Projekt: Image_Recognition

1. Preliminary considerations:
   - When selecting the data set in the area of image recognition, I decided on a small size so that the running times of the individual models remained within limits. This means I can compare different methods and their results in a timely manner in order to filter out the best results.
   - Image recognition is used in many areas and is therefore essential. What is important to me is the requirement of an unprocessed data set that is at best categorized but by no means pre-trained. So they should be pure image files in .jpg, .png or similar format.
   - This project is about preparing and categorizing an existing data set so that I can identify a single image. The prepared data is then implemented with Keras using the Tensorflow neural networks libraries, which allows me to create models with which I can implement my image recognition. This falls into the area of deep learning. 2 basic methods are used here:
     1. Functional model → the split data (train/test) is pre-trained and then trained and evaluated with the best-working model
     2. Sequential model → a model is built with layers in which split data (X_train/y_train/X_test/y_test) is then trained and evaluated
2. Target:
   - ✔ From this deep learning area I choose 3 different Keras models with which I can best identify an image.
   - ✔ The most important factors here are the accuracy and running time, although it must of course be considered where it makes sense to make compromises, since high accuracy is often associated with long running times.

Inhaltsverzeichnis

. Get data - Image_Recognition

1. Import a free dataset from Kaggle

- Small image dataset of vehicles
- Download the dataset in .jpg format

. Functional model from Keras → Comparison of models

1. Prepare the data set into a data frame:

A) Data set contains images in separate folders of the respective category:

- importing the necessary libraries to get the data

```python

# import libraries
import numpy as np
import pandas as pd
from pathlib import Path
import matplotlib.pyplot as plt
import seaborn as sns
# execute if warnings should be ignored
import warnings
warnings.filterwarnings('ignore')

```

- all files with jpg format in a list → number of images for train/test

```python

# Create a list with the imagepaths for training and testing
#        from windows:  C:\Users\dietm\Documents\Data Science_
dloadpath = Path('./data_vehicle_recognition/')
imagepaths = list(dloadpath.rglob('**/*.jpg'))

```

```python
len(imagepaths)
```

```
522
```

- Create list of categories using folder names:

```python

# create a list with labels of the images:
labels = [str(imagepaths[i]).split("\\")[-2] for i in range(len(imagepaths))]

```

B) Create data frame:

- 1 Feature with the file paths as a string → Data
- 1 Feature with the category as a string → Label

```python

# create a DataFrame with the filepath (as string) and the labels of the images

paths = pd.Series(imagepaths, name='Filepath').astype(str)
labels = pd.Series(labels, name='Label')

# concatenate paths and labels
df = pd.concat([paths, labels], axis=1)

# shuffle the DataFrame and reset index and remove the old one:
df01 = df.sample(frac=1,random_state=42).reset_index(drop=True)

```

- Dataframe with 2 Features:

```python

df01.head()

```

	Filepath	Label
0	data_vehicle_recognition\vehicles\scooty\image...	scooty
1	data_vehicle_recognition\vehicles\scooty\image...	scooty
2	data_vehicle_recognition\vehicles\bike\2Q__(6...	bike
3	data_vehicle_recognition\vehicles\car\images (...	car
4	data_vehicle_recognition\vehicles\boat\images ...	boat

- there are 9 categories:

```
# 9 different labels and their titles:
df01.Label.unique()
```

```
array(['scooty', 'bike', 'car', 'boat', 'helicopter', 'truck', 'bus',
       'plane', 'cycle'], dtype=object)
```

- There are a total of 522 images with a different distribution in terms of categories

```
# total number of images:
df01.shape[0]
```

```
522
```

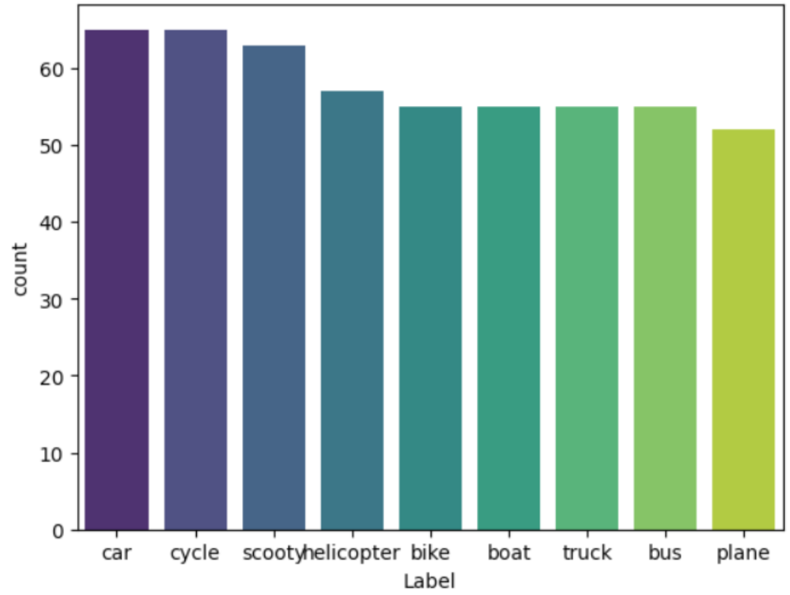
```
# total number of images each category:
df01.Label.value_counts()
```

```
Label
car           65
cycle         65
scooty        63
helicopter    57
bike          55
boat          55
truck         55
bus           55
plane         52
Name: count, dtype: int64
```

- the graphic makes this clear:

```
# number of images each category visualized: --> different numbers
sns.countplot(data=df01,x="Label",palette="viridis",order=df01.Label.value_counts().index)
```

```
<Axes: xlabel='Label', ylabel='count'>
```



- For better training conditions, I reduce the data frame so that each category has the same number of images available:

```python

# for a better train_data_set --> all categories with same size: 52
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]

```

- all categories now contain 52 images:

```python

# delete the rows of the data set:
df01.drop(index=list_indizes,inplace=True)

```

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

```
Label
truck         52
bus           52
plane         52
bike          52
boat          52
helicopter    52
car           52
scooty        52
cycle         52
Name: count, dtype: int64
```

- In total there are now 468 images that I re-index and discard the old indices:

```python

# create new indices for the data set and remove the old one:
df01.reset_index(drop=True,inplace=True)

