s 6s/step - loss: 0.9222 - accura
```




```
one_hot_labels = [x for _,x in  ds_test.unbatch().as_numpy_iterator()]
test_labels = np.argmax(one_hot_labels, axis=1)
test_labels[:100]
```

```
array([0, 3, 2, 1, 3, 3, 3, 3, 2, 2, 3, 3, 3, 1, 3, 0, 2, 3, 3, 3, 1, 1,
       4, 1, 0, 2, 1, 2, 1, 2, 3, 3, 4, 3, 3, 0, 3, 1, 3, 2, 3, 3, 3, 1,
       2, 2, 3, 3, 3, 2, 3, 3, 1, 3, 2, 2, 4, 1, 3, 3, 1, 3, 4, 3, 3, 1,
       3, 1, 3, 2, 4, 3, 2, 2, 1, 3, 3, 3, 1, 1, 1, 3, 3, 1, 1, 2, 1, 3,
       0, 3, 1, 3, 1, 4, 1, 3, 3, 1, 1, 3])
```

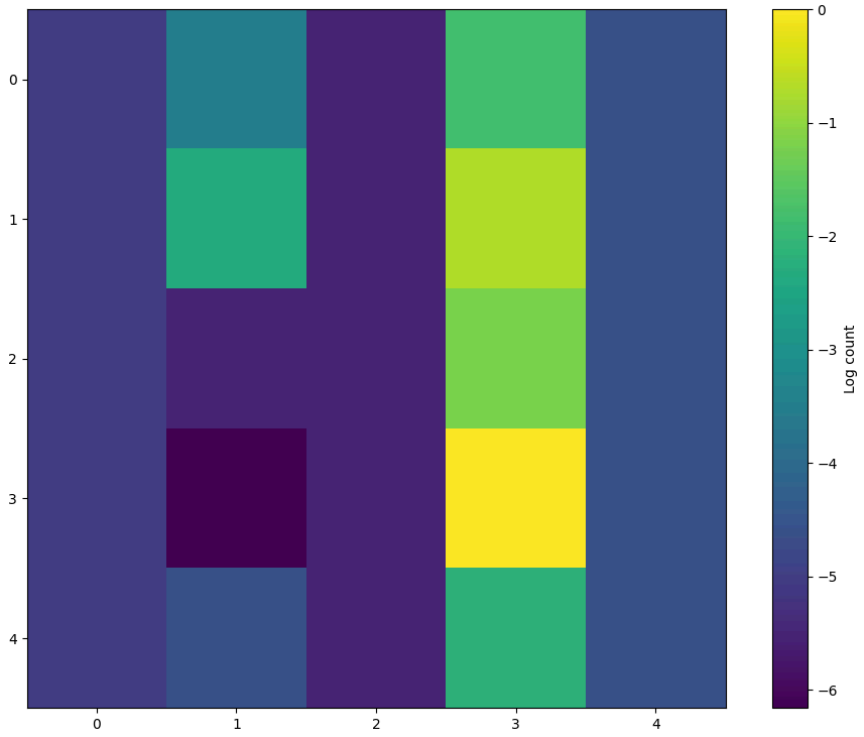
```
predictions = model0.predict(ds_test, batch_size=1)
predicted = predictions.argmax(axis=1)
predicted[:100]
```

```
29/29 [==============================] - 43s 1s/step
array([3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1,
       1, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3])
```

```
print('Overall accuracy of EfficientNetB3 model = ' + str(round(metrics.accuracy_score(test_labels, predicted), 4)))

unique_labels, label_counts = np.unique(test_labels, return_counts=True)
class_precision = metrics.precision_score(test_labels, predicted, labels=unique_labels, average=None)
class_recall = metrics.recall_score(test_labels, predicted, labels=unique_labels, average=None)

sum_label_counts = np.sum(label_counts)
weighted_average = lambda x: round(np.sum(np.divide(x * label_counts, sum_label_counts)), 4)
print('Average precision of EfficientNetB3 model = ' + str(weighted_average(class_precision)))
print('Average recall of EfficientNetB3 model = ' + str(weighted_average(class_recall)))
```

```
Overall accuracy of EfficientNetB3 model = 0.493
Average precision of EfficientNetB3 model = 0.4099
Average recall of EfficientNetB3 model = 0.493
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is il
  _warn_prf(average, modifier, msg_start, len(result))
```

```
confusion_matrix = metrics.confusion_matrix(test_labels, predicted)

plt.figure(figsize = (12,9))
p = plt.imshow(np.log(np.divide(confusion_matrix + 1.0, np.sum(confusion_matrix, axis=1))))
cb = plt.colorbar(p)
_=cb.set_label('Log count')
```



Model 2 - Transfer learning (InceptionResNetV2) from pre-trained weights

Here we initialize the InceptionResNetV2 model with pre-trained ImageNet weights, and we fine-tune it on our own dataset

- The model is compiled using the Adam optimizer with a learning rate of 1e-2
- The categorical cross-entropy loss function is employed for multi-class classification tasks. Additionally, the model's performance is evaluated during training using standard accuracy metrics and a custom metric for top-3 accuracy

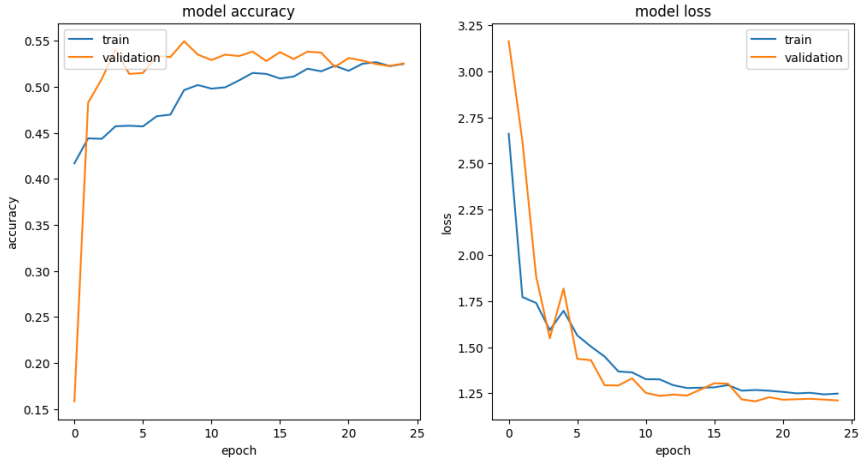
```
def build_model1(num_classes):
    inputs = layers.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
    base_model = InceptionResNetV2(include_top=False, input_tensor=inputs, weights="imagenet")

    # Freeze the pretrained weights
    base_model.trainable = False

    # Rebuild top
    x = layers.GlobalAveragePooling2D(name="avg_pool")(base_model.output)
    x = layers.BatchNormalization()(x)
    top_dropout_rate = 0.2
    x = layers.Dropout(top_dropout_rate, name="top_dropout")(x)
    outputs = layers.Dense(num_classes, activation="softmax", name="pred")(x)

    # Compile
    model1 = keras.Model(inputs, outputs, name="InceptionResNetV2")
    optimizer = keras.optimizers.Adam(learning_rate=1e-2)

    # Custom metric for top 3 accuracy
    top_3_accuracy = TopKCategoricalAccuracy(k=3)

    model1.compile(
        optimizer=optimizer,
        loss="categorical_crossentropy",
        metrics=["accuracy", top_3_accuracy]
    )
    return model1
```

Transfer learning (InceptionResNetV2) Output and Insights:

- The training loss decreased from 2.66 to 1.25, while the training accuracy increased from around 0.42 to 0.52. This suggests that the model was learning and improving its performance over the course of training
- The validation loss decreased from about 3.16 to 1.21, and the validation accuracy increased from 0.16 to 0.53. This indicates that the model's performance on unseen data improved significantly during training
- The top-k categorical accuracy also showed improvement during training for both training and validation sets
- The training and validation metrics appear to stabilize towards the later epochs, indicating that the model has converged to a relatively stable performance level
- There is no significant indication of overfitting as both training and validation metrics follow a similar trend, and the validation performance is close to the training performance throughout the training process.

```
model1 = build_model1(num_classes=NUM_CLASSES)

epochs = 25
hist = model1.fit(ds_train, epochs=epochs, validation_data=ds_test)
plot_hist(hist)
```

```
88/88 [==============================] - 90s 1s/step - loss: 1.5925 - accuracy
Epoch 5/25
88/88 [==============================] - 89s 1s/step - loss: 1.6978 - accuracy
Epoch 6/25
88/88 [==============================] - 89s 1s/step - loss: 1.5643 - accuracy
Epoch 7/25
88/88 [==============================] - 89s 1s/step - loss: 1.5032 - accuracy
Epoch 8/25
88/88 [==============================] - 91s 1s/step - loss: 1.4485 - accuracy
Epoch 9/25
88/88 [==============================] - 90s 1s/step - loss: 1.3677 - accuracy
Epoch 10/25
88/88 [==============================] - 90s 1s/step - loss: 1.3627 - accuracy
Epoch 11/25
88/88 [==============================] - 94s 1s/step - loss: 1.3256 - accuracy
Epoch 12/25
88/88 [==============================] - 92s 1s/step - loss: 1.3248 - accuracy
Epoch 13/25
88/88 [==============================] - 93s 1s/step - loss: 1.2932 - accuracy
Epoch 14/25
88/88 [==============================] - 90s 1s/step - loss: 1.2776 - accuracy
Epoch 15/25
88/88 [==============================] - 92s 1s/step - loss: 1.2792 - accuracy
Epoch 16/25
88/88 [==============================] - 92s 1s/step - loss: 1.2813 - accuracy
Epoch 17/25
88/88 [==============================] - 94s 1s/step - loss: 1.2934 - accuracy
Epoch 18/25
88/88 [==============================] - 94s 1s/step - loss: 1.2634 - accuracy
Epoch 19/25
88/88 [==============================] - 94s 1s/step - loss: 1.2669 - accuracy
Epoch 20/25
88/88 [==============================] - 92s 1s/step - loss: 1.2628 - accuracy
Epoch 21/25
88/88 [==============================] - 92s 1s/step - loss: 1.2567 - accuracy
Epoch 22/25
88/88 [==============================] - 94s 1s/step - loss: 1.2486 - accuracy
Epoch 23/25
88/88 [==============================] - 91s 1s/step - loss: 1.2516 - accuracy
Epoch 24/25
88/88 [==============================] - 93s 1s/step - loss: 1.2429 - accuracy
Epoch 25/25
88/88 [==============================] - 88s 1s/step - loss: 1.2473 - accuracy
```




```
one_hot_labels = [x for _,x in  ds_test.unbatch().as_numpy_iterator()]
test_labels = np.argmax(one_hot_labels, axis=1)
test_labels[:100]
```

