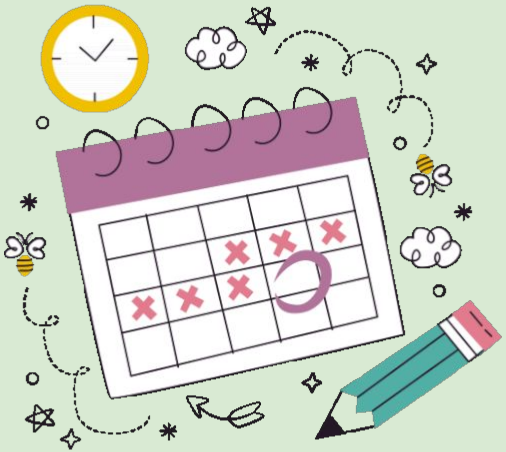
DATA SCIENCE
-
PROJECT

VEHICLE
IMAGE RECOGNITION

PROCEDURE:

- ➔ small dataset images vehicles
- ➔ preparing the data set
- ➔ plots Jupyter Notebook
- ➔ functional models
- ➔ sequential model CNN



DATA SET:

PRELIMINARY CONSIDERATIONS:

➔ Image recognition is an important area that enables image analysis and processing. Deep learning algorithms that can recognize complex patterns and structures in images play a central role here.

➔ These algorithms use artificial neural networks that are based on the structure of the human brain and can independently recognize patterns and connections in large amounts of data.

AIM:

➔ This project is about predicting images using neural models from the Tensorflow framework with sequential and functional models.

FUNCTIONAL MODEL:

- ## PREPARING DATASET:

```
from pathlib import Path
import os.path
from PIL import Image
import tensorflow as tf

Create a list with the filepaths for training and testing:


dloadpaths = Path('./data_vehicle_recognition/vehicles')

imagepaths = list(dloadpaths.rglob('**/*.jpg'))

len(imagepaths)      | --> 522 images
```

- **creating a data frame:**

```python
def images(imagepath):

    """ Create a DataFrame with the imagepath and the labels of the pictures
    """

    labels = [str(imagepath[i]).split("\\")[-2] for i in range(len(imagepath))]

    # samples (pd.Series)
    imagepath = pd.Series(imagepath, name='Imagepath').astype(str)
    labels = pd.Series(labels, name='Label')

    # Concatenate imagepath and labels
    df01 = pd.concat([imagepath, labels], axis=1)

    # Shuffle the DataFrame and reset index
    df01 = df01.sample(frac=1,random_state=42).reset_index(drop = True)

    return df01
```

- **details about the data frame:**

```
Dataframe with imagepaths:

df01 = images(imagepaths)

print(f'Number of pictures: {df01.shape[0]}\n')
print(f'Number of different labels: {len(df01.Label.unique())}\n')
print(f'Labels: {df01.Label.unique()}')
```

```
Number of pictures: 522

Number of different labels: 9

Labels: ['scooty' 'bike' 'car' 'boat' 'helicopter' 'truck' 'bus' 'plane' 'cycle']
```

	Imagepath	Label
0	data_vehicle_recognition\vehicles\scooty\images (15).jpg	scooty
1	data_vehicle_recognition\vehicles\scooty\images (20).jpg	scooty
2	data_vehicle_recognition\vehicles\bike\2Q__ (6).jpg	bike
3	data_vehicle_recognition\vehicles\car\images (10).jpg	car
4	data_vehicle_recognition\vehicles\boat\images (12).jpg	boat

- for a better training data set --> all categories with the same size:

```
list_indizes = []

for i in df01.Label.unique():
    if len(df01[df01.Label==f'{i}']) > 52:
        [list_indizes.append(i) for i in df01[df01.Label == f'{i}'].iloc[:(len(df01[df01.Label==f'{i}']) - 52),:].
         index]
```

original size

```
df01.Label.value_counts()
```

```
Label
car            65
cycle          65
scooty         63
helicopter     57
bike           55
boat           55
truck          55
bus            55
plane          52
```

resized

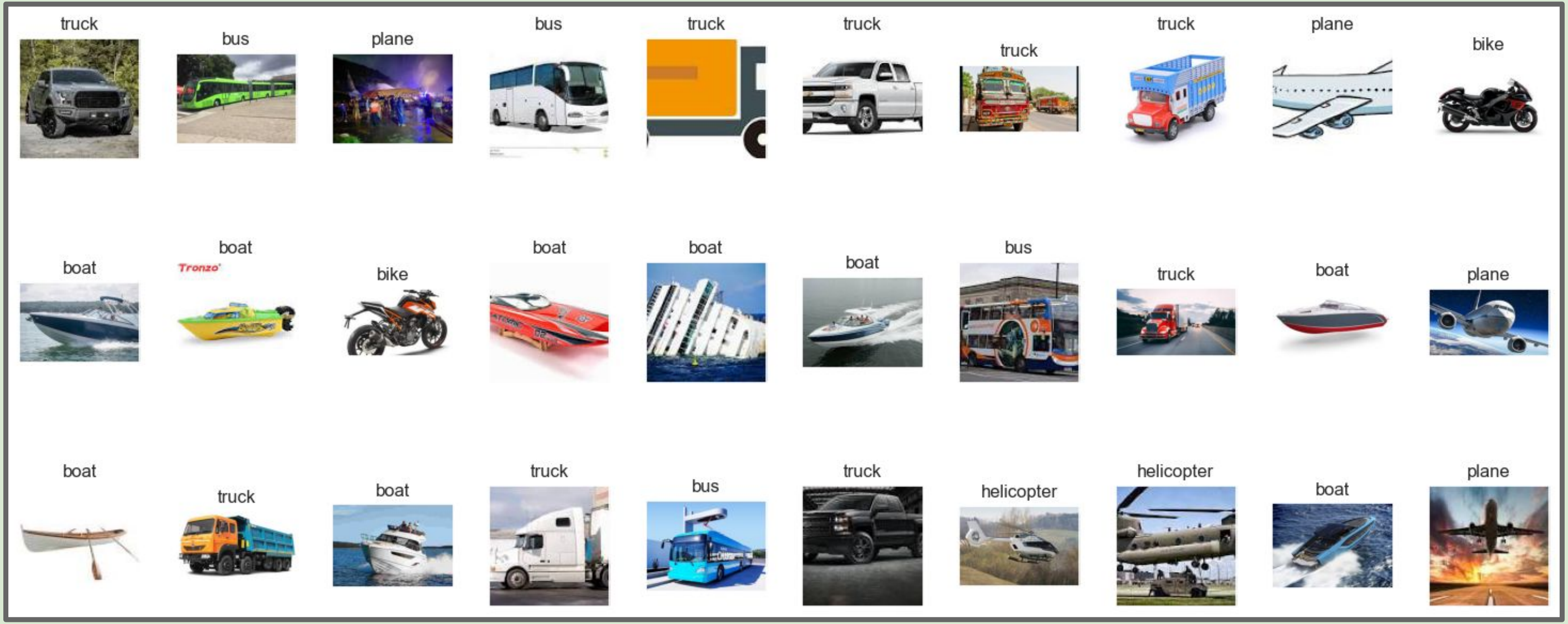
```
df01.drop(index=list_indizes,inplace=True)

df01.Label.value_counts()
```

```
Label
truck          52
bus            52
plane          52
bike           52
boat           52
helicopter     52
car            52
scooty         52
cycle          52
```

