

Додатки

A:

Перша модель

```
import cv2
import mlflow.keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Flatten
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator

# Initialize image data generator with rescaling
train_data_gen = ImageDataGenerator(rescale=1. / 255)
validation_data_gen = ImageDataGenerator(rescale=1. / 255)

# Preprocess all test images
train_generator = train_data_gen.flow_from_directory(
    'data/train',
    target_size=(48, 48),
    batch_size=64,
    color_mode="grayscale",
    class_mode='categorical')

# Preprocess all train images
validation_generator = validation_data_gen.flow_from_directory(
    'data/test',
    target_size=(48, 48),
    batch_size=64,
    color_mode="grayscale",
    class_mode='categorical')

# create model structure
emotion_model = Sequential()

emotion_model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=(48, 48, 1)))
emotion_model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Dropout(0.25))

emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
emotion_model.add(Dropout(0.25))

emotion_model.add(Flatten())
emotion_model.add(Dense(1024, activation='relu'))
emotion_model.add(Dropout(0.5))
emotion_model.add(Dense(7, activation='softmax'))

cv2ocl.setUseOpenCL(False)

emotion_model.compile(loss='categorical_crossentropy',
optimizer=Adam(learning_rate=0.0001, decay=1e-6),
metrics=['accuracy'])

# Запустити експеримент MLflow
mlflow.set_experiment("Emotion_Detection_Training")

# Запуск контексту MLflow
with mlflow.start_run():
    # Збереження гіперпараметрів моделі
    mlflow.log_param("epochs", 50)
    mlflow.log_param("learning_rate", 0.0001)
    mlflow.log_param("decay", 1e-6)

    # Train the neural network/model
    emotion_model_info = emotion_model.fit(
        train_generator,
        steps_per_epoch=28709 // 64,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=7178 // 64)

    # Збереження метрик навчання (точність та втрати) у MLflow
    train_loss, train_accuracy =
emotion_model.evaluate(train_generator)
    mlflow.log_metric("train_loss", train_loss)
    mlflow.log_metric("train_accuracy", train_accuracy)

    validation_loss, validation_accuracy =
emotion_model.evaluate(validation_generator)
    mlflow.log_metric("validation_loss", validation_loss)
    mlflow.log_metric("validation_accuracy", validation_accuracy)

    # Збереження навченої моделі у MLflow
    mlflow.keras.log_model(emotion_model, "EmotionDetection")
```

Друга модель

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import mlflow
import mlflow.tensorflow
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import roc_curve, auc, roc_auc_score
from IPython.display import clear_output
import warnings

warnings.filterwarnings('ignore')

train_dir = "data/train"
test_dir = "data/test"

SEED = 12
IMG_HEIGHT = 48
IMG_WIDTH = 48
BATCH_SIZE = 64
EPOCHS = 30
FINE_TUNING_EPOCHS = 20
LR = 0.01
NUM_CLASSES = 7
EARLY_STOPPING_CRITERIA = 3
CLASS_LABELS = ['Angry', 'Disgusted', 'Fearful', 'Happy', 'Neutral',
'Sad', "Surprised"]

preprocess_fun = tf.keras.applications.densenet.preprocess_input

train_datagen = ImageDataGenerator(horizontal_flip=True,
width_shift_range=0.1,
height_shift_range=0.05,
rescale=1. / 255,
validation_split=0.2,

preprocessing_function=preprocess_fun
)
test_datagen = ImageDataGenerator(rescale=1. / 255,
validation_split=0.2,

preprocessing_function=preprocess_fun)

train_generator =
train_datagen.flow_from_directory(directory=train_dir,

target_size=(IMG_HEIGHT, IMG_WIDTH),
```

```

batch_size=BATCH_SIZE,
                                                                    shuffle=True,
                                                                    color_mode="rgb",

class_mode="categorical",
                                                                    subset="training",
                                                                    seed=12
                                                                    )

validation_generator =
test_datagen.flow_from_directory(directory=train_dir,

target_size=(IMG_HEIGHT, IMG_WIDTH),

batch_size=BATCH_SIZE,
                                                                    shuffle=True,

color_mode="rgb",

class_mode="categorical",

subset="validation",
                                                                    seed=12
                                                                    )

test_generator = test_datagen.flow_from_directory(directory=test_dir,

target_size=(IMG_HEIGHT, IMG_WIDTH),

batch_size=BATCH_SIZE,
                                                                    shuffle=False,
                                                                    color_mode="rgb",

class_mode="categorical",
                                                                    seed=12
                                                                    )