```
array([0, 3, 2, 1, 3, 3, 3, 3, 2, 2, 3, 3, 3, 1, 3, 0, 2, 3, 3, 3, 1, 1,
       4, 1, 0, 2, 1, 2, 1, 2, 3, 3, 4, 3, 3, 0, 3, 1, 3, 2, 3, 3, 3, 1,
       2, 2, 3, 3, 3, 2, 3, 3, 1, 3, 2, 2, 4, 1, 3, 3, 1, 3, 4, 3, 3, 1,
       3, 1, 3, 2, 4, 3, 2, 2, 1, 3, 3, 3, 1, 1, 1, 3, 3, 1, 1, 2, 1, 3,
       0, 3, 1, 3, 1, 4, 1, 3, 3, 1, 1, 3])
```

```
predictions = model1.predict(ds_test, batch_size=1)
predicted = predictions.argmax(axis=1)
predicted[:100]
```

```
29/29 [==============================] - 25s 764ms/step
array([0, 3, 3, 3, 3, 3, 3, 3, 1, 1, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3,
       1, 1, 3, 3, 3, 3, 3, 1, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 1, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 1, 1, 3, 3, 3, 1, 3, 3, 3,
       0, 3, 3, 3, 3, 3, 1, 3, 3, 3, 1, 1])
```

- The overall accuracy of the model is 52.53%, which is lower than the previous transfer learning model used.
- The average precision of the model is 40.08% which is also not too encouraging
- The average recall of the model is also 52.53%

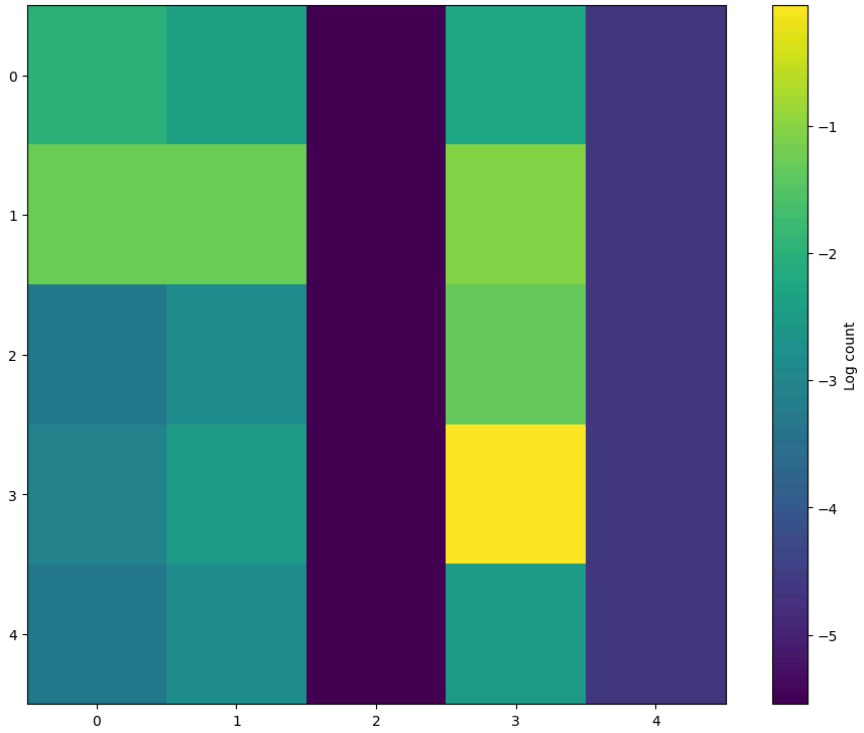
```
print('Overall accuracy of InceptionResNetV2 model = ' + str(round(metrics.accuracy_score(test_labels, predicted), 4)))

unique_labels, label_counts = np.unique(test_labels, return_counts=True)
class_precision = metrics.precision_score(test_labels, predicted, labels=unique_labels, average=None, zero_division=0)
class_recall = metrics.recall_score(test_labels, predicted, labels=unique_labels, average=None)

sum_label_counts = np.sum(label_counts)
weighted_average = lambda x: round(np.sum(np.divide(x * label_counts, sum_label_counts)), 4)
print('Average precision of InceptionResNetV2 model = ' + str(weighted_average(class_precision)))
print('Average recall of InceptionResNetV2 model = ' + str(weighted_average(class_recall)))
```

```
Overall accuracy of InceptionResNetV2 model = 0.5253
Average precision of InceptionResNetV2 model = 0.4008
Average recall of InceptionResNetV2 model = 0.5253
```

The x-axis and y-axis represent the true labels and the predicted labels. The confusion matrix below tells that the model is unable to correctly classify some of the labels

- The cells along the diagonal represent correct classifications, where the true label matches the predicted label. **We observe misclassifications for some of the labels along the diagonals and scope for improvement.**

```
confusion_matrix = metrics.confusion_matrix(test_labels, predicted)

plt.figure(figsize=(12, 9))
p = plt.imshow(np.log(np.divide(confusion_matrix + 1.0, np.sum(confusion_matrix, axis=1))))
cb = plt.colorbar(p)
_= cb.set_label('Log count')
```

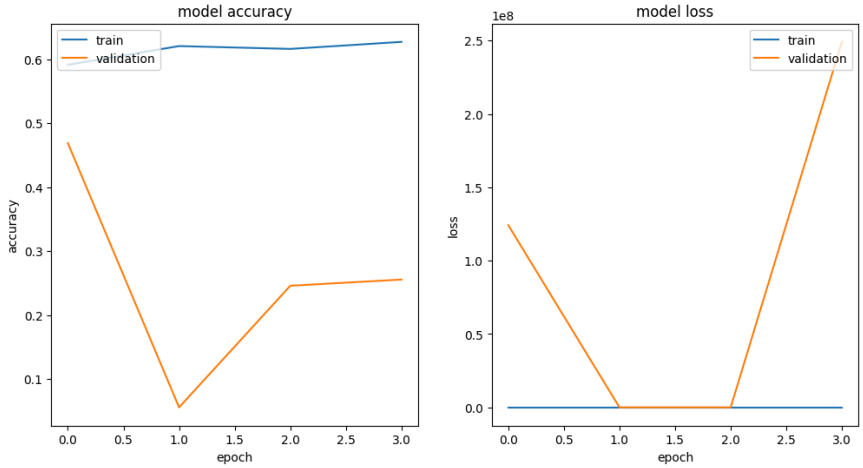


The function, unfreeze_model, unfreezes the InceptionResNetV2 model and then selectively unfreezes the top 20 layers while keeping the BatchNormalization layers frozen. Finally, it compiles the model with Adam optimizer with a new learning_rate=1e-5 to facilitate further training.

```
def unfreeze_model(model):
    # Unfreeze the entire model
    model.trainable = True

    # We unfreeze the top 20 layers while leaving BatchNorm layers frozen
    for layer in model.layers[-20:]:
        if not isinstance(layer, layers.BatchNormalization):
            layer.trainable = True

    optimizer = keras.optimizers.Adam(learning_rate=1e-5)
    model.compile(
        optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"]
    )

# Unfreeze the InceptionResNetV2 model
unfreeze_model(model1)
```

```
epochs = 4
top_k_accuracy = TopKCategoricalAccuracy(k=3)
optimizer = keras.optimizers.Adam(learning_rate=1e-2)

# Compile the model with top_k_accuracy as a metric and Adam optimizer
model1.compile(
    optimizer=optimizer,
    loss="categorical_crossentropy",
    metrics=["accuracy", top_k_accuracy]
)

hist = model1.fit(ds_train, epochs=epochs, validation_data=ds_test)
plot_hist(hist)
```

```
Epoch 1/4
88/88 [==============================] - 489s 5s/step - loss: 1.1509 - accura
Epoch 2/4
88/88 [==============================] - 443s 5s/step - loss: 1.0727 - accura
Epoch 3/4
88/88 [==============================] - 434s 5s/step - loss: 1.0939 - accura
Epoch 4/4
88/88 [==============================] - 434s 5s/step - loss: 1.0425 - accura
```




```
one_hot_labels = [x for _,x in  ds_test.unbatch().as_numpy_iterator()]
test_labels = np.argmax(one_hot_labels, axis=1)
test_labels[:100]
```

```
array([0, 3, 2, 1, 3, 3, 3, 3, 2, 2, 3, 3, 3, 1, 3, 0, 2, 3, 3, 3, 1, 1,
       4, 1, 0, 2, 1, 2, 1, 2, 3, 3, 4, 3, 3, 0, 3, 1, 3, 2, 3, 3, 3, 1,
       2, 2, 3, 3, 3, 2, 3, 3, 1, 3, 2, 2, 4, 1, 3, 3, 1, 3, 4, 3, 3, 1,
       3, 1, 3, 2, 4, 3, 2, 2, 1, 3, 3, 3, 1, 1, 1, 3, 3, 1, 1, 2, 1, 3,
       0, 3, 1, 3, 1, 4, 1, 3, 3, 1, 1, 3])
```

```
predictions = model1.predict(ds_test, batch_size=1)
predicted = predictions.argmax(axis=1)
predicted[:100]
```

```
29/29 [==============================] - 26s 786ms/step
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

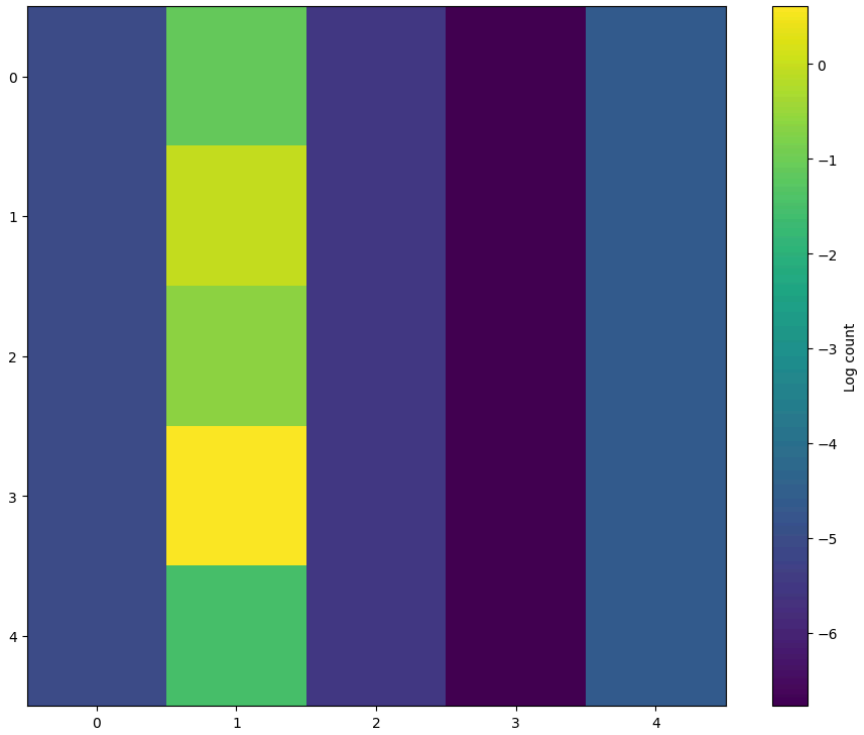
Insights from unfreezing the InceptionResNetV2 model