```

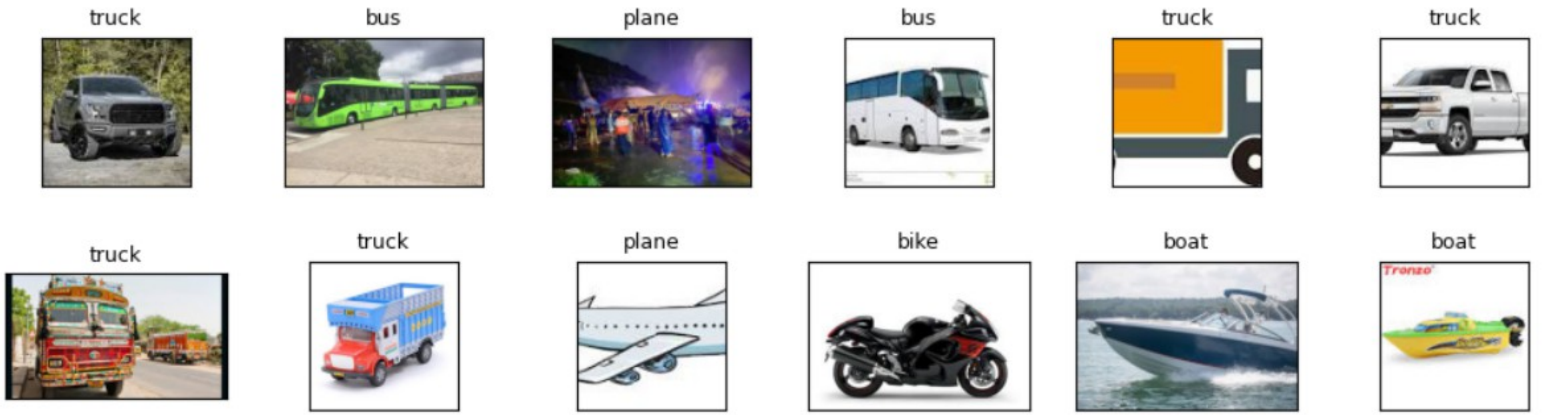
```python

# shape data set: 522 --> 468 rows with 2 features:
df01

```

	Filepath	Label
0	data_vehicle_recognition\vehicles\truck\2Q__.jpg	truck
1	data_vehicle_recognition\vehicles\bus\Z (12).jpg	bus
2	data_vehicle_recognition\vehicles\plane\9k_ (6...	plane
3	data_vehicle_recognition\vehicles\bus\images (...	bus
4	data_vehicle_recognition\vehicles\truck\images...	truck
...	...	...
463	data_vehicle_recognition\vehicles\boat\9k_ (5)...	boat
464	data_vehicle_recognition\vehicles\boat\Z (6).jpg	boat
465	data_vehicle_recognition\vehicles\cycle\images...	cycle
466	data_vehicle_recognition\vehicles\scooty\image...	scooty
467	data_vehicle_recognition\vehicles\boat\Z (2).jpg	boat

468 rows × 2 columns

- Visualizing a sample of images from my dataset:

```
# visualization of images with subplots without scalenumbers:
fig, axes = plt.subplots(nrows=2,ncols=6,subplot_kw={'xticks':[],'yticks':[]},figsize=(10,3))

for i,ax in enumerate(axes.flat):
    ax.imshow(plt.imread(df01.Filepath[i]))
    ax.set_title(df01.Label[i],fontsize=9)

plt.tight_layout(pad=2)
plt.show()
```



2. Prepare data

A) Load libraries for image recognition and train/test/split (For larger data sets, I would only use part of them so as not to run too long)

```python

# import library for image recognition:
import tensorflow as tf

```

```python

from sklearn.model_selection import train_test_split

```

```python

# train/test split
# use only a small part of the data set in the case of being very big
# --> "df01.sample(frac=0.1)"
train_df,test_df = train_test_split(df01, test_size=0.2,random_state=42)

```

B) I'm creating a function that loads and pre-trains the images so I can call it when I want to compare multiple models:

```
# load the images with a generator and data augmentation with a function:
def image_gen():

    train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
        preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input,
        validation_split=0.2)

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

    train_images = train_generator.flow_from_dataframe(dataframe=train_df,x_col='Filepath',y_col='Label',
                                                       target_size=(224, 224),class_mode='categorical',
                                                       batch_size=32,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")

    val_images = train_generator.flow_from_dataframe(dataframe=train_df,x_col='Filepath',y_col='Label',
                                                     target_size=(224, 224),class_mode='categorical',batch_size=32,
                                                     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")

    test_images = test_generator.flow_from_dataframe(dataframe=test_df,x_col='Filepath',y_col='Label',
                                                     target_size=(224, 224),class_mode='categorical',batch_size=32,
                                                     shuffle=False)

    return train_generator,test_generator,train_images,val_images,test_images
```

C) I create another function to call it and use the respective Keras model to turn the pre-trained model into a model that is then to be trained:

```python

# loading the pretrained model with an initialized keras_model:
def model_choice(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)

    # number of labels --> 9:
    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

```

D) a dictionary consisting of several Keras models and their runtimes with start time “0”. The compilation of the models is taken from a list from Keras, which indicates the calculated accuracies and their running times. I made sure that the accuracies had the highest values without exceeding worse running times

```python

# dictionary with the keras_models:
# --> a collection of models with good values in accuracy and training times:
models = {
    "ResNet101V2": {"model":tf.keras.applications.DenseNet121, "perf":0},
    "Xception": {"model":tf.keras.applications.Xception, "perf":0},
    "InceptionResNetV2": {"model":tf.keras.applications.InceptionResNetV2, "perf":0},
    "MobileNet": {"model":tf.keras.applications.MobileNet, "perf":0},
    "ResNet50V2": {"model":tf.keras.applications.ResNet50V2, "perf":0},
    "InceptionV3": {"model":tf.keras.applications.InceptionV3, "perf":0},
    "MobileNetV2": {"model":tf.keras.applications.MobileNetV2, "perf":0},
    "VGG16": {"model":tf.keras.applications.VGG16, "perf":0},
    "VGG19": {"model":tf.keras.applications.VGG19, "perf":0}
}

```

E) I need the library to calculate the running times: „perf_counter“

```python

# import library for counting the training time:
from time import perf_counter

```

F) Now I call my function for pre-training the images and train all models using a for loop → this way I get the running times and other values for precise evaluation

```

# create the generators:
train_generator,test_generator,train_images,val_images,test_images = image_gen()

# fit with all models:
for name, model in models.items():

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

    start = perf_counter()

    # fit the model
    history = m.fit(train_images,validation_data=val_images,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.
ResNet101V2          trained in 102.18 sec
Xception             trained in 111.27 sec
WARNING:tensorflow:From C:\Users\dietm\anaconda3\lib\site-packages\keras\src\backend\tensorflow\core.py:170: The name tf.p
laceholder is deprecated. Please use tf.compat.v1.placeholder instead.

InceptionResNetV2    trained in 156.81 sec
MobileNet            trained in 27.41 sec
ResNet50V2           trained in 90.05 sec
InceptionV3          trained in 65.98 sec
MobileNetV2          trained in 31.25 sec
VGG16                trained in 249.48 sec
VGG19                trained in 326.92 sec
```

3. Evaluation:

A) I import libraries for further evaluation:

```
# import libraries for evaluation:
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

B) I output the predictions using a for loop:

```
# evaluation of all models
for name, model in models.items():

