Project 3 - Artificial Neural Networks

In this project, you will work with a custom flower species dataset and a car object detection dataset. You will implement artificial neural networks for classification and object detection tasks with `TensorFlow` implementations.

The goal of this assignment include:

1. Set up datasets in format needed to run `TensorFlow` objects
2. Implement artificial neural networks with `TensorFlow`
3. Carry out experiments to select best ANN hyperparameters
4. Implement best training practices with `TensorFlow`, including checkpoints, early stopping, adaptive learning rate
5. Experiment with transfer learning using `TensorFlow`
6. Implement qualitative and quantitative performance measures
7. Report observations, propose business-centric solutions and propose mitigating strategies

Deliverables

As part of this project, you should deliver the following materials:

1. **4-page IEEE-format paper**. Write a paper with no more than 4 pages addressing the questions posed below. When writing this report, consider a business-oriented person as your reader (e.g. your PhD advisor, your internship manager, etc.). Tell *the story* for each datasets' goal and propose solutions by addressing (at least) the questions posed below.

2. **Python Code**. Create two separate Notebooks: (1) "training.ipynb" used for training and hyperparameter tuning, (2) "test.ipynb" for evaluating the final trained model in the test set. The "test.ipynb" should load all trained objects and simply evaluate the performance. So don't forget to **push the trained models** to your repository to allow us to run it.

All of your code should run without any errors and be well-documented.

3. **README.md file**. Edit the readme.md file in your repository and how to use your code. If there are user-defined parameters, your readme.md file must clearly indicate so and demonstrate how to use your code. **Consider the case where the user wants**

to utilize your code to run on a different test set. Indicate in your readme.md file how this can be achieved.

This is an **individual assignment**.

These deliverables are **due Wednesday, December 6 @ 11:59pm**. Late submissions will not be accepted, so please plan accordingly.

Dataset 1: Flower Species Dataset

The training dataset is saved as a `numpy` array and contains a total of 1678 images from 10 classes. Each RBG image is of size $300 \times 300 \times 3$. The 10 classes and its label encodings are:

Flower Species	Roses	Magnolias	Lilies	Sunflowers	Orchids	Marigold	Hibiscus	Firebush	Pentas	Bougainvillea
Label	0	1	2	3	4	5	6	7	8	9

In [1]:
```
class_names = ['Roses', 'Magnolias', 'Lilies', 'Sunflowers', 'Orchids',
               'Marigold', 'Hibiscus', 'Firebush', 'Pentas', 'Bougainvillea']
```

The goal is to utilize the training images to train a flower species classifier (an artificial neural network architecture), and make predictions for the test set.

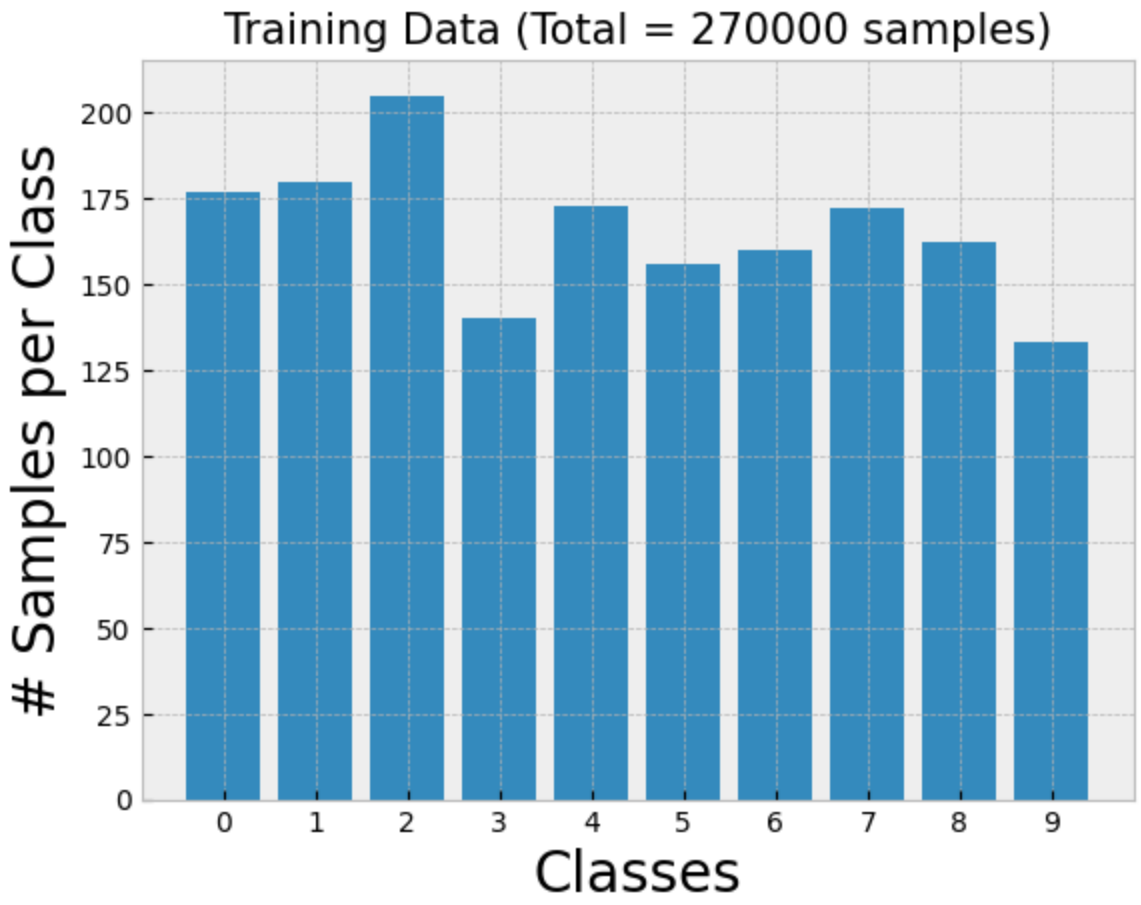
Let's visualize the dataset:

In [2]:
```
import numpy as np
import numpy.random as npr
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')

# Loading Training Data
# X_train = np.load('flower_species_classification/data_train.npy').T
# t_train = np.load('flower_species_classification/labels_train.npy')
```

```
X_train = np.load("C:/Users/barla/Music/flower_species_classification/data_train.npy").T
t_train = np.load("C:/Users/barla/Music/flower_species_classification/labels_train.npy")


print(X_train.shape, t_train.shape)
```

```
(1658, 270000) (1658,)
```

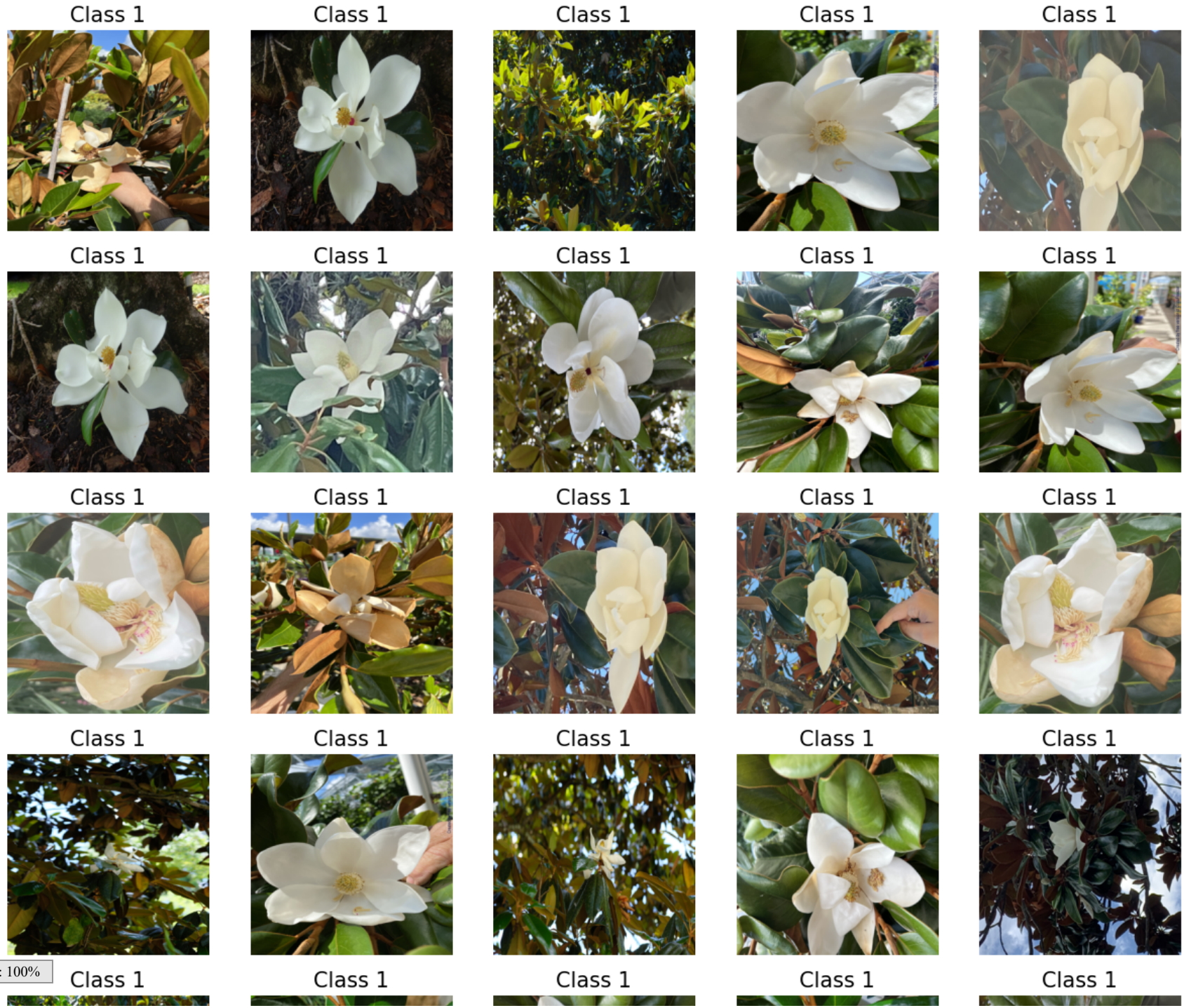
In [3]:
```
# Counting number samples per class
vals, counts = np.unique(t_train, return_counts=True)

plt.bar(vals, counts)
plt.xticks(range(10),range(10))
plt.xlabel('Classes',size=20)
plt.ylabel('# Samples per Class', size=20)
plt.title('Training Data (Total = '+str(X_train.shape[1])+' samples)',size=15);
```



In [3]:
```
# Displaying some random examples per class

for i in range(0,10):
    rnd_sample = npr.permutation(np.where(t_train==i)[0])
    fig=plt.figure(figsize=(15,15))
    for j in range(25):
        fig.add_subplot(5,5,j+1)
        plt.imshow(X_train[rnd_sample[j],:].reshape((300,300,3)))
        plt.axis('off');plt.title('Class '+str(int(t_train[rnd_sample[j]])),size=15)
    plt.show()
    print('\n\n')
```

