```
!pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: ml-dtypes~=0.2.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: tensorboard<2.16,>=2.15 in /usr/local/lib/python3.10/d
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /usr/local/lib/p
Requirement already satisfied: keras<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dis
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /usr/local/lib/python3
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/li
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/di
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dis
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-pack
```

```
import cv2
import os
import random
import tensorflow as tf
from shutil import copyfile
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.image as mpimg
from tensorflow.keras.optimizers.legacy import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Flatten
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator,img_to_array, load_im
from sklearn.metrics import accuracy_score, precision_score, recall_score

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

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

```
data_dirs = ['Tomato_YellowLeaf_Curl_Virus', 'Tomato_healthy', 'Tomato_Septoria_leaf_spot
source_path = "/content/gdrive/MyDrive/Colab Notebooks/dataset/"

DATA_CLASSES = ['Tomato_YellowLeaf_Curl_Virus', 'Tomato_healthy', 'Tomato_Septoria_leaf_s

for i, dir in enumerate(data_dirs):
    class_dir = os.path.join(source_path, dir)
    print(f'There are {len(os.listdir(class_dir))} images of {DATA_CLASSES[i]}')
```

```
There are 3209 images of Tomato_YellowLeaf_Curl_Virus
There are 1591 images of Tomato_healthy
There are 1771 images of Tomato_Septoria_leaf_spot
```

Бинарный классификатор

Создадим папки для тренировочных, валидационных, и тестированных данных

```
root_binary = 'binary_data'

train_dir = os.path.join(root_binary, 'train')
if not os.path.exists(train_dir):
    os.makedirs(os.path.join(root_binary, 'train'))

# Создание директории "validation", если она не существует
validation_dir = os.path.join(root_binary, 'validation')
if not os.path.exists(validation_dir):
    os.makedirs(os.path.join(root_binary, 'validation'))

# Создание директории "testing", если она не существует
testing_dir = os.path.join(root_binary, 'testing')
if not os.path.exists(testing_dir):
    os.makedirs(os.path.join(root_binary, 'testing'))

os.makedirs(os.path.join(root_binary, 'train/tomato_virus'))
os.makedirs(os.path.join(root_binary, 'train/tomato_healthy'))

os.makedirs(os.path.join(root_binary, 'validation/tomato_virus'))
os.makedirs(os.path.join(root_binary, 'validation/tomato_healthy'))
```

```
os.makedirs(os.path.join(root_binary, 'testing/tomato_virus'))
os.makedirs(os.path.join(root_binary, 'testing/tomato_healthy'))

!ls binary_data/validation
```

```
 tomato_healthy  tomato_virus
```

Разделим данные на тренировки, валидации, и тестирования

```
def split_data(SOURCE, TRAINING, VALIDATION, TESTING, TRAINING_SPLIT, VALIDATION_SPLIT, T
    filtered_files = []

    for file_name in os.listdir(SOURCE):
        file_path = os.path.join(SOURCE, file_name)

        if os.path.getsize(file_path):
            filtered_files.append(file_name)
        else:
            print('{} is zero length, so ignoring.'.format(file_name))

    # Split the files
    random_files = random.sample(filtered_files, len(filtered_files))
    split_train = int(TRAINING_SPLIT * len(filtered_files))
    split_val = split_train + int(VALIDATION_SPLIT * len(filtered_files))
    split_test = split_val + int(TESTING_SPLIT * len(filtered_files))

    training_files = random_files[:split_train]
    val_files = random_files[split_train:split_val]
    test_files = random_files[split_val:]

    # Copy the files
    for file_name in training_files:
        copyfile(os.path.join(SOURCE, file_name), os.path.join(TRAINING, file_name))

    for file_name in val_files:
        copyfile(os.path.join(SOURCE, file_name), os.path.join(VALIDATION, file_name))

    for file_name in test_files:
        copyfile(os.path.join(SOURCE, file_name), os.path.join(TESTING, file_name))


TOMATO_VIRUS_DIR = source_path + data_dirs[0]
TOMATO_HEALTHY_DIR = source_path + data_dirs[1]

TRAINING_DIR = "binary_data/train/"
VAL_DIR = "binary_data/validation/"
TEST_DIR = "binary_data/testing/"

TRAINING_TOMATO_VIRUS_DIR = os.path.join(TRAINING_DIR, "tomato_virus/")
VAL_TOMATO_VIRUS_DIR = os.path.join(VAL_DIR, "tomato_virus/")
```

```
TEST_TOMATO_VIRUS_DIR = os.path.join(TEST_DIR, "tomato_virus/")

TRAINING_TOMATO_HEALTHY_DIR = os.path.join(TRAINING_DIR, "tomato_healthy/")
VAL_TOMATO_HEALTHY_DIR = os.path.join(VAL_DIR, "tomato_healthy/")
TEST_TOMATO_HEALTHY_DIR = os.path.join(TEST_DIR, "tomato_healthy/")

# Empty directories in case you run this cell multiple times
if len(os.listdir(TRAINING_TOMATO_VIRUS_DIR)) > 0:
    for file in os.scandir(TRAINING_TOMATO_VIRUS_DIR):
        os.remove(file.path)
if len(os.listdir(TRAINING_TOMATO_HEALTHY_DIR)) > 0:
    for file in os.scandir(TRAINING_TOMATO_HEALTHY_DIR):
        os.remove(file.path)

if len(os.listdir(VAL_TOMATO_VIRUS_DIR)) > 0:
    for file in os.scandir(VAL_TOMATO_VIRUS_DIR):
        os.remove(file.path)
if len(os.listdir(VAL_TOMATO_HEALTHY_DIR)) > 0:
    for file in os.scandir(VAL_TOMATO_HEALTHY_DIR):
        os.remove(file.path)

if len(os.listdir(TEST_TOMATO_VIRUS_DIR)) > 0:
    for file in os.scandir(TEST_TOMATO_VIRUS_DIR):
        os.remove(file.path)
if len(os.listdir(TEST_TOMATO_HEALTHY_DIR)) > 0:
    for file in os.scandir(TEST_TOMATO_HEALTHY_DIR):
        os.remove(file.path)


