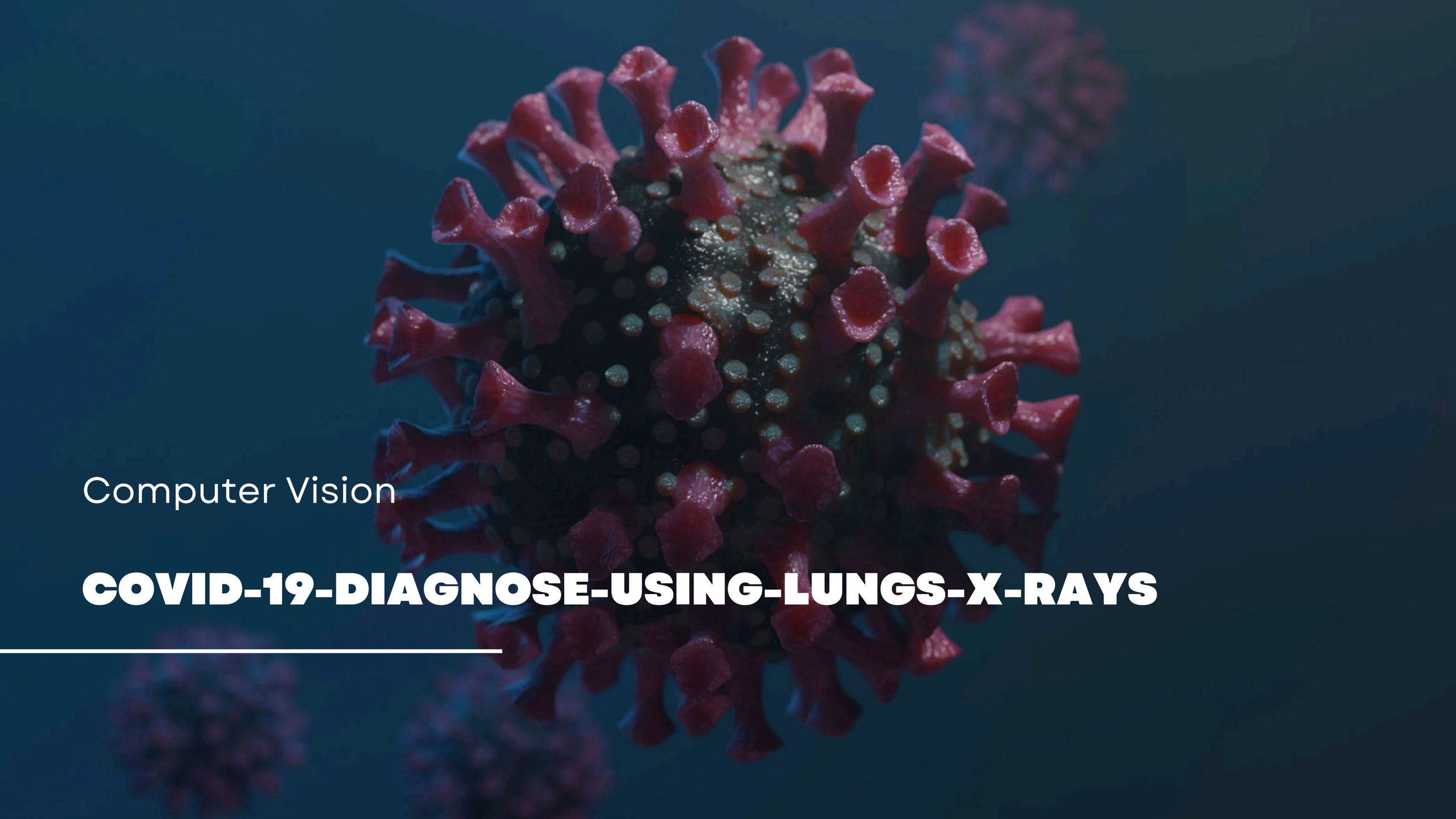


Computer Vision

COVID-19-DIAGNOSE-USING-LUNGS-X-RAYS

OUTLINE

Phase 1: Easy Round – **CNN MODEL**

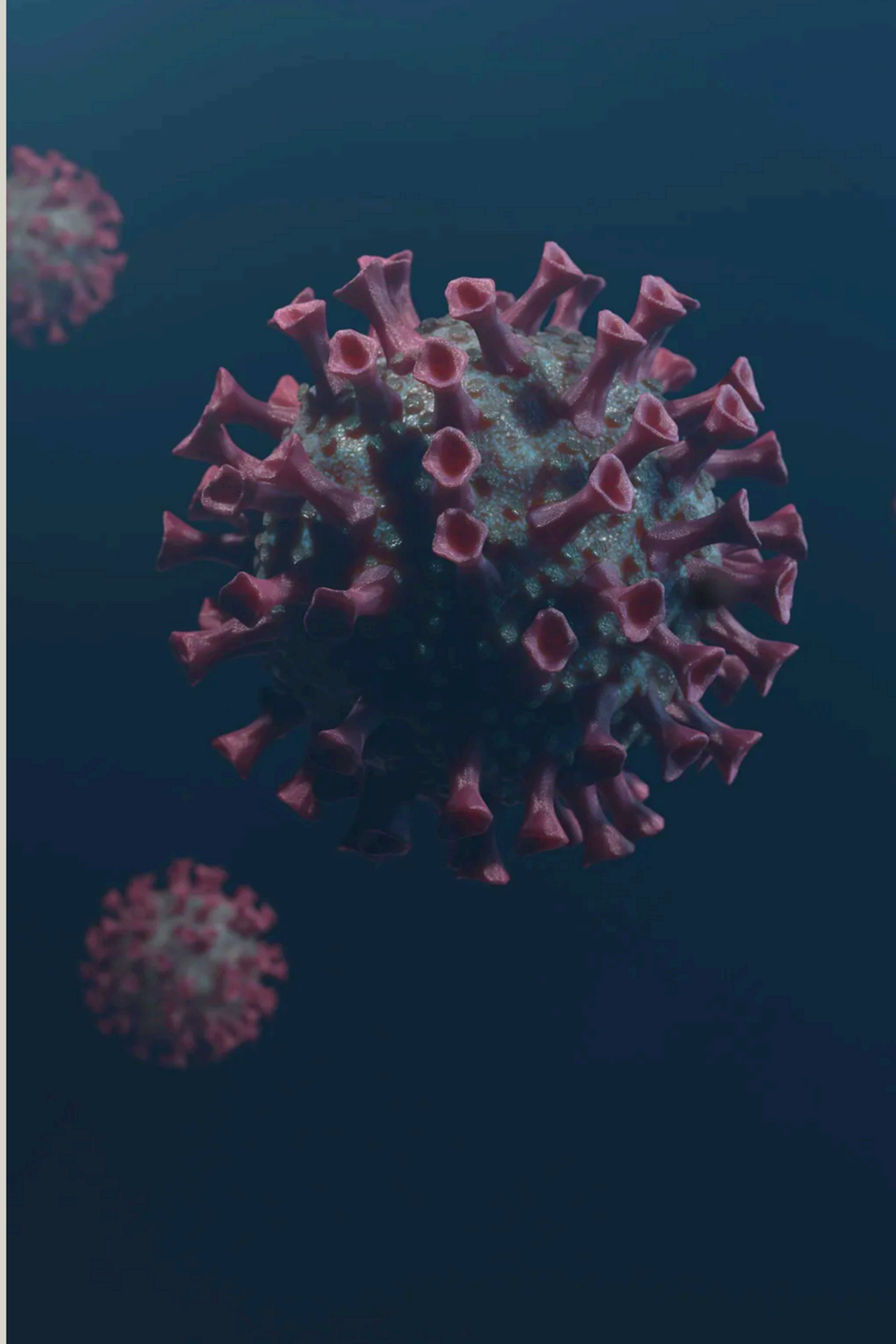
Phase 2: Medium Round – **CNN MODEL-PRE_TRAINED**

Phase 3: Hard Round-**Integration on web**


PHASE 1:
CNN-MODEL

Libraries

```
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPool2D
from keras.layers import Flatten
from keras.layers import Dense
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.preprocessing import image
```



preprocessing

testing

tranining

```
testing_data_generator = ImageDataGenerator(