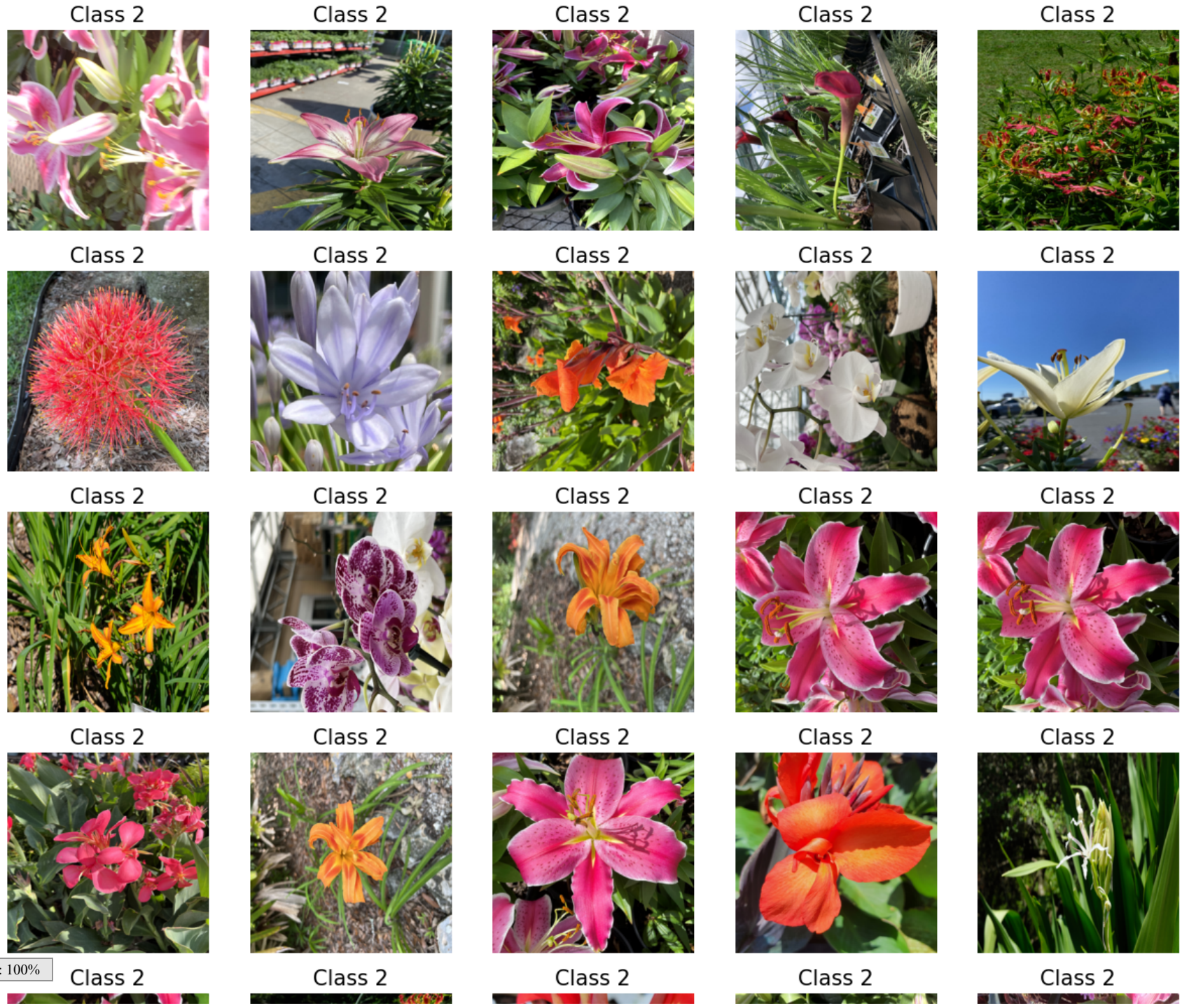






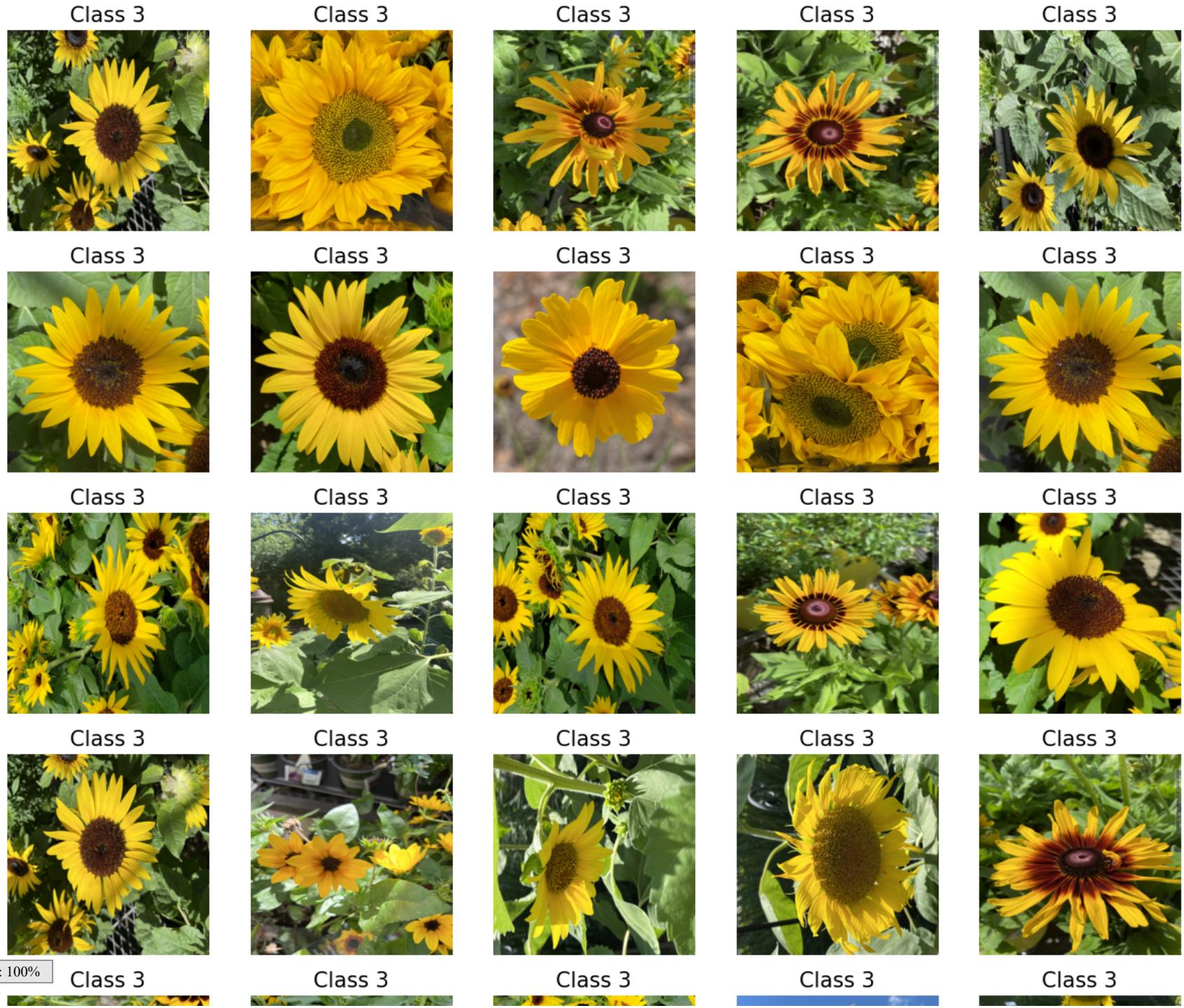




































Assignment

1. Train an artificial neural network for **flower species classification** using the training set of dataset 1.
    - Experiment with several hyperparameters of the network.
    - You are welcome to use any architecture or approach you wish.
    - Report performance results in training and validation sets.
    - Display the learning curves.

Training a Neural Network from scratch

In [3]:
```
import tensorflow as tf
import keras
import cv2
from sklearn.model_selection import train_test_split
```

In [6]:
```
# Reshaping and normalizing the data
X_train_scaled = X_train.reshape(-1, 300, 300, 3) / 255.0

# Reducing the pixel size of the images to facilitate limited computational resources
D = 300
X_train = np.array([cv2.resize(x.reshape(300,300,3),(D,D)) for x in X_train_scaled])
```

```
# Splitting the dataset
X_train, X_val, t_train, t_val = train_test_split(X_train, t_train,
                                                  test_size=0.2, random_state=43)
```

In [7]:
```
# Defining the model architecture
flower_model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, 3, activation='relu', input_shape=(D, D, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, 3, activation='relu', strides=2),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(256, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(512, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation='softmax')
])

flower_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 298, 298, 32)      896

 max_pooling2d (MaxPooling2  (None, 149, 149, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 147, 147, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 73, 73, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 36, 36, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 18, 18, 128)       0
 g2D)

 conv2d_3 (Conv2D)           (None, 16, 16, 256)       295168

 max_pooling2d_3 (MaxPoolin  (None, 8, 8, 256)         0
 g2D)

 conv2d_4 (Conv2D)           (None, 6, 6, 512)         1180160

 max_pooling2d_4 (MaxPoolin  (None, 3, 3, 512)         0
 g2D)

 flatten (Flatten)           (None, 4608)              0

 dense (Dense)               (None, 128)               589952

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 10)                650
```



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

```
Total params: 2167434 (8.27 MB)
Trainable params: 2167434 (8.27 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [8]:
```
# Compiling the model
# keras.optimizers.Nadam()
# keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
flower_model.compile(optimizer=keras.optimizers.Nadam(),
                     loss=keras.losses.SparseCategoricalCrossentropy(),
                     metrics=['accuracy'])
```

In [9]:
```
# Defining the Callbacks
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)
checkpoint_cb = tf.keras.callbacks.ModelCheckpoint("flower_model.h5", save_best_only=True)

# Training the model
history = flower_model.fit(X_train, t_train,
                           epochs=50,
                           batch_size=64,
                           validation_data=(X_val, t_val),
                           callbacks=[early_stopping, checkpoint_cb])
```

```
Epoch 1/50
21/21 [==============================] - 58s 3s/step - loss: 2.2911 - accuracy: 0.1214 - val_loss: 2.2229 - val_accur
acy: 0.1747
Epoch 2/50
C:\ProgramData\anaconda3\envs\jupyterlab\Lib\site-packages\keras\src\engine\training.py:3000: UserWarning: You are sa
ving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead
the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

```
21/21 [==============================] - 54s 3s/step - loss: 2.1828 - accuracy: 0.1606 - val_loss: 2.0624 - val_accur
acy: 0.2289
Epoch 3/50
21/21 [==============================] - 53s 3s/step - loss: 2.0638 - accuracy: 0.2029 - val_loss: 1.8959 - val_accur
acy: 0.2982
Epoch 4/50
21/21 [==============================] - 52s 2s/step - loss: 1.9848 - accuracy: 0.2443 - val_loss: 1.7507 - val_accur
acy: 0.3343
Epoch 5/50
21/21 [==============================] - 52s 2s/step - loss: 1.9220 - accuracy: 0.2617 - val_loss: 1.7165 - val_accur
acy: 0.3494
Epoch 6/50
21/21 [==============================] - 52s 2s/step - loss: 1.7957 - accuracy: 0.3175 - val_loss: 1.5555 - val_accur
acy: 0.4639
Epoch 7/50
21/21 [==============================] - 51s 2s/step - loss: 1.6513 - accuracy: 0.3658 - val_loss: 1.4372 - val_accur
acy: 0.5030
Epoch 8/50
21/21 [==============================] - 51s 2s/step - loss: 1.5856 - accuracy: 0.3937 - val_loss: 1.3625 - val_accur
acy: 0.5060
Epoch 9/50
21/21 [==============================] - 51s 2s/step - loss: 1.4551 - accuracy: 0.4404 - val_loss: 1.2481 - val_accur
acy: 0.5120
Epoch 10/50
21/21 [==============================] - 52s 2s/step - loss: 1.4290 - accuracy: 0.4600 - val_loss: 1.1271 - val_accur
acy: 0.6114
Epoch 11/50
21/21 [==============================] - 51s 2s/step - loss: 1.3449 - accuracy: 0.4962 - val_loss: 1.1109 - val_accur
acy: 0.6386
Epoch 12/50
21/21 [==============================] - 50s 2s/step - loss: 1.2514 - accuracy: 0.5234 - val_loss: 1.0303 - val_accur
acy: 0.6506
Epoch 13/50
21/21 [==============================] - 51s 2s/step - loss: 1.1243 - accuracy: 0.5830 - val_loss: 0.9214 - val_accur
acy: 0.6687
Epoch 14/50
21/21 [==============================] - 50s 2s/step - loss: 1.0654 - accuracy: 0.6169 - val_loss: 0.9393 - val_accur
acy: 0.6958
Epoch 15/50
21/21 [==============================] - 52s 2s/step - loss: 0.9960 - accuracy: 0.6312 - val_loss: 0.8412 - val_accur
acy: 0.7199
Epoch 16/50
```

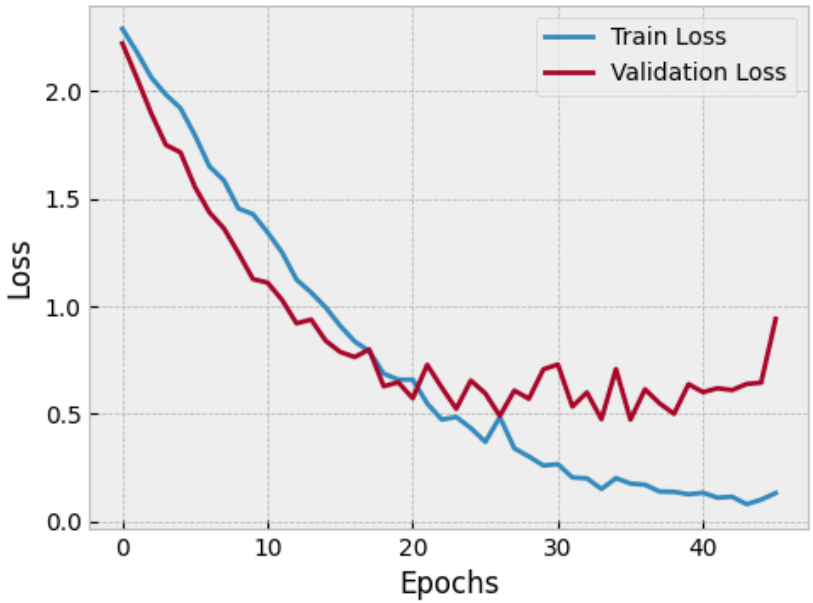
```
21/21 [==============================] - 53s 3s/step - loss: 0.9106 - accuracy: 0.6712 - val_loss: 0.7888 - val_accur
acy: 0.6988
Epoch 17/50
21/21 [==============================] - 51s 2s/step - loss: 0.8375 - accuracy: 0.6878 - val_loss: 0.7646 - val_accur
acy: 0.7139
Epoch 18/50
21/21 [==============================] - 51s 2s/step - loss: 0.7926 - accuracy: 0.6983 - val_loss: 0.8000 - val_accur
acy: 0.7259
Epoch 19/50
21/21 [==============================] - 51s 2s/step - loss: 0.6877 - accuracy: 0.7511 - val_loss: 0.6288 - val_accur
acy: 0.7801
Epoch 20/50
21/21 [==============================] - 51s 2s/step - loss: 0.6602 - accuracy: 0.7677 - val_loss: 0.6476 - val_accur
acy: 0.7741
Epoch 21/50
21/21 [==============================] - 51s 2s/step - loss: 0.6596 - accuracy: 0.7692 - val_loss: 0.5732 - val_accur
acy: 0.8163
Epoch 22/50
21/21 [==============================] - 51s 2s/step - loss: 0.5489 - accuracy: 0.8115 - val_loss: 0.7289 - val_accur
acy: 0.7801
Epoch 23/50
21/21 [==============================] - 51s 2s/step - loss: 0.4743 - accuracy: 0.8296 - val_loss: 0.6231 - val_accur
acy: 0.8163
Epoch 24/50
21/21 [==============================] - 51s 2s/step - loss: 0.4860 - accuracy: 0.8265 - val_loss: 0.5241 - val_accur
acy: 0.8434
Epoch 25/50
21/21 [==============================] - 51s 2s/step - loss: 0.4348 - accuracy: 0.8529 - val_loss: 0.6555 - val_accur
acy: 0.7982
Epoch 26/50
21/21 [==============================] - 51s 2s/step - loss: 0.3703 - accuracy: 0.8748 - val_loss: 0.5960 - val_accur
acy: 0.8193
Epoch 27/50
21/21 [==============================] - 51s 2s/step - loss: 0.4859 - accuracy: 0.8567 - val_loss: 0.4912 - val_accur
acy: 0.8524
Epoch 28/50
21/21 [==============================] - 51s 2s/step - loss: 0.3404 - accuracy: 0.8899 - val_loss: 0.6093 - val_accur
acy: 0.8494
Epoch 29/50
21/21 [==============================] - 51s 2s/step - loss: 0.3027 - accuracy: 0.8937 - val_loss: 0.5709 - val_accur
acy: 0.8524
Epoch 30/50
```

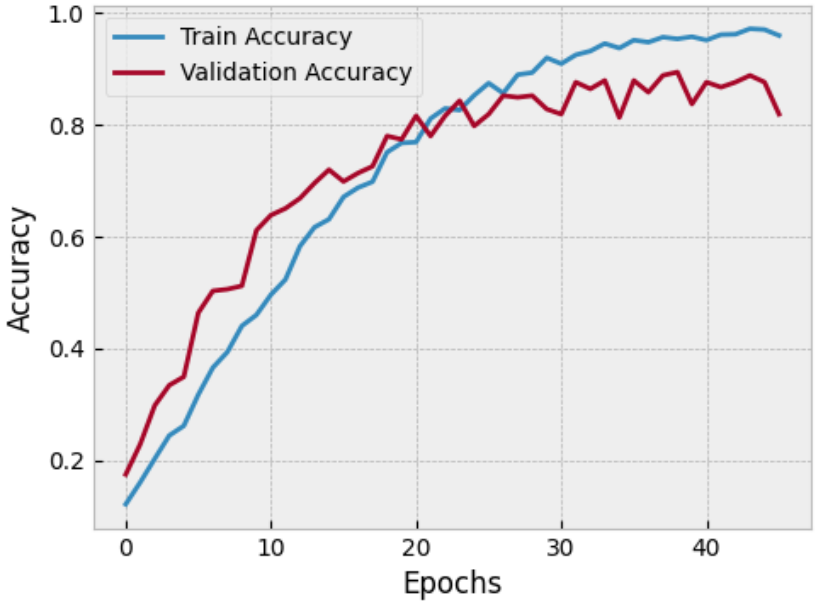
```
21/21 [==============================] - 51s 2s/step - loss: 0.2605 - accuracy: 0.9201 - val_loss: 0.7077 - val_accur
acy: 0.8283
Epoch 31/50
21/21 [==============================] - 51s 2s/step - loss: 0.2668 - accuracy: 0.9095 - val_loss: 0.7303 - val_accur
acy: 0.8193
Epoch 32/50
21/21 [==============================] - 51s 2s/step - loss: 0.2050 - accuracy: 0.9253 - val_loss: 0.5346 - val_accur
acy: 0.8765
Epoch 33/50
21/21 [==============================] - 51s 2s/step - loss: 0.2011 - accuracy: 0.9321 - val_loss: 0.6005 - val_accur
acy: 0.8645
Epoch 34/50
21/21 [==============================] - 51s 2s/step - loss: 0.1516 - accuracy: 0.9457 - val_loss: 0.4758 - val_accur
acy: 0.8795
Epoch 35/50
21/21 [==============================] - 52s 2s/step - loss: 0.2017 - accuracy: 0.9374 - val_loss: 0.7089 - val_accur
acy: 0.8133
Epoch 36/50
21/21 [==============================] - 52s 2s/step - loss: 0.1769 - accuracy: 0.9517 - val_loss: 0.4738 - val_accur
acy: 0.8795
Epoch 37/50
21/21 [==============================] - 51s 2s/step - loss: 0.1712 - accuracy: 0.9480 - val_loss: 0.6150 - val_accur
acy: 0.8584
Epoch 38/50
21/21 [==============================] - 52s 2s/step - loss: 0.1396 - accuracy: 0.9570 - val_loss: 0.5497 - val_accur
acy: 0.8886
Epoch 39/50
21/21 [==============================] - 51s 2s/step - loss: 0.1381 - accuracy: 0.9540 - val_loss: 0.5009 - val_accur
acy: 0.8946
Epoch 40/50
21/21 [==============================] - 51s 2s/step - loss: 0.1269 - accuracy: 0.9578 - val_loss: 0.6387 - val_accur
acy: 0.8373
Epoch 41/50
21/21 [==============================] - 51s 2s/step - loss: 0.1337 - accuracy: 0.9517 - val_loss: 0.6015 - val_accur
acy: 0.8765
Epoch 42/50
21/21 [==============================] - 52s 2s/step - loss: 0.1108 - accuracy: 0.9615 - val_loss: 0.6199 - val_accur
acy: 0.8675
Epoch 43/50
21/21 [==============================] - 51s 2s/step - loss: 0.1155 - accuracy: 0.9623 - val_loss: 0.6112 - val_accur
acy: 0.8765
Epoch 44/50
```

```
21/21 [==============================] - 51s 2s/step - loss: 0.0807 - accuracy: 0.9721 - val_loss: 0.6396 - val_accur
acy: 0.8886
Epoch 45/50
21/21 [==============================] - 52s 2s/step - loss: 0.1021 - accuracy: 0.9706 - val_loss: 0.6457 - val_accur
acy: 0.8765
Epoch 46/50
21/21 [==============================] - 51s 2s/step - loss: 0.1324 - accuracy: 0.9600 - val_loss: 0.9421 - val_accur
acy: 0.8193
```

