Технології машинного навчання, лаб 1

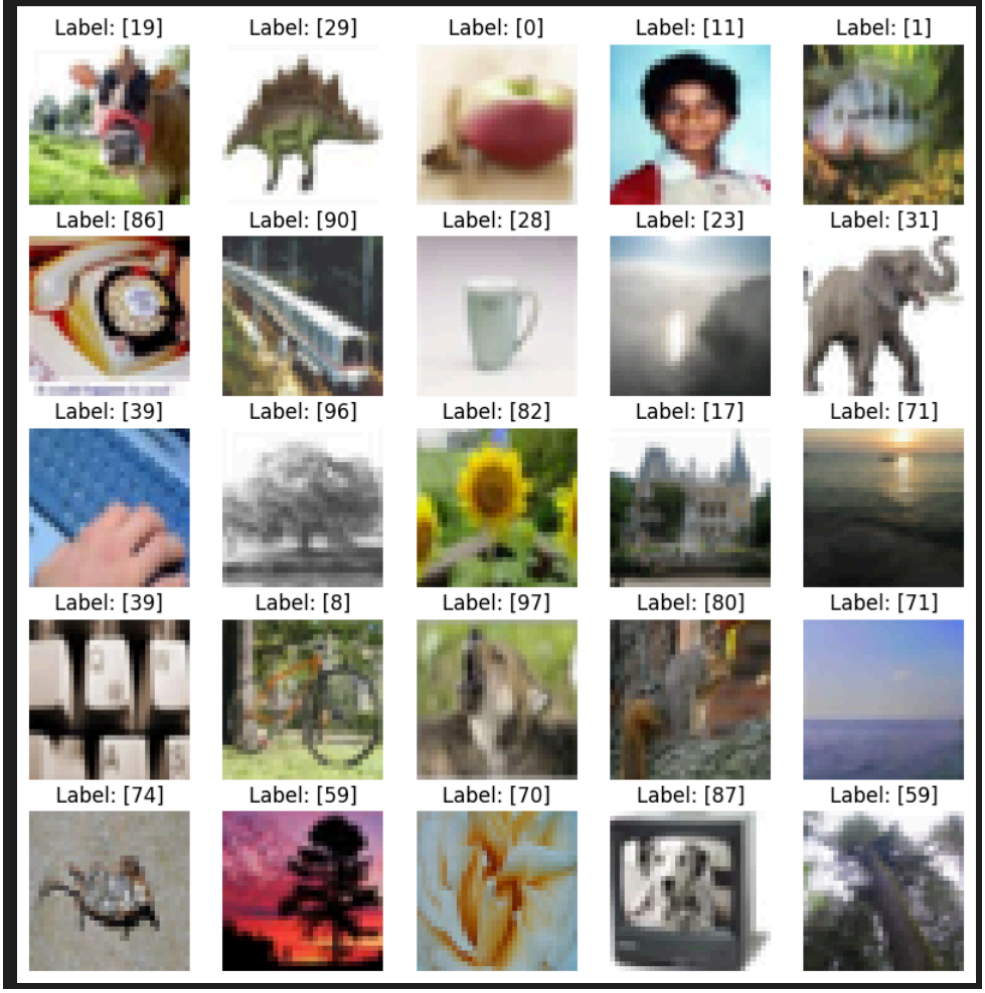
Пашковський Євгеній, ІП-41мн

task1 code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
import tensorflow as tf

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar100.load_data()

print(f'x_train shape: {x_train.shape}')
print(f'x_test shape: {x_test.shape}')

print(f'y_train shape: {y_train.shape}')
print(f'y_test shape: {y_test.shape}')


plt.figure(figsize=(10, 10))
for i in range(25):
    ax = plt.subplot(5, 5, i + 1)
    plt.imshow(x_train[i])
    plt.title(f'Label: {y_train[i]}')
    plt.axis('off')
plt.show()
```

```
x_train shape: (50000, 32, 32, 3)
x_test shape: (10000, 32, 32, 3)
y_train shape: (50000, 1)
y_test shape: (10000, 1)
```