# Define proportion of images used for training
split_train_size = .7
split_val_size = .2
split_test_size = .1

# Run the function
# NOTE: Messages about zero length images should be printed out
split_data(TOMATO_VIRUS_DIR, TRAINING_TOMATO_VIRUS_DIR, VAL_TOMATO_VIRUS_DIR, TEST_TOMATO
           split_train_size, split_val_size, split_test_size)
split_data(TOMATO_HEALTHY_DIR, TRAINING_TOMATO_HEALTHY_DIR, VAL_TOMATO_HEALTHY_DIR, TEST_
           split_train_size, split_val_size, split_test_size)


# Check that the number of images matches the expected output
print(f"There are {len(os.listdir(TRAINING_TOMATO_VIRUS_DIR))} images of {DATA_CLASSES[0]
print(f"There are {len(os.listdir(TRAINING_TOMATO_HEALTHY_DIR))} images of {DATA_CLASSES[

print(f"\nThere are {len(os.listdir(VAL_TOMATO_VIRUS_DIR))} images of {DATA_CLASSES[0]} f
print(f"There are {len(os.listdir(VAL_TOMATO_HEALTHY_DIR))} images of {DATA_CLASSES[1]} f

print(f"\nThere are {len(os.listdir(TEST_TOMATO_VIRUS_DIR))} images of {DATA_CLASSES[0]}
print(f"There are {len(os.listdir(TEST_TOMATO_HEALTHY_DIR))} images of {DATA_CLASSES[1]}
```

```
svn-r6Yb5c is zero length, so ignoring.
There are 2245 images of Tomato_YellowLeaf_Curl_Virus for training
There are 1113 images of Tomato_healthy for training
```

```
There are 641 images of Tomato_YellowLeaf_Curl_Virus for validation
There are 318 images of Tomato_healthy for validation

There are 322 images of Tomato_YellowLeaf_Curl_Virus for testing
There are 160 images of Tomato_healthy for testing
```

Генерации данных для классификации

```
def train_val_testing_generators(TRAINING_DIR, VALIDATION_DIR, TESTING_DIR):
    # Instantiate the ImageDataGenerator class
    train_datagen = ImageDataGenerator(rescale = 1.0 / 255.,
                                       rotation_range = 40,
                                       width_shift_range = .2,
                                       height_shift_range = .2,
                                       shear_range = .2,
                                       zoom_range = .2,
                                       horizontal_flip = True,
                                       fill_mode = 'nearest')

    # Pass in the appropriate arguments to the flow_from_directory method
    train_generator = train_datagen.flow_from_directory(directory = TRAINING_DIR,
                                                        batch_size = 64,
                                                        class_mode = 'categorical',
                                                        target_size = (256, 256))

    # Instantiate the ImageDataGenerator class
    validation_datagen = ImageDataGenerator(rescale = 1.0 / 255.,
                                            rotation_range = 40,
                                            width_shift_range = .2,
                                            height_shift_range = .2,
                                            shear_range = .2,
                                            zoom_range = .2,
                                            horizontal_flip = True,
                                            fill_mode = 'nearest')

    # Pass in the appropriate arguments to the flow_from_directory method
    validation_generator = validation_datagen.flow_from_directory(directory = VALIDATION_D
                                                                  batch_size = 64,
                                                                  class_mode = 'categorical'
                                                                  target_size = (256, 256))

    # Instantiate the ImageDataGenerator class
    testing_datagen = ImageDataGenerator(rescale = 1.0 / 255.,
                                         rotation_range = 40,
                                         width_shift_range = .2,
                                         height_shift_range = .2,
                                         shear_range = .2,
                                         zoom_range = .2,
                                         horizontal_flip = True,
                                         fill_mode = 'nearest')

    # Pass in the appropriate arguments to the flow_from_directory method
    testing_generator = testing_datagen.flow_from_directory(directory = TESTING_DIR,
```

```
                                                    batch_size = 64,
                                                    class_mode = 'categorical'
                                                    target_size = (256, 256))


    return train_generator, validation_generator, testing_generator


# Test your generators
train_generator, validation_generator, testing_generator = train_val_testing_generators(T

     Found 3358 images belonging to 2 classes.
     Found 959 images belonging to 2 classes.
     Found 482 images belonging to 2 classes.
```

Построим модель

```
model = Sequential()

model.add(Conv2D(32, (3,3), activation='relu', input_shape=(256, 256, 3)))
model.add(MaxPooling2D(2,2))

model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D(2,2))

model.add(Conv2D(128, (3,3), activation='relu'))
model.add(MaxPooling2D(2,2))

model.add(Conv2D(128, (3,3), activation='relu'))
model.add(MaxPooling2D(2,2))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))


model.summary()

     Model: "sequential_2"
     _________________________________________________________________
      Layer (type)                 Output Shape              Param #
     =================================================================
      conv2d_4 (Conv2D)            (None, 254, 254, 32)      896

      max_pooling2d_4 (MaxPoolin   (None, 127, 127, 32)      0
      g2D)

      conv2d_5 (Conv2D)            (None, 125, 125, 64)      18496

      max_pooling2d_5 (MaxPoolin   (None, 62, 62, 64)        0
      g2D)

      conv2d_6 (Conv2D)            (None, 60, 60, 128)       73856
```

```
 max_pooling2d_6 (MaxPoolin  (None, 30, 30, 128)       0
 g2D)

 conv2d_7 (Conv2D)           (None, 28, 28, 128)       147584

 max_pooling2d_7 (MaxPoolin  (None, 14, 14, 128)       0
 g2D)

 flatten_2 (Flatten)         (None, 25088)             0

 dense_4 (Dense)             (None, 128)               3211392

 dropout_2 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 2)                 258

=================================================================
Total params: 3452482 (13.17 MB)
Trainable params: 3452482 (13.17 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.compile(optimizer=Adam(),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Тренировать модель