# Helper Functions
def display_one_image(image, title, subplot, color):
    plt.subplot(subplot)
    plt.axis('off')
    plt.imshow(image)
    plt.title(title, fontsize=16)

def display_nine_images(images, titles, title_colors=None):
    subplot = 331
    plt.figure(figsize=(13, 13))
    for i in range(9):
        color = 'black' if title_colors is None else title_colors[i]
        display_one_image(images[i], titles[i], 331 + i, color)

```

```

plt.tight_layout()
plt.subplots_adjust(wspace=0.1, hspace=0.1)
plt.show()

def image_title(label, prediction):
    # Both prediction (probabilities) and label (one-hot) are arrays
    with one item per class.
    class_idx = np.argmax(label, axis=-1)
    prediction_idx = np.argmax(prediction, axis=-1)
    if class_idx == prediction_idx:
        return f'{CLASS_LABELS[prediction_idx]} [correct]', 'black'
    else:
        return f'{CLASS_LABELS[prediction_idx]} [incorrect, should be
{CLASS_LABELS[class_idx]}}', 'red'

def get_titles(images, labels, model):
    predictions = model.predict(images)
    titles, colors = [], []
    for label, prediction in zip(classes, predictions):
        title, color = image_title(label, prediction)
        titles.append(title)
        colors.append(color)
    return titles, colors

img_datagen = ImageDataGenerator(rescale=1. / 255)
img_generator = img_datagen.flow_from_directory(directory=train_dir,

target_size=(IMG_HEIGHT, IMG_WIDTH),

batch_size=BATCH_SIZE,
shuffle=True,
color_mode="rgb",

class_mode="categorical",

seed=12
)

clear_output()

images, classes = next(img_generator)
class_idxs = np.argmax(classes, axis=-1)
labels = [CLASS_LABELS[idx] for idx in class_idxs]
display_nine_images(images, labels)

def feature_extractor(inputs):
    feature_extractor =
tf.keras.applications.DenseNet169(input_shape=(IMG_HEIGHT, IMG_WIDTH,
3),

include_top=False,

```

```

weights="imagenet")(inputs)

    return feature_extractor

def classifier(inputs):
    x = tf.keras.layers.GlobalAveragePooling2D()(inputs)
    x = tf.keras.layers.Dense(256, activation="relu",
kernel_regularizer=tf.keras.regularizers.l2(0.01))(x)
    x = tf.keras.layers.Dropout(0.3)(x)
    x = tf.keras.layers.Dense(1024, activation="relu",
kernel_regularizer=tf.keras.regularizers.l2(0.01))(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    x = tf.keras.layers.Dense(512, activation="relu",
kernel_regularizer=tf.keras.regularizers.l2(0.01))(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    x = tf.keras.layers.Dense(NUM_CLASSES, activation="softmax",
name="classification")(x)

    return x

def final_model(inputs):
    densenet_feature_extractor = feature_extractor(inputs)
    classification_output = classifier(densenet_feature_extractor)

    return classification_output

def define_compile_model():
    inputs = tf.keras.layers.Input(shape=(IMG_HEIGHT, IMG_WIDTH, 3))
    classification_output = final_model(inputs)
    model = tf.keras.Model(inputs=inputs,
outputs=classification_output)

    model.compile(optimizer=tf.keras.optimizers.SGD(0.1),
loss='categorical_crossentropy',
metrics=['accuracy'])

    return model

model = define_compile_model()
clear_output()

mlflow.start_run()

mlflow.tensorflow.autolog()

# Hyperparameters
params = {
    "img_height": IMG_HEIGHT,
    "img_width": IMG_WIDTH,

```

```

    "batch_size": BATCH_SIZE,
    "epochs": EPOCHS,
    "fine_tuning_epochs": FINE_TUNING_EPOCHS,
    "lr": LR,
    "num_classes": NUM_CLASSES,
    "early_stopping_criteria": EARLY_STOPPING_CRITERIA,
}