```
      import keras
      from keras import models, layers

      def MyConv2D(x, filters = 32, kernel_size = (3, 3), strides = (1, 1),
padding = "same", activation = "relu"):
        x = layers.Conv2D(
          filters=filters,
          kernel_size=kernel_size,
          strides=strides,
          padding=padding,
        )(x)

        x = layers.BatchNormalization(axis=-1)(x)
        x = layers.Activation("relu")(x)
        return x
```

```
      def Stem(input):
         x = MyConv2D(input, filters=32, kernel_size=(3,3), strides=(2,2),
padding="valid", activation="relu")
          x = MyConv2D(x, filters=32, kernel_size=(3,3), padding="valid",
activation="relu")
          x = MyConv2D(x, filters=64, kernel_size=(3,3), padding="same",
activation="relu")
        x = layers.MaxPooling2D((3, 3), strides=(2, 2), padding="valid")(x)
          x = MyConv2D(x, filters=80, kernel_size=(1,1), padding="valid",
activation="relu")
          x = MyConv2D(x, filters=192, kernel_size=(3,3), padding="valid",
activation="relu")
        x = layers.MaxPooling2D((3, 3), strides=(2, 2), padding="valid")(x)
        return x

      def InceptionA(input, filters=64):
         x1 = MyConv2D(input, filters=64, kernel_size=(1,1), padding="same",
activation="relu")
          x1 = MyConv2D(x1, filters=96, kernel_size=(3,3), padding="same",
activation="relu")
          x1 = MyConv2D(x1, filters=96, kernel_size=(3,3), padding="same",
activation="relu")

         x2 = MyConv2D(input, filters=48, kernel_size=(1,1), padding="same",
activation="relu")
          x2 = MyConv2D(x2, filters=64, kernel_size=(3,3), padding="same",
activation="relu")

             x3 = layers.AveragePooling2D((3, 3), strides=(1, 1),
padding="same")(input)
        x3 = MyConv2D(x3, filters=filters, kernel_size=(1,1), padding="same",
activation="relu")

         x4 = MyConv2D(input, filters=64, kernel_size=(1,1), padding="same",
activation="relu")

        x = layers.Concatenate(axis=-1)([x1, x2, x3, x4])
        return x

      def ReductionA(input):
         x1 = MyConv2D(input, filters=64, kernel_size=(1,1), padding="same",
activation="relu")
```

```
        x1 = MyConv2D(x1, filters=96, kernel_size=(3,3), padding="same",
activation="relu")
        x1 = MyConv2D(x1, filters=96, kernel_size=(3,3), strides=(2,2),
padding="valid", activation="relu")

        x2 = MyConv2D(input, filters=384, kernel_size=(3,3), strides=(2,2),
padding="valid", activation="relu")

            x3 = layers.MaxPooling2D((3, 3), strides=(2, 2),
padding="valid")(input)

        x = layers.Concatenate(axis=-1)([x1, x2, x3])
        return x

    def InceptionB(input, filters=192):
            x1 = MyConv2D(input, filters=filters, kernel_size=(1,1),
padding="same", activation="relu")
        x1 = MyConv2D(x1, filters=filters, kernel_size=(7,1), padding="same",
activation="relu")
        x1 = MyConv2D(x1, filters=filters, kernel_size=(1,7), padding="same",
activation="relu")
        x1 = MyConv2D(x1, filters=filters, kernel_size=(7,1), padding="same",
activation="relu")
        x1 = MyConv2D(x1, filters=192, kernel_size=(1,7), padding="same",
activation="relu")

            x2 = MyConv2D(input, filters=filters, kernel_size=(1,1),
padding="same", activation="relu")
        x2 = MyConv2D(x2, filters=filters, kernel_size=(1,7), padding="same",
activation="relu")
        x2 = MyConv2D(x2, filters=192, kernel_size=(7,1), padding="same",
activation="relu")