In [10]:
```
# Evaluating the model
model = tf.keras.models.load_model("flower_model.h5") # Loading the best model
train_loss, train_accuracy = model.evaluate(X_train, t_train)
val_loss, val_accuracy = model.evaluate(X_val, t_val)
print(f"Train accuracy: {train_accuracy}, Validation accuracy: {val_accuracy}")

# Plotting the learning curves
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
42/42 [==============================] - 9s 212ms/step - loss: 0.0223 - accuracy: 0.9970
11/11 [==============================] - 2s 192ms/step - loss: 0.4738 - accuracy: 0.8795
Train accuracy: 0.9969834089279175, Validation accuracy: 0.8795180916786194
```





Now training on the entire training dataset,

In [19]:
```
flower_model_preds_train = model.predict(X_train)
flower_model_preds_val = model.predict(X_val)
```

```
42/42 [==============================] - 9s 218ms/step
11/11 [==============================] - 2s 194ms/step
```

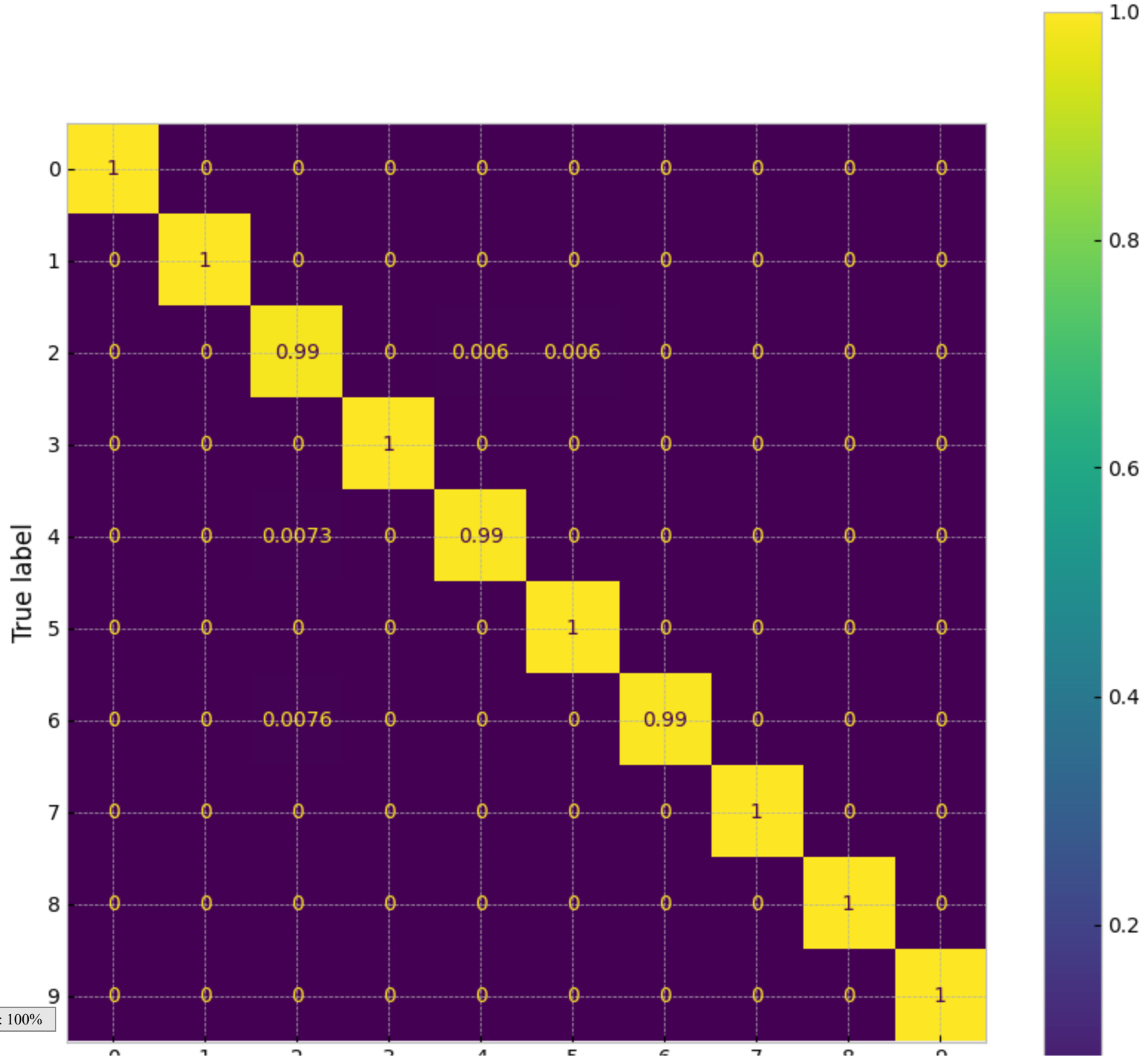
In [21]:
```
from sklearn.metrics import classification_report
print('Classification report on the Training set:')
print(classification_report(t_train,np.argmax(flower_model_preds_train, axis=1), target_names=class_names))
print('Classification report on the Validation set:')
print(classification_report(t_val,np.argmax(flower_model_preds_val, axis=1), target_names=class_names))
```

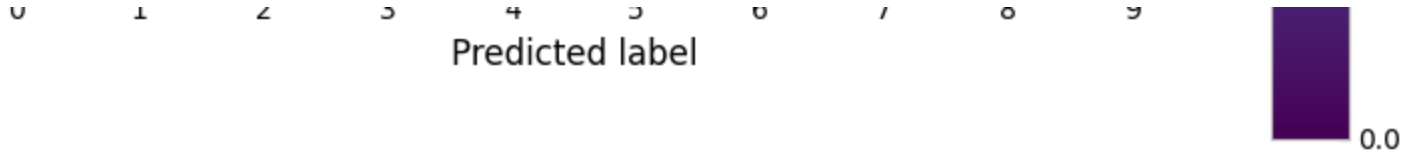
```
Classification report on the Training set:
              precision    recall  f1-score   support

       Roses       1.00      1.00      1.00       138
   Magnolias       1.00      1.00      1.00       147
      Lilies       0.99      0.99      0.99       168
  Sunflowers       1.00      1.00      1.00       108
     Orchids       0.99      0.99      0.99       137
    Marigold       0.99      1.00      1.00       123
    Hibiscus       1.00      0.99      1.00       131
    Firebush       1.00      1.00      1.00       132
      Pentas       1.00      1.00      1.00       140
Bougainvillea      1.00      1.00      1.00       102

    accuracy                           1.00      1326
   macro avg       1.00      1.00      1.00      1326
weighted avg       1.00      1.00      1.00      1326

Classification report on the Validation set:
              precision    recall  f1-score   support

       Roses       0.83      0.87      0.85        39
   Magnolias       0.91      0.91      0.91        33
      Lilies       0.67      0.78      0.72        37
  Sunflowers       1.00      1.00      1.00        32
     Orchids       0.88      0.81      0.84        36
    Marigold       0.94      1.00      0.97        33
    Hibiscus       0.85      0.76      0.80        29
    Firebush       0.95      0.95      0.95        40
      Pentas       0.86      0.86      0.86        22
Bougainvillea      0.96      0.84      0.90        31

    accuracy                           0.88       332
   macro avg       0.89      0.88      0.88       332
weighted avg       0.88      0.88      0.88       332
```

In [29]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

print('Confusion Matrix for the Training set:')
cm = confusion_matrix(t_train, np.argmax(flower_model_preds_train, axis=1), normalize='true')
disp = ConfusionMatrixDisplay(cm)
```

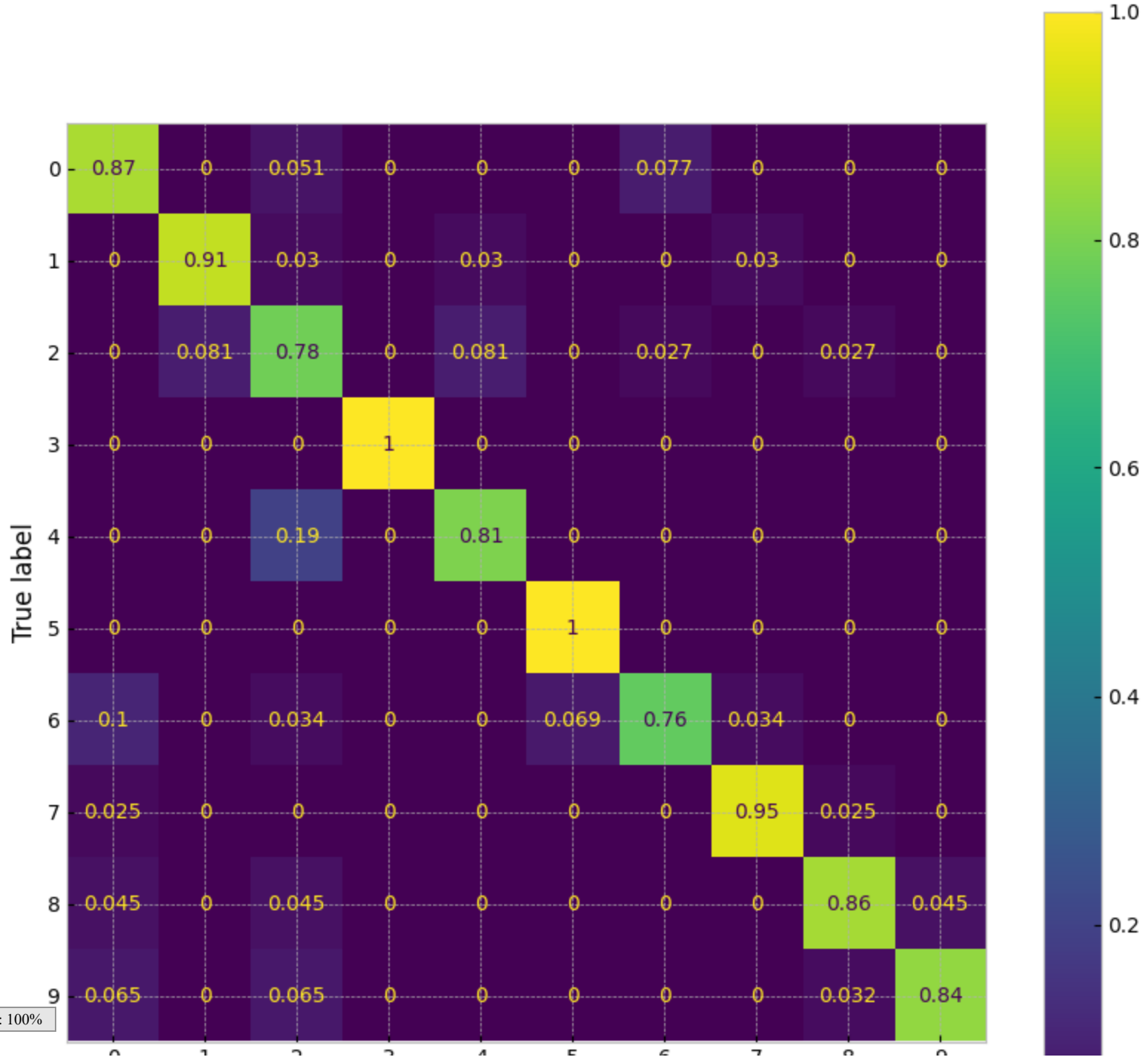
```
fig, ax = plt.subplots(figsize=(10,10))
disp.plot(ax=ax)
plt.show()

print('Confusion Matrix for the validation set:')
cm = confusion_matrix(t_val, np.argmax(flower_model_preds_val, axis=1), normalize='true')
disp = ConfusionMatrixDisplay(cm)
fig, ax = plt.subplots(figsize=(10,10))
disp.plot(ax=ax)
plt.show()
```

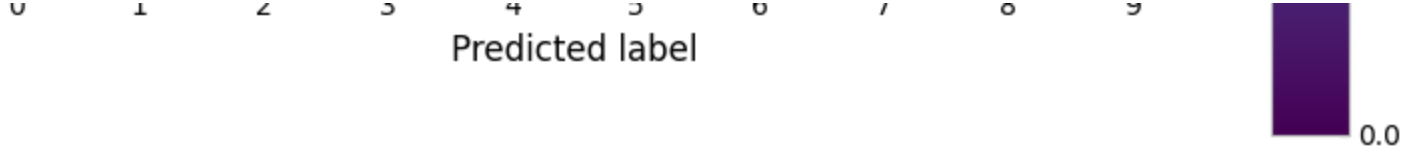
```
Confusion Matrix for the Training set:
```






