In [2]:

```
#LBL1

import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.utils import to_categorical
import cv2
```

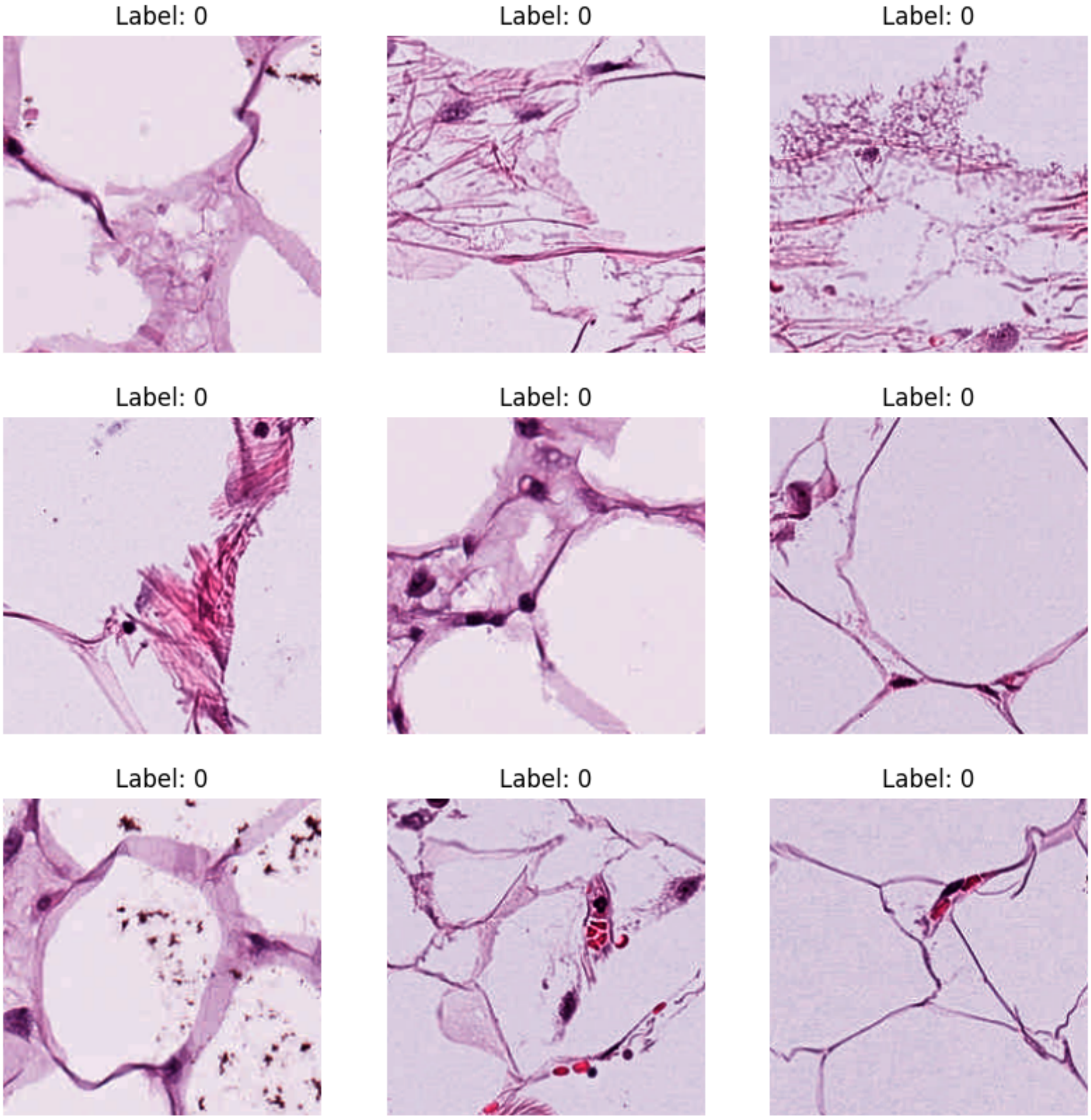
In [3]:

```
#LBL2

def explore_dataset(file_path):
    # Загрузка данных
    data = np.load(file_path)
    images = data['data']  # Извлечение изображений
    labels = data['labels']  # Извлечение меток

    print(f"Файл: {file_path}")
    print(f"Количество изображений: {images.shape[0]}")
    print(f"Размер каждого изображения: {images[0].shape}")
    print(f"Количество меток: {len(labels)}")
    print(f"Пример меток: {np.unique(labels)}")

    # Визуализация нескольких изображений
    plt.figure(figsize=(10, 10))
    for i in range(9):  # Покажем первые 9 изображений
        plt.subplot(3, 3, i + 1)
        plt.imshow(images[i])
        plt.title(f"Label: {labels[i]}")
        plt.axis('off')
    plt.show()
```

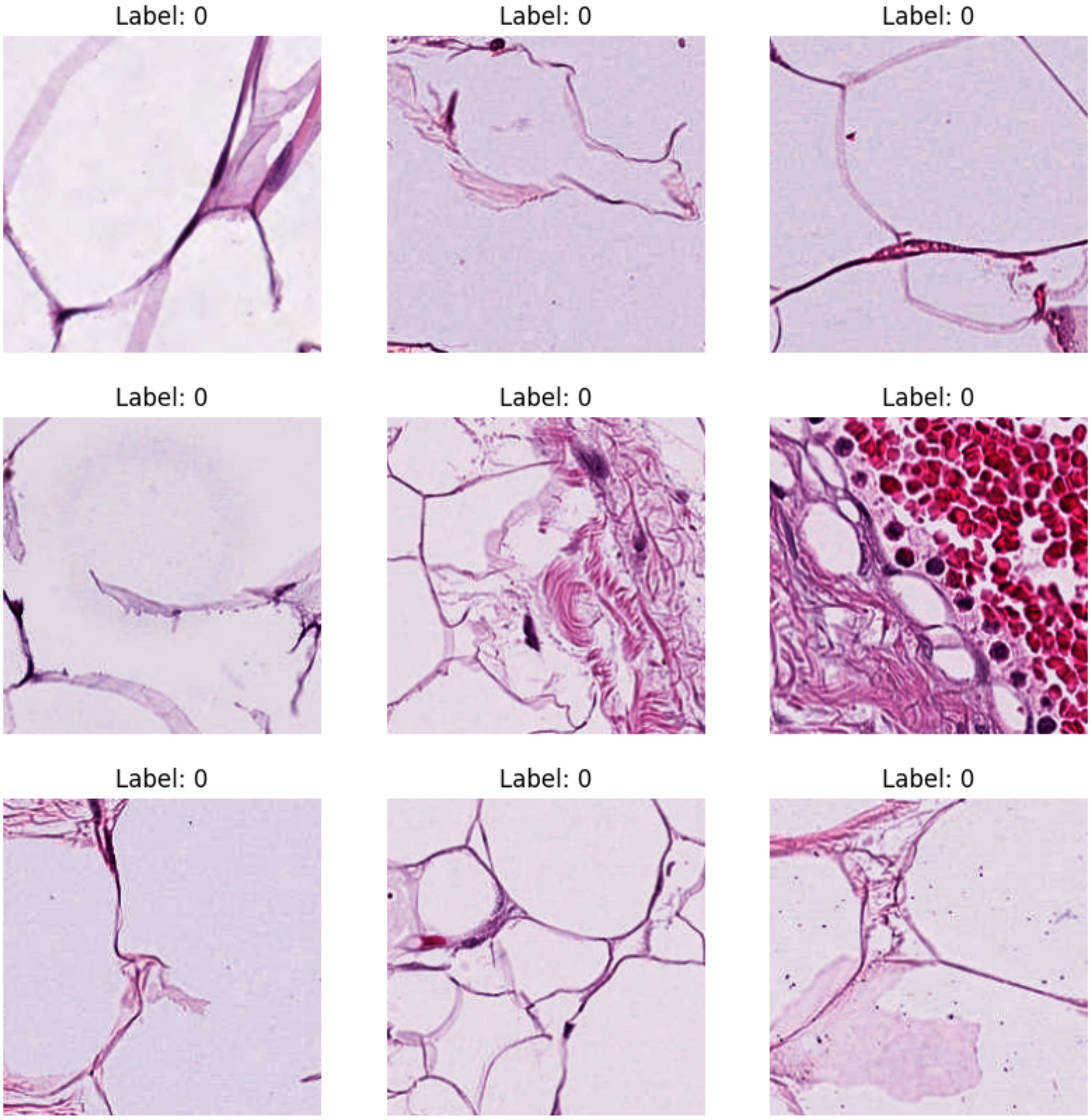
In [4]:

```
# Просмотр train и test данных
explore_dataset('train.npz')  #LBL2
explore_dataset('test.npz')  #LBL2
```

```
Файл: train.npz
Количество изображений: 18000
Размер каждого изображения: (224, 224, 3)
Количество меток: 18000
Пример меток: [0 1 2 3 4 5 6 7 8]
```



```
Файл: test.npz
Количество изображений: 4500
Размер каждого изображения: (224, 224, 3)
Количество меток: 4500
Пример меток: [0 1 2 3 4 5 6 7 8]
```



In [5]:
```
#LBL3

class Dataset:
    def __init__(self, name, resize_to=(64, 64)):  # Размер 64x64
        self.name = name