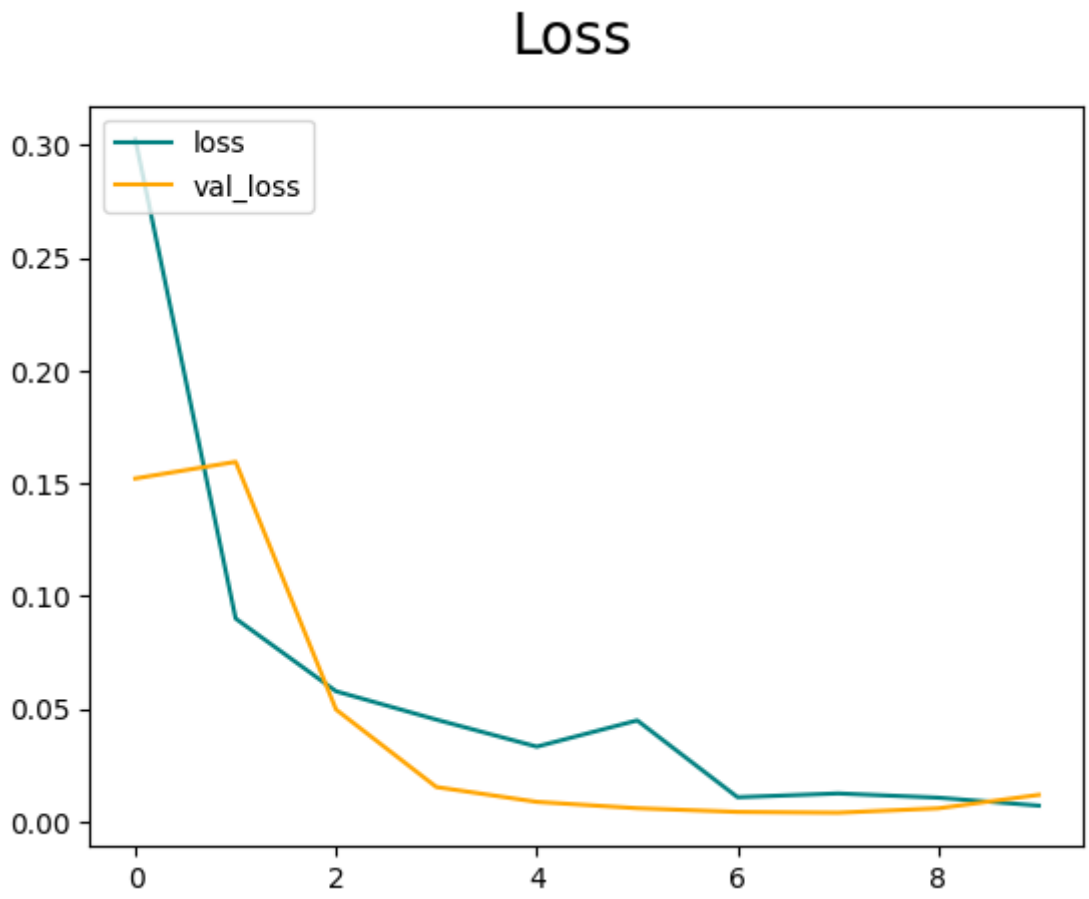
```
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator,
    verbose=1
)
```

```
 Epoch 1/10
 53/53 [==============================] - 85s 2s/step - loss: 0.3025 - accuracy: 0.862
 Epoch 2/10
 53/53 [==============================] - 82s 2s/step - loss: 0.0900 - accuracy: 0.975
 Epoch 3/10
 53/53 [==============================] - 82s 2s/step - loss: 0.0577 - accuracy: 0.982
 Epoch 4/10
 53/53 [==============================] - 86s 2s/step - loss: 0.0451 - accuracy: 0.989
 Epoch 5/10
 53/53 [==============================] - 82s 2s/step - loss: 0.0332 - accuracy: 0.992
 Epoch 6/10
 53/53 [==============================] - 82s 2s/step - loss: 0.0448 - accuracy: 0.984
 Epoch 7/10
 53/53 [==============================] - 83s 2s/step - loss: 0.0107 - accuracy: 0.997
 Epoch 8/10
 53/53 [==============================] - 83s 2s/step - loss: 0.0124 - accuracy: 0.997
 Epoch 9/10
 53/53 [==============================] - 82s 2s/step - loss: 0.0105 - accuracy: 0.997
```

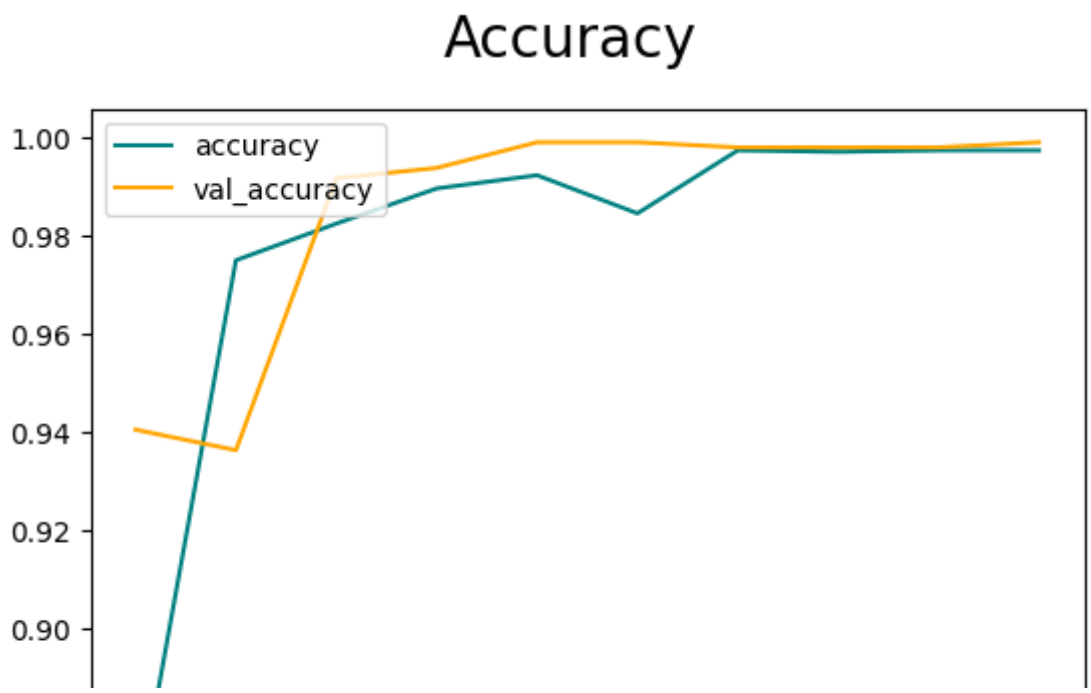
```
Epoch 10/10
53/53 [==============================] - 85s 2s/step - loss: 0.0070 - accuracy: 0.997
```