    rescale=1.0/255
)
```

```
training_data_generator = ImageDataGenerator(
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)
```

Click to collapse the range.

```python
training_dataset = training_data_generator.flow_from_directory(
    'Covid-19/train',
    target_size=(64,64),
    batch_size=32,
    class_mode='categorical'
)
```

Python

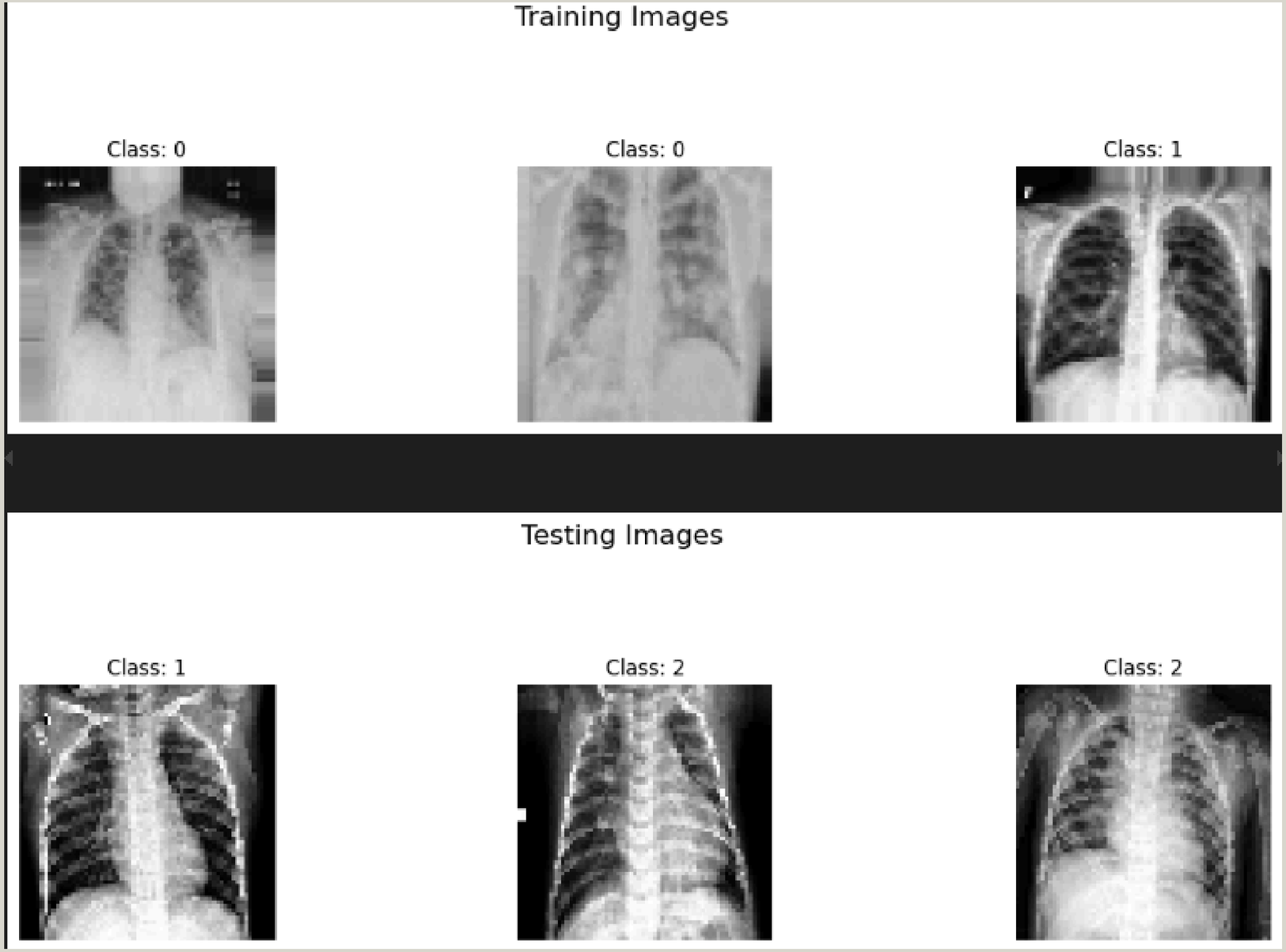
```
Found 251 images belonging to 3 classes.
```

```python
testing_dataset = testing_data_generator.flow_from_directory(
    'Covid-19/test',
    target_size=(64,64),
    batch_size=32,
    class_mode='categorical'
)
```

