Transfer Learning using InceptionResNetV2

Gayanthika Shankar

School of Computing and Data Science

gayanthika.s-26@scds.saiuniversity.edu.in

```
#import file

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

```
Mounted at /content/gdrive
```

```
#Importing necessary libraries and setting random seed
import tensorflow as tf
print(tf.__version__)

from tensorflow import keras
tf.random.set_seed(42)

import numpy as np
np.random.seed(42)

import matplotlib.pyplot as plt
%matplotlib inline

import glob
import PIL
from PIL import Image
```

```
2.17.1
```

```
from tqdm import tqdm
zip_ref = zipfile.ZipFile("/content/gdrive/MyDrive/DL_Data/Airplanes_Motorbikes_S
for file in tqdm(zip_ref.namelist()):
 zip_ref.extract(file, "/content/gdrive/MyDrive/DL_Data/")
zip_ref.close()
```

```
100%|██████████| 3332/3332 [26:54<00:00,  2.06it/s]
```

```
#Importing the images
imgFiles = glob.glob("/content/gdrive/MyDrive/DL_Data/Airplanes_Motorbikes_Schoon
for items in imgFiles[:8]:
  print(items)
```

```
/content/gdrive/MyDrive/DL_Data/Airplanes_Motorbikes_Schooners/Motorbikes/ima
/content/gdrive/MyDrive/DL_Data/Airplanes_Motorbikes_Schooners/Motorbikes/ima
/content/gdrive/MyDrive/DL_Data/Airplanes_Motorbikes_Schooners/Motorbikes/ima
/content/gdrive/MyDrive/DL_Data/Airplanes_Motorbikes_Schooners/Motorbikes/ima
/content/gdrive/MyDrive/DL_Data/Airplanes_Motorbikes_Schooners/Motorbikes/ima
/content/gdrive/MyDrive/DL_Data/Airplanes_Motorbikes_Schooners/Motorbikes/ima
/content/gdrive/MyDrive/DL_Data/Airplanes_Motorbikes_Schooners/Motorbikes/ima
/content/gdrive/MyDrive/DL_Data/Airplanes_Motorbikes_Schooners/Motorbikes/ima
```

Data Preprocessing and Labelling

Requirements

- Input shape: (299, 299, 3)
- 3 channels
- Width and height should be no smaller than 75
- Scale input pixels between -1 and 1

```
from tensorflow.keras.applications.inception_resnet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
X = []
y = []
for fName in imgFiles:
 img = load_img(fName, target_size=(299, 299))
 img_array = img_to_array(img)
 img_preprocessed = preprocess_input(img_array)
 X.append(img_preprocessed)
 label = fName.split("/")[-2]
 y.append(label)
# Convert lists to numpy arrays
X = np.array(X)
y = np.array(y)
```

```
#Check the first few entries and their type
print(f"Type of imgFiles: {type(imgFiles)}")
print(f"Length of imgFiles: {len(imgFiles)}")
print("First few entries:")
for f in list(imgFiles)[:3]:
    print(f"- {f}, type: {type(f)}")
```

```
Type of imgFiles: <class 'list'>
Length of imgFiles: 1661
First few entries:
- /content/gdrive/MyDrive/DL_Data/Airplanes_Motorbikes_Schooners/Motorbikes/ir
- /content/gdrive/MyDrive/DL_Data/Airplanes_Motorbikes_Schooners/Motorbikes/ir
- /content/gdrive/MyDrive/DL_Data/Airplanes_Motorbikes_Schooners/Motorbikes/ir
```

```
print(y)
```

```
['Motorbikes' 'Motorbikes' 'Motorbikes' ... 'schooner' 'schooner'
 'schooner']
```

```
class_counts = dict()

# Count images for each class
for file_path in imgFiles:
    class_name = file_path.split("/")[-2]
    if class_name not in class_counts:
        class_counts[class_name] = 1
    else:
        class_counts[class_name] += 1

for class_name, count in class_counts.items():
    print(f"Class: {class_name}, Count: {count}")
```

```
Class: Motorbikes, Count: 798
Class: airplanes, Count: 800
Class: schooner, Count: 63
```