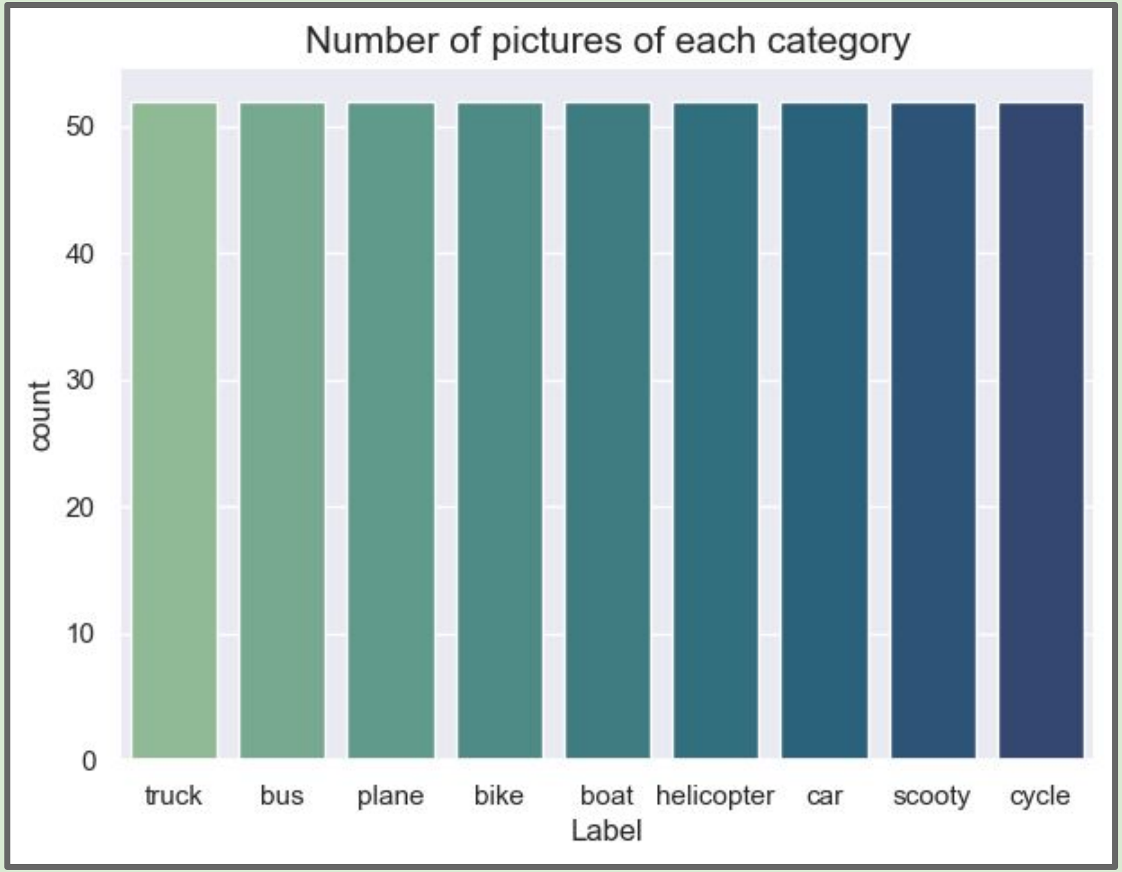
➢ PLOTS:

- **Shows some images from the dataset:**

```
fig, axes = plt.subplots(nrows=3, ncols=10, figsize=(15, 7),subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(df01.Imagepath[i]))
    ax.set_title(df01.Label[i], fontsize = 12)

# distance between the pictures
plt.tight_layout(pad=2)
plt.show()
```


truck
bus
plane
bus
truck
truck
truck
truck
plane
bike
boat
boat
Tronzo
bike
boat
boat
boat
bus
truck
boat
plane
boat
truck
boat
truck
bus
truck
helicopter
helicopter
boat
plane

- **each category with 52 samples**



- **preparing images:**

```
TRAIN TEST SPLIT:

from sklearn.model_selection import train_test_split

train_df01, test_df01 = train_test_split(df01, test_size=0.2, random_state=42)
```

```
IMAGE LOADING AND DATA AUGMENTATION

# initialize generators - validation_split(testsize)
train_gen = tf.keras.preprocessing.image.ImageDataGenerator(
            preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input, validation_split=0.2)

test_gen = tf.keras.preprocessing.image.ImageDataGenerator(
            preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input)
```

- **loading images with subset split to X-Train and X-Test (80 % / 20 %)**

```
def create_gen():

# X-Train
    train_img = train_gen.flow_from_dataframe(dataframe=train_df01, x_col="Imagepath", y_col="Label",
            target_size=(224, 224), color_mode='rgb', class_mode='categorical', batch_size=32, shuffle=True,
            seed=0, subset='training', rotation_range=30, zoom_range=0.15, width_shift_range=0.2,
            height_shift_range=0.2, shear_range=0.15, horizontal_flip=True, fill_mode="nearest")

# y-Train
    val_img = train_gen.flow_from_dataframe(dataframe=train_df01, x_col="Imagepath", y_col="Label",
          target_size=(224, 224), color_mode='rgb', class_mode='categorical', batch_size=32, shuffle=True,
          seed=0, subset='validation', rotation_range=30, zoom_range=0.15, width_shift_range=0.2,
          height_shift_range=0.2, shear_range=0.15, horizontal_flip=True, fill_mode="nearest")

# X-Test
    test_img = test_gen.flow_from_dataframe(dataframe=test_df01, x_col="Imagepath", y_col="Label",
           target_size=(224, 224), color_mode='rgb', class_mode='categorical', batch_size=32, shuffle=False)

    return train_gen, test_gen, train_img, val_img, test_img
```

- **creating 10 functional models:**

```python
def get_model(model):

    # Loading model:

    kwargs = {'input_shape':(224, 224, 3),'include_top':False,'weights':'imagenet','pooling':'avg'}

    pretrained_model = model(**kwargs)
    pretrained_model.trainable = False

    inputs = pretrained_model.input

    x = tf.keras.layers.Dense(128, activation='relu')(pretrained_model.output)
    x = tf.keras.layers.Dense(128, activation='relu')(x)

    outputs = tf.keras.layers.Dense(9, activation='softmax')(x)

    model = tf.keras.Model(inputs=inputs, outputs=outputs)

    model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])

    return model
```

```python
Dictionary with the models

models = {"ResNet101V2": {"model":tf.keras.applications.ResNet101V2, "perf":0},
          "Xception": {"model":tf.keras.applications.Xception, "perf":0},
          "InceptionResNetV2": {"model":tf.keras.applications.InceptionResNetV2, "perf":0},
          "DenseNet169": {"model":tf.keras.applications.DenseNet169, "perf":0},
          "DenseNet201": {"model":tf.keras.applications.DenseNet201, "perf":0},
          "NASNetMobile": {"model":tf.keras.applications.NASNetMobile, "perf":0},
          "MobileNet": {"model":tf.keras.applications.MobileNet, "perf":0},
          "MobileNetV2": {"model":tf.keras.applications.MobileNetV2, "perf":0},
          "ResNet152V2": {"model":tf.keras.applications.ResNet152V2, "perf":0},
          "VGG16": {"model":tf.keras.applications.VGG16, "perf":0}}
```

- **creating the generators and training 10 models with 3 epochs:**

```python
from time import perf_counter

train_gen,test_gen,train_img,val_img,test_img = create_gen()
```

```python
# Fit the models
for name, model in models.items():

    # Get the model
    m = get_model(model['model'])
    models[name]['model'] = m

    start = perf_counter()

    # Fit the model
    history = m.fit(train_img,validation_data=val_img,epochs=3,verbose=0)

    # Save the duration and the val_accuracy
    duration = perf_counter() - start
    duration = round(duration,2)
    models[name]['perf'] = duration
    print(f"{name:20} trained in {duration} sec")

    val_acc = history.history['val_accuracy']
    models[name]['val_acc'] = [round(v,4) for v in val_acc]
```

```
Found 300 validated image filenames belonging to 9 classes.
Found 74 validated image filenames belonging to 9 classes.
Found 94 validated image filenames belonging to 9 classes.
```

- **predicting the 10 models:**

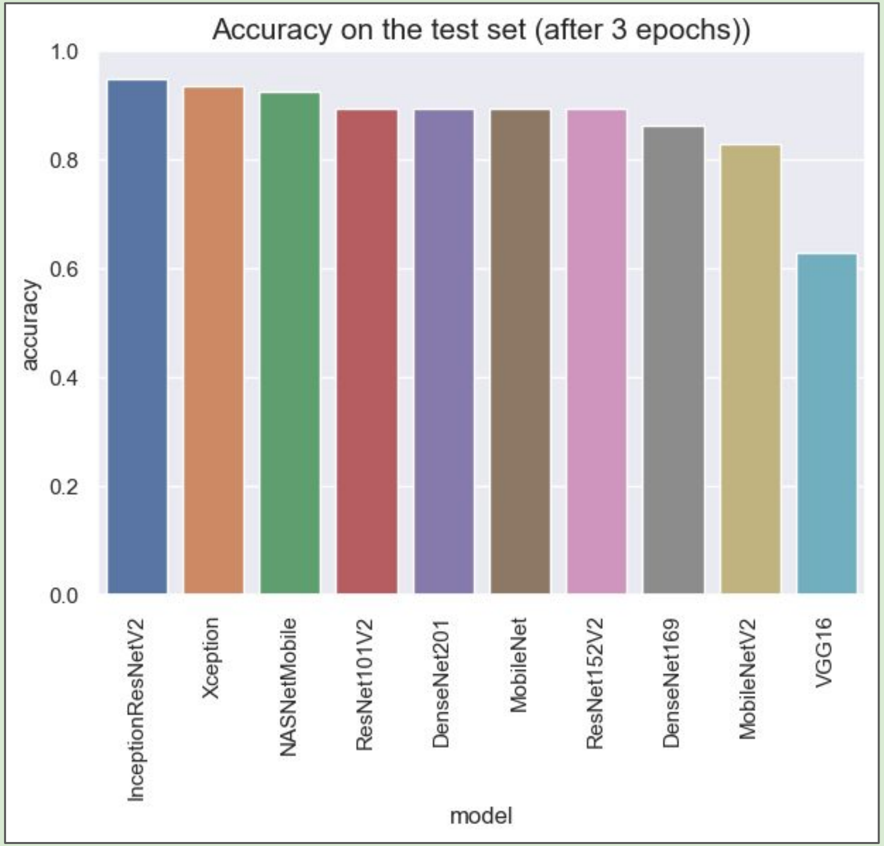
```
for name, model in models.items():

    # Predict the label of the test_images
    pred = models[name]['model'].predict(test_img)
    pred = np.argmax(pred,axis=1)

    # Map the label
    labels = (train_img.class_indices)
    labels = dict((v,k) for k,v in labels.items())
    pred = [labels[k] for k in pred]

    y_test = list(test_df01.Label)
    acc = accuracy_score(y_test,pred)
    models[name]['acc'] = round(acc,4)
```