Python

```
Found 66 images belonging to 3 classes.
```

DATASET


Training Images
Class: 0
Class: 0
Class: 1
Testing Images
Class: 1
Class: 2
Class: 2

MODEL

CNN

```
classifier = Sequential()
classifier.add(Convolution2D(32,(3,3),input_shape=(64,64,3), activation='relu'))
classifier.add(MaxPool2D(pool_size=(2,2)))

classifier.add(Convolution2D(16,(3,3), activation='relu'))
classifier.add(MaxPool2D(pool_size=(2,2)))

classifier.add(Flatten())

classifier.add(Dense(units= 128, activation='relu'))
classifier.add(Dense(units= 3, activation='softmax'))

classifier.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

SUMMARY

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 62, 62, 32)	896
max_pooling2d_6 (MaxPooling2D)	(None, 31, 31, 32)	0
conv2d_7 (Conv2D)	(None, 29, 29, 16)	4,624
max_pooling2d_7 (MaxPooling2D)	(None, 14, 14, 16)	0
flatten_3 (Flatten)	(None, 3136)	0
dense_6 (Dense)	(None, 128)	401,536
dense_7 (Dense)	(None, 3)	387

Total params: 407,443 (1.55 MB)

Trainable params: 407,443 (1.55 MB)

Non-trainable params: 0 (0.00 B)

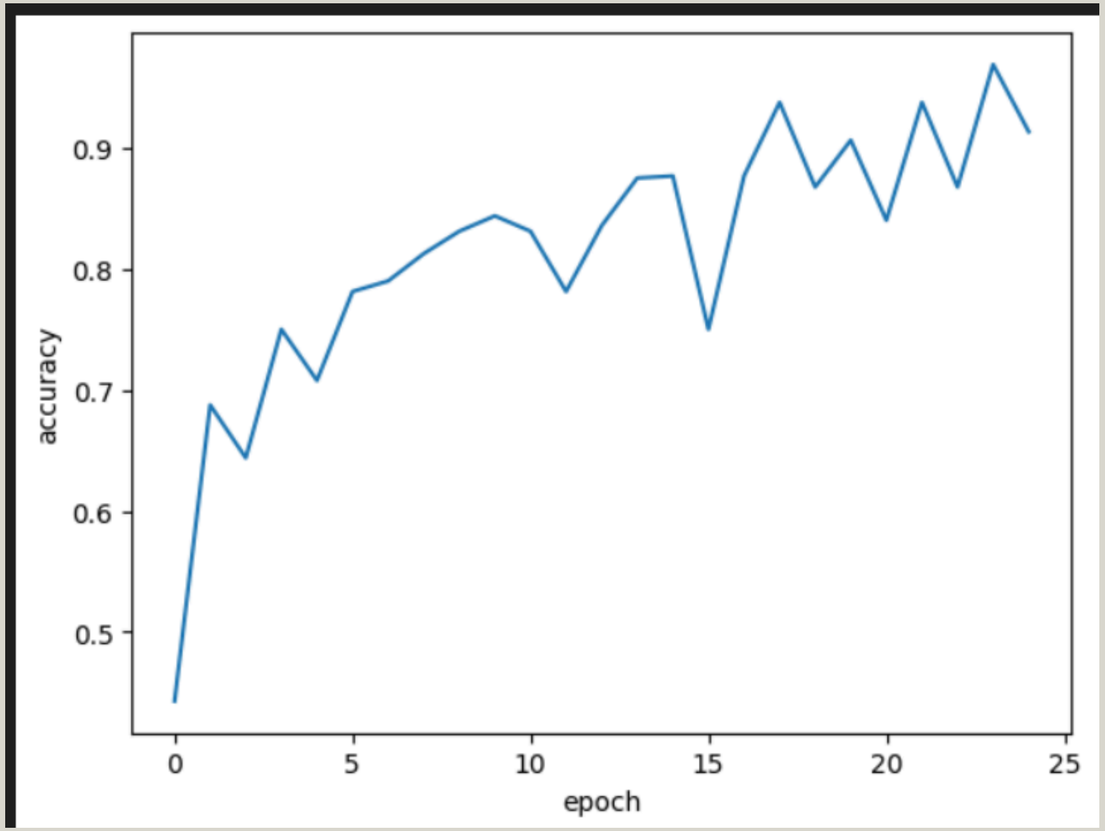
TRAINING