```
from sklearn.preprocessing import LabelEncoder
lEncoder = LabelEncoder()
y = lEncoder.fit_transform(y)

print(set(y))
print(lEncoder.classes_)
```

```
{0, 1, 2}
['Motorbikes' 'airplanes' 'schooner']
```

```
X = np.array(X)
y = np.array(y)

print(X.shape)
print(y.shape)
```

```
(1661, 299, 299, 3)
(1661,)
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
                                                    stratify=y, random_state=42)
```

```
print("X_train_shape: {}".format(X_train.shape))
print("X_test_shape: {}".format(X_test.shape))
```

```
X_train_shape: (1245, 299, 299, 3)
X_test_shape: (416, 299, 299, 3)
```

```
mu = X_train.mean()
std = X_train.std()

X_train_std = (X_train-mu)/std
X_test_std = (X_test-mu)/std
```

```
X_train_std.shape
```

```
(1245, 299, 299, 3)
```

```
y_train.shape
```

```
(1245,)
```

```
X_train
```

```
         ...,
         [ 1.        ,  1.        ,  1.        ],
         [ 1.        ,  1.        ,  1.        ],
         [ 1.        ,  1.        ,  1.        ]],

        ...,

        [[ 1.        ,  1.        ,  1.        ],
         [ 1.        ,  1.        ,  1.        ],
         [ 1.        ,  1.        ,  1.        ],
         ...,
         [ 1.        ,  1.        ,  1.        ],
         [ 1.        ,  1.        ,  1.        ],
         [ 1.        ,  1.        ,  1.        ]],

        [[ 1.        ,  1.        ,  1.        ],
         [ 1           1            1           ]
```

```
  [ 1.        ,  1.        ,  1.        ],
  [ 1.        ,  1.        ,  1.        ],
  ...,
  [ 1.        ,  1.        ,  1.        ],
  [ 1.        ,  1.        ,  1.        ],
  [ 1.        ,  1.        ,  1.        ]],

 [[ 1.        ,  1.        ,  1.        ],
  [ 1.        ,  1.        ,  1.        ],
  [ 1.        ,  1.        ,  1.        ],
  ...,
  [ 1.        ,  1.        ,  1.        ],
  [ 1.        ,  1.        ,  1.        ],
  [ 1.        ,  1.        ,  1.        ]]],


[[[-0.7411765 , -0.09019607,  0.3176471 ],
  [-0.7411765 , -0.09019607,  0.3176471 ],
  [-0.7411765 , -0.09019607,  0.3176471 ],
  ...,
  [-0.7254902 , -0.12156862,  0.23921573],
  [-0.64705884, -0.1607843 ,  0.14509809],
  [ 0.78039217,  1.        ,  1.        ]],

 [[-0.7411765 , -0.09019607,  0.3176471 ],
  [-0.7411765 , -0.09019607,  0.3176471 ],
  [-0.7411765 , -0.09019607,  0.3176471 ],
  ...,
  [-0.7254902 , -0.12156862,  0.23921573],
  [-0.64705884, -0.1607843 ,  0.14509809],
  [ 0.78039217,  1.        ,  1.        ]],

 [[-0.7411765 , -0.09019607,  0.3176471 ],
  [-0.7411765 , -0.09019607,  0.3176471 ],
  [-0.7411765 , -0.09019607,  0.3176471 ],
  ...,
  [-0.7254902 , -0.12156862,  0.23921573],
  [-0.64705884, -0.1607843 ,  0.14509809],
  [ 0.78039217,  1.        ,  1.        ]],

 ...,

 [[-0.64705884, -0.00392157,  0.38823533],
  [-0.64705884, -0.00392157,  0.38823533],
#Load a single image and check its values before preprocessing
img = load_img(imgFiles[1], target_size=(299, 299))
img_array = img_to_array(img)
print("Original image values range:", img_array.min(), "to", img_array.max())
print("Sample of original image values:\n", img_array[0:2, 0:2])

#after preprocessing
img_preprocessed = preprocess_input(img_array)
print("\nPreprocessed image values range:", img_preprocessed.min(), "to", img_pre
print("Sample of preprocessed values:\n", img_preprocessed[0:2, 0:2])
```

```
Original image values range: 0.0 to 255.0
Sample of original image values:
 [[[255. 255. 255.]
  [255. 255. 255.]]