```
Confusion Matrix for the validation set:
```





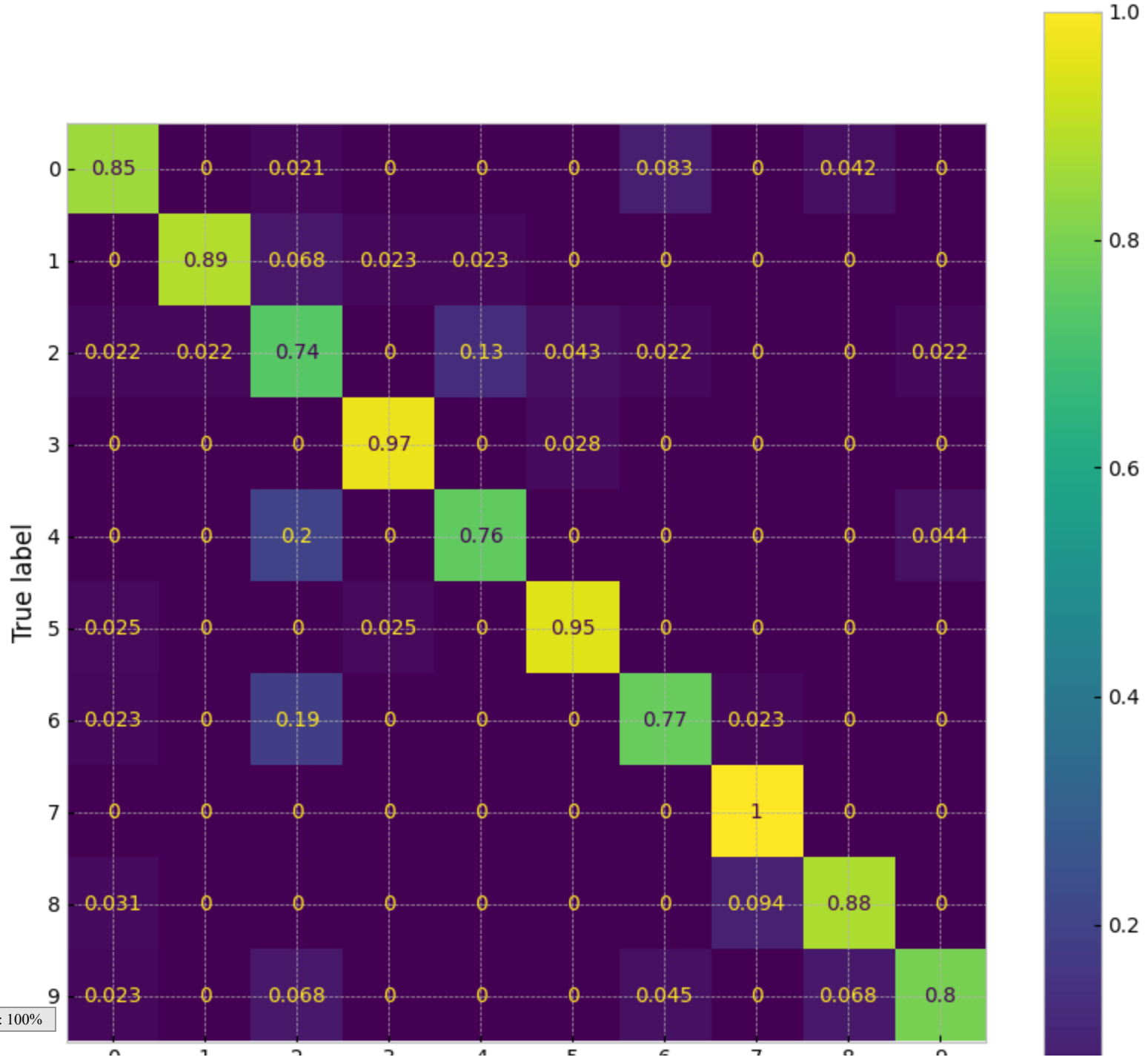
Testing the Neural Network from scratch

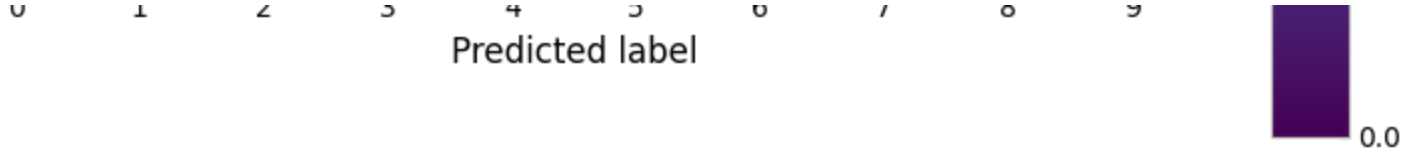
In [26]:
```
# Loading Training Data
# X_train = np.load('flower_species_classification/data_train.npy').T
# t_train = np.load('flower_species_classification/labels_train.npy')

X_test = np.load("C:/Users/barla/Music/flower_species_classification/data_test.npy").T
t_test = np.load("C:/Users/barla/Music/flower_species_classification/labels_test.npy")

# Reshaping and normalizing the data
X_test = X_test.reshape(-1, 300, 300, 3) / 255.0

# Loading the trained model
flower_model = keras.models.load_model('flower_model.h5')

flower_model_preds_test = flower_model.predict(X_test)

print(classification_report(t_test,np.argmax(flower_model_preds_test, axis=1), target_names=class_names))
```

```
13/13 [==============================] - 3s 223ms/step
              precision    recall  f1-score   support

       Roses       0.89      0.85      0.87        48
   Magnolias       0.97      0.89      0.93        44
      Lilies       0.59      0.74      0.65        46
  Sunflowers       0.95      0.97      0.96        36
     Orchids       0.83      0.76      0.79        45
    Marigold       0.93      0.95      0.94        40
    Hibiscus       0.82      0.77      0.80        43
     Firebush       0.90      1.00      0.95        37
       Pentas       0.85      0.88      0.86        32
Bougainvillea       0.92      0.80      0.85        44

    accuracy                           0.85       415
   macro avg       0.87      0.86      0.86       415
weighted avg       0.86      0.85      0.86       415
```

In [28]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

print('Confusion Matrix for the test set:')
cm = confusion_matrix(t_test, np.argmax(flower_model_preds_test, axis=1), normalize='true')
disp = ConfusionMatrixDisplay(cm)
fig, ax = plt.subplots(figsize=(10,10))
disp.plot(ax=ax)
plt.show()
```

```
Confusion Matrix for the test set:
```





Training using Transfer learning with ResNet50

In [4]:
```
# Loading Training Data
# X_train = np.load('flower_species_classification/data_train.npy').T
# t_train = np.load('flower_species_classification/labels_train.npy')

X_train = np.load("C:/Users/barla/Music/flower_species_classification/data_train.npy").T
t_train = np.load("C:/Users/barla/Music/flower_species_classification/labels_train.npy")

# Reshaping and normalizing the data
X_train_scaled = X_train.reshape(-1, 300, 300, 3) / 255.0

# Reducing the pixel size of the images to facilitate limited computational resources
D = 150
X_train = np.array([cv2.resize(x.reshape(300,300,3),(D,D)) for x in X_train_scaled])

# Splitting the dataset
X_train, X_val, t_train, t_val = train_test_split(X_train, t_train, stratify=t_train,
                                                  test_size=0.2, random_state=42)
```

In [5]:
```
# Importing trained ResNet50 model
base_model = keras.applications.ResNet50V2(
    input_shape=(D, D, 3),
    weights='imagenet',
    include_top=False)

for layer in base_model.layers[:-5]:
    layer.trainable = False

# Adding more layers
flower_resnet = tf.keras.models.Sequential([
    base_model,
    keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Flatten(),
```

```
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation='softmax')
])

flower_resnet.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50v2 (Functional)     (None, 5, 5, 2048)        23564800

 global_average_pooling2d (  (None, 2048)              0
 GlobalAveragePooling2D)

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 256)               524544

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 64)                16448

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 10)                650

=================================================================
Total params: 24106442 (91.96 MB)
Trainable params: 1596362 (6.09 MB)
Non-trainable params: 22510080 (85.87 MB)
_________________________________________________________________
```

In [6]:
```
# Compiling the model
flower_resnet.compile(optimizer=keras.optimizers.Nadam(),
                      loss=keras.losses.SparseCategoricalCrossentropy(),
                      metrics=['accuracy'])

# Defining the Callbacks
```

```
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)
checkpoint_cb = tf.keras.callbacks.ModelCheckpoint("flower_resnet.h5", save_best_only=True)

# Training the model
history = flower_resnet.fit(X_train, t_train,
                            validation_data=(X_val,t_val),
                            batch_size=64,
                            epochs=75,
                            callbacks=[early_stopping,checkpoint_cb])
```

```
Epoch 1/75
21/21 [==============================] - ETA: 0s - loss: 1.9765 - accuracy: 0.3560
```

```
C:\ProgramData\anaconda3\envs\jupyterlab\Lib\site-packages\keras\src\engine\training.py:3000: UserWarning: You are sa
ving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead
the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

```
21/21 [==============================] - 34s 1s/step - loss: 1.9765 - accuracy: 0.3560 - val_loss: 1.1474 - val_accur
acy: 0.6596
Epoch 2/75
21/21 [==============================] - 29s 1s/step - loss: 1.0986 - accuracy: 0.6456 - val_loss: 0.9730 - val_accur
acy: 0.7319
Epoch 3/75
21/21 [==============================] - 27s 1s/step - loss: 0.7608 - accuracy: 0.7632 - val_loss: 0.8096 - val_accur
acy: 0.8012
Epoch 4/75
21/21 [==============================] - 26s 1s/step - loss: 0.5674 - accuracy: 0.8205 - val_loss: 0.7909 - val_accur
acy: 0.8133
Epoch 5/75
21/21 [==============================] - 26s 1s/step - loss: 0.3793 - accuracy: 0.8906 - val_loss: 0.6034 - val_accur
acy: 0.8464
Epoch 6/75
21/21 [==============================] - 26s 1s/step - loss: 0.3023 - accuracy: 0.9042 - val_loss: 0.6562 - val_accur
acy: 0.8434
Epoch 7/75
21/21 [==============================] - 26s 1s/step - loss: 0.2532 - accuracy: 0.9261 - val_loss: 0.6657 - val_accur
acy: 0.8614
Epoch 8/75
21/21 [==============================] - 26s 1s/step - loss: 0.1939 - accuracy: 0.9487 - val_loss: 0.6920 - val_accur
acy: 0.8735
Epoch 9/75
21/21 [==============================] - 27s 1s/step - loss: 0.1665 - accuracy: 0.9570 - val_loss: 0.6028 - val_accur
acy: 0.8886
Epoch 10/75
21/21 [==============================] - 26s 1s/step - loss: 0.1198 - accuracy: 0.9668 - val_loss: 0.7767 - val_accur
acy: 0.8645
Epoch 11/75
21/21 [==============================] - 26s 1s/step - loss: 0.0978 - accuracy: 0.9721 - val_loss: 0.7237 - val_accur
acy: 0.8735
Epoch 12/75
21/21 [==============================] - 27s 1s/step - loss: 0.0694 - accuracy: 0.9804 - val_loss: 0.8708 - val_accur
acy: 0.8373
Epoch 13/75
21/21 [==============================] - 29s 1s/step - loss: 0.0701 - accuracy: 0.9796 - val_loss: 0.7403 - val_accur
acy: 0.8886
Epoch 14/75
21/21 [==============================] - 28s 1s/step - loss: 0.0524 - accuracy: 0.9834 - val_loss: 0.8643 - val_accur
acy: 0.8554
Epoch 15/75
```

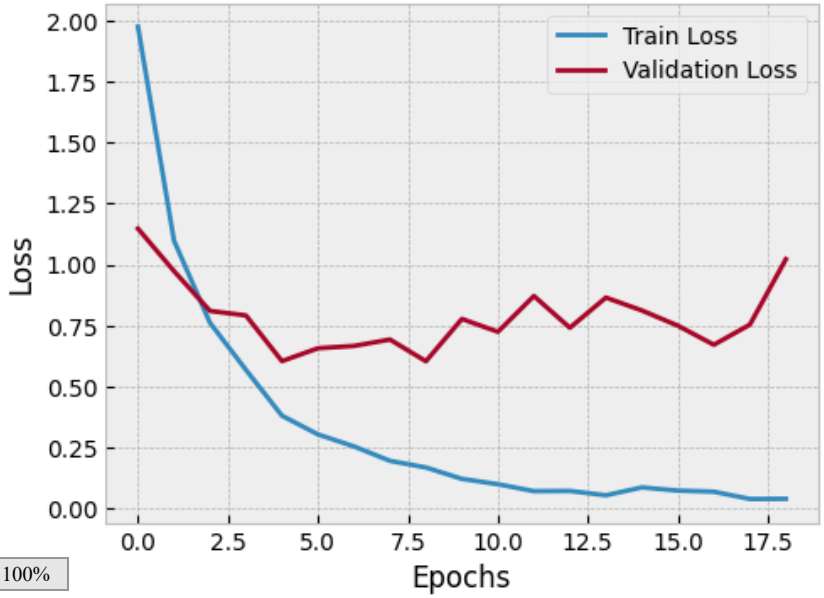
```
21/21 [==============================] - 27s 1s/step - loss: 0.0847 - accuracy: 0.9744 - val_loss: 0.8113 - val_accur
acy: 0.8765
Epoch 16/75
21/21 [==============================] - 28s 1s/step - loss: 0.0716 - accuracy: 0.9766 - val_loss: 0.7489 - val_accur
acy: 0.8705
Epoch 17/75
21/21 [==============================] - 27s 1s/step - loss: 0.0671 - accuracy: 0.9774 - val_loss: 0.6705 - val_accur
acy: 0.8886
Epoch 18/75
21/21 [==============================] - 26s 1s/step - loss: 0.0372 - accuracy: 0.9917 - val_loss: 0.7532 - val_accur
acy: 0.8735
Epoch 19/75
21/21 [==============================] - 26s 1s/step - loss: 0.0379 - accuracy: 0.9910 - val_loss: 1.0218 - val_accur
acy: 0.8584
```

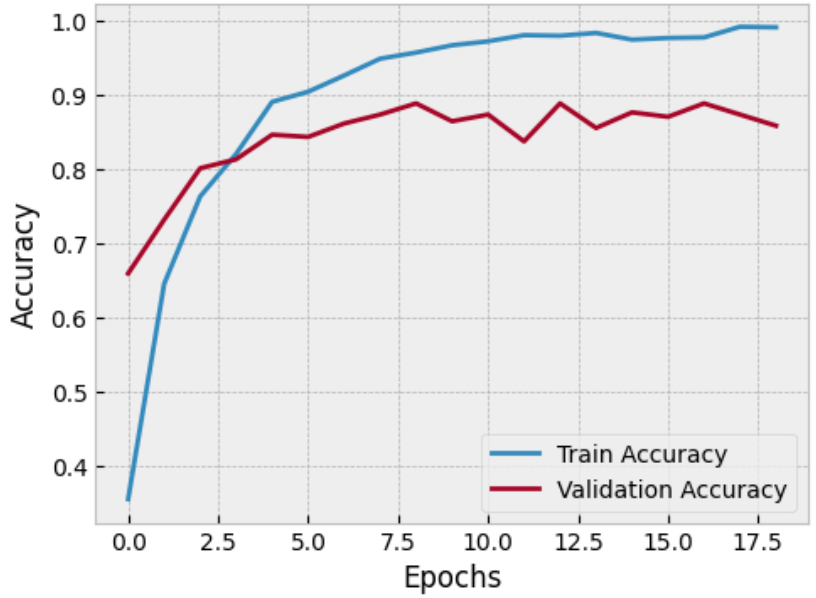
In [7]:

```
# Evaluating the model
model = tf.keras.models.load_model("flower_resnet.h5") # Loading the best model
train_loss, train_accuracy = model.evaluate(X_train, t_train)
val_loss, val_accuracy = model.evaluate(X_val, t_val)
print(f"Train accuracy: {train_accuracy}, Validation accuracy: {val_accuracy}")