```
history = classifier.fit(
    training_dataset,
    steps_per_epoch=int(251/32),
    epochs=25,
    validation_data=testing_dataset,
    validation_steps= int(66/32)
)
```

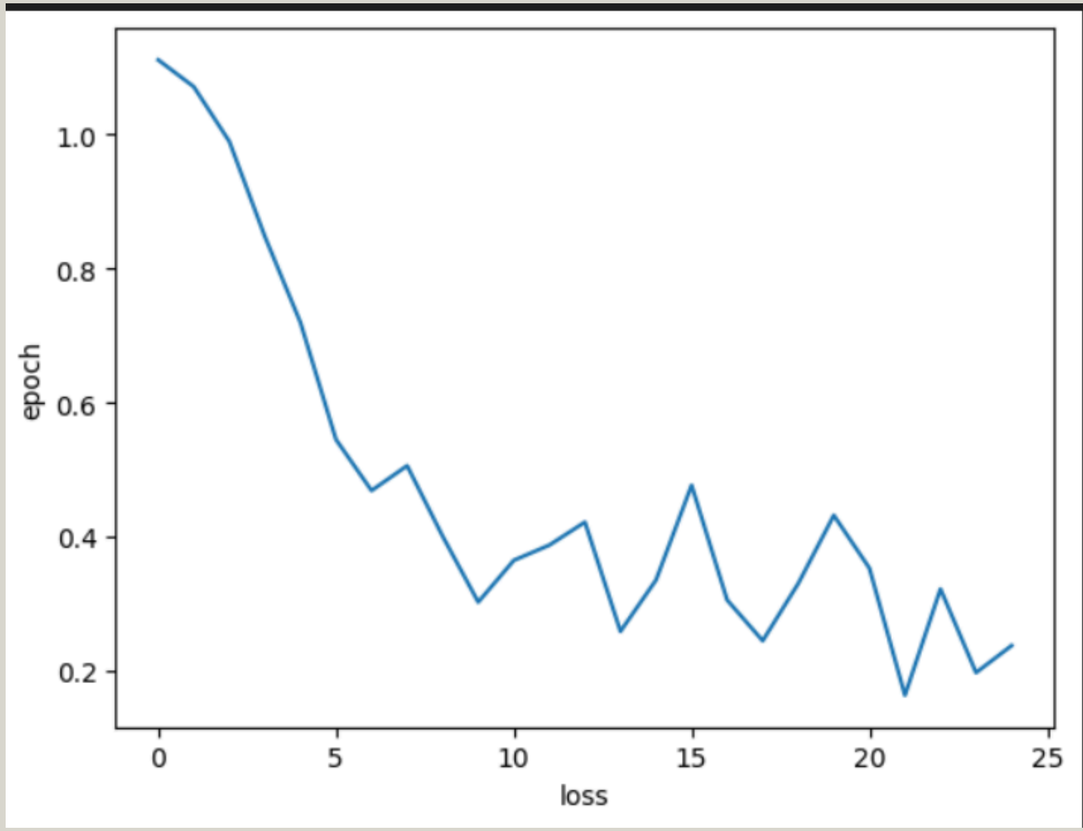
Accuracy:

```
Training Accuracy: 0.9083665609359741
Test Accuracy: 0.9242424368858337
```

Visualization



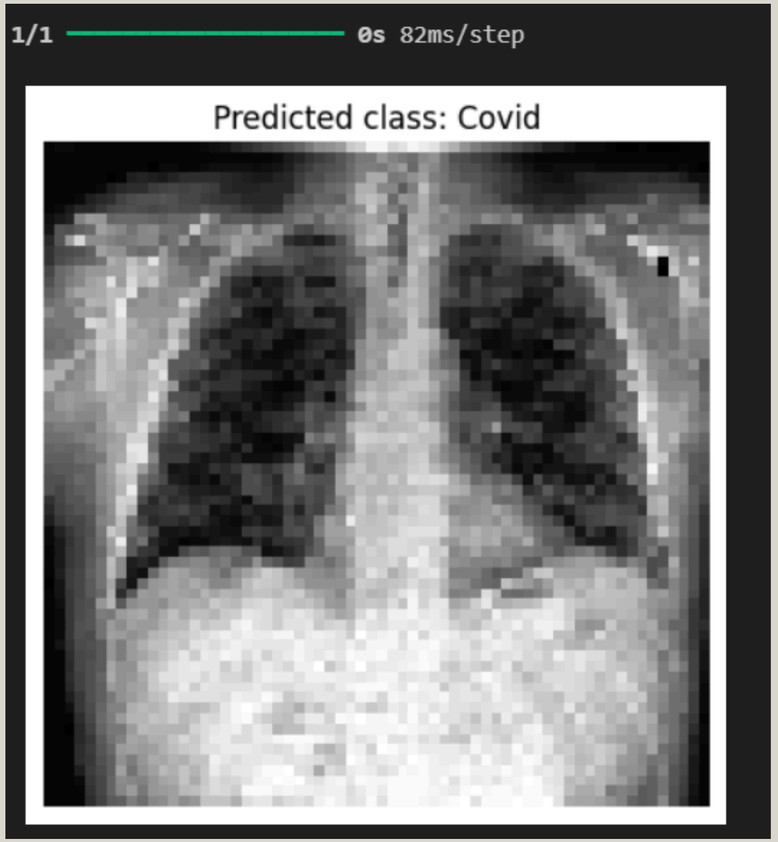
validation accuracy

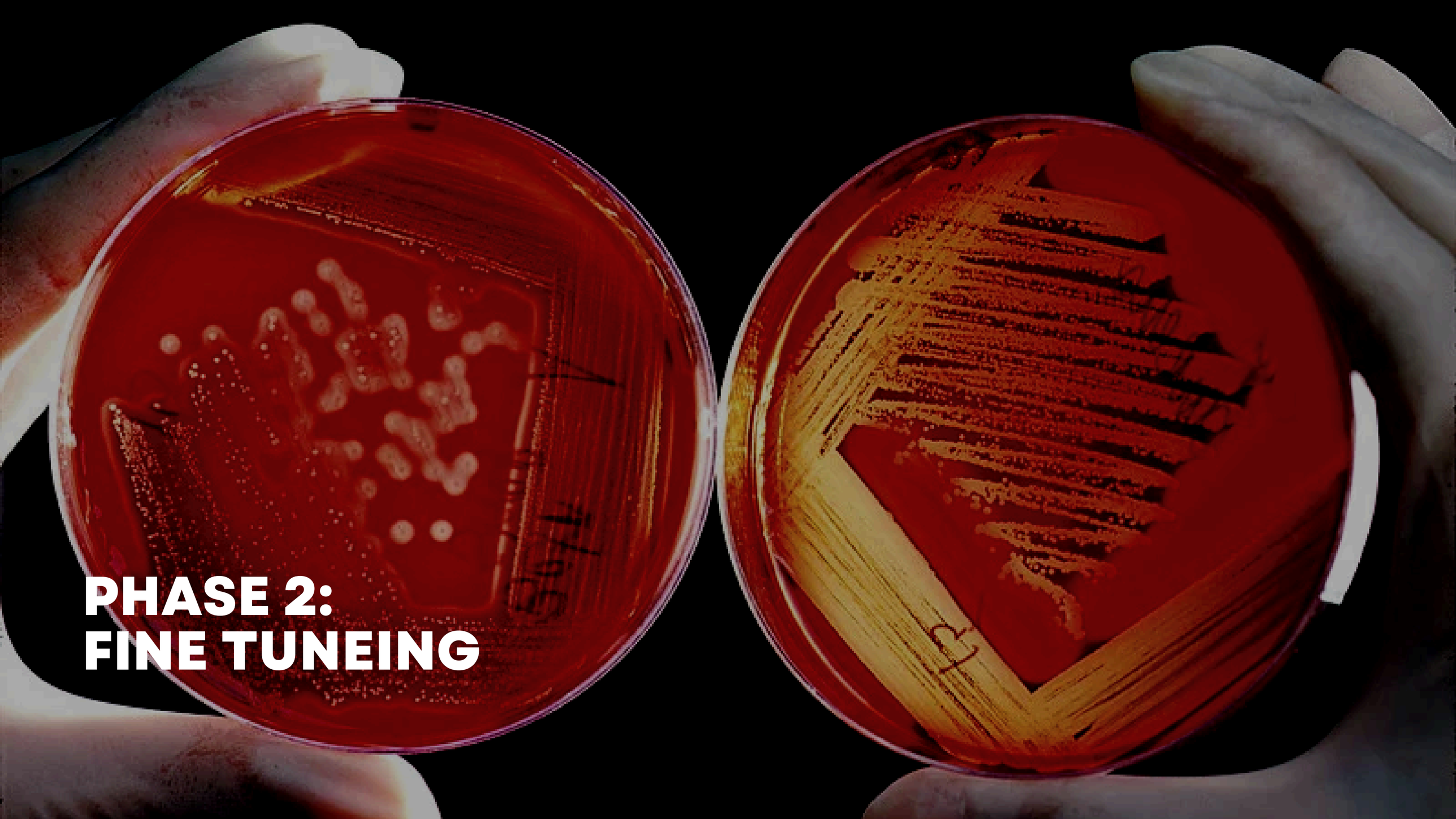


validation loss

prediction