Plot Performance

```
fig = plt.figure()
plt.plot(history.history['loss'], color='teal', label='loss')
plt.plot(history.history['val_loss'], color='orange', label='val_loss')
fig.suptitle('Loss', fontsize=20)
plt.legend(loc='upper left')
plt.show()
```




```
fig = plt.figure()
plt.plot(history.history['accuracy'], color='teal', label='accuracy')
plt.plot(history.history['val_accuracy'], color='orange', label='val_accuracy')
fig.suptitle('Accuracy', fontsize=20)
plt.legend(loc='upper left')
plt.show()
```



Evaluate with test data

0.86

```
from tensorflow.keras.metrics import Precision, Recall, BinaryAccuracy


pre = Precision()
re = Recall()
acc = BinaryAccuracy()


final_predictions = model.predict(testing_generator)
```

```
8/8 [==============================] - 29s 3s/step
```

```
true_labels = np.squeeze(testing_generator.classes)


binary_preds = np.argmax(final_predictions, axis=1)


pre.update_state(true_labels, binary_preds)
re.update_state(true_labels, binary_preds)
acc.update_state(true_labels, binary_preds)
```

```
<tf.Variable 'UnreadVariable' shape=() dtype=float32, numpy=482.0>
```

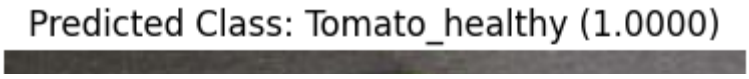
```
print(f'Precision: {pre.result().numpy()}, Recall: {re.result().numpy()}, Accuracy: {acc.
```

```
Precision: 0.6635220050811768, Recall: 0.6552795171737671, Accuracy: 0.5477178692817
```

Получились не слишком хорошие характеристики, скорее всего из-за неравномерности датасета

```
# Function to load and preprocess an image for prediction
def load_and_preprocess_image(image_path):
    img = load_img(image_path, target_size=(256, 256))
    img_array = img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0
    return img_array


# Path to the image you want to predict
image_path0 = '/content/gdrive/MyDrive/Colab Notebooks/dataset/Tomato_YellowLeaf_Curl_Vir
image_path1 = '/content/gdrive/MyDrive/Colab Notebooks/dataset/Tomato_healthy/4f5dde42-a6


loaded_image = image_path0

# Load and preprocess the image
input_image = load_and_preprocess_image(loaded_image)

# Make a prediction
predictions = model.predict(input_image)

# Decode the predictions
predicted_class_index = np.argmax(predictions)
predicted_class_label = DATA_CLASSES[predicted_class_index]
predicted_probability = predictions[0][predicted_class_index]

# Display the image
img = mpimg.imread(loaded_image)
plt.imshow(img)
plt.axis('off')
plt.title(f'Predicted Class: {predicted_class_label} ({predicted_probability:.4f})')
plt.show()
```

```
1/1 [==============================] - 0s 53ms/step
```




```
loaded_image = image_path1

# Load and preprocess the image
input_image = load_and_preprocess_image(loaded_image)

# Make a prediction
predictions = model.predict(input_image)

# Decode the predictions
predicted_class_index = np.argmax(predictions)
predicted_class_label = DATA_CLASSES[predicted_class_index]
predicted_probability = predictions[0][predicted_class_index]

# Display the image
img = mpimg.imread(loaded_image)
plt.imshow(img)
plt.axis('off')
plt.title(f'Predicted Class: {predicted_class_label} ({predicted_probability:.4f})')
plt.show()
```

```
1/1 [==============================] - 0s 52ms/step
```

Модель выдает противоположные результаты

Многоклассовый классификатор

Model A (Without pre-trained model)

Создадим папку

```
root_binary = 'multi_class'

os.makedirs(os.path.join(root_binary, 'train'))
os.makedirs(os.path.join(root_binary, 'validation'))
os.makedirs(os.path.join(root_binary, 'testing'))

os.makedirs(os.path.join(root_binary, 'train/tomato_virus'))
os.makedirs(os.path.join(root_binary, 'train/tomato_healthy'))
os.makedirs(os.path.join(root_binary, 'train/tomato_spot'))

os.makedirs(os.path.join(root_binary, 'validation/tomato_virus'))
os.makedirs(os.path.join(root_binary, 'validation/tomato_healthy'))
os.makedirs(os.path.join(root_binary, 'validation/tomato_spot'))

os.makedirs(os.path.join(root_binary, 'testing/tomato_virus'))
os.makedirs(os.path.join(root_binary, 'testing/tomato_healthy'))
os.makedirs(os.path.join(root_binary, 'testing/tomato_spot'))
```