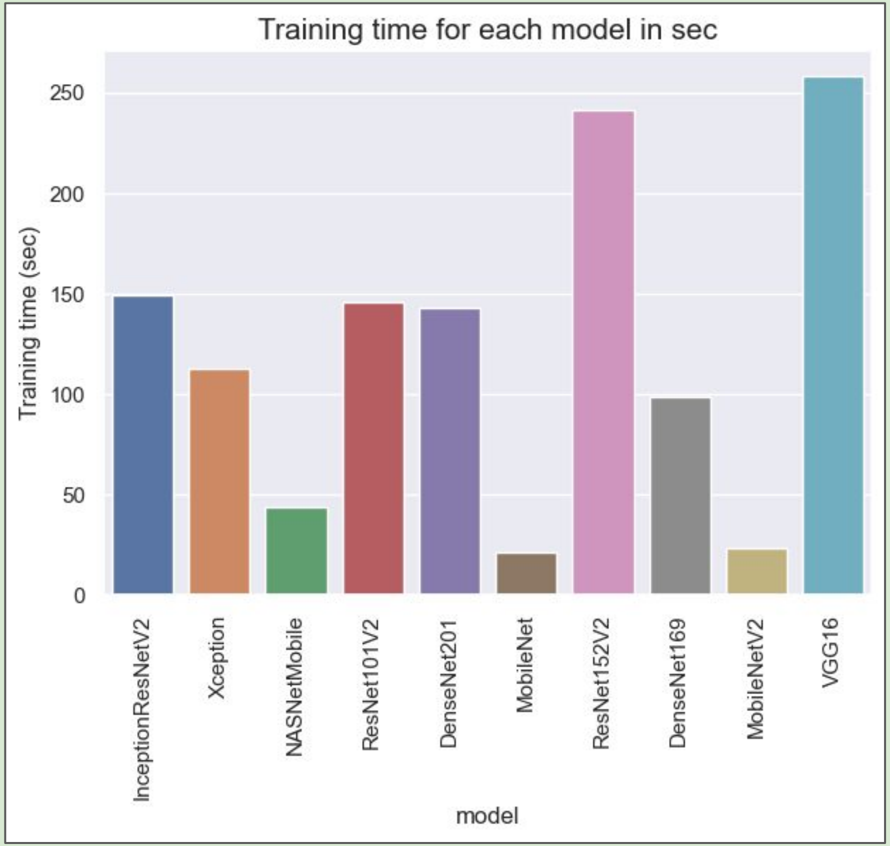
- **data frame with the results:**

```
models_result = []

for name, v in models.items():
    models_result.append([ name, models[name]['val_acc'][-1], models[name]['acc'], models[name]['perf']])

df_results = pd.DataFrame(models_result, columns = ['model','val_accuracy','accuracy','Training time (sec)'])
df_results.sort_values(by='accuracy', ascending=False, inplace=True)
df_results.reset_index(inplace=True,drop=True)
df_results
```

	model	val_accuracy	accuracy	Training time (sec)
0	InceptionResNetV2	0.9189	0.9468	148.90
1	Xception	0.9189	0.9362	112.26
2	NASNetMobile	0.9459	0.9255	43.55
3	ResNet101V2	0.8919	0.8936	145.62
4	DenseNet201	0.9595	0.8936	142.68
5	MobileNet	0.8919	0.8936	21.23
6	ResNet152V2	0.9054	0.8936	241.04
7	DenseNet169	0.8514	0.8617	98.63
8	MobileNetV2	0.8649	0.8298	22.99
9	VGG16	0.6486	0.6277	257.95

