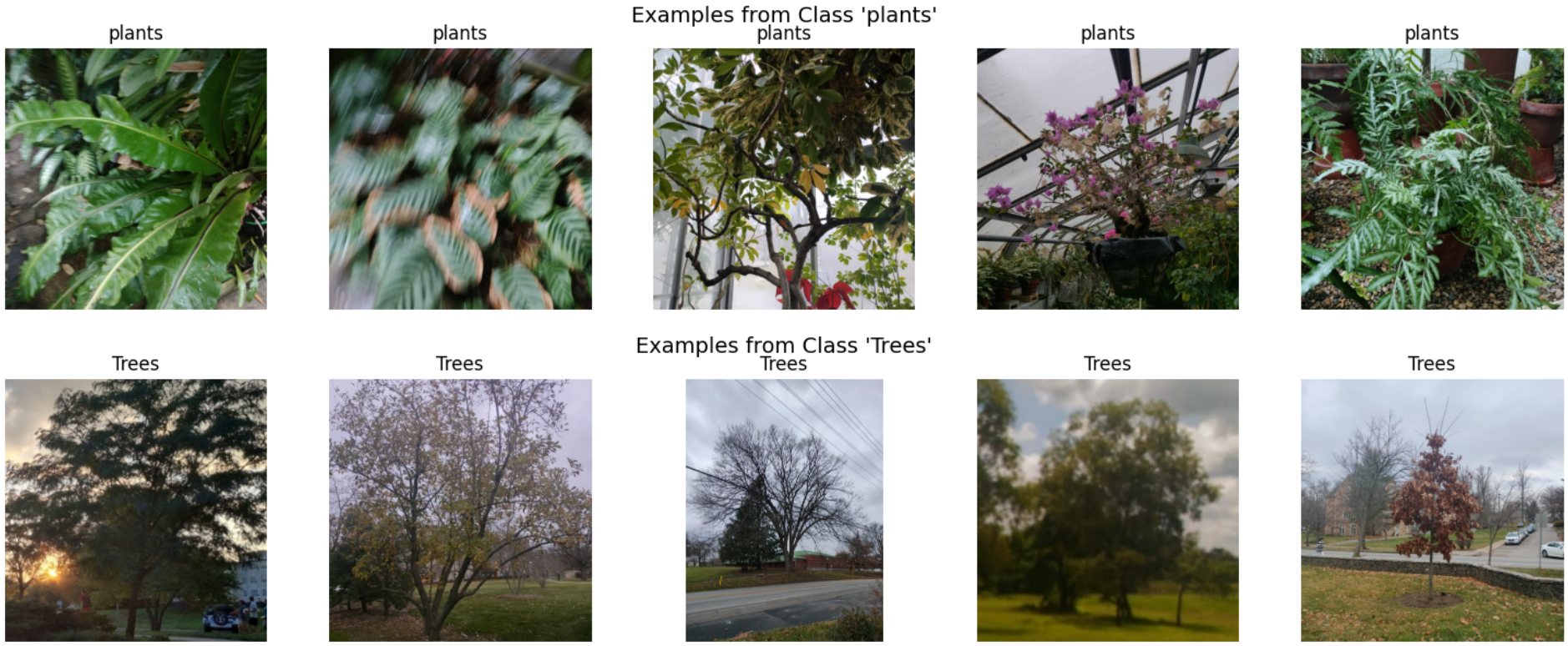
In [ ]:
```
import pandas as pd
import numpy as np
from google.colab import drive
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten, Dropout

from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
import numpy as np
from glob import glob
import matplotlib.pyplot as plt

# Mounting Drive
drive.mount('/content/drive')
```

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

1. Take at least 100 images per class with at least 3 classes using your phone/camera (e.g. take photos of different types of trees, flowers or animals). Display 5 examples from each class. [10 points]

In [ ]:
```
import matplotlib.pyplot as plt
import os
import random
from PIL import Image

# Assuming your dataset is structured with a separate folder for each class
dataset_path = '/content/drive/My Drive/Colab Notebooks/AML/AML_HW4/Dataset/Train' #'/content/drive/MyDrive/Dataset/Tra

# Number of images to display per class
num_images = 5

# Loop through each folder (class) in the dataset
for class_name in os.listdir(dataset_path):
    class_path = os.path.join(dataset_path, class_name)
```

```python
    # Check if it's a directory
    if os.path.isdir(class_path):
        # Get all image files in this directory
        image_files = [file for file in os.listdir(class_path) if file.endswith(('jpg', 'jpeg', 'png'))]

        # Randomly select 5 images (or fewer if not enough images)
        selected_images = random.sample(image_files, min(num_images, len(image_files)))

        # Display the images
        plt.figure(figsize=(15, 3))
        for i, image_file in enumerate(selected_images):
            img_path = os.path.join(class_path, image_file)
            img = Image.open(img_path)
            plt.subplot(1, num_images, i + 1)
            plt.imshow(img)
            plt.title(f"{class_name}")
            plt.axis("off")
        plt.suptitle(f"Examples from Class '{class_name}'", fontsize=14)
        plt.tight_layout()
        plt.subplots_adjust(top=0.85)  # Adjust the top margin
        plt.show()
```



Ans 1. -

As seen above, 100 images per 3 classes have been created. 5 of each have been displayed.

2. Split the images into a training set, a validation set, and a test set. [5 points]

   And,

3. Build the input pipeline, including the appropriate preprocessing operations, and add data augmentation. [10 points]

In [ ]:
```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os

# Initialize the data generator without any augmentation
datagen = ImageDataGenerator()

# Assuming your dataset is structured with a separate folder for each class
dataset_path = '/content/drive/My Drive/Colab Notebooks/AML/AML_HW4/Dataset/Train'# '/content/drive/MyDrive/Dataset/Tra

# Load the data using flow_from_directory
data_generator = datagen.flow_from_directory(
    dataset_path,
    target_size=(224, 224),  # Or any size you require
    batch_size=32,           # Batch size (can be any number since we're just counting)
    class_mode='categorical' # Assuming you're doing a categorical classification
)

# Count the images in each class
class_counts = {class_name: 0 for class_name in data_generator.class_indices.keys()}

for class_index in data_generator.classes:
    class_name = list(data_generator.class_indices.keys())[class_index]
    class_counts[class_name] += 1

# Print the counts
for class_name, count in class_counts.items():
    print(f"Class '{class_name}': {count} images")
```

```
Found 353 images belonging to 4 classes.
Class 'Cacti': 65 images
Class 'Succulents': 102 images
Class 'Trees': 64 images
Class 'plants': 122 images
```

In [ ]:
```python
# Pipeline & Splitting the Data

# Rescale all images by 1./255
Train_datagen = ImageDataGenerator(rescale=1./255,
                                   validation_split=0.2,
                                   rotation_range=40,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
```

```
                              fill_mode='nearest')

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

# Define the batch size
batch_size = 32

# Define the image size
target_size = (224, 224)

# Define the class mode
class_mode = 'categorical'

# Loading and splitting the training data for training and validation
train_generator = Train_datagen.flow_from_directory(
    '/content/drive/My Drive/Colab Notebooks/AML/AML_HW4/Dataset/Train', #'/content/drive/MyDrive/Dataset/Train',
    target_size=target_size,
    batch_size=batch_size,
    class_mode=class_mode,
    subset='training',  # Set as training data
    seed=42)  # For reproducibility

validation_generator = validation_test_datagen.flow_from_directory(
    '/content/drive/My Drive/Colab Notebooks/AML/AML_HW4/Dataset/Train',#'/content/drive/MyDrive/Dataset/Train',  # Sam
    target_size=target_size,
    batch_size=batch_size,
    class_mode=class_mode,
    subset='validation',  # Set as validation data
    seed=42)  # For reproducibility

# Loading the test data
test_generator = validation_test_datagen.flow_from_directory(
    '/content/drive/My Drive/Colab Notebooks/AML/AML_HW4/Dataset/Test', #'/content/drive/MyDrive/Dataset/Test',
    target_size=target_size,
    batch_size=batch_size,
    class_mode=class_mode)
```

```
Found 284 images belonging to 4 classes.
Found 69 images belonging to 4 classes.
Found 75 images belonging to 4 classes.
```

Ans 3. -

- In the above cell we have built a Pipeline to Augment & Pre-process our data.
- In the same cell we have Splitting the Data into Training, Testing & Validation set.

In [ ]:
```python
# Classifying Lables

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Initialize the data generator
datagen = ImageDataGenerator()

# Path to your dataset directory
dataset_path = '/content/drive/My Drive/Colab Notebooks/AML/AML_HW4/Dataset/Train'#'/content/drive/MyDrive/Dataset/Trai

# Load the data
data_generator = datagen.flow_from_directory(
    dataset_path,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

# Get the class indices
class_indices = data_generator.class_indices

# Print the class indices
print("Class labels assigned by ImageDataGenerator:")
for class_name, class_label in class_indices.items():
    print(f"Class '{class_name}': Label {class_label}")
```

```
Found 353 images belonging to 4 classes.
Class labels assigned by ImageDataGenerator:
Class 'Cacti': Label 0
Class 'Succulents': Label 1
Class 'Trees': Label 2
Class 'plants': Label 3
```

Ans 2. & 3. -

1. Dataset Make-up -

- Class 'Cacti': 65 images
- Class 'Succulents': 102 images
- Class 'Trees': 64 images
- Class 'plants': 122 images

2. Created training, validation and testing set

- Training set: 285 images
- Validation set: 69 images
- Test set: 75 images

3. Labels Classificied as -

- Class 'Cacti': Label 0
- Class 'Succulents': Label 1
- Class 'Trees': Label 2
- Class 'plants': Label 3

4. Fine-tune a pretrained model of your choice on this dataset (the one you created in part 3). Report classification accuracy and give a few examples of correct/incorrect classification (show a few images that were correctly/incorrectly classified).