We observe that the overall accuracy, as well as the average precision and recall decreased significantly on unfreezing the InceptionResNetV2 model

```
print('Overall accuracy of InceptionResNetV2 model = ' + str(round(metrics.accuracy_score(test_labels, predicted), 4)))

unique_labels, label_counts = np.unique(test_labels, return_counts=True)
class_precision = metrics.precision_score(test_labels, predicted, labels=unique_labels, average=None, zero_division=0)
class_recall = metrics.recall_score(test_labels, predicted, labels=unique_labels, average=None)

sum_label_counts = np.sum(label_counts)
weighted_average = lambda x: round(np.sum(np.divide(x * label_counts, sum_label_counts)), 4)
print('Average precision of InceptionResNetV2 model = ' + str(weighted_average(class_precision)))
print('Average recall of InceptionResNetV2 model = ' + str(weighted_average(class_recall)))
```

```
Overall accuracy of InceptionResNetV2 model = 0.2554
Average precision of InceptionResNetV2 model = 0.0652
Average recall of InceptionResNetV2 model = 0.2554
```

```
confusion_matrix = metrics.confusion_matrix(test_labels, predicted)

plt.figure(figsize=(12, 9))
p = plt.imshow(np.log(np.divide(confusion_matrix + 1.0, np.sum(confusion_matrix, axis=1))))
cb = plt.colorbar(p)
_= cb.set_label('Log count')
```



Model 3 - Transfer learning (EfficientNetV2S) from pre-trained weights

Here we initialize the EfficientNetV2S model with pre-trained ImageNet weights, and we fine-tune it on our own dataset.

- The model is compiled using the Adam optimizer with a learning rate of 1e-2
- The categorical cross-entropy loss function is employed for multi-class classification tasks. Additionally, the model's performance is evaluated during training using standard accuracy metrics and a custom metric for top-3 accuracy

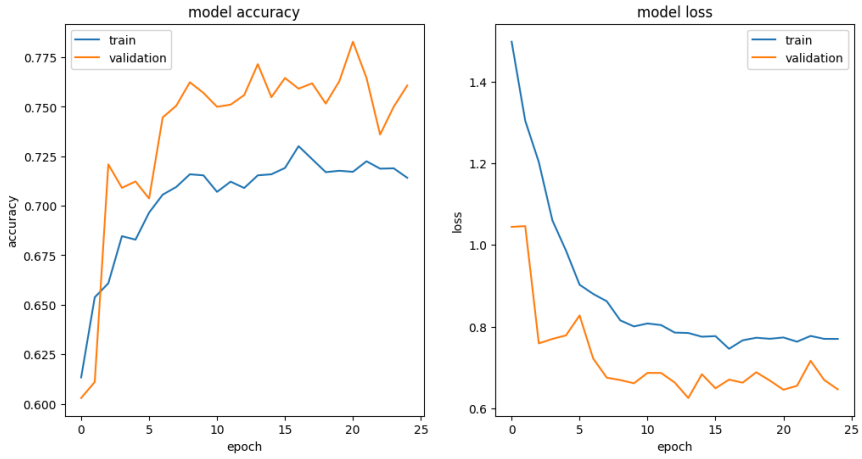
```
def build_model2(num_classes):
    inputs = layers.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
    model2 = EfficientNetV2S(include_top=False, input_tensor=inputs, weights="imagenet")

    # Freeze the pretrained weights
    model2.trainable = False

    # Rebuild top
    x = layers.GlobalAveragePooling2D(name="avg_pool")(model2.output)
    x = layers.BatchNormalization()(x)

    top_dropout_rate = 0.2
    x = layers.Dropout(top_dropout_rate, name="top_dropout")(x)
    outputs = layers.Dense(num_classes, activation="softmax", name="pred")(x)

    # Compile
    model2 = keras.Model(inputs, outputs, name="EfficientNetV2S")
    optimizer = keras.optimizers.Adam(learning_rate=1e-2)
    top_3_accuracy = TopKCategoricalAccuracy(k=3)
    model2.compile(
        optimizer=optimizer,
        loss="categorical_crossentropy",
        metrics=["accuracy", top_3_accuracy]
    )
    return model2
```

Transfer learning (EfficientNetV2S) Output and Insights:

- The training accuracy steadily increased from 0.6133 to 0.7141 over the 25 epochs, which is high suggests that the model effectively captured patterns and features from the training dataset
- The validation accuracy also demonstrated significant improvement, rising from 0.6029 to 0.7608. This is the best performing model among all models
- The top-k categorical accuracy for both training and validation sets also displayed a positive trend
- Towards the later epochs, the training and validation metrics stabilizez, suggesting that the model has converged to a relatively stable performance level
- The loss values for both training and validation sets decreased throughout the epochs, indicating that the model's parameters were optimized to minimize prediction errors

```
model2 = build_model2(num_classes=NUM_CLASSES)

epochs = 25
hist = model2.fit(ds_train, epochs=epochs, validation_data=ds_test)
plot_hist(hist)
```

```
88/88 [==============================] - 109s 1s/step - loss: 1.2030 - accurac
Epoch 4/25
88/88 [==============================] - 102s 1s/step - loss: 1.0600 - accurac
Epoch 5/25
88/88 [==============================] - 108s 1s/step - loss: 0.9862 - accurac
Epoch 6/25
88/88 [==============================] - 113s 1s/step - loss: 0.9026 - accurac
Epoch 7/25
88/88 [==============================] - 116s 1s/step - loss: 0.8801 - accurac
Epoch 8/25
88/88 [==============================] - 113s 1s/step - loss: 0.8621 - accurac
Epoch 9/25
88/88 [==============================] - 104s 1s/step - loss: 0.8152 - accurac
Epoch 10/25
88/88 [==============================] - 103s 1s/step - loss: 0.8005 - accurac
Epoch 11/25
88/88 [==============================] - 113s 1s/step - loss: 0.8077 - accurac
Epoch 12/25
88/88 [==============================] - 108s 1s/step - loss: 0.8038 - accurac
Epoch 13/25
88/88 [==============================] - 108s 1s/step - loss: 0.7854 - accurac
Epoch 14/25
88/88 [==============================] - 105s 1s/step - loss: 0.7844 - accurac
Epoch 15/25
88/88 [==============================] - 109s 1s/step - loss: 0.7753 - accurac
Epoch 16/25
88/88 [==============================] - 100s 1s/step - loss: 0.7767 - accurac
Epoch 17/25
88/88 [==============================] - 106s 1s/step - loss: 0.7457 - accurac
Epoch 18/25
88/88 [==============================] - 105s 1s/step - loss: 0.7664 - accurac
Epoch 19/25
88/88 [==============================] - 102s 1s/step - loss: 0.7728 - accurac
Epoch 20/25
88/88 [==============================] - 104s 1s/step - loss: 0.7702 - accurac
Epoch 21/25
88/88 [==============================] - 106s 1s/step - loss: 0.7734 - accurac
Epoch 22/25
88/88 [==============================] - 107s 1s/step - loss: 0.7631 - accurac
Epoch 23/25
88/88 [==============================] - 97s 1s/step - loss: 0.7772 - accuracy
Epoch 24/25
88/88 [==============================] - 101s 1s/step - loss: 0.7700 - accurac
Epoch 25/25
88/88 [==============================] - 102s 1s/step - loss: 0.7699 - accurac
```



```
one_hot_labels = [x for _,x in  ds_test.unbatch().as_numpy_iterator()]
test_labels = np.argmax(one_hot_labels, axis=1)
test_labels[:100]
```

```
array([0, 3, 2, 1, 3, 3, 3, 3, 2, 2, 3, 3, 3, 1, 3, 0, 2, 3, 3, 3, 1, 1,
       4, 1, 0, 2, 1, 2, 1, 2, 3, 3, 4, 3, 3, 0, 3, 1, 3, 2, 3, 3, 3, 1,
       2, 2, 3, 3, 3, 2, 3, 3, 1, 3, 2, 2, 4, 1, 3, 3, 1, 3, 4, 3, 3, 1,
       3, 1, 3, 2, 4, 3, 2, 2, 1, 3, 3, 3, 3, 1, 1, 1, 3, 3, 1, 1, 2, 1, 3,
       0, 3, 1, 3, 1, 4, 1, 3, 3, 1, 1, 3])
```

```
predictions = model2.predict(ds_test, batch_size=1)
predicted = predictions.argmax(axis=1)
predicted[:100]
```