Разделим данных на данные для обучения, валидации, и тестирования

```
TOMATO_VIRUS_DIR = source_path + data_dirs[0]
TOMATO_HEALTHY_DIR = source_path + data_dirs[1]
TOMATO_SPOT_DIR = source_path + data_dirs[2]

TRAINING_DIR = "multi_class/train/"
VAL_DIR = "multi_class/validation/"
TEST_DIR = "multi_class/testing/"

TRAINING_TOMATO_VIRUS_DIR = os.path.join(TRAINING_DIR, "tomato_virus/")
VAL_TOMATO_VIRUS_DIR = os.path.join(VAL_DIR, "tomato_virus/")
TEST_TOMATO_VIRUS_DIR = os.path.join(TEST_DIR, "tomato_virus/")

TRAINING_TOMATO_HEALTHY_DIR = os.path.join(TRAINING_DIR, "tomato_healthy/")
VAL_TOMATO_HEALTHY_DIR = os.path.join(VAL_DIR, "tomato_healthy/")
TEST_TOMATO_HEALTHY_DIR = os.path.join(TEST_DIR, "tomato_healthy/")

TRAINING_TOMATO_SPOT_DIR = os.path.join(TRAINING_DIR, "tomato_spot/")
VAL_TOMATO_SPOT_DIR = os.path.join(VAL_DIR, "tomato_spot/")
TEST_TOMATO_SPOT_DIR = os.path.join(TEST_DIR, "tomato_spot/")

# Empty directories in case you run this cell multiple times
if len(os.listdir(TRAINING_TOMATO_VIRUS_DIR)) > 0:
```

```
    for file in os.scandir(TRAINING_TOMATO_VIRUS_DIR):
        os.remove(file.path)
if len(os.listdir(TRAINING_TOMATO_HEALTHY_DIR)) > 0:
    for file in os.scandir(TRAINING_TOMATO_HEALTHY_DIR):
        os.remove(file.path)
if len(os.listdir(TRAINING_TOMATO_SPOT_DIR)) > 0:
    for file in os.scandir(TRAINING_TOMATO_SPOT_DIR):
        os.remove(file.path)

if len(os.listdir(VAL_TOMATO_VIRUS_DIR)) > 0:
    for file in os.scandir(VAL_TOMATO_VIRUS_DIR):
        os.remove(file.path)
if len(os.listdir(VAL_TOMATO_HEALTHY_DIR)) > 0:
    for file in os.scandir(VAL_TOMATO_HEALTHY_DIR):
        os.remove(file.path)
if len(os.listdir(VAL_TOMATO_SPOT_DIR)) > 0:
    for file in os.scandir(VAL_TOMATO_SPOT_DIR):
        os.remove(file.path)

if len(os.listdir(TEST_TOMATO_VIRUS_DIR)) > 0:
    for file in os.scandir(TEST_TOMATO_VIRUS_DIR):
        os.remove(file.path)
if len(os.listdir(TEST_TOMATO_HEALTHY_DIR)) > 0:
    for file in os.scandir(TEST_TOMATO_HEALTHY_DIR):
        os.remove(file.path)
if len(os.listdir(TEST_TOMATO_SPOT_DIR)) > 0:
    for file in os.scandir(TEST_TOMATO_SPOT_DIR):
        os.remove(file.path)



# Define proportion of images used for training
split_train_size = .75
split_val_size = .2
split_test_size = .05

# Run the function
# NOTE: Messages about zero length images should be printed out
split_data(TOMATO_VIRUS_DIR, TRAINING_TOMATO_VIRUS_DIR, VAL_TOMATO_VIRUS_DIR, TEST_TOMATO
           split_train_size, split_val_size, split_test_size)
split_data(TOMATO_HEALTHY_DIR, TRAINING_TOMATO_HEALTHY_DIR, VAL_TOMATO_HEALTHY_DIR, TEST_
           split_train_size, split_val_size, split_test_size)
split_data(TOMATO_SPOT_DIR, TRAINING_TOMATO_SPOT_DIR, VAL_TOMATO_SPOT_DIR, TEST_TOMATO_SP
           split_train_size, split_val_size, split_test_size)


# Check that the number of images matches the expected output
print(f"There are {len(os.listdir(TRAINING_TOMATO_VIRUS_DIR))} images of {DATA_CLASSES[0]
print(f"There are {len(os.listdir(TRAINING_TOMATO_HEALTHY_DIR))} images of {DATA_CLASSES[
print(f"There are {len(os.listdir(TRAINING_TOMATO_SPOT_DIR))} images of {DATA_CLASSES[2]}

print(f"\nThere are {len(os.listdir(VAL_TOMATO_VIRUS_DIR))} images of {DATA_CLASSES[0]} f
print(f"There are {len(os.listdir(VAL_TOMATO_HEALTHY_DIR))} images of {DATA_CLASSES[1]} f
print(f"There are {len(os.listdir(VAL_TOMATO_SPOT_DIR))} images of {DATA_CLASSES[2]} for
```

```
print(f"\nThere are {len(os.listdir(TEST_TOMATO_VIRUS_DIR))} images of {DATA_CLASSES[0]}
print(f"There are {len(os.listdir(TEST_TOMATO_HEALTHY_DIR))} images of {DATA_CLASSES[1]}
print(f"There are {len(os.listdir(TEST_TOMATO_SPOT_DIR))} images of {DATA_CLASSES[2]} for
```

```
svn-r6Yb5c is zero length, so ignoring.
There are 2406 images of Tomato_YellowLeaf_Curl_Virus for training
There are 1193 images of Tomato_healthy for training
There are 1328 images of Tomato_Septoria_leaf_spot for training

There are 641 images of Tomato_YellowLeaf_Curl_Virus for validation
There are 318 images of Tomato_healthy for validation
There are 354 images of Tomato_Septoria_leaf_spot for validation

There are 161 images of Tomato_YellowLeaf_Curl_Virus for testing
There are 80 images of Tomato_healthy for testing
There are 89 images of Tomato_Septoria_leaf_spot for testing
```

Генерирование этих данных

```
# Test your generators
train_generator, validation_generator, testing_generator = train_val_testing_generators(T
```

```
Found 4927 images belonging to 3 classes.
Found 1313 images belonging to 3 classes.
Found 330 images belonging to 3 classes.
```