```
img_path = r'Covid-19\test\Covid\0100.jpeg'
img = image.load_img(img_path, target_size=(64, 64))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = img_array / 255.0  # Normalize the image as per t

# Predict the class label using the ResNet50 model
prediction_cnn = classifier.predict(img_array)
predicted_class_cnn = np.argmax(prediction_cnn)
class_labels = ["Covid", "Normal", "Viral Pneumonia"]
predicted_label_resnet50 = class_labels[predicted_class_cnn]

# Display the image
plt.imshow(img)
plt.axis('off')  # Hide axes
plt.title("Predicted class: " + predicted_label_resnet50)  #
plt.show()
```




PHASE 2:
FINE TUNEING

Libraries

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from sklearn.metrics import precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import numpy as np
```

FINE TUNING

```
# Define your CNN architecture
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 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(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

RESNET_50

```python
# Load the ResNet50 model with pre-trained weights, excluding the top layer
base_model_resnet50 = ResNet50(weights='imagenet', include_top=False, input_shape

x = base_model_resnet50.output
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(3, activation='softmax')(x)

model_resnet50 = Model(inputs=base_model_resnet50.input, outputs=predictions)

for layer in base_model_resnet50.layers:
    layer.trainable = False
```

Python

+ Code + Markdown

VVG16

```python
base_model_vgg16 = VGG16(weights='imagenet', include_top=False, input_shape=(64,

x = base_model_vgg16.output
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
predictions_vgg16 = Dense(3, activation='softmax')(x)

model_vgg16 = Model(inputs=base_model_vgg16.input, outputs=predictions_vgg16)

for layer in base_model_vgg16.layers:
    layer.trainable = False
```

Python

RESULTS

FINE TUNED_cnn