 [[255. 255. 255.]
  [255. 255. 255.]]]

Preprocessed image values range: -1.0 to 1.0
Sample of preprocessed values:
 [[[1. 1. 1.]
  [1. 1. 1.]]

 [[1. 1. 1.]
  [1. 1. 1.]]]
```

```
print("X_train shape:", X_train.shape)
print("Value range:", X_train.min(), "to", X_train.max())
```

```
X_train shape: (1245, 299, 299, 3)
Value range: -1.0 to 1.0
```

```
import matplotlib.pyplot as plt

def show_images(X, n=5):
    plt.figure(figsize=(15, 3))
    for i in range(n):
        plt.subplot(1, n, i+1)
        #Convert back from preprocessed form for visualization
        img = (X[i] + 1) / 2  #Scale range
        plt.imshow(img)
        plt.axis('off')
    plt.show()

show_images(X_train)
```

Building the model

```
import tensorflow as tf
print(tf.__version__)

keras.applications.InceptionResNetV2(
    include_top=True,
    weights="imagenet",
    input_tensor=None,
    input_shape=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
    name="inception_resnet_v2",
)
```

```
2.17.1
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicat
225209952/225209952 ━━━━━━━━━━━━━━━━━━━━ 6s 0us/step
<Functional name=inception_resnet_v2, built=True>
```

```
base_model3 = keras.applications.InceptionResNetV2(include_top=False,input_shape
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicat
219055592/219055592 ━━━━━━━━━━━━━━━━━━━━ 6s 0us/step
```

```
base_model3.trainable = False

for layer in base_model3.layers:
  layer.trainable = False
```

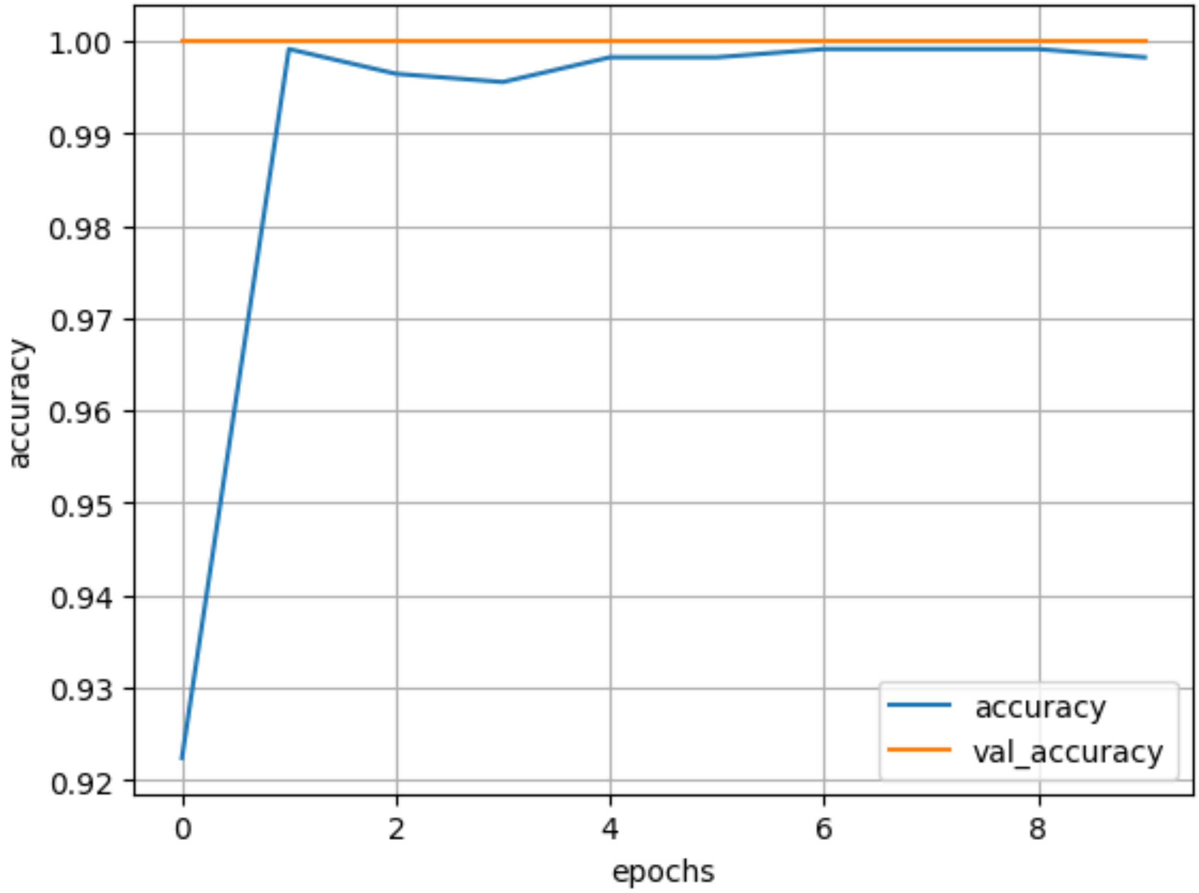
```
from tensorflow.keras import layers
#classifier