# Log the parameters
for param, value in params.items():
    mlflow.log_param(param, value)

# Freezing the feature extraction layers
model.layers[1].trainable = False

model.summary()

earlyStoppingCallback =
tf.keras.callbacks.EarlyStopping(monitor='val_loss',
patience=EARLY_STOPPING_CRITERIA,
verbose=1,
restore_best_weights=True
)

history = model.fit(x=train_generator,
                    epochs=EPOCHS,
                    validation_data=validation_generator,
                    callbacks=[earlyStoppingCallback])

history = pd.DataFrame(history.history)

# Un-Freezing the feature extraction layers for fine tuning
model.layers[1].trainable = True

model.compile(optimizer=tf.keras.optimizers.SGD(0.001), # lower
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history_ = model.fit(x=train_generator, epochs=FINE_TUNING_EPOCHS,
                    validation_data=validation_generator)
history = history.append(pd.DataFrame(history_.history),
                        ignore_index=True)

x = px.line(data_frame=history, y=["accuracy", "val_accuracy"],
            markers=True)
x.update_xaxes(title="Number of Epochs")
x.update_yaxes(title="Accuracy")
x.update_layout(showlegend=True,
                title={
                    'text': 'Accuracy vs Number of Epochs',

```

```

        'y': 0.94,
        'x': 0.5,
        'xanchor': 'center',
        'yanchor': 'top'})
x.show()

x = px.line(data_frame=history,
            y=["loss", "val_loss"], markers=True)
x.update_xaxes(title="Number of Epochs")
x.update_yaxes(title="Loss")
x.update_layout(showlegend=True,
                title={
                    'text': 'Loss vs Number of Epochs',
                    'y': 0.94,
                    'x': 0.5,
                    'xanchor': 'center',
                    'yanchor': 'top'})

x.show()

model.evaluate(test_generator)
preds = model.predict(test_generator)
y_preds = np.argmax(preds, axis=1)
y_test = np.array(test_generator.labels)

cm_data = confusion_matrix(y_test, y_preds)
cm = pd.DataFrame(cm_data, columns=CLASS_LABELS, index=CLASS_LABELS)
cm.index.name = 'Actual'
cm.columns.name = 'Predicted'
plt.figure(figsize=(20, 10))
plt.title('Confusion Matrix', fontsize=20)
sns.set(font_scale=1.2)
ax = sns.heatmap(cm, cbar=False, cmap="Blues", annot=True,
                 annot_kws={"size": 16}, fmt='g')

print(classification_report(y_test, y_preds))

fig, c_ax = plt.subplots(1, 1, figsize=(15, 8))

def multiclass_roc_auc_score(y_test, y_pred, average="macro"):
    lb = LabelBinarizer()
    lb.fit(y_test)
    y_test = lb.transform(y_test)
    for (idx, c_label) in enumerate(CLASS_LABELS):
        fpr, tpr, thresholds = roc_curve(y_test[:, idx].astype(int),
                                         y_pred[:, idx])
        c_ax.plot(fpr, tpr, lw=2, label='%s (AUC:%0.2f)' % (c_label,
                                                         auc(fpr, tpr)))
        c_ax.plot(fpr, fpr, 'black', linestyle='dashed', lw=4,
                 label='Random Guessing')
    return roc_auc_score(y_test, y_pred, average=average)

```



```

print('ROC AUC score:', multiclass_roc_auc_score(y_test, preds,
average="micro"))
plt.xlabel('FALSE POSITIVE RATE', fontsize=18)
plt.ylabel('TRUE POSITIVE RATE', fontsize=16)
plt.legend(fontsize=11.5)
plt.show()

print("ROC-AUC Score = ", roc_auc_score(to_categorical(y_test),
preds))

# Log metrics
mlflow.log_metric("accuracy", accuracy)
mlflow.log_metric("val_accuracy", val_accuracy)
mlflow.log_metric("loss", loss)
mlflow.log_metric("val_loss", val_loss)
mlflow.log_metric("roc_auc_score",
roc_auc_score(to_categorical(y_test), preds))

# Save the model
mlflow.keras.save_model(model, "EmotionDetection")

mlflow.end_run()

```

FLASK Сервер:

```

from flask import Flask, request, jsonify, make_response
import mlflow.pyfunc
import numpy as np
import cv2
import os
import base64

app = Flask(__name__)

emotion_dict = {0: "Angry", 1: "Disgusted", 2: "Fearful", 3: "Happy", 4: "Neutral",
5: "Sad", 6: "Surprised"}

# Set MLflow Tracking URI
mlflow.set_tracking_uri("mlruns")

# Load MLflow model
model_name = "EmotionRecognition"
model_version = 1
model_path = f"models://{model_name}/{model_version}"

model = mlflow.pyfunc.load_model(model_path)

@app.route('/predict', methods=['POST'])
def predict():
    try:
        img_data = request.form['img_data']
        img_data = base64.b64decode(img_data)
        nparr = np.frombuffer(img_data, np.uint8)
        img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)

```

```

gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Find haar cascade to draw bounding box around face
face_detector =
cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_default.xml')
faces = face_detector.detectMultiScale(gray_img, scaleFactor=1.3,
minNeighbors=5)

if len(faces) > 0:
    x, y, w, h = faces[0]
    roi_gray_img = gray_img[y:y + h, x:x + w]
    roi_gray_img = cv2.resize(roi_gray_img, (48, 48))
    roi_gray_img = roi_gray_img.reshape(1, 48, 48, 1)
    roi_gray_img = roi_gray_img.astype('float32') / 255

    with mlflow.start_run() as run:
        predictions = model.predict(roi_gray_img)
        maxindex = int(np.argmax(predictions))
        emotion = emotion_dict[maxindex]

        # Log model performance
        mlflow.log_param("emotion", emotion)

        return jsonify(x=int(x), y=int(y), w=int(w), h=int(h), emotion=emotion)
    else:
        return jsonify(error='No face detected')
except Exception as e:
    print(f"Error: {e}")
    return make_response(jsonify(error='Error occurred during processing'), 500)

if __name__ == '__main__':
    port = int(os.environ.get("PORT", 5001))
    app.run(host="0.0.0.0", port=port)

```

B:

```
//  
// HumanVerifyApp.swift  
// HumanVerify  
//  
// Created by Max Stefankiv on 17.05.2023.  
//
```

import

```
@main  
struct HumanVerifyApp App  
    var body some Scene  
        WindowGroup  
            ContentView
```

```
//  
// CameraCaptureHelper.swift  
// HumanVerify IOS APP  
//  
// Created by Max Stefankiv on 18.04.2023.  
//
```

```
import  
import  
import  
import
```

```
class CameraCaptureHelper NSObject  
    let captureSession AVCaptureSession  
    let cameraPosition AVCaptureDevice Position  
  
    weak var delegate CameraCaptureHelperDelegate  
  
    required init cameraPosition AVCaptureDevice Position  
        self cameraPosition  
  
    super init  
  
    initialiseCaptureSession
```

```
fileprivate func initialiseCaptureSession
    captureSession sessionPreset AVCaptureSession.Preset.hd1280x720
```

```
    let AVCaptureDevice.DiscoverySession
    builtInWideAngleCamera video cameraPosition
```

```
    guard let devices first else
        fatalError("Unable to access camera")
```

```
    print("Camera is accessible")
```

```
    do
        let try AVCaptureDeviceInput device
```

```
        captureSession addInput
        print("Camera input added successfully")
```

```
    catch
        fatalError("Unable to access back camera")
```

```
    let AVCaptureVideoDataOutput
```

```
        setSampleBufferDelegate self
    queue "sample buffer delegate"
```

```
    if captureSession.canAddOutput
        captureSession.addOutput
        print("Camera output added successfully")
```

```
    captureSession.startRunning
    print("Camera session started running")
```

```
extension CameraCaptureHelper AVCaptureVideoDataOutputSampleBufferDelegate
    func captureOutput _ AVCaptureOutput didOutput CMSampleBuffer
    from AVCaptureConnection
    var AVCaptureVideoOrientation portrait
        main sync
        portrait
```

```
videoOrientation
```

```
guard let CMSampleBufferGetImageBuffer else  
return
```

```
main async  
self.delegate = newCameraImage self  
image = CImage cvPixelBuffer
```

```
protocol CameraCaptureHelperDelegate AnyObject  
func newCameraImage _ CameraCaptureHelper image CImage
```

