USEK

SCHOOL OF ENGINEERING
DEPARTMENT OF **COMPUTER**, **ELECTRICAL**,
AND **TELECOMMUNICATION** ENGINEERING

Presented By:

Catherina **EL KHOURY - 202101204**

Anthony **EL CHEMALY - 202100079**

Presented to:

Dr. Hayssam **SERHAN**

GEL521 - Machine Learning

Final Project

IMAGENET

10/5/2024

Agenda

1 — What is ImageNet?

2 — What is Tiny ImageNet?

3 — Training CNN - Keras on Tiny ImageNet dataset from scratch.

4 — Pre-Trained Models in Keras.

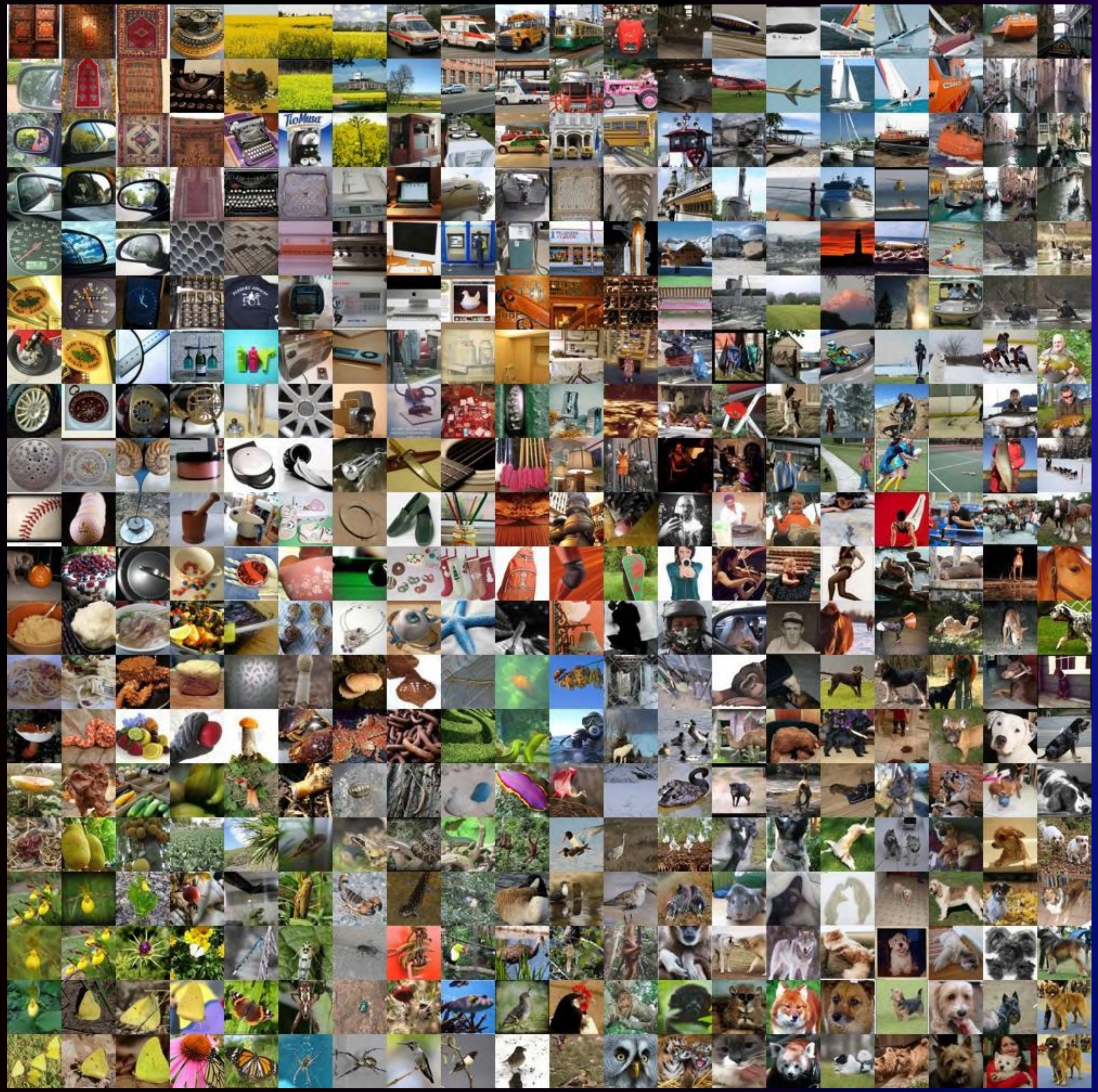
5 — Finetuning VGG16 on a new set of classes.

6 — Testing with OpenCV.

What is ImageNet?

ImageNet is a large visual database designed for use in visual object recognition software research.

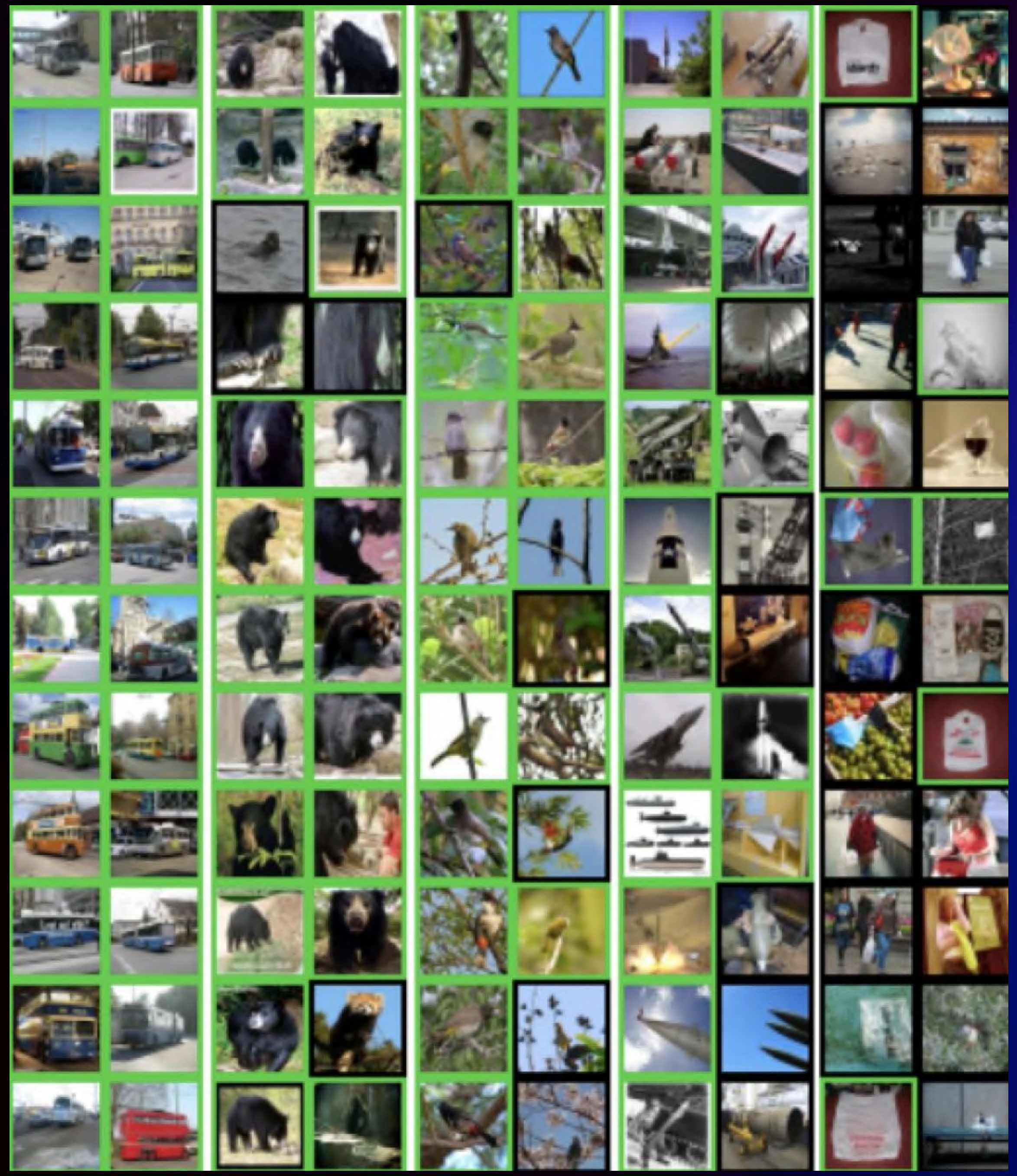
ImageNet offer tens of millions of cleanly labeled and sorted images and contains more than 20,000 categories.



What is Tiny ImageNet?

Tiny ImageNet, a subset of the large ImageNet dataset, contains 100,000 images of 200 classes downsized to 64×64 colored images.

Each class has 500 training images,
50 validation images and 50 test images.



