```
!pip install tensorflow opencv-python

Requirement already satisfied: tensorflow in
/usr/local/lib/python3.10/dist-packages (2.15.0)
Requirement already satisfied: opencv-python in
/usr/local/lib/python3.10/dist-packages (4.8.0.76)
Requirement already satisfied: absl-py>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1
in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes~=0.2.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.25.2)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!
=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.20.3)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.37.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.64.1)
Requirement already satisfied: tensorboard<2.16,>=2.15 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.2)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0
in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: keras<2.16,>=2.15.0 in
```

```
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0-
>tensorflow) (0.43.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15-
>tensorflow) (2.27.0)
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15-
>tensorflow) (1.2.0)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15-
>tensorflow) (3.6)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15-
>tensorflow) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0
in /usr/local/lib/python3.10/dist-packages (from
tensorboard<2.16,>=2.15->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15-
>tensorflow) (3.0.3)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.16,>=2.15->tensorflow) (5.3.3)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.16,>=2.15->tensorflow) (0.4.0)
Requirement already satisfied: rsa<5,>=3.1.4 in
/usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.16,>=2.15->tensorflow) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from google-auth-
oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.16,>=2.15->tensorflow) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.16,>=2.15->tensorflow) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.16,>=2.15->tensorflow) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.16,>=2.15->tensorflow) (2024.6.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1-
>tensorboard<2.16,>=2.15->tensorflow) (2.1.5)
```

```
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in
/usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1-
>google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (0.6.0)
Requirement already satisfied: oauthlib>=3.0.0 in
/usr/local/lib/python3.10/dist-packages (from requests-
oauthlib>=0.7.0->google-auth-oauthlib<2,>=0.5-
>tensorboard<2.16,>=2.15->tensorflow) (3.2.2)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
import os
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator


# Define the directories for input images, resized images, and
augmented images
input_dir = '/content/drive/MyDrive/Injured dog'
resized_dir = '/content/drive/MyDrive/resized_images'
augmented_dir = '/content/drive/MyDrive/augmented_images'

# Ensure the directories exist
os.makedirs(resized_dir, exist_ok=True)
os.makedirs(augmented_dir, exist_ok=True)

# Resize images to (200, 200)
for filename in os.listdir(input_dir):
    if filename.endswith(('.png', '.jpg', '.jpeg')):
        img_path = os.path.join(input_dir, filename)
        img = cv2.imread(img_path)
        resized_img = cv2.resize(img, (200, 200))
        cv2.imwrite(os.path.join(resized_dir, filename), resized_img)

# Initialize ImageDataGenerator for data augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Load resized images
```

```
resized_images = []
for filename in os.listdir(resized_dir):
    if filename.endswith(('.png', '.jpg', '.jpeg')):
        img_path = os.path.join(resized_dir, filename)
        img = cv2.imread(img_path)
        resized_images.append(img)

resized_images = np.array(resized_images)

# Number of augmented images to generate
num_augmented_images = 2500  # Change this value to generate a
different number of images

# Apply data augmentation
i = 0
for batch in datagen.flow(resized_images, batch_size=1,
save_to_dir=augmented_dir, save_prefix='aug', save_format='jpg'):
    i += 1
    if i >= num_augmented_images:
        break

print(f'Resizing and data augmentation completed.
{num_augmented_images} images saved in {augmented_dir}.')
```

```
Resizing and data augmentation completed. 2500 images saved in
/content/drive/MyDrive/augmented_images.
```