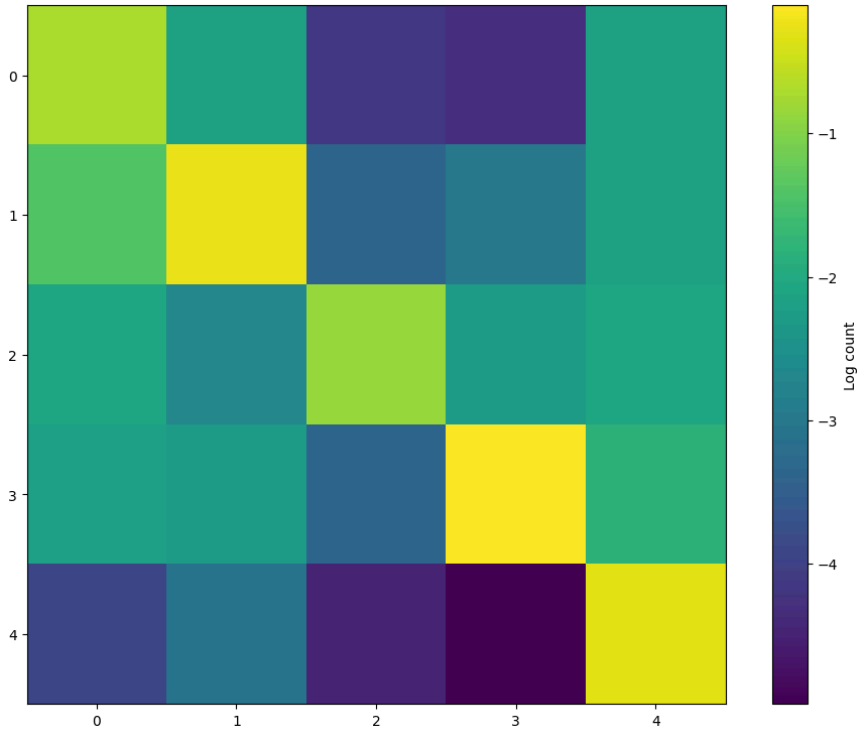
```
29/29 [==============================] - 25s 750ms/step
array([0, 3, 2, 1, 3, 3, 3, 3, 2, 3, 3, 3, 3, 1, 3, 0, 3, 3, 3, 3, 1, 1,
       4, 1, 1, 2, 1, 1, 3, 1, 3, 3, 4, 3, 3, 1, 3, 1, 3, 3, 3, 3, 3, 1,
       2, 2, 3, 3, 0, 4, 1, 3, 1, 3, 2, 1, 4, 0, 3, 3, 1, 3, 2, 1, 3, 0,
       3, 1, 3, 3, 4, 3, 2, 2, 0, 3, 1, 1, 1, 1, 1, 3, 3, 3, 1, 2, 1, 3,
       1, 3, 3, 3, 1, 4, 3, 3, 1, 1, 0, 3])
```

```
print('Overall accuracy of EfficientNetV2S model = ' + str(round(metrics.accuracy_score(test_labels, predicted), 4)))

unique_labels, label_counts = np.unique(test_labels, return_counts=True)
class_precision = metrics.precision_score(test_labels, predicted, labels=unique_labels, average=None)
class_recall = metrics.recall_score(test_labels, predicted, labels=unique_labels, average=None)

sum_label_counts = np.sum(label_counts)
weighted_average = lambda x: round(np.sum(np.divide(x * label_counts, sum_label_counts)), 4)
print('Average precision of EfficientNetV2S model = ' + str(weighted_average(class_precision)))
print('Average recall of EfficientNetV2S model = ' + str(weighted_average(class_recall)))
```

```
Overall accuracy of EfficientNetV2S model = 0.7608
Average precision of EfficientNetV2S model = 0.7659
Average recall of EfficientNetV2S model = 0.7608
```

The overall accuracy, average precision and average recall of this model is high with all three greater than 76%. From the confusion matrix below, we can confidently state that the EfficientNetV2S model is much better at correct classifications than other models explored

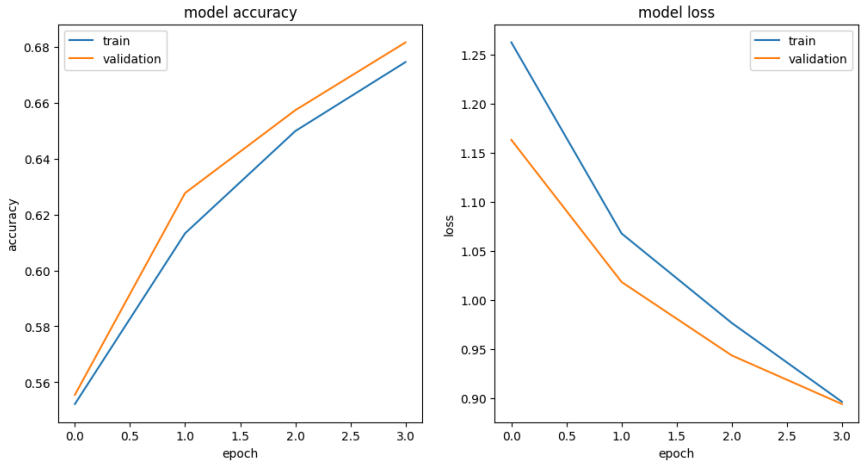
```
confusion_matrix = metrics.confusion_matrix(test_labels, predicted)

plt.figure(figsize = (12,9))
p = plt.imshow(np.log(np.divide(confusion_matrix + 1.0, np.sum(confusion_matrix, axis=1))))
cb = plt.colorbar(p)
_=cb.set_label('Log count')
```




```
def unfreeze_model(model):
    # Unfreeze the entire model
    model.trainable = True

    # We unfreeze the top 20 layers while leaving BatchNorm layers frozen
    for layer in model.layers[-20:]:
        if not isinstance(layer, layers.BatchNormalization):
            layer.trainable = True

    optimizer = keras.optimizers.Adam(learning_rate=1e-5)
    model.compile(
        optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"]
    )

# Unfreeze the model
unfreeze_model(model2)
```

```
epochs = 4
top_k_accuracy = TopKCategoricalAccuracy(k=3)
optimizer = keras.optimizers.Adam(learning_rate=1e-5)

# Compile the model with top_k_accuracy as a metric and Adam optimizer
model2.compile(
    optimizer=optimizer,
    loss="categorical_crossentropy",
    metrics=["accuracy", top_k_accuracy]
)

hist = model2.fit(ds_train, epochs=epochs, validation_data=ds_test)
plot_hist(hist)
```

```
Epoch 1/4
88/88 [==============================] - 522s 5s/step - loss: 1.2626 - accura
Epoch 2/4
88/88 [==============================] - 485s 6s/step - loss: 1.0678 - accura
Epoch 3/4
88/88 [==============================] - 489s 6s/step - loss: 0.9765 - accura
Epoch 4/4
88/88 [==============================] - 476s 5s/step - loss: 0.8961 - accura
```




```
one_hot_labels = [x for _,x in  ds_test.unbatch().as_numpy_iterator()]
test_labels = np.argmax(one_hot_labels, axis=1)
test_labels[:100]
```

```
array([0, 3, 2, 1, 3, 3, 3, 3, 2, 2, 3, 3, 3, 1, 3, 0, 2, 3, 3, 3, 1, 1,
       4, 1, 0, 2, 1, 2, 1, 2, 3, 3, 4, 3, 3, 0, 3, 1, 3, 2, 3, 3, 3, 1,
       2, 2, 3, 3, 3, 2, 3, 3, 1, 3, 2, 2, 4, 1, 3, 3, 1, 3, 4, 3, 3, 1,
       3, 1, 3, 2, 4, 3, 2, 2, 1, 3, 3, 3, 1, 1, 1, 3, 3, 1, 1, 2, 1, 3,
       0, 3, 1, 3, 1, 4, 1, 3, 3, 1, 1, 3])
```

```
predictions = model2.predict(ds_test, batch_size=1)
predicted = predictions.argmax(axis=1)
predicted[:100]
```

```
29/29 [==============================] - 30s 929ms/step
array([4, 3, 2, 1, 3, 3, 3, 3, 2, 3, 3, 3, 3, 1, 3, 0, 3, 3, 3, 3, 1, 1,
       4, 3, 1, 4, 1, 4, 1, 0, 3, 3, 4, 3, 3, 1, 3, 1, 2, 3, 3, 3, 3, 1,
       2, 4, 1, 3, 3, 1, 1, 3, 1, 1, 3, 1, 4, 2, 3, 3, 1, 3, 3, 1, 3, 1,
       3, 1, 3, 3, 4, 3, 2, 1, 0, 3, 1, 1, 4, 1, 1, 2, 3, 3, 1, 2, 1, 3,
       1, 3, 3, 3, 3, 4, 1, 3, 3, 1, 1, 3])
```

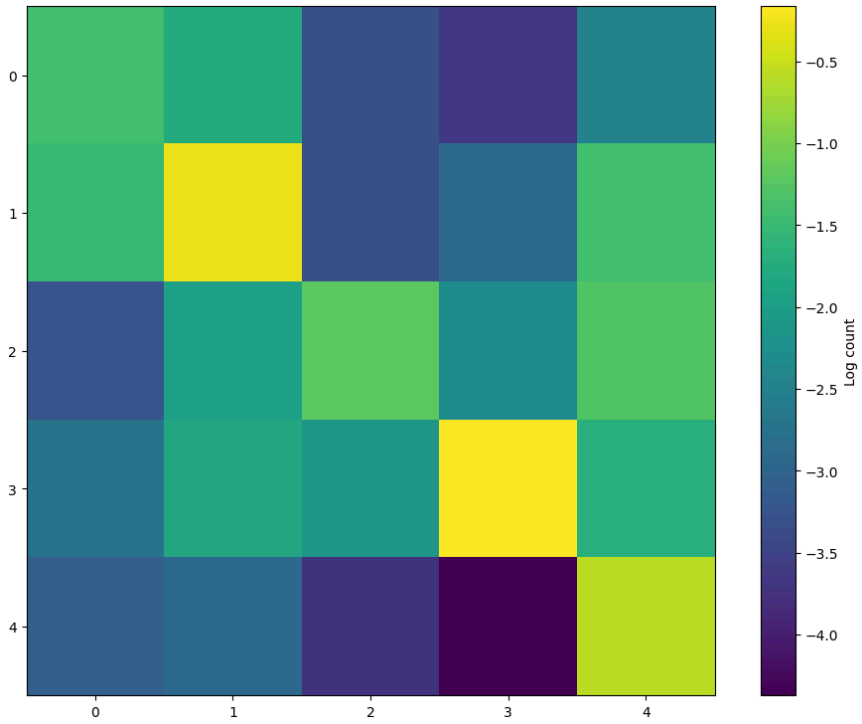
Insights from unfreezing the EfficientNetV2S model

We observe that the overall accuracy, as well as the average precision and recall decreased from ~76% to ~68% on unfreezing the EfficientNetV2S model

```
print('Overall accuracy of EfficientNetV2S model = ' + str(round(metrics.accuracy_score(test_labels, predicted), 4)))

unique_labels, label_counts = np.unique(test_labels, return_counts=True)
class_precision = metrics.precision_score(test_labels, predicted, labels=unique_labels, average=None)
class_recall = metrics.recall_score(test_labels, predicted, labels=unique_labels, average=None)

sum_label_counts = np.sum(label_counts)
weighted_average = lambda x: round(np.sum(np.divide(x * label_counts, sum_label_counts)), 4)
print('Average precision of EfficientNetV2S model = ' + str(weighted_average(class_precision)))
print('Average recall of EfficientNetV2S model = ' + str(weighted_average(class_recall)))
```

```
Overall accuracy of EfficientNetV2S model = 0.6816
Average precision of EfficientNetV2S model = 0.6731
Average recall of EfficientNetV2S model = 0.6816
```

```
confusion_matrix = metrics.confusion_matrix(test_labels, predicted)

plt.figure(figsize = (12,9))
p = plt.imshow(np.log(np.divide(confusion_matrix + 1.0, np.sum(confusion_matrix, axis=1))))
cb = plt.colorbar(p)
_=cb.set_label('Log count')
```



Model 4 - Transfer learning (MobileNetV2) from pre-trained weights

Here we initialize the MobileNetV2 model with pre-trained ImageNet weights, and we fine-tune it on our own dataset.