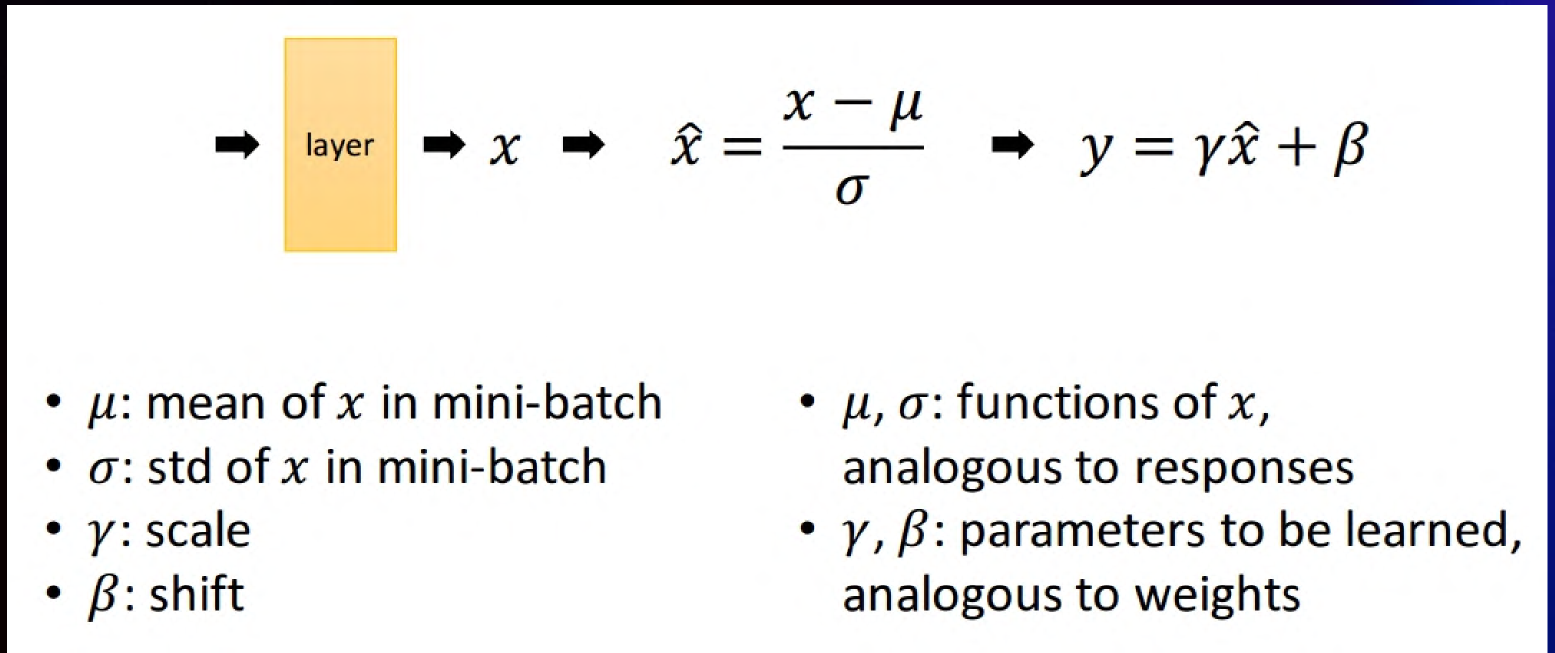
Training CNN - Keras on Tiny ImageNet dataset from scratch

Batch Normalization

This technique normalizes the input layer by adjusting and scaling activations.

It helps to stabilize and speed up the training of deep neural networks.

It's applied after each convolutional operation and before activation, which has been shown to work well in practice.



Training CNN - Keras on Tiny ImageNet dataset from scratch

L2 Regularisation

Applied to the convolutional layers, helping to keep the weights small and improving the generalization capabilities of the model.

Adds a penalty on the norm of the layer weights and is used to regularize the learning, preventing the model weights from fitting too perfectly to the train data which can lead to overfitting.

L2 Regularization

$$\text{Modified loss function} = \text{Loss function} + \lambda \sum_{i=1}^{n} W_i^2$$

We add the square of the weights as a regularisation term to the loss function.

Training CNN - Keras on Tiny ImageNet dataset from scratch

He Normalisation

This method is used for initializing the weights of deep neural networks that use **ReLU activation functions**.

It initialize the weights of the network in such a way that the variance of the outputs from a layer with ReLU activation remains the same as the variance of its inputs.

This helps in maintaining a stable gradient flow through deep networks, avoiding the vanishing gradients problem and preventing the activations from becoming too small (vanishing) or too large (exploding).

$$W \sim N(0, \sigma)$$

where $\sigma = \sqrt{\frac{2}{fan_{in}}}$

fan in: number of inputs in layer

Training CNN - Keras on Tiny ImageNet dataset from scratch

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 32)	896
batch_normalization (Batch Normalization)	(None, 64, 64, 32)	128
activation (Activation)	(None, 64, 64, 32)	0
conv2d_1 (Conv2D)	(None, 64, 64, 128)	36992
batch_normalization_1 (Bat chNormalization)	(None, 64, 64, 128)	512
activation_1 (Activation)	(None, 64, 64, 128)	0
max_pooling2d (MaxPooling2 D)	(None, 32, 32, 128)	0
conv2d_2 (Conv2D)	(None, 32, 32, 256)	295168
batch_normalization_2 (Bat chNormalization)	(None, 32, 32, 256)	1024
activation_2 (Activation)	(None, 32, 32, 256)	0
max_pooling2d_1 (MaxPoolin g2D)	(None, 16, 16, 256)	0
conv2d_3 (Conv2D)	(None, 16, 16, 512)	1180160
batch_normalization_3 (Bat chNormalization)	(None, 16, 16, 512)	2048
activation_3 (Activation)	(None, 16, 16, 512)	0
max_pooling2d_2 (MaxPoolin g2D)	(None, 8, 8, 512)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dense (Dense)	(None, 200)	102600

Batch Size: 128

Epochs: 24

After 51 minutes, we got:

Accuracy: 61.19%

Validation Accuracy: 26.84%

```
Epoch 24: val_loss did not improve from 3.43753
781/781 [==============================] - 106s 136ms/step - loss: 1.9220 - accuracy: 0.6119 - val_loss: 4.5249 - val_accuracy: 0.2684
<keras.src.callbacks.History at 0x7d371ea0f130>
```

Training CNN - Keras on Tiny ImageNet dataset from scratch

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 32)	896
batch_normalization (BatchNormalization)	(None, 64, 64, 32)	128
activation (Activation)	(None, 64, 64, 32)	0
conv2d_1 (Conv2D)	(None, 64, 64, 128)	36992
batch_normalization_1 (BatchNormalization)	(None, 64, 64, 128)	512
activation_1 (Activation)	(None, 64, 64, 128)	0
max_pooling2d (MaxPooling2D)	(None, 32, 32, 128)	0
conv2d_2 (Conv2D)	(None, 32, 32, 256)	295168
batch_normalization_2 (BatchNormalization)	(None, 32, 32, 256)	1024
activation_2 (Activation)	(None, 32, 32, 256)	0
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 256)	0
conv2d_3 (Conv2D)	(None, 16, 16, 512)	1180160
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 512)	2048
activation_3 (Activation)	(None, 16, 16, 512)	0
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 512)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dense (Dense)	(None, 200)	102600

Batch Size: 128

Epochs: 106

After 4 hours, we got:

Accuracy: 91.32%

Validation Accuracy: 30.56%

```
Epoch 106/240
781/781 [==============================] - ETA: 0s - loss: 0.9744 - accuracy: 0.9132
Epoch 106:781/781 [==============================] - 103s 132ms/step - loss: 0.9744 - accuracy: 0.9132 - val_loss: 7.7282 - val_accuracy: 0.3056
```

After that, we were interrupted by Google Colab limits

Pre-Trained Models in Keras

A Pre-trained Model refers to a neural network model that has been trained on a large dataset for a specific task.