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

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

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

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 15s 4s/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 11s 3s/step
WARNING:tensorflow:5 out of the last 7 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_dat
a_distributed at 0x000002756B1BD240> triggered tf.function retracing. Tracing is expensive and the excessive number of tra
cings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passi
ng Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function
has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.or
g/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
2/3 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step  WARNING:tensorflow:6 out of the last 9 calls to <function TensorFlowTrainer.make_pred
ict_function.<locals>.one_step_on_data_distributed at 0x000002756B1BD240> triggered tf.function retracing. Tracing is expe
nsive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing ten
sors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outsid
e of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), pleas
e refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/
tf/function for  more details.
3/3 ━━━━━━━━━━━━━━━━━━━━ 23s 6s/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 954ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 10s 3s/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 10s 3s/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 21s 7s/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 27s 9s/step
```

C) I create a dataframe with the results

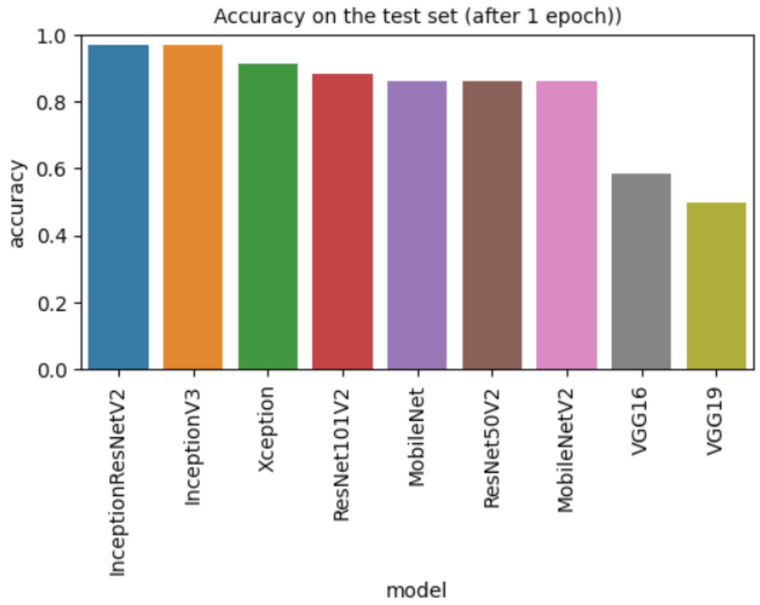
```python

# create a DataFrame 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	0.9681	156.81
1	InceptionV3	0.9459	0.9681	65.98
2	Xception	0.9324	0.9149	111.27
3	ResNet101V2	0.8514	0.8830	102.18
4	MobileNet	0.9189	0.8617	27.41
5	ResNet50V2	0.8919	0.8617	90.05
6	MobileNetV2	0.8919	0.8617	31.25
7	VGG16	0.6486	0.5851	249.48
8	VGG19	0.5811	0.5000	326.92

4. Visualizations of accuracy and runtimes:

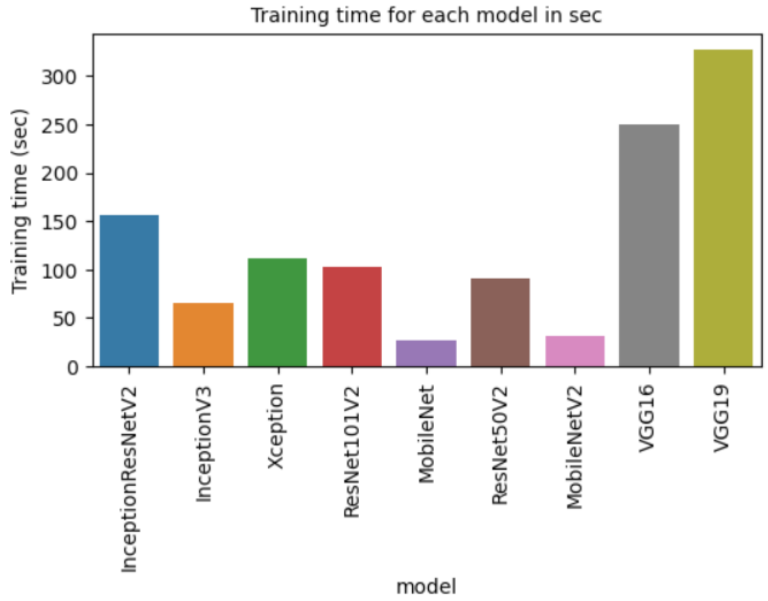
Accuracy

```python
plt.figure(figsize = (6,3))
sns.barplot(x = 'model', y = 'accuracy', data = df_results)
plt.title('Accuracy on the test set (after 1 epoch))', fontsize = 10)
plt.ylim(0,1)
plt.xticks(rotation=90)
plt.show()
```



Duration

```python
plt.figure(figsize = (6,3))
sns.barplot(x = 'model', y = 'Training time (sec)', data = df_results)
plt.title('Training time for each model in sec', fontsize = 10)
# plt.ylim(0,20)
plt.xticks(rotation=90)
plt.show()
```



. Functional model from Keras with InceptionResNetV2 (best in comparison)

1 Initialize generators:

```
# load the images with a generator and Data Augmentation
train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input,
    validation_split=0.2)

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

2 Load images and pre-train with the already split data set: → train_df, test_df

```
train_images = train_generator.flow_from_dataframe(dataframe=train_df,x_col='Filepath',y_col='Label',
                                                    target_size=(224, 224),class_mode='categorical',
                                                    batch_size=32,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")

val_images = train_generator.flow_from_dataframe(dataframe=train_df,x_col='Filepath',y_col='Label',
                                                  target_size=(224, 224),class_mode='categorical',batch_size=32,
                                                  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")

test_images = test_generator.flow_from_dataframe(dataframe=test_df,x_col='Filepath',y_col='Label',
                                                  target_size=(224, 224),class_mode='categorical',batch_size=32,
                                                  shuffle=False)
```

```
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.
```

3 Initializing the Keras model: → InceptionResNetV2

```
# initializing the best keras_model of my test:
# --> "recognition_vehicle_03-models.ipynb"
model = tf.keras.applications.InceptionResNetV2
```

4 Loading the Keras model

```
# Loading the pretrained 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)

# number of labels --> 9:
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'])

```

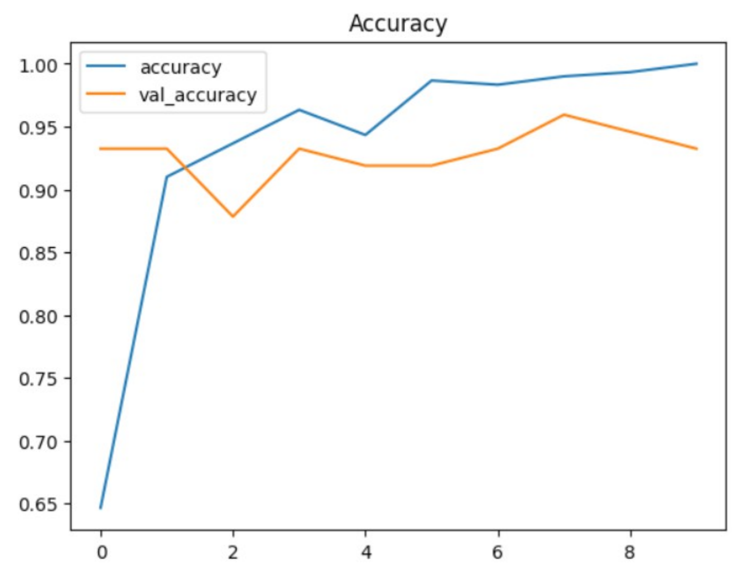
5 Training the model