# Plotting the learning curves
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

flower_resnet_preds_train = model.predict(X_train)
flower_resnet_preds_val = model.predict(X_val)

from sklearn.metrics import classification_report
```

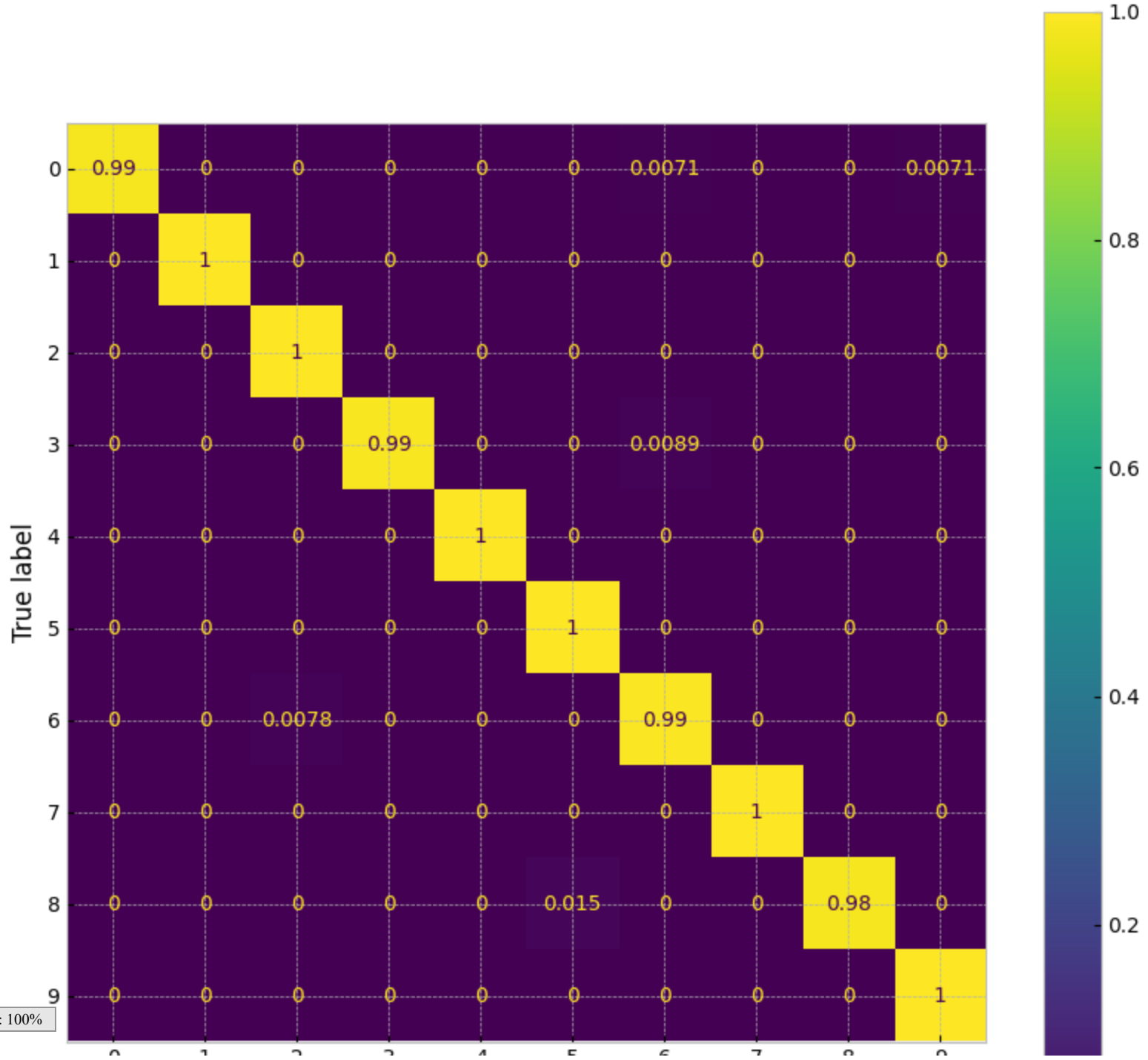
```
print('Classification report on the Training set:')
print(classification_report(t_train,np.argmax(flower_resnet_preds_train, axis=1), target_names=class_names))
print('Classification report on the Validation set:')
print(classification_report(t_val,np.argmax(flower_resnet_preds_val, axis=1), target_names=class_names))

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

print('Confusion Matrix for the Training set:')
cm = confusion_matrix(t_train, np.argmax(flower_resnet_preds_train, axis=1), normalize='true')
disp = ConfusionMatrixDisplay(cm)
fig, ax = plt.subplots(figsize=(10,10))
disp.plot(ax=ax)
plt.show()

print('Confusion Matrix for the validation set:')
cm = confusion_matrix(t_val, np.argmax(flower_resnet_preds_val, axis=1), normalize='true')
disp = ConfusionMatrixDisplay(cm)
fig, ax = plt.subplots(figsize=(10,10))
disp.plot(ax=ax)
plt.show()
```

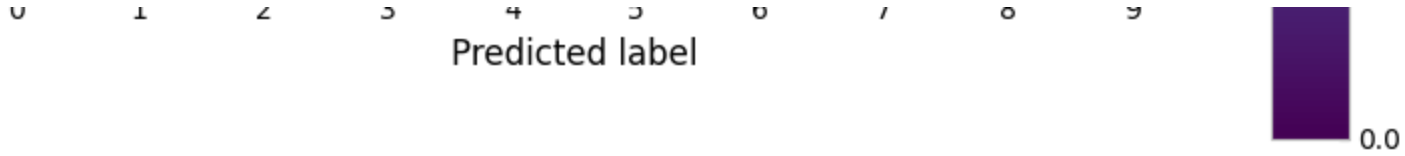
```
42/42 [==============================] - 24s 549ms/step - loss: 0.0175 - accuracy: 0.9955
11/11 [==============================] - 6s 532ms/step - loss: 0.6028 - accuracy: 0.8886
Train accuracy: 0.9954751133918762, Validation accuracy: 0.8885542154312134
```



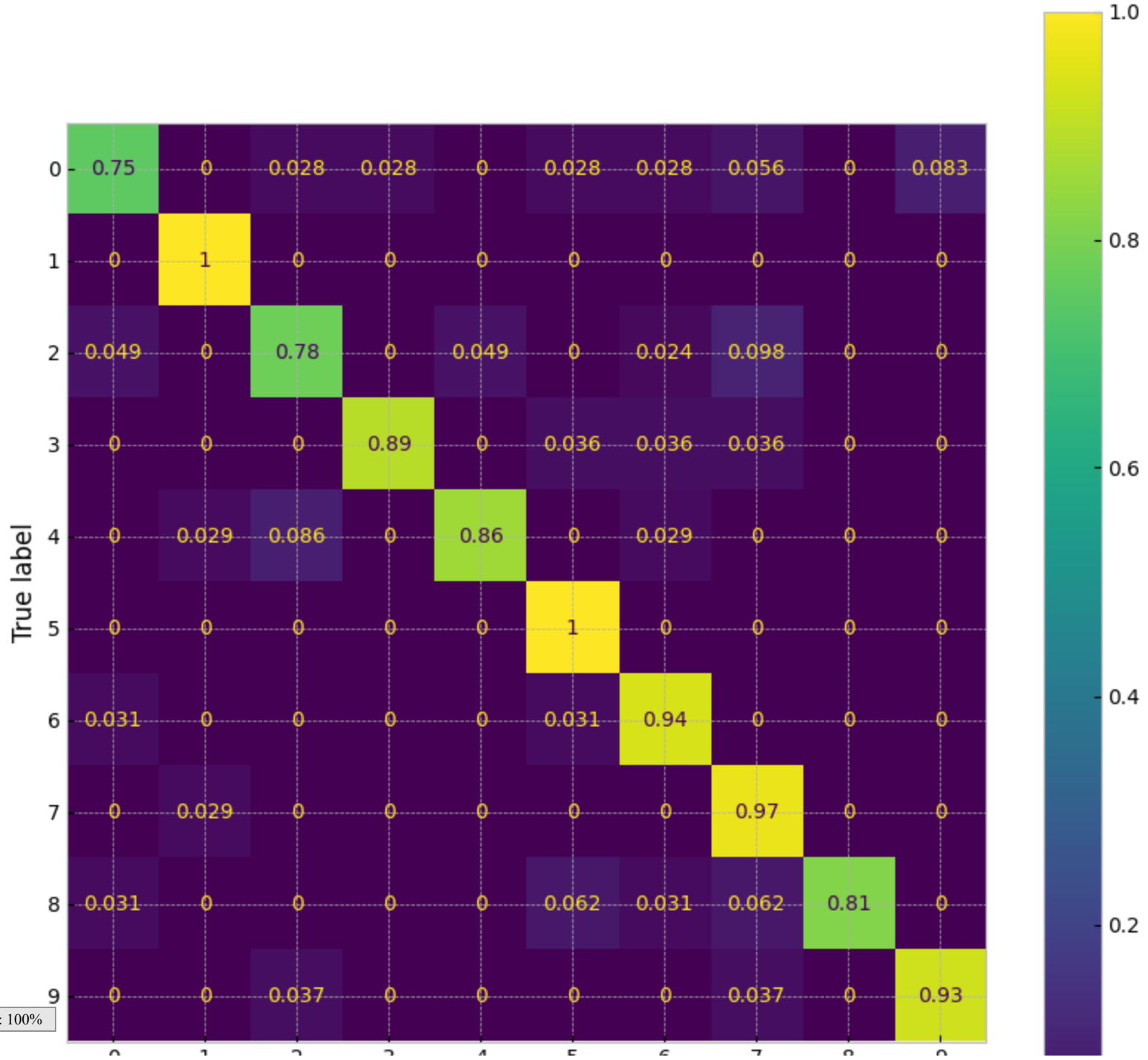


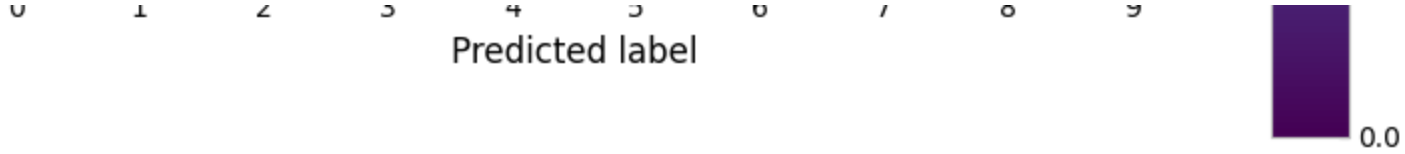
```
42/42 [==============================] - 24s 552ms/step
11/11 [==============================] - 6s 543ms/step
Classification report on the Training set:
               precision    recall  f1-score   support

        Roses       1.00      0.99      0.99       141
    Magnolias       1.00      1.00      1.00       144
       Lilies       0.99      1.00      1.00       164
   Sunflowers       1.00      0.99      1.00       112
      Orchids       1.00      1.00      1.00       138
     Marigold       0.98      1.00      0.99       125
     Hibiscus       0.98      0.99      0.99       128
     Firebush       1.00      1.00      1.00       138
       Pentas       1.00      0.98      0.99       130
Bougainvillea       0.99      1.00      1.00       106

     accuracy                           1.00      1326
    macro avg       1.00      1.00      1.00      1326
 weighted avg       1.00      1.00      1.00      1326

Classification report on the Validation set:
               precision    recall  f1-score   support

        Roses       0.87      0.75      0.81        36
    Magnolias       0.95      1.00      0.97        36
       Lilies       0.86      0.78      0.82        41
   Sunflowers       0.96      0.89      0.93        28
      Orchids       0.94      0.86      0.90        35
     Marigold       0.86      1.00      0.93        31
     Hibiscus       0.86      0.94      0.90        32
     Firebush       0.77      0.97      0.86        34
       Pentas       1.00      0.81      0.90        32
Bougainvillea       0.89      0.93      0.91        27

     accuracy                           0.89       332
    macro avg       0.90      0.89      0.89       332
 weighted avg       0.89      0.89      0.89       332

Confusion Matrix for the Training set:
```






```
Confusion Matrix for the validation set:
```





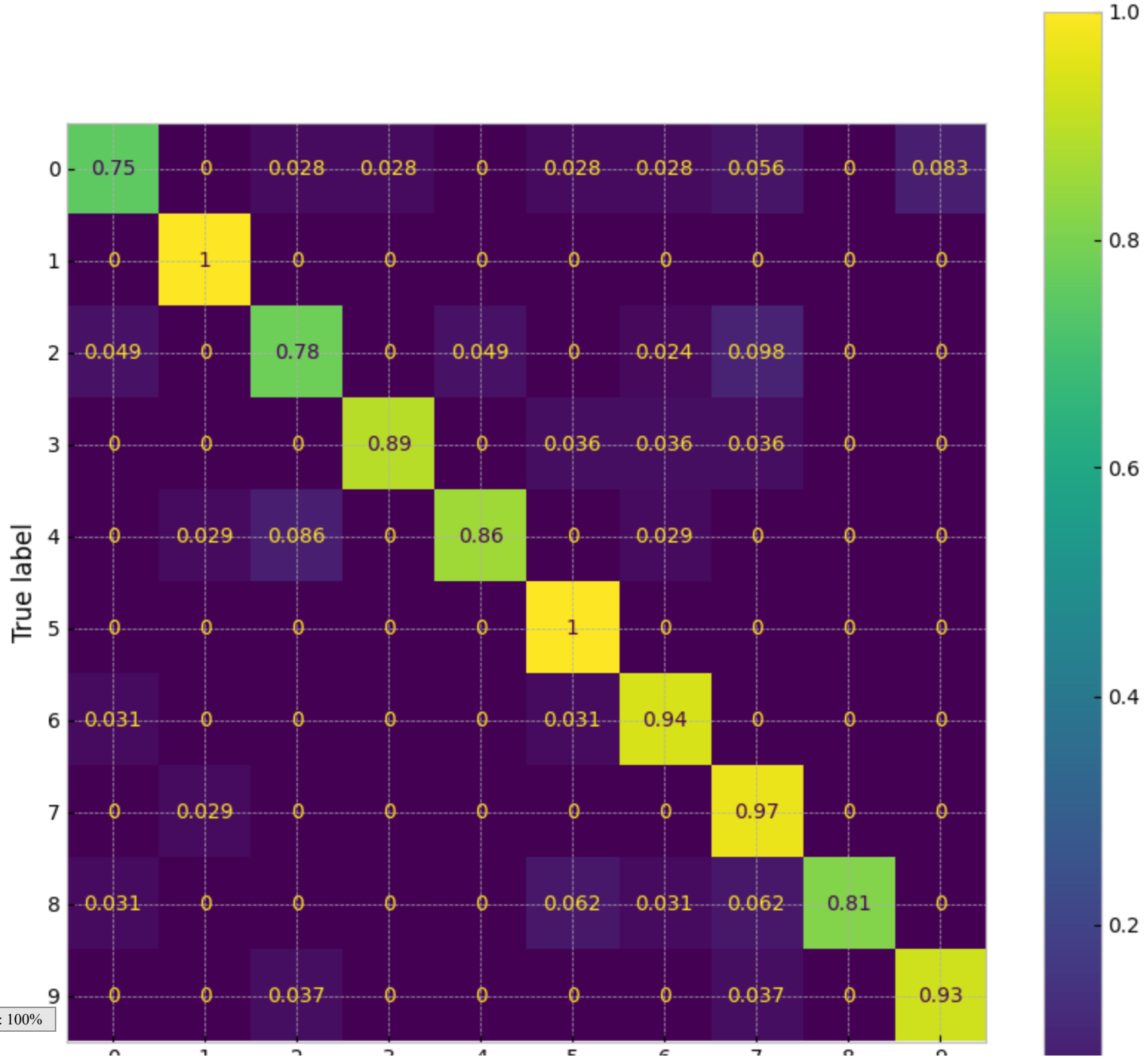
Testing using Transfer learning with ResNet50

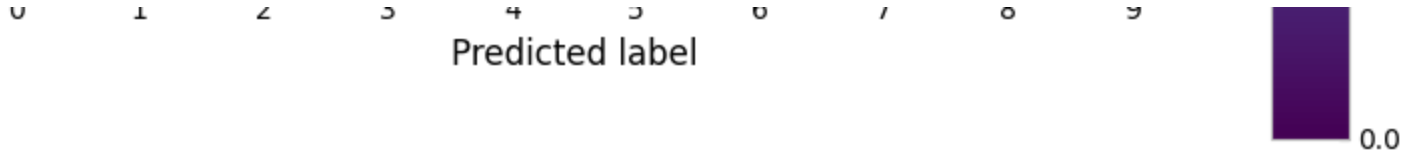
In [11]:
```
# Loading Training Data
# X_train = np.load('flower_species_classification/data_train.npy').T
# t_train = np.load('flower_species_classification/labels_train.npy')