        self.is_loaded = False
        output = f'{name}.npz'
        print(f'Loading dataset {self.name} from npz.')
        np_obj = np.load(output)
        resized_images = [cv2.resize(img, resize_to) for img in np_obj['d
        self.images = np.array(resized_images, dtype=np.float32) / 255.0
        self.labels = np_obj['labels']
        self.n_files = self.images.shape[0]
        self.is_loaded = True
        print(f'Done. Dataset {self.name} consists of {self.n_files} imag

    def get_data_and_labels(self):
        return self.images, to_categorical(self.labels, num_classes=9)
```

In [6]:
```
#LBL4

class CNNModel:
    def __init__(self):
        self.model = Sequential([
            Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)
            BatchNormalization(),
            MaxPooling2D((2, 2)),
            Conv2D(64, (3, 3), activation='relu'),
            BatchNormalization(),
            MaxPooling2D((2, 2)),
            Conv2D(128, (3, 3), activation='relu'),
            BatchNormalization(),
            MaxPooling2D((2, 2)),
            Flatten(),
            Dense(256, activation='relu'),
            Dropout(0.5),
            Dense(9, activation='softmax')
        ])
        self.model.compile(optimizer=Adam(learning_rate=0.0001),
                           loss='categorical_crossentropy',
                           metrics=['accuracy'])

    def train(self, X_train, y_train, X_val, y_val, epochs=20, batch_size