In [ ]:
```
# re-size all the images to this
IMAGE_SIZE = [224, 224]
train_path = '/content/drive/My Drive/Colab Notebooks/AML/AML_HW4/Dataset/Train'#'/content/drive/MyDrive/Dataset/Train/
valid_path = '/content/drive/My Drive/Colab Notebooks/AML/AML_HW4/Dataset/Test'#'/content/drive/MyDrive/Dataset/Test/'
```

In [ ]:
```
# add preprocessing layer to the front of VGG
vgg = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)

# don't train existing weights
for layer in vgg.layers:
    layer.trainable = False
```

In [ ]:
```
  # useful for getting number of classes
folders = glob('/content/drive/My Drive/Colab Notebooks/AML/AML_HW4/Dataset/Train/*')#'/content/drive/MyDrive/Dataset/T
```

Pre-Trained Model Tuning

Dropout = 0.7 | Epochs = 15

In [ ]:
```
# Flatten the output layer to 1 dimension
x = Flatten()(vgg.output)

# Add a fully connected layer with 512 hidden units and ReLU activation
x = Dense(64, activation='relu')(x)

# Add a dropout rate of 0.5
x = Dropout(0.7)(x)

# Add a final softmax layer for classification
predictions = Dense(len(folders), activation='softmax')(x)
```

In [ ]:
```
# create a model object
model_1 = Model(inputs=vgg.input, outputs=predictions)
```

In [ ]:
```
# view the structure of the model
model_1.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 input_1 (InputLayer)         [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0

 flatten_3 (Flatten)          (None, 25088)             0

 dense_6 (Dense)              (None, 64)                1605696
```

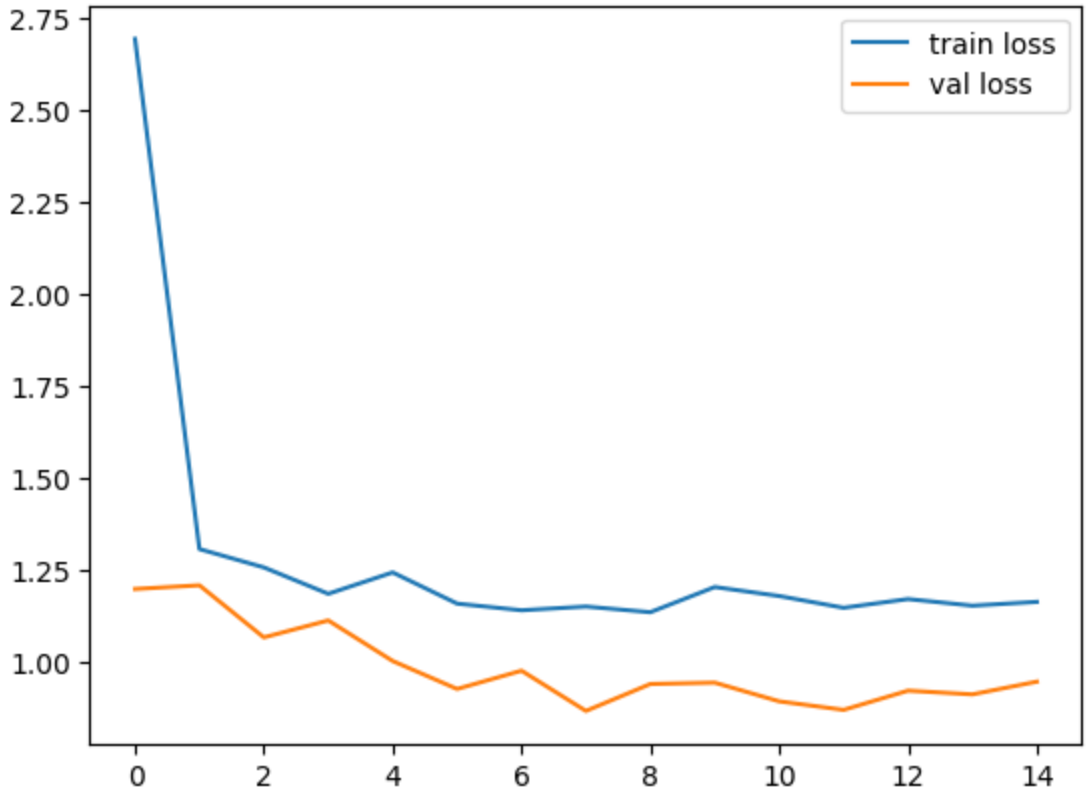
```
 dropout_3 (Dropout)         (None, 64)                0

 dense_7 (Dense)             (None, 4)                 260

=================================================================
Total params: 16320644 (62.26 MB)
Trainable params: 1605956 (6.13 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

In [ ]:
```python
# tell the model what cost and optimization method to use
model_1.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
```

In [ ]:
```python
# Fit the model
r = model_1.fit(
    train_generator,  # Use the training generator
    validation_data=validation_generator,  # Use the validation generator for validation data
    epochs= 15,
    steps_per_epoch=len(train_generator),
    validation_steps=len(validation_generator)
)
```

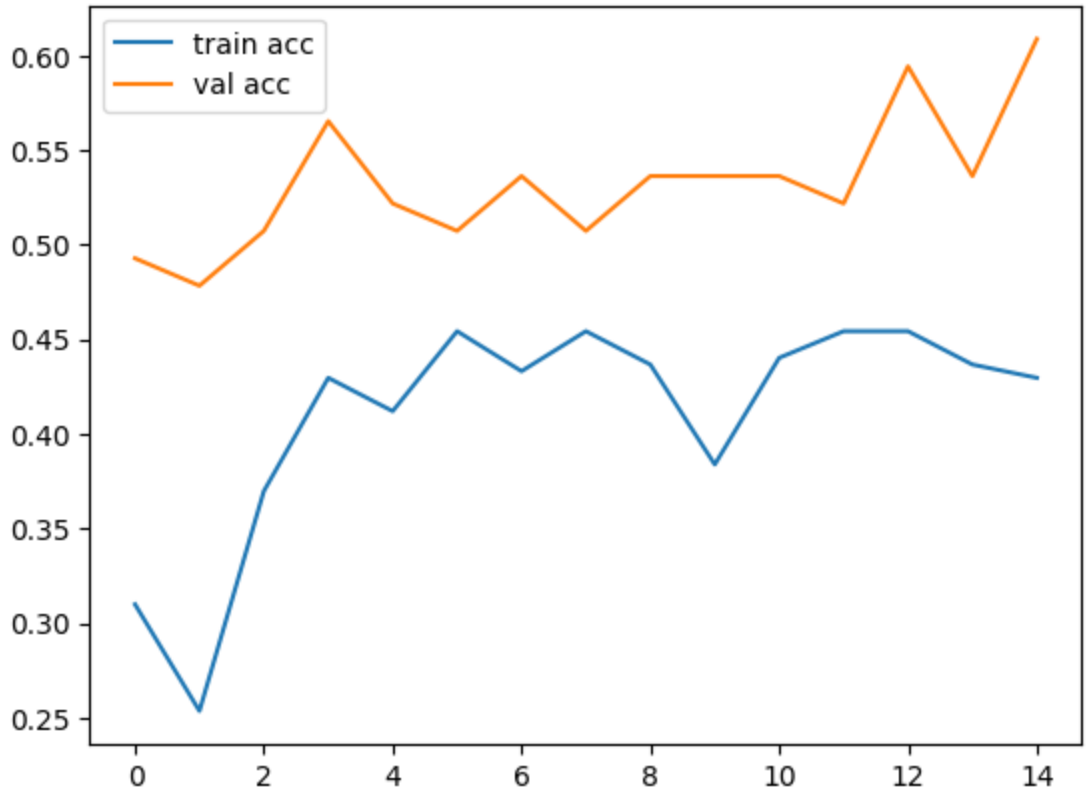
```
Epoch 1/15
9/9 [==============================] - 51s 6s/step - loss: 2.6955 - accuracy: 0.3099 - val_loss: 1.1972 - val_accuracy:
0.4928
Epoch 2/15
9/9 [==============================] - 50s 6s/step - loss: 1.3060 - accuracy: 0.2535 - val_loss: 1.2073 - val_accuracy:
0.4783
Epoch 3/15
9/9 [==============================] - 45s 5s/step - loss: 1.2560 - accuracy: 0.3697 - val_loss: 1.0653 - val_accuracy:
0.5072
Epoch 4/15
9/9 [==============================] - 42s 5s/step - loss: 1.1841 - accuracy: 0.4296 - val_loss: 1.1115 - val_accuracy:
0.5652
Epoch 5/15
9/9 [==============================] - 47s 5s/step - loss: 1.2424 - accuracy: 0.4120 - val_loss: 1.0010 - val_accuracy:
0.5217
Epoch 6/15
9/9 [==============================] - 43s 5s/step - loss: 1.1573 - accuracy: 0.4542 - val_loss: 0.9248 - val_accuracy:
0.5072
Epoch 7/15
9/9 [==============================] - 43s 5s/step - loss: 1.1392 - accuracy: 0.4331 - val_loss: 0.9746 - val_accuracy:
0.5362
Epoch 8/15
9/9 [==============================] - 43s 5s/step - loss: 1.1494 - accuracy: 0.4542 - val_loss: 0.8650 - val_accuracy:
0.5072
Epoch 9/15
9/9 [==============================] - 45s 5s/step - loss: 1.1339 - accuracy: 0.4366 - val_loss: 0.9384 - val_accuracy:
0.5362
Epoch 10/15
9/9 [==============================] - 44s 5s/step - loss: 1.2026 - accuracy: 0.3838 - val_loss: 0.9420 - val_accuracy:
0.5362
Epoch 11/15
9/9 [==============================] - 50s 6s/step - loss: 1.1781 - accuracy: 0.4401 - val_loss: 0.8911 - val_accuracy:
0.5362
Epoch 12/15
9/9 [==============================] - 51s 6s/step - loss: 1.1460 - accuracy: 0.4542 - val_loss: 0.8679 - val_accuracy:
0.5217
Epoch 13/15
9/9 [==============================] - 47s 5s/step - loss: 1.1698 - accuracy: 0.4542 - val_loss: 0.9202 - val_accuracy:
0.5942
Epoch 14/15
9/9 [==============================] - 41s 5s/step - loss: 1.1518 - accuracy: 0.4366 - val_loss: 0.9101 - val_accuracy:
0.5362
Epoch 15/15
9/9 [==============================] - 43s 5s/step - loss: 1.1621 - accuracy: 0.4296 - val_loss: 0.9448 - val_accuracy:
0.6087
```

In [ ]:
```
# loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')
```