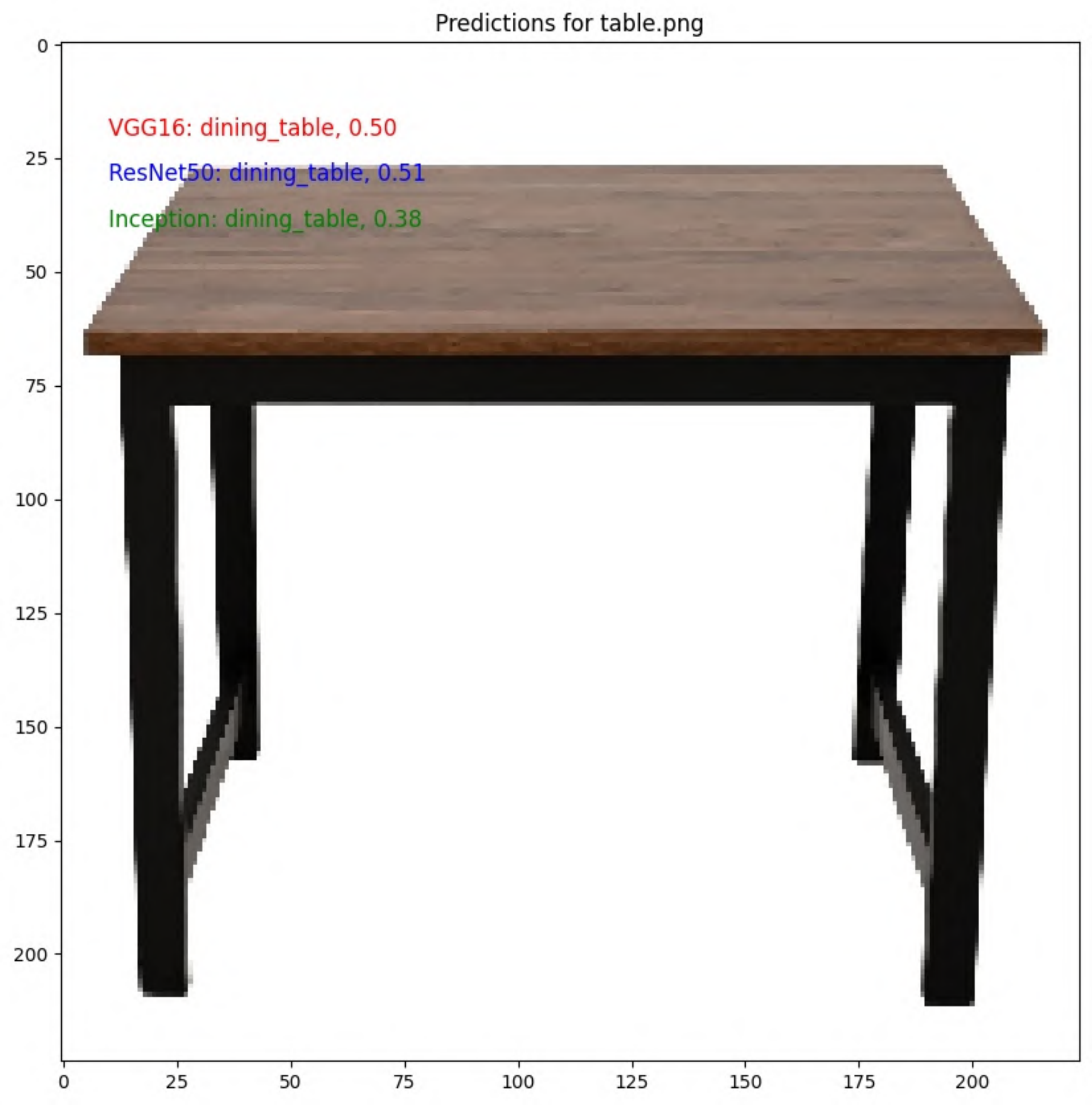
Using pre-trained models can be advantageous because they have already learned to recognize patterns and features from large amount of data, which can save significant time compared to training a model from scratch.

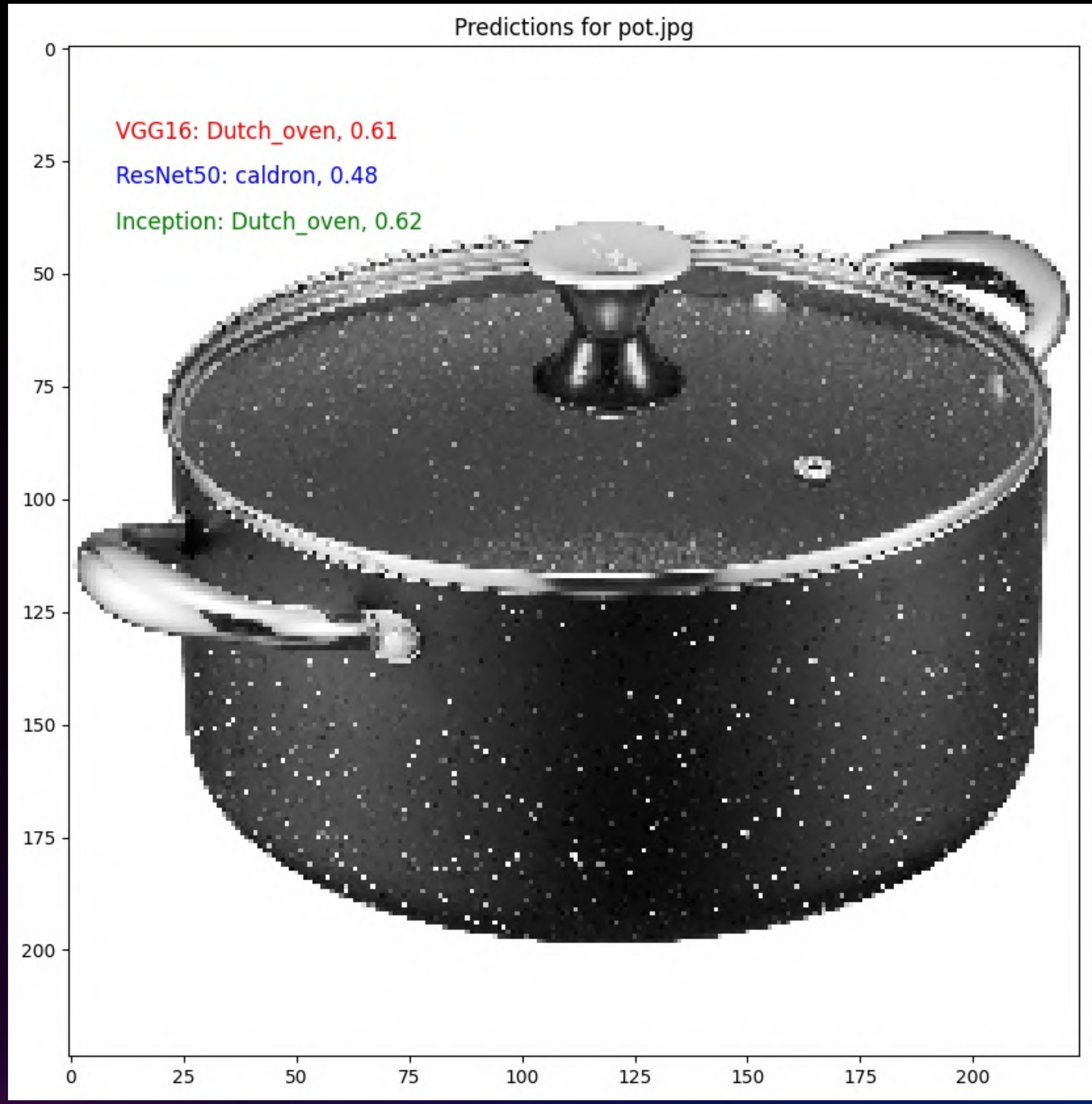
Pre-Trained Models in Keras

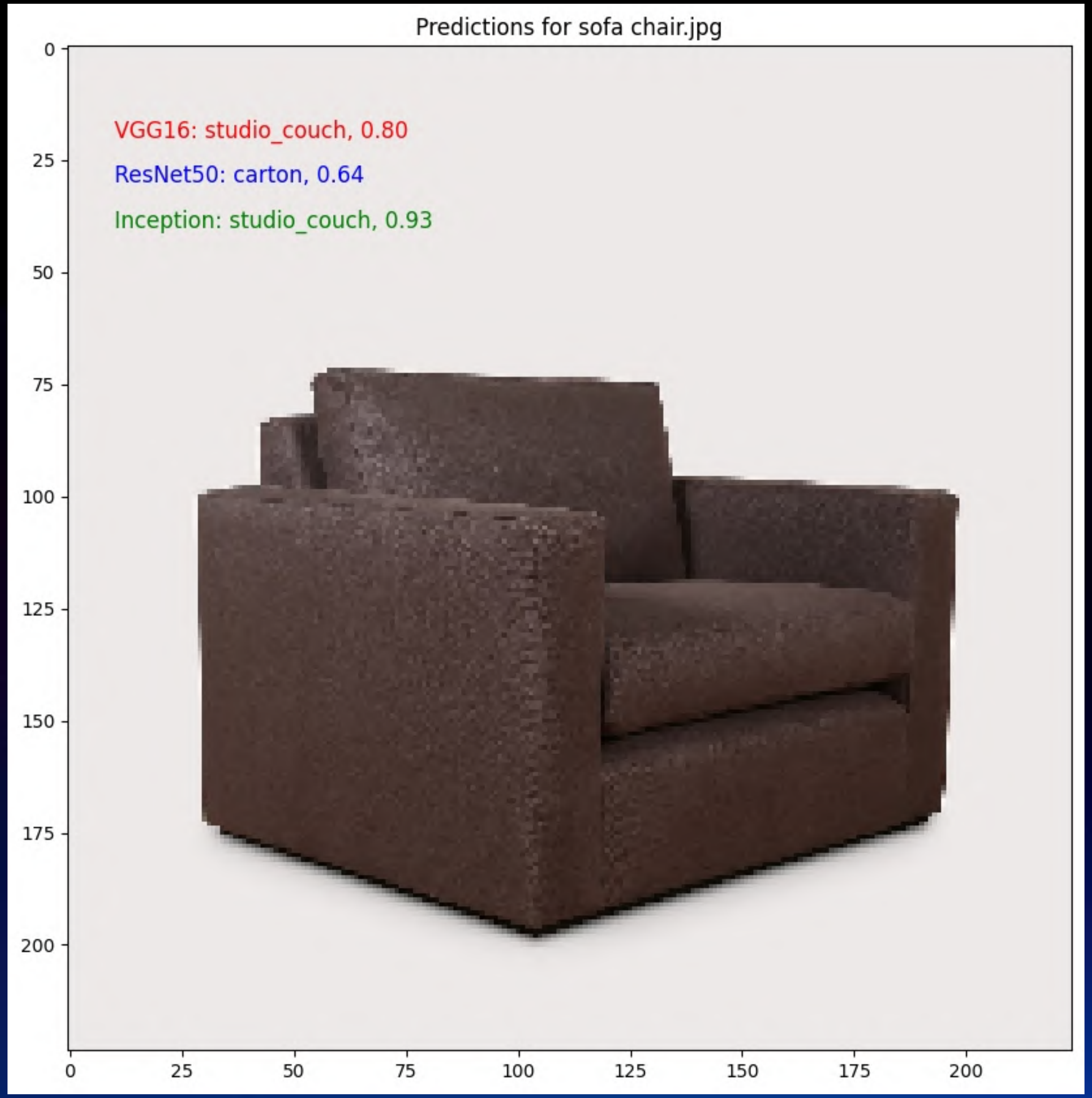
We will experiment with models pre-trained on a very popular subset of ImageNet called **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)** which includes 1,000 classes, 1,281,167 training images, 50,000 validation images and 100,000 test images.