        early_stopping = EarlyStopping(monitor='val_accuracy', patience=5
        lr_reduction = ReduceLROnPlateau(monitor='val_loss', patience=3,

        print("Начало обучения модели CNN...")
        history = self.model.fit(X_train, y_train,
                                 validation_data=(X_val, y_val),
                                 epochs=epochs,
                                 batch_size=batch_size,
                                 callbacks=[early_stopping, lr_reduction]
        print("Обучение завершено.")
        return history

    def save(self, name: str):
        self.model.save(f'{name}.h5')
        print(f'Model saved at: {name}.h5')

    def load(self, name: str):
        self.model = tf.keras.models.load_model(f'{name}.h5')
        print(f'Model loaded from: {name}.h5')

    def evaluate(self, X_test, y_test):
        test_loss, test_accuracy = self.model.evaluate(X_test, y_test)
        print(f'Test Accuracy: {test_accuracy:.4f}')
        return test_accuracy
```

In [7]:

```
#LBL5
d_train = Dataset('train')
d_test = Dataset('test')

X_train, y_train = d_train.get_data_and_labels()
X_test, y_test = d_test.get_data_and_labels()

#LBL6
X_train_split, X_val, y_train_split, y_val = train_test_split(X_train, y_

#LBL7
model = CNNModel()
history = model.train(X_train_split, y_train_split, X_val, y_val, epochs=

#LBL8
```

```python
model.save('cnn_model')

#LBL9
test_accuracy = model.evaluate(X_test, y_test)
```

```
Loading dataset train from npz.