- The model is compiled using the Adam optimizer with a learning rate of 1e-2
- The categorical cross-entropy loss function is employed for multi-class classification tasks. Additionally, the model's performance is evaluated during training using standard accuracy metrics and a custom metric for top-3 accuracy

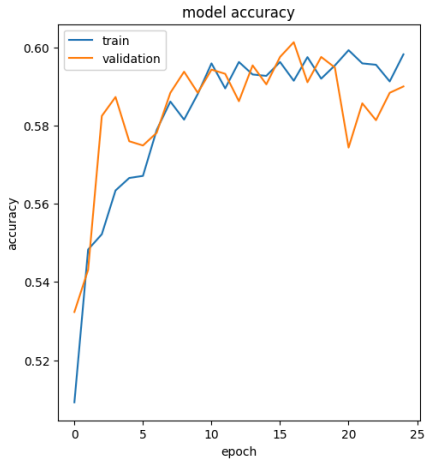
```
def build_model3(num_classes):
    inputs = layers.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
    model3 = MobileNetV2(include_top=False, input_tensor=inputs)  # Use MobileNetV2 model

    # Freeze the pretrained weights
    model3.trainable = False

    # Rebuild top
    x = layers.GlobalAveragePooling2D(name="avg_pool")(model3.output)
    x = layers.BatchNormalization()(x)

    top_dropout_rate = 0.2
    x = layers.Dropout(top_dropout_rate, name="top_dropout")(x)
    outputs = layers.Dense(num_classes, activation="softmax", name="pred")(x)

    # Compile
    model3 = keras.Model(inputs, outputs, name="MobileNetV2")
    optimizer = keras.optimizers.Adam(learning_rate=1e-2)
    top_3_accuracy = TopKCategoricalAccuracy(k=3)
    model3.compile(
        optimizer=optimizer,
        loss="categorical_crossentropy",
        metrics=["accuracy", top_3_accuracy]
    )
    return model3
```

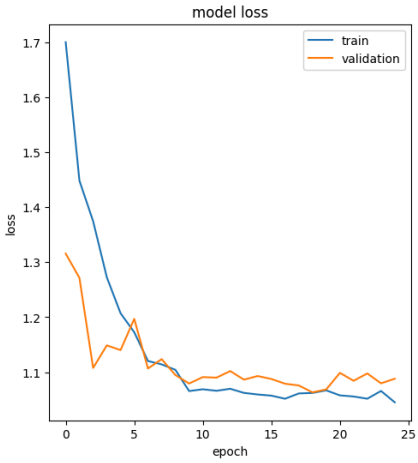
Transfer learning (MobileNetV2) Output and Insights:

- The training accuracy showed a slight increase from 0.5092 to 0.5982 over the 25 epochs, indicating moderate improvement in the model's ability to correctly classify training samples
- The validation accuracy also exhibited a similar trend, rising from 0.5323 to 0.59. While the validation accuracy improved, it remained relatively low suggesting potential issues with generalization to unseen data
- The top-k categorical accuracy for both training and validation sets showed an upward trend
- The training and validation metrics appeared to stabilize towards the later epochs
- The loss values for both training and validation sets fluctuated but generally decreased over the epochs, indicating that the model's parameters were optimized to minimize prediction errors. However, the validation loss remained relatively high compared to the training loss, indicating potential overfitting

```
model3 = build_model3(num_classes=NUM_CLASSES)

epochs = 25
hist = model3.fit(ds_train, epochs=epochs, validation_data=ds_test)
plot_hist(hist)
```

```
Epoch 3/25
88/88 [==============================] - 48s 541ms/step - loss: 1.3745 - accu
Epoch 4/25
88/88 [==============================] - 49s 552ms/step - loss: 1.2725 - accu
Epoch 5/25
88/88 [==============================] - 47s 534ms/step - loss: 1.2072 - accu
Epoch 6/25
88/88 [==============================] - 47s 535ms/step - loss: 1.1725 - accu
Epoch 7/25
88/88 [==============================] - 47s 534ms/step - loss: 1.1202 - accu
Epoch 8/25
88/88 [==============================] - 47s 539ms/step - loss: 1.1141 - accu
Epoch 9/25
88/88 [==============================] - 48s 547ms/step - loss: 1.1042 - accu
Epoch 10/25
88/88 [==============================] - 48s 549ms/step - loss: 1.0656 - accu
Epoch 11/25
88/88 [==============================] - 48s 548ms/step - loss: 1.0688 - accu
Epoch 12/25
88/88 [==============================] - 48s 549ms/step - loss: 1.0661 - accu
Epoch 13/25
88/88 [==============================] - 48s 548ms/step - loss: 1.0697 - accu
Epoch 14/25
88/88 [==============================] - 48s 541ms/step - loss: 1.0622 - accu
Epoch 15/25
88/88 [==============================] - 49s 555ms/step - loss: 1.0594 - accu
Epoch 16/25
88/88 [==============================] - 48s 541ms/step - loss: 1.0572 - accu
Epoch 17/25
88/88 [==============================] - 48s 544ms/step - loss: 1.0516 - accu
Epoch 18/25
88/88 [==============================] - 48s 550ms/step - loss: 1.0612 - accu
Epoch 19/25
88/88 [==============================] - 49s 560ms/step - loss: 1.0622 - accu
Epoch 20/25
88/88 [==============================] - 49s 555ms/step - loss: 1.0669 - accu
Epoch 21/25
88/88 [==============================] - 49s 559ms/step - loss: 1.0577 - accu
Epoch 22/25
88/88 [==============================] - 48s 549ms/step - loss: 1.0557 - accu
Epoch 23/25
88/88 [==============================] - 49s 558ms/step - loss: 1.0517 - accu
Epoch 24/25
88/88 [==============================] - 49s 558ms/step - loss: 1.0658 - accu
Epoch 25/25
88/88 [==============================] - 49s 560ms/step - loss: 1.0451 - accu
```





```
one_hot_labels = [x for _,x in  ds_test.unbatch().as_numpy_iterator()]
test_labels = np.argmax(one_hot_labels, axis=1)
test_labels[:100]
```

```
array([0, 3, 2, 1, 3, 3, 3, 3, 2, 2, 3, 3, 3, 1, 3, 0, 2, 3, 3, 3, 1, 1,
       4, 1, 0, 2, 1, 2, 1, 2, 3, 3, 4, 3, 3, 0, 3, 1, 3, 2, 3, 3, 3, 1,
       2, 2, 3, 3, 3, 2, 3, 3, 1, 3, 2, 2, 4, 1, 3, 3, 1, 3, 4, 3, 3, 1,
       3, 1, 3, 2, 4, 3, 2, 2, 1, 3, 3, 3, 3, 1, 1, 1, 3, 3, 1, 1, 2, 1, 3,
       0, 3, 1, 3, 1, 4, 1, 3, 3, 1, 1, 3])
```

```
predictions = model3.predict(ds_test, batch_size=1)
predicted = predictions.argmax(axis=1)
predicted[:100]
```

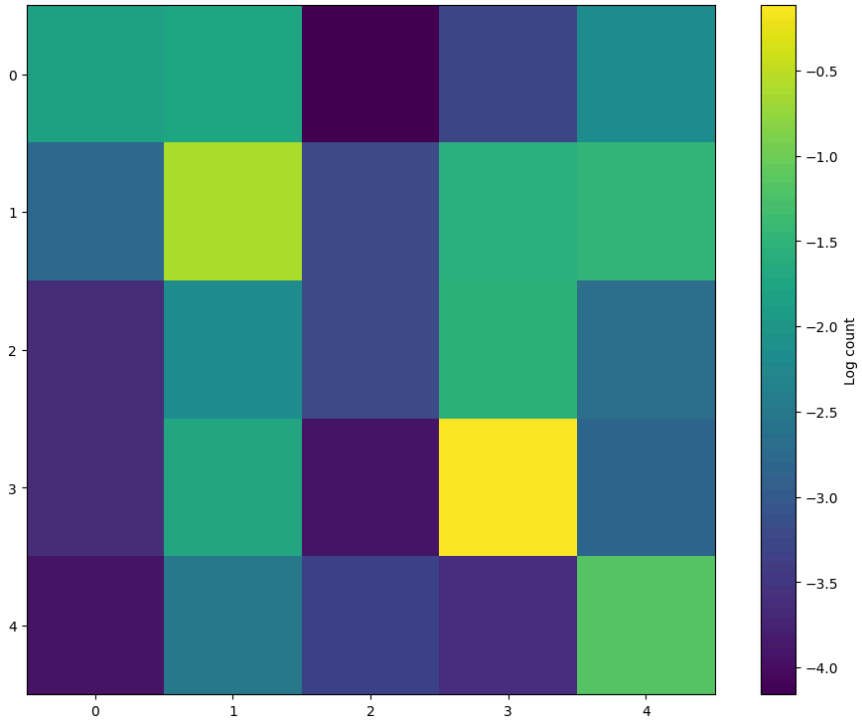
```
29/29 [==============================] - 12s 392ms/step
array([3, 3, 1, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 1, 3, 3, 3, 3, 3, 3,
       2, 3, 3, 3, 3, 3, 1, 3, 1, 3, 3, 1, 3, 3, 1, 3, 1, 3, 1, 3, 3, 3, 1,
       1, 3, 3, 1, 1, 1, 1, 3, 1, 1, 3, 3, 4, 2, 3, 1, 1, 3, 3, 3, 3, 1,
       3, 1, 3, 3, 1, 3, 3, 3, 3, 3, 1, 3, 1, 3, 1, 3, 3, 3, 3, 3, 1, 3,
       1, 3, 3, 3, 3, 1, 3, 3, 3, 2, 1, 3])
```

```
print('Overall accuracy of MobileNetV2 model = ' + str(round(metrics.accuracy_score(test_labels, predicted), 4)))

unique_labels, label_counts = np.unique(test_labels, return_counts=True)
class_precision = metrics.precision_score(test_labels, predicted, labels=unique_labels, average=None)
class_recall = metrics.recall_score(test_labels, predicted, labels=unique_labels, average=None)

sum_label_counts = np.sum(label_counts)
weighted_average = lambda x: round(np.sum(np.divide(x * label_counts, sum_label_counts)), 4)
print('Average precision of MobileNetV2 model = ' + str(weighted_average(class_precision)))
print('Average recall of MobileNetV2 model = ' + str(weighted_average(class_recall)))
```

```
Overall accuracy of MobileNetV2 model = 0.59
Average precision of MobileNetV2 model = 0.5408
Average recall of MobileNetV2 model = 0.59
```