- **accuracy after 3 epochs and training time:**



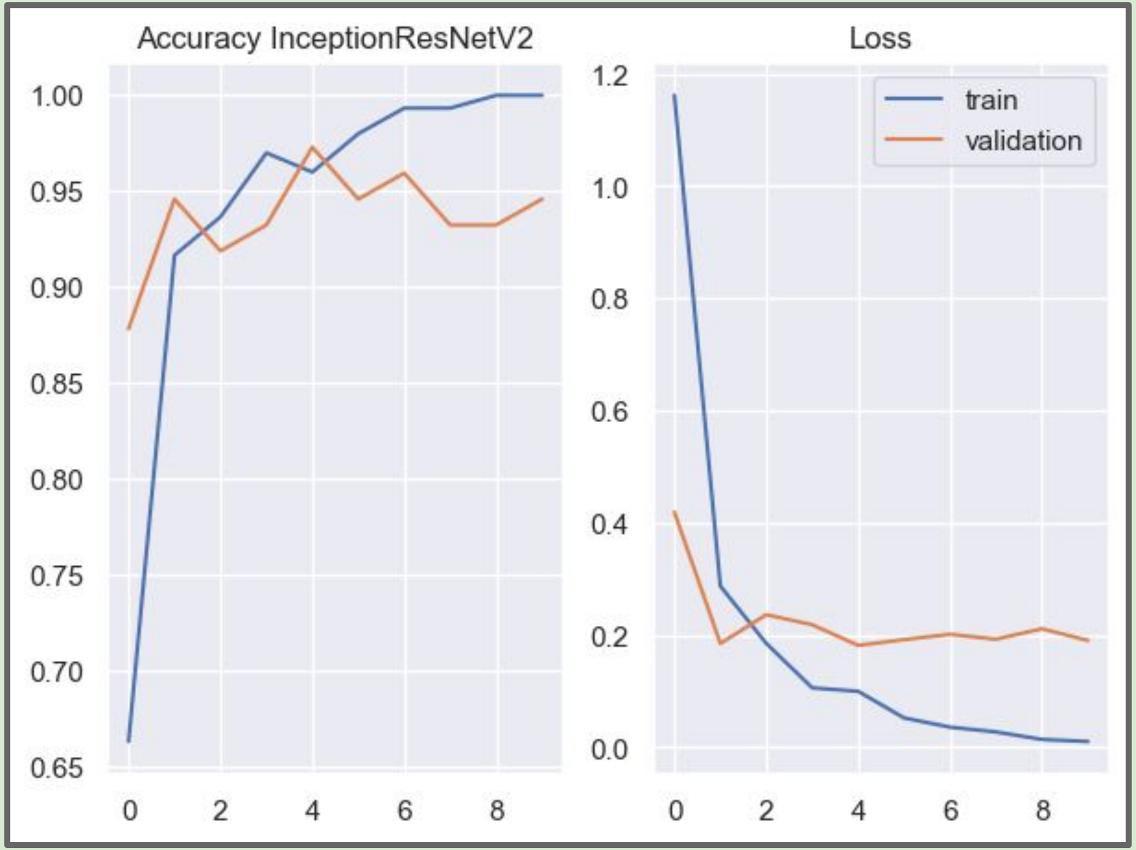


- **training the best 3 models with 10 epochs:**

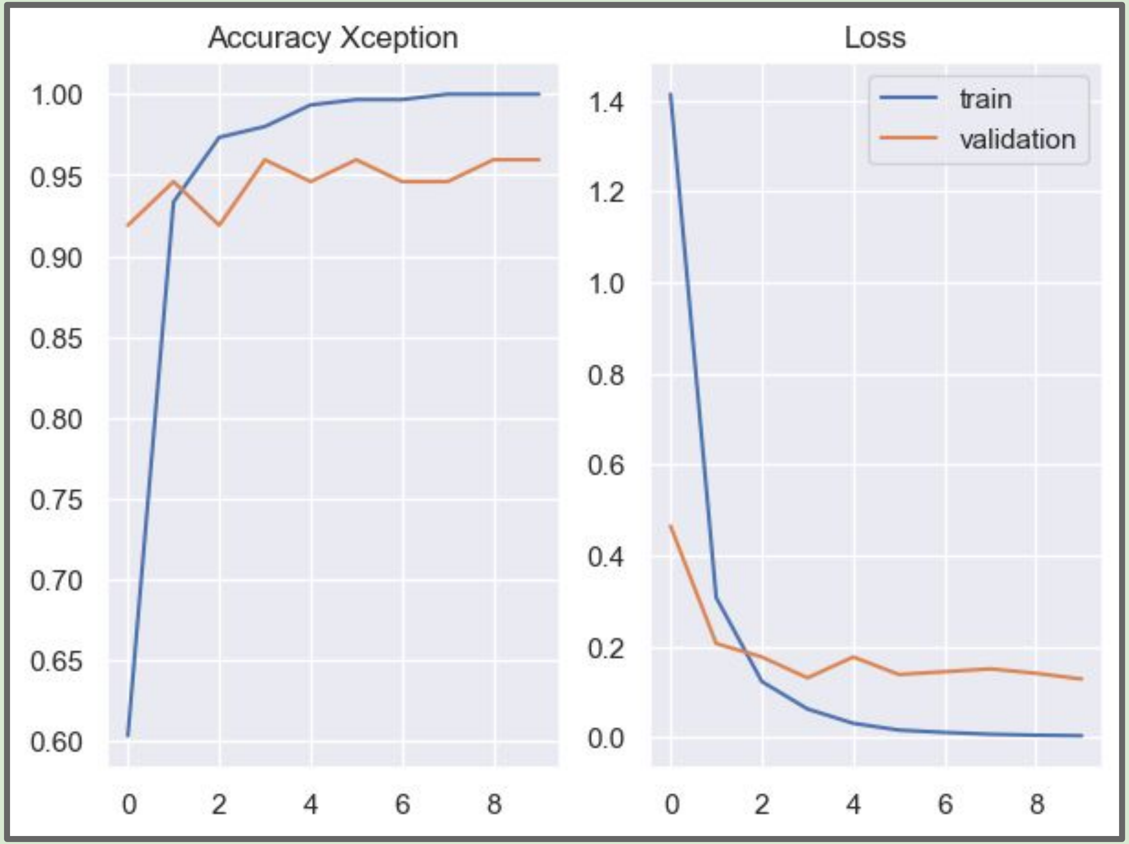
```
MODELS:

model01 = get_model(tf.keras.applications.InceptionResNetV2)
model02 = get_model(tf.keras.applications.Xception)
model03 = get_model(tf.keras.applications.NASNetMobile)
```

```
TRAINING:

history01 = model01.fit(train_img,validation_data=val_img,epochs=10)
history02 = model02.fit(train_img,validation_data=val_img,epochs=10)
history03 = model03.fit(train_img,validation_data=val_img,epochs=10)
```

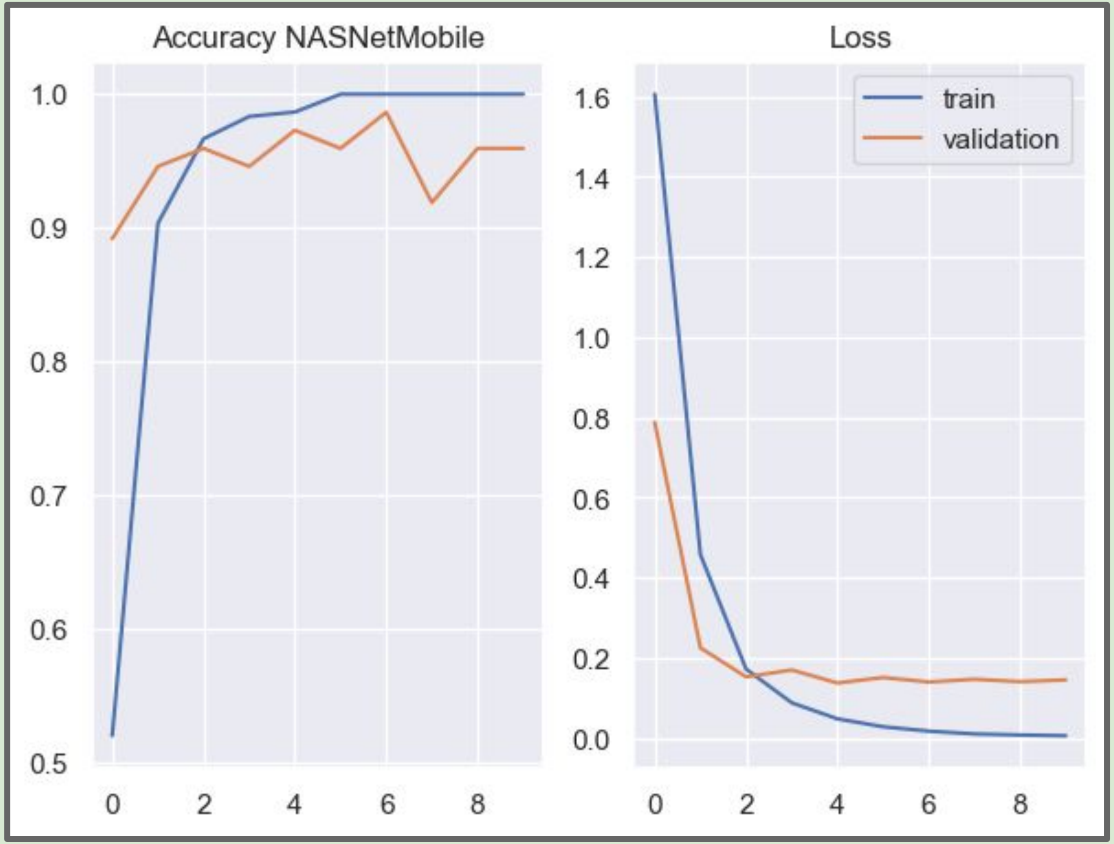
- **visualization accuracy and loss model “InceptionResNetV2”:**



- **<u>visualization accuracy and loss model “Xception”:</u>**



- **visualization accuracy and loss model “NASNetMobile”:**



- **predicting the labels of the test_images:**

```
InceptionResNetV2:

pred01 = model01.predict(test_img)
pred01 = np.argmax(pred01,axis=1)

# Map the label
labels = (train_img.class_indices)
labels = dict((v,k) for k,v in labels.items())
pred01 = [labels[k] for k in pred01]
```

```
Xception:

pred02 = model02.predict(test_img)
pred02 = np.argmax(pred02,axis=1)

# Map the label
labels = (train_img.class_indices)
labels = dict((v,k) for k,v in labels.items())
pred02 = [labels[k] for k in pred02]
```

```
NASNetMobile:

pred03 = model03.predict(test_img)
pred03 = np.argmax(pred03,axis=1)

# Map the label
labels = (train_img.class_indices)
labels = dict((v,k) for k,v in labels.items())
pred03 = [labels[k] for k in pred03]
```

- **accuracies:**

```
Accuracy InceptionResNetV2:

y_test = list(test_df01.Label)
acc = accuracy_score(y_test,pred01)
print(f'--> Accuracy on the test set: {acc * 100:.2f}%')
```

```
--> Accuracy on the test set: 98.94%
```

```
Accuracy Xception:

y_test = list(test_df01.Label)
acc = accuracy_score(y_test,pred02)
print(f'--> Accuracy on the test set: {acc * 100:.2f}%')
```

```
--> Accuracy on the test set: 95.74%
```

```
Accuracy NASNetMobile:

y_test = list(test_df01.Label)
acc = accuracy_score(y_test,pred03)
print(f'--> Accuracy on the test set: {acc * 100:.2f}%')
```

```
--> Accuracy on the test set: 94.68%
```

- **classification reports:**

```
Classification Report InceptionResNetV2:

class_report = classification_report(y_test, pred01, zero_division=1)
print(class_report)
```

	precision	recall	f1-score	support
bike	1.00	1.00	1.00	13
boat	1.00	1.00	1.00	13
bus	1.00	1.00	1.00	11
car	1.00	1.00	1.00	8
cycle	1.00	1.00	1.00	10
helicopter	1.00	0.91	0.95	11
plane	0.93	1.00	0.97	14
scooty	1.00	1.00	1.00	9
truck	1.00	1.00	1.00	5
accuracy			0.99	94
macro avg	0.99	0.99	0.99	94
weighted avg	0.99	0.99	0.99	94