x = keras.layers.GlobalAveragePooling2D()(base_model3.output)
x = keras.layers.Dropout(0.15)(x)
output_ = layers.Dense(3, activation='softmax')(x)

model3 = keras.models.Model(inputs=[base_model3.input], outputs=[output_])
```

Compiling and training the model

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np

model3.compile(loss='sparse_categorical_crossentropy',
               optimizer='adam',
               metrics=['accuracy'])

callbacks3 = [keras.callbacks.ModelCheckpoint("best_InceptionResNetV2_TL.weights.
                                              monitor='val_accuracy',
                                              save_weights_only=True,
                                              save_best_only=True)]

datagen = ImageDataGenerator(
    rotation_range=20,  #randomly rotate images by up to 20 degrees
    width_shift_range=0.2,  #randomly shift images horizontally by up to 20%
    height_shift_range=0.2,  #randomly shift images vertically by up to 20%
    horizontal_flip=True,  #randomly flip images horizontally
    fill_mode='nearest',  #strategy for filling in newly created pixels
    validation_split=0.1 #10% validation split defined here
)

#Create train generator
train_generator = datagen.flow(
    X_train_std,
    y_train,
    batch_size=32,
    subset='training'    #Specify this is for training
)

#Create validation generator
validation_generator = datagen.flow(
    X_train_std,
    y_train,
    batch_size=32,
    subset='validation'  #Specify this is for validation
)

#Train the model
history = model3.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=10,
    callbacks=callbacks3
)
```

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_da
  self._warn_if_super_not_called()
36/36 ━━━━━━━━━━━━━━━━━━━━ 88s 1s/step - accuracy: 0.8216 - loss: 0.4465 - va
Epoch 2/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 24s 561ms/step - accuracy: 0.9997 - loss: 0.0260 -
Epoch 3/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 24s 569ms/step - accuracy: 0.9933 - loss: 0.0311 -
Epoch 4/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 24s 563ms/step - accuracy: 0.9952 - loss: 0.0134 -
Epoch 5/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 24s 547ms/step - accuracy: 0.9973 - loss: 0.0135 -
Epoch 6/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 24s 555ms/step - accuracy: 0.9994 - loss: 0.0084 -
Epoch 7/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 24s 555ms/step - accuracy: 0.9991 - loss: 0.0063 -
Epoch 8/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 24s 551ms/step - accuracy: 0.9997 - loss: 0.0046 -
Epoch 9/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 24s 566ms/step - accuracy: 0.9984 - loss: 0.0058 -
Epoch 10/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 24s 544ms/step - accuracy: 0.9985 - loss: 0.0071 -
```