The overall accuracy (59%), average precision (54.08%) and average recall (59%) of this model is average. From the confusion matrix below, we can see that some of the labels are not correctly classified. The EfficientNetV2S model is much better at correct classifications than this model

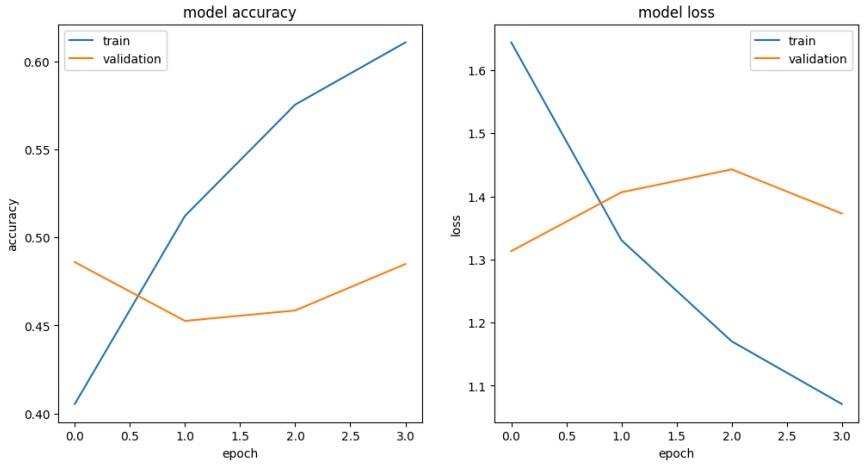
```
confusion_matrix = metrics.confusion_matrix(test_labels, predicted)

plt.figure(figsize = (12,9))
p = plt.imshow(np.log(np.divide(confusion_matrix + 1.0, np.sum(confusion_matrix, axis=1))))
cb = plt.colorbar(p)
_=cb.set_label('Log count')
```




```
def unfreeze_model(model):
    # Unfreeze the entire model
    model.trainable = True

    # We unfreeze the top 20 layers while leaving BatchNorm layers frozen
    for layer in model.layers[-20:]:
        if not isinstance(layer, layers.BatchNormalization):
            layer.trainable = True

    optimizer = keras.optimizers.Adam(learning_rate=1e-5)
    model.compile(
        optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"]
    )

# Unfreeze the model
unfreeze_model(model3)
```

```
epochs = 4
top_k_accuracy = TopKCategoricalAccuracy(k=3)
optimizer = keras.optimizers.Adam(learning_rate=1e-5)

# Compile the model with top_k_accuracy as a metric and Adam optimizer
model3.compile(
    optimizer=optimizer,
    loss="categorical_crossentropy",
    metrics=["accuracy", top_k_accuracy]
)

hist = model3.fit(ds_train, epochs=epochs, validation_data=ds_test)
plot_hist(hist)
```

```
Epoch 1/4
88/88 [==============================] - 170s 2s/step - loss: 1.6440 - accura
Epoch 2/4
88/88 [==============================] - 158s 2s/step - loss: 1.3306 - accura
Epoch 3/4
88/88 [==============================] - 157s 2s/step - loss: 1.1703 - accura
Epoch 4/4
88/88 [==============================] - 160s 2s/step - loss: 1.0709 - accura
```




```
one_hot_labels = [x for _,x in  ds_test.unbatch().as_numpy_iterator()]
test_labels = np.argmax(one_hot_labels, axis=1)
test_labels[:100]
```

```
array([0, 3, 2, 1, 3, 3, 3, 3, 2, 2, 3, 3, 3, 1, 3, 0, 2, 3, 3, 3, 1, 1,
       4, 1, 0, 2, 1, 2, 1, 2, 3, 3, 4, 3, 3, 0, 3, 1, 3, 2, 3, 3, 3, 1,
       2, 2, 3, 3, 3, 2, 3, 3, 1, 3, 2, 2, 4, 1, 3, 3, 1, 3, 4, 3, 3, 1,
       3, 1, 3, 2, 4, 3, 2, 2, 1, 3, 3, 3, 1, 1, 1, 3, 3, 1, 1, 2, 1, 3,
       0, 3, 1, 3, 1, 4, 1, 3, 3, 1, 1, 3])
```

```
predictions = model3.predict(ds_test, batch_size=1)
predicted = predictions.argmax(axis=1)
predicted[:100]
```

```
29/29 [==============================] - 13s 413ms/step
array([4, 0, 0, 1, 0, 0, 3, 2, 0, 1, 0, 2, 4, 4, 3, 2, 0, 0, 3, 4, 0, 3,
       4, 3, 3, 4, 3, 0, 3, 1, 3, 3, 4, 3, 3, 0, 3, 4, 3, 3, 3, 3, 3, 1,
       0, 4, 0, 3, 3, 0, 3, 3, 1, 3, 2, 4, 4, 4, 3, 0, 0, 2, 2, 1, 3, 3,
       3, 1, 3, 3, 2, 3, 2, 2, 1, 2, 1, 2, 3, 1, 1, 0, 3, 3, 3, 0, 1, 3,
       1, 3, 2, 3, 4, 4, 1, 1, 4, 1, 1, 1])
```

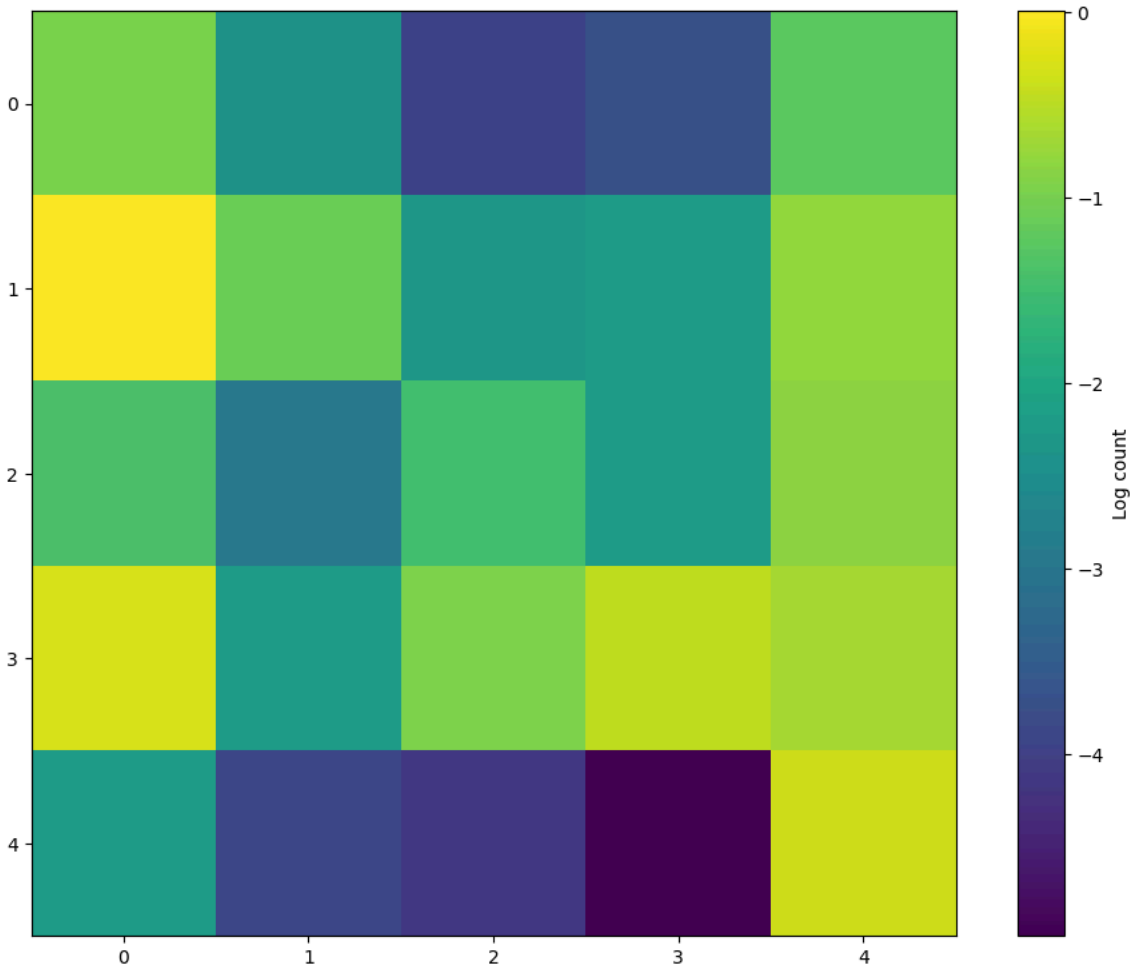
Insights from unfreezing the EfficientNetV2S model

We observe that the overall accuracy, as well as the average recall decreased from ~59% to ~48%, however average precion increased slightlt from ~54% to ~55% on unfreezing the MobileNetV2 model

```
print('Overall accuracy of MobileNetV2 model = ' + str(round(metrics.accuracy_score(test_labels, predicted), 4)))

unique_labels, label_counts = np.unique(test_labels, return_counts=True)
class_precision = metrics.precision_score(test_labels, predicted, labels=unique_labels, average=None)
class_recall = metrics.recall_score(test_labels, predicted, labels=unique_labels, average=None)

sum_label_counts = np.sum(label_counts)
weighted_average = lambda x: round(np.sum(np.divide(x * label_counts, sum_label_counts)), 4)
print('Average precision of MobileNetV2 model = ' + str(weighted_average(class_precision)))
print('Average recall of MobileNetV2 model = ' + str(weighted_average(class_recall)))
```

```
Overall accuracy of MobileNetV2 model = 0.4849
Average precision of MobileNetV2 model = 0.5539
Average recall of MobileNetV2 model = 0.4849
```

```
confusion_matrix = metrics.confusion_matrix(test_labels, predicted)

plt.figure(figsize = (12,9))
p = plt.imshow(np.log(np.divide(confusion_matrix + 1.0, np.sum(confusion_matrix, axis=1))))
cb = plt.colorbar(p)
_=cb.set_label('Log count')
```



```
import numpy as np
import tensorflow_datasets as tfds
import tensorflow as tf
import matplotlib.pyplot as plt
import keras
from keras import layers
import pandas as pd
from time import time
from datetime import datetime
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, TensorBoard, CSVLogger
from keras.optimizers import Adam
from keras.metrics import TopKCategoricalAccuracy
from keras.models import Model, load_model
from keras.layers import Lambda
import sklearn.metrics as metrics
from sklearn.metrics import confusion_matrix, classification_report
from keras import backend as K
from scipy.io import savemat
from scipy.interpolate import splprep, splev, splrep, interp1d
import matplotlib.image as mpimg
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import animation
%matplotlib inline
```