```
Training Accuracy: 0.9442231059074402
Test Accuracy: 0.9545454382896423
```

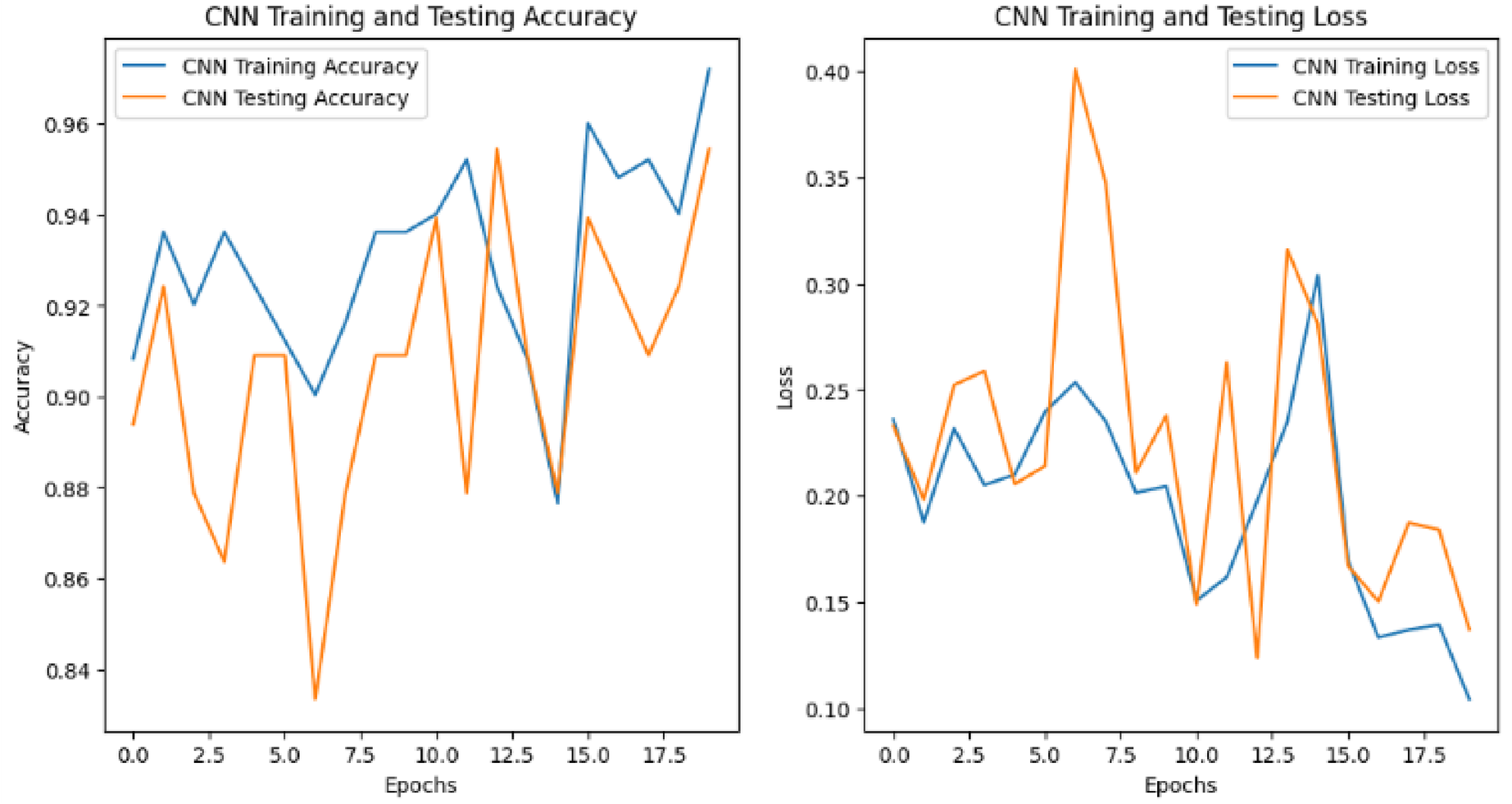
RESNET_50

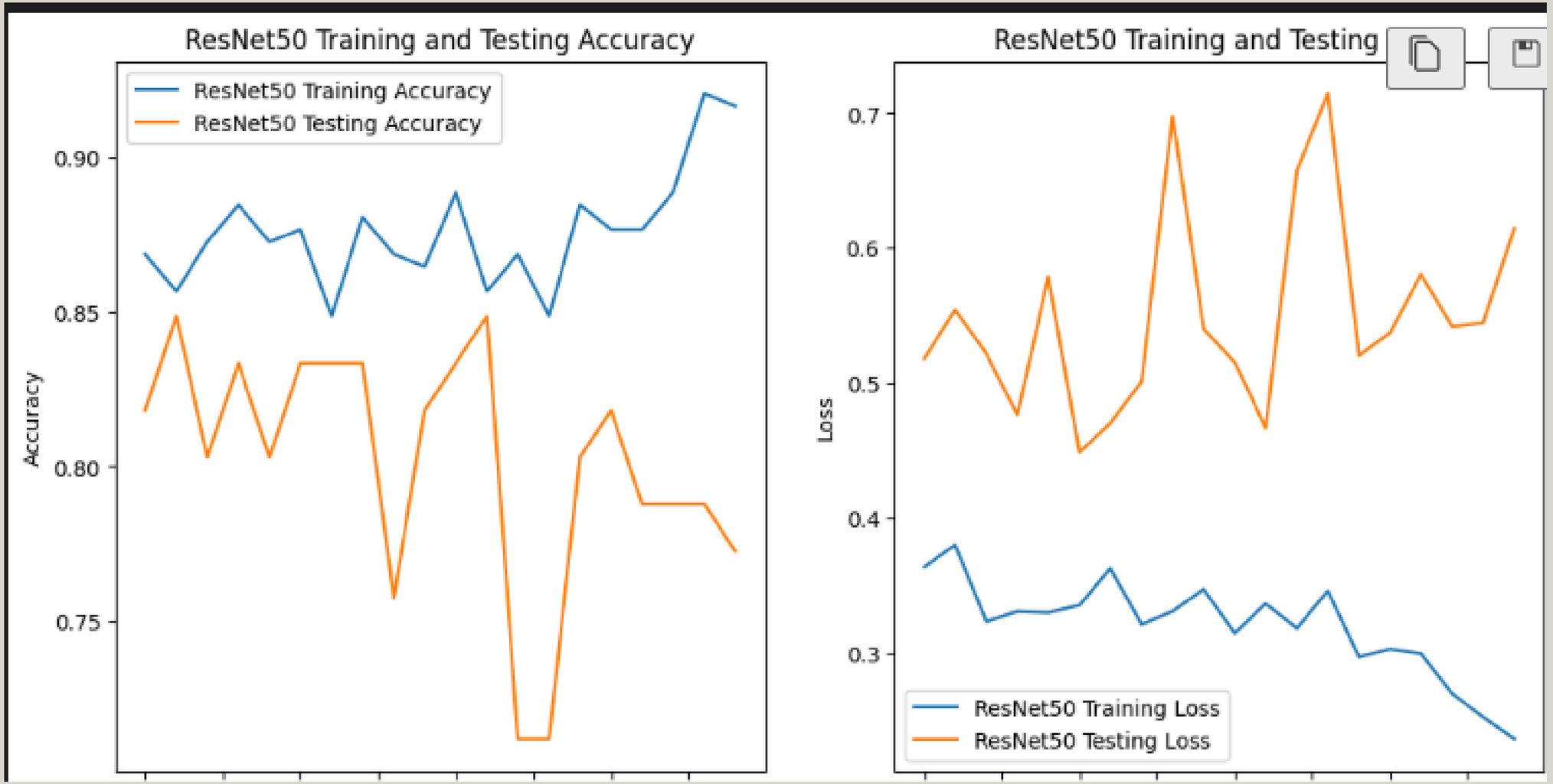
```
ResNet50 Training Accuracy: 0.8764940500259399
ResNet50 Test Accuracy: 0.7727272510528564
```

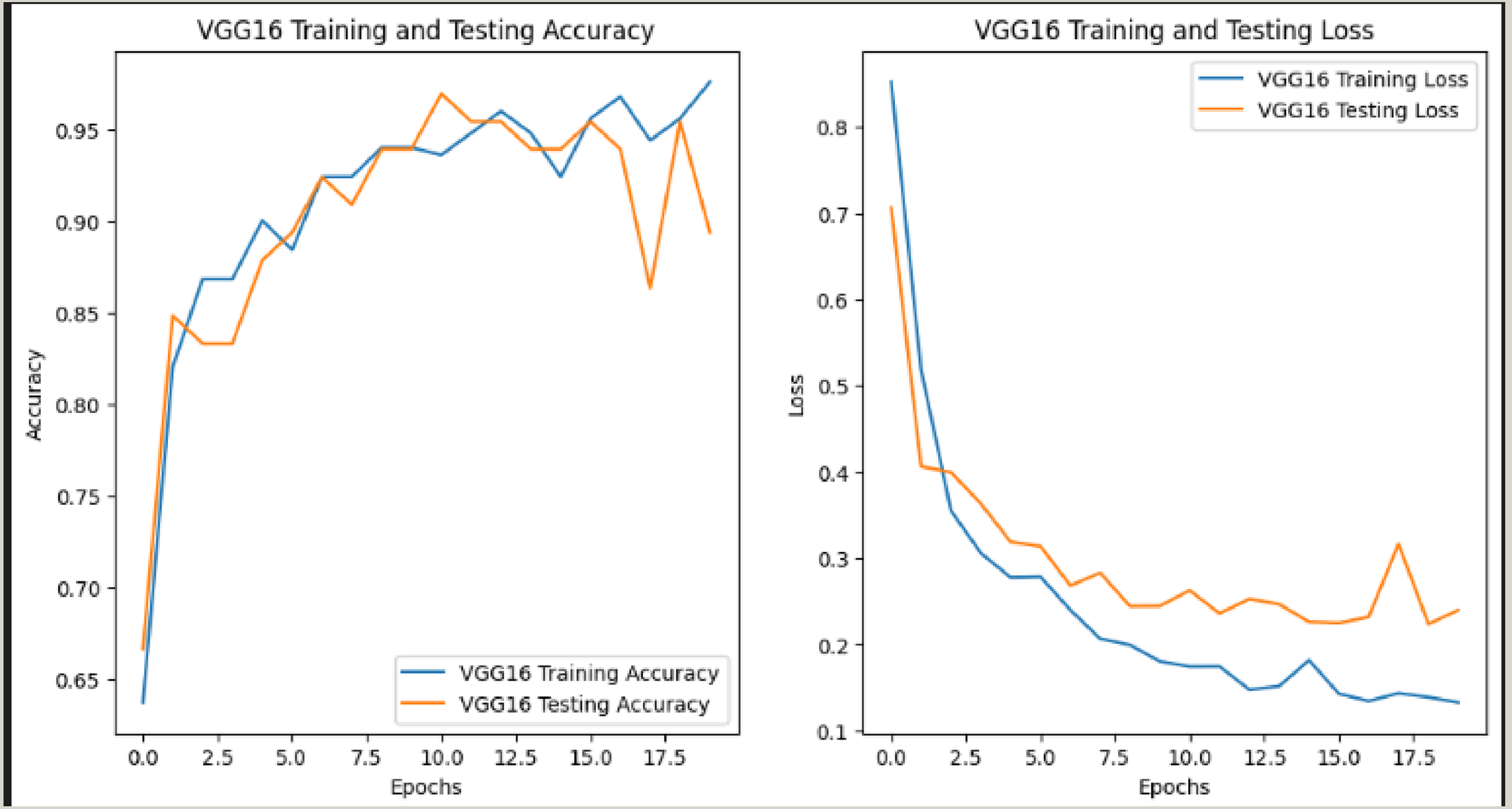
VGG_16