```
#visualize accuracy

keys = ['accuracy', 'val_accuracy']
progress = {k:v for k,v in history.history.items() if k in keys}

import pandas as pd
pd.DataFrame(progress).plot()

plt.xlabel("epochs")
plt.ylabel("accuracy")

plt.grid(True)
plt.show()
```



Update model with best weight

```
model3.load_weights("best_InceptionResNetV2_TL.weights.h5")

testLoss3, testAccuracy3 = model3.evaluate(x = X_test_std, y = y_test)

print("Test-loss: %f, Test-accuracy: %f" % (testLoss3, testAccuracy3))
```

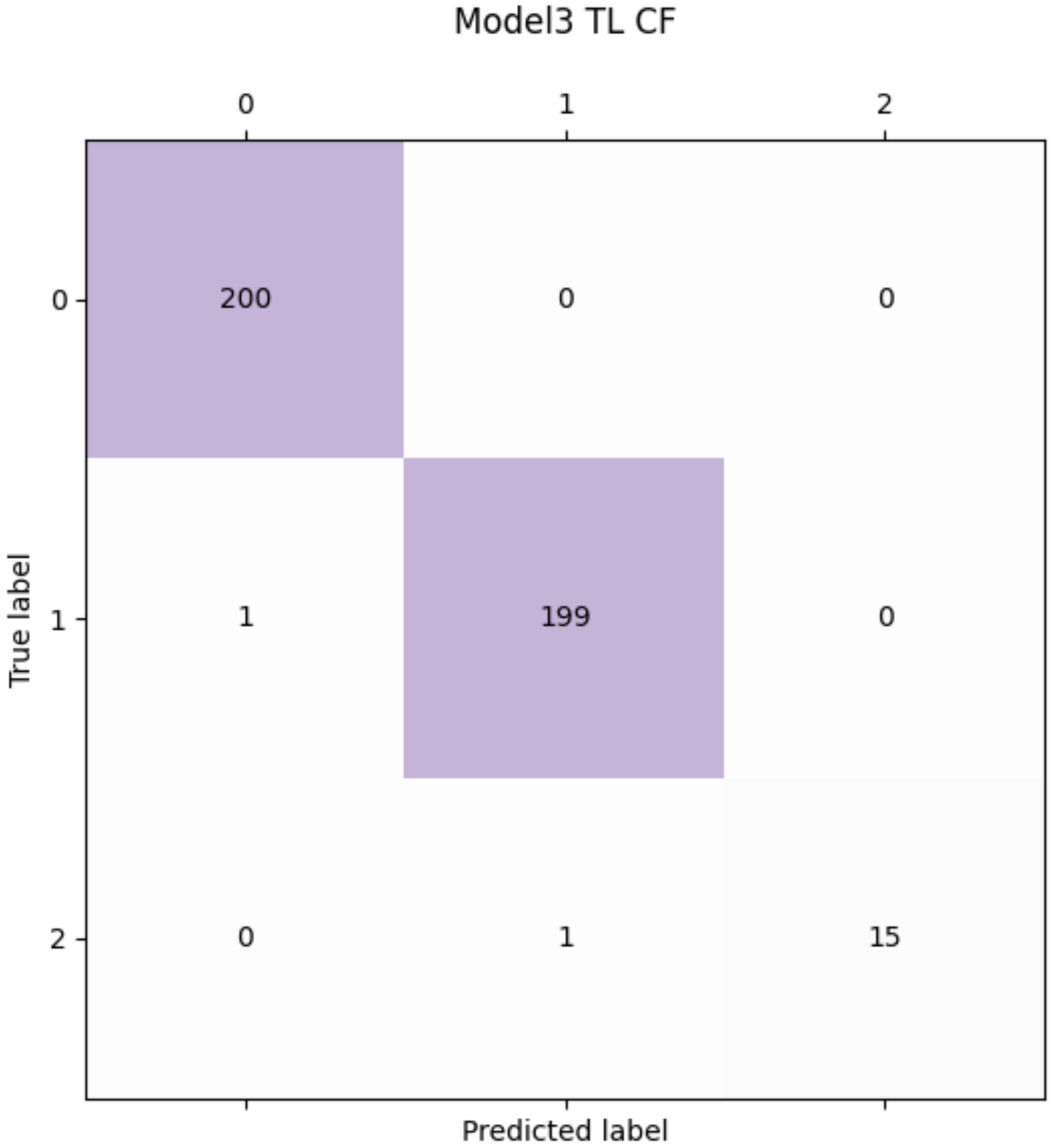
```
13/13 ━━━━━━━━━━━━━━━━━━━━ 4s 264ms/step - accuracy: 0.9993 - loss: 0.0301
Test-loss: 0.043365, Test-accuracy: 0.995192
```

Performance