```python

# fit the model
history = model.fit(train_images,validation_data=val_images,epochs=10)

```

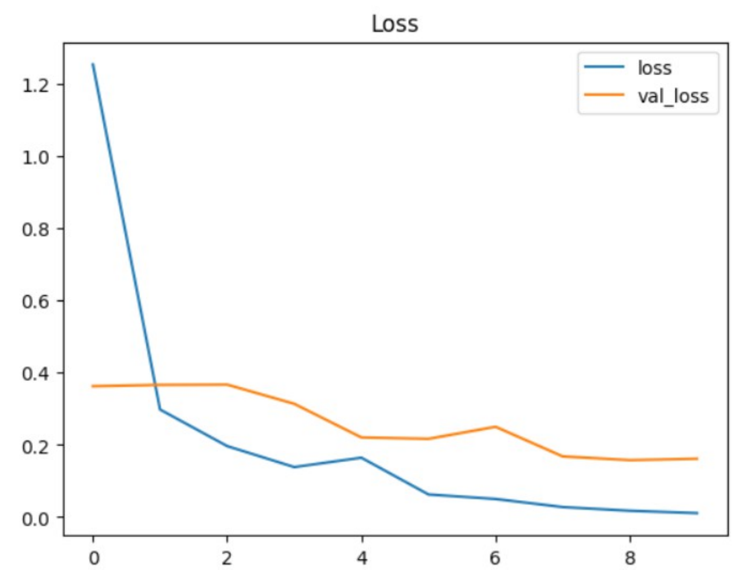
```
Epoch 1/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 66s 5s/step - accuracy: 0.4653 - loss: 1.6635 - val_accuracy: 0.9324 - val_loss: 0.3626
Epoch 2/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 44s 4s/step - accuracy: 0.8847 - loss: 0.3614 - val_accuracy: 0.9324 - val_loss: 0.3663
Epoch 3/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 44s 5s/step - accuracy: 0.9285 - loss: 0.2054 - val_accuracy: 0.8784 - val_loss: 0.3671
Epoch 4/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 45s 4s/step - accuracy: 0.9642 - loss: 0.1463 - val_accuracy: 0.9324 - val_loss: 0.3139
Epoch 5/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 45s 4s/step - accuracy: 0.9454 - loss: 0.1654 - val_accuracy: 0.9189 - val_loss: 0.2207
Epoch 6/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 45s 5s/step - accuracy: 0.9857 - loss: 0.0704 - val_accuracy: 0.9189 - val_loss: 0.2169
Epoch 7/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 44s 4s/step - accuracy: 0.9838 - loss: 0.0495 - val_accuracy: 0.9324 - val_loss: 0.2502
Epoch 8/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 45s 5s/step - accuracy: 0.9919 - loss: 0.0263 - val_accuracy: 0.9595 - val_loss: 0.1682
Epoch 9/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 44s 4s/step - accuracy: 0.9981 - loss: 0.0139 - val_accuracy: 0.9459 - val_loss: 0.1578
Epoch 10/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 44s 4s/step - accuracy: 1.0000 - loss: 0.0091 - val_accuracy: 0.9324 - val_loss: 0.1618
```

6 Visualization of accuracy and loss

```python
# visualization of accuracy and val_accuracy:
pd.DataFrame(history.history)[['accuracy','val_accuracy']].plot()
plt.title("Accuracy")
plt.show()
```




```python
# visualization of loss and val_loss:
pd.DataFrame(history.history)[['loss','val_loss']].plot()
plt.title("Loss")
plt.show()
```



7 Predictions

```python
# predict the label of the test_images
pred = model.predict(test_images)
pred = np.argmax(pred,axis=1)

# map the label
labels = (train_images.class_indices)
labels = dict((v,k) for k,v in labels.items())
pred = [labels[k] for k in pred]
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 24s 7s/step
```

8 Evaluation

A) Import libraries:

```
# import libraries for evaluation:
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

B) Calculate accuracy:

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

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

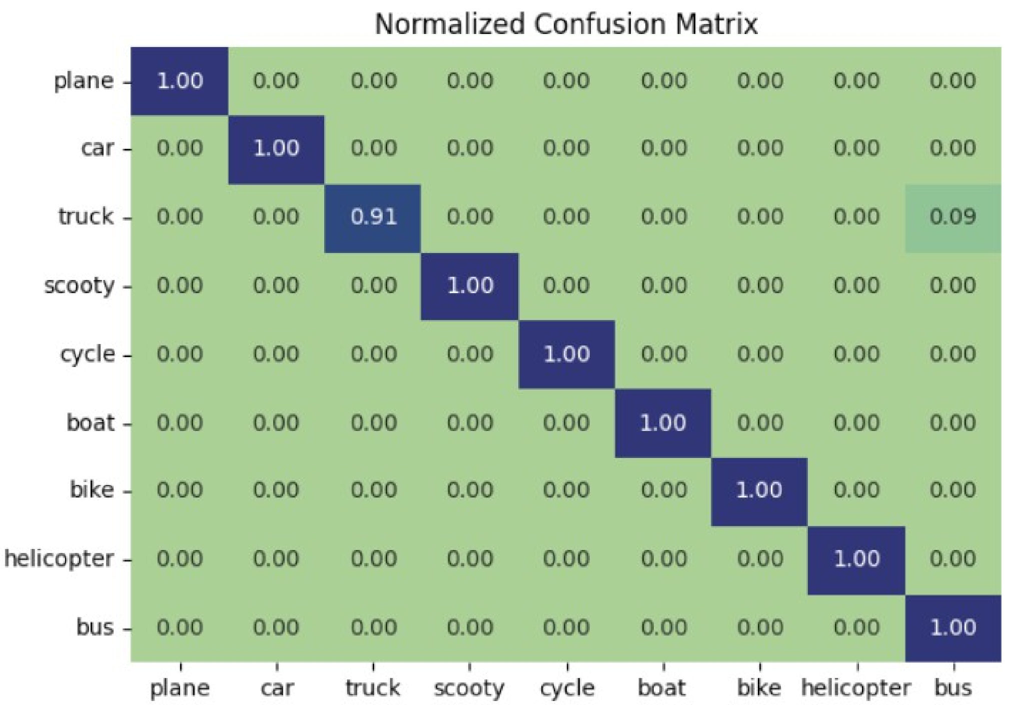
C) Classification-Report:

```
class_report = classification_report(y_test, pred, 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      0.91      0.95        11
         car       1.00      1.00      1.00         8
       cycle       1.00      1.00      1.00        10
  helicopter       1.00      1.00      1.00        11
       plane       1.00      1.00      1.00        14
      scooty       1.00      1.00      1.00         9
       truck       0.83      1.00      0.91         5

    accuracy                           0.99        94
   macro avg       0.98      0.99      0.98        94
weighted avg       0.99      0.99      0.99        94
```

D) Visualization of the evaluation:

```
cf_matrix = confusion_matrix(y_test, pred, normalize='true')
plt.figure(figsize = (7,5))
sns.heatmap(cf_matrix,annot=True,xticklabels=set(y_test),yticklabels=set(y_test),cmap="crest",cbar=False,fmt=".2f")
plt.title('Normalized Confusion Matrix', fontsize = 12)
plt.xticks()
plt.yticks()
plt.show()
```



E) Visualization of the evaluation with the images

```python

# display picture of the dataset with their labels/predictions:
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(20,10),subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    color = 'black'
    if test_df.Label.iloc[i].split('_')[0] != pred[i].split('_')[0]:
        # in case of a false prediction --> red font
        color = 'r'
    ax.imshow(plt.imread(test_df.Filepath.iloc[i]))
    ax.set_title(f"True: {test_df.Label.iloc[i].split('_')[0]}\nPredicted: {pred[i].split('_')[0]}", fontsize = 15,color=color)
plt.tight_layout(pad=4)
plt.show()