Model comparison and analysis

```
models = ['EfficientNetB3', 'InceptionResNetV2', 'EfficientNetV2S', 'MobileNetV2']

accuracy = [0.4688, 0.5253, 0.7608, 0.59]
precision = [0.2197, 0.4008, 0.7659, 0.5408]
recall = [0.4688, 0.5253, 0.7608, 0.59]
```

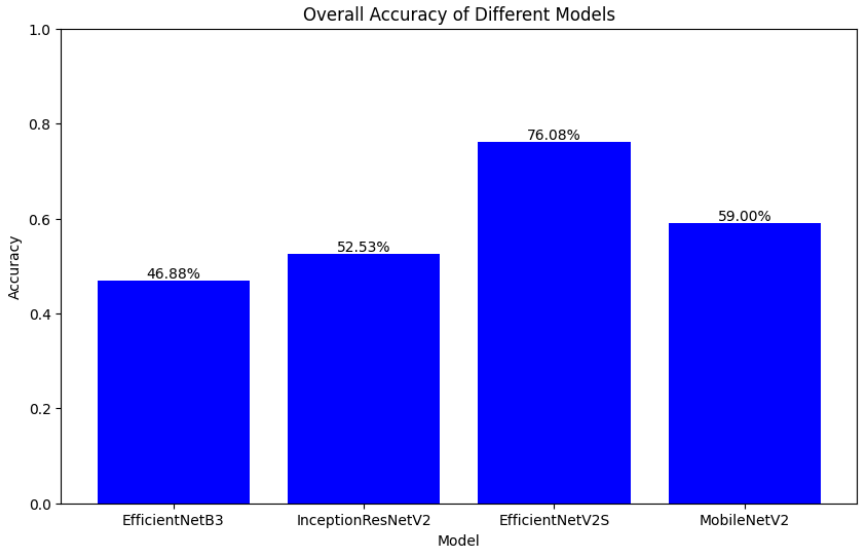
On creating a table output of accuracy, precision and recall for all four models, we observe that the EfficientNetV2S model is the best performer for our task of plant disease identification. The next best performance are from MobileNetV2 and InceptionResNetV2. EfficientNetB3 performed the worst among all models

```
# Convert numbers to string format with percentage
accuracy_str = [f"{acc*100:.2f}%" for acc in accuracy]
precision_str = [f"{prec*100:.2f}%" for prec in precision]
recall_str = [f"{rec*100:.2f}%" for rec in recall]

# Create a dictionary with the data
data = {
    'Model': models,
    'Accuracy': accuracy_str,
    'Precision': precision_str,
    'Recall': recall_str
}

df = pd.DataFrame(data)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.colheader_justify', 'center')
pd.set_option('display.max_rows', None)

styled_df = df.style.set_properties(**{'color': 'black', 'border-color': 'black', 'font-weight': 'bold'})
styled_df
```

	Model	Accuracy	Precision	Recall
0	EfficientNetB3	46.88%	21.97%	46.88%
1	InceptionResNetV2	52.53%	40.08%	52.53%
2	EfficientNetV2S	76.08%	76.59%	76.08%
3	MobileNetV2	59.00%	54.08%	59.00%

Bar Chart comparison for models

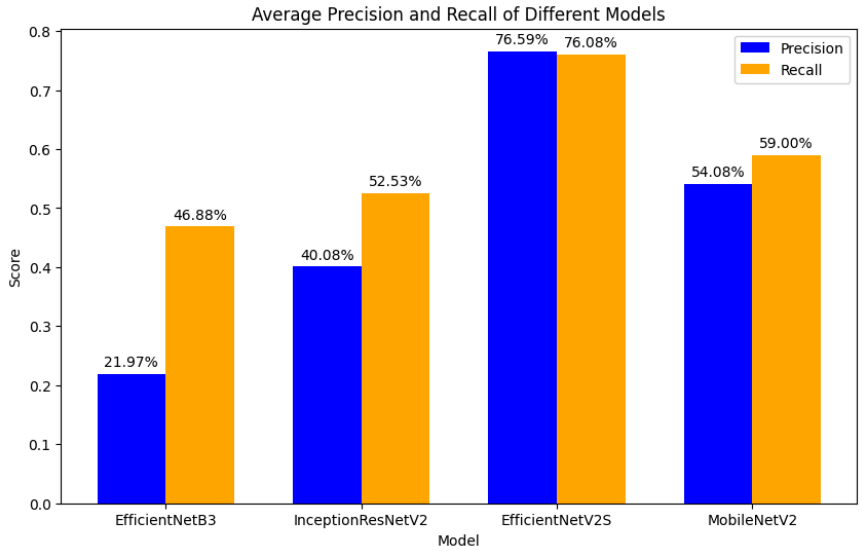
```
plt.figure(figsize=(10, 6))
bars = plt.bar(models, accuracy, color='blue')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Overall Accuracy of Different Models')
plt.ylim(0, 1)

for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height, f'{height*100:.2f}%', ha='center', va='bottom')

plt.show()
```



Precision and Recall comparsion for models

```
# Define colors for the bars
colors = ['blue'] * len(models)

# Set up the positions for the bars
x = np.arange(len(models))

# Width of the bars
width = 0.35

# Create the clustered bar chart
fig, ax = plt.subplots(figsize=(10, 6))
bars_precision = ax.bar(x - width/2, precision, width, color=colors, label='Precision')
bars_recall = ax.bar(x + width/2, recall, width, color='orange', label='Recall')

# Add labels, title, and legend
ax.set_xlabel('Model')
ax.set_ylabel('Score')
ax.set_title('Average Precision and Recall of Different Models')
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()

# Add percentage labels for precision
for bar in bars_precision:
    height = bar.get_height()
    ax.annotate(f'{height*100:.2f}%',
                xy=(bar.get_x() + bar.get_width() / 2, height),
                xytext=(0, 3),  # 3 points vertical offset
                textcoords="offset points",
                ha='center', va='bottom')

# Add percentage labels for recall
for i, bar in enumerate(bars_recall):
    height = bar.get_height()
    ax.annotate(f'{height*100:.2f}%',
```

```
                xy=(bar.get_x() + bar.get_width() / 2, height),
                xytext=(0, 3),  # 3 points vertical offset
                textcoords="offset points",
                ha='center', va='bottom')

plt.show()
```



Final Model chosen

Based on the overall performance on accuracy, precision, recall and confusion matrix, I decided to choose the EfficientNetV2S for conducting future work on plant disease identification

Particle Swarm Optimization (PSO) setup for ground robots:

The below code blocks if for creating a Particle Swarm Optimization (PSO) algorithm to optimize the trajectory of a ground robots by minimizing the path length and energy consumption while avoiding obstacles in the environment. The PSO algorithm also searches for an optimal solution by adjusting the velocity and position of particles in a multidimensional search space.