- VGG16
- ResNet50
- InceptionV3

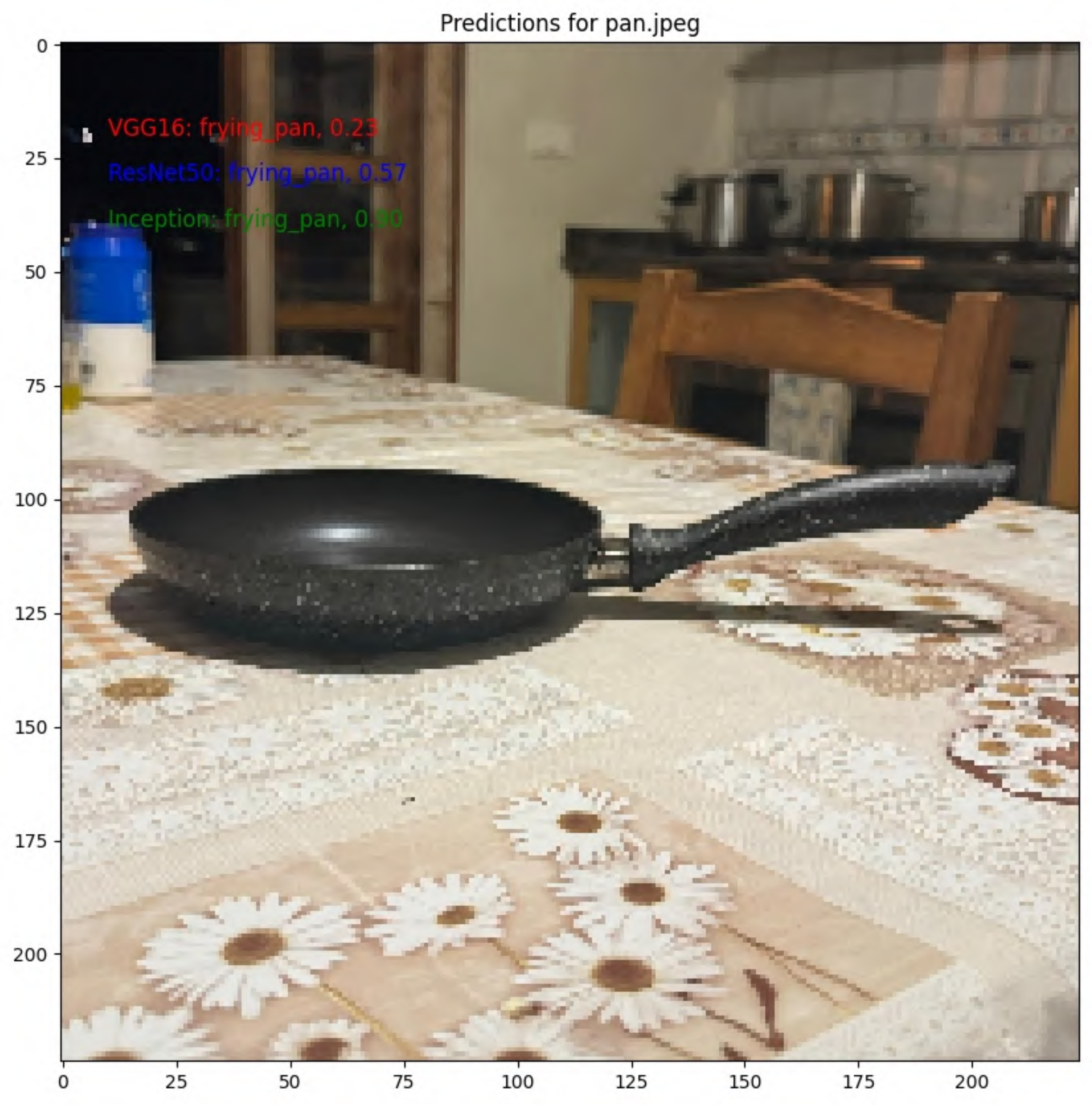
Pre-Trained Models in Keras

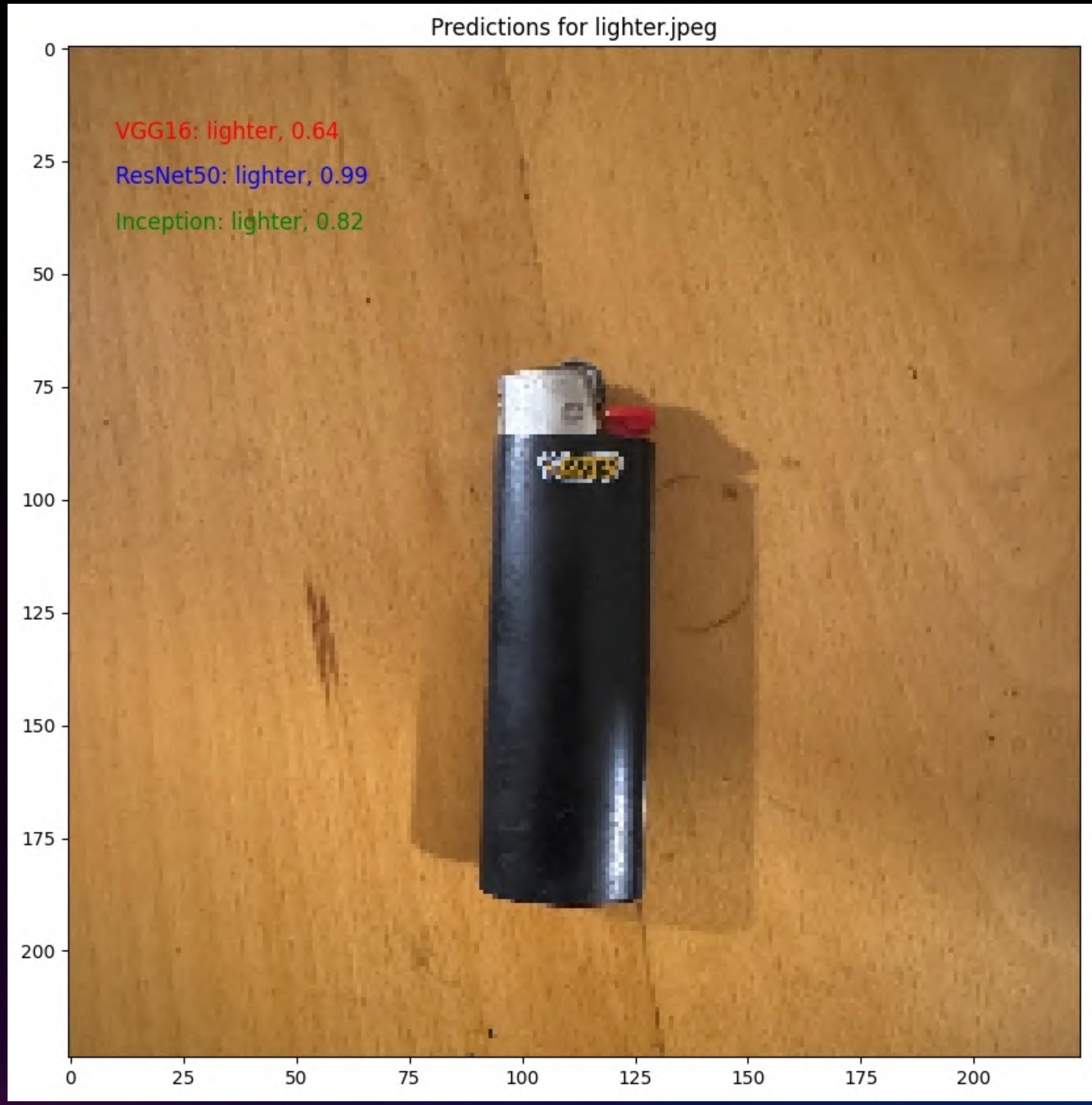


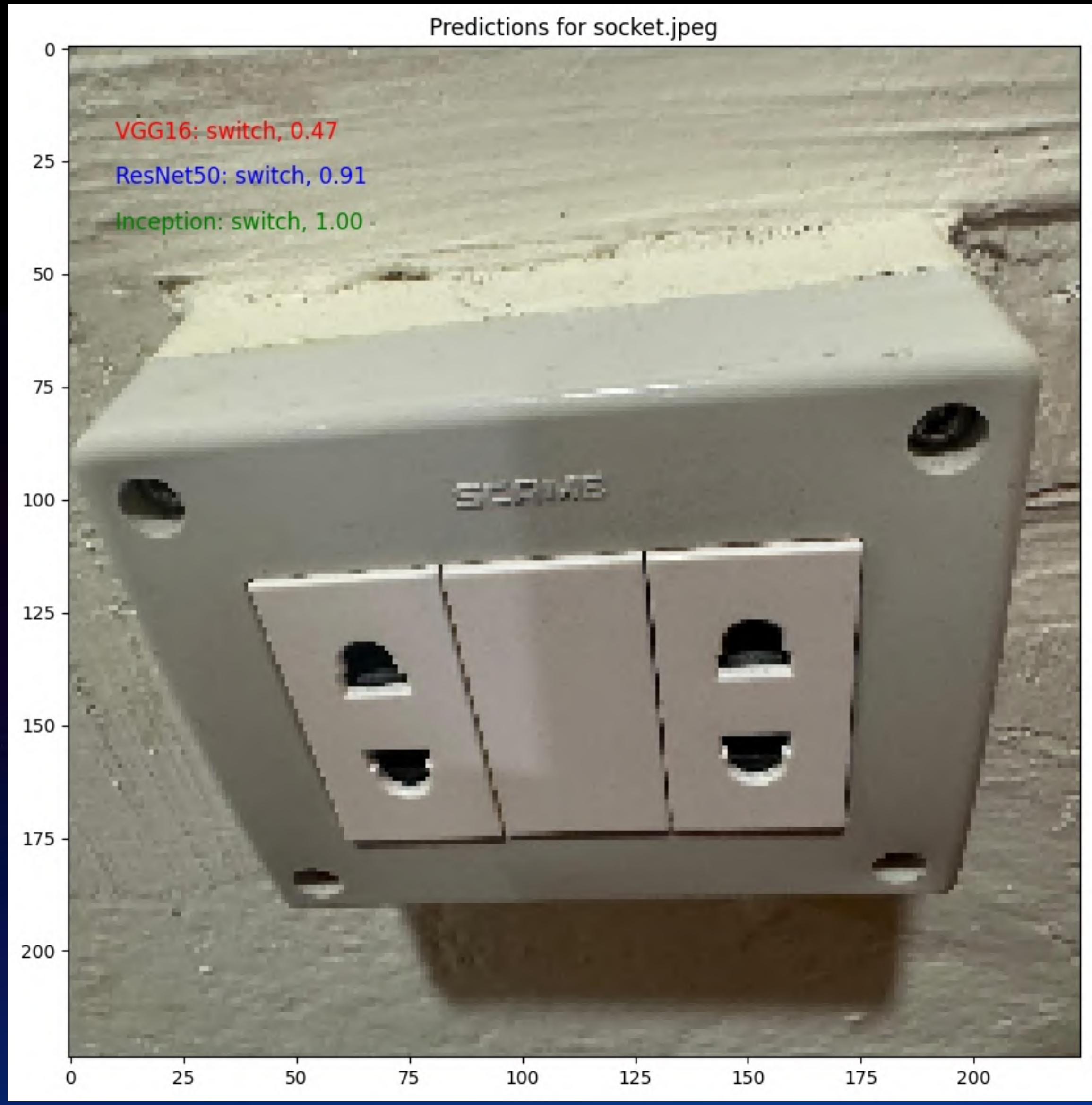




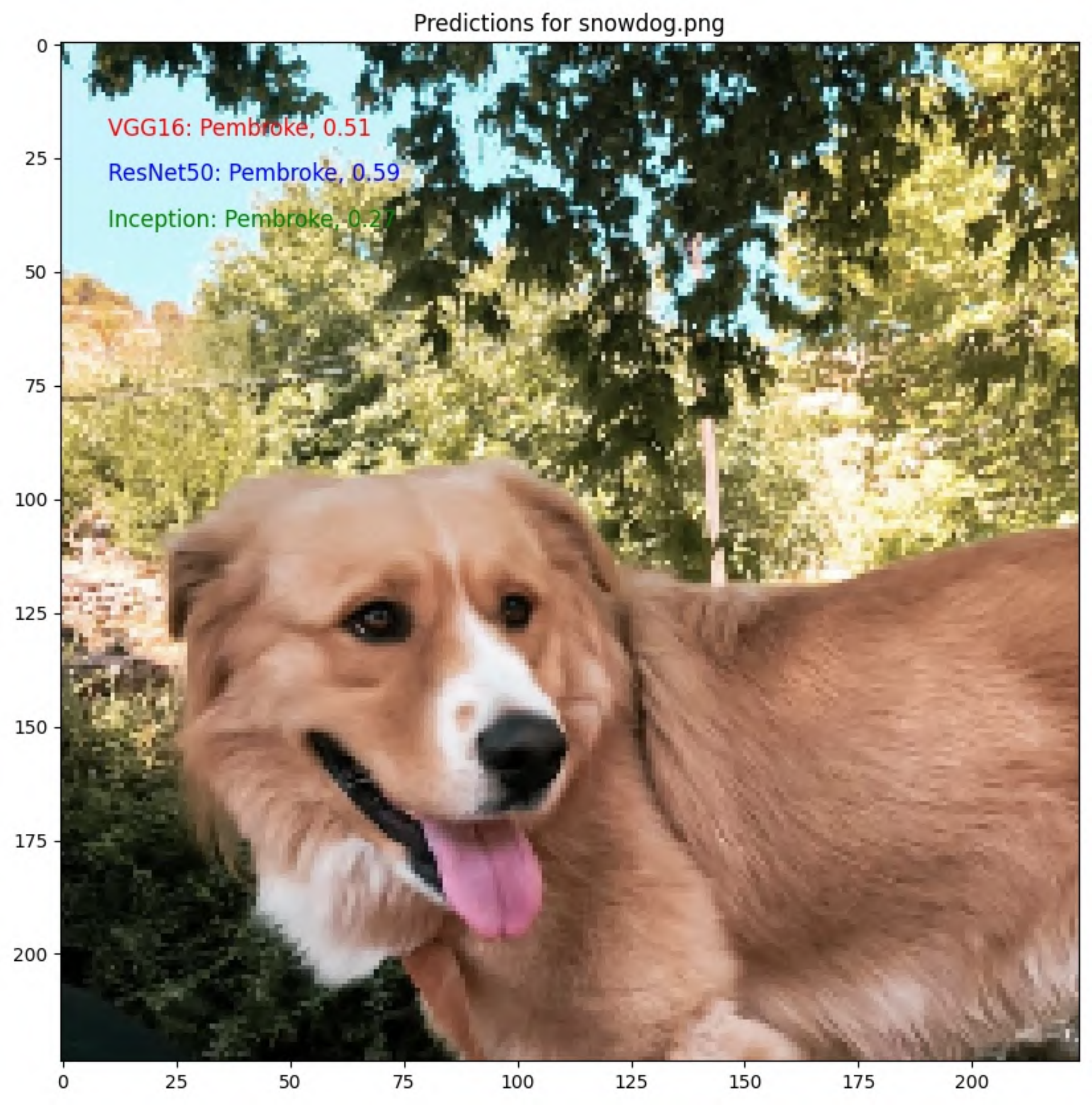
Pre-Trained Models in Keras

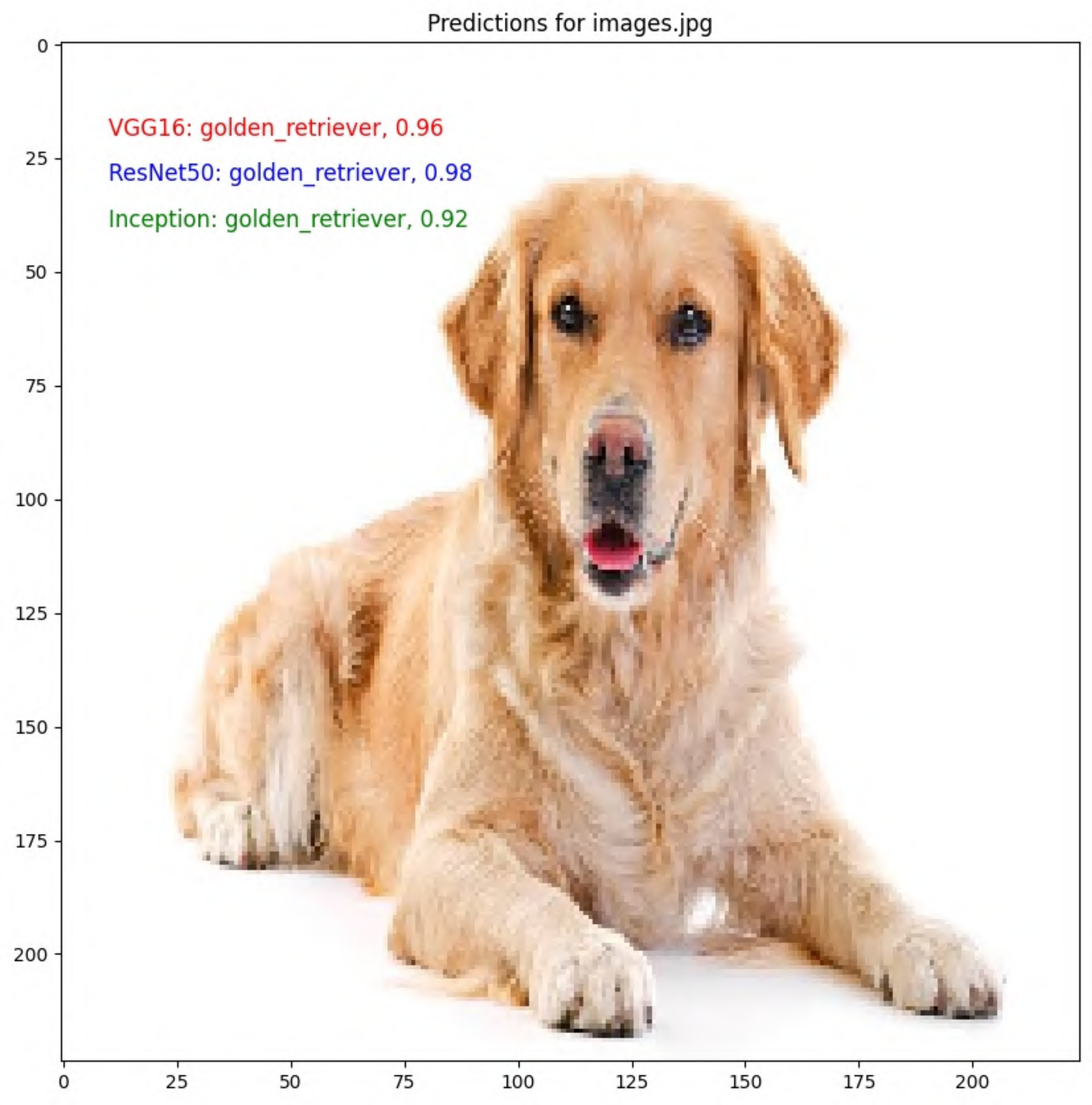