```
<Figure size 640x480 with 0 Axes>
```

In [ ]:
```
# accuracies
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')
```




```
<Figure size 640x480 with 0 Axes>
```

Dropout = 0.4 | Epochs = 12

In [ ]:
```python
# Flatten the output layer to 1 dimension
x = Flatten()(vgg.output)

# Add a fully connected layer with 512 hidden units and ReLU activation
x = Dense(64, activation='relu')(x)

# Add a dropout rate of 0.4
x = Dropout(0.4)(x)

# Add a final softmax layer for classification
predictions = Dense(len(folders), activation='softmax')(x)

# create a model object
model_2 = Model(inputs=vgg.input, outputs=predictions)
```

```
# view the structure of the model
model_2.summary()
```

```
Model: "model_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten_9 (Flatten)         (None, 25088)             0

 dense_18 (Dense)            (None, 64)                1605696
```

```
 dropout_9 (Dropout)         (None, 64)                0

 dense_19 (Dense)            (None, 4)                 260

=================================================================
Total params: 16320644 (62.26 MB)
Trainable params: 1605956 (6.13 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

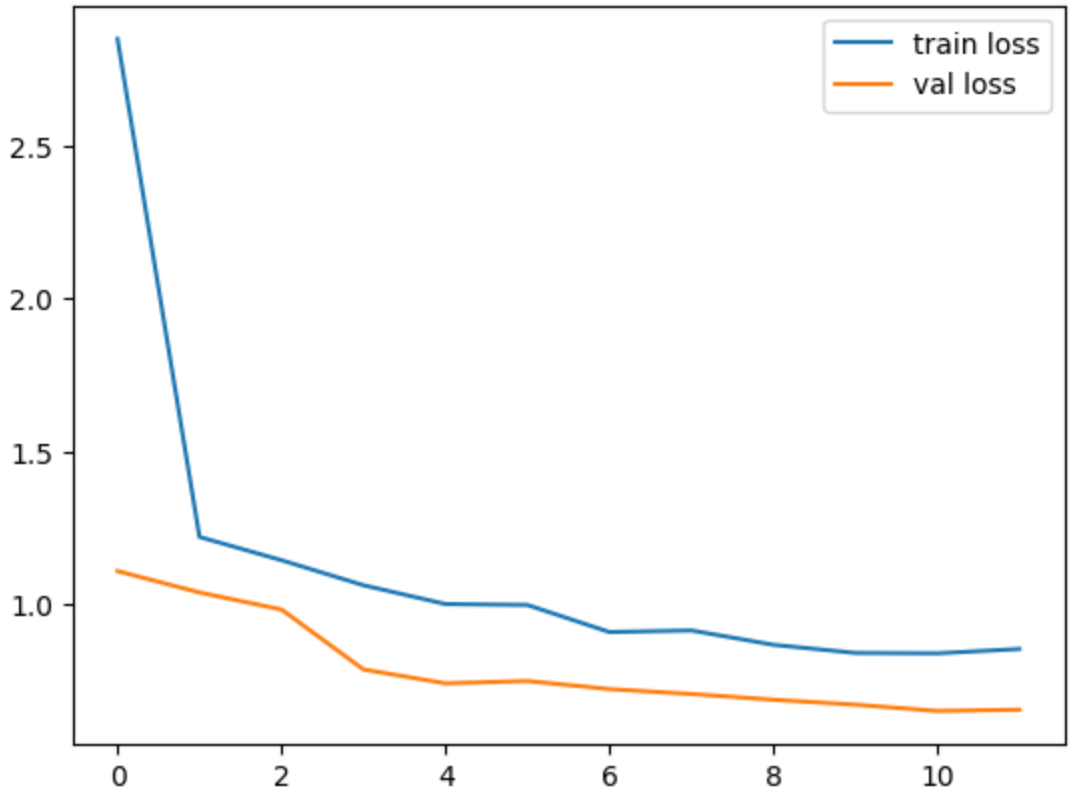
In [ ]:
```python
# tell the model what cost and optimization method to use
model_2.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
```

In [ ]:
```python
# Fit the model
r = model_2.fit(
    train_generator,  # Use the training generator
    validation_data=validation_generator,  # Use the validation generator for validation data
    epochs= 12,
    steps_per_epoch=len(train_generator),
    validation_steps=len(validation_generator)
)
```

```
Epoch 1/12
9/9 [==============================] - 47s 5s/step - loss: 2.8476 - accuracy: 0.3345 - val_loss: 1.1094 - val_accuracy:
0.5507
Epoch 2/12
9/9 [==============================] - 42s 5s/step - loss: 1.2209 - accuracy: 0.3944 - val_loss: 1.0395 - val_accuracy:
0.5217
Epoch 3/12
9/9 [==============================] - 43s 5s/step - loss: 1.1449 - accuracy: 0.4296 - val_loss: 0.9840 - val_accuracy:
0.6087
Epoch 4/12
9/9 [==============================] - 42s 5s/step - loss: 1.0631 - accuracy: 0.4507 - val_loss: 0.7880 - val_accuracy:
0.6377
Epoch 5/12
9/9 [==============================] - 42s 5s/step - loss: 1.0017 - accuracy: 0.4225 - val_loss: 0.7427 - val_accuracy:
0.6957
Epoch 6/12
9/9 [==============================] - 43s 5s/step - loss: 0.9987 - accuracy: 0.5810 - val_loss: 0.7504 - val_accuracy:
0.6812
Epoch 7/12
9/9 [==============================] - 40s 4s/step - loss: 0.9102 - accuracy: 0.5915 - val_loss: 0.7240 - val_accuracy:
0.6522
Epoch 8/12
9/9 [==============================] - 42s 5s/step - loss: 0.9155 - accuracy: 0.5563 - val_loss: 0.7079 - val_accuracy:
0.7101
Epoch 9/12
9/9 [==============================] - 40s 5s/step - loss: 0.8686 - accuracy: 0.6162 - val_loss: 0.6892 - val_accuracy:
0.7101
Epoch 10/12
9/9 [==============================] - 41s 5s/step - loss: 0.8420 - accuracy: 0.6268 - val_loss: 0.6731 - val_accuracy:
0.7101
Epoch 11/12
9/9 [==============================] - 41s 5s/step - loss: 0.8407 - accuracy: 0.6444 - val_loss: 0.6526 - val_accuracy:
0.7101
Epoch 12/12
9/9 [==============================] - 46s 5s/step - loss: 0.8550 - accuracy: 0.6021 - val_loss: 0.6566 - val_accuracy:
0.7101
```

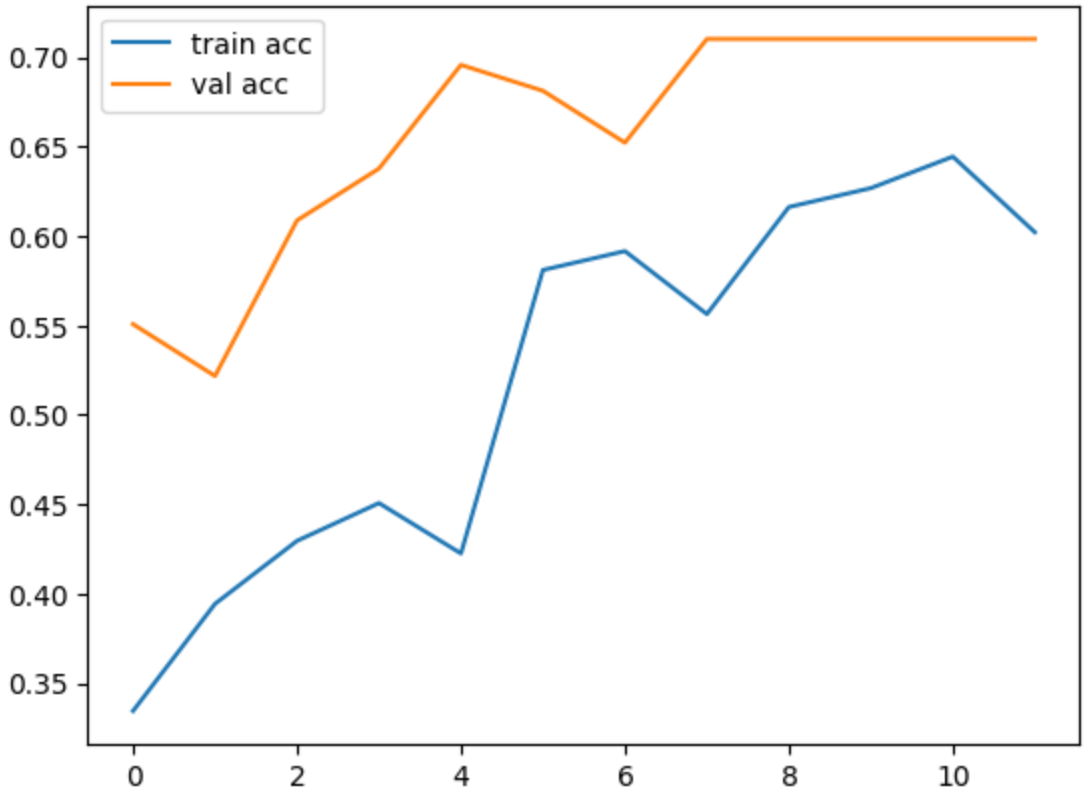
In [ ]:

```
# loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')
```




```
<Figure size 640x480 with 0 Axes>
```