Построить модель

```
model = Sequential()

model.add(Conv2D(32, (3,3), activation='relu', input_shape=(256, 256, 3)))
model.add(MaxPooling2D(2,2))

model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D(2,2))

model.add(Conv2D(128, (3,3), activation='relu'))
model.add(MaxPooling2D(2,2))

model.add(Conv2D(256, (3,3), activation='relu'))
model.add(MaxPooling2D(2,2))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))


model.summary()
```

```
Model: "sequential"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 32)      896

 max_pooling2d (MaxPooling2  (None, 127, 127, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 125, 125, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 62, 62, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 30, 30, 128)       0
 g2D)

 conv2d_3 (Conv2D)           (None, 28, 28, 256)       295168

 max_pooling2d_3 (MaxPoolin  (None, 14, 14, 256)       0
 g2D)

 flatten (Flatten)           (None, 50176)             0

 dense (Dense)               (None, 128)               6422656

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 3)                 387

=================================================================
Total params: 6811459 (25.98 MB)
Trainable params: 6811459 (25.98 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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

Тренировки модели

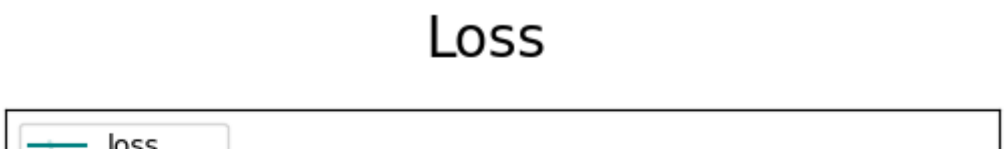
```
history = model.fit(
    train_generator,batch_size=32,
    epochs=15,
    validation_data=validation_generator,
    verbose=1
)
```

```
 Epoch 1/15
 77/77 [==============================] - 141s 2s/step - loss: 0.6621 - accuracy: 0.71
 Epoch 2/15
 77/77 [==============================] - 127s 2s/step - loss: 0.3960 - accuracy: 0.84
 Epoch 3/15
 77/77 [==============================] - 141s 2s/step - loss: 0.2134 - accuracy: 0.92
 Epoch 4/15
```

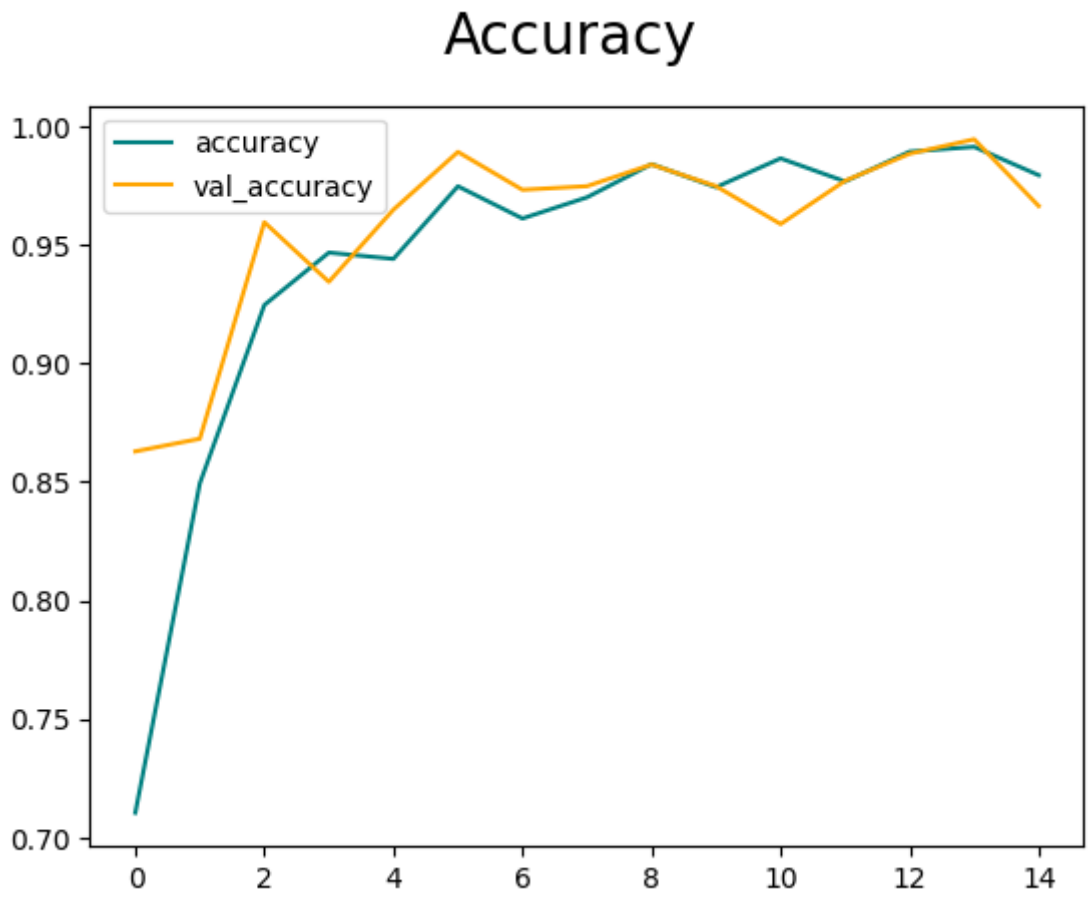
```
77/77 [==============================] - 121s 2s/step - loss: 0.1578 - accuracy: 0.94
Epoch 5/15
77/77 [==============================] - 122s 2s/step - loss: 0.1609 - accuracy: 0.94
Epoch 6/15
77/77 [==============================] - 124s 2s/step - loss: 0.0728 - accuracy: 0.97
Epoch 7/15
77/77 [==============================] - 139s 2s/step - loss: 0.1135 - accuracy: 0.96
Epoch 8/15
77/77 [==============================] - 138s 2s/step - loss: 0.0894 - accuracy: 0.97
Epoch 9/15
77/77 [==============================] - 138s 2s/step - loss: 0.0480 - accuracy: 0.98
Epoch 10/15
77/77 [==============================] - 138s 2s/step - loss: 0.0842 - accuracy: 0.97
Epoch 11/15
77/77 [==============================] - 124s 2s/step - loss: 0.0430 - accuracy: 0.98
Epoch 12/15
77/77 [==============================] - 124s 2s/step - loss: 0.0698 - accuracy: 0.97
Epoch 13/15
77/77 [==============================] - 124s 2s/step - loss: 0.0324 - accuracy: 0.98
Epoch 14/15
77/77 [==============================] - 124s 2s/step - loss: 0.0312 - accuracy: 0.99
Epoch 15/15
77/77 [==============================] - 138s 2s/step - loss: 0.0653 - accuracy: 0.97
```

Plot performance