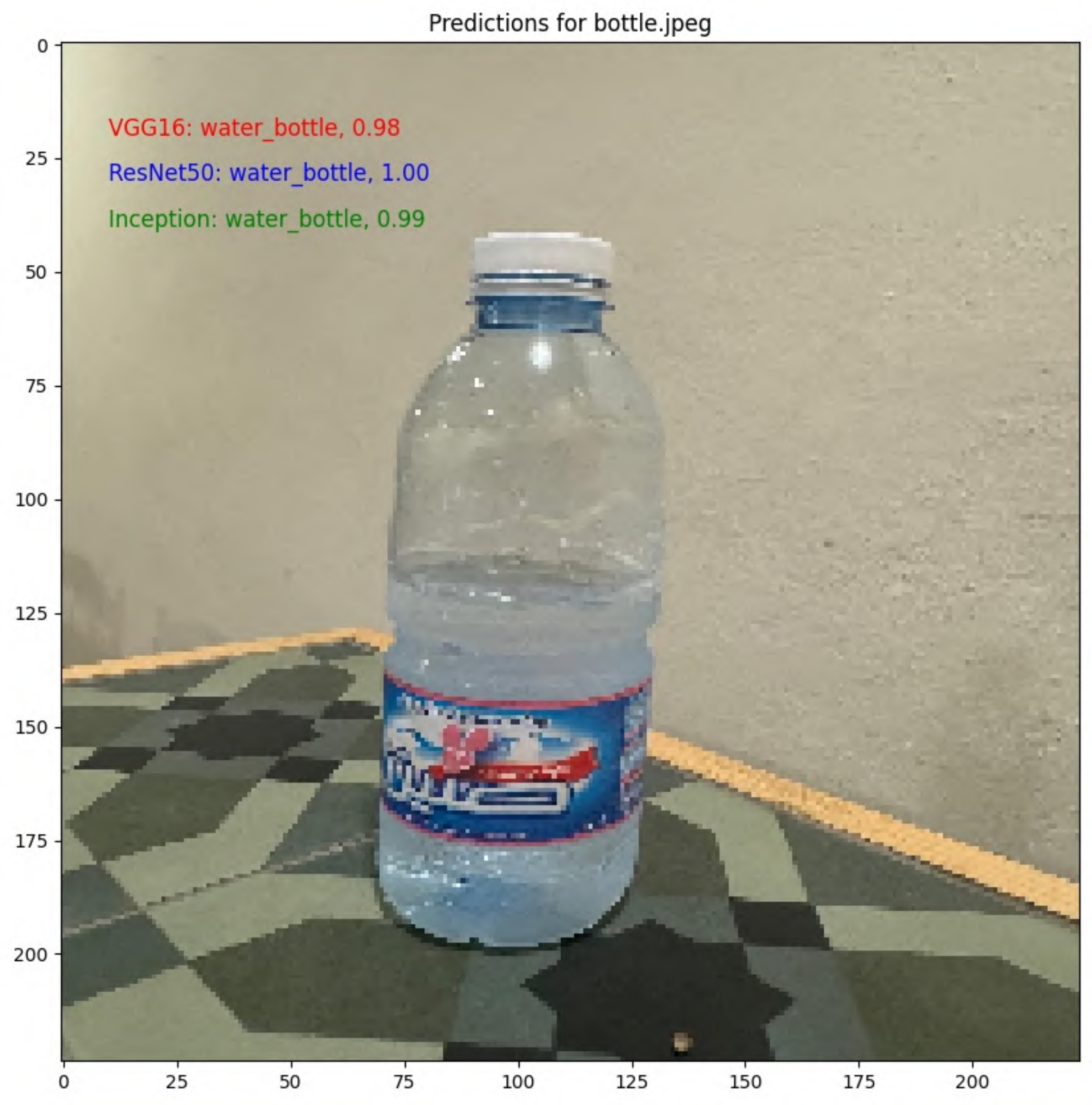




Pre-Trained Models in Keras







Finetuning VGG16 on a new set of classes

Fine-tuning is the process of taking a pre-trained machine learning model and further training it on a smaller, targeted dataset.