            x3 = layers.AveragePooling2D((3, 3), strides=(1, 1),
padding="same")(input)
        x3 = MyConv2D(x3, filters=192, kernel_size=(1,1), padding="same",
activation="relu")

        x4 = MyConv2D(input, filters=192, kernel_size=(1,1), padding="same",
activation="relu")

        x = layers.Concatenate(axis=-1)([x1, x2, x3, x4])
```

```
        return x

    def ReductionB(input):
        x1 = MyConv2D(input, filters=192, kernel_size=(1,1), padding="same",
activation="relu")
        x1 = MyConv2D(x1, filters=192, kernel_size=(1,7), padding="same",
activation="relu")
        x1 = MyConv2D(x1, filters=192, kernel_size=(7,1), padding="same",
activation="relu")
        x1 = MyConv2D(x1, filters=192, kernel_size=(3,3), strides=(2,2),
padding='valid', activation="relu")


        x2 = MyConv2D(input, filters=192, kernel_size=(1,1), padding="same",
activation="relu")
        x2 = MyConv2D(input, filters=320, kernel_size=(3,3), strides=(2,2),
padding='valid', activation="relu")

            x3 = layers.MaxPooling2D((3, 3), strides=(2, 2),
padding="valid")(input)

        x = layers.Concatenate(axis=-1)([x1, x2, x3])
        return x

    def InceptionC(input):
        x1 = MyConv2D(input, filters=448, kernel_size=(1,1), padding="same",
activation="relu")
        x1 = MyConv2D(x1, filters=384, kernel_size=(3,3), padding="same",
activation="relu")
        x1_1 = MyConv2D(x1, filters=384, kernel_size=(1,3), padding="same",
activation="relu")
        x1_2 = MyConv2D(x1, filters=384, kernel_size=(3,1), padding="same",
activation="relu")

        x2 = MyConv2D(input, filters=384, kernel_size=(1,1), padding="same",
activation="relu")
        x2_1 = MyConv2D(x2, filters=384, kernel_size=(1,3), padding="same",
activation="relu")
        x2_2 = MyConv2D(x2, filters=384, kernel_size=(3,1), padding="same",
activation="relu")
```

```
            x3    =    layers.AveragePooling2D((3,    3),    strides=(1,    1),
padding="same")(input)
         x3  =  MyConv2D(x3,  filters=192,  kernel_size=(1,1),  padding="same",
activation="relu")

        x4 = MyConv2D(input, filters=320, kernel_size=(1,1), padding="same",
activation="relu")

        x = layers.Concatenate(axis=-1)([x1_1, x1_2, x2_1, x2_2, x3, x4])
        return x

      def Aux(input, n_classes = 1000):
            x    =    layers.AveragePooling2D((5,    5),    strides=(3,    3),
padding="same")(input)
          x  =  MyConv2D(x,  filters=128,  kernel_size=(1,1),  padding="same",
activation="relu")
        x = layers.Flatten()(x)
        x = layers.Dense(768, activation="relu")(x)
        x = layers.Dropout(rate = 0.2) (x)
        x = layers.Dense(n_classes, activation="softmax")(x)
        return x

      def InceptionV3(input_shape=(299, 299, 3), n_classes = 1000):
        input = layers.Input(input_shape)
        x = layers.Rescaling(1./255)(input)
        x = layers.Resizing(299, 299)(x)
        x = layers.RandomColorJitter(
          value_range=(0, 1),
          brightness_factor=0.2,
          contrast_factor=0.2,
          saturation_factor=0.2,
          hue_factor=0.2
        )(x)
        x = layers.RandomFlip(
          mode="horizontal_and_vertical"
        )(x)

        x = Stem(x)

        x = InceptionA(x, 32)
        x = InceptionA(x)
        x = InceptionA(x)
```

```
        x = ReductionA(x)

        x = InceptionB(x, 128)
        x = InceptionB(x, 160)
        x = InceptionB(x, 160)
        x = InceptionB(x, 192)