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

```
cp: cannot stat 'kaggle.json': No such file or directory
```

```
!kaggle datasets download -d salader/dogs-vs-cats
```

```
Dataset URL: https://www.kaggle.com/datasets/salader/dogs-vs-cats
License(s): unknown
Downloading dogs-vs-cats.zip to /content
 98% 1.04G/1.06G [00:07<00:00, 144MB/s]
100% 1.06G/1.06G [00:07<00:00, 145MB/s]
```

```
import zipfile
zip_ref = zipfile.ZipFile("/content/dogs-vs-cats.zip","r")
zip_ref.extractall("/content")
zip_ref.close()

import tensorflow as tf
from tensorflow import keras
from keras import Sequential
from keras.layers import
Dense,Conv2D,MaxPooling2D,Flatten,BatchNormalization,Dropout
```

```
import shutil

# Define paths
main_dir = '/content/train'  # Change this if you want the folder in
Google Drive, e.g., '/content/drive/MyDrive/train'
sub_dir1 = '/content/drive/MyDrive/augmented_images'  # Replace with
the actual path to your first folder
sub_dir2 = '/content/dogs_vs_cats/test/dogs'  # Replace with the
actual path to your second folder

# Create the main directory
os.makedirs(main_dir, exist_ok=True)

# Define new paths for subdirectories inside the main directory
new_sub_dir1 = os.path.join(main_dir, 'subdir1')
new_sub_dir2 = os.path.join(main_dir, 'subdir2')

# Move or copy the existing directories into the main directory
# Use shutil.move to move or shutil.copytree to copy
shutil.move(sub_dir1, new_sub_dir1)
shutil.move(sub_dir2, new_sub_dir2)

print(f'Directories moved/copied to {main_dir}')
```

```
Directories moved/copied to /content/train
```

```
import os
import shutil
import random

# Define paths
main_dir = '/content/train'
sub_dir1 = os.path.join(main_dir, 'subdir1')
sub_dir2 = os.path.join(main_dir, 'subdir2')

test_dir = '/content/test'
sub_dir1_test = os.path.join(test_dir, 'sub_dir1_test')
sub_dir2_test = os.path.join(test_dir, 'sub_dir2_test')

# Create test directories
os.makedirs(sub_dir1_test, exist_ok=True)
os.makedirs(sub_dir2_test, exist_ok=True)

# Function to move a percentage of files
def move_percentage(src_dir, dest_dir, percentage):
    files = [f for f in os.listdir(src_dir) if
os.path.isfile(os.path.join(src_dir, f))]
    num_files_to_move = int(len(files) * percentage)
    files_to_move = random.sample(files, num_files_to_move)
```

```
    for file in files_to_move:
        shutil.move(os.path.join(src_dir, file),
os.path.join(dest_dir, file))

# Move 20% of the images to the test directories
move_percentage(sub_dir1, sub_dir1_test, 0.2)
move_percentage(sub_dir2, sub_dir2_test, 0.2)

print('20% of the images moved to the test directories.')
```

```
20% of the images moved to the test directories.
```

```
# Define paths for the folders to be removed
folders_to_remove = [
    '/content/test/cats',
    '/content/test/dogs',
    '/content/train/cats',
    '/content/train/dogs',
]

# Remove the folders
for folder in folders_to_remove:
    if os.path.exists(folder):
        shutil.rmtree(folder)
        print(f'Removed {folder}')
    else:
        print(f'{folder} does not exist')

print('Specified folders removed from train and test directories.')
```

```
Removed /content/test/cats
Removed /content/test/dogs
Removed /content/train/cats
Removed /content/train/dogs
Specified folders removed from train and test directories.
```