```
VGG16 Training Accuracy: 0.9721115827560425
VGG16 Test Accuracy: 0.893939375877380
```

Visualization




ResNet50 Training and Testing Accuracy
ResNet50 Training Accuracy
ResNet50 Testing Accuracy
Accuracy
0.90
0.85
0.80
0.75
ResNet50 Training and Testing
Loss
0.7
0.6
0.5
0.4
0.3
ResNet50 Training Loss
ResNet50 Testing Loss


VGG16 Training and Testing Accuracy
Accuracy
Epochs
VGG16 Training Accuracy
VGG16 Testing Accuracy
VGG16 Training and Testing Loss
Loss
Epochs
VGG16 Training Loss
VGG16 Testing Loss

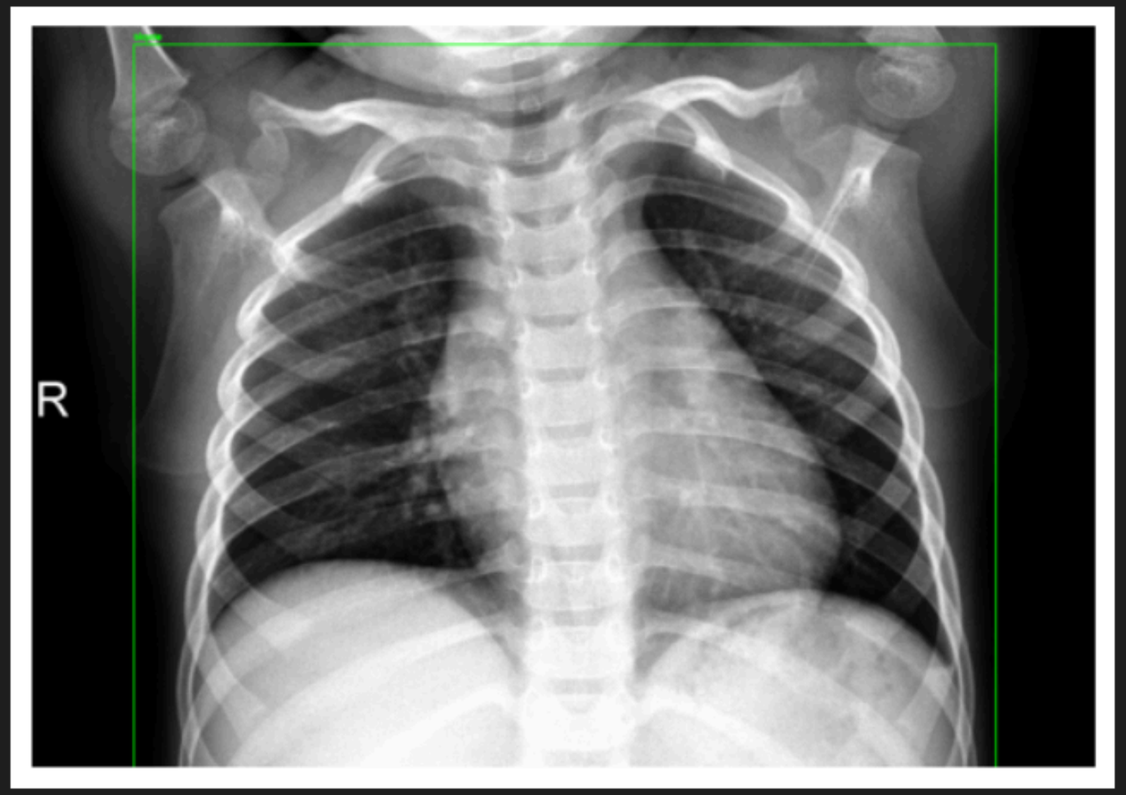

YOLO:
IDENTIFICATION

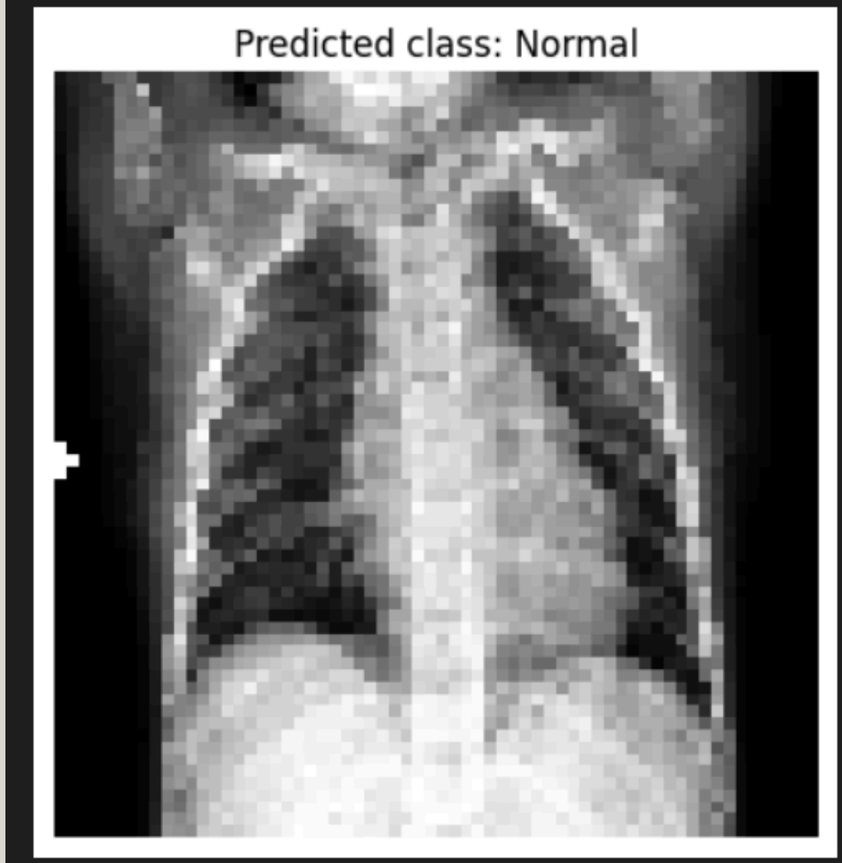
yolo interaction with vgg16