Done. Dataset train consists of 18000 images with size (64, 64).
Loading dataset test from npz.
Done. Dataset test consists of 4500 images with size (64, 64).
```

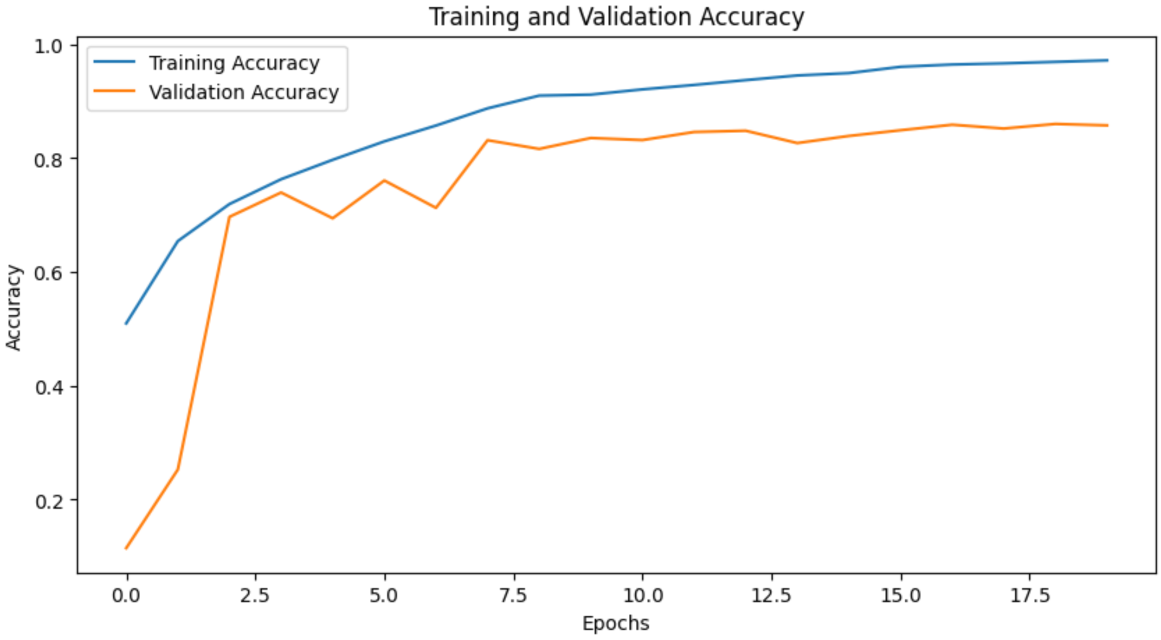
```
/home/fantom/jupyter_env/lib/python3.12/site-packages/keras/src/layers/con
volutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`in
put_dim` argument to a layer. When using Sequential models, prefer using a
n `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2024-12-02 23:51:39.196954: E external/local_xla/xla/stream_executor/cuda/
cuda_driver.cc:152] failed call to cuInit: INTERNAL: CUDA error: Failed ca
ll to cuInit: UNKNOWN ERROR (303)
Начало обучения модели CNN...
