In [4]:

```
from PIL import Image
import cv2
import os
import numpy as np
from tqdm import tqdm
```

Раскадровка

In [3]:

```
import cv2
def video_to_frame(video, name):
    global count
    k=0

    if not os.path.exists(name):
        os.makedirs(name)

    video_capture = cv2.VideoCapture(video)
    fps = video_capture.get(cv2.CAP_PROP_FPS)
    frame_count = int(video_capture.get(cv2.CAP_PROP_FRAME_COUNT))

    success, image = video_capture.read()
    while success:
        if count % int(fps) == 0:
            cv2.imwrite(f'{name}/{name}{count}.jpg', image)
        success, image = video_capture.read()
        count += 1
        k += 1

    video_capture.release()
```

In [91]:

```
files = os.listdir('Samokat1')
```

In [6]:

```
count=0
```

In [93]:

```
# Самокат
for file in tqdm(files):
    video_to_frame('Samokat1/'+file, 'Samokat')
```

```
100%|████████████████████████████████████████████████████████████████
████████████████| 9/9 [11:55<00:00, 79.45s/it]
```

Обнаружение и обрезка

In [10]:

```
# Загрузка детектора объектов YOLO и конфигурационного файла
net = cv2.dnn.readNetFromDarknet('yolov3.cfg', 'yolov3.weights')

# Список названий классов объектов
classes = ['bicycle', 'motorbike']
```

In [11]:

```
def crop1(file, output, name):
    global k

    # Загрузка изображения
    image = Image.open(file)
    image = np.asarray(image)

    if not os.path.exists(output):
        os.makedirs(output)

    # Изменение размера изображения
    height, width, _ = image.shape
    scale = 0.00392
    blob = cv2.dnn.blobFromImage(image, scale, (416,416), (0,0,0), True, crop=False)

    # Подача изображения на вход детектору объектов
    net.setInput(blob)

    # Получение выходных данных детектора объектов
    outs = net.forward(net.getUnconnectedOutLayersNames())

    # Обход всех обнаруженных объектов
    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]

            # Если обнаружен объект класса "велосипед" или "мотоцикл" с достаточно высок
            if class_id in [0,1] and confidence > 0.5:
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width) + 50
                h = int(detection[3] * height) + 50

                # Вычисление координат для обрезки изображения
                x = center_x - w // 2
                y = center_y - h // 2

                # Обрезка изображения
                try:
                    cropped_image = image[y:y+h, x:x+w]
                except:
                    print(f"Error: unable to crop image {name}{k}.jpg")
                    continue

                # Изменение размера изображения до 256x256
                try:
                    if not cropped_image.size == 0:
                        resized_image = cv2.resize(cropped_image, (256, 256))
                    else:
                        print(f"Error: resized image {name}{k}.jpg is empty")
                        continue
                except cv2.error as e:
                    print(f"Error: {e}")
                    continue

                # Сохранение обрезанного и измененного изображения
                try:
```

```
                cv2.imwrite(f'{output}/{name}{k}.jpg', resized_image)
            except:
                print(f"Error: unable to save image {name}{k}.jpg")
                continue
            k += 1
```

In [128]:

```
files = os.listdir('Samokat')
```

In [129]:

```
k = 0
```

In [130]:

```
# Самокат
for file in tqdm(files):
    crop1(f'Samokat/{file}', 'Class1', 'Samokat')
```

```
  1%|▌
| 25/3241 [00:09<22:35,  2.37it/s]

Error: resized image Samokat54.jpg is empty
Error: resized image Samokat54.jpg is empty

  2%|▉
| 61/3241 [00:24<21:11,  2.50it/s]

Error: resized image Samokat114.jpg is empty

  5%|███
| 148/3241 [00:58<19:11,  2.69it/s]

Error: resized image Samokat262.jpg is empty
Error: resized image Samokat262.jpg is empty

  5%|███▍
| 169/3241 [01:06<18:20,  2.79it/s]

Error: resized image Samokat291.jpg is empty
```

In [ ]:

In [ ]:

In [ ]:

Основная часть

In [1]:

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
import random
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
```

In [2]:

```
scooter_folder = 'Class1'
bike_folder = 'Class3'
mono_folder = 'Class2'

all_scooter_images = os.listdir(scooter_folder)
scooter_images = random.sample(all_scooter_images, k=1700)

all_bike_images = os.listdir(bike_folder)
bike_images = random.sample(all_bike_images, k=1700)

all_mono_images = os.listdir(mono_folder)
mono_images = random.sample(all_mono_images, k=1700)
```

In [3]:

```
scooter_data = []
for image in scooter_images:
    img = Image.open(os.path.join(scooter_folder, image))
    img_array = np.array(img)
    scooter_data.append(img_array)

bike_data = []
for image in bike_images:
    img = Image.open(os.path.join(bike_folder, image))
    img_array = np.array(img)
    bike_data.append(img_array)

mono_data = []
for image in mono_images:
    img = Image.open(os.path.join(mono_folder, image))
    img_array = np.array(img)
    mono_data.append(img_array)

X = np.concatenate((scooter_data, bike_data, mono_data), axis=0)
y = np.concatenate((np.zeros(len(scooter_data)), np.ones(len(bike_data)),
                    np.full(len(mono_data), 2)))
```

Данные в данном случае представляют собой набор изображений трех классов транспортных средств: скутеров, мотоциклов и моноколес. Изображения были собраны из трех разных папок: Class1 содержит изображения скутеров, Class2 - изображения моноколес, а Class3 - изображения мотоциклов.

Для анализа и использования этих данных они были загружены в память с помощью библиотеки Pillow и преобразованы в массивы numpy, используя функцию np.array().

Для каждого изображения был создан соответствующий класс в виде числа: 0 - для скутеров, 1 - для мотоциклов и 2 - для моноколес. Классы были объединены в один массив y, а массивы изображений были объединены в массив X.

In [ ]:

```
```

Для начала можно посмотреть на размерность данных:

In [4]:

```
print('Размерность матрицы признаков X:', X.shape)
print('Размерность вектора целевой переменной y:', y.shape)
```

```
Размерность матрицы признаков X: (5100, 256, 256, 3)
Размерность вектора целевой переменной y: (5100,)
```

Проверим, какие размеры имеют изображения в нашем наборе данных:

In [5]:

```
image_shapes = []
for img in X:
    image_shapes.append(img.shape)

print(set(image_shapes))
```

```
{(256, 256, 3)}
```

Вывод: {(256, 256, 3)} - все изображения имеют одинаковый размер 256x256 и 3 канала цвета (RGB).

Распределение классов:

In [6]:

```
unique, counts = np.unique(y, return_counts=True)
print(dict(zip(unique, counts)))
```

```
{0.0: 1700, 1.0: 1700, 2.0: 1700}
```

Вывод: {0.0: 1700, 1.0: 1700, 2.0: 1700} - классы сбалансированы, по 1700 изображений на каждый класс.

Отображение нескольких изображений:

In [7]:

```
fig, axs = plt.subplots(3, 3, figsize=(10,10))
fig.subplots_adjust(hspace=.5, wspace=.5)
axs = axs.ravel()

for i in np.arange(0, 9):
    index = np.random.randint(0, X.shape[0])
    axs[i].imshow(X[index])
    axs[i].set_title(y[index])
    axs[i].axis('off')
```

Вывод: изображения в наборе данных имеют различную яркость, цветовую гамму и угол обзора.

Проверка наличия пустых значений:

In [8]:

```
np.isnan(X).sum()
```

Out[8]:

```
0
```

Вывод: 0 - в нашем наборе данных нет пустых значений.

Статистические характеристики:

In [9]:

```
print('Минимальное значение пикселя: ', X.min())
print('Максимальное значение пикселя: ', X.max())
print('Среднее значение пикселя: ', X.mean())
print('Стандартное отклонение: ', X.std())
```

```
Минимальное значение пикселя:  0
Максимальное значение пикселя:  255
Среднее значение пикселя:  100.33852713690864
Стандартное отклонение:  55.188647875401735
```

Вывод: значение пикселей находится в диапазоне от 0 до 255, что является типичным диапазоном значений для изображений в формате RGB. Среднее значение пикселя и стандартное отклонение указывают на среднюю яркость изображений и их изменчивость.

На этом этапе предварительного анализа мы получили общее представление о нашем наборе данных. Мы убедились, что данные не содержат пустых значений, распределены сбалансированно по классам и имеют одинаковый размер. Мы также установили, что изображения имеют различную яркость, цветовую гамму и угол обзора. Эта информация может быть полезна для выбора подходящих методов обработки и анализа данных в будущем.Вывод: значение пикселей находится в диапазоне от 0 до 255, что является типичным диапазоном значений для изображений в формате RGB. Среднее значение пикселя и стандартное отклонение указывают на среднюю яркость изображений и их изменчивость.

На этом этапе предварительного анализа мы получили общее представление о нашем наборе данных. Мы убедились, что данные не содержат пустых значений, распределены сбалансированно по классам и имеют одинаковый размер. Мы также установили, что изображения имеют различную яркость, цветовую гамму и угол обзора. Эта информация может быть полезна для выбора подходящих методов обработки и анализа данных в будущем.

In [ ]:

```

```

Разделить выборку на обучающую и тестовую

In [10]:

```
X = X.reshape(X.shape[0], -1)
```

In [11]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Здесь мы выбрали test_size = 0.2, что означает, что мы используем 20% данных для тестирования нашей модели, а оставшиеся 80% для обучения.

In [69]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import classification_report, accuracy_score, precision_score, reca
import time
```

In [13]:

```
results = pd.DataFrame(columns=['Model', 'Time', 'Accuracy', 'Precision', 'Recall', 'F1-
```

Модель 1 - Логистическая регрессия

In [15]:

```
start = time.time()
clf = LogisticRegression(max_iter=1000)
clf.fit(X_train, y_train)
end = time.time()

time_taken = end - start
print(f'Время обучения модели {time_taken} секунд')
```

```
Время обучения модели 706.134596824646 секунд
```

In [16]:

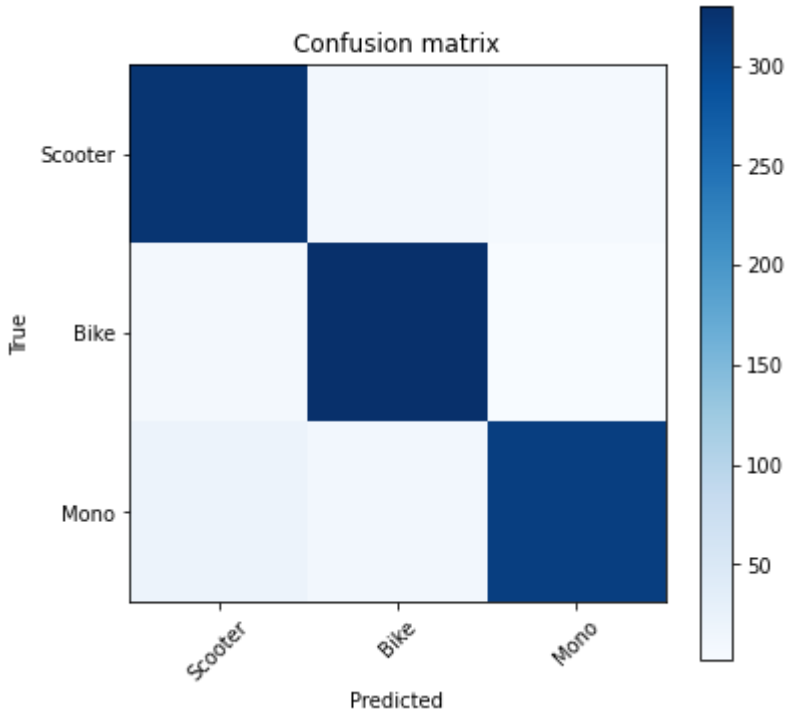
```
y_pred_clf = clf.predict(X_test)
```

In [17]:

```
lr_accuracy = accuracy_score(y_test, y_pred_clf)
lr_precision = precision_score(y_test, y_pred_clf, average='macro')
lr_recall = recall_score(y_test, y_pred_clf, average='macro')
lr_f1 = f1_score(y_test, y_pred_clf, average='macro')
results = results.append({'Model': 'Logistic Regression', 'Time': time_taken, 'Accuracy'
```

In [18]:

```
# Построение матрицы ошибок
labels = ['Scooter', 'Bike', 'Mono']
cm = confusion_matrix(y_test, y_pred_clf, labels=range(3))
plt.figure(figsize=(6, 6))
plt.imshow(cm, cmap='Blues')
plt.title('Confusion matrix')
plt.colorbar()
tick_marks = np.arange(len(labels))
plt.xticks(tick_marks, labels, rotation=45)
plt.yticks(tick_marks, labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```



In [19]:

```
print(classification_report(y_test, y_pred_clf))
```

```
              precision    recall  f1-score   support

         0.0       0.92      0.95      0.94       339
         1.0       0.94      0.97      0.96       340
         2.0       0.97      0.91      0.94       341

    accuracy                           0.95      1020
   macro avg       0.95      0.95      0.95      1020
weighted avg       0.95      0.95      0.95      1020
```

Модель 2 - метод опорных векторов

In [20]:

```
# Метод опорных векторов
start = time.time()
svm = SVC()
svm.fit(X_train, y_train)
end = time.time()

time_taken = end - start
print(f'Время обучения модели {time_taken} секунд')
```

```
Время обучения модели 408.98392271995544 секунд
```

In [21]:

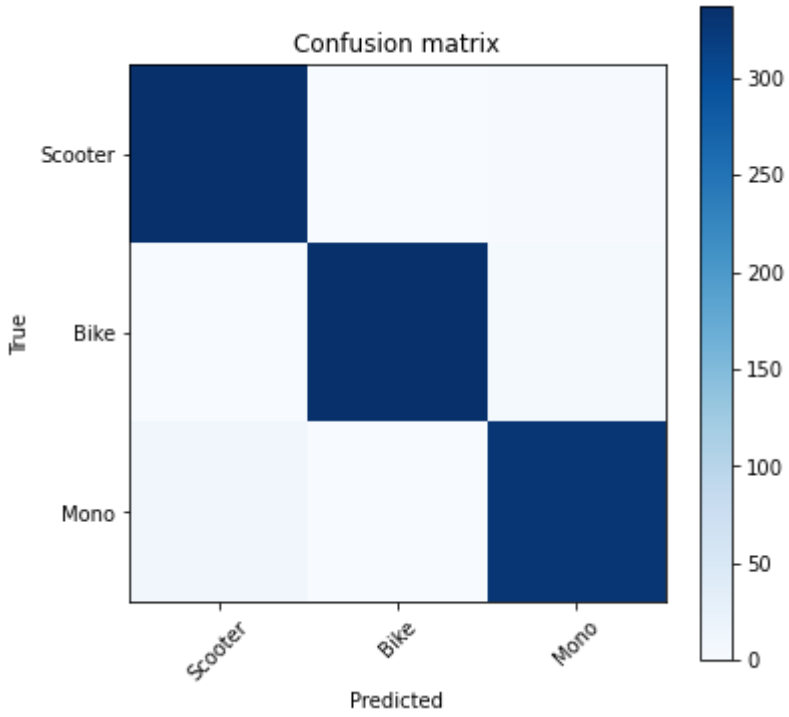
```
y_pred_svm = svm.predict(X_test)
```

In [22]:

```
svm_accuracy = accuracy_score(y_test, y_pred_svm)
svm_precision = precision_score(y_test, y_pred_svm, average='macro')
svm_recall = recall_score(y_test, y_pred_svm, average='macro')
svm_f1 = f1_score(y_test, y_pred_svm, average='macro')
results = results.append({'Model': 'SVM', 'Time': time_taken, 'Accuracy': svm_accuracy,
```

In [23]:

```
# Построение матрицы ошибок
labels = ['Scooter', 'Bike', 'Mono']
cm = confusion_matrix(y_test, y_pred_svm, labels=range(3))
plt.figure(figsize=(6, 6))
plt.imshow(cm, cmap='Blues')
plt.title('Confusion matrix')
plt.colorbar()
tick_marks = np.arange(len(labels))
plt.xticks(tick_marks, labels, rotation=45)
plt.yticks(tick_marks, labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```



In [24]:

```
print(classification_report(y_test, y_pred_svm,))
```

```
              precision    recall  f1-score   support

         0.0       0.97      0.99      0.98       339
         1.0       1.00      0.99      0.99       340
         2.0       0.98      0.96      0.97       341

    accuracy                           0.98      1020
   macro avg       0.98      0.98      0.98      1020
weighted avg       0.98      0.98      0.98      1020
```

Модель 3 - решающие деревья

In [25]:

```
start = time.time()
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
end=time.time()

time_taken = end - start
print(f'Время обучения модели {time_taken} секунд')
```

```
Время обучения модели 2155.4454333782196 секунд
```

In [27]:

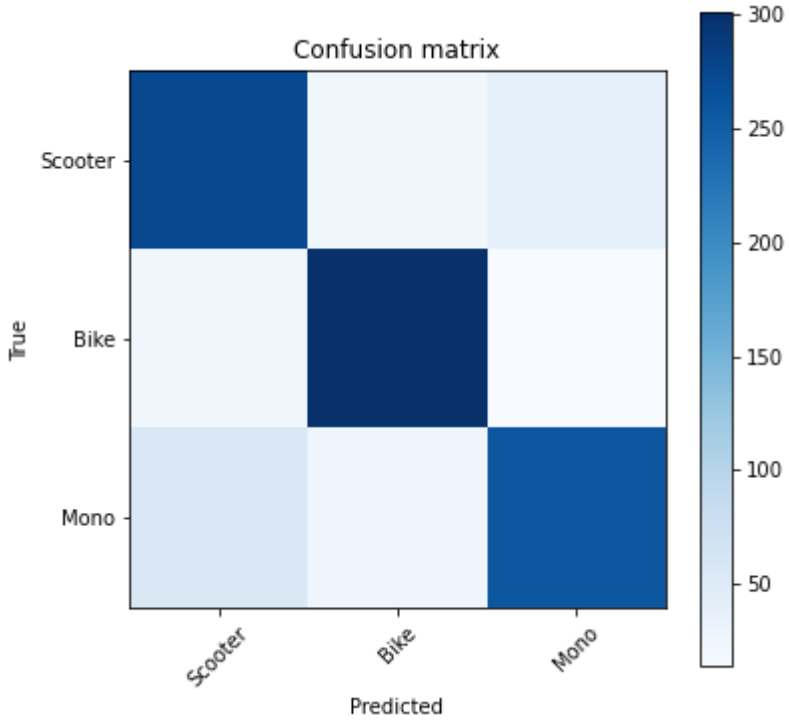
```
y_pred_dt = dt.predict(X_test)
```

In [28]:

```
dt_accuracy = accuracy_score(y_test, y_pred_dt)
dt_precision = precision_score(y_test, y_pred_dt, average='macro')
dt_recall = recall_score(y_test, y_pred_dt, average='macro')
dt_f1 = f1_score(y_test, y_pred_dt, average='macro')
results = results.append({'Model': 'Decision Trees', 'Time': time_taken, 'Accuracy': dt_
```

In [29]:

```
# Построение матрицы ошибок
labels = ['Scooter', 'Bike', 'Mono']
cm = confusion_matrix(y_test, y_pred_dt, labels=range(3))
plt.figure(figsize=(6, 6))
plt.imshow(cm, cmap='Blues')
plt.title('Confusion matrix')
plt.colorbar()
tick_marks = np.arange(len(labels))
plt.xticks(tick_marks, labels, rotation=45)
plt.yticks(tick_marks, labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```



In [30]:

```
print(classification_report(y_test, y_pred_dt,))
```

```
              precision    recall  f1-score   support

         0.0       0.77      0.81      0.79       339
         1.0       0.85      0.89      0.87       340
         2.0       0.83      0.76      0.79       341

    accuracy                           0.82      1020
   macro avg       0.82      0.82      0.82      1020
weighted avg       0.82      0.82      0.82      1020
```

Модель 4 - случайный лес

In [31]:

```
# Случайный лес
start = time.time()
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
end=time.time()

time_taken = end - start
print(f'Время обучения модели {time_taken} секунд')
```

```
Время обучения модели 187.539231300354 секунд
```

In [32]:

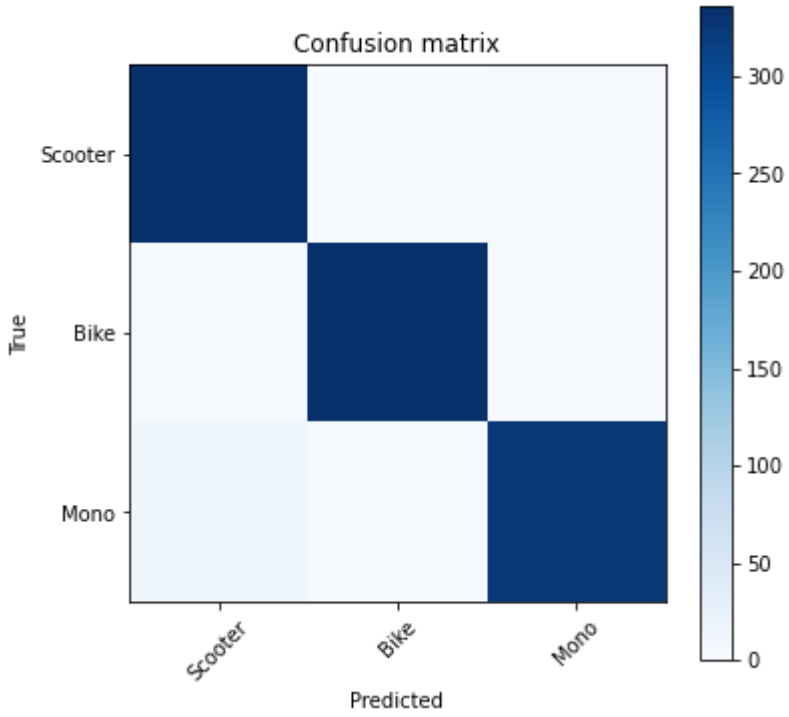
```
y_pred_rf = rf.predict(X_test)
```

In [34]:

```
rf_accuracy = accuracy_score(y_test, y_pred_rf)
rf_precision = precision_score(y_test, y_pred_rf, average='macro')
rf_recall = recall_score(y_test, y_pred_rf, average='macro')
rf_f1 = f1_score(y_test, y_pred_rf, average='macro')
results = results.append({'Model': 'Random Forest', 'Time': time_taken, 'Accuracy': rf_a
```

In [35]:

```
labels = ['Scooter', 'Bike', 'Mono']
cm = confusion_matrix(y_test, y_pred_rf, labels=range(3))
plt.figure(figsize=(6, 6))
plt.imshow(cm, cmap='Blues')
plt.title('Confusion matrix')
plt.colorbar()
tick_marks = np.arange(len(labels))
plt.xticks(tick_marks, labels, rotation=45)
plt.yticks(tick_marks, labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```



In [36]:

```
print(classification_report(y_test, y_pred_rf,))
```

```
              precision    recall  f1-score   support

         0.0       0.95      0.99      0.97       339
         1.0       0.99      0.98      0.99       340
         2.0       0.98      0.95      0.97       341

    accuracy                           0.98      1020
   macro avg       0.98      0.98      0.98      1020
weighted avg       0.98      0.98      0.98      1020
```

In [37]:

```
f1 = f1_score(y_test, y_pred_rf, average='weighted')
print("F1 score:", f1)
```

```
F1 score: 0.975482202032627
```

Модель 5 - К ближайших соседей

In [38]:

```
start = time.time()
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
end=time.time()

time_taken = end - start
print(f'Время обучения модели {time_taken} секунд')
```

```
Время обучения модели 0.0020017623901367188 секунд
```

In [39]:

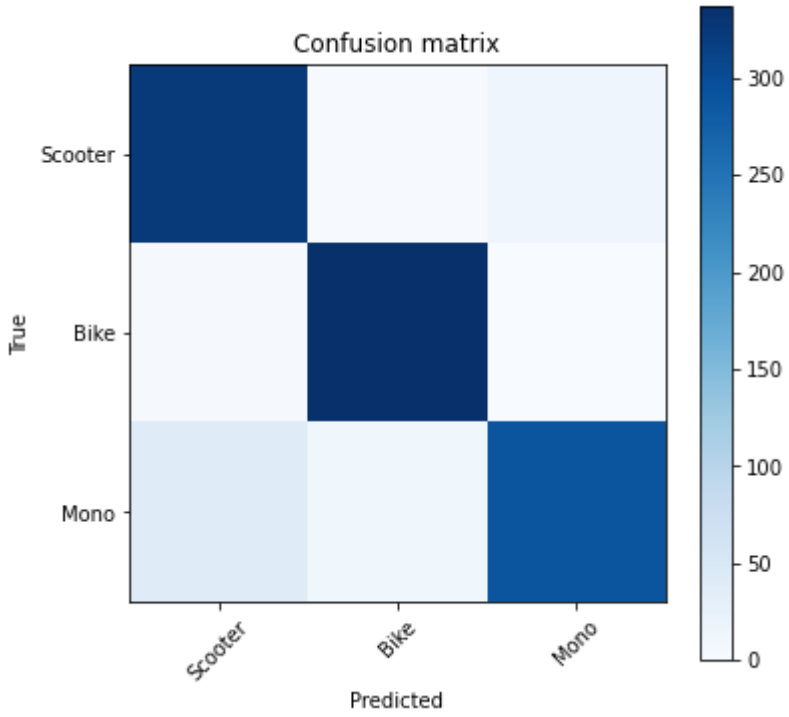
```
y_pred_knn = knn.predict(X_test)
```

In [40]:

```
knn_accuracy = accuracy_score(y_test, y_pred_knn)
knn_precision = precision_score(y_test, y_pred_knn, average='macro')
knn_recall = recall_score(y_test, y_pred_knn, average='macro')
knn_f1 = f1_score(y_test, y_pred_knn, average='macro')
results = results.append({'Model': 'KNN', 'Time': time_taken, 'Accuracy': knn_accuracy,
```

In [41]:

```
# Построение матрицы ошибок
labels = ['Scooter', 'Bike', 'Mono']
cm = confusion_matrix(y_test, y_pred_knn, labels=range(3))
plt.figure(figsize=(6, 6))
plt.imshow(cm, cmap='Blues')
plt.title('Confusion matrix')
plt.colorbar()
tick_marks = np.arange(len(labels))
plt.xticks(tick_marks, labels, rotation=45)
plt.yticks(tick_marks, labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```



In [42]:

```
print(classification_report(y_test, y_pred_knn,))
```

```
              precision    recall  f1-score   support

         0.0       0.88      0.95      0.92       339
         1.0       0.96      0.99      0.98       340
         2.0       0.95      0.85      0.90       341

    accuracy                           0.93      1020
   macro avg       0.93      0.93      0.93      1020
weighted avg       0.93      0.93      0.93      1020
```

Модель 6 - наивный байесовский классификатор

In [43]:

```
start = time.time()
nb = GaussianNB()
nb.fit(X_train, y_train)
end=time.time()

time_taken = end - start
print(f'Время обучения модели {time_taken} секунд')
```

```
Время обучения модели 8.53427243232727 секунд
```

In [44]:

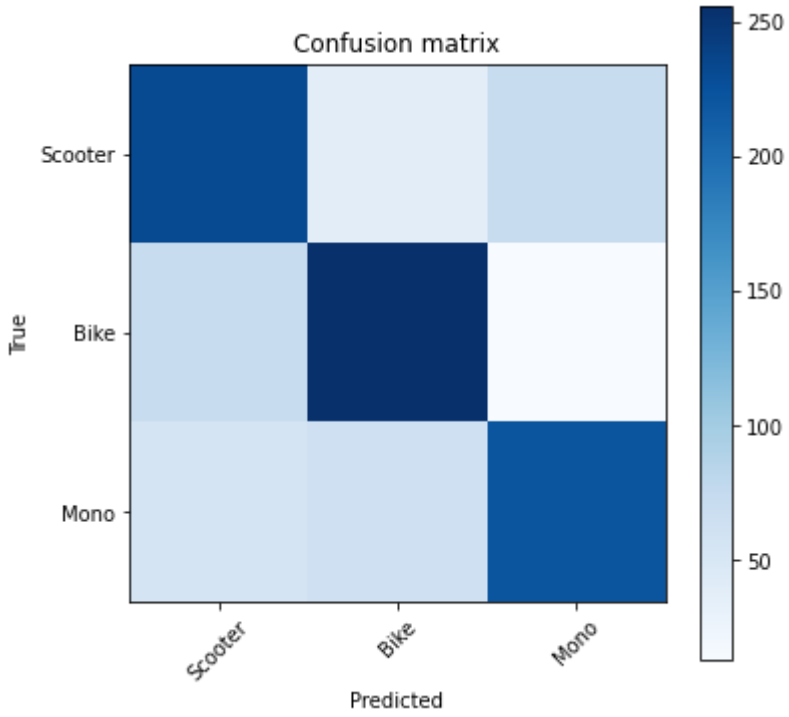
```
y_pred_nb = nb.predict(X_test)
```

In [45]:

```
nb_accuracy = accuracy_score(y_test, y_pred_nb)
nb_precision = precision_score(y_test, y_pred_nb, average='macro')
nb_recall = recall_score(y_test, y_pred_nb, average='macro')
nb_f1 = f1_score(y_test, y_pred_nb, average='macro')
results = results.append({'Model': 'Naive Bayes', 'Time': time_taken, 'Accuracy': nb_acc
```

In [46]:

```
# Построение матрицы ошибок
labels = ['Scooter', 'Bike', 'Mono']
cm = confusion_matrix(y_test, y_pred_nb, labels=range(3))
plt.figure(figsize=(6, 6))
plt.imshow(cm, cmap='Blues')
plt.title('Confusion matrix')
plt.colorbar()
tick_marks = np.arange(len(labels))
plt.xticks(tick_marks, labels, rotation=45)
plt.yticks(tick_marks, labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```



In [47]:

```
print(classification_report(y_test, y_pred_nb,))
```

```
              precision    recall  f1-score   support

         0.0       0.65      0.68      0.66       339
         1.0       0.72      0.75      0.73       340
         2.0       0.73      0.65      0.69       341

    accuracy                           0.69      1020
   macro avg       0.70      0.69      0.69      1020
weighted avg       0.70      0.69      0.69      1020
```

Модель 7 - Линейный дискриминантный анализ

In [48]:

```
start = time.time()
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)
end=time.time()

time_taken = end - start
print(f'Время обучения модели {time_taken} секунд')
```

```
Время обучения модели 965.9278516769409 секунд
```

In [49]:

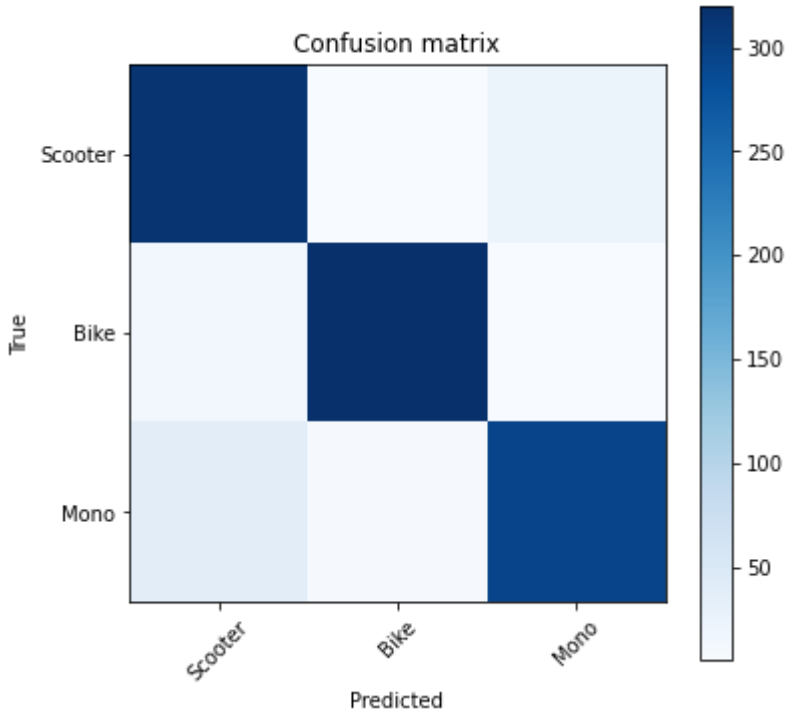
```
y_pred_lda = lda.predict(X_test)
```