```python
# Function to detect defects using YOLO and classify using VGG16
def detect_defects_yolo(image_path, net, output_layers, classes, vgg_model):
    img = cv2.imread(image_path)
    height, width, channels = img.shape

    blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
    net.setInput(blob)
    outs = net.forward(output_layers)

    class_ids = []
    confidences = []
    boxes = []

    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:  # Confidence threshold
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                boxes.append([x, y, w, h])
                confidences.append(float(confidence))
                class_ids.append(class_id)
```

```python
    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
    detected_regions = []
    detected_boxes = []
    detected_labels = []

    for i in range(len(boxes)):
        if i in indexes:
            x, y, w, h = boxes[i]
            detected_regions.append(img[y:y+h, x:x+w])  # Extract region of interest
            detected_boxes.append((x, y, x+w, y+h))  # Store box coordinates

            # Perform classification using VGG16 model
            region_img = cv2.resize(img[y:y+h, x:x+w], (64, 64))  # Resize image to VGG16 in
            region_img = cv2.cvtColor(region_img, cv2.COLOR_BGR2RGB)  # Convert to RGB (VGG1
            region_img = region_img.astype(np.float32) / 255.0  # Normalize

            # Predict using VGG16 model
            prediction = vgg_model.predict(np.expand_dims(region_img, axis=0))
            predicted_class = np.argmax(prediction)
            label = classes[predicted_class]
            detected_labels.append(label)

            # Draw rectangle and label on the original image
            cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
            cv2.putText(img, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0),

    return img, detected_regions, detected_boxes, detected_labels
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 237ms/step
```





+ Code + Markdown

Flask

Detection with flask

COVID-19 Detection App

Choose file No file chosen

Upload Image

Uploaded Image: 0108.jpeg

Model Predictions:

CNN: Covid
ResNet50: Normal
VGG16: Covid

Thank you for you attention!