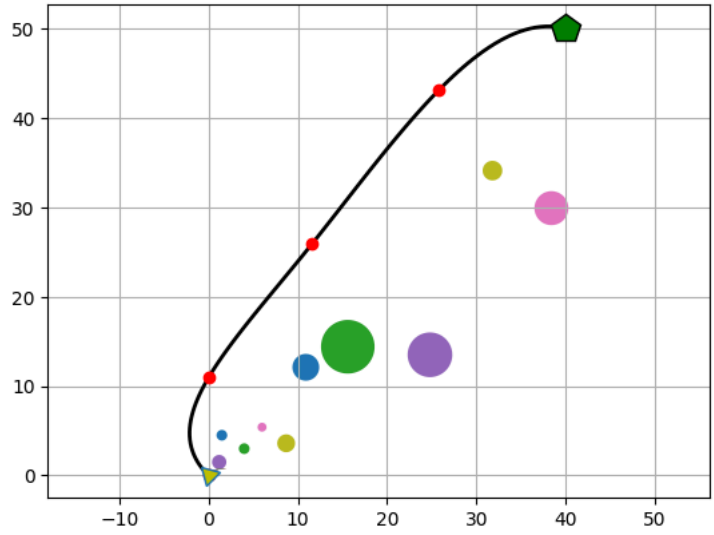
```
def create_model():

    # Source
    xs = 0
    ys = 0
    zs = 0

    # Target (Destination)
    xt = 40
    yt = 50
    zt = 5

    xobs = [1.5, 4.0, 1.2, 6.0, 8.7, 10.9, 15.6, 24.8, 38.4, 31.8]
    yobs = [4.5, 3.0, 1.5, 5.4, 3.6, 12.1, 14.4, 13.5, 29.9, 34.1]
    zobs = [4.5, 3.0, 1.5, 2.2, 3.3, 0.5, 1.9, 4.8, -0.8, -1.6]
    robs = [0.6, 0.6, 0.8, 0.5, 1.0, 1.5, 3.0, 2.5, 1.9, 1.1]

    n = 10

    xmin = 0
    xmax = 100

    ymin = 0
    ymax = 100

    zmin = -10
    zmax = 10

    model = {
        "xs": xs,
        "ys": ys,
        "zs": zs,
        "xt": xt,
        "yt": yt,
        "zt": zt,
        "xobs": xobs,
        "yobs": yobs,
        "zobs": zobs,
        "robs": robs,
        "n": n,
        "xmin": xmin,
        "xmax": xmax,
        "ymin": ymin,
        "ymax": ymax,
        "zmin": zmin,
        "zmax": zmax,
    }

    return model


def create_random_solution(model):

    n = model["n"]

    xmin = model["xmin"]
    xmax = model["xmax"]

    ymin = model["ymin"]
    ymax = model["ymax"]

    sol1 = {
        "x": np.random.uniform(xmin, xmax, size=n),
        "y": np.random.uniform(ymin, ymax, size=n),
    }

    return sol1



def my_cost(sol1, model):

    sol = parse_solution(sol1, model)

    beta = 100
    z = sol["L"] * (1 + beta * sol["Violation"])

    return z, sol
```

```
def parse_solution(sol1, model):

    x = sol1["x"]
    y = sol1["y"]

    xs = model["xs"]
    ys = model["ys"]
    xt = model["xt"]
    yt = model["yt"]
    xobs = model["xobs"]
    yobs = model["yobs"]
    robs = model["robs"]

    XS = [xs] + x.tolist() + [xt]
    YS = [ys] + y.tolist() + [yt]

    k = len(XS)
    TS = np.linspace(0, 1, k)

    tt = np.linspace(0, 1, 100)
    xx = splrep(TS, XS, k=3)
    yy = splrep(TS, YS, k=3)
    x_interpolated = splev(tt, xx)
    y_interpolated = splev(tt, yy)

    dx = np.diff(x_interpolated)
    dy = np.diff(y_interpolated)

    L = np.sqrt(np.sum(np.square(dx) + np.square(dy)))

    nobs = len(xobs)
    Violation = 0
    for k in range(nobs):
        d = np.sqrt((x_interpolated - xobs[k])**2 + (y_interpolated - yobs[k])**2)
        v = np.maximum(1 - d/robs[k], 0)
        Violation += np.mean(v)

    sol2 = {
        "TS": TS,
        "XS": XS,
        "YS": YS,
        "tt": np.linspace(0, 1, 100),
        "xx": x_interpolated,
        "yy": y_interpolated,
        "dx": dx,
        "dy": dy,
        "L": L,
        "Violation": Violation,
        "IsFeasible": (Violation == 0),
    }

    return sol2


def PlotSolution(sol, model):
    xs = model['xs']
    ys = model['ys']
    xt = model['xt']
    yt = model['yt']
    xobs = model['xobs']
    yobs = model['yobs']
    robs = model['robs']

    XS = sol['XS']
    YS = sol['YS']
    xx = sol['xx']
    yy = sol['yy']

    theta = np.linspace(0, 2*np.pi, 100)
    for k in range(len(xobs)):
        plt.fill(xobs[k] + robs[k]*np.cos(theta), yobs[k] + robs[k]*np.sin(theta), [0.5, 0.7, 0.8])

    plt.plot(xx, yy, 'k', linewidth=2)
    plt.plot(XS, YS, 'ro')
    plt.plot(xs, ys, 'bs', markersize=12, markerfacecolor='y')
    plt.plot(xt, yt, 'kp', markersize=16, markerfacecolor='g')
    plt.grid(True)
    plt.axis('equal')
    plt.show()
```

```
def PlotSolution2(sol, model):

    fig, ax = plt.subplots()
    xs = model['xs']
    ys = model['ys']
    xt = model['xt']
    yt = model['yt']
    xobs = model['xobs']
    yobs = model['yobs']
    robs = model['robs']

    XS = sol['XS']
    YS = sol['YS']
    xx = sol['xx']
    yy = sol['yy']

    ax.clear()

    theta = np.linspace(0, 2*np.pi, 100)
    for k in range(len(xobs)):
        ax.fill(xobs[k] + robs[k]*np.cos(theta), yobs[k] + robs[k]*np.sin(theta), [0.5, 0.7, 0.8])


    ax.plot(xx, yy, 'k', linewidth=2)
    ax.plot(XS, YS, 'ro')
    ax.plot(xs, ys, marker=(3, 0, np.rad2deg(np.pi/4)), markersize=12, markerfacecolor='y')

    ax.plot(xt, yt, 'kp', markersize=16, markerfacecolor='g')
    ax.grid(True)
    ax.axis('equal')
    plt.show()


model = create_model()

model["n"] = 3  # number of Handle Points

CostFunction = lambda x: my_cost(x, model)    # Cost Function

nVar = model["n"]       # Number of Decision Variables

VarSize = (1, nVar)   # Size of Decision Variables Matrix

VarMin = {"x": model["xmin"], "y": model["ymin"]}          # Lower Bound of Variables
VarMax = {"x": model["xmax"], "y": model["ymax"]}          # Upper Bound of Variables


## PSO Parameters

MaxIt = 220           # Maximum Number of Iterations

nPop = 150           # Population Size (Swarm Size)

w = 1                # Inertia Weight
wdamp = 0.98         # Inertia Weight Damping Ratio
c1 = 1.5             # Personal Learning Coefficient
c2 = 1.5             # Global Learning Coefficient

alpha = 0.1
VelMax = {"x": alpha * (VarMax["x"] - VarMin["x"]), "y": alpha * (VarMax["y"] - VarMin["y"])}   # Maximum Velocity
VelMin = {"x": -VelMax["x"], "y": -VelMax["y"]}                                                 # Minimum Velocity

## Initialization

# Create Empty Particle Structure
empty_particle = {"Position": None,
                  "Velocity": None,
                  "Cost": None,
                  "Sol": None,
                  "Best": {"Position": None,
                           "Cost": None,
                           "Sol": None}}

# Initialize Global Best
GlobalBest = {"Cost": np.inf}

# Create Particles Matrix
particle = np.tile(empty_particle, (nPop, 1))
```

```
# create an empty particle list
particle = []

# Initialization Loop
for i in range(nPop):

    # Initialize Particle
    xx = np.array([0.1, 3,5.0])
    yy = np.array([0.1, 3,5.0])

    new_particle = {
        'Position': {'x': xx, 'y': yy},
        'Velocity': {'x': None, 'y': None},
        'Cost': None,
        'Sol': None,
        'Best': {
            'Position': {'x': None, 'y': None},
            'Cost': None,
            'Sol': None
        }
    }

    # Initialize Position
    if i > 0:
        solu_1 = create_random_solution(model)
        new_particle['Position'] = {'x': solu_1['x'], 'y': solu_1['y']}

    # Initialize Velocity
    new_particle['Velocity']['x'] = np.zeros(VarSize)
    new_particle['Velocity']['y'] = np.zeros(VarSize)

    # Evaluation
    new_particle['Cost'], new_particle['Sol'] = CostFunction(new_particle['Position'])

    # Update Personal Best
    new_particle['Best']['Position'] = new_particle['Position']
    new_particle['Best']['Cost'] = new_particle['Cost']
    new_particle['Best']['Sol'] = new_particle['Sol']

    # Add particle to the list
    particle.append(new_particle)

    # Update Global Best
    if new_particle['Best']['Cost'] < GlobalBest['Cost']:
        GlobalBest = new_particle['Best']
```

```
BestCost = np.zeros(MaxIt)
frames = []

for it in range(MaxIt):
    for i in range(nPop):
        # Update x part
        # Update velocity
        particle[i]['Velocity']['x'] = w*particle[i]['Velocity']['x'] \
            + c1*np.random.rand(*VarSize)*(particle[i]['Best']['Position']['x']-particle[i]['Position']['x']) \
            + c2*np.random.rand(*VarSize)*(GlobalBest['Position']['x']-particle[i]['Position']['x'])
        # Update velocity bounds
        particle[i] ['Velocity'] ['x'] = np.maximum(particle[i] ['Velocity'] ['x'], VelMin ['x'])
        particle[i] ['Velocity'] ['x'] = np.minimum(particle[i] ['Velocity'] ['x'], VelMax ['x'])
        # Update position
        particle[i] ['Position'] ['x'] = particle[i] ['Position'] ['x'] + particle[i] ['Velocity'] ['x']
        # Velocity mirroring
        OutOfTheRange = (particle[i] ['Position'] ['x']<VarMin ['x']) | (particle[i] ['Position'] ['x']>VarMax ['x'])
        particle[i] ['Velocity'] ['x'][OutOfTheRange] = -particle[i] ['Velocity'] ['x'][OutOfTheRange]
        # Update position bounds
        particle[i] ['Position'] ['x'] = np.maximum(particle[i] ['Position'] ['x'], VarMin ['x']).flatten()
        particle[i] ['Position'] ['x'] = np.minimum(particle[i] ['Position'] ['x'], VarMax ['x']).flatten()

        # Update y part
        # Update velocity
        particle[i] ['Velocity']['y'] = w*particle[i] ['Velocity']['y'] \
            + c1*np.random.rand(*VarSize)*(particle[i]['Best']['Position']['y']-particle[i] ['Position']['y']) \
            + c2*np.random.rand(*VarSize)*(GlobalBest['Position']['y']-particle[i] ['Position']['y'])
        # Update velocity bounds
        particle[i] ['Velocity']['y'] = np.maximum(particle[i] ['Velocity']['y'], VelMin['y'])
        particle[i] ['Velocity']['y'] = np.minimum(particle[i] ['Velocity']['y'], VelMax['y'])
        # Update position
        particle[i] ['Position']['y'] = particle[i] ['Position']['y'] + particle[i] ['Velocity']['y']
        # Velocity mirroring
        OutOfTheRange = (particle[i] ['Position']['y']<VarMin['y']) | (particle[i] ['Position']['y']>VarMax['y'])
        particle[i] ['Velocity']['y'][OutOfTheRange] = -particle[i] ['Velocity']['y'][OutOfTheRange]
        # Update position bounds
        particle[i] ['Position']['y'] = np.maximum(particle[i] ['Position']['y'], VarMin['y']).flatten()
        particle[i] ['Position']['y'] = np.minimum(particle[i] ['Position']['y'], VarMax['y']).flatten()

        # Evaluation
        particle[i]['Cost'], particle[i]['Sol'] = CostFunction(particle[i]['Position'])

        # Update personal best
        if particle[i]['Cost'] < particle[i]['Best']['Cost']:
            particle[i]['Best']['Position'] = particle[i] ['Position']
            particle[i]['Best']['Cost'] = particle[i]['Cost']
            particle[i]['Best']['Sol'] = particle[i]['Sol']

            # Update global best
            if particle[i]['Best']['Cost'] < GlobalBest['Cost']:
                GlobalBest = particle[i]['Best']

    # Update best cost ever found
    BestCost[it] = GlobalBest['Cost']

    # Inertia weight damping
    w = w*wdamp

    # Show iteration information
    if GlobalBest['Sol']['IsFeasible']:
        Flag = ' *'
    else:
        Flag = ', Violation = ' + str(GlobalBest['Sol']['Violation'])

PlotSolution2(GlobalBest['Sol'], model)
```



Final Reflections

This was a very interesting and challlenging problem to solve. I had to make sure to add additional pre-processing and data augmnentation techniques as the data set images were not uniformly distributed, with the two disease classes CMD and CBSD having 72% of the images.

In summary, two of the four models trained by me showed very good performance and, with some additional effort, they would be suitable for the task that I have set out to achieve.

Key highlights & learnings:

- The use of transfer learning is a critical enabler for success in computer vision tasks. Additionally, given that the image dataset was skewed, I have been able to achieve considerably high accuracy in this problem.
- Improvements in data augmentation (rotation, translation, and flipping; geometric transformations; contrast and brightness adjustments; Random gamma adjustment) lead to significant improvements in model performance.
- I have used complex model architectures in this project which have been an extremely fruitful learning experience.

Future Work:

- I would like to continue working on this project and try more image processing techniques to further improve model performance.
- I would like to test performace via Tranfer Learning on other models listed here (https://keras.io/api/applications/) apart from the 4 models I tested
- I would like to combine the CV model performance along with Particle Swarm Optimization (PSO) algorithm for ground robots and drones to check practical effectiveness of the project.

Submitted by: Bethun Bhowmik
Date: 05/06/2024