        aux = Aux(x, n_classes)

        x = ReductionB(x)

        x = InceptionC(x)
        x = InceptionC(x)

        x = layers.GlobalAveragePooling2D()(x)

        x = layers.Dense(2048, activation="relu")(x)
        x = layers.Dropout(rate = 0.2) (x)
        output = layers.Dense(n_classes, activation="softmax")(x)

        model = models.Model(inputs=input, outputs=output)
        return model
```

```
      from keras import optimizers

      epochs = 50

      initial_learning_rate = 1e-3
      final_learning_rate = 1e-8
      learning_rate_decay_factor        =        (final_learning_rate        /
initial_learning_rate) ** (1 / epochs)
      batch_size = 32
      steps_per_epoch = len(x_train) * 8 // 10 // batch_size

      print(f'Initial learning rate: {initial_learning_rate}')
      print(f'Final learning rate: {final_learning_rate}')
      print(f'Learning rate decay factor: {learning_rate_decay_factor}')
      print(f'Batch size: {batch_size}')
      print(f'Steps per epoch: {steps_per_epoch}')
```

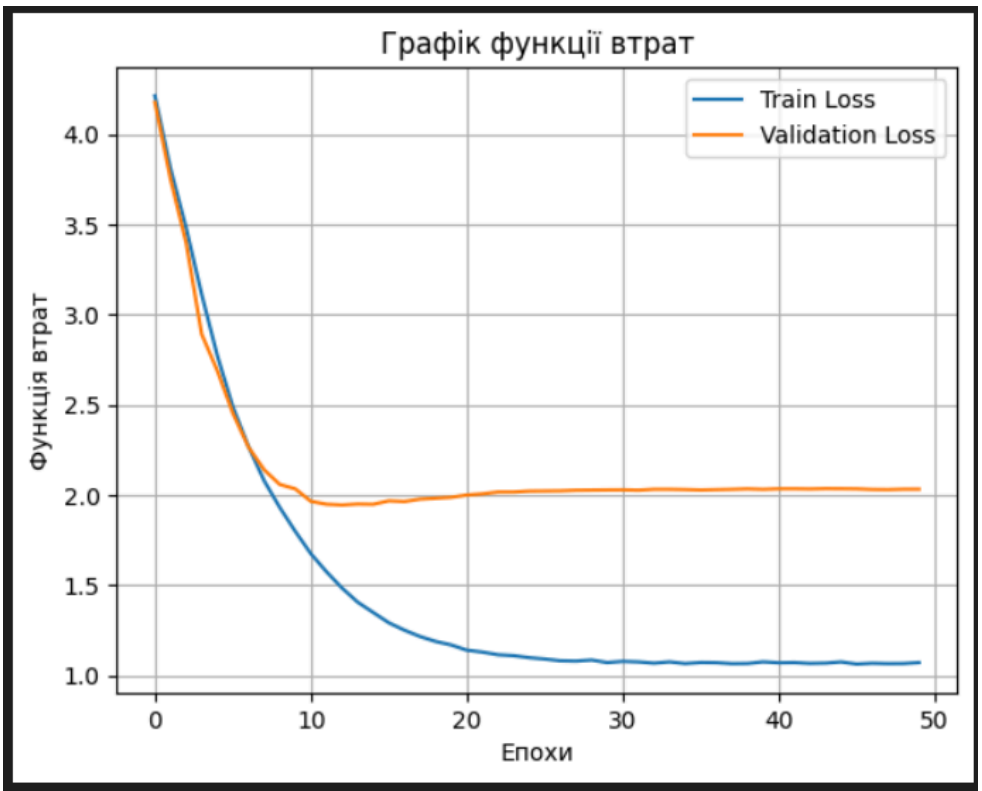
```
    learning_rate = optimizers.schedules.ExponentialDecay(
      initial_learning_rate=initial_learning_rate,
      decay_steps=steps_per_epoch,
      decay_rate=learning_rate_decay_factor
    )