In [50]:

```
lda_accuracy = accuracy_score(y_test, y_pred_lda)
lda_precision = precision_score(y_test, y_pred_lda, average='macro')
lda_recall = recall_score(y_test, y_pred_lda, average='macro')
lda_f1 = f1_score(y_test, y_pred_lda, average='macro')
results = results.append({'Model': 'LDA', 'Time': time_taken, 'Accuracy': lda_accuracy,
```

In [51]:

```
# Построение матрицы ошибок
labels = ['Scooter', 'Bike', 'Mono']
cm = confusion_matrix(y_test, y_pred_lda, labels=range(3))
plt.figure(figsize=(6, 6))
plt.imshow(cm, cmap='Blues')
plt.title('Confusion matrix')
plt.colorbar()
tick_marks = np.arange(len(labels))
plt.xticks(tick_marks, labels, rotation=45)
plt.yticks(tick_marks, labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```



In [52]:

```
print(classification_report(y_test, y_pred_lda,))
```

```
              precision    recall  f1-score   support

         0.0       0.86      0.92      0.89       339
         1.0       0.96      0.94      0.95       340
         2.0       0.92      0.87      0.89       341

    accuracy                           0.91      1020
   macro avg       0.91      0.91      0.91      1020
weighted avg       0.91      0.91      0.91      1020
```

Сравнение моделей

In [53]:

```
results
```

Out[53]:

	Model	Time	Accuracy	Precision	Recall	F1-score
0	Logistic Regression	706.134597	0.945098	0.946002	0.945138	0.945059
1	SVM	408.983923	0.981373	0.981576	0.981401	0.981383
2	Decision Trees	2155.445433	0.817647	0.818060	0.817695	0.817167
3	Random Forest	187.539231	0.975490	0.975853	0.975528	0.975486
4	KNN	0.002002	0.930392	0.932141	0.930495	0.929742
5	Naive Bayes	8.534272	0.694118	0.695770	0.694150	0.693879
6	LDA	965.927852	0.909804	0.911548	0.909861	0.910022

- SVM показал лучшие результаты по точности (accuracy), точности (precision), полноте (recall) и F1-мере (F1-score), чем остальные модели.
- KNN имеет очень низкое время обучения (Time) по сравнению с другими моделями.

Попробуем улучшить модель KNN

In [56]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
```

Определить модель KNN и словарь с гиперпараметрами, которые необходимо перебрать:

In [57]:

```
knn = KNeighborsClassifier()
params = {'n_neighbors': [3, 5, 7, 9, 11],
          'weights': ['uniform', 'distance']}
```

Создать объект GridSearchCV и указать модель, словарь гиперпараметров, количество фолдов для кросс-валидации (cv) и метрику для оценки (в данном случае используется accuracy):

In [58]:

```
grid = GridSearchCV(knn, params, cv=5, scoring='accuracy')
```

In [59]:

```
grid.fit(X_train, y_train)
```

Out[59]:

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [3, 5, 7, 9, 11],
                         'weights': ['uniform', 'distance']},
             scoring='accuracy')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Найти наилучшие гиперпараметры и соответствующую им точность:

In [60]:

```
best_params = grid.best_params_
best_score = grid.best_score_

print(f"Best parameters: {best_params}")
print(f"Best score: {best_score}")
```

```
Best parameters: {'n_neighbors': 3, 'weights': 'distance'}
Best score: 0.9438725490196077
```

In [61]:

```
y_pred = grid.predict(X_test)
```

In [63]:

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')

print(f"Accuracy: {accuracy}")
print(f"precision: {precision}")
print(f"recall: {recall}")
print(f"f1: {f1}")
```

```
Accuracy: 0.9441176470588235
precision: 0.9454443276787318
recall: 0.9442008030125778
f1: 0.9437482621453542
```

Показатели улучшились

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```