```

. Sequential model from Keras → CNN (Convolutional Neural Network) with 2 layers

1 Get the data set:

- ✗ Import os:

```python

# Input data files from windows:
# "C:\Users\dietm\Documents\Data Science_Didi
import os

```

- ✗ This allows me to output the path name/file name:

```python

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

```

```
./data_vehicle_recognition/vehicles\bike\2Q__ (6).jpg
./data_vehicle_recognition/vehicles\bike\2Q__ (7).jpg
./data_vehicle_recognition/vehicles\bike\2Q__ (8).jpg
./data_vehicle_recognition/vehicles\bike\2Q__ (9).jpg
./data_vehicle_recognition/vehicles\bike\2Q__.jpg
./data_vehicle_recognition/vehicles\bike\9k_ (1).jpg
./data_vehicle_recognition/vehicles\bike\9k_ (10).jpg
./data_vehicle_recognition/vehicles\bike\9k_ (11).jpg
./data_vehicle_recognition/vehicles\bike\9k_ (2).jpg
./data_vehicle_recognition/vehicles\bike\9k_ (3).jpg
```

✗ I prepare the path:

```python

# preparing the path:
#C:\Users\dietm\Documents\Data Science_Didi\Projekt\Projekt Recognition\data_vehicle_recognition\vehicles
root_dir = './data_vehicle_recognition/vehicles/'

```

✗ so I can read the .jpg files → import cv2:

```python

# for reading images:
import cv2

```

✗ prepare data and labels:

```python

# preparing datas and labels in lists:
data = []
labels = []
label_names = []

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 label_names:
            label_names.append(label)

```

✗ the names of the individual categories:

```

# labels unique:
label_names

```

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

✗ I pack image data and labels into the correct shape:

```

# preparing right shape for datas/labels:
data = np.array(data)

```

```
data.shape
```

```
(525, 32, 32, 3)
```

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

```
labels.shape
```

```
(525,)
```

✗ I still have to convert the labels into numerical form:

```python

# import for preparing labels:
from sklearn.preprocessing import LabelEncoder

```

```python

# initialize and preparing:
le = LabelEncoder()
labels = le.fit_transform(labels)

```

```python

# import for converting to categorical:
from tensorflow.keras.utils import to_categorical

```

```python

labels = to_categorical(labels)

```

```python

# right shape:
labels.shape,data.shape

```

```
((525, 9), (525, 32, 32, 3))
```

✗ I use the indices to randomize the data and labels:

```python

# before train/test/split --> a shu
new = np.arange(525)

```

```python
new
```

```
array([  0,    1,    2,    3,    4,    5,
        13,   14,   15,   16,   17,   18,   1
```

```
np.random.shuffle(new)
```

```
new
```

```
array([123, 164, 215, 157,  82, 264,  87, 30
       252, 426, 334, 257, 266, 313, 213, 33
```

```
# --> shuffle with indices:
data = data[new]
labels = labels[new]
```

2 train_test_split

✗ Create train-/test-data:

```
# import for --> 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)
```

3 Sequential model

- Import layers and models:

```python

# import librarys from tensorflow for layers and models:
from tensorflow.keras.layers import Dense,Conv2D,Flatten,MaxPool2D,Dropout,LeakyReLU
from tensorflow.keras.models import Sequential

```

- Create a model with multiple layers:

```python

# create model:
model = Sequential([
    Conv2D(32,(3,3),padding="same",activation="relu",input_shape=(32,32,3)),
    Conv2D(32,(3,3),activation="relu"),

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

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

```

✗ Compile for improved use of the model:

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

4 Train model

```
# train:
history = model.fit(X_train,y_train,batch_size=32,epochs=8,validation_split=0.25)
```

```
Epoch 1/8
10/10 ━━━━━━━━━━━━━━━━━━━━ 3s 157ms/step - accuracy: 0.1266 - loss: 57.6988 - val_accuracy: 0.1333 - val_loss: 2.6192
Epoch 2/8
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 132ms/step - accuracy: 0.1297 - loss: 2.6400 - val_accuracy: 0.1238 - val_loss: 2.1691
Epoch 3/8
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 131ms/step - accuracy: 0.1327 - loss: 2.1504 - val_accuracy: 0.2381 - val_loss: 2.1530
Epoch 4/8
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 132ms/step - accuracy: 0.1543 - loss: 2.1167 - val_accuracy: 0.1429 - val_loss: 2.1540
Epoch 5/8
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 132ms/step - accuracy: 0.2242 - loss: 2.0339 - val_accuracy: 0.2190 - val_loss: 2.0965
Epoch 6/8
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 133ms/step - accuracy: 0.2416 - loss: 1.9633 - val_accuracy: 0.1905 - val_loss: 2.0756
Epoch 7/8
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 144ms/step - accuracy: 0.2547 - loss: 1.8828 - val_accuracy: 0.3143 - val_loss: 1.9523
Epoch 8/8
10/10 ━━━━━━━━━━━━━━━━━━━━ 2s 150ms/step - accuracy: 0.3393 - loss: 1.7670 - val_accuracy: 0.2857 - val_loss: 1.8996
```

5 Evaluation

✗ Creating a data frame with the evaluated data:

```python
# values of accuracy and loss:
history_df = pd.DataFrame(history.history)
```

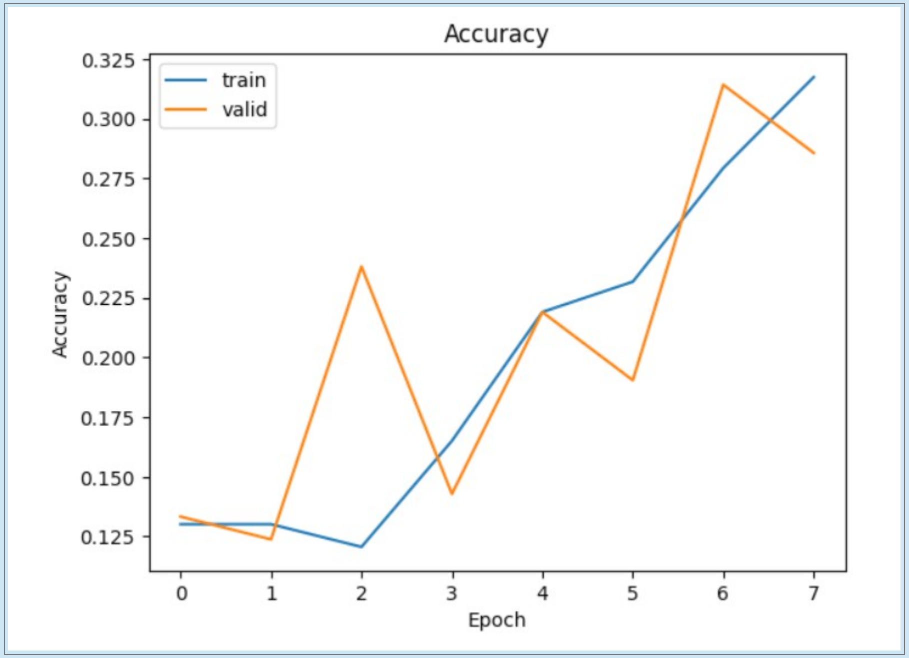
```python
history_df.tail(5)
```

	accuracy	loss	val_accuracy	val_loss
3	0.165079	2.110657	0.142857	2.153957
4	0.219048	2.025803	0.219048	2.096495
5	0.231746	1.988327	0.190476	2.075573
6	0.279365	1.862987	0.314286	1.952262
7	0.317460	1.755574	0.285714	1.899643