We will use the pre-trained model **VGG16** to train on a dataset that contains: **Our images, Barack Obama's images, and George Bush's images.**

Transfer Learning

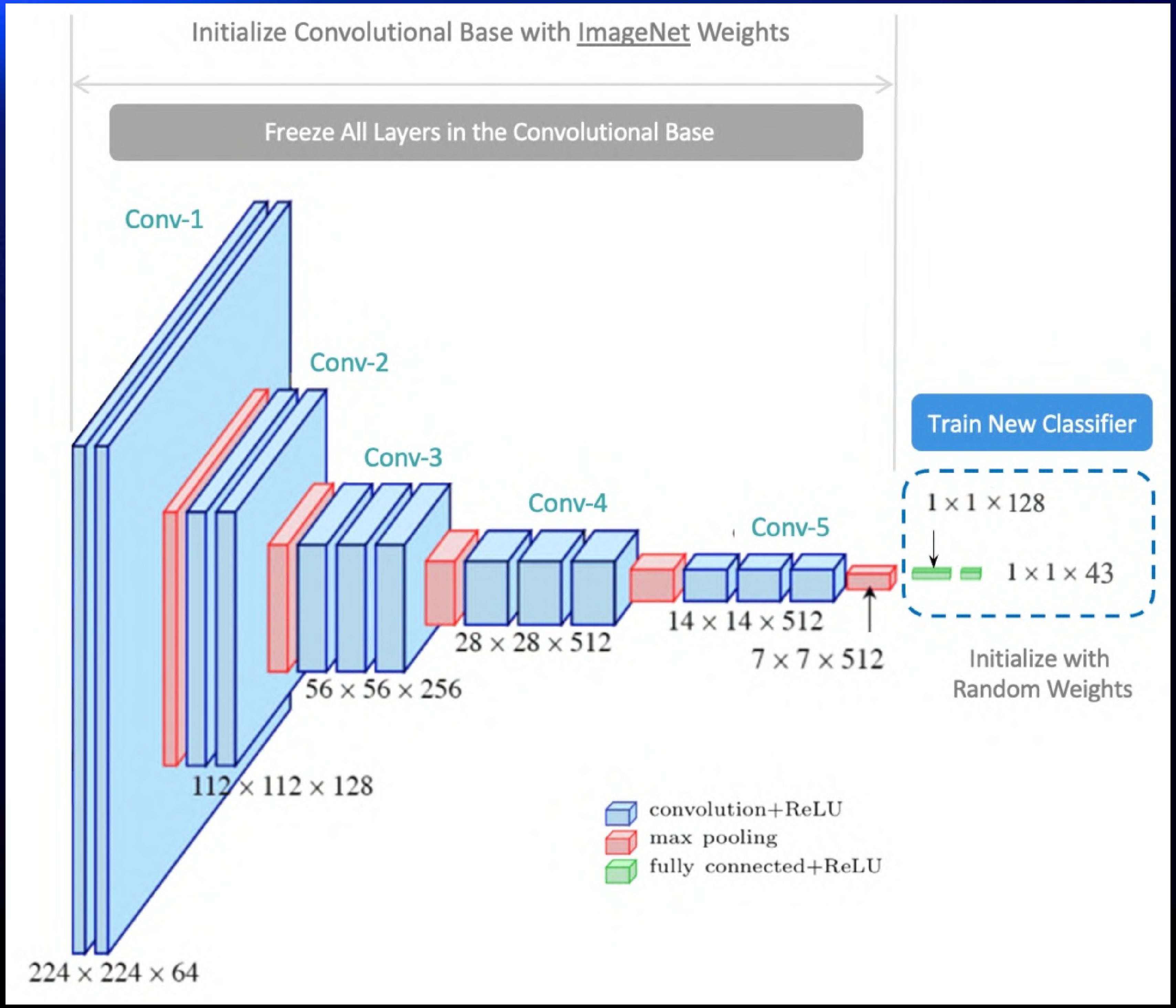
In transfer learning, a machine exploits the knowledge gained from a previous task to improve generalization about another.