```
y_prob = model3.predict(X_test_std)
y_predict = np.argmax(y_prob, axis=-1)
print(y_predict)
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 12s 254ms/step
[1 1 0 0 0 0 0 1 0 0 0 1 0 0 1 1 1 1 0 1 1 1 1 1 1 0 0 0 1 1 0 1 0 0 0 0 1
 0 0 0 1 0 1 0 0 1 1 0 0 1 1 1 1 1 0 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 0 0 1 0
 0 1 1 0 1 0 1 0 2 1 0 1 0 0 0 0 1 1 1 0 0 2 0 0 1 0 0 0 0 0 1 1 1 1 1 0 0
 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 0 2 0 1 1 1 0 1 0 0 1 1 0 1 1 1 0 1 1 1
 1 0 1 0 1 1 1 1 0 2 1 0 0 1 0 1 1 0 0 0 1 0 0 1 1 1 0 1 0 0 0 0 0 0 0 1 1
 1 0 1 0 0 1 1 0 1 1 0 1 0 2 0 0 0 1 1 1 1 0 0 0 1 0 0 0 1 2 0 0 1 0 0 0 1
 0 1 0 0 0 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 0 1 0 1 1 1 0 0 1
 0 1 0 1 0 0 1 1 1 0 0 1 0 0 1 1 1 0 0 0 0 1 0 1 0 1 0 0 1 0 2 2 1 1 1 1 0
 0 2 1 0 0 1 0 1 1 0 0 1 0 1 1 1 0 0 1 1 0 2 1 2 0 0 0 0 1 0 1 0 0 0 1 1 2
 1 0 1 1 1 1 0 0 1 0 1 2 0 2 0 1 1 0 0 0 0 1 1 0 0 0 1 0 1 1 0 1 1 1 1 1 1
 0 0 1 1 0 0 1 0 1 1 1 0 0 0 0 0 1 1 0 1 1 0 2 0 1 1 1 1 1 1 0 1 0 1 1 0 1
 0 1 0 1 1 0 1 0 0]
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_true = y_test, y_pred = y_predict)

fig, ax = plt.subplots(figsize=(6, 6))
ax.matshow(cm, cmap=plt.cm.Purples, alpha=0.3)

for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(x=j, y=i, s=cm[i, j], va='center', ha='center')
ax.title.set_text('Model3 TL CF\n')
plt.xlabel('Predicted label')
plt.ylabel('True label')

plt.tight_layout()
plt.savefig("ConfusionMatrix.png", dpi=300, format='png', pad_inches=0.3)
plt.show()

print(set(y))
print(lEncoder.classes_)
```



```
{0, 1, 2}
['Motorbikes' 'airplanes' 'schooner']
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

pScore = precision_score(y_true= y_test, y_pred = y_predict, average = 'weighted'
print("Precision: ", pScore)

rScore = recall_score(y_true= y_test, y_pred = y_predict, average = 'weighted')
print("Recall: ", rScore)

fScore = f1_score(y_true= y_test, y_pred = y_predict, average = 'weighted')
print("F1-score: ", fScore)

print("\n\n\n")
```

```
Precision:  0.9952042671259089
Recall:  0.9951923076923077
F1-score:  0.9951565332945551
```

Save the model and the dataset

```
model3.save('/content/gdrive/MyDrive/DL/InceptionResNetV2_Best_Model_TL.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `
```

```
from numpy import save

save('/content/gdrive/MyDrive/DL/X_train_std_model3.npy', X_train_std)
save('/content/gdrive/MyDrive/DL/X_test_std_model3.npy', X_test_std)

save('/content/gdrive/MyDrive/DL/y_train_model3.npy', y_train)
save('/content/gdrive/MyDrive/DL/y_test_model3.npy', y_test)
```