```
fig = plt.figure()
plt.plot(history.history['loss'], color='teal', label='loss')
plt.plot(history.history['val_loss'], color='orange', label='val_loss')
fig.suptitle('Loss', fontsize=20)
plt.legend(loc='upper left')
plt.show()
```




```
fig = plt.figure()
plt.plot(history.history['accuracy'], color='teal', label='accuracy')
plt.plot(history.history['val_accuracy'], color='orange', label='val_accuracy')
fig.suptitle('Accuracy', fontsize=20)
plt.legend(loc='upper left')
plt.show()
```



Evaluate with test data

```
pre = Precision()
re = Recall()
acc = BinaryAccuracy()
```

```
final_predictions = model.predict(testing_generator)
```

```
     6/6 [==============================] - 6s 852ms/step
```

```
true_labels = np.squeeze(testing_generator.classes)
```

```
binary_preds = np.argmax(final_predictions, axis=1)
```

```
pre.update_state(true_labels, binary_preds)
re.update_state(true_labels, binary_preds)
acc.update_state(true_labels, binary_preds)
```

```
<tf.Variable 'UnreadVariable' shape=() dtype=float32, numpy=330.0>
```

```
print(f'Precision: {pre.result().numpy()}, Recall: {re.result().numpy()}, Accuracy: {acc.
```

```
Precision: 0.7627118825912476, Recall: 0.7200000286102295, Accuracy: 0.26969698071479
```

Точность получилась плохая

Evaluate with some data

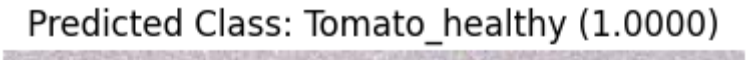
```
# Path to the image you want to predict
image_path0 = '/content/gdrive/MyDrive/Colab Notebooks/dataset/Tomato_YellowLeaf_Curl_Vir
image_path1 = '/content/gdrive/MyDrive/Colab Notebooks/dataset/Tomato_healthy/45b3e6ad-f7
image_path2 = '/content/gdrive/MyDrive/Colab Notebooks/dataset/Tomato_Septoria_leaf_spot/
```

```
loaded_image = image_path0

# Load and preprocess the image
input_image = load_and_preprocess_image(loaded_image)

# Make a prediction
predictions = model.predict(input_image)

# Decode the predictions
predicted_class_index = np.argmax(predictions)
predicted_class_label = DATA_CLASSES[predicted_class_index]
predicted_probability = predictions[0][predicted_class_index]

# Display the image
img = mpimg.imread(loaded_image)
plt.imshow(img)
plt.axis('off')
plt.title(f'Predicted Class: {predicted_class_label} ({predicted_probability:.4f})')
plt.show()
```

```
1/1 [==============================] - 0s 20ms/step
```


```
loaded_image = image_path2

# Load and preprocess the image
input_image = load_and_preprocess_image(loaded_image)

# Make a prediction
predictions = model.predict(input_image)

# Decode the predictions
predicted_class_index = np.argmax(predictions)
predicted_class_label = DATA_CLASSES[predicted_class_index]
predicted_probability = predictions[0][predicted_class_index]

# Display the image
img = mpimg.imread(loaded_image)
plt.imshow(img)
plt.axis('off')
plt.title(f'Predicted Class: {predicted_class_label} ({predicted_probability:.4f})')
plt.show()
```

```
1/1 [==============================] - 0s 52ms/step
```

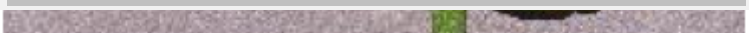


Модель B (with pre-trained model)

```
from tensorflow.keras.applications import ResNet50
```

```
# Load the pre-trained ResNet50 model without the top (fully connected) layers
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(256, 256, 3))

# Freeze the pre-trained layers so that they are not trainable
for layer in base_model.layers:
    layer.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/re
94765736/94765736 [==============================] - 3s 0us/step
```



```
# Test your generators
train_generator, validation_generator, testing_generator = train_val_testing_generators(T
```

```
Found 4927 images belonging to 3 classes.
Found 1313 images belonging to 3 classes.
Found 330 images belonging to 3 classes.
```

```
# Make the model
model = tf.keras.models.Sequential([
    base_model,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(3, activation='softmax')
])
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 8, 8, 2048)        23587712

 flatten_1 (Flatten)         (None, 131072)            0

 dense_2 (Dense)             (None, 512)               67109376

 dropout_1 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 3)                 1539