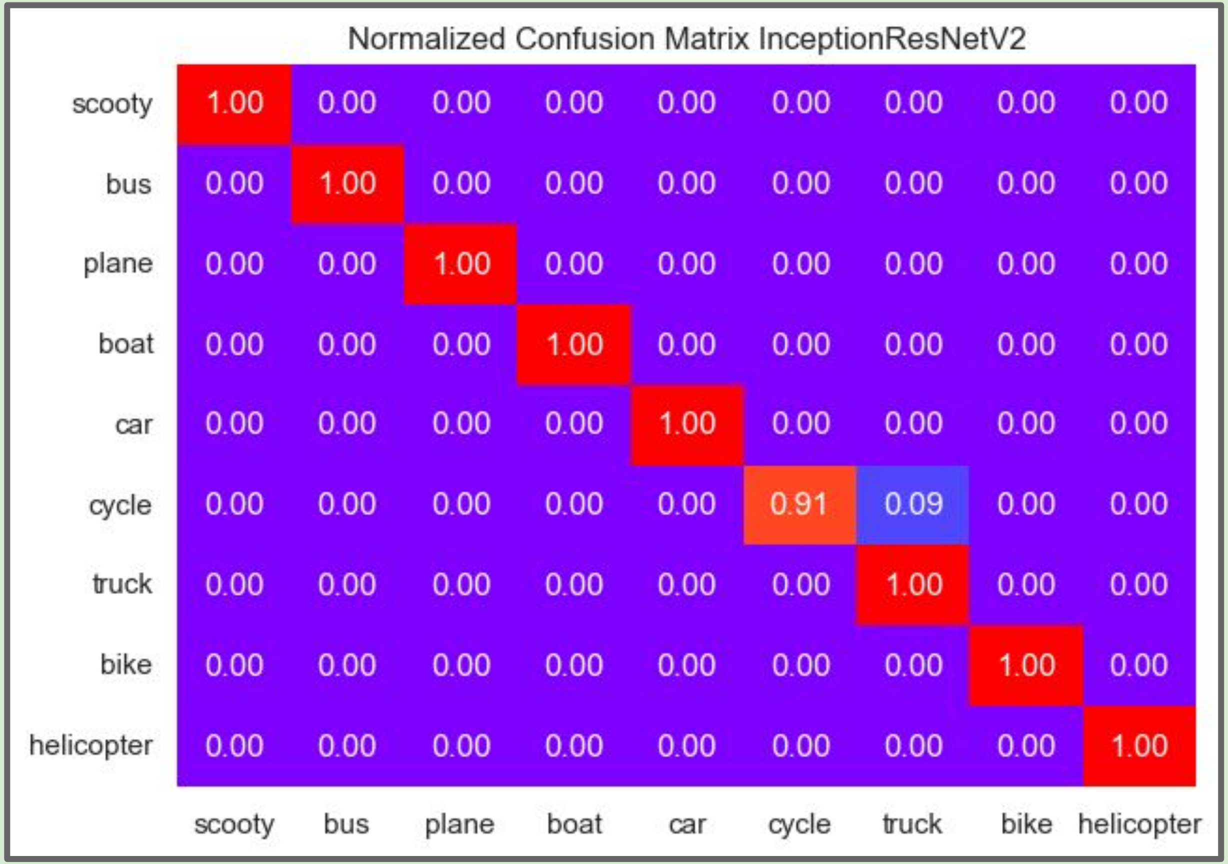
```
Classification Report Xception:

class_report = classification_report(y_test, pred02, zero_division=1)
print(class_report)
```

```
              precision    recall  f1-score   support

        bike       1.00      0.85      0.92        13
        boat       1.00      1.00      1.00        13
         bus       0.92      1.00      0.96        11
         car       1.00      1.00      1.00         8
       cycle       1.00      1.00      1.00        10
  helicopter       1.00      0.91      0.95        11
       plane       0.93      1.00      0.97        14
      scooty       0.82      1.00      0.90         9
       truck       1.00      0.80      0.89         5

    accuracy                           0.96        94
   macro avg       0.96      0.95      0.95        94
weighted avg       0.96      0.96      0.96        94
```

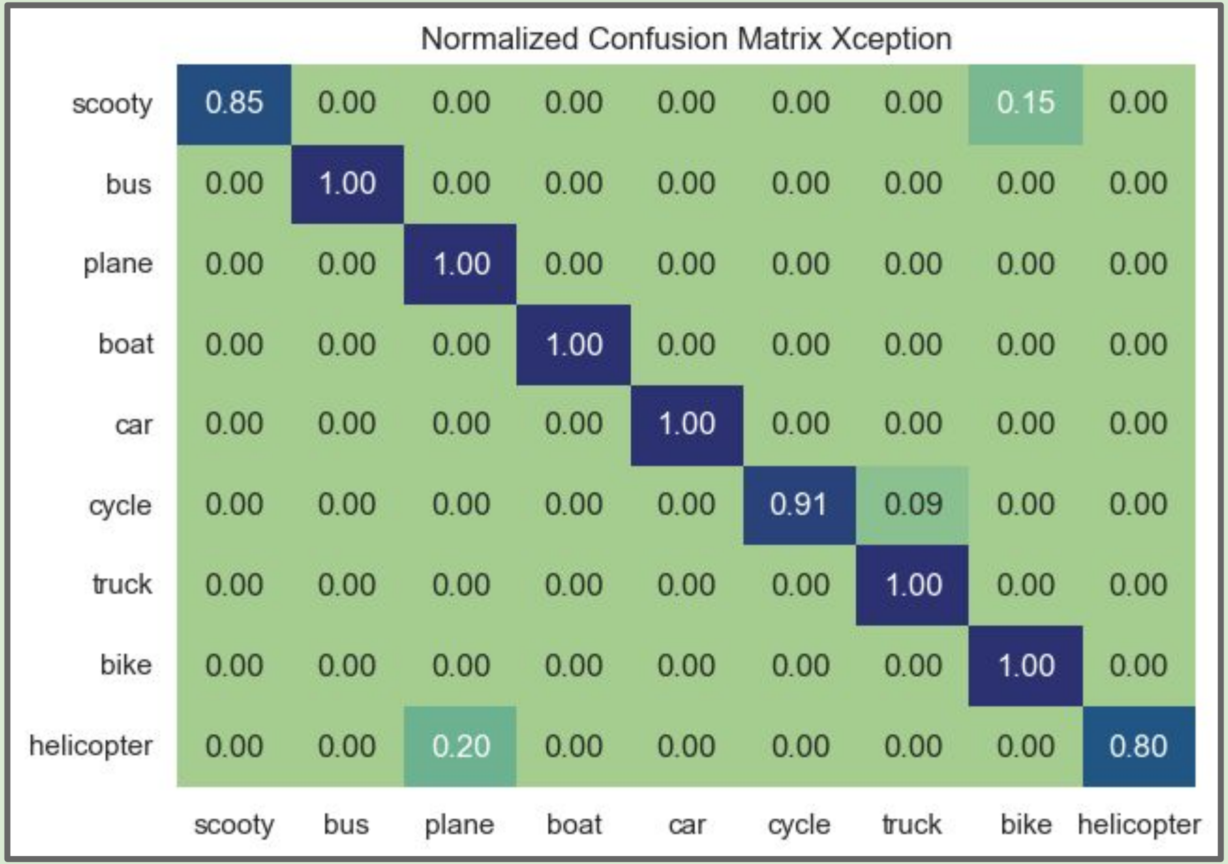
```
Classification Report NASNetMobile:

class_report = classification_report(y_test, pred03, zero_division=1)
print(class_report)
```

```
              precision    recall  f1-score   support

        bike       1.00      1.00      1.00        13
        boat       0.93      1.00      0.96        13
         bus       0.91      0.91      0.91        11
         car       1.00      1.00      1.00         8
       cycle       1.00      1.00      1.00        10
  helicopter       0.83      0.91      0.87        11
       plane       1.00      0.93      0.96        14
      scooty       1.00      0.89      0.94         9
       truck       0.80      0.80      0.80         5

    accuracy                           0.95        94
   macro avg       0.94      0.94      0.94        94
weighted avg       0.95      0.95      0.95        94
```