```
//  
// Emotion.swift  
// HumanVerify IOS APP  
//  
// Created by Max Stefankiv on 18.04.2023.  
//
```

```
import
```

```
struct EmotionResponse Decodable  
let x Int  
let y Int  
let w Int  
let h Int  
let emotion String  
let error String
```

```
//  
// CameraViewModel.swift  
// HumanVerify IOS APP  
//  
// Created by Max Stefankiv on 18.04.2023.  
//
```

```
import  
import  
import
```

```
import
import
```

```
class CameraViewModel NSObject ObservableObject CameraCaptureHelperDelegate
```

```
  @Published var session AVCaptureSession
```

```
  @Published var detectedFace CGRect
```

```
  @Published var emotionText ""
```

```
  private var output AVCaptureVideoDataOutput
```

```
  private var faceDetectionRequest VNDetectFaceRectanglesRequest
```

```
  private let sequenceHandler VNSequenceRequestHandler
```

```
  private var cameraCaptureHelper CameraCaptureHelper
```

```
  // Add the frame counter property
```

```
  private var frameCounter 0
```

```
  override init
```

```
    super init
```

```
    configureSession
```

```
  func startSession
```

```
    if session isRunning
```

```
      DispatchQueue.global(qos: .userInitiated).async {
```

```
        self.session.startRunning()
      }
```

```
  func stopSession
```

```
    if session isRunning
```

```
      session.stopRunning()
```

```
  private func configureSession
```

```
    checkCameraPermissions { weak self in
```

```
      guard let self else return
```

```
      if
```

```
        self.setupSession()
```

```
      else
```

```
        print("Camera permission denied")
```

```

private var screenOrientation AVCaptureVideoOrientation
if let UIApplication.shared.connectedScenes.first as UIWindowScene
    switch interfaceOrientation
    case portrait
        return portrait
    case landscapeLeft
        return landscapeLeft
    case landscapeRight
        return landscapeRight
    case portraitUpsideDown
        return portraitUpsideDown
    default
        return portrait

return portrait

```

```

private func scale faceRect CGRect imageSize CGSize viewSize CGSize CGRect
// faceRect = (295.0, 913.0, 498.0, 498.0)
// imageSize = (1080.0, 1920.0)
// viewSize = (390.0, 844.0)

/*
295.0 = 1080
x = 390

x = (point * 390)/1080
x = (point * viewSize.width)/imageSize.width
x = point * (viewSize.width/imageSize.width)
widthScale = viewSize.width / imageSize.width
x = point * widthScale

----

913 = 1920
y = 844

y = 913 * 844 / 1920
heightScale = viewSize.height / imageSize.height
y = point * heightScale

--
*/
let width minX width

```

```

let width width
let height height

print "faceRect.minX: minX ; scaled: minX "

return CGRect
    x
    y minY height 0.1
    width width
    height height

```

```

private func checkCameraPermissions completion @escaping Bool Void
switch AVCaptureDevice authorizationStatus for video
case authorized
    true
case notDetermined
    AVCaptureDevice requestAccess for video in

default
    false

```

```

private func setupSession
    global qos userInitiated async
    self session beginConfiguration

    // Setup camera input
    if let AVCaptureDevice default builtInWideAngleCamera for video position
front
    print "Camera initialized"
    do
        let try AVCaptureDeviceInput device
        if self session canAddInput
            self session addInput

        catch
            print "Error: Unable to add camera input to AVCaptureSession"

    else

```



```

    print "Error: Camera not found"

// Setup video output
let AVCaptureVideoDataOutput
    setSampleBufferDelegate self queue
    label "camera output"
if self session canAddOutput
    self session addOutput

self session commitConfiguration

// Initialize and set the CameraCaptureHelper delegate
self cameraCaptureHelper CameraCaptureHelper cameraPosition front
self cameraCaptureHelper delegate self

    global qos userInitiated async
self session startRunning
print "Session started running" // Debug print

private var exifOrientation Int32
let exifOrientationForCurrentDeviceOrientation
return Int32(rawValue