    model = InceptionV3(input_shape=(32, 32, 3), n_classes=100)
    model.compile(optimizer=optimizers.Adam(learning_rate=learning_rate),
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    model.summary()
```

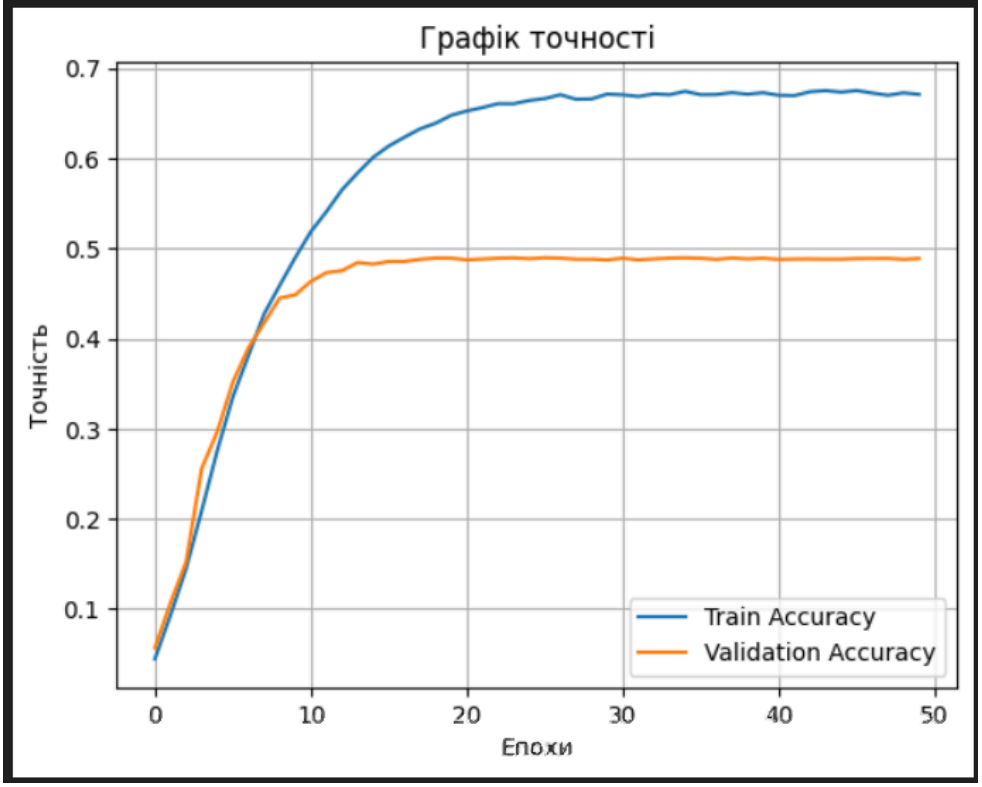
```
    history = model.fit(x_train, y_train, epochs=epochs,
validation_split=0.2, steps_per_epoch=steps_per_epoch, batch_size=batch_size)
```

```
    model.evaluate(x_test, y_test)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 11s 34ms/step - accuracy: 0.5111 - loss: 1.9735

[2.0017733573913574, 0.5048999786376953]
```

```
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.xlabel('Епохи')
    plt.ylabel('Функція втрат')
    plt.title('Графік функції втрат')
    plt.legend()
    plt.grid()
    plt.show()

    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.xlabel('Епохи')
    plt.ylabel('Точність')
    plt.title('Графік точності')
    plt.legend()
    plt.grid()
    plt.show()
```






```
predictions = model.predict(x_test).argmax(axis=1)

predictions
```

```
models.save_model(model, 'model.keras')
```

```
model = models.load_model('model.keras')
model.evaluate(x_test, y_test)
```

task2 code for .tflite file generation:

```
from keras import models
import tensorflow as tf

model = models.load_model('../task1/model.keras')
```

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

with tf.io.gfile.GFile('./model/model.tflite', 'wb') as f:
  f.write(tflite_model)
```

task2 code for server with /predict endpoint:

```
from fastapi import FastAPI, File, UploadFile
from fastapi.responses import JSONResponse
from PIL import Image
import io
import numpy as np
import tensorflow as tf

app = FastAPI()

cifar100_fine_labels = [
     'apple', 'aquarium_fish', 'baby', 'bear', 'beaver', 'bed', 'bee',
'beetle',
    'bicycle', 'bottle', 'bowl', 'boy', 'bridge', 'bus', 'butterfly',
       'camel', 'can', 'castle', 'caterpillar', 'cattle', 'chair',
'chimpanzee',
     'clock', 'cloud', 'cockroach', 'couch', 'crab', 'crocodile', 'cup',
'dinosaur',
       'dolphin', 'elephant', 'flatfish', 'forest', 'fox', 'girl',
'hamster', 'house',
```

```
            'kangaroo', 'keyboard', 'lamp', 'lawn_mower', 'leopard', 'lion',
'lizard',
          'lobster', 'man', 'maple_tree', 'motorcycle', 'mountain', 'mouse',
'mushroom',
            'oak_tree', 'orange', 'orchid', 'otter', 'palm_tree', 'pear',
'pickup_truck',
            'pine_tree', 'plain', 'plate', 'poppy', 'porcupine', 'possum',
'rabbit',
          'raccoon', 'ray', 'road', 'rocket', 'rose', 'sea', 'seal', 'shark',
'shrew',
            'skunk', 'skyscraper', 'snail', 'snake', 'spider', 'squirrel',
'streetcar',
              'sunflower', 'sweet_pepper', 'table', 'tank', 'telephone',
'television', 'tiger',
          'tractor', 'train', 'trout', 'tulip', 'turtle', 'wardrobe', 'whale',
'willow_tree',
          'wolf', 'woman', 'worm'
      ]

      print(len(cifar100_fine_labels))

      model_content = open("./model/model.tflite", "rb").read()
      interpreter = tf.lite.Interpreter(model_content=model_content)

      def infer(img):
        interpreter.allocate_tensors()

        input_details = interpreter.get_input_details()
        output_details = interpreter.get_output_details()

        input_data = np.array([np.array(img)], dtype=np.float32)
        interpreter.set_tensor(input_details[0]['index'], input_data)

        interpreter.invoke()

        output_data = interpreter.get_tensor(output_details[0]['index'])
        return output_data

      @app.get("/ping")
      async def ping():

        return "pong"
```

```
@app.post("/predict")
async def predict(file: UploadFile = File(...)):
    if not file.content_type.startswith("image/"):
        return JSONResponse(
            status_code=400,
              content={"error": "Invalid file type. Only image files are allowed."}
        )

    contents = await file.read()
    image = Image.open(io.BytesIO(contents))

    predictions = infer(image)
    labelIndex = int(np.array(predictions).argmax())

    return {
      "prediction": labelIndex,
      "label": cifar100_fine_labels[labelIndex]
    }
```

task2 Dockerfile:

```
FROM tensorflow/tensorflow
WORKDIR /app
COPY . .
RUN pip install --upgrade pip
RUN pip install fastapi pillow uvicorn numpy python-multipart tensorflow-cpu
EXPOSE 3000
CMD ["uvicorn", "--host", "0.0.0.0", "--port", "3000", "server:app"]
```

task2 process of running and querying the api:

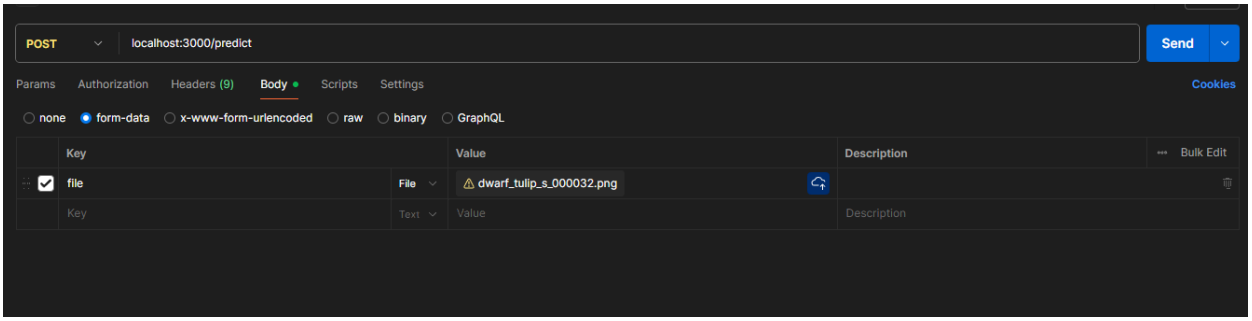
1. build docker image

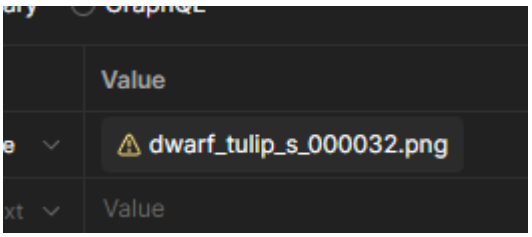
```
PS E:\Maricтр\semestr2\machine-learning-technologies> cd .\L1\task2\
PS E:\Maricтр\semestr2\machine-learning-technologies\L1\task2> docker build -t l1task2:1.0 .
[+] Building 56.5s (10/10) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 283B
 => [internal] load metadata for docker.io/tensorflow/tensorflow:latest
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/5] FROM docker.io/tensorflow/tensorflow:latest@sha256:f24e8494d43a4e613edd7fbd3782594368d52d79af1e216051139ed2d7830682
 => [internal] load build context
 => => transferring context: 626.51kB
 => CACHED [2/5] WORKDIR /app
 => [3/5] COPY . .
 => [4/5] RUN pip install --upgrade pip
 => [5/5] RUN pip install fastapi pillow uvicorn numpy python-multipart tensorflow-cpu
 => exporting to image
 => => exporting layers
 => => writing image sha256:4db0774acf8a5768be68eb57fba9e938c07c6e2562e20297199f5cd5c13b0f81
 => => naming to docker.io/library/l1task2:1.0

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/7qe7sia8un2obuusijqamhxy0

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
PS E:\Maricтр\semestr2\machine-learning-technologies\L1\task2>
```

2. run docker container & expose port

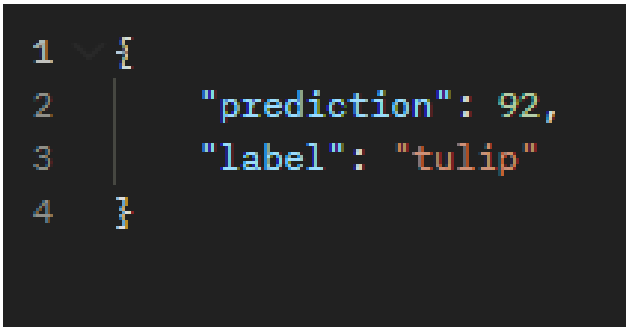
```
PS E:\Maricтр\semestr2\machine-learning-technologies\L1\task2> docker run -p 3000:3000 l1task2:1.0
2025-05-16 03:17:13.971795: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You
. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-05-16 03:17:16.002878: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow bi
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow wi
2025-05-16 03:17:34.548278: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You
. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
/usr/local/lib/python3.11/dist-packages/tensorflow/lite/python/interpreter.py:457: UserWarning:
    TF 2.20. Please use the LiteRT interpreter from the ai_edge_litert package.
    See the [migration guide](https://ai.google.dev/edge/litert/migration)
    for details.

  warnings.warn(_INTERPRETER_DELETION_WARNING)
INFO:     Started server process [1]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:3000 (Press CTRL+C to quit)
```

3. Enter url & attach image file in postman





4. Make request POST /predict with image attached



Result is correct!