In [ ]:
```
# accuracies
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')
```




```
<Figure size 640x480 with 0 Axes>
```

Dropout = 0.3 | Epochs = 10

In [ ]:

```python
# Flatten the output layer to 1 dimension
x = Flatten()(vgg.output)

# Add a fully connected layer with 512 hidden units and ReLU activation
x = Dense(64, activation='relu')(x)

# Add a dropout rate of 0.3
x = Dropout(0.3)(x)

# Add a final softmax layer for classification
predictions = Dense(len(folders), activation='softmax')(x)

# create a model object
model_3 = Model(inputs=vgg.input, outputs=predictions)
```

```
# view the structure of the model
model_3.summary()
```

```
Model: "model_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten_6 (Flatten)         (None, 25088)             0

 dense_12 (Dense)            (None, 64)                1605696
```

```
 dropout_6 (Dropout)         (None, 64)                0

 dense_13 (Dense)            (None, 4)                 260

=================================================================
Total params: 16320644 (62.26 MB)
Trainable params: 1605956 (6.13 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

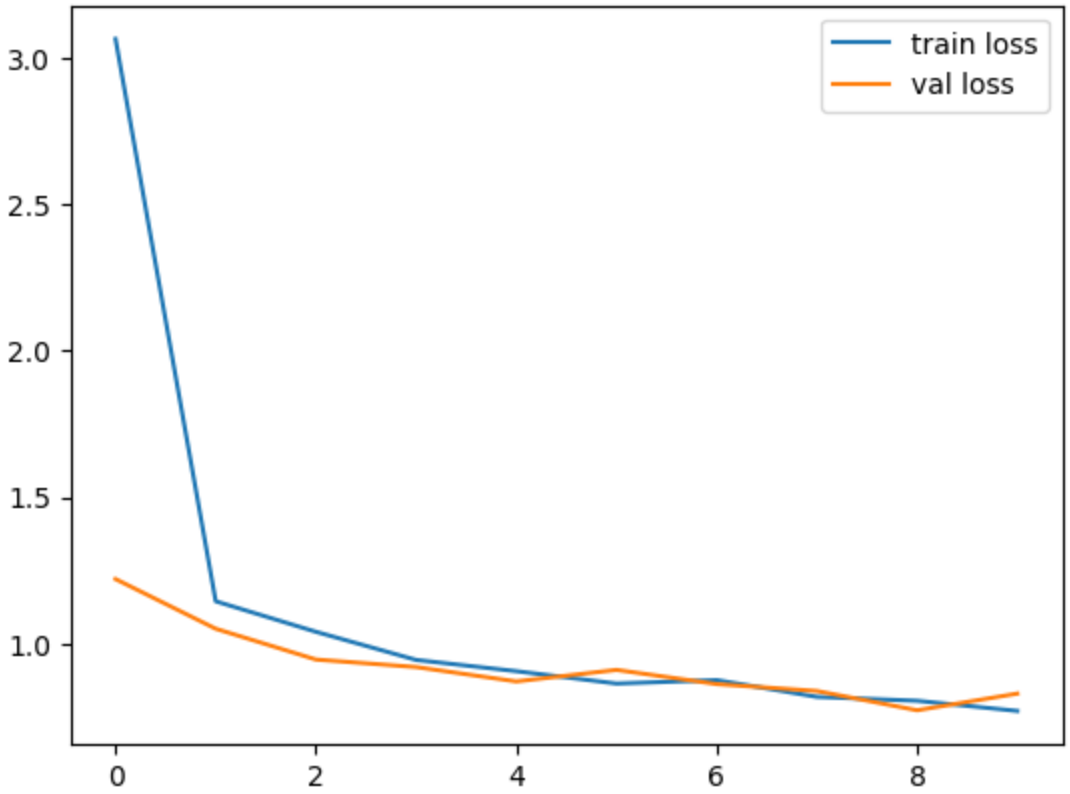
In [ ]:
```python
# tell the model what cost and optimization method to use
model_3.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
```

In [ ]:
```python
# Fit the model
r = model_3.fit(
    train_generator,  # Use the training generator
    validation_data=validation_generator,  # Use the validation generator for validation data
    epochs= 10,
    steps_per_epoch=len(train_generator),
    validation_steps=len(validation_generator)
)
```

```
Epoch 1/10
9/9 [==============================] - 48s 5s/step - loss: 3.0631 - accuracy: 0.3697 - val_loss: 1.2212 - val_accuracy:
0.4058
Epoch 2/10
9/9 [==============================] - 41s 5s/step - loss: 1.1454 - accuracy: 0.4754 - val_loss: 1.0517 - val_accuracy:
0.5652
Epoch 3/10
9/9 [==============================] - 43s 5s/step - loss: 1.0413 - accuracy: 0.5282 - val_loss: 0.9467 - val_accuracy:
0.6232
Epoch 4/10
9/9 [==============================] - 44s 5s/step - loss: 0.9454 - accuracy: 0.6056 - val_loss: 0.9210 - val_accuracy:
0.6812
Epoch 5/10
9/9 [==============================] - 40s 5s/step - loss: 0.9070 - accuracy: 0.5951 - val_loss: 0.8720 - val_accuracy:
0.6667
Epoch 6/10
9/9 [==============================] - 41s 5s/step - loss: 0.8648 - accuracy: 0.5915 - val_loss: 0.9114 - val_accuracy:
0.6232
Epoch 7/10
9/9 [==============================] - 43s 5s/step - loss: 0.8765 - accuracy: 0.6092 - val_loss: 0.8630 - val_accuracy:
0.7101
Epoch 8/10
9/9 [==============================] - 41s 5s/step - loss: 0.8193 - accuracy: 0.6232 - val_loss: 0.8393 - val_accuracy:
0.6957
Epoch 9/10
9/9 [==============================] - 48s 5s/step - loss: 0.8060 - accuracy: 0.6338 - val_loss: 0.7737 - val_accuracy:
0.7101
Epoch 10/10
9/9 [==============================] - 43s 5s/step - loss: 0.7714 - accuracy: 0.6444 - val_loss: 0.8306 - val_accuracy:
0.6522
```

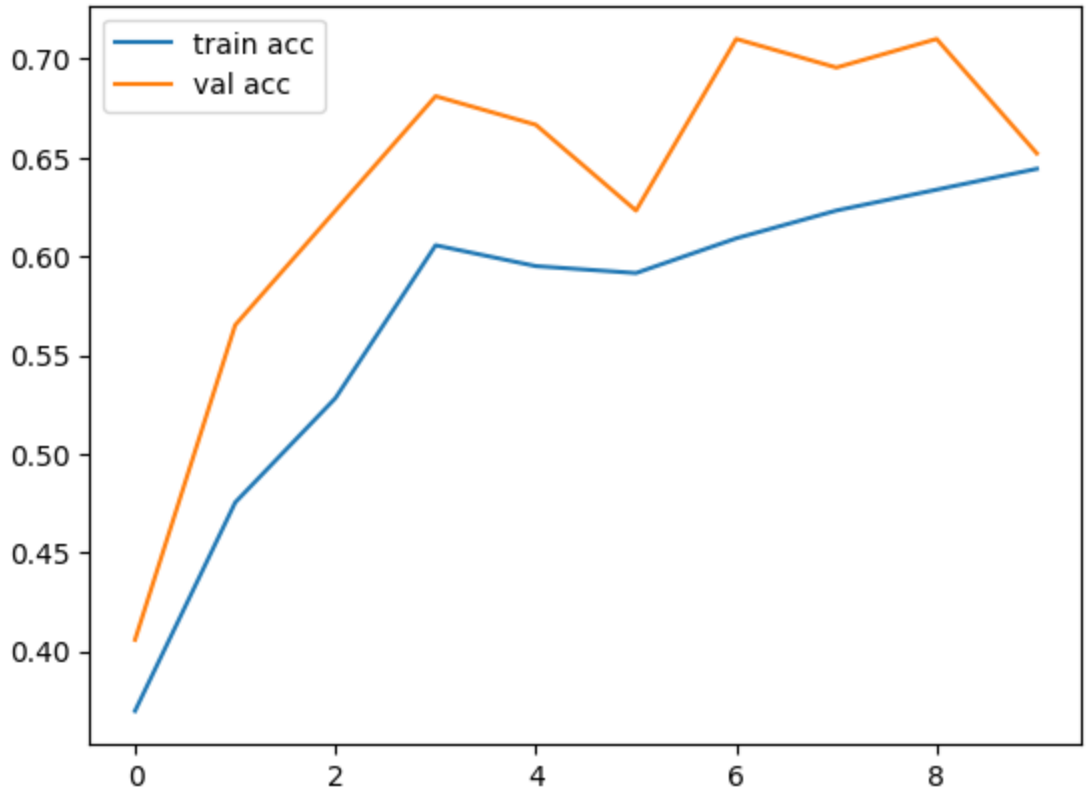
In [ ]:

```
# loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')
```




```
<Figure size 640x480 with 0 Axes>
```

In [ ]:
```
# accuracies
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')
```




```
<Figure size 640x480 with 0 Axes>
```

Ans 4. -

We train/test 3 models with different hyperparameters and selected the model with the best Val_Accuracy -

1. Dropout = 0.7 | Epochs = 15 | Val_Accuracy = smaller than (<) 70%
2. Dropout = 0.4 | Epochs = 12 | Val_Accuracy = greater than (>) 70%
3. Dropout = 0.3 | Epochs = 10 | Val_Accuracy = smaller than (<) 70%

Best Model Selected = Dropout = 0.4 | Epochs = 12 | Accuracy = greater than (>) 70%

In [ ]:
```
import tensorflow as tf

from tensorflow.keras.models import load_model

model.save('facerecognition_Vgg16.h5')
```

In [ ]:
```
# Evaluate the model on the test data using `evaluate`
test_loss, test_accuracy = model_2.evaluate(test_generator, steps=len(test_generator))