private func exifOrientationForCurrentDeviceOrientation
if let UIApplication shared connectedScenes first as UIWindowScene
    switch interfaceOrientation
    case portrait
        return right
    case landscapeLeft
        return up
    case landscapeRight
        return down
    case portraitUpsideDown
        return left
    default
        return right

return right

```

```

extension CameraViewModel AVCaptureVideoDataOutputSampleBufferDelegate
    func captureOutput _ AVCaptureOutput didOutput CMSampleBuffer
from AVCaptureConnection
    guard let CMSampleBufferGetImageBuffer else
        return

    let CImage cvPixelBuffer

    // Update the frame counter
    frameCounter 1

    // Send every Nth frame to the server, for example, every 10th frame
    if frameCounter 8 0
        main async
        // to get the correct orientation of the image
        let orientedForExifOrientation self exifOrientation
        self cameraCaptureHelper delegate newCameraImage self cameraCaptureHelper
image

```

```

extension CameraViewModel
    func newCameraImage _ CameraCaptureHelper image CImage
    // Load CImage into UIImage and convert it to Data for transfer to the server
    if let CGContext createCGImage from extent
        let UIImage cgImage 0.5

        let base64EncodedString

        let Parameters
            "img_data"

        AF request "http://192.168.0.101:5001/predict"
            method post
            parameters
            encoding URLEncoding httpBody
            headers nil responseDecodable of EmotionResponse self weak self
    in

```

```
guard let self else return
```

```
switch result  
case success let  
    if let emotion  
        main async  
            guard  
                let x  
                let y  
                let w  
                let h  
            else  
                return
```

```
let CGRect x y width height  
if let UIApplication shared connectedScenes first as  
UIWindowScene  
    let windows first rootViewController view  
    let scale faceRect imageSize  
    extent size viewSize bounds size  
    detectedFace  
    emotionText
```

```
else if let error "No face detected"  
    main async  
        detectedFace CGRect  
        emotionText ""
```

```
case failure let  
    print "Request error: localizedDescription "  
    if let data  
        let String data encoding utf8  
        print "Server response: "No error message" "
```

```
//  
// ContentView.swift
```

```

// HumanVerify IOS APP
//
// Created by Max Stefankiv on 18.04.2023.
//

import
import

struct ContentView View
    @StateObject private var cameraViewModel CameraViewModel
    @State private var showCamera false

    var body some View
        ZStack
            if showCamera
                CameraView session cameraViewModel session
                    ignoresSafeArea
                    overlay OverlayShape rect cameraViewModel detectedFace
                    onAppear
                        cameraViewModel startSession

                    onDisappear
                        cameraViewModel stopSession

                Text cameraViewModel emotionText
                    font system size 24
                    bold
                    foregroundColor red
                    position x cameraViewModel detectedFace midX y
cameraViewModel detectedFace minY 15
                    opacity cameraViewModel emotionText isEmpty 0 1
                Button action
                    showCamera false

                Text "Закрити камеру"
                    font title
                    bold
                    padding
                    background Color red
                    foregroundColor white
                    cornerRadius 10

                    position x UIScreen main bounds width 180 y 50
            else
                VStack

```

```
Text "HumanVerify"  
    font largeTitle  
    bold  
    padding bottom 50  
Button action  
    showCamera toggle
```

```
Text "Відкрити камеру"  
    font title  
    bold  
    padding  
    background Color blue  
    foregroundColor white  
    cornerRadius 10
```

```
struct CameraView UIViewControllerRepresentable  
    let session AVCaptureSession
```

```
func makeUIViewController context Context UIViewController  
    let viewController UIViewController  
    let previewLayer AVCaptureVideoPreviewLayer session session  
        videoGravity resizeAspectFill  
        view layer addSublayer  
        frame view bounds  
        view layer masksToBounds true  
    return
```

```
func updateUIViewController _ UIViewController context Context  
    if let previewLayer AVCaptureVideoPreviewLayer  
        previewLayer view layer sublayers first as  
        frame view bounds
```

```
struct ContentView_Previews PreviewProvider  
    static var previews some View  
        ContentView
```

```
struct OverlayShape View
  var rect CGRect
```

```
var body some View
  RoundedRectangle cornerRadius 50
    stroke Color.red lineWidth 4
    frame width rect.width height rect.height
    position x rect.midX y rect.midY
    opacity rect.CGRect 0 1
```