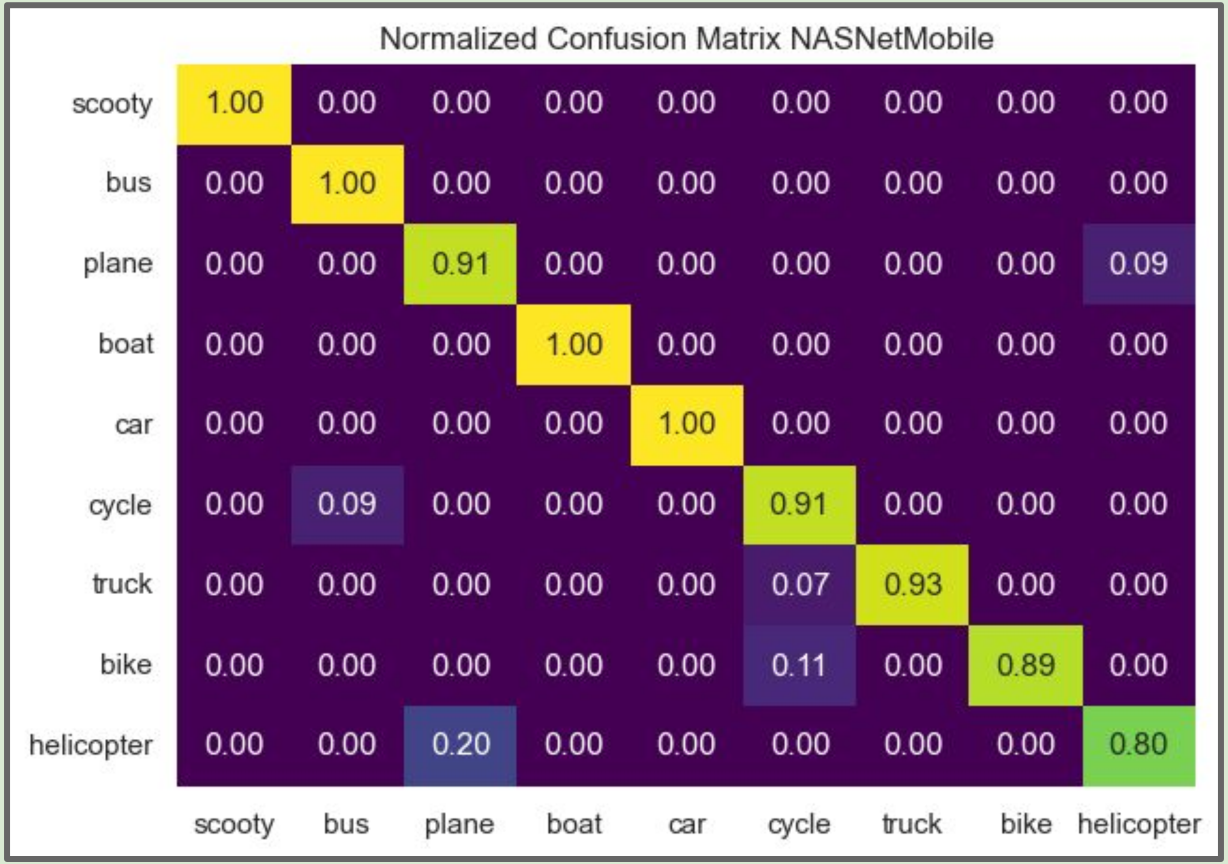
- **<u>confusion matrix “InceptionResNetV2”:</u>**