6 Visualization

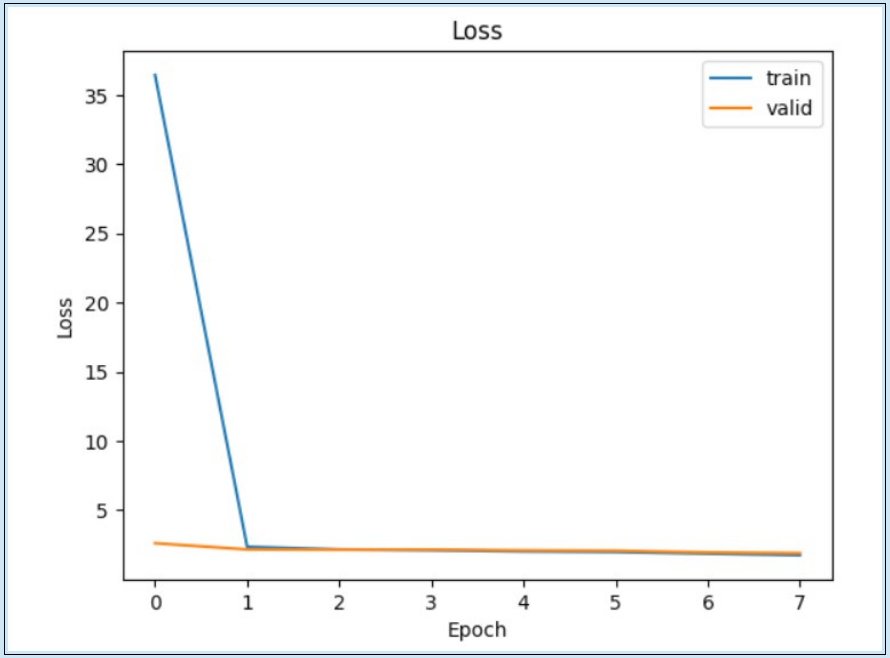
Accuracay

```
# visualization of accuracy and val_accuracy:
history_df[['accuracy','val_accuracy']].plot()
plt.title("Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend(["train", "valid"])
plt.show()
```



Loss

```

# visualization of accuracy and val_accuracy:
history_df[['loss','val_loss']].plot()
plt.title("Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(["train", "valid"])
plt.show()

```



7 Predictions

✗ evaluate:

```
# evaluate:
model.evaluate(X_test,y_test)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.3722 - loss: 1.9289

[1.9143372774124146, 0.3523809611797333]
```

✗ predict:

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

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step

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

✗ I need to put the label test data into the same form as the predictions:

```

# convert y_test to an array like label_classes
liste_indices = []
for i in y_test:
    a = 0
    for j in i:
        if j == 1:
            liste_indices.append(a)
        a += 1

```

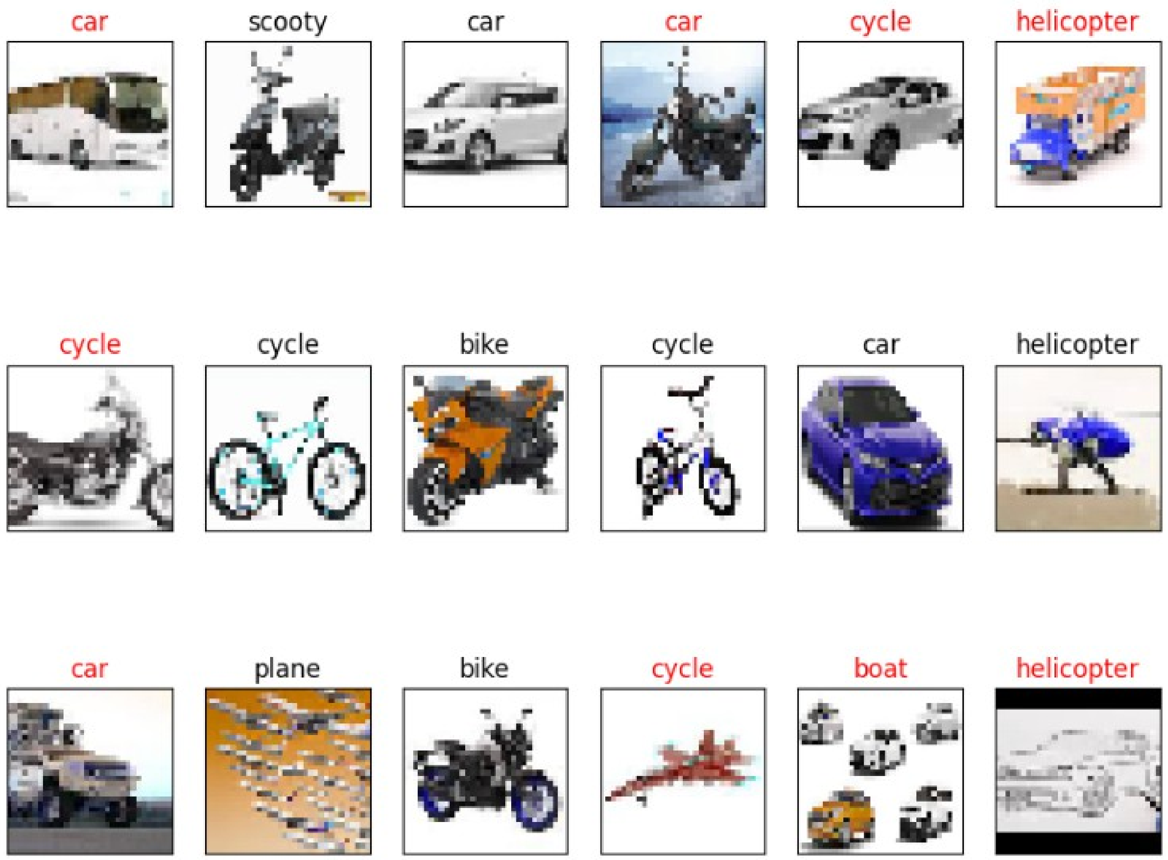
```
liste_indices_arr = np.asarray(liste_indices)
```

```
liste_indices_arr
```

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

✗ Now I can also visually evaluate my predictions: → false predictions in red

```python

fig, ax = plt.subplots(3, 6, subplot_kw=dict(xticks=[], yticks=[]), figsize=(10,8))
for i, axi in enumerate(ax.flat):
    color = 'black'
    if label_classes[i]!=liste_indices_arr[i]:
        color='r'
    axi.imshow(X_test[i], cmap='Blues_r')
    axi.set_title(label_names[label_classes[i]], color=color)
plt.show()

```



. Sequential model from Keras → CNN (Convolutional Neural Network) with 1 layer

1 Get the data set:

✗ Import os:

```python

# Input data files from windows:
# "C:\Users\dietm\Documents\Data Science_Didi
import os

```

✗ This allows me to output the path name/file name:

```python

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