Transfer Learning is a simple approach for re-purposing a pre-trained model to make predictions on a new dataset.

This approach requires **much less** data and computational resources than training from scratch.

```python
vgg_conv = vgg16.VGG16(weights='imagenet', include_top=False,
input_shape=(image_size, image_size, 3))

for layer in vgg_conv.layers[:]:
    layer.trainable = False
```

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14714688
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 1024)	25691136
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 4)	4100



Try Pitch

Transfer Learning

This code is used to resize all images in the dataset to 224x224 pixels and crop only the face.

```python
def resize_image_to_face(input_image_path, output_image_path, target_size=(224, 224)):
    # Load the image using OpenCV
    image = cv2.imread(input_image_path)

    # Convert the image to grayscale (required for face detection)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Load a pre-trained face detector model (Haar cascade)
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

    # Detect faces in the image
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30),
                                          flags=cv2.CASCADE_SCALE_IMAGE)

    if len(faces) == 0:
        print(f"No faces found in {input_image_path}. Skipping image.")
        return

    # Focus on the first detected face
    (x, y, w, h) = faces[0]

    # Crop the image around the first face
    face_cropped = image[y:y + h, x:x + w]

    # Convert the cropped BGR face to RGB before converting to a PIL Image
    face_cropped_rgb = cv2.cvtColor(face_cropped, cv2.COLOR_BGR2RGB)

    # Convert the RGB cropped face to a PIL Image for easier manipulation
    pil_image = Image.fromarray(face_cropped_rgb)

    # Resize the image to the target size using high-quality resampling
    resized_image = pil_image.resize(target_size, Image.Resampling.LANCZOS)

    # Save the resized image
    resized_image.save(output_image_path)
```

This code is used to resize all images in the dataset to 224x224 pixels and crop only the face.

```
def process_directory(input_dir, output_dir, target_size=(224, 224)):
    # Check if output directory exists, create if not
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    # Process each file in the directory
    for filename in os.listdir(input_dir):
        if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
            input_image_path = os.path.join(input_dir, filename)
            output_image_path = os.path.join(output_dir, filename)
            resize_image_to_face(input_image_path, output_image_path, target_size)


input_directory = './Barack_Obama'
output_directory = './barack-obama'
process_directory(input_directory, output_directory)
```

Each class contains 200 images for training and 50 images for testing.