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

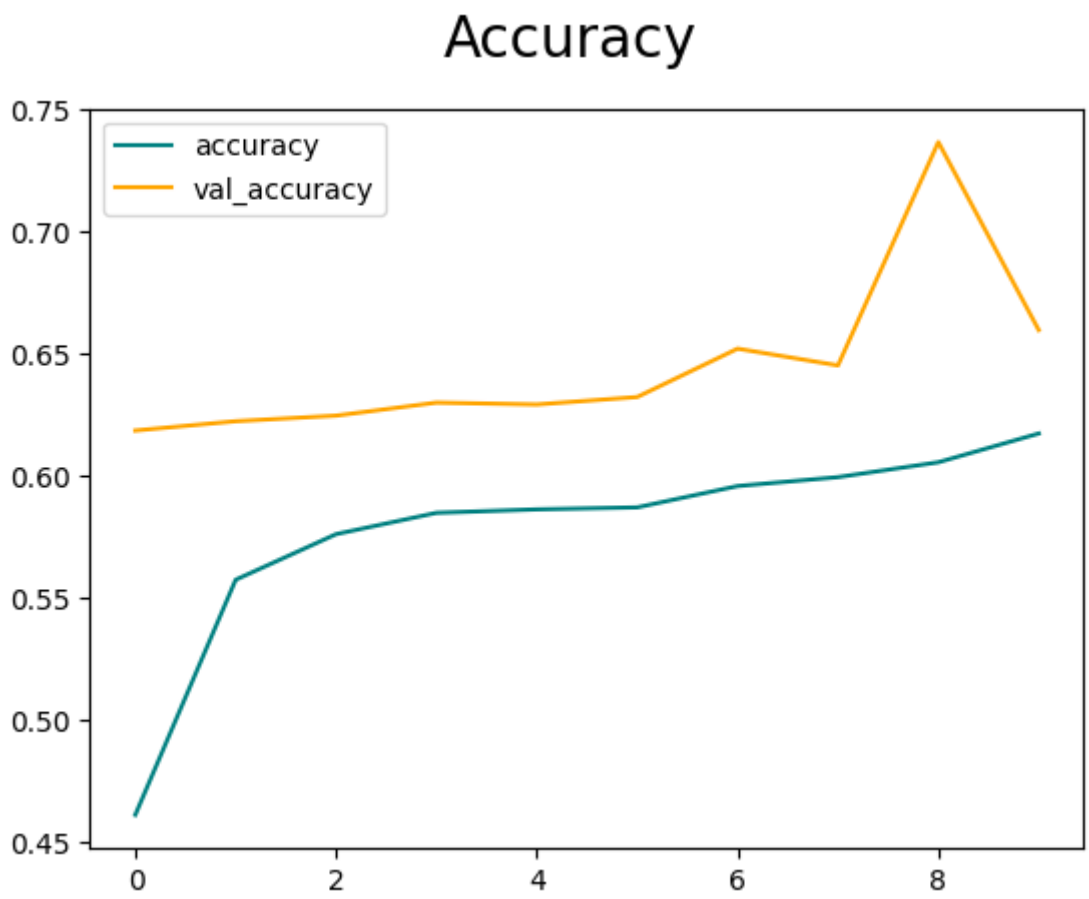
```
Total params: 90698627 (345.99 MB)
Trainable params: 67110915 (256.01 MB)
Non-trainable params: 23587712 (89.98 MB)
_________________________________________________________________
```

```
# Compile the model
model.compile(optimizer=Adam(),
              loss='categorical_crossentropy',
              metrics=['accuracy'])


# Train the model
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator,
    verbose=1
)
```

```
Epoch 1/10
77/77 [==============================] - 143s 2s/step - loss: 5.9224 - accuracy: 0.46
Epoch 2/10
77/77 [==============================] - 131s 2s/step - loss: 0.9709 - accuracy: 0.55
Epoch 3/10
77/77 [==============================] - 131s 2s/step - loss: 0.9359 - accuracy: 0.57
Epoch 4/10
77/77 [==============================] - 144s 2s/step - loss: 0.9191 - accuracy: 0.58
Epoch 5/10
77/77 [==============================] - 146s 2s/step - loss: 0.8742 - accuracy: 0.58
Epoch 6/10
77/77 [==============================] - 157s 2s/step - loss: 0.9392 - accuracy: 0.58
Epoch 7/10
77/77 [==============================] - 147s 2s/step - loss: 0.8625 - accuracy: 0.59
Epoch 8/10
77/77 [==============================] - 147s 2s/step - loss: 0.8194 - accuracy: 0.59
Epoch 9/10
77/77 [==============================] - 147s 2s/step - loss: 0.8295 - accuracy: 0.60
Epoch 10/10
77/77 [==============================] - 147s 2s/step - loss: 0.8035 - accuracy: 0.61
```

```
fig = plt.figure()
plt.plot(history.history['loss'], color='teal', label='loss')
plt.plot(history.history['val_loss'], color='orange', label='val_loss')
fig.suptitle('Loss', fontsize=20)
plt.legend(loc='upper left')
plt.show()
```


```
fig = plt.figure()
plt.plot(history.history['accuracy'], color='teal', label='accuracy')
plt.plot(history.history['val_accuracy'], color='orange', label='val_accuracy')
fig.suptitle('Accuracy', fontsize=20)
plt.legend(loc='upper left')
plt.show()
```




```
pre = Precision()
re = Recall()
acc = BinaryAccuracy()
```

```
final_predictions = model.predict(testing_generator)
```

```
     6/6 [==============================] - 14s 2s/step
```

```
true_labels = np.squeeze(testing_generator.classes)
```

```
binary_preds = np.argmax(final_predictions, axis=1)
```

```
pre.update_state(true_labels, binary_preds)
re.update_state(true_labels, binary_preds)
acc.update_state(true_labels, binary_preds)
```

```
     <tf.Variable 'UnreadVariable' shape=() dtype=float32, numpy=330.0>
```

```
print(f'Precision: {pre.result().numpy()}, Recall: {re.result().numpy()}, Accuracy: {acc.
```

```
     Precision: 0.7471697926521301, Recall: 0.7919999957084656, Accuracy: 0.2575757503509
```

Точность не улучшилась

```
# Path to the image you want to predict
image_path0 = '/content/gdrive/MyDrive/Colab Notebooks/dataset/Tomato_YellowLeaf_Curl_Vir
image_path1 = '/content/gdrive/MyDrive/Colab Notebooks/dataset/Tomato_healthy/45b3e6ad-f7
image_path2 = '/content/gdrive/MyDrive/Colab Notebooks/dataset/Tomato_Septoria_leaf_spot/


loaded_image = image_path1

# Load and preprocess the image
input_image = load_and_preprocess_image(loaded_image)

# Make a prediction
predictions = model.predict(input_image)

# Decode the predictions
predicted_class_index = np.argmax(predictions)
predicted_class_label = DATA_CLASSES[predicted_class_index]
predicted_probability = predictions[0][predicted_class_index]

# Display the image
img = mpimg.imread(loaded_image)
plt.imshow(img)
plt.axis('off')
plt.title(f'Predicted Class: {predicted_class_label} ({predicted_probability:.4f})')
plt.show()
```

```
1/1 [==============================] - 2s 2s/step
```

В целом модели показали себя не очень хорошо, скорее всего их-за неравномерности данных.