X_test = np.load("C:/Users/barla/Music/flower_species_classification/data_test.npy").T
t_test = np.load("C:/Users/barla/Music/flower_species_classification/labels_test.npy")

# Reshaping and normalizing the data
X_test = X_test.reshape(-1, 300, 300, 3) / 255.0

# Reshaping and normalizing the data
X_test_scaled = X_test.reshape(-1, 300, 300, 3) / 255.0

# Reducing the pixel size of the images to facilitate limited computational resources
D = 150
X_test = np.array([cv2.resize(x.reshape(300,300,3),(D,D)) for x in X_test_scaled])

# Loading the trained model
flower_resnet = keras.models.load_model("flower_resnet.h5")

flower_resnet_preds_test = flower_resnet.predict(X_test)

print('Classification report on the Test set:')
print(classification_report(t_test,np.argmax(flower_resnet_preds_test, axis=1), target_names=class_names))
```

```
11/11 [==============================] - 5s 489ms/step - loss: 0.6028 - accuracy: 0.8886
Classification report on the Test set:
               precision    recall  f1-score   support

        Roses       0.87      0.75      0.81        36
    Magnolias       0.95      1.00      0.97        36
       Lilies       0.86      0.78      0.82        41
   Sunflowers       0.96      0.89      0.93        28
      Orchids       0.94      0.86      0.90        35
     Marigold       0.86      1.00      0.93        31
     Hibiscus       0.86      0.94      0.90        32
     Firebush       0.77      0.97      0.86        34
       Pentas       1.00      0.81      0.90        32
Bougainvillea       0.89      0.93      0.91        27

     accuracy                           0.89       332
    macro avg       0.90      0.89      0.89       332
 weighted avg       0.89      0.89      0.89       332
```

In [12]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

print('Confusion Matrix for the test set:')
cm = confusion_matrix(t_test, np.argmax(flower_resnet_preds_test, axis=1), normalize='true')
disp = ConfusionMatrixDisplay(cm)
fig, ax = plt.subplots(figsize=(10,10))
disp.plot(ax=ax)
plt.show()
```

```
Confusion Matrix for the test set:
```





Dataset 2: Car Detection Dataset

This dataset contains labeled annotations for 559 training samples. Each annotation corresponds to a bounding box of the object **car**.

The goal is to train an object (car) detection artificial neural network using the training samples, and make predictions for the images in test.

Let's visualize the data:

In [1]:
```
from PIL import Image
import cv2 # install opencv, if you don't already have it (https://pypi.org/project/opencv-python/)
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
bbox = pd.read_csv(r'C:\Users\barla\Music\car_detection_dataset\train_bounding_boxes.csv')
bbox
```

Out[2]:

	image	xmin	ymin	xmax	ymax
0	vid_4_1000.jpg	281.259045	187.035071	327.727931	223.225547
1	vid_4_10000.jpg	15.163531	187.035071	120.329957	236.430180
2	vid_4_10040.jpg	239.192475	176.764801	361.968162	236.430180
3	vid_4_10020.jpg	496.483358	172.363256	630.020260	231.539575
4	vid_4_10060.jpg	16.630970	186.546010	132.558611	238.386422
...	...	...	...	...	...
554	vid_4_9860.jpg	0.000000	198.321729	49.235251	236.223284
555	vid_4_9880.jpg	329.876184	156.482351	536.664239	250.497895
556	vid_4_9900.jpg	0.000000	168.295823	141.797524	239.176652
557	vid_4_9960.jpg	487.428988	172.233646	616.917699	228.839864
558	vid_4_9980.jpg	221.558631	182.570434	348.585579	238.192196

559 rows × 5 columns

In [3]:
```
N = len(bbox) # no. of training samples

# Create a numpy array with all images
for i in range(N):
    filename='car_detection_dataset/training_images/'+bbox['image'][i]
    image = np.array(Image.open(filename))
    image_col = image.ravel()[:,np.newaxis]

    if i==0:
        X_train = image_col
    else:
        X_train = np.hstack((X_train, image_col))

# Training feature matrices
X_train = X_train.T

# Training labels
```

```
t_train = bbox.drop('image', axis=1).round().to_numpy().astype(int)

X_train.shape, t_train.shape
```

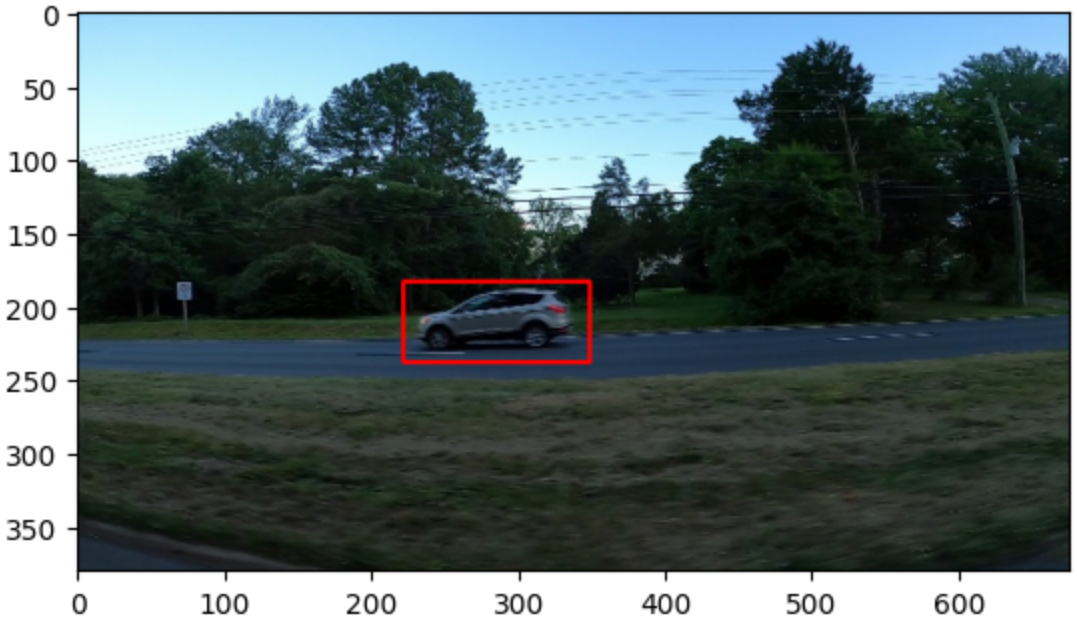
Out[3]: ((559, 770640), (559, 4))

Note that this code cell only collects data for images that contain a car within it. You may consider modifying to include all images. If no annotations is included, then assume there is no object and assign the target label [0,0,0,0].

In [4]:
```
# size of each RGB image
(Nx,Ny,Nz) = image.shape

Nx, Ny, Nz
```

Out[4]: (380, 676, 3)

In [6]:
```
# Example of object visualization using opencv rectangle function
idx=N-1
x= image

plt.imshow(x)
cv2.rectangle(x, (t_train[idx,0], t_train[idx,1]),
              (t_train[idx,2], t_train[idx,3]),
              (255, 0, 0), 2);
```



Assignment

2. Train an artificial neural network for **object detection** using the training set of dataset 2.

   - Experiment with several hyperparameters of the network.
   - You are welcome to use any architecture or approach you wish.
   - Report performance results in training and validation sets.
   - Display the learning curves.

Using YOLO v8 for Object Detection

Data Augmentation

In [ ]:
```
import pandas as pd
import os

# Converting the bounding box labels into YOLO format
def convert_to_yolo_format(row, image_width, image_height):
    x_center = ((row['xmax'] + row['xmin']) / 2) / image_width
    y_center = ((row['ymax'] + row['ymin']) / 2) / image_height
    width = (row['xmax'] - row['xmin']) / image_width
    height = (row['ymax'] - row['ymin']) / image_height
    return f'0 {x_center} {y_center} {width} {height}'

def main():
    csv_file = 'train_bounding_boxes.csv'
    df = pd.read_csv(csv_file)

    # Image dimensions
    image_width = 676
    image_height = 380

    output_folder = 'training_labels'
    os.makedirs(output_folder, exist_ok=True)

    for i, row in df.iterrows():
        yolo_label = convert_to_yolo_format(row, image_width, image_height)
        base_filename = os.path.splitext(row['image'])[0]
        output_file = os.path.join(output_folder, base_filename + '.txt')

        with open(output_file, 'a') as file:
            file.write(yolo_label + '\n')

main()
```

Splitting the Data into Training and Validation

In [ ]:
```
import os
import shutil
```

```
images = os.listdir('train/images/')

moving_list = []

for i in range(150):
    moving_list.append(images[i])

img_source = 'train/images/'
img_dest = 'validation/images/'

label_source = 'train/labels/'
label_dest = 'validation/labels/'

for img in moving_list:
    label = img.split('.jpg')[0]+'.txt'
    shutil.move(img_source+img,img_dest+img)
    shutil.move(label_source+label,label_dest+label)
```

Importing YOLOv8 and Training on the augmented test set

Only the images with bounding box labels (with atleast a car in it)

In [2]:
```
from ultralytics import YOLO

# Load a model
YOLO8_model = YOLO("yolov8n.pt")  # load a pretrained model (recommended for training)

# Use the model
YOLO8_model.train(data=r"C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\data_custom.yaml",
                  epochs=5)  # train the model
metrics = YOLO8_model.val()  # evaluate model performance on the validation set

path = YOLO8_model.export(format="torchscript")
```

```
Ultralytics YOLOv8.0.222 🚀 Python-3.11.6 torch-2.1.1+cpu CPU (AMD Ryzen 7 4800H with Radeon Graphics)
engine\trainer: task=detect, mode=train, model=yolov8n.pt, data=C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\data
_custom.yaml, epochs=5, patience=50, batch=16, imgsz=640, save=True, save_period=-1, cache=False, device=None, worker
s=8, project=None, name=train2, exist_ok=False, pretrained=True, optimizer=auto, verbose=True, seed=0, deterministic=
True, single_cls=False, rect=False, cos_lr=False, close_mosaic=10, resume=False, amp=True, fraction=1.0, profile=Fals
e, freeze=None, overlap_mask=True, mask_ratio=4, dropout=0.0, val=True, split=val, save_json=False, save_hybrid=Fals
e, conf=None, iou=0.7, max_det=300, half=False, dnn=False, plots=True, source=None, vid_stride=1, stream_buffer=Fals
e, visualize=False, augment=False, agnostic_nms=False, classes=None, retina_masks=False, show=False, save_frames=Fals
e, save_txt=False, save_conf=False, save_crop=False, show_labels=True, show_conf=True, show_boxes=True, line_width=No
ne, format=torchscript, keras=False, optimize=False, int8=False, dynamic=False, simplify=False, opset=None, workspace
=4, nms=False, lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmu
p_bias_lr=0.1, box=7.5, cls=0.5, dfl=1.5, pose=12.0, kobj=1.0, label_smoothing=0.0, nbs=64, hsv_h=0.015, hsv_s=0.7, h
sv_v=0.4, degrees=0.0, translate=0.1, scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, mosaic=1.0, mixu
p=0.0, copy_paste=0.0, cfg=None, tracker=botsort.yaml, save_dir=runs\detect\train2
Overriding model.yaml nc=80 with nc=1

                   from  n    params  module                                       arguments
  0                  -1  1       464  ultralytics.nn.modules.conv.Conv             [3, 16, 3, 2]
  1                  -1  1      4672  ultralytics.nn.modules.conv.Conv             [16, 32, 3, 2]
  2                  -1  1      7360  ultralytics.nn.modules.block.C2f             [32, 32, 1, True]
  3                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  4                  -1  2     49664  ultralytics.nn.modules.block.C2f             [64, 64, 2, True]
  5                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  6                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  7                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  8                  -1  1    460288  ultralytics.nn.modules.block.C2f             [256, 256, 1, True]
  9                  -1  1    164608  ultralytics.nn.modules.block.SPPF            [256, 256, 5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                  -1  1    148224  ultralytics.nn.modules.block.C2f             [384, 128, 1]
 13                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                  -1  1     37248  ultralytics.nn.modules.block.C2f             [192, 64, 1]
 16                  -1  1     36992  ultralytics.nn.modules.conv.Conv             [64, 64, 3, 2]
 17            [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                  -1  1    123648  ultralytics.nn.modules.block.C2f             [192, 128, 1]
 19                  -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
 20             [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                  -1  1    493056  ultralytics.nn.modules.block.C2f             [384, 256, 1]
 22        [15, 18, 21]  1    751507  ultralytics.nn.modules.head.Detect           [1, [64, 128, 256]]
Model summary: 225 layers, 3011043 parameters, 3011027 gradients, 8.2 GFLOPs
```

```
Transferred 319/355 items from pretrained weights
TensorBoard: Start with 'tensorboard --logdir runs\detect\train2', view at http://localhost:6006/
Freezing layer 'model.22.dfl.conv.weight'
train: Scanning C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\train\labels... 915 images, 0 backgrounds, 0 corrup
t:
train: New cache created: C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\train\labels.cache
val: Scanning C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\validation\labels... 150 images, 0 backgrounds, 0 corr
up
val: New cache created: C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\validation\labels.cache
Plotting labels to runs\detect\train2\labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' a
nd 'momentum' automatically...
optimizer: AdamW(lr=0.002, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias
(decay=0.0)
Image sizes 640 train, 640 val
Using 0 dataloader workers
Logging results to runs\detect\train2
Starting training for 5 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
        1/5         0G      1.658      3.541      1.163         70        640:   2%|▌          | 1/58 [00:06<05:47,
6.0
Downloading https://ultralytics.com/assets/Arial.ttf to 'C:\Users\barla\AppData\Roaming\Ultralytics\Arial.ttf'...
```

```
  0%|                                                                                | 0.00/755k [00:00<?, ?B/
s]
 22%|████████████████                                                              | 168k/755k [00:00<00:00, 1.66MB/
s]
 44%|████████████████████████████████                                               | 336k/755k [00:00<00:00, 763kB/
s]
 58%|███████████████████████████████████████                                        | 440k/755k [00:00<00:00, 676kB/
s]
 69%|████████████████████████████████████████████████                               | 520k/755k [00:00<00:00, 662kB/
s]
 78%|████████████████████████████████████████████████████████                       | 592k/755k [00:00<00:00, 674kB/
s]
 88%|████████████████████████████████████████████████████████████████               | 664k/755k [00:00<00:00, 681kB/
s]
100%|████████████████████████████████████████████████████████████████████████████| 755k/755k [00:01<00:00, 709kB/
s]
       1/5         0G      1.364      1.819      1.117         10        640: 100%|██████████| 58/58 [04:54<00:00,
5.
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 5/5 [00:10
<0
                   all        150        237      0.892     0.0348      0.747      0.452

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       2/5         0G       1.28      1.096      1.095         11        640: 100%|██████████| 58/58 [04:21<00:00,
4.
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 5/5 [00:09
<0
                   all        150        237      0.949      0.933      0.965      0.575

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       3/5         0G      1.252     0.9669      1.086          6        640: 100%|██████████| 58/58 [04:12<00:00,
4.
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 5/5 [00:09
<0
                   all        150        237      0.949      0.944      0.983      0.625

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       4/5         0G      1.254     0.8805       1.09          8        640: 100%|██████████| 58/58 [04:13<00:00,
4.
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 5/5 [00:09
<0
                   all        150        237      0.944      0.932       0.98      0.631

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
```