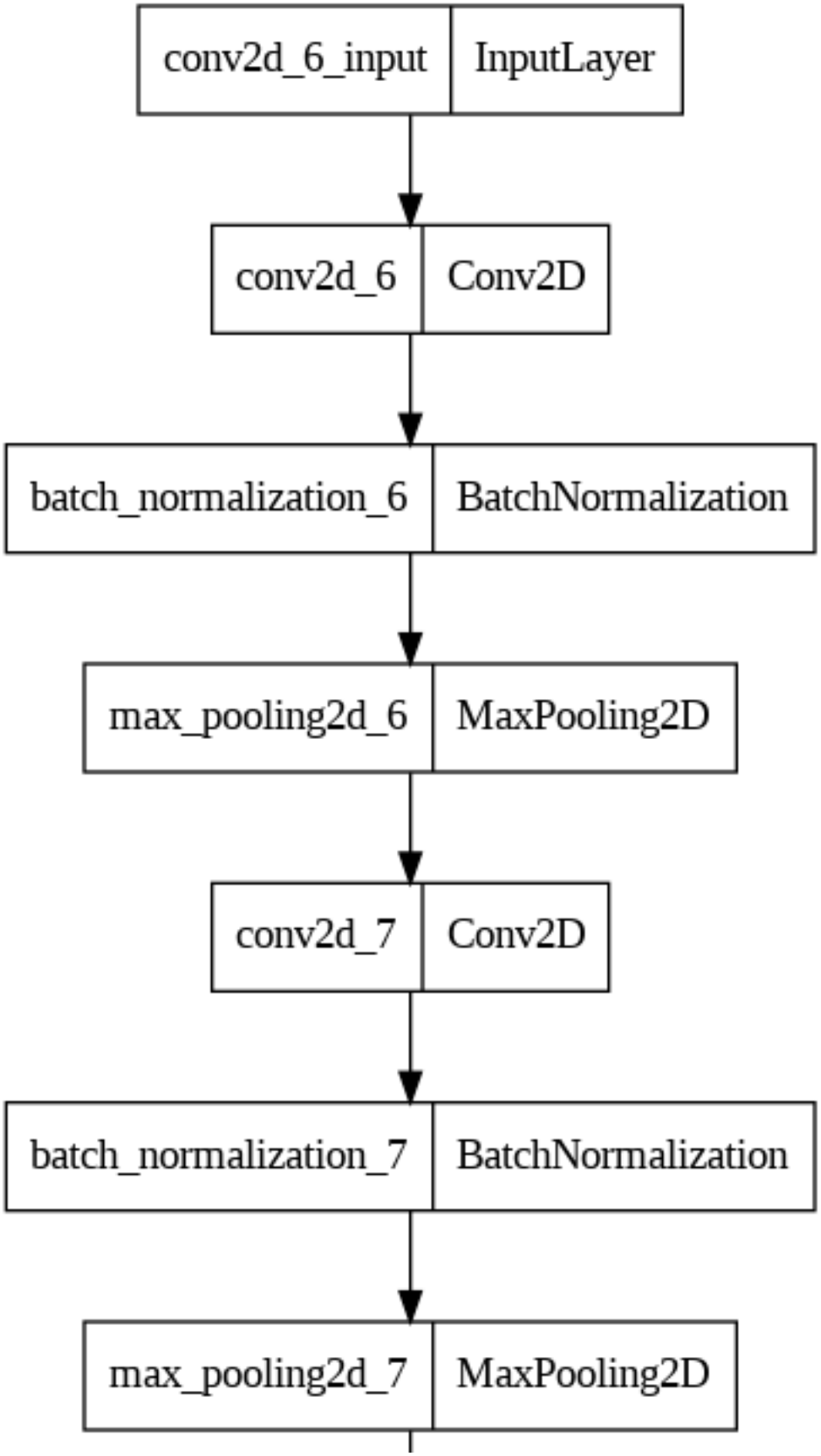
```
# generators   (To take all the images as input )

train_dataset = keras.utils.image_dataset_from_directory(
    directory ="/content/train",
    labels="inferred",
    label_mode="int",
    batch_size=32,
    image_size=(256, 256),
)


validation_dataset = keras.utils.image_dataset_from_directory(
    directory ="/content/test",
```

```
    labels="inferred",
    label_mode="int",
    batch_size=32,
    image_size=(256, 256),
)
```

```
Found 3999 files belonging to 2 classes.
Found 999 files belonging to 2 classes.
```

```
# Normalization
def process(image,label):
  image =  tf.cast(image/255  , tf.float32)
  return image, label

train_dataset = train_dataset.map(process)
validation_dataset = validation_dataset.map(process)

# # Creating a CNN Model
# from tensorflow.keras.regularizers import l2

# model = Sequential()

# # First Convolutional Layer
# model.add(Conv2D(32, kernel_size=(3, 3), padding="valid",
activation="relu", input_shape=(256, 256, 3),
kernel_regularizer=l2(0.001)))
# model.add(BatchNormalization())
# model.add(MaxPooling2D(pool_size=(2, 2), strides=2,
padding="valid"))

# # Second Convolutional Layer
# model.add(Conv2D(64, kernel_size=(3, 3), padding="valid",
activation="relu", kernel_regularizer=l2(0.001)))
# model.add(BatchNormalization())
# model.add(MaxPooling2D(pool_size=(2, 2), strides=2,
padding="valid"))

# # Third Convolutional Layer
# model.add(Conv2D(128, kernel_size=(3, 3), padding="valid",
activation="relu", kernel_regularizer=l2(0.001)))
# model.add(BatchNormalization())
# model.add(MaxPooling2D(pool_size=(2, 2), strides=2,
padding="valid"))

# # Flattening
# model.add(Flatten())

# # Fully Connected Layers
# model.add(Dense(128, activation="relu",
kernel_regularizer=l2(0.001)))
# model.add(Dropout(0.3))  # Increased dropout rate
```

```
# model.add(Dense(64, activation="relu",
kernel_regularizer=l2(0.001)))
# model.add(Dropout(0.3))  # Increased dropout rate

# # Output Layer
# model.add(Dense(1, activation="sigmoid"))

# # Compile the model
# model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["accuracy"])

#
---------------------------------------------------------------------
---------------------------------------------------------------------

model = Sequential()

# First Convolutional Layer
model.add(Conv2D(32, kernel_size=(3, 3), padding="valid",
activation="relu", input_shape=(256, 256, 3),
kernel_regularizer=l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=2, padding="valid"))

# Second Convolutional Layer
model.add(Conv2D(64, kernel_size=(3, 3), padding="valid",
activation="relu", kernel_regularizer=l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=2, padding="valid"))

# Flattening
model.add(Flatten())

# Fully Connected Layers
model.add(Dense(64, activation="relu", kernel_regularizer=l2(0.001)))
model.add(Dropout(0.3))  # Increased dropout rate

# Output Layer
model.add(Dense(1, activation="sigmoid"))

# Compile the model
model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["accuracy"])

from keras.utils import plot_model
plot_model(model)
```


conv2d_6_input
InputLayer
conv2d_6
Conv2D
batch_normalization_6
BatchNormalization
max_pooling2d_6
MaxPooling2D
conv2d_7
Conv2D
batch_normalization_7
BatchNormalization
max_pooling2d_7
MaxPooling2D

```
model.summary()
```

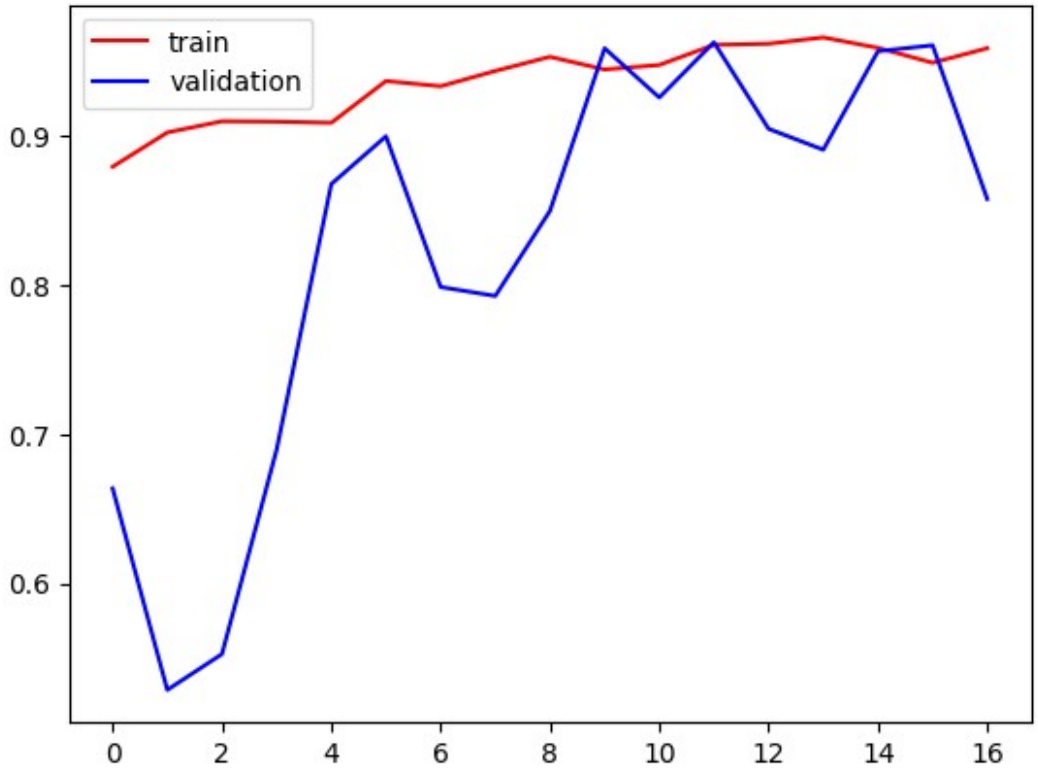
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 254, 254, 32)      896

 batch_normalization_6 (Bat  (None, 254, 254, 32)      128
 chNormalization)

 max_pooling2d_6 (MaxPoolin  (None, 127, 127, 32)      0
 g2D)

 conv2d_7 (Conv2D)           (None, 125, 125, 64)      18496

 batch_normalization_7 (Bat  (None, 125, 125, 64)      256
 chNormalization)

 max_pooling2d_7 (MaxPoolin  (None, 62, 62, 64)        0
 g2D)

 flatten_2 (Flatten)         (None, 246016)            0

 dense_6 (Dense)             (None, 64)                15745088

 dropout_4 (Dropout)         (None, 64)                0

 dense_7 (Dense)             (None, 1)                 65

=================================================================
Total params: 15764929 (60.14 MB)
Trainable params: 15764737 (60.14 MB)
Non-trainable params: 192 (768.00 Byte)
_________________________________________________________________
```

```
model.compile(optimizer="adam",loss="binary_crossentropy",metrics=["accuracy"] )

from tensorflow.keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

history = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=[early_stopping]
)
```

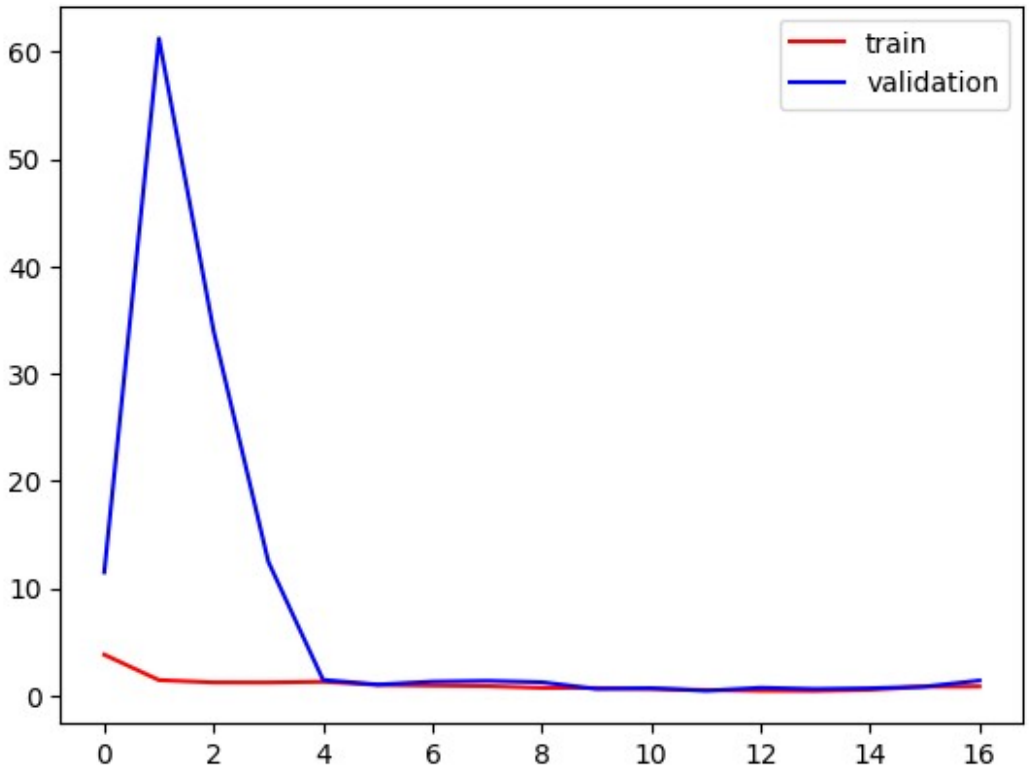
```
Epoch 1/50
125/125 [==============================] - 15s 105ms/step - loss:
3.7990 - accuracy: 0.8795 - val_loss: 11.4823 - val_accuracy: 0.6637
Epoch 2/50
125/125 [==============================] - 13s 102ms/step - loss:
1.4264 - accuracy: 0.9025 - val_loss: 61.2585 - val_accuracy: 0.5285
Epoch 3/50
125/125 [==============================] - 13s 100ms/step - loss:
1.2104 - accuracy: 0.9100 - val_loss: 33.9609 - val_accuracy: 0.5526
Epoch 4/50
125/125 [==============================] - 12s 95ms/step - loss:
1.2023 - accuracy: 0.9097 - val_loss: 12.4603 - val_accuracy: 0.6897
Epoch 5/50
125/125 [==============================] - 12s 95ms/step - loss:
1.2706 - accuracy: 0.9090 - val_loss: 1.4462 - val_accuracy: 0.8679
Epoch 6/50
125/125 [==============================] - 13s 99ms/step - loss:
0.9768 - accuracy: 0.9370 - val_loss: 1.0067 - val_accuracy: 0.8999
Epoch 7/50
125/125 [==============================] - 13s 102ms/step - loss:
0.9136 - accuracy: 0.9335 - val_loss: 1.2829 - val_accuracy: 0.7988
Epoch 8/50
125/125 [==============================] - 12s 93ms/step - loss:
0.8667 - accuracy: 0.9437 - val_loss: 1.3546 - val_accuracy: 0.7928
Epoch 9/50
125/125 [==============================] - 12s 94ms/step - loss:
0.7021 - accuracy: 0.9532 - val_loss: 1.2304 - val_accuracy: 0.8498
Epoch 10/50
125/125 [==============================] - 12s 98ms/step - loss:
0.6887 - accuracy: 0.9447 - val_loss: 0.5985 - val_accuracy: 0.9590
Epoch 11/50
125/125 [==============================] - 13s 103ms/step - loss:
0.5979 - accuracy: 0.9477 - val_loss: 0.6736 - val_accuracy: 0.9259
Epoch 12/50
125/125 [==============================] - 12s 97ms/step - loss:
0.5007 - accuracy: 0.9612 - val_loss: 0.4328 - val_accuracy: 0.9630
Epoch 13/50
125/125 [==============================] - 13s 103ms/step - loss:
0.4323 - accuracy: 0.9620 - val_loss: 0.7132 - val_accuracy: 0.9049
Epoch 14/50
125/125 [==============================] - 13s 99ms/step - loss:
0.4195 - accuracy: 0.9662 - val_loss: 0.5965 - val_accuracy: 0.8909
Epoch 15/50
125/125 [==============================] - 12s 93ms/step - loss:
0.5252 - accuracy: 0.9592 - val_loss: 0.6646 - val_accuracy: 0.9570
Epoch 16/50
125/125 [==============================] - 12s 92ms/step - loss:
0.8107 - accuracy: 0.9492 - val_loss: 0.8180 - val_accuracy: 0.9610
Epoch 17/50
```

```
125/125 [==============================] - 13s 100ms/step - loss:
0.8347 - accuracy: 0.9590 - val_loss: 1.3831 - val_accuracy: 0.8579
```

```
import matplotlib.pyplot as plt

plt.plot(history.history["accuracy"],color="red",label="train")
plt.plot(history.history["val_accuracy"],color="blue",label="validatio
n")
plt.legend()
plt.show()
```

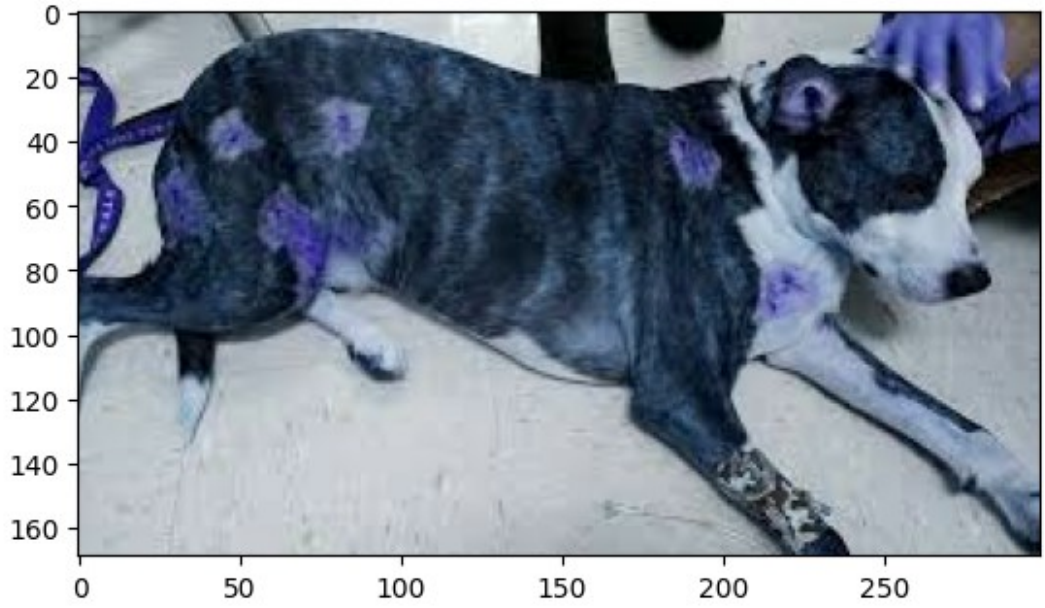



```
import matplotlib.pyplot as plt

plt.plot(history.history["loss"],color="red",label="train")
plt.plot(history.history["val_loss"],color="blue",label="validation")
plt.legend()
plt.show()
```




```
# testing model on unseen images
import cv2
test_img_1 = cv2.imread("/test.jpeg")
plt.imshow(test_img_1)
```

```
<matplotlib.image.AxesImage at 0x79a1f45b09d0>
```



```
test_img_1.shape
```

```
(169, 299, 3)
```

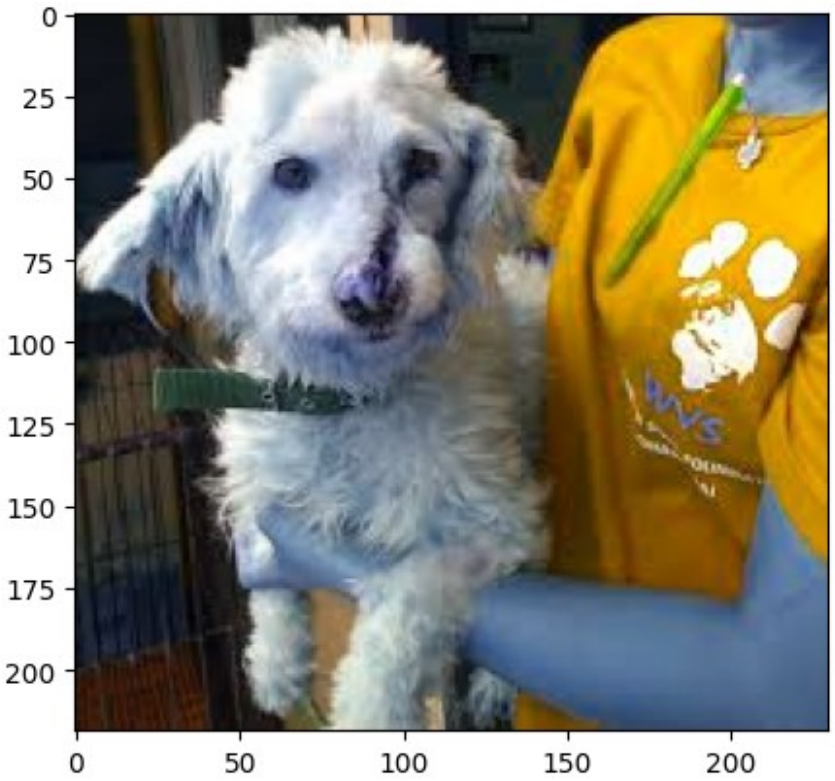
```
test_img_1  = cv2.resize(test_img_1,(256,256))

test_input = test_img_1.reshape((1,256,256,3)) # b/c we have only one image as input

model.predict(test_input)
```

```
1/1 [==============================] - 0s 130ms/step
```

```
array([[1.]], dtype=float32)
```

```
test_img = cv2.imread("/100.jpeg")
plt.imshow(test_img)
```

```
<matplotlib.image.AxesImage at 0x79a1f448de10>
```




```
test_img.shape
```

```
(219, 230, 3)
```

```
test_img  = cv2.resize(test_img,(256,256))
```

```
test_input = test_img.reshape((1,256,256,3)) # b/c we have only one
image as input

model.predict(test_input)
```

```
1/1 [==============================] - 0s 17ms/step
```

```
array([[1.]], dtype=float32)
```

```
# Load and preprocess new image
from tensorflow.keras.preprocessing import image

img_path = '/dod.jpg'
img = image.load_img(img_path, target_size=(256, 256))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0) / 255.0  # Scale the
image

# Predict
prediction = model.predict(img_array)
print(prediction)

# Convert to class label
class_label = (prediction > 0.5).astype(int)
print("Predicted class:", "Injured Dog" if class_label == 1 else
"Healthy Dog")
```

```
1/1 [==============================] - 0s 19ms/step
[[1.]]
Predicted class: Injured Dog
```