Epoch 1/20
```

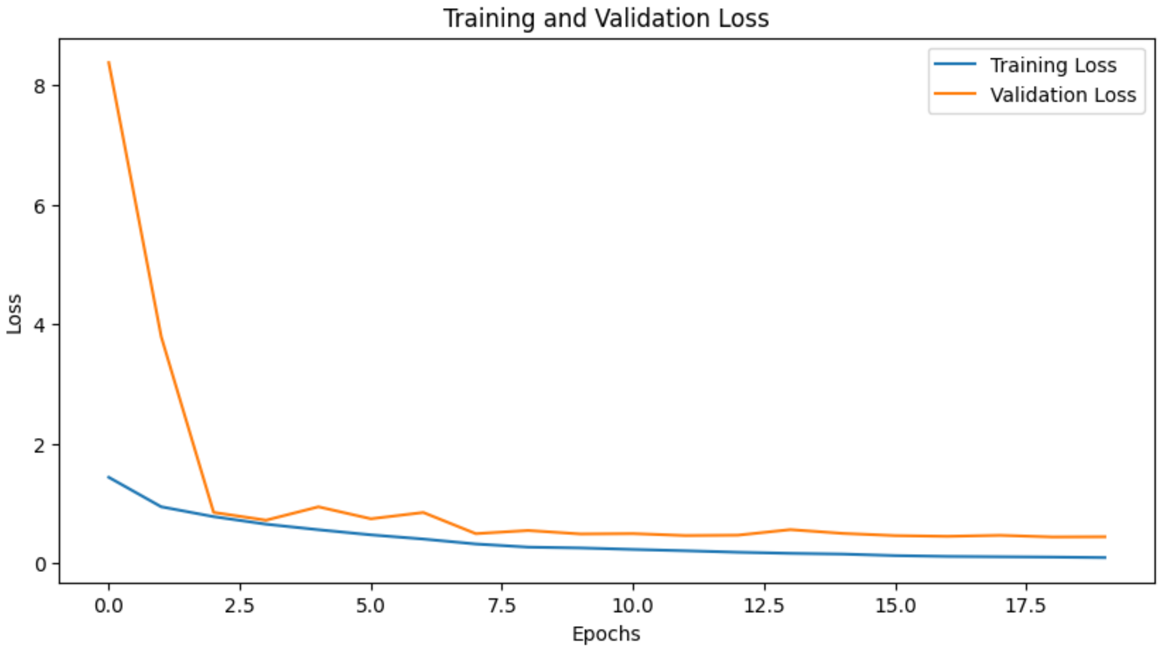
```
2024-12-02 23:51:39.478198: W external/local_xla/xla/tsl/framework/cpu_all
ocator_impl.cc:83] Allocation of 707788800 exceeds 10% of free system memo
ry.
```

```
225/225 ━━━━━━━━━━━━━━━━━━━━ 68s 291ms/step - accuracy: 0.4280 - loss: 1.9
289 - val_accuracy: 0.1147 - val_loss: 8.3743 - learning_rate: 1.0000e-04
Epoch 2/20
225/225 ━━━━━━━━━━━━━━━━━━━━ 63s 278ms/step - accuracy: 0.6386 - loss: 0.9
795 - val_accuracy: 0.2528 - val_loss: 3.7920 - learning_rate: 1.0000e-04
Epoch 3/20
225/225 ━━━━━━━━━━━━━━━━━━━━ 67s 298ms/step - accuracy: 0.7138 - loss: 0.7
905 - val_accuracy: 0.6969 - val_loss: 0.8527 - learning_rate: 1.0000e-04
Epoch 4/20
225/225 ━━━━━━━━━━━━━━━━━━━━ 71s 314ms/step - accuracy: 0.7615 - loss: 0.6
607 - val_accuracy: 0.7397 - val_loss: 0.7220 - learning_rate: 1.0000e-04
Epoch 5/20
225/225 ━━━━━━━━━━━━━━━━━━━━ 63s 280ms/step - accuracy: 0.7969 - loss: 0.5
547 - val_accuracy: 0.6944 - val_loss: 0.9458 - learning_rate: 1.0000e-04
Epoch 6/20
225/225 ━━━━━━━━━━━━━━━━━━━━ 62s 275ms/step - accuracy: 0.8368 - loss: 0.4
757 - val_accuracy: 0.7608 - val_loss: 0.7455 - learning_rate: 1.0000e-04
Epoch 7/20
225/225 ━━━━━━━━━━━━━━━━━━━━ 66s 294ms/step - accuracy: 0.8571 - loss: 0.4
091 - val_accuracy: 0.7128 - val_loss: 0.8511 - learning_rate: 1.0000e-04
Epoch 8/20
225/225 ━━━━━━━━━━━━━━━━━━━━ 70s 310ms/step - accuracy: 0.8855 - loss: 0.3
307 - val_accuracy: 0.8317 - val_loss: 0.4972 - learning_rate: 5.0000e-05
Epoch 9/20
225/225 ━━━━━━━━━━━━━━━━━━━━ 63s 280ms/step - accuracy: 0.9127 - loss: 0.2
670 - val_accuracy: 0.8164 - val_loss: 0.5499 - learning_rate: 5.0000e-05
Epoch 10/20
225/225 ━━━━━━━━━━━━━━━━━━━━ 63s 280ms/step - accuracy: 0.9152 - loss: 0.2
540 - val_accuracy: 0.8356 - val_loss: 0.4921 - learning_rate: 5.0000e-05
Epoch 11/20
225/225 ━━━━━━━━━━━━━━━━━━━━ 63s 278ms/step - accuracy: 0.9222 - loss: 0.2
337 - val_accuracy: 0.8319 - val_loss: 0.4977 - learning_rate: 5.0000e-05
Epoch 12/20
225/225 ━━━━━━━━━━━━━━━━━━━━ 63s 280ms/step - accuracy: 0.9317 - loss: 0.2
