

2. Questions on basic neural network model designing and compiling

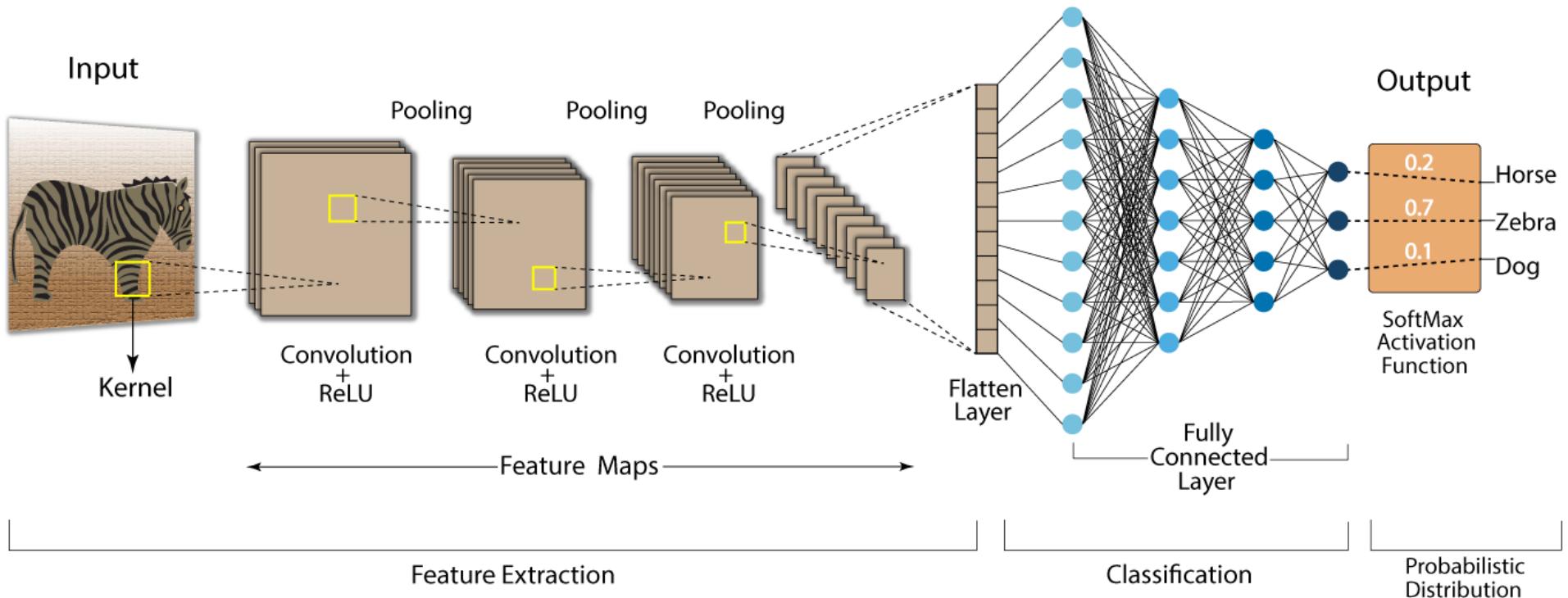
Dataset : The dataset has been attached to the assignment with two folders which consists of two objects i.e., dogs and wolves images. Please develop a CNN to identify the shapes as either dog or wolf. The task has been categorized into binary classification

2.1 Print shape of the data and understand how many images of different classes exist in this dataset. Visualize some images using matplotlib. Convert the RGB Image to Grayscale (For easier computation) . Normalize the data so that data is in range 0-1. Reshape train and test images into one dimensional vector 3.0 pts

2.2 Construct the Deep Neural Network to classify the 2 classes of images available in the dataset. Compile and fit the model (No restrictions on CNN architecture. Feel free to explore

and optimize.) Compute the performance accuracy of the model created. 7.0 pts

Convolution Neural Network (CNN)



```
In [... import matplotlib.pyplot as plt  
import tensorflow as tf  
import pandas as pd  
import numpy as np  
import warnings  
  
# Suppress warnings
```

```
warnings.filterwarnings('ignore')

# Import necessary modules
from tensorflow import keras
from keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing import image
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.utils import image_dataset_from_directory
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load

import os
import matplotlib.image as mpimg
from zipfile import ZipFile
```

In [6]: *# Define the path to your data zip file*
data_path = "DogvsWolf.zip"

```
# Extract the dataset from the zip file
with ZipFile(data_path, 'r') as zip:
    zip.extractall()
    print('The data set has been extracted.')
```

The data set has been extracted.

```
In [7]: # Define the base directory where the 'Dataset' folder is located
base_dir = 'Dataset'

# Get the list of subdirectories inside 'Train' to extract class names
train_dir = os.path.join(base_dir, 'Train')
class_names = os.listdir(train_dir)

# Display the class names
print("Class names:", class_names)

    Class names: ['dogs', 'wolves']

In... # Specify the directory paths for dogs and wolves
dogs_directory = 'Dataset/Train/dogs' # Directory path for 'dogs'
wolves_directory = 'Dataset/Train/wolves' # Directory path for 'wolves'

# Get the list of file names in the directories
dog_filenames = os.listdir(dogs_directory)
wolf_filenames = os.listdir(wolves_directory)

# Set the index to start displaying images
start_index = 210

# Create lists of image file paths to display
dog_images = [os.path.join(dogs_directory, fname) for fname in dog_filenames]
```

```
wolf_images = [os.path.join(wolves_directory, fname) for fname in wolf_fil

# Create a 4x4 grid to display images
plt.figure(figsize=(16, 16))

# Display dog images
for i, dog_image_path in enumerate(dog_images):
    subplot = plt.subplot(4, 4, i + 1)
    subplot.axis('off')
    img = mpimg.imread(dog_image_path)
    plt.imshow(img)
    plt.title('Dog')

# Display wolf images
for i, wolf_image_path in enumerate(wolf_images):
    subplot = plt.subplot(4, 4, i + 9) # Start from the 9th subplot
    subplot.axis('off')
    img = mpimg.imread(wolf_image_path)
    plt.imshow(img)
    plt.title('Wolf')

plt.show()
```

Dog



Dog



Dog



Dog





Wolf



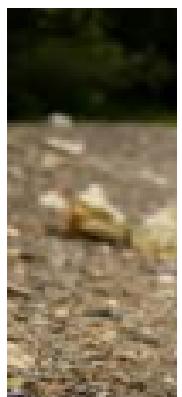
Wolf



Wolf



Wolf





```
In [...] # Define the image size and batch size for the dataset
image_size = (200, 200)
batch_size = 32

# Create the training dataset using TensorFlow's image_dataset_from_directory
train_dataset = image_dataset_from_directory(
    directory=os.path.join(base_dir, 'Train'),
    labels='inferred',
    label_mode='binary', # Binary classification
    validation_split=0.1,
    subset='training',
    seed=1,
    image_size=image_size,
    batch_size=batch_size,
    shuffle=True # Shuffle the data for better training
)

# Create the validation dataset with similar improvements
validation_dataset = image_dataset_from_directory(
    directory=os.path.join(base_dir, 'Train'),
    labels='inferred',
    label_mode='binary',
```

```
validation_split=0.1,  
subset='validation',  
seed=1,  
image_size=image_size,  
batch_size=batch_size,  
shuffle=False # No need to shuffle validation data  
)
```

Found 940 files belonging to 2 classes.

Using 846 files for training.

Found 940 files belonging to 2 classes.

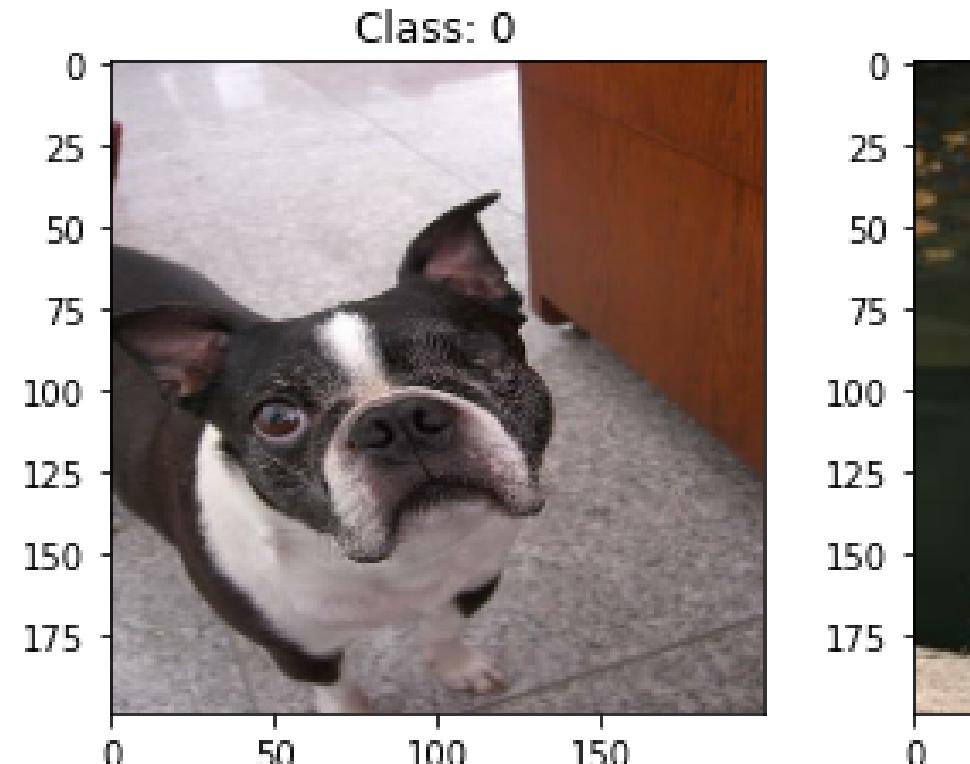
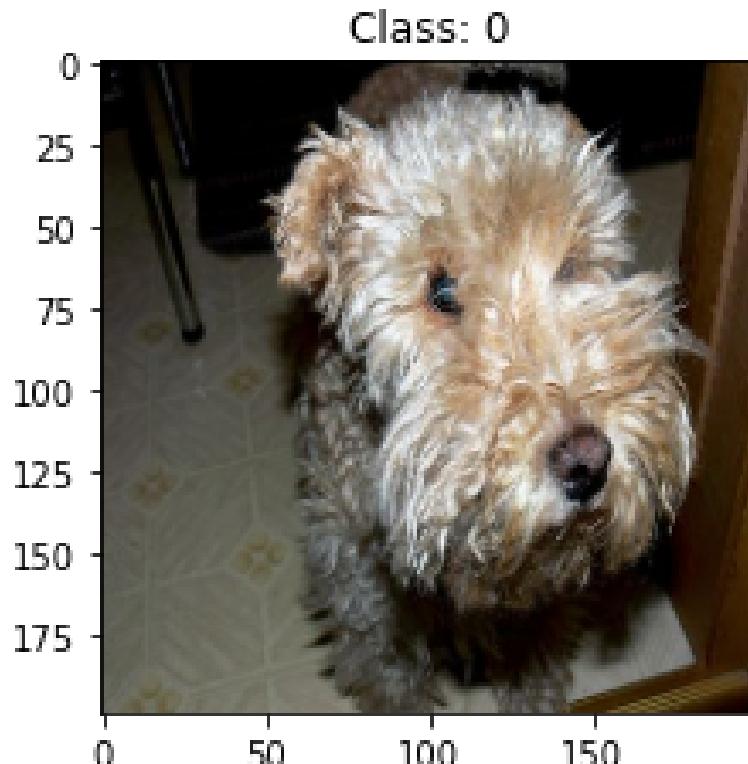
Using 94 files for validation.

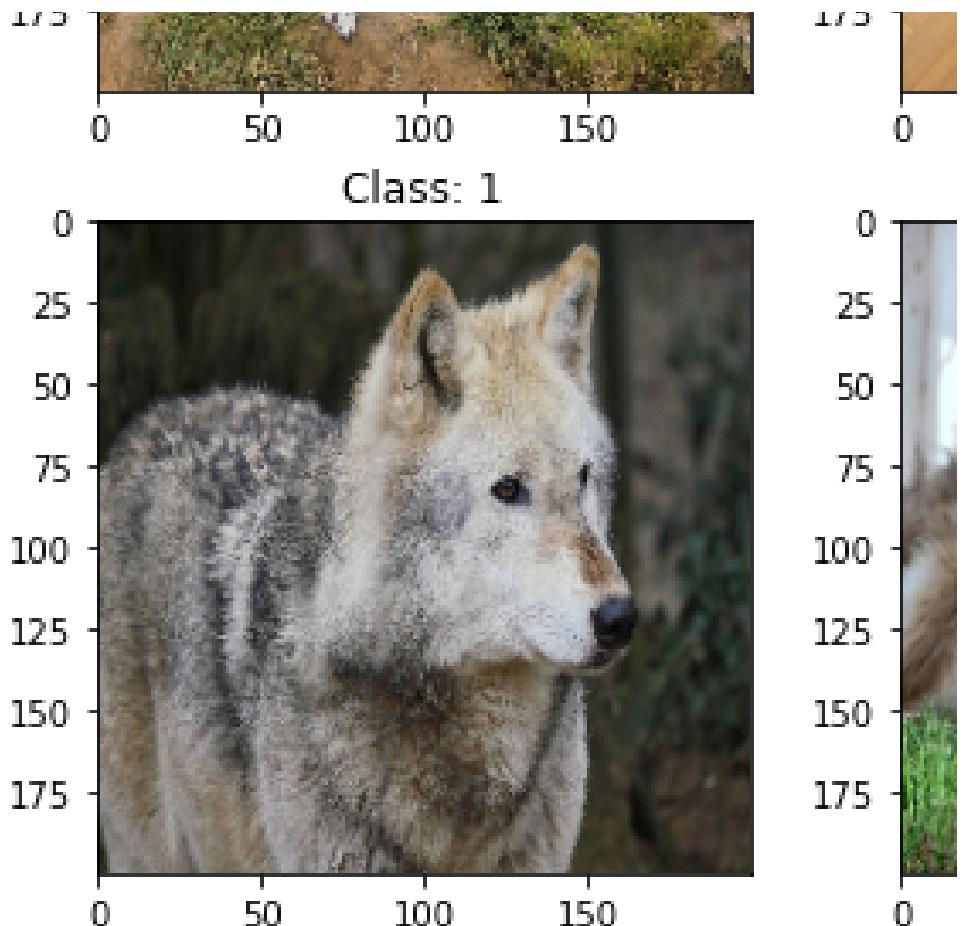
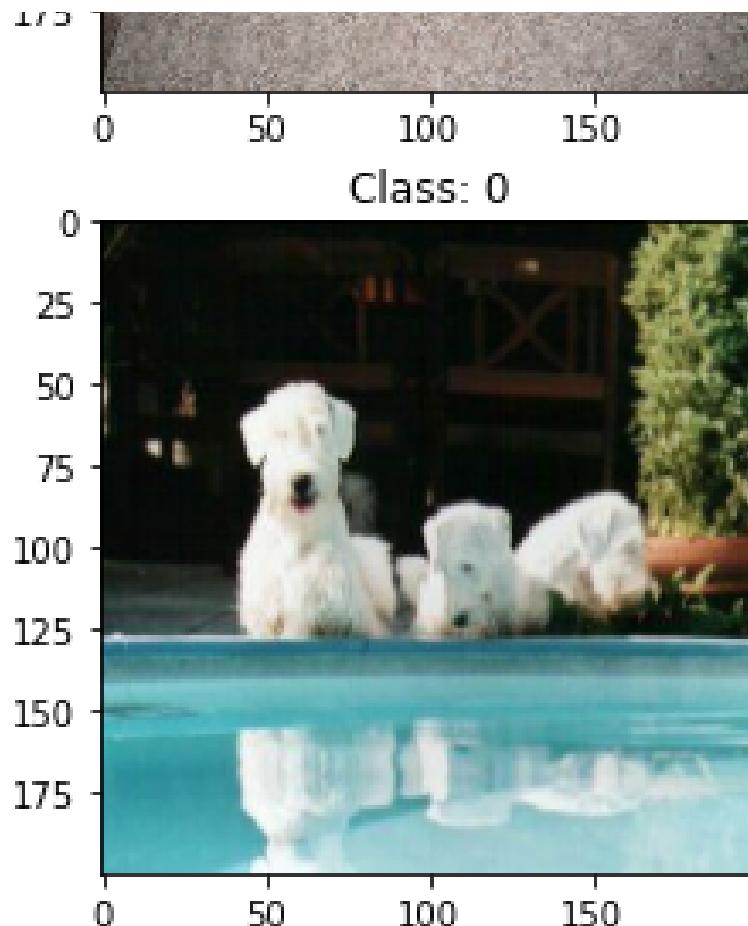


```
In [1...]: # Print the shape of the data  
for images, labels in train_dataset:  
    print("Train data shape:", images.shape)  
    break # Break after the first batch to avoid long output  
  
# Count the number of images in each class  
num_dogs = len(os.listdir(os.path.join(base_dir, 'Train', 'dogs')))  
num_wolves = len(os.listdir(os.path.join(base_dir, 'Train', 'wolves')))  
print("Number of dog images:", num_dogs)  
print("Number of wolf images:", num_wolves)
```

Train data shape: (32, 200, 200, 3)
Number of dog images: 470
Number of wolf images: 470

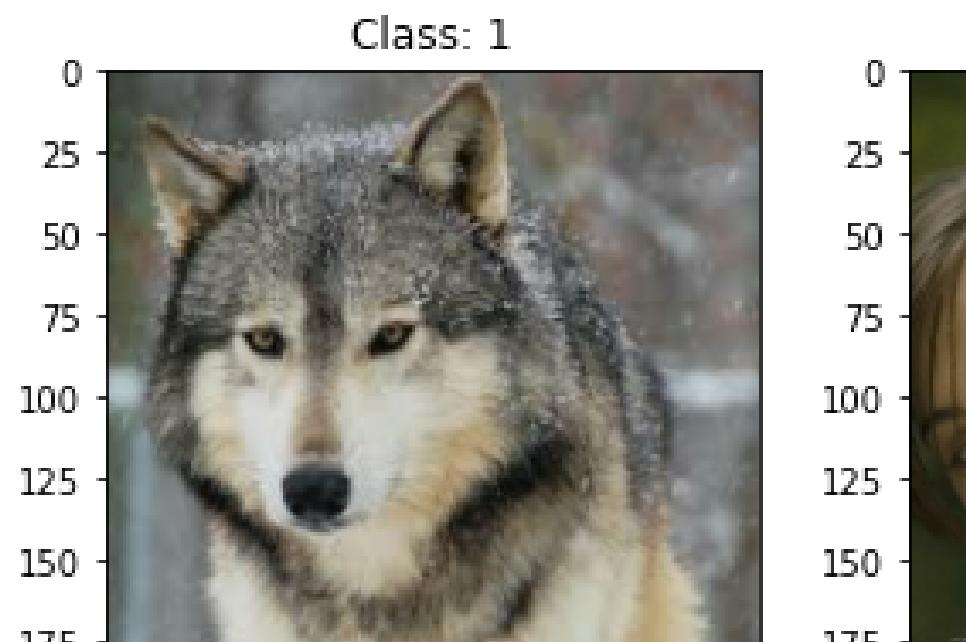
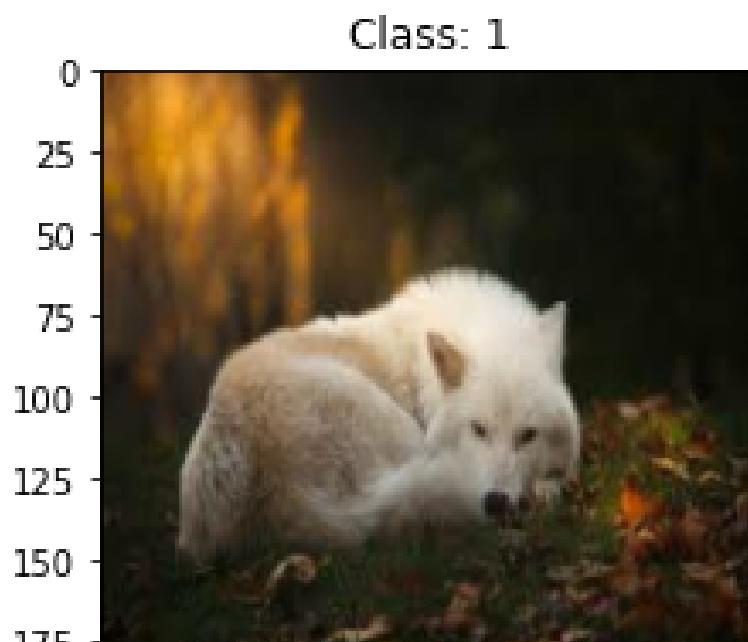