```

```
./data_vehicle_recognition/vehicles\bike\2Q__ (6).jpg
./data_vehicle_recognition/vehicles\bike\2Q__ (7).jpg
./data_vehicle_recognition/vehicles\bike\2Q__ (8).jpg
./data_vehicle_recognition/vehicles\bike\2Q__ (9).jpg
./data_vehicle_recognition/vehicles\bike\2Q__.jpg
./data_vehicle_recognition/vehicles\bike\9k_ (1).jpg
./data_vehicle_recognition/vehicles\bike\9k_ (10).jpg
./data_vehicle_recognition/vehicles\bike\9k_ (11).jpg
./data_vehicle_recognition/vehicles\bike\9k_ (2).jpg
./data_vehicle_recognition/vehicles\bike\9k_ (3).jpg
```

- ✗ I prepare the path:

```
# preparing the path:
#C:\Users\dietm\Documents\Data Science_Didi\Projekt\Projekt Recognition\data_vehicle_recognition\vehicles
root_dir = './data_vehicle_recognition/vehicles/'
```

- ✗ so that I can read the .jpg files → import cv2:

```
# for reading images:
import cv2
```

- ✗ prepare data and labels:

```
# preparing datas and labels in lists:
data = []
labels = []
label_names = []

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 label_names:
            label_names.append(label)
```

- ✗ the names of the individual categories:

```
# labels unique:
label_names
```

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

- ✗ I pack image data and labels into the correct shape:

```
# preparing right shape for datas/labels:
data = np.array(data)
```

```
data.shape
```

```
(525, 32, 32, 3)
```

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

```
labels.shape
```

```
(525,)
```

✗ I still have to convert the labels into numerical form:

```
# import for preparing labels:
from sklearn.preprocessing import LabelEncoder
```

```
# initialize and preparing:
le = LabelEncoder()
labels = le.fit_transform(labels)
```

```
# import for converting to categorical:
from tensorflow.keras.utils import to_categorical
```

```
labels = to_categorical(labels)
```

```
# right shape:
labels.shape,data.shape
```

```
((525, 9), (525, 32, 32, 3))
```

✗ I use the indices to randomize the data and labels:

```
# before train/test/split --> a shu
new = np.arange(525)
```

```
new
```

```
array([  0,    1,    2,    3,    4,    5,
```

```python
np.random.shuffle(new)
```

```python
new
```

```
array([123, 164, 215, 157,  82, 264,  87, 30
       252, 426, 334, 257, 266, 313, 213, 33
```

```python
# --> shuffle with indices:
data = data[new]
labels = labels[new]
```

2 train_test_split

✗ Create train/test data:

```python
# import for --> train/test/split
from sklearn.model_selection import train_test_split
```

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

3 Sequential model

✗ Import layers and models:

```
# import librarys from tensorflow for layers and models:
from tensorflow.keras.layers import Dense,Conv2D,Flatten,MaxPooling2D,Dropout
from tensorflow.keras.models import Sequential
```

✗ Create a model with one layer:

```
# create model:
num_filters = 8
filter_size = 3
pool_size = 2

model = Sequential([
  Conv2D(num_filters, filter_size, input_shape=(100, 100, 3), activation='relu'),
  MaxPooling2D(pool_size=pool_size),
  Flatten(),
  Dense(9, activation='softmax'),
])
```

✗ Compile for improved use of the model:

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

4 Train model

```python

# train:
history = model.fit(X_train,y_train,batch_size=32,epochs=8, validation_data=(X_test, y_test))

```

```
Epoch 1/8
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.1154 - loss: 948.7361 - val_accuracy: 0.1238 - val_loss: 779.6876
Epoch 2/8
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.2081 - loss: 431.2361 - val_accuracy: 0.1905 - val_loss: 218.6804
Epoch 3/8
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.4100 - loss: 111.9271 - val_accuracy: 0.3238 - val_loss: 41.6199
Epoch 4/8
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.6350 - loss: 17.7357 - val_accuracy: 0.4000 - val_loss: 29.4796
Epoch 5/8
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.8154 - loss: 4.4171 - val_accuracy: 0.4190 - val_loss: 26.8124
Epoch 6/8
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.9289 - loss: 0.6398 - val_accuracy: 0.4000 - val_loss: 25.4787
Epoch 7/8
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.9743 - loss: 0.2234 - val_accuracy: 0.4571 - val_loss: 24.8750
Epoch 8/8
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.9882 - loss: 0.2197 - val_accuracy: 0.4571 - val_loss: 23.9534
```

5 Evaluation

✗ Creating a data frame with the evaluated data:

```python

# values of accuracy and loss:
history_df = pd.DataFrame(history.history)

```

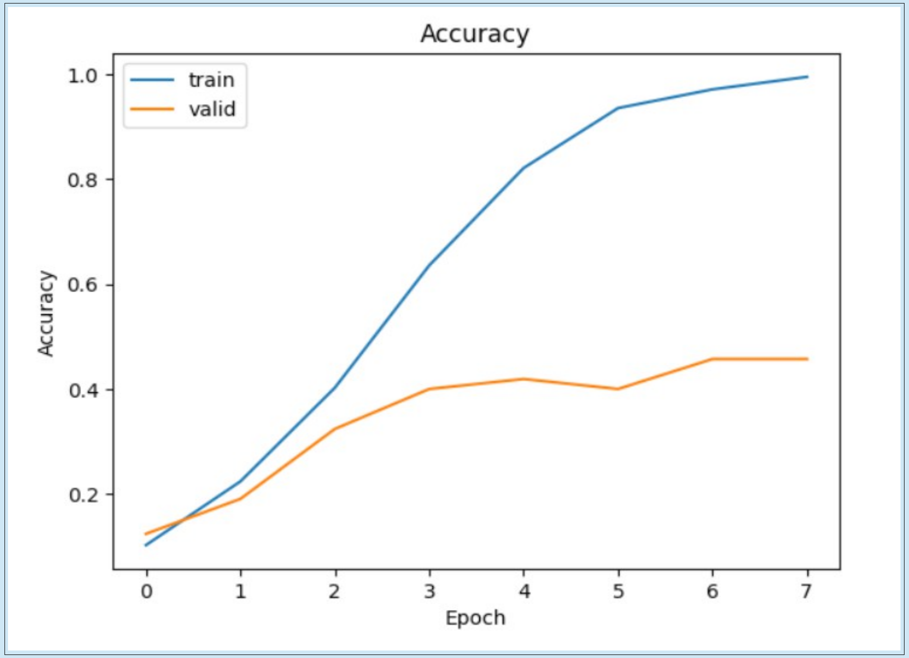
```python

history_df.tail(5)

```

	accuracy	loss	val_accuracy	val_loss
3	0.635714	15.546529	0.400000	29.479616
4	0.821429	4.320020	0.419048	26.812412
5	0.935714	0.527543	0.400000	25.478745
6	0.971429	0.273153	0.457143	24.875044
7	0.995238	0.083526	0.457143	23.953356