# Print the test loss and accuracy
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
3/3 [==============================] - 9s 3s/step - loss: 0.7387 - accuracy: 0.6267
Test Loss: 0.7386859059333801
Test Accuracy: 0.6266666650772095
```

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

# Number of classes
num_classes = len(test_generator.class_indices)

# Initialize a dictionary to store images
class_images = {class_id: [] for class_id in range(num_classes)}
class_labels = {class_id: [] for class_id in range(num_classes)}

# Iterate over the test set to collect 10 images for each class
for test_images, test_labels in test_generator:
    # Predict the labels for this batch of test images
    predicted_labels = model_2.predict(test_images)
    predicted_labels = np.argmax(predicted_labels, axis=1)
    actual_labels = np.argmax(test_labels, axis=1)

    for i in range(len(test_images)):
        actual_label = actual_labels[i]
        # Collect images for each class
        if len(class_images[actual_label]) < 10:
            class_images[actual_label].append(test_images[i])
            class_labels[actual_label].append(predicted_labels[i])

    # Break the loop if 10 images of each class have been collected
    if all(len(images) == 10 for images in class_images.values()):
        break
```

```python
# Display 10 images of each class
# Create a dictionary mapping from class ID to class name
id_to_class = {v: k for k, v in test_generator.class_indices.items()}

# Display 10 images of each class
for class_id in range(num_classes):
    plt.figure(figsize=(15, 15))
    class_name = id_to_class[class_id]  # Get the class name using class ID
    for i, (image, predicted_label) in enumerate(zip(class_images[class_id], class_labels[class_id])):
        plt.subplot(2, 5, i + 1)
        plt.imshow(image)
        plt.title(f"Actual: {class_name} (ID: {class_id})\nPredicted: {predicted_label}")
        plt.axis("off")
    plt.suptitle(f"Class {class_name} (ID: {class_id}) Images")
    plt.tight_layout()
    plt.show()
```

```
1/1 [==============================] - 0s 145ms/step
1/1 [==============================] - 0s 27ms/step
```

Class Cacti (ID: 0) Images

Class Succulents (ID: 1) Images

Actual: Succulents (ID: 1)
Predicted: 1

Actual: Succulents (ID: 1)
Predicted: 1

Actual: Succulents (ID: 1)
Predicted: 1

Actual: Succulents (ID: 1)
Predicted: 1

Actual: Succulents (ID: 1)
Predicted: 1

Actual: Succulents (ID: 1)
Predicted: 3

Actual: Succulents (ID: 1)
Predicted: 3

Actual: Succulents (ID: 1)
Predicted: 3

Actual: Succulents (ID: 1)
Predicted: 1

Actual: Succulents (ID: 1)
Predicted: 1

Class Tress (ID: 2) Images

Class plants (ID: 3) Images

Ans 4. -

1. As seen from above there are certain images that have been correctly identified, and some have not been correctly identified.
2. We observed that the incorrect image classification was majorily for the 'Cacti' class.
3. Minor incorrections were observed when predicting 'Plants' & 'Succulents' classes.
4. Whereas the class 'Tree' was classified perfectly (~100% accuracy).

5. Train from scratch (without pretraining) a deep neural network that contains convolutional layers on this dataset (the one you created in part 3). Report classification accuracy and give a few examples of correct/incorrect classification (show a few images that were correctly/incorrectly classified). Note: The objective of this question is to illustrate that training deep networks from scratch requires a lot of data so it is ok if your classification accuracy is low. [15 points]

Tuning our CNN Model -

Model 1 -

In [ ]:
```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense

SIZE = 224

INPUT_SHAPE = (SIZE, SIZE, 3)   #change to (SIZE, SIZE, 3)

model_cnn = Sequential()
```

```
model_cnn.add(Conv2D(128, (3, 3), input_shape=INPUT_SHAPE))
model_cnn.add(Activation('relu'))
model_cnn.add(MaxPooling2D(pool_size=(2, 2)))

model_cnn.add(Conv2D(64, (3, 3)))
model_cnn.add(Activation('relu'))
model_cnn.add(MaxPooling2D(pool_size=(2, 2)))

model_cnn.add(Conv2D(32, (3, 3)))
model_cnn.add(Activation('relu'))
model_cnn.add(MaxPooling2D(pool_size=(2, 2)))

model_cnn.add(Conv2D(16, (3, 3)))
model_cnn.add(Activation('relu'))
model_cnn.add(MaxPooling2D(pool_size=(2, 2)))

model_cnn.add(Flatten())
model_cnn.add(Dense(8))
model_cnn.add(Activation('relu'))
model_cnn.add(Dropout(0.2))
model_cnn.add(Dense(4))
model_cnn.add(Activation('softmax'))

# tell the model what cost and optimization method to use
model_cnn.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)

print(model_cnn.summary())
```

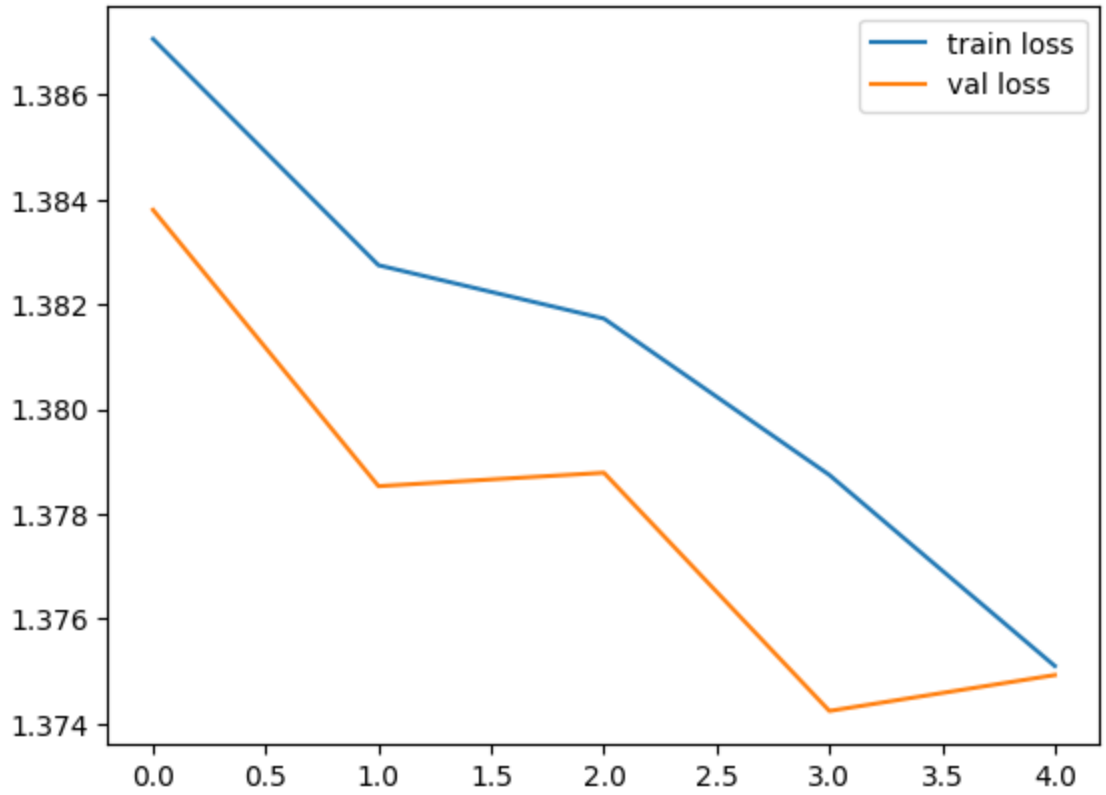
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 222, 222, 128)     3584

 activation_12 (Activation)  (None, 222, 222, 128)     0

 max_pooling2d_8 (MaxPoolin  (None, 111, 111, 128)     0
 g2D)

 conv2d_9 (Conv2D)           (None, 109, 109, 64)      73792

 activation_13 (Activation)  (None, 109, 109, 64)      0

 max_pooling2d_9 (MaxPoolin  (None, 54, 54, 64)        0
 g2D)

 conv2d_10 (Conv2D)          (None, 52, 52, 32)        18464

 activation_14 (Activation)  (None, 52, 52, 32)        0

 max_pooling2d_10 (MaxPooli  (None, 26, 26, 32)        0
 ng2D)

 conv2d_11 (Conv2D)          (None, 24, 24, 16)        4624

 activation_15 (Activation)  (None, 24, 24, 16)        0

 max_pooling2d_11 (MaxPooli  (None, 12, 12, 16)        0
 ng2D)

 flatten_12 (Flatten)        (None, 2304)              0

 dense_24 (Dense)            (None, 8)                 18440

 activation_16 (Activation)  (None, 8)                 0

 dropout_12 (Dropout)        (None, 8)                 0

 dense_25 (Dense)            (None, 4)                 36

 activation_17 (Activation)  (None, 4)                 0

=================================================================
```

```
Total params: 118940 (464.61 KB)
Trainable params: 118940 (464.61 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

In [ ]:
```python
# Fit the model
cnn = model_cnn.fit(
    train_generator,  # Use the training generator
    validation_data=validation_generator,  # Use the validation generator for validation data
    epochs=5,
    steps_per_epoch=len(train_generator),
    validation_steps=len(validation_generator)
)
```

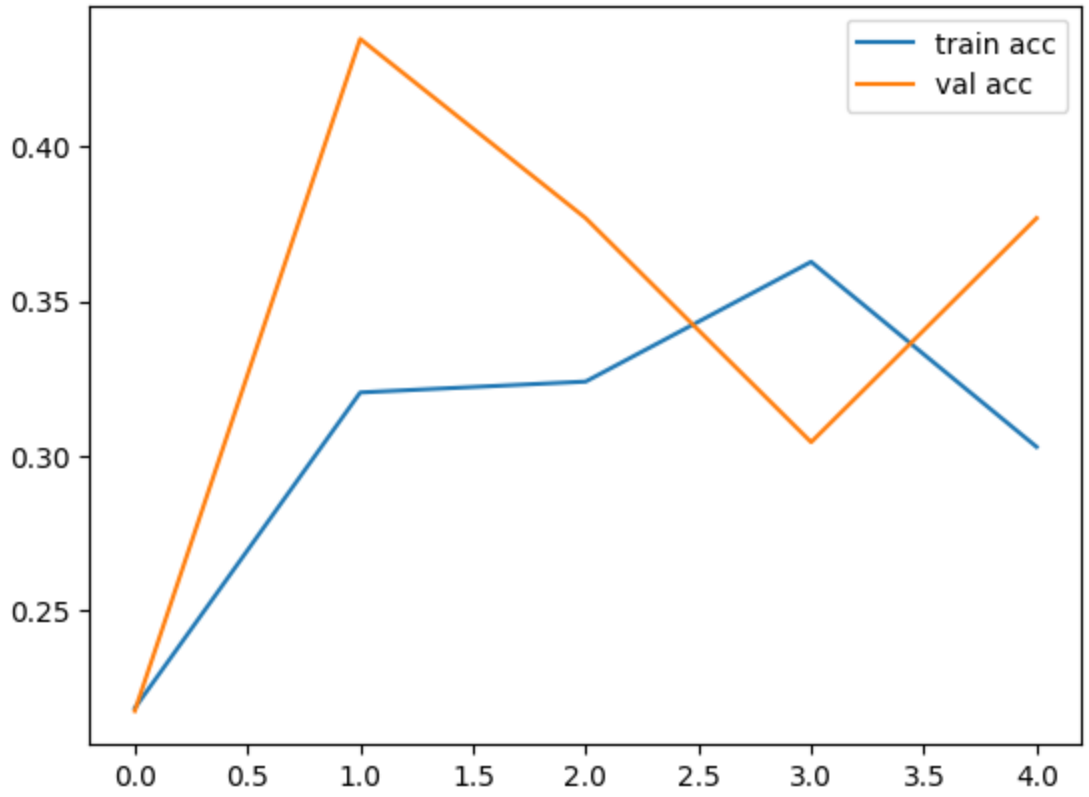
```
Epoch 1/5
9/9 [==============================] - 46s 5s/step - loss: 1.3871 - accuracy: 0.2183 - val_loss: 1.3838 - val_accuracy:
0.2174
Epoch 2/5
9/9 [==============================] - 43s 5s/step - loss: 1.3827 - accuracy: 0.3204 - val_loss: 1.3785 - val_accuracy:
0.4348
Epoch 3/5
9/9 [==============================] - 40s 5s/step - loss: 1.3817 - accuracy: 0.3239 - val_loss: 1.3788 - val_accuracy:
0.3768
Epoch 4/5
9/9 [==============================] - 42s 5s/step - loss: 1.3787 - accuracy: 0.3627 - val_loss: 1.3742 - val_accuracy:
0.3043
Epoch 5/5
9/9 [==============================] - 42s 5s/step - loss: 1.3751 - accuracy: 0.3028 - val_loss: 1.3749 - val_accuracy:
0.3768
```

In [ ]:
```python
# loss
plt.plot(cnn.history['loss'], label='train loss')
plt.plot(cnn.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')
```




```
<Figure size 640x480 with 0 Axes>
```

In [ ]:
```
# accuracies
plt.plot(cnn.history['accuracy'], label='train acc')
plt.plot(cnn.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')
```




```
<Figure size 640x480 with 0 Axes>
```

In [ ]:
```python
# Evaluate the model on the test data using `evaluate`
test_loss, test_accuracy = model_cnn.evaluate(test_generator, steps=len(test_generator))

# Print the test loss and accuracy
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
3/3 [==============================] - 9s 3s/step - loss: 1.3857 - accuracy: 0.2800
Test Loss: 1.385651707649231
Test Accuracy: 0.2800000011920929
```

Model 2 -

In [112…
```python
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
```

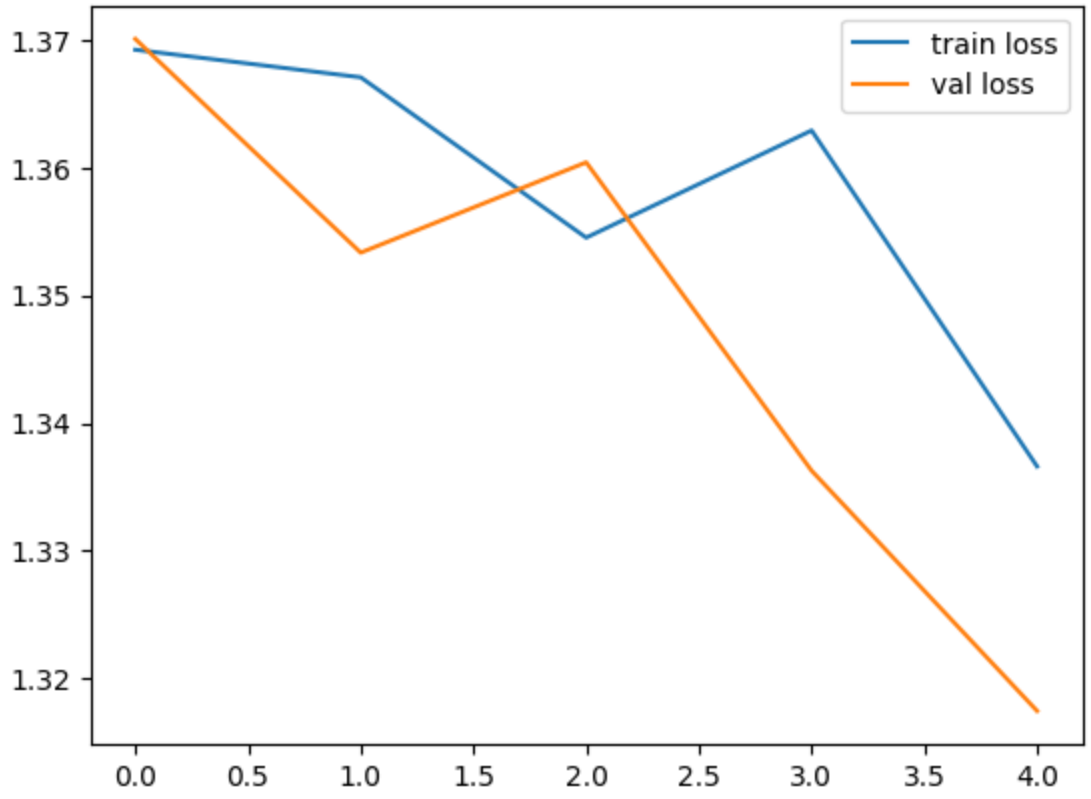
```
SIZE = 224

INPUT_SHAPE = (SIZE, SIZE, 3)   #change to (SIZE, SIZE, 3)

model_cnn_2 = Sequential()
model_cnn_2.add(Conv2D(256, (4, 4), input_shape=INPUT_SHAPE))
model_cnn_2.add(Activation('relu'))
model_cnn_2.add(MaxPooling2D(pool_size=(2, 2)))

model_cnn_2.add(Conv2D(64, (4, 4)))
model_cnn_2.add(Activation('relu'))
model_cnn_2.add(MaxPooling2D(pool_size=(2, 2)))

# model_cnn_2.add(Conv2D(32, (3, 3)))
# model_cnn_2.add(Activation('relu'))
# model_cnn_2.add(MaxPooling2D(pool_size=(2, 2)))

model_cnn_2.add(Conv2D(16, (4, 4)))
model_cnn_2.add(Activation('relu'))
model_cnn_2.add(MaxPooling2D(pool_size=(2, 2)))

model_cnn_2.add(Flatten())
model_cnn_2.add(Dense(8))
model_cnn_2.add(Activation('relu'))
model_cnn_2.add(Dropout(0.3))
model_cnn_2.add(Dense(4))
model_cnn_2.add(Activation('softmax'))

# tell the model what cost and optimization method to use
model_cnn_2.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)

print(model_cnn_2.summary())
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_15 (Conv2D)          (None, 221, 221, 256)     12544

 activation_23 (Activation)  (None, 221, 221, 256)     0

 max_pooling2d_15 (MaxPooli  (None, 110, 110, 256)     0
 ng2D)

 conv2d_16 (Conv2D)          (None, 107, 107, 64)      262208

 activation_24 (Activation)  (None, 107, 107, 64)      0

 max_pooling2d_16 (MaxPooli  (None, 53, 53, 64)        0
 ng2D)

 conv2d_17 (Conv2D)          (None, 50, 50, 16)        16400

 activation_25 (Activation)  (None, 50, 50, 16)        0

 max_pooling2d_17 (MaxPooli  (None, 25, 25, 16)        0
 ng2D)

 flatten_14 (Flatten)        (None, 10000)             0

 dense_28 (Dense)            (None, 8)                 80008

 activation_26 (Activation)  (None, 8)                 0

 dropout_14 (Dropout)        (None, 8)                 0

 dense_29 (Dense)            (None, 4)                 36

 activation_27 (Activation)  (None, 4)                 0

=================================================================
Total params: 371196 (1.42 MB)
Trainable params: 371196 (1.42 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

In [113…
```
# Fit the model
cnn_2 = model_cnn.fit(
    train_generator,  # Use the training generator
    validation_data=validation_generator,  # Use the validation generator for validation data
    epochs=5,
    steps_per_epoch=len(train_generator),
    validation_steps=len(validation_generator)
)
```

```
Epoch 1/5
9/9 [==============================] - 45s 5s/step - loss: 1.3692 - accuracy: 0.3486 - val_loss: 1.3701 - val_accuracy:
0.3333
Epoch 2/5
9/9 [==============================] - 39s 4s/step - loss: 1.3671 - accuracy: 0.3380 - val_loss: 1.3533 - val_accuracy:
0.4203
Epoch 3/5
9/9 [==============================] - 40s 4s/step - loss: 1.3545 - accuracy: 0.3803 - val_loss: 1.3604 - val_accuracy:
0.3623
Epoch 4/5
9/9 [==============================] - 40s 5s/step - loss: 1.3629 - accuracy: 0.3556 - val_loss: 1.3363 - val_accuracy:
0.4058
Epoch 5/5
9/9 [==============================] - 41s 5s/step - loss: 1.3366 - accuracy: 0.3838 - val_loss: 1.3175 - val_accuracy:
0.3623
```

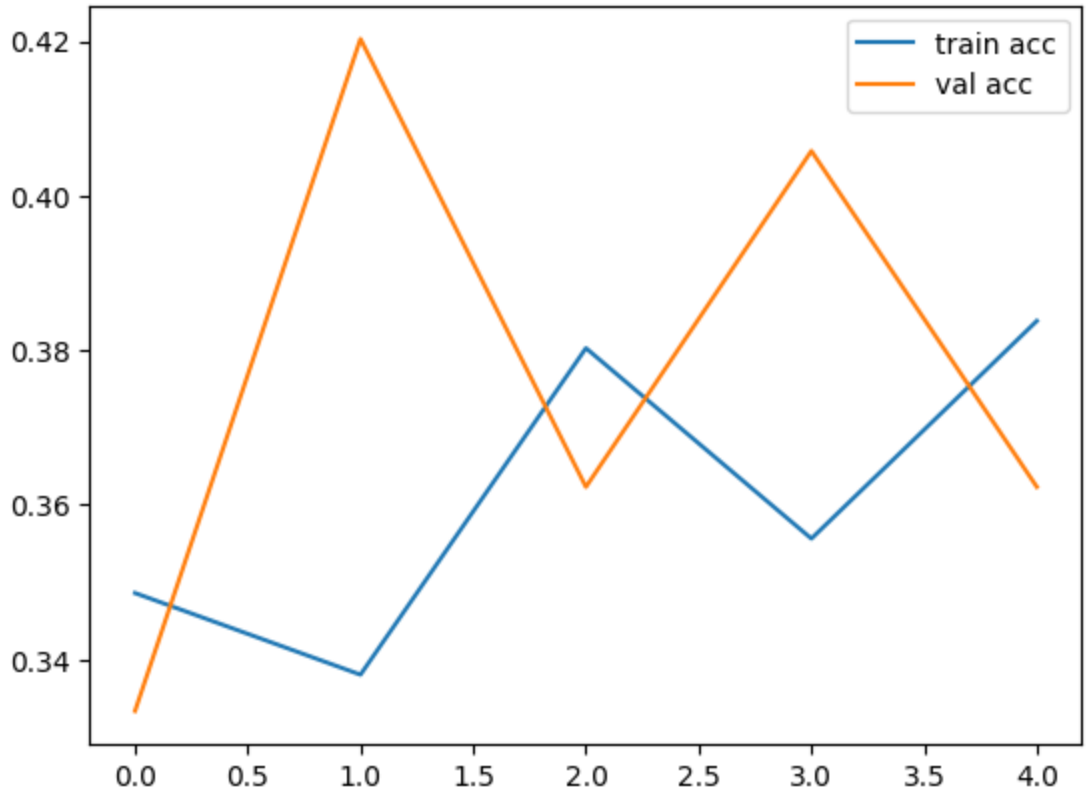
In [114…
```
# loss
plt.plot(cnn_2.history['loss'], label='train loss')
plt.plot(cnn_2.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')
```




```
<Figure size 640x480 with 0 Axes>
```

In [115…

```
# accuracies
plt.plot(cnn_2.history['accuracy'], label='train acc')
plt.plot(cnn_2.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')
```




```
<Figure size 640x480 with 0 Axes>
```

Model 3 -

In [116…

```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense

SIZE = 224

INPUT_SHAPE = (SIZE, SIZE, 3)   #change to (SIZE, SIZE, 3)

model_cnn_3 = Sequential()
model_cnn_3.add(Conv2D(128, (4, 4), input_shape=INPUT_SHAPE))
model_cnn_3.add(Activation('relu'))
model_cnn_3.add(MaxPooling2D(pool_size=(2, 2)))

model_cnn_3.add(Conv2D(64, (3, 3)))
```

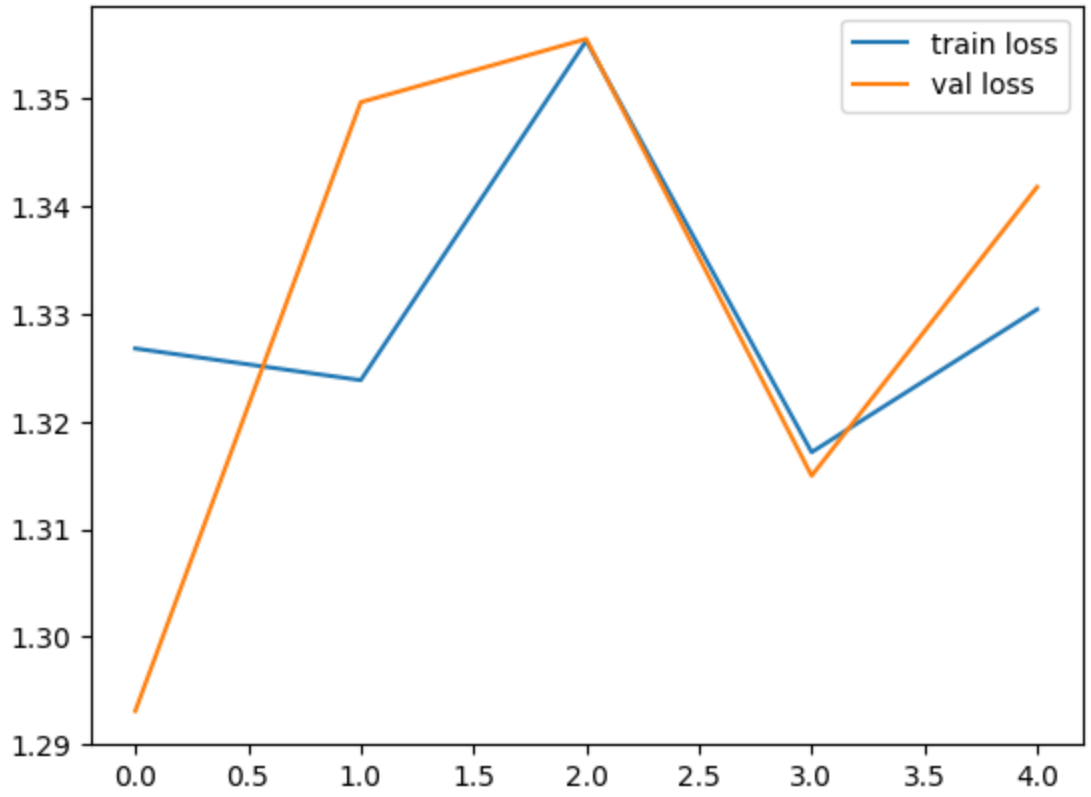
```
model_cnn_3.add(Activation('relu'))
model_cnn_3.add(MaxPooling2D(pool_size=(2, 2)))

# model_cnn_3.add(Conv2D(32, (3, 3)))
# model_cnn_3.add(Activation('relu'))
# model_cnn_3.add(MaxPooling2D(pool_size=(2, 2)))

model_cnn_3.add(Conv2D(16, (2, 2)))
model_cnn_3.add(Activation('relu'))
model_cnn_3.add(MaxPooling2D(pool_size=(2, 2)))

model_cnn_3.add(Flatten())
model_cnn_3.add(Dense(8))
model_cnn_3.add(Activation('relu'))
model_cnn_3.add(Dropout(0.1))
model_cnn_3.add(Dense(4))
model_cnn_3.add(Activation('softmax'))

# tell the model what cost and optimization method to use
model_cnn_3.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)

print(model_cnn_3.summary())
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_18 (Conv2D)          (None, 221, 221, 128)     6272

 activation_28 (Activation)  (None, 221, 221, 128)     0

 max_pooling2d_18 (MaxPooli  (None, 110, 110, 128)     0
 ng2D)

 conv2d_19 (Conv2D)          (None, 108, 108, 64)      73792

 activation_29 (Activation)  (None, 108, 108, 64)      0

 max_pooling2d_19 (MaxPooli  (None, 54, 54, 64)        0
 ng2D)

 conv2d_20 (Conv2D)          (None, 53, 53, 16)        4112

 activation_30 (Activation)  (None, 53, 53, 16)        0

 max_pooling2d_20 (MaxPooli  (None, 26, 26, 16)        0
 ng2D)

 flatten_15 (Flatten)        (None, 10816)             0

 dense_30 (Dense)            (None, 8)                 86536

 activation_31 (Activation)  (None, 8)                 0

 dropout_15 (Dropout)        (None, 8)                 0

 dense_31 (Dense)            (None, 4)                 36

 activation_32 (Activation)  (None, 4)                 0

=================================================================
Total params: 170748 (666.98 KB)
Trainable params: 170748 (666.98 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

In [117…
```python
# Fit the model
cnn_3 = model_cnn.fit(
    train_generator,  # Use the training generator
    validation_data=validation_generator,  # Use the validation generator for validation data
    epochs=5,
    steps_per_epoch=len(train_generator),
    validation_steps=len(validation_generator)
)
```

```
Epoch 1/5
9/9 [==============================] - 44s 5s/step - loss: 1.3268 - accuracy: 0.4014 - val_loss: 1.2931 - val_accuracy:
0.3913
Epoch 2/5
9/9 [==============================] - 40s 5s/step - loss: 1.3238 - accuracy: 0.3873 - val_loss: 1.3497 - val_accuracy:
0.3913
Epoch 3/5
9/9 [==============================] - 42s 5s/step - loss: 1.3554 - accuracy: 0.3486 - val_loss: 1.3555 - val_accuracy:
0.3623
Epoch 4/5
9/9 [==============================] - 42s 5s/step - loss: 1.3171 - accuracy: 0.3697 - val_loss: 1.3150 - val_accuracy:
0.4058
Epoch 5/5
9/9 [==============================] - 39s 5s/step - loss: 1.3304 - accuracy: 0.3838 - val_loss: 1.3418 - val_accuracy:
0.3623
```

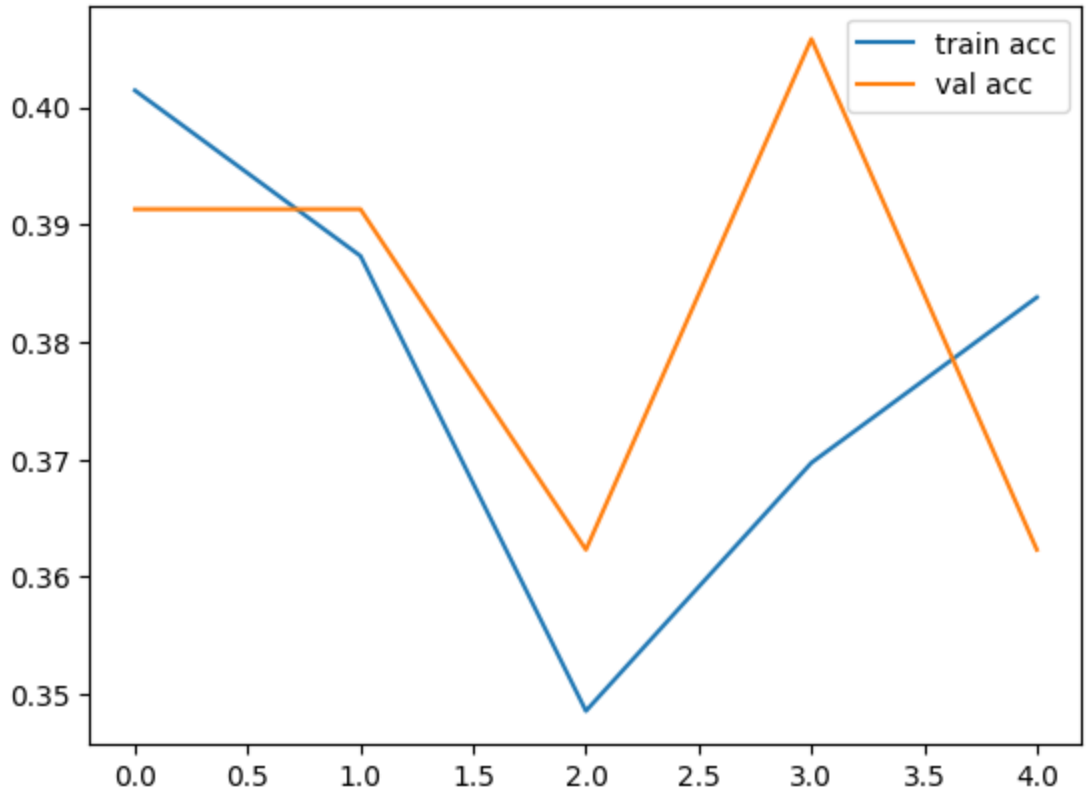
In [118…
```python
# loss
plt.plot(cnn_3.history['loss'], label='train loss')
plt.plot(cnn_3.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')
```




```
<Figure size 640x480 with 0 Axes>
```

In [119…

```
# accuracies
plt.plot(cnn_3.history['accuracy'], label='train acc')
plt.plot(cnn_3.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')
```




```
<Figure size 640x480 with 0 Axes>
```

Ans 5. -

We train/test 3 models with different hyperparameters and selected the model which fits the data best.
Best Model Selected = Model_3

In [120…

```python
import matplotlib.pyplot as plt
import numpy as np

# Number of classes
num_classes = len(test_generator.class_indices)

# Initialize a dictionary to store images
class_images = {class_id: [] for class_id in range(num_classes)}
```

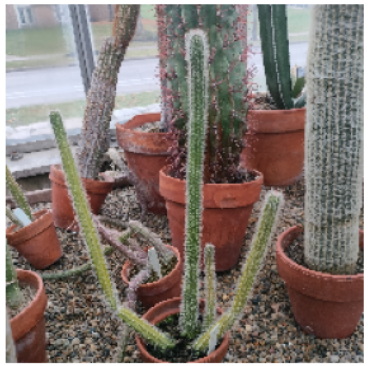
```python
class_labels = {class_id: [] for class_id in range(num_classes)}

# Iterate over the test set to collect 10 images for each class
for test_images, test_labels in test_generator:
    # Predict the labels for this batch of test images
    predicted_labels = model_cnn_3.predict(test_images)
    predicted_labels = np.argmax(predicted_labels, axis=1)
    actual_labels = np.argmax(test_labels, axis=1)

    for i in range(len(test_images)):
        actual_label = actual_labels[i]
        # Collect images for each class
        if len(class_images[actual_label]) < 10:
            class_images[actual_label].append(test_images[i])
            class_labels[actual_label].append(predicted_labels[i])

    # Break the loop if 10 images of each class have been collected
    if all(len(images) == 10 for images in class_images.values()):
        break

# Display 10 images of each class
# Create a dictionary mapping from class ID to class name
id_to_class = {v: k for k, v in test_generator.class_indices.items()}

# Display 10 images of each class
for class_id in range(num_classes):
    plt.figure(figsize=(15, 15))
    class_name = id_to_class[class_id]  # Get the class name using class ID
    for i, (image, predicted_label) in enumerate(zip(class_images[class_id], class_labels[class_id])):
        plt.subplot(2, 5, i + 1)
        plt.imshow(image)
        plt.title(f"Actual: {class_name} (ID: {class_id})\nPredicted: {predicted_label}")
        plt.axis("off")
    plt.suptitle(f"Class {class_name} (ID: {class_id}) Images")
    plt.tight_layout()
    plt.show()
```

```
WARNING:tensorflow:5 out of the last 10 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7e512e09c550> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

```
1/1 [==============================] - 0s 432ms/step
1/1 [==============================] - 1s 607ms/step
1/1 [==============================] - 0s 28ms/step
```

Class Cacti (ID: 0) Images

Actual: Cacti (ID: 0)
Predicted: 1



Actual: Cacti (ID: 0)
Predicted: 1

Actual: Cacti (ID: 0)
Predicted: 1

Actual: Cacti (ID: 0)
Predicted: 3

Actual: Cacti (ID: 0)
Predicted: 1

Actual: Cacti (ID: 0)
Predicted: 3

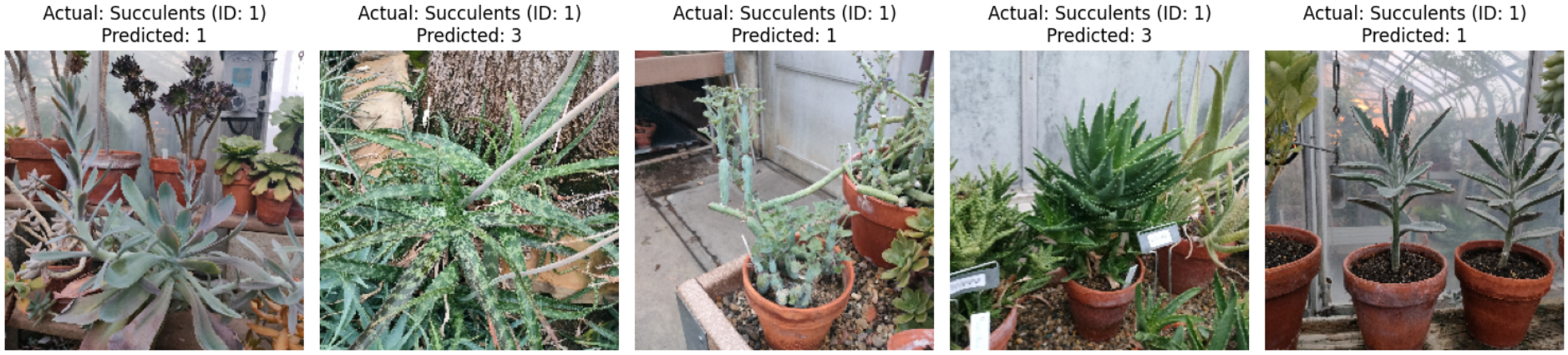
Actual: Cacti (ID: 0)
Predicted: 1

Actual: Cacti (ID: 0)
Predicted: 1

Actual: Cacti (ID: 0)
Predicted: 1

Actual: Cacti (ID: 0)
Predicted: 3

Class Succulents (ID: 1) Images



Class Tress (ID: 2) Images

Class plants (ID: 3) Images

Ans 5. -

1. As seen from the above output images, the classification accuracy is very low, with majority of classes being mis-classified.
2. 'Tree' class is being classified with a very high accuracy, whereas the other classes are being classified very poorly.
3. 'Plant' class is being classified with the higgest in-accuracy, where it is unable to classify any of the 'Plants' appropriately.
4. Therefore, we can see that additional training data is required, as the classification Accuracy is very low due to which we can say that the model is underfitting.
5. Classification Accuracy can be increased by using additional trainnig data, or making the model more complex.