George Bush

Anthony

Barack Obama

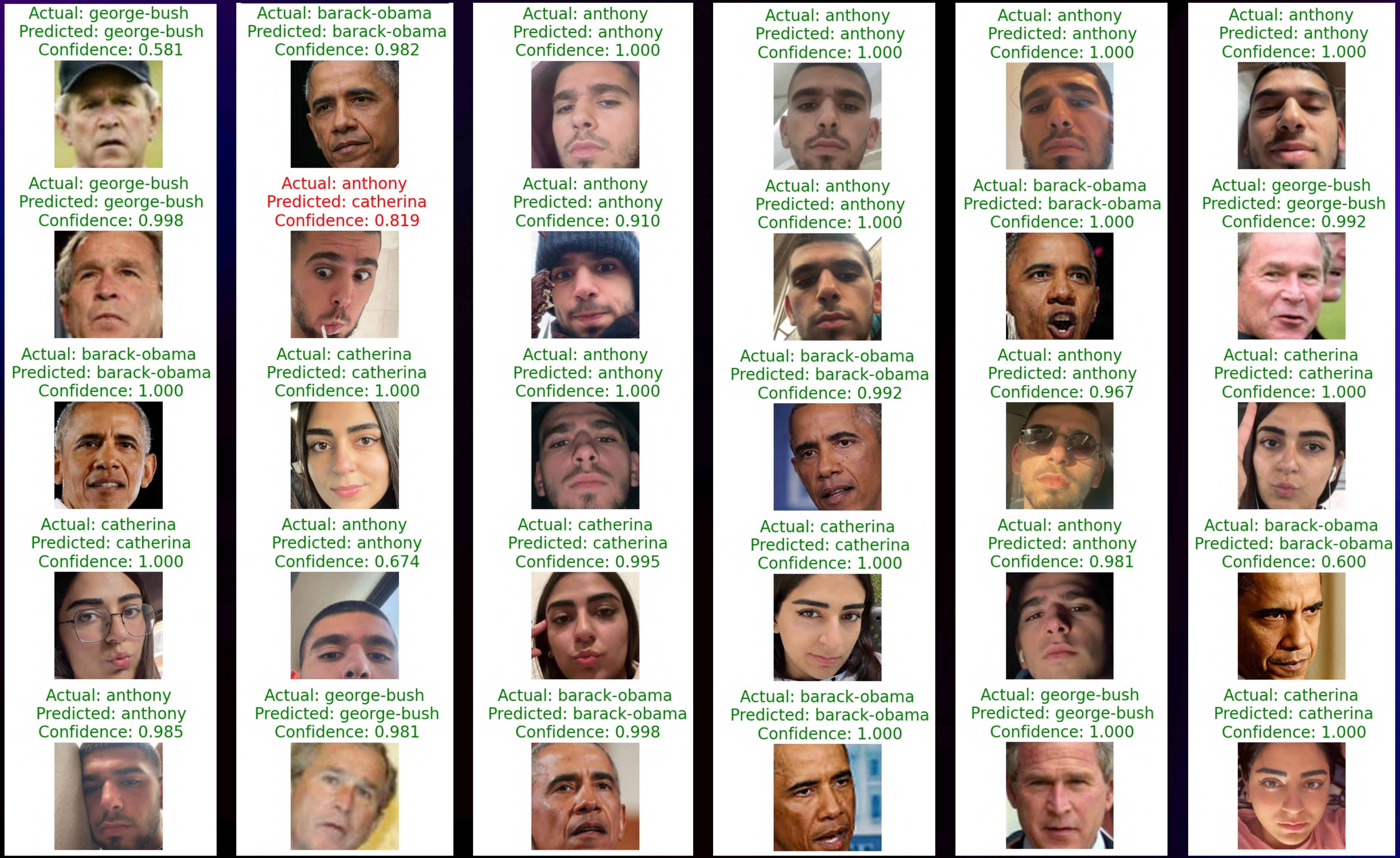
Catherina

Batch sizes:
Training: 100
Testing:10

Learning Rate:
1e-4

Training Accuracy:
99.87%

Testing Accuracy:
98.50%



OpenCV Code

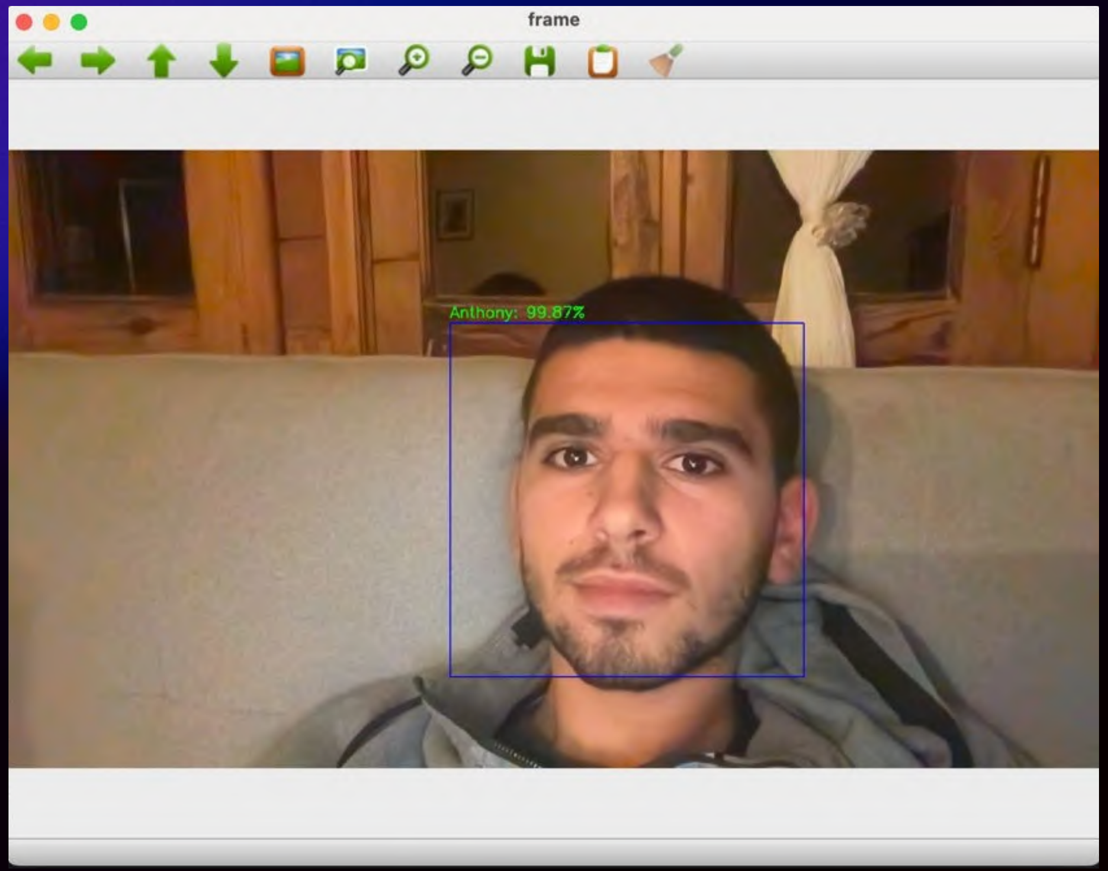
```python
import cv2
import numpy as np
import tensorflow as tf

# Load the trained model
model = tf.keras.models.load_model('model.h5')

# Define class labels based on your training directory structure
class_labels = ['Anthony', 'Obama', 'Catherina', 'Bush']

# Start video capture
cap = cv2.VideoCapture(0, cv2.CAP_ANY)

# Load the Haar Cascade for face detection
cascade_path = "./haarcascade_frontalface_default.xml"
face_cascade = cv2.CascadeClassifier(cascade_path)

# Resize windows to a good size
cv2.namedWindow('frame', cv2.WINDOW_NORMAL)
cv2.resizeWindow('frame', 800, 600)
```

```python
while True:
    ret, frame = cap.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for (x, y, w, h) in faces:
        face_roi = frame[y:y + h, x:x + w]
        face_roi = cv2.resize(face_roi, (224, 224))  # Resize to the expected input size of the model
        face_roi = np.array(face_roi) / 255.0  # Normalize pixel values if needed
        face_roi = np.expand_dims(face_roi, axis=0)  # Add the batch dimension

        predictions = model.predict(face_roi)
        best_class_idx = np.argmax(predictions)
        best_class = class_labels[best_class_idx]
        confidence = np.max(predictions)

        cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)
        text = '{}: {:.2f}%'.format(best_class, confidence * 100)
        cv2.putText(frame, text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

    cv2.imshow('frame', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the capture
cap.release()
cv2.destroyAllWindows()
```

OpenCV Results

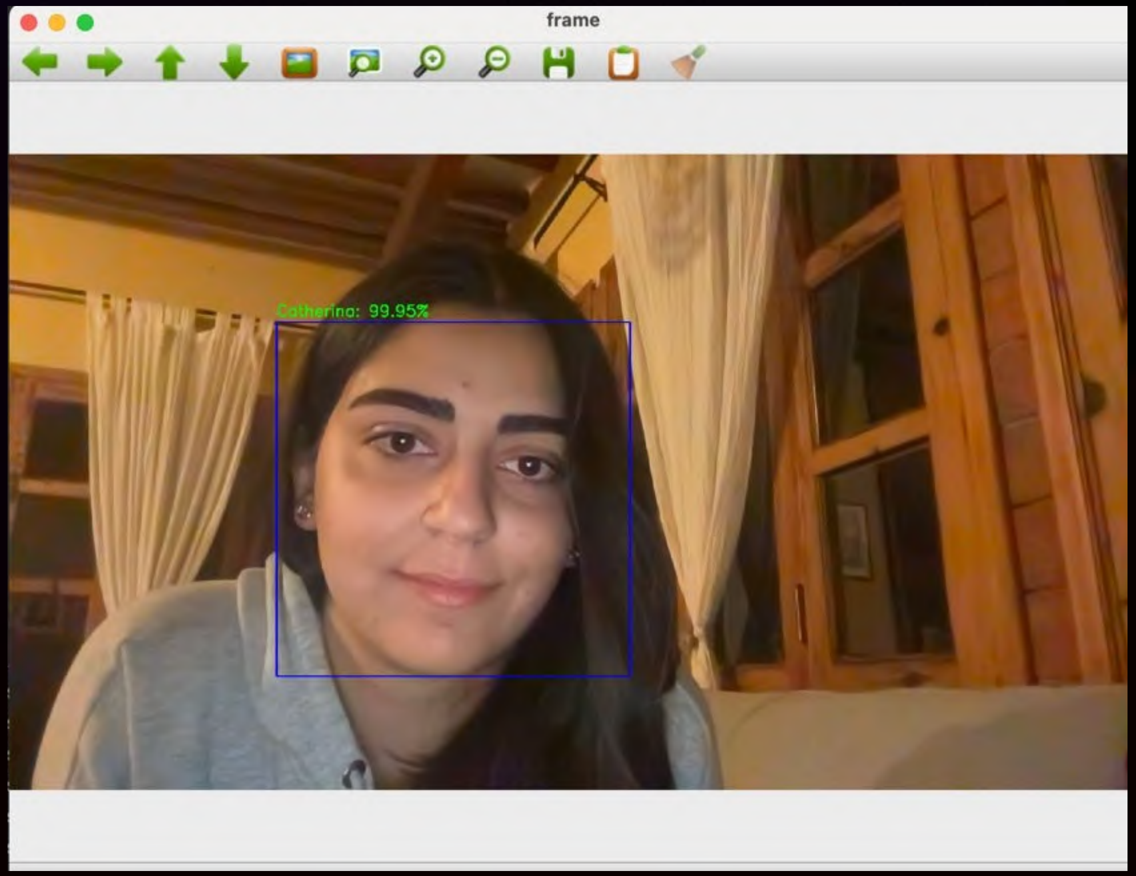
Anthony Alone

Confidence: 99.87%



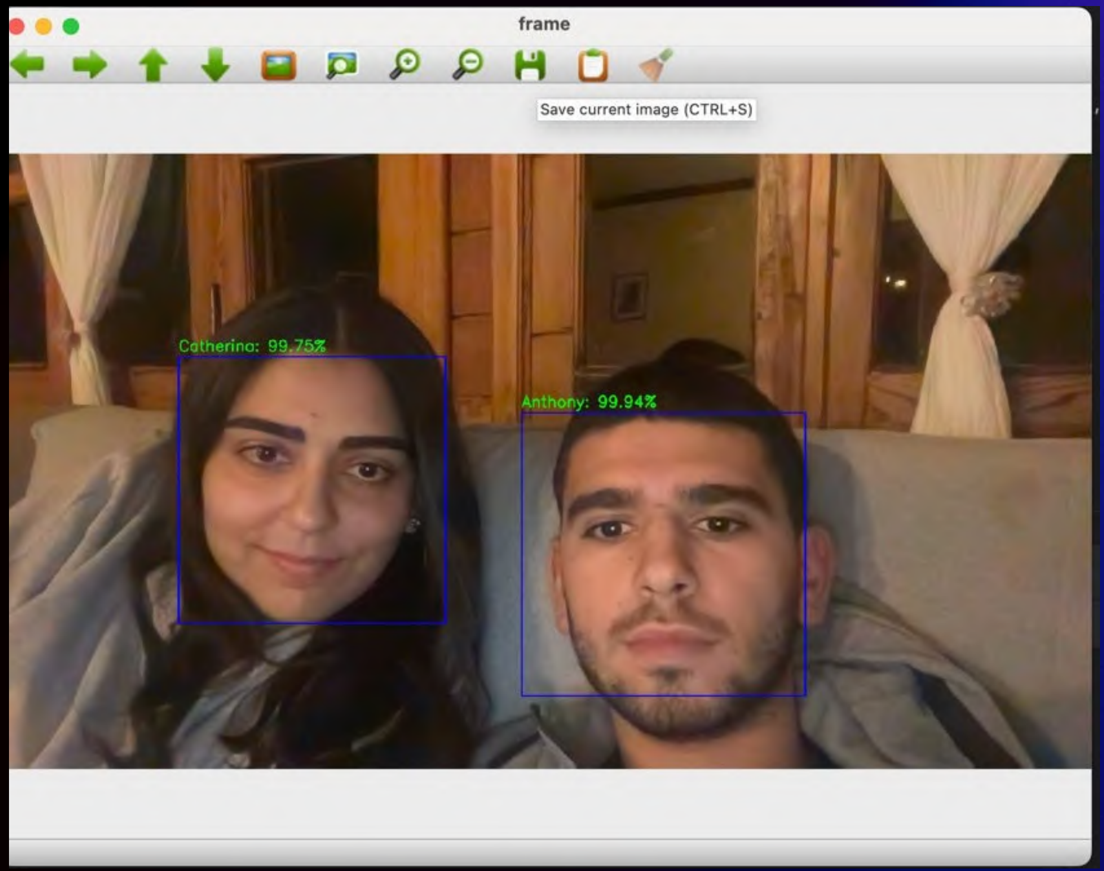
Catherina Alone

Confidence: 99.95%



Catherina & Anthony

Confidence: 99.75% & 99.94%



Anthony **El Chemaly - 20210079**
Catherina **El Khoury - 202101204**

USEK

THANK YOU!