6 Visualization

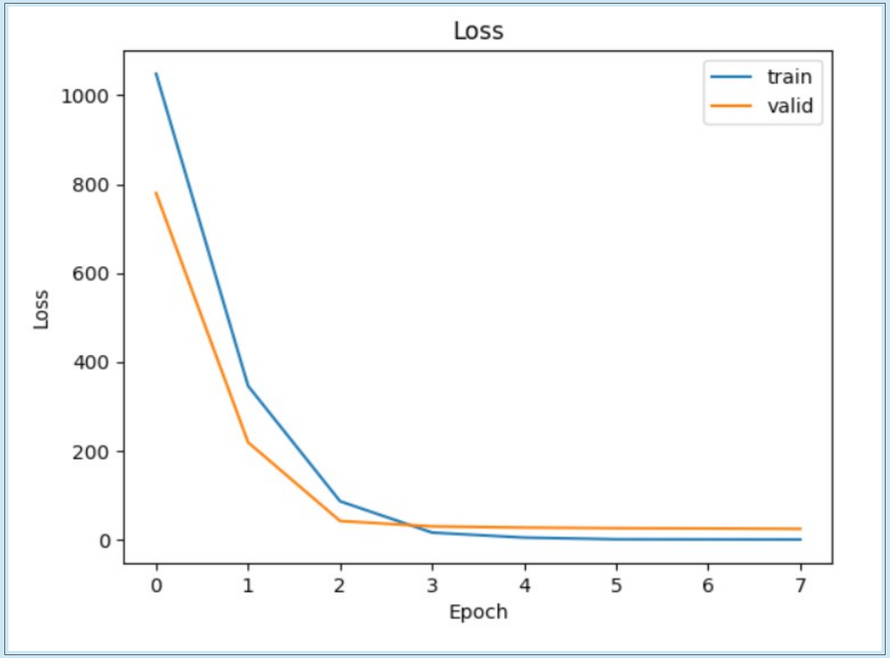
Accuracay

```python
# visualization of accuracy and val_accuracy:
history_df[['accuracy','val_accuracy']].plot()
plt.title("Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend(["train", "valid"])
plt.show()
```



Loss

```python

# visualization of accuracy and val_accuracy:
history_df[['loss','val_loss']].plot()
plt.title("Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(["train", "valid"])
plt.show()

```



7 Predictions

✗ evaluate:

```
# evaluate:
model.evaluate(X_test,y_test)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.4339 - loss: 23.4211

[23.95335578918457, 0.4571428596973419]
```

✗ predict:

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

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step

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

✗ I need to put the label test data into the same form as the predictions:

```python

# convert y_test to an array like label_classes
liste_indices = []
for i in y_test:
    a = 0
    for j in i:
        if j == 1:
            liste_indices.append(a)
        a += 1

```

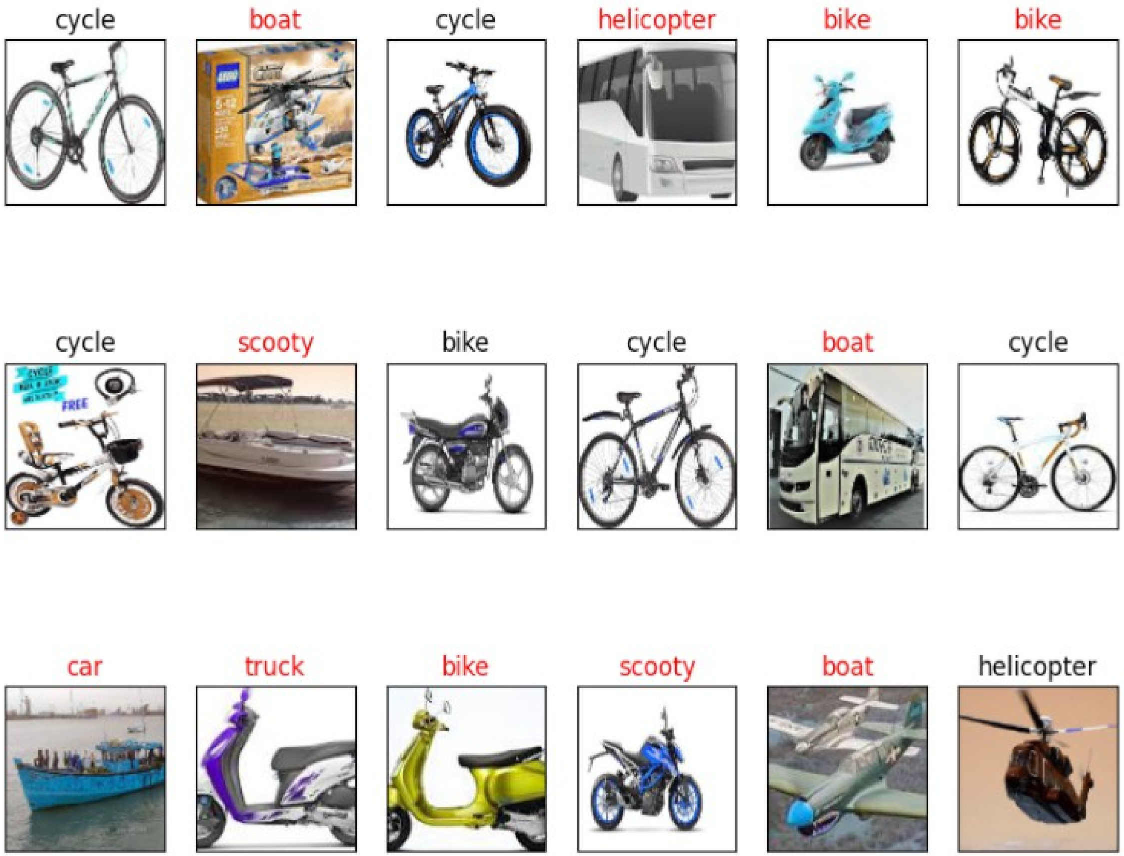
```python
liste_indices_arr = np.asarray(liste_indices)
```

```python
liste_indices_arr
```

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

✗ Now I can also visually evaluate my predictions: → false predictions in red

```
fig, ax = plt.subplots(3, 6, subplot_kw=dict(xticks=[], yticks=[]), figsize=(10,8))
for i, axi in enumerate(ax.flat):
    color = 'black'
    if label_classes[i]!=liste_indices_arr[i]:
        color='r'
    axi.imshow(X_test[i], cmap='Blues_r')
    axi.set_title(label_names[label_classes[i]], color=color)
plt.show()
```



✔ **conclusion**:

CNN (multiple Layers)	**CNN (one Layer)**
• • More effective correct predictions → see visualization!	• Much faster run times ◦ higher accuracy • requires more image data

. Overall conclusion:

- A very small data set of .jpg files could be easily divided into 2 features
  - Image data as a data feature
  - the second feature as a label
- I was able to achieve very good results by comparing several good, functional models after evaluating the best one.
- The results with the two sequential models were rather mediocre and, in contrast to the functional model, could not be improved with the small volume of the data set. However, the running times for the sequential models were significantly shorter; even the one equipped with just one layer was a lot faster than the one with 2 layers. The fact is that the sequential models are much less complex than the functional ones.
- Evaluation of the 3 Keras models:

Functional Model:	**Sequential Model with 2 Layers:**	**Sequential Model with 1 Layer:**
InceptionResNetV2: Accuracy: 98,94 %	Conv2D: Accuracy: 37,22 %	Conv2D: Accuracy: 43,39 %

. outlook into the future:

- Predictions: There will probably always be a discussion about how the respective problem of image recognition relates to accuracy and runtimes. Especially in the case of permanent image recognition during surveillance, for example, accuracy is less important than the time factor. In contrast, perhaps there is control in the production of mechanical objects that must not have any tolerances. However, the challenge remains that you can continue to optimize indefinitely.