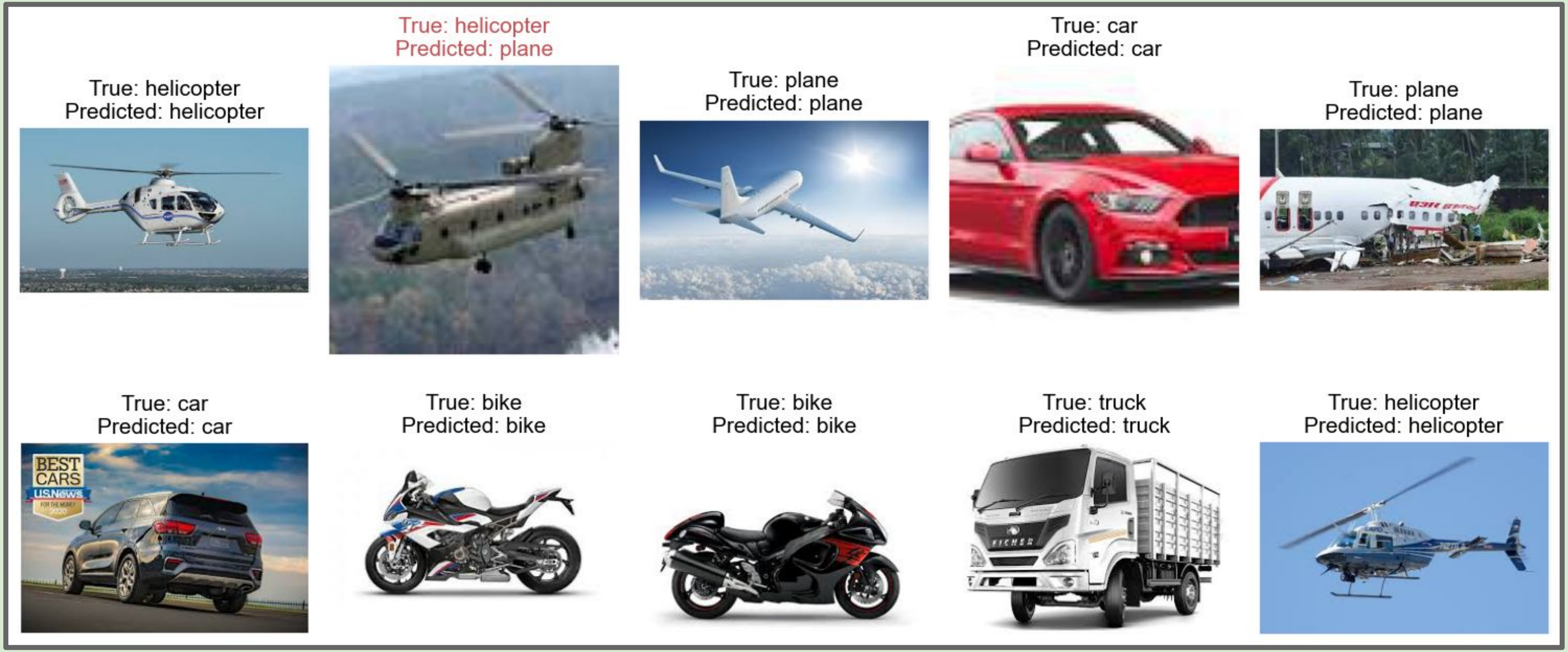
- **confusion matrix “Xception”:**



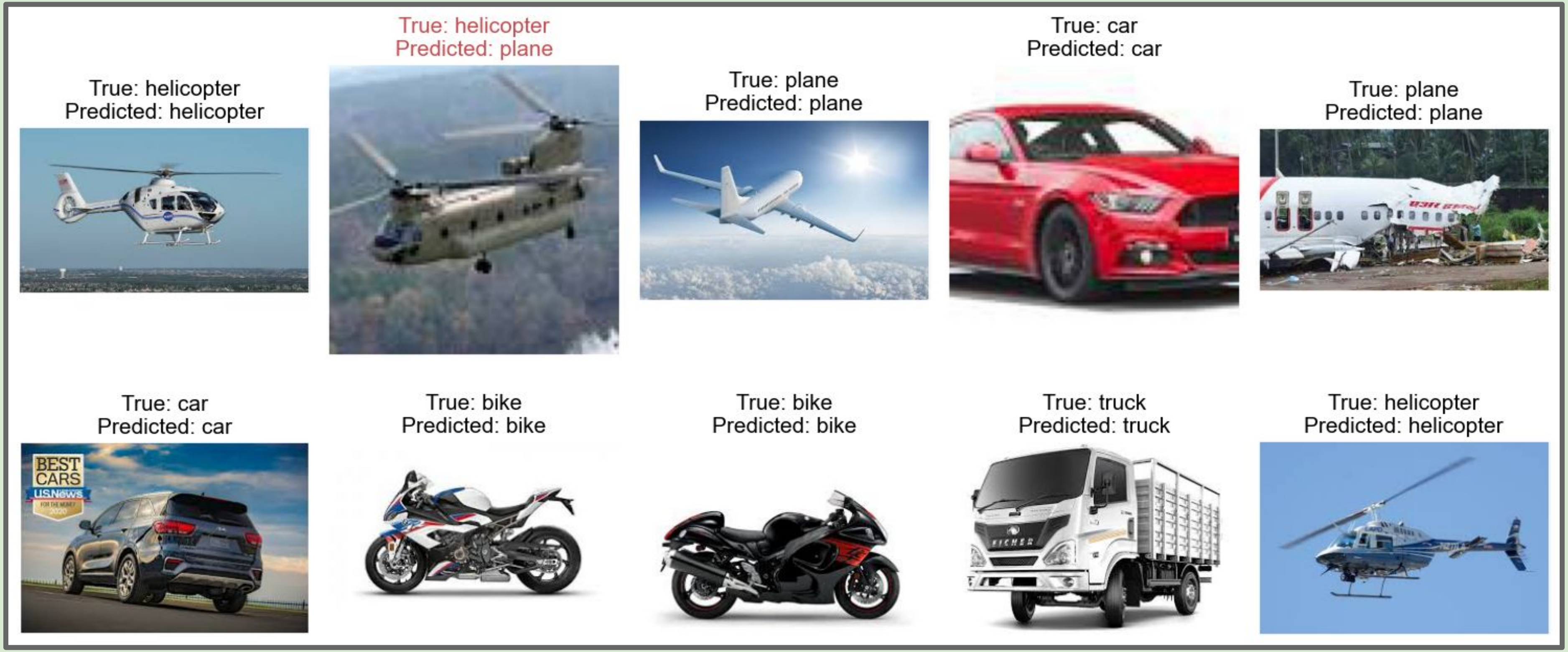
- **confusion matrix “NASNetMobile”:**



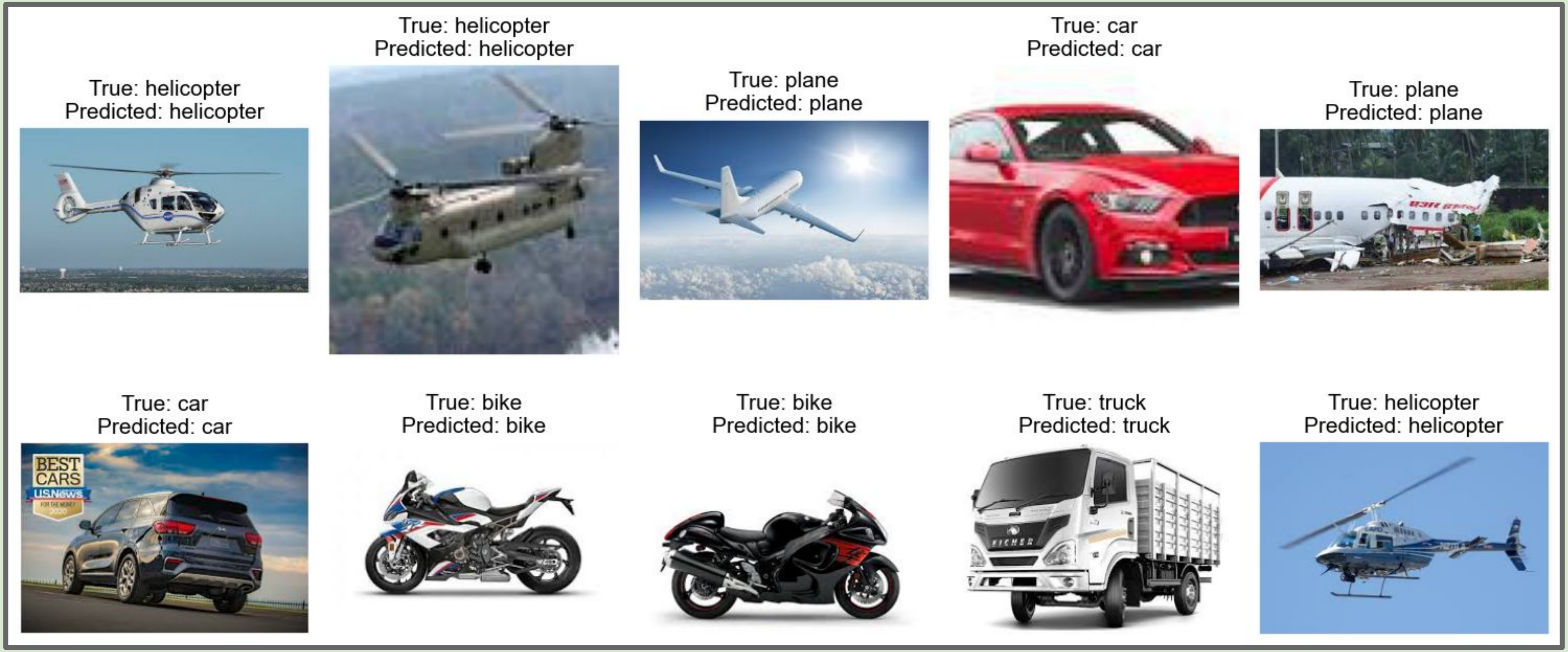
- **some pictures of the dataset with their predicted labels - InceptionResNetV2:**



- **some pictures of the dataset with their predicted labels - Xception:**



- **some pictures of the dataset with their predicted labels - NASNetMobile:**



SEQUENTIAL MODEL:

➢ PREPARING DATASET:

```
PREPARING FILENAMES:

import os

for dirname,_, filenames in os.walk('./data_vehicle_recognition/'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
PREPARING PATH:

root_dir = './data_vehicle_recognition/vehicles/'
```

- **data, labels and labelnames:**

```
PREPARING DATA, LABELS AND LABELNAMES IN LISTS

# Reading images
import cv2

data = []
labels = []
labelnames = []

for label in os.listdir(root_dir):
    path = './data_vehicle_recognition/vehicles/{0}/'.format(label)
    folder_data = os.listdir(path)
    for image_path in folder_data:
        img = cv2.imread(path + image_path)
        img = cv2.resize(img, (32, 32))
        data.append(img)
        labels.append(label)
        if not label in labelnames:
            labelnames.append(label)
```

- **transforming in a numpy array:**

```
TRANSFORMING IN A NUMPY ARRAY:

data = np.array(data)
labels = np.array(labels)
```

```

data.shape, labels.shape
```

```
((526, 32, 32, 3), (526,))
```

```
labelnames
```

```
['bike',
 'boat',
 'bus',
 'car',
 'cycle',
 'helicopter',
 'plane',
 'scooty',
 'truck']
```

- **encoding labels and shuffle data + labels:**

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
labels = le.fit_transform(labels)
```

```
MAKE CATEGORIES:

from tensorflow.keras.utils import to_categorical

labels = to_categorical(labels)
```

```
SHUFFLE DATA AND LABELS:

new = np.arange(526)

np.random.shuffle(new)

data = data[new]
labels = labels[new]
```

```
2  data.shape, labels.shape

((526, 32, 32, 3), (526, 9))
```

• convolutional neural network model:

```
TRAIN TEST SPLIT:

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=33)
```

```
CREATING MODEL:

from tensorflow.keras.layers import Dense, Conv2D, Flatten, MaxPool2D, Dropout, LeakyReLU
from tensorflow.keras.models import Sequential

model = Sequential([
    Conv2D(32, (3, 3), padding="same", activation='relu', input_shape=(32, 32, 3)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPool2D((2, 2)),

    Conv2D(64, (3, 3), padding="same", activation=LeakyReLU(0.001)),
    Conv2D(64, (3, 3), activation=LeakyReLU(0.001)),
    MaxPool2D((2, 2)),

    Dropout(0.25),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(9, activation="softmax")])
```

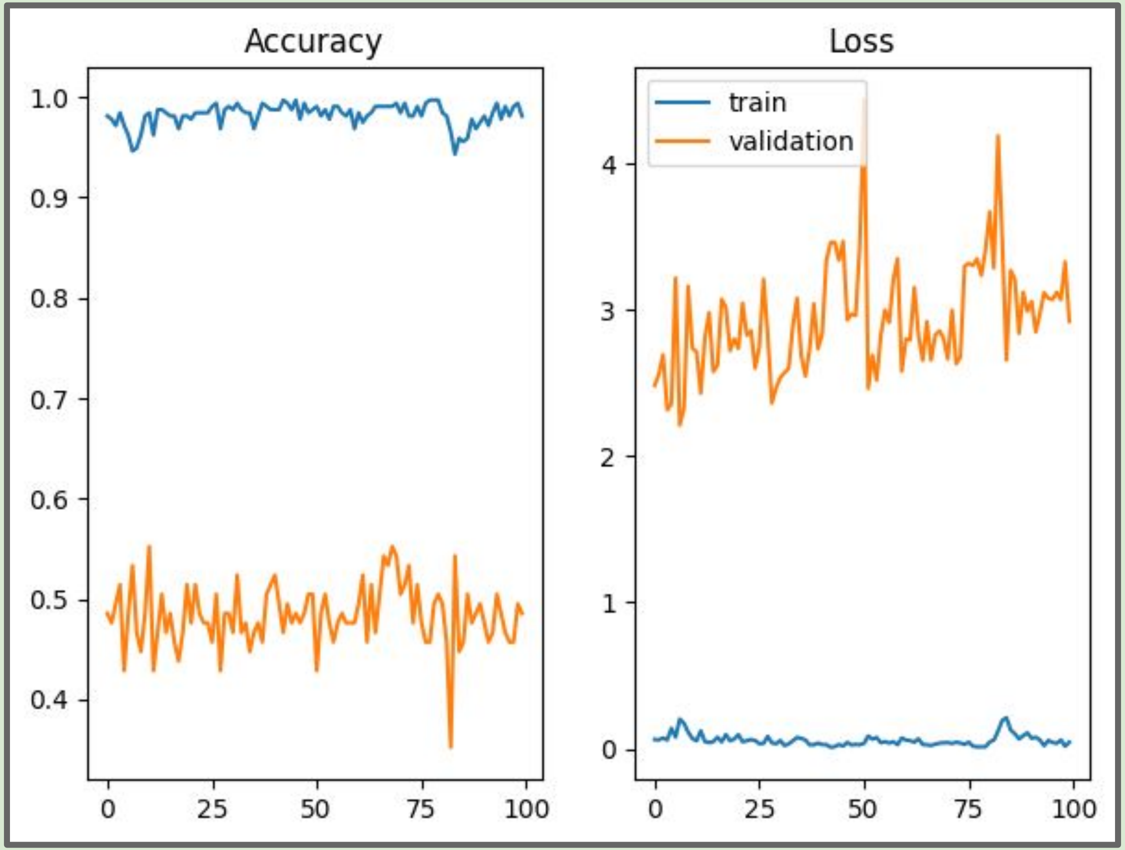
```
MODEL COMPLING:

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
```

```
TRAINING WITH 100 EPOCHS:

history = model.fit(X_train, y_train, epochs=100, validation_split=0.25, batch_size=32)
```

```
ACCURACY AND LOSS IN A DATAFRAME:

history_df = pd.DataFrame(history.history)
history_df.tail(5)
```

	loss	accuracy	val_loss	val_accuracy
95	0.042132	0.990476	3.070535	0.466667
96	0.036421	0.980952	3.119775	0.457143
97	0.059176	0.990476	3.069177	0.457143
98	0.016064	0.993651	3.328260	0.495238
99	0.044331	0.980952	2.922155	0.485714

- **visualization accuracy and loss model “CNN”:**



- **accuracy and prediction:**

```
ACCURACY:

model.evaluate(X_test, y_test)

```

```
4/4 [==============================] - 0s 23ms/step - loss: 2.6932 - accuracy: 0.6226

[2.693232297897339, 0.6226415038108826]
```

```
PREDICTION:

y_pred = model.predict(X_test)
label_classes = y_pred.argmax(axis=-1)
label_classes
```

```
4/4 [==============================] - 0s 20ms/step

array([1, 6, 2, 3, 4, 6, 3, 0, 7, 4, 4, 4, 4, 1, 6, 8, 4, 1, 5, 8, 1, 2,
       7, 1, 1, 8, 8, 7, 5, 4, 5, 7, 3, 4, 0, 1, 2, 5, 8, 5, 5, 7, 0, 7,
       5, 7, 3, 6, 6, 2, 3, 4, 0, 8, 5, 0, 0, 5, 1, 4, 7, 4, 1, 5, 1, 4,
       0, 1, 2, 0, 5, 3, 7, 3, 3, 6, 3, 3, 5, 5, 5, 1, 7, 6, 2, 1, 3, 2,
       2, 2, 0, 1, 8, 8, 5, 4, 8, 4, 3, 7, 4, 5, 3, 0, 7, 7], dtype=int64)
```

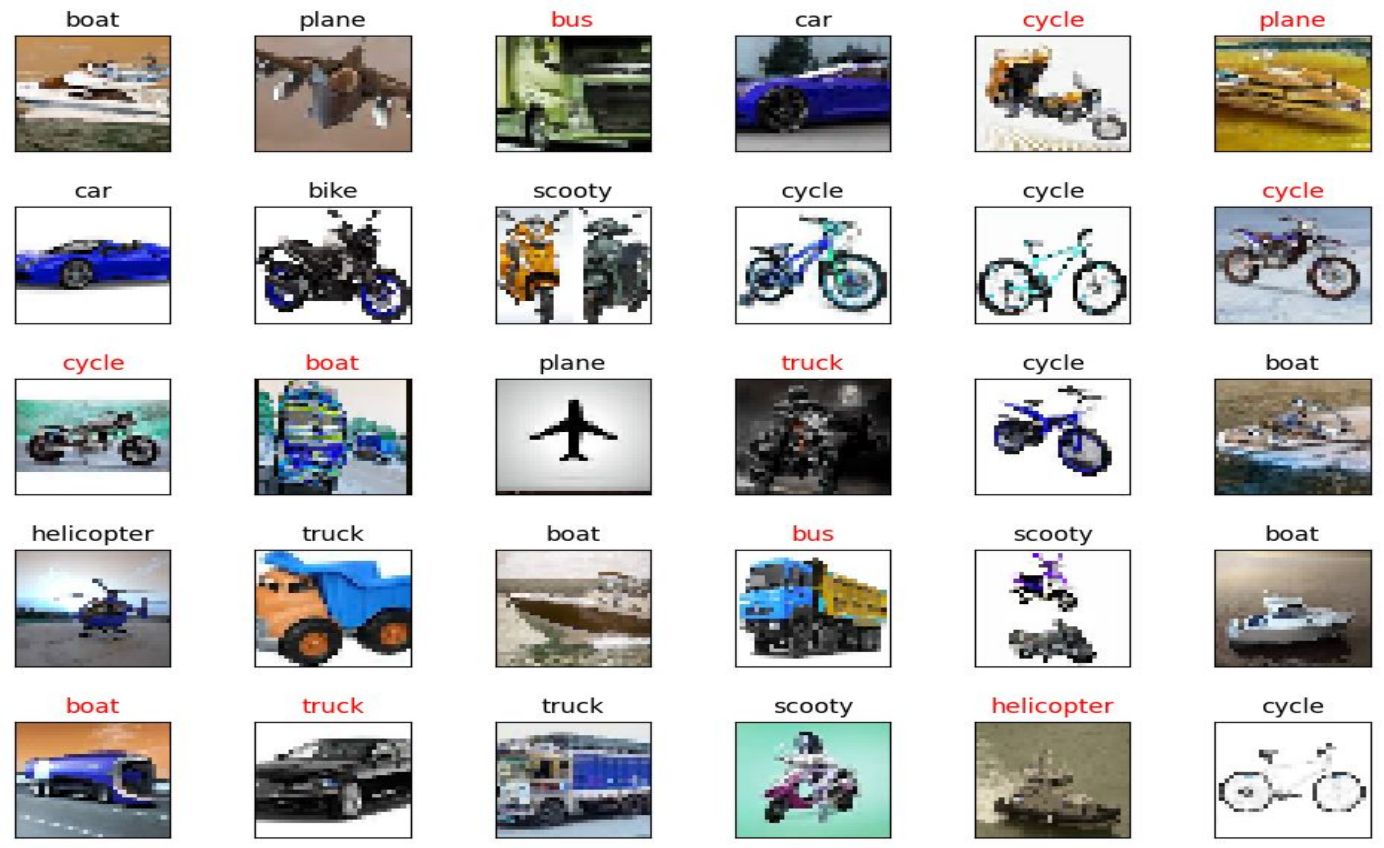
- **transforming_y_test:**

```
TRANSFORMING y_test:

liste_indizes = []

for i in y_test:
    a = 0
    for j in i:
        if j == 1:
            liste_indizes.append(a)
        a += 1
```

```
TRANSFORM IT IN AN ARRAY:

liste_indizes_arr = np.asarray(liste_indizes)
liste_indizes_arr
```

```
array([1, 6, 8, 3, 0, 1, 3, 0, 7, 4, 4, 0, 0, 8, 6, 0, 4, 1, 5, 8, 1, 8,
       7, 1, 8, 3, 8, 7, 1, 4, 1, 7, 8, 4, 3, 3, 0, 5, 3, 1, 5, 7, 0, 7,
       6, 0, 2, 3, 6, 2, 3, 7, 0, 8, 7, 0, 0, 5, 1, 4, 7, 4, 1, 5, 3, 4,
       0, 2, 2, 0, 1, 3, 3, 3, 5, 7, 3, 3, 5, 5, 1, 1, 7, 1, 2, 6, 3, 7,
       1, 6, 0, 1, 8, 8, 5, 5, 3, 4, 5, 8, 4, 5, 3, 0, 7, 7])
```

- **some pictures of the dataset with their predicted labels:**



CONCLUSION:

	MODEL	ACCURACY	TRAINING TIME
best Results	InceptionResNetV2	98,94%	150 sec - 10 epochs
minimally worse	Xception	95,74%	110 sec - 10 epochs
	NASNetMobile	94,66%	40 sec - 10 epochs
satisfying results	Conv2D(CNN)	62,26%	60 sec - 100 epochs

OVERALL CONCLUSION:

- ➔ A manageable number of images allowed the data set to be processed and analyzed well
- ➔ Very good results could be achieved with the functional model
- ➔ Satisfying results with the sequential model CNN
- ➔ If high accuracy is required, the functional models are clearly recommended
- ➔ For large data sets, a preselection can be made using the sequential model in order to avoid longer computing times