061 - val_accuracy: 0.8461 - val_loss: 0.4650 - learning_rate: 5.0000e-05
Epoch 13/20
225/225 ━━━━━━━━━━━━━━━━━━━━ 63s 280ms/step - accuracy: 0.9412 - loss: 0.1
841 - val_accuracy: 0.8483 - val_loss: 0.4714 - learning_rate: 5.0000e-05
Epoch 14/20
225/225 ━━━━━━━━━━━━━━━━━━━━ 63s 278ms/step - accuracy: 0.9470 - loss: 0.1
652 - val_accuracy: 0.8267 - val_loss: 0.5644 - learning_rate: 5.0000e-05
Epoch 15/20
225/225 ━━━━━━━━━━━━━━━━━━━━ 62s 276ms/step - accuracy: 0.9471 - loss: 0.1
575 - val_accuracy: 0.8392 - val_loss: 0.5014 - learning_rate: 5.0000e-05
Epoch 16/20
225/225 ━━━━━━━━━━━━━━━━━━━━ 63s 280ms/step - accuracy: 0.9573 - loss: 0.1
337 - val_accuracy: 0.8492 - val_loss: 0.4640 - learning_rate: 2.5000e-05
Epoch 17/20
225/225 ━━━━━━━━━━━━━━━━━━━━ 63s 278ms/step - accuracy: 0.9665 - loss: 0.1
166 - val_accuracy: 0.8589 - val_loss: 0.4521 - learning_rate: 2.5000e-05
Epoch 18/20
225/225 ━━━━━━━━━━━━━━━━━━━━ 63s 281ms/step - accuracy: 0.9657 - loss: 0.1
091 - val_accuracy: 0.8522 - val_loss: 0.4687 - learning_rate: 2.5000e-05
Epoch 19/20
225/225 ━━━━━━━━━━━━━━━━━━━━ 63s 279ms/step - accuracy: 0.9689 - loss: 0.1
044 - val_accuracy: 0.8603 - val_loss: 0.4417 - learning_rate: 2.5000e-05
Epoch 20/20
225/225 ━━━━━━━━━━━━━━━━━━━━ 63s 280ms/step - accuracy: 0.9701 - loss: 0.1
051 - val_accuracy: 0.8578 - val_loss: 0.4446 - learning_rate: 2.5000e-05
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()`
or `keras.saving.save_model(model)`. This file format is considered legacy
. We recommend using instead the native Keras format, e.g. `model.save('my
_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Обучение завершено.
Model saved at: cnn_model.h5
141/141 ━━━━━━━━━━━━━━━━━━━━ 4s 29ms/step - accuracy: 0.9101 - loss: 0.274
3
Test Accuracy: 0.8591
```

```
In [8]: #LBL10
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




Training and Validation Loss
Training Loss
Validation Loss
Loss
8
6
4
2
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs

```
In [ ]:
```