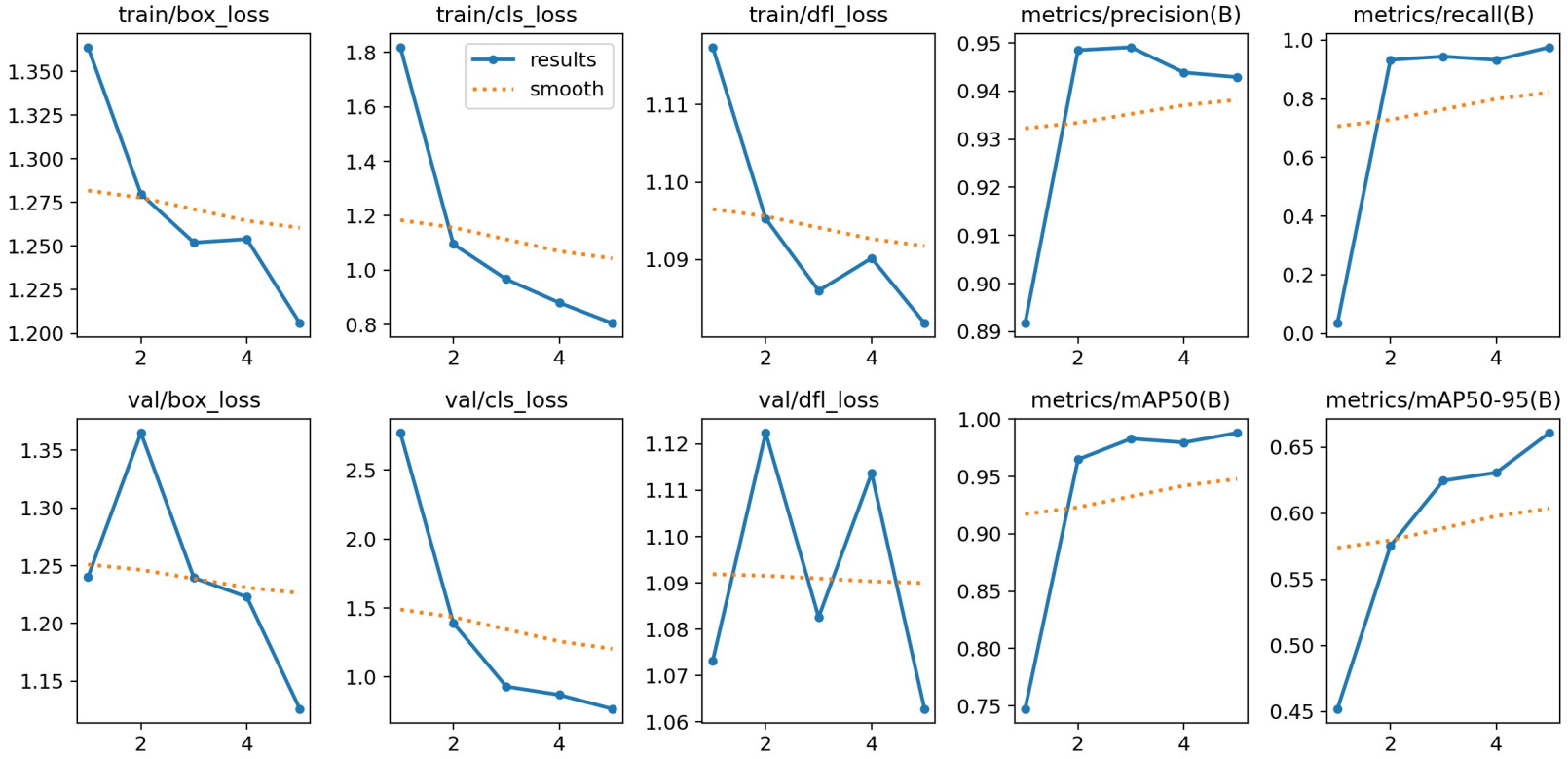
```
       5/5         0G      1.206      0.805      1.082         14        640: 100%|██████████| 58/58 [04:15<00:00,
4.
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 5/5 [00:09
<0
                   all        150        237      0.943      0.976      0.988      0.661
5 epochs completed in 0.381 hours.
Optimizer stripped from runs\detect\train2\weights\last.pt, 6.2MB
Optimizer stripped from runs\detect\train2\weights\best.pt, 6.2MB

Validating runs\detect\train2\weights\best.pt...
Ultralytics YOLOv8.0.222 🚀 Python-3.11.6 torch-2.1.1+cpu CPU (AMD Ryzen 7 4800H with Radeon Graphics)
Model summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 5/5 [00:08
<0
                   all        150        237      0.943      0.976      0.988      0.661
Speed: 1.2ms preprocess, 44.5ms inference, 0.0ms loss, 1.1ms postprocess per image
Results saved to runs\detect\train2
Ultralytics YOLOv8.0.222 🚀 Python-3.11.6 torch-2.1.1+cpu CPU (AMD Ryzen 7 4800H with Radeon Graphics)
Model summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs
val: Scanning C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\validation\labels.cache... 150 images, 0 backgrounds,
0
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 10/10 [00:
08
                   all        150        237      0.943      0.976      0.988      0.661
Speed: 1.1ms preprocess, 44.4ms inference, 0.0ms loss, 1.2ms postprocess per image
Results saved to runs\detect\train22
Ultralytics YOLOv8.0.222 🚀 Python-3.11.6 torch-2.1.1+cpu CPU (AMD Ryzen 7 4800H with Radeon Graphics)

PyTorch: starting from 'runs\detect\train2\weights\best.pt' with input shape (1, 3, 640, 640) BCHW and output shape
(s) (1, 5, 8400) (5.9 MB)

TorchScript: starting export with torch 2.1.1+cpu...
TorchScript: export success ✅ 1.9s, saved as 'runs\detect\train2\weights\best.torchscript' (11.9 MB)

Export complete (3.6s)
Results saved to C:\Users\barla\Music\runs\detect\train2\weights
Predict:         yolo predict task=detect model=runs\detect\train2\weights\best.torchscript imgsz=640
Validate:        yolo val task=detect model=runs\detect\train2\weights\best.torchscript imgsz=640 data=C:\Users\barla
\Music\CAR DETECTION\CAR DETECTION\data_custom.yaml
Visualize:       https://netron.app
```

In [6]:
```
YOLO8_model.info()
```

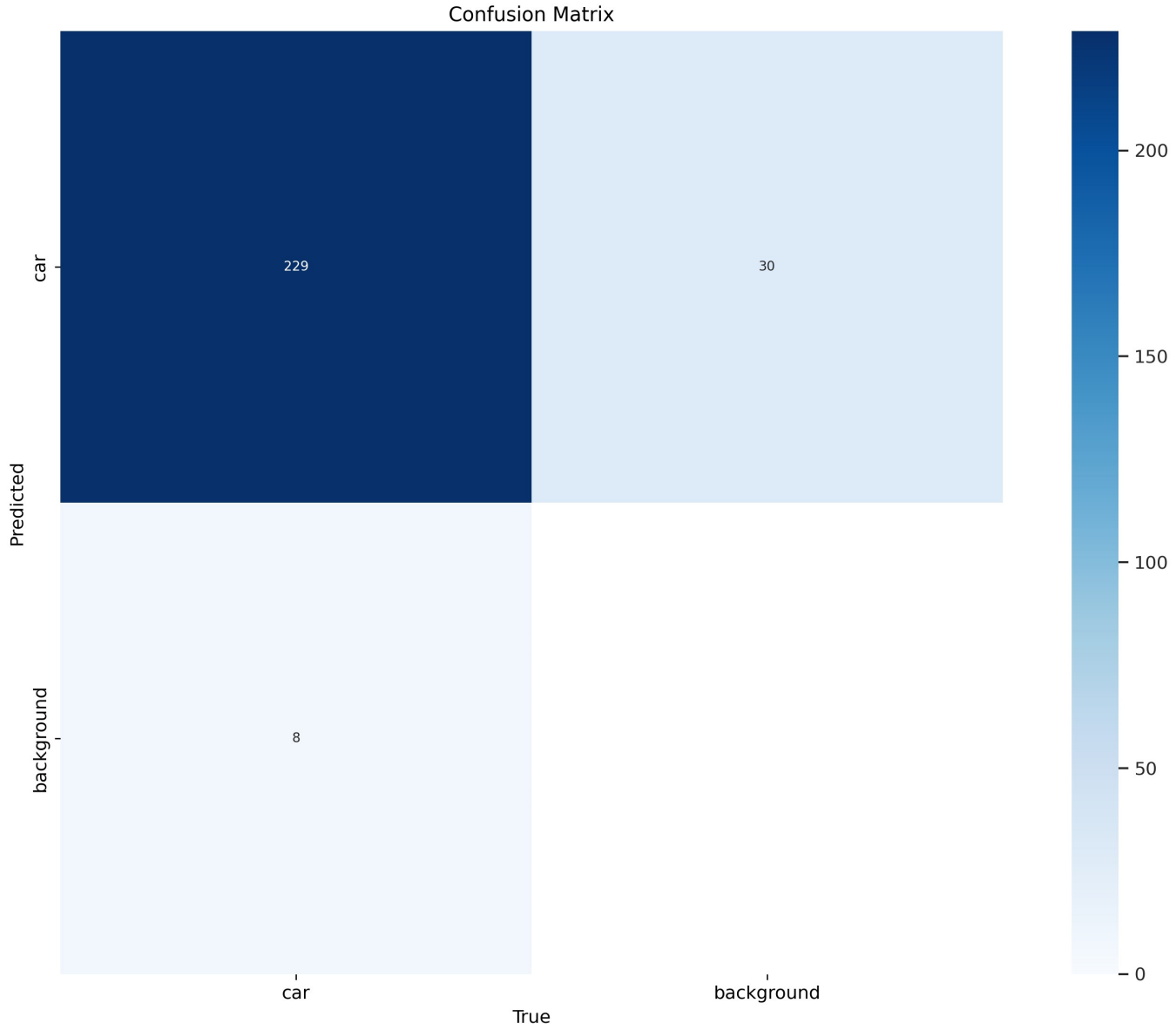
```
Model summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs
```

Out[6]: (168, 3005843, 0, 8.0851968)

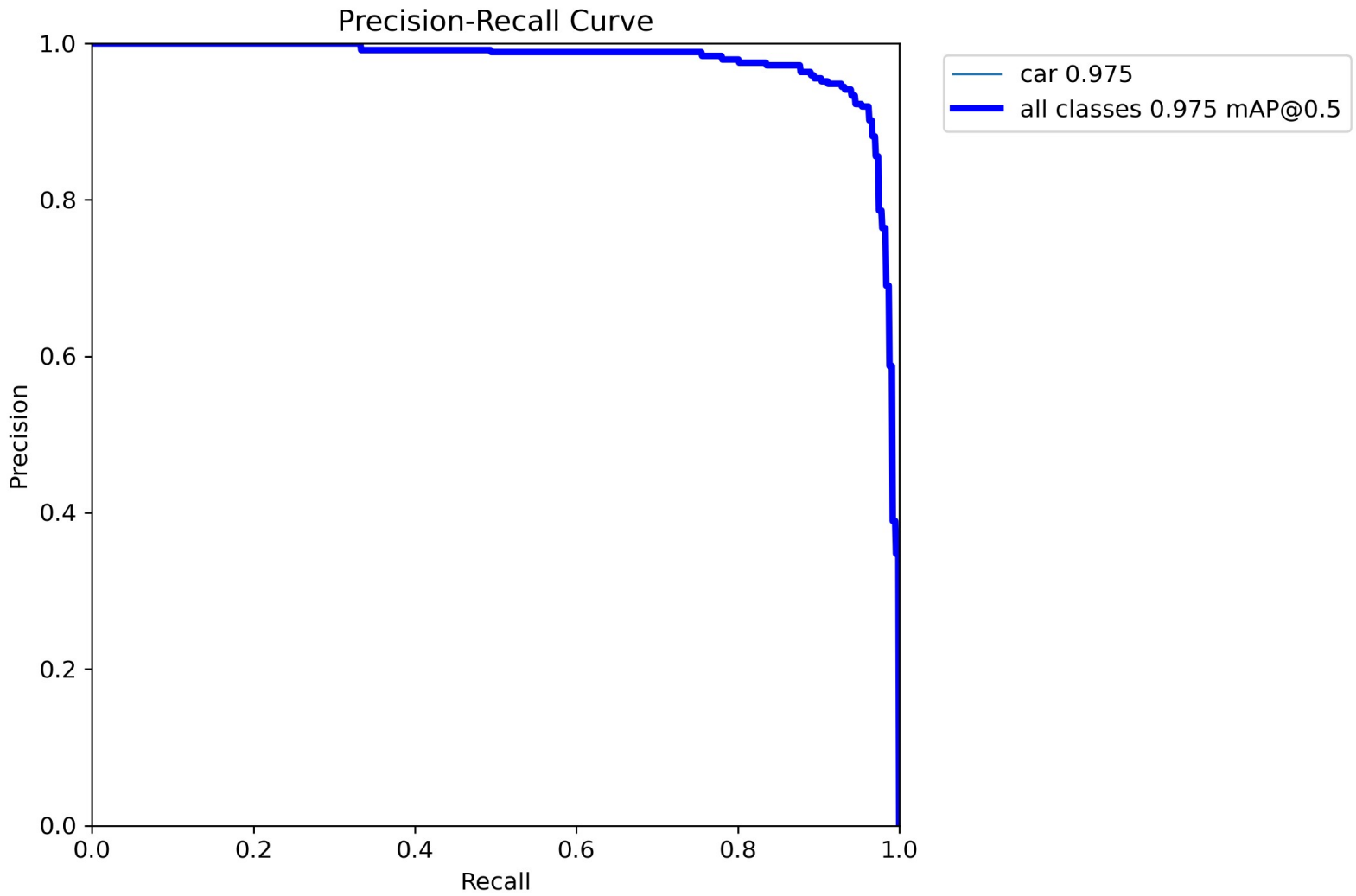
Performance in Training:



Confusion Matrix:



Precision-Recall curve:



Validation Examples:

Testing the YOLOv8 on the test set

In [1]:
```
from ultralytics import YOLO

# Importing the model
Yolo_trained = YOLO(r"C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\runs\detect\train\weights\best.torchscript")

# Testing the model on the Test set
results = Yolo_trained(r"C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images/")
```

```
Loading C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\runs\detect\train\weights\best.torchscript for TorchScript i
nference...

image 1/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_25100.jpg: 640x640 (no detections),
245.0ms
image 2/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_25120.jpg: 640x640 (no detections),
134.1ms
image 3/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_25140.jpg: 640x640 (no detections),
65.0ms
image 4/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_25160.jpg: 640x640 (no detections),
64.6ms
image 5/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_25180.jpg: 640x640 (no detections),
66.5ms
image 6/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_25200.jpg: 640x640 (no detections),
69.0ms
image 7/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_25220.jpg: 640x640 (no detections),
65.0ms
image 8/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_25240.jpg: 640x640 (no detections),
67.2ms
image 9/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_25260.jpg: 640x640 (no detections),
63.5ms
image 10/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26320.jpg: 640x640 (no detection
s), 66.0ms
image 11/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26400.jpg: 640x640 (no detection
s), 67.5ms
image 12/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26420.jpg: 640x640 (no detection
s), 65.6ms
image 13/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26560.jpg: 640x640 1 car, 69.5ms
image 14/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26580.jpg: 640x640 1 car, 67.0ms
image 15/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26600.jpg: 640x640 2 cars, 69.5ms
image 16/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26620.jpg: 640x640 1 car, 69.5ms
image 17/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26640.jpg: 640x640 4 cars, 68.5ms
image 18/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26660.jpg: 640x640 1 car, 68.0ms
image 19/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26680.jpg: 640x640 3 cars, 63.0ms
image 20/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26700.jpg: 640x640 2 cars, 67.3ms
image 21/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26720.jpg: 640x640 3 cars, 67.5ms
image 22/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26740.jpg: 640x640 3 cars, 68.5ms
image 23/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26760.jpg: 640x640 5 cars, 66.0ms
image 24/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26780.jpg: 640x640 2 cars, 65.4ms
image 25/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26800.jpg: 640x640 3 cars, 68.5ms
image 26/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26820.jpg: 640x640 3 cars, 68.6ms
image 27/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26840.jpg: 640x640 1 car, 69.0ms
```

```
image 28/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26860.jpg: 640x640 2 cars, 68.6ms
image 29/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26880.jpg: 640x640 1 car, 68.5ms
image 30/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26900.jpg: 640x640 2 cars, 72.5ms
image 31/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26920.jpg: 640x640 2 cars, 67.0ms
image 32/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26940.jpg: 640x640 2 cars, 74.7ms
image 33/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26960.jpg: 640x640 1 car, 70.5ms
image 34/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_26980.jpg: 640x640 (no detection
s), 72.5ms
image 35/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27240.jpg: 640x640 1 car, 70.6ms
image 36/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27260.jpg: 640x640 (no detection
s), 68.0ms
image 37/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27280.jpg: 640x640 (no detection
s), 68.6ms
image 38/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27300.jpg: 640x640 1 car, 69.6ms
image 39/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27320.jpg: 640x640 2 cars, 74.6ms
image 40/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27360.jpg: 640x640 1 car, 75.5ms
image 41/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27380.jpg: 640x640 2 cars, 68.0ms
image 42/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27400.jpg: 640x640 2 cars, 67.5ms
image 43/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27420.jpg: 640x640 3 cars, 74.2ms
image 44/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27440.jpg: 640x640 3 cars, 69.2ms
image 45/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27460.jpg: 640x640 2 cars, 72.0ms
image 46/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27480.jpg: 640x640 3 cars, 68.0ms
image 47/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27500.jpg: 640x640 1 car, 73.7ms
image 48/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27520.jpg: 640x640 2 cars, 72.6ms
image 49/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27540.jpg: 640x640 4 cars, 72.5ms
image 50/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27560.jpg: 640x640 1 car, 72.5ms
image 51/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27580.jpg: 640x640 (no detection
s), 79.0ms
image 52/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27600.jpg: 640x640 (no detection
s), 82.0ms
image 53/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27620.jpg: 640x640 1 car, 73.2ms
image 54/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27640.jpg: 640x640 1 car, 68.6ms
image 55/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27660.jpg: 640x640 (no detection
s), 71.5ms
image 56/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27680.jpg: 640x640 (no detection
s), 67.1ms
image 57/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27700.jpg: 640x640 (no detection
s), 70.0ms
image 58/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27720.jpg: 640x640 (no detection
s), 72.6ms
image 59/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27740.jpg: 640x640 (no detection
s), 72.5ms
```

```
image 60/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27760.jpg: 640x640 (no detection
s), 68.7ms
image 61/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27780.jpg: 640x640 (no detection
s), 72.0ms
image 62/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27800.jpg: 640x640 1 car, 70.5ms
image 63/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27820.jpg: 640x640 (no detection
s), 69.6ms
image 64/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27840.jpg: 640x640 1 car, 67.6ms
image 65/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27860.jpg: 640x640 1 car, 82.5ms
image 66/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27880.jpg: 640x640 1 car, 68.0ms
image 67/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27900.jpg: 640x640 1 car, 65.6ms
image 68/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27920.jpg: 640x640 1 car, 67.6ms
image 69/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27940.jpg: 640x640 2 cars, 67.3ms
image 70/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27960.jpg: 640x640 (no detection
s), 70.0ms
image 71/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_27980.jpg: 640x640 (no detection
s), 67.0ms
image 72/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28000.jpg: 640x640 1 car, 66.5ms
image 73/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28020.jpg: 640x640 (no detection
s), 65.5ms
image 74/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28040.jpg: 640x640 (no detection
s), 66.0ms
image 75/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28060.jpg: 640x640 (no detection
s), 67.5ms
image 76/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28080.jpg: 640x640 (no detection
s), 67.5ms
image 77/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28180.jpg: 640x640 1 car, 71.6ms
image 78/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28260.jpg: 640x640 1 car, 65.0ms
image 79/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28320.jpg: 640x640 (no detection
s), 66.6ms
image 80/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28340.jpg: 640x640 (no detection
s), 63.5ms
image 81/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28360.jpg: 640x640 (no detection
s), 65.5ms
image 82/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28380.jpg: 640x640 1 car, 65.0ms
image 83/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28420.jpg: 640x640 1 car, 66.5ms
image 84/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28440.jpg: 640x640 1 car, 68.6ms
image 85/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28460.jpg: 640x640 (no detection
s), 66.5ms
image 86/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28480.jpg: 640x640 (no detection
s), 67.1ms
image 87/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28500.jpg: 640x640 1 car, 64.5ms
```

```
image 88/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28520.jpg: 640x640 3 cars, 64.7ms
image 89/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28540.jpg: 640x640 1 car, 65.5ms
image 90/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28560.jpg: 640x640 (no detection
s), 65.0ms
image 91/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28580.jpg: 640x640 (no detection
s), 66.6ms
image 92/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28600.jpg: 640x640 (no detection
s), 68.7ms
image 93/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28620.jpg: 640x640 (no detection
s), 66.5ms
image 94/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28640.jpg: 640x640 1 car, 77.0ms
image 95/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28660.jpg: 640x640 (no detection
s), 74.5ms
image 96/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28680.jpg: 640x640 (no detection
s), 65.5ms
image 97/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_28700.jpg: 640x640 (no detection
s), 62.5ms
image 98/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29000.jpg: 640x640 2 cars, 66.0ms
image 99/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29020.jpg: 640x640 1 car, 69.7ms
image 100/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29040.jpg: 640x640 1 car, 75.6ms
image 101/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29060.jpg: 640x640 (no detection
s), 71.5ms
image 102/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29080.jpg: 640x640 (no detection
s), 66.6ms
image 103/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29100.jpg: 640x640 (no detection
s), 69.0ms
image 104/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29400.jpg: 640x640 1 car, 68.6ms
image 105/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29420.jpg: 640x640 2 cars, 63.5ms
image 106/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29440.jpg: 640x640 2 cars, 66.6ms
image 107/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29460.jpg: 640x640 1 car, 64.0ms
image 108/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29480.jpg: 640x640 1 car, 67.6ms
image 109/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29500.jpg: 640x640 (no detection
s), 63.5ms
image 110/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29520.jpg: 640x640 (no detection
s), 66.6ms
image 111/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29540.jpg: 640x640 2 cars, 67.0ms
image 112/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29560.jpg: 640x640 1 car, 65.5ms
image 113/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29580.jpg: 640x640 2 cars, 68.6ms
image 114/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29600.jpg: 640x640 (no detection
s), 68.5ms
image 115/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29620.jpg: 640x640 (no detection
s), 72.0ms
```

```
image 116/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29640.jpg: 640x640 (no detection
s), 80.6ms
image 117/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29720.jpg: 640x640 1 car, 75.6ms
image 118/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29740.jpg: 640x640 1 car, 76.6ms
image 119/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29760.jpg: 640x640 1 car, 63.5ms
image 120/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29820.jpg: 640x640 1 car, 67.0ms
image 121/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29840.jpg: 640x640 1 car, 71.6ms
image 122/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29860.jpg: 640x640 (no detection
s), 71.2ms
image 123/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29880.jpg: 640x640 (no detection
s), 69.6ms
image 124/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29900.jpg: 640x640 (no detection
s), 62.0ms
image 125/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_29980.jpg: 640x640 (no detection
s), 66.0ms
image 126/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_30000.jpg: 640x640 (no detection
s), 64.5ms
image 127/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_30020.jpg: 640x640 (no detection
s), 67.6ms
image 128/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_30040.jpg: 640x640 (no detection
s), 66.0ms
image 129/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_30120.jpg: 640x640 (no detection
s), 64.5ms
image 130/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_30140.jpg: 640x640 (no detection
s), 63.5ms
image 131/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_30160.jpg: 640x640 (no detection
s), 64.5ms
image 132/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_30180.jpg: 640x640 (no detection
s), 61.0ms
image 133/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_30640.jpg: 640x640 1 car, 67.5ms
image 134/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_30660.jpg: 640x640 (no detection
s), 64.6ms
image 135/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_30680.jpg: 640x640 (no detection
s), 68.1ms
image 136/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_30700.jpg: 640x640 (no detection
s), 70.0ms
image 137/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_30720.jpg: 640x640 (no detection
s), 65.6ms
image 138/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_30740.jpg: 640x640 1 car, 72.5ms
image 139/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_30760.jpg: 640x640 1 car, 77.5ms
image 140/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_30820.jpg: 640x640 1 car, 67.0ms
image 141/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_30840.jpg: 640x640 (no detection
```

```
s), 66.1ms
image 142/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_30860.jpg: 640x640 1 car, 75.3ms
image 143/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_30920.jpg: 640x640 1 car, 75.5ms
image 144/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_30940.jpg: 640x640 1 car, 68.6ms
image 145/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31020.jpg: 640x640 3 cars, 67.0ms
image 146/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31040.jpg: 640x640 3 cars, 64.5ms
image 147/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31060.jpg: 640x640 1 car, 66.5ms
image 148/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31080.jpg: 640x640 2 cars, 60.6ms
image 149/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31100.jpg: 640x640 1 car, 67.0ms
image 150/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31120.jpg: 640x640 2 cars, 62.7ms
image 151/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31140.jpg: 640x640 1 car, 65.6ms
image 152/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31160.jpg: 640x640 1 car, 67.0ms
image 153/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31180.jpg: 640x640 (no detection
s), 67.0ms
image 154/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31200.jpg: 640x640 (no detection
s), 63.5ms
image 155/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31260.jpg: 640x640 (no detection
s), 63.6ms
image 156/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31280.jpg: 640x640 (no detection
s), 67.0ms
image 157/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31300.jpg: 640x640 (no detection
s), 68.4ms
image 158/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31360.jpg: 640x640 (no detection
s), 61.6ms
image 159/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31380.jpg: 640x640 (no detection
s), 67.6ms
image 160/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31400.jpg: 640x640 (no detection
s), 66.0ms
image 161/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31420.jpg: 640x640 (no detection
s), 60.6ms
image 162/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31480.jpg: 640x640 (no detection
s), 65.5ms
image 163/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31500.jpg: 640x640 (no detection
s), 65.0ms
image 164/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31520.jpg: 640x640 (no detection
s), 67.0ms
image 165/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31560.jpg: 640x640 1 car, 65.6ms
image 166/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31600.jpg: 640x640 2 cars, 68.6ms
image 167/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31620.jpg: 640x640 1 car, 68.5ms
image 168/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31640.jpg: 640x640 (no detection
s), 67.0ms
image 169/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31660.jpg: 640x640 (no detection
```

```
s), 67.7ms
image 170/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31680.jpg: 640x640 (no detection
s), 71.1ms
image 171/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31700.jpg: 640x640 1 car, 76.6ms
image 172/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_31720.jpg: 640x640 1 car, 72.0ms
image 173/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_400.jpg: 640x640 1 car, 66.0ms
image 174/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_420.jpg: 640x640 1 car, 65.5ms
image 175/175 C:\Users\barla\Music\CAR DETECTION\CAR DETECTION\testing_images\vid_5_440.jpg: 640x640 1 car, 66.5ms
Speed: 2.8ms preprocess, 69.7ms inference, 0.7ms postprocess per image at shape (1, 3, 640, 640)
```

Question 3 is required for completion for the EEL 5934 section only. Individuals in EEL 4930 are welcome to solve these tasks but no extra credit will be credited.

3. Discuss how you would validate performance in the test set given that no target labels are provided. Discuss also how you would address the case where no car is present in the image. For the latter, it may be useful consider all training images and add a fix target label (e.g. [0,0,0,0]) for all images without a car within it.

    - Consider the case where the exact bounding box location is not particularly the target. There an acceptable margin an error. Perhaps describe your approach in the context of overlapping Region of Interest (ROI).
    - You may use MakeSenseAI to create your own test labels (a small subset of test samples suffices) to demonstrate your validation metric.

For the Object Detection task, since there are no target labels provided for the test set, one simple way to validate is by visual inspection. But this process is not optimal for large datasets, so by choosing a subset of dataset and manually giving them labels, we can validate the performance based on the ground truth labels.

In the availability of bounding box labels, to validate the performance on the test set, we can use Mean Average Precision (mAP) and Intersection over Union (IoU) metrics, which take into account the overlapping Regions of Interest (ROIs).

IoU: Intersection over Union measures the overlap between predicted and actual bounding boxes. It's calculated by dividing the area of overlap by the area of union of these boxes. In scenarios where there's an acceptable margin of error in bounding box location, IoU helps determine how 'close' the predictions are to potential true positives. A threshold (like 0.5) is often set, above which a prediction is considered a true positive.

mAP: This metric evaluates the model's precision across different levels of recall, essentially assessing how well the model detects objects (cars, in this case) across varying confidence thresholds. For each confidence level, precision and recall are calculated, and the average precision (AP) is derived. mAP is the mean of these APs across all classes or IoU thresholds. It's crucial when exact bounding box locations aren't the target but rather the model's overall ability to detect cars with a reasonable degree of accuracy.

For images without cars, we can add a fixed target label [0,0,0,0] during training. This teaches the model to recognize 'no car' scenarios. During validation, if the model predicts a bounding box where none should exist, it would negatively impact the mAP, as this would be a false positive.

Test Function

For the test notebook, you will report performance in the test set for both dataset 1 and dataset 2.

- For dataset 1 (flower species dataset), report quantitative metrics.
- For dataset 2 (object detection), include qualitative measures with bounding box visualizations as well as your quantitative approach of choice (required for eel5934 only).

Your test Notebook should include:

1. Loading trained models from problems 1 and 3.
    - If the models are too big (which is to be expect especially if you are using transfer learning), upload the models' weights to a cloud service. Share the link in your readme.md file.
2. Make predictions to the respective test sets.
3. Include any necessary visualizations to include in the report.

Submit Your Solution

Confirm that you've successfully completed the assignment.

Along with the Notebook, include a PDF of the notebook with your solutions.

`add` and `commit` the final version of your work, and `push` your code to your GitHub repository.

Submit the URL of your GitHub Repository as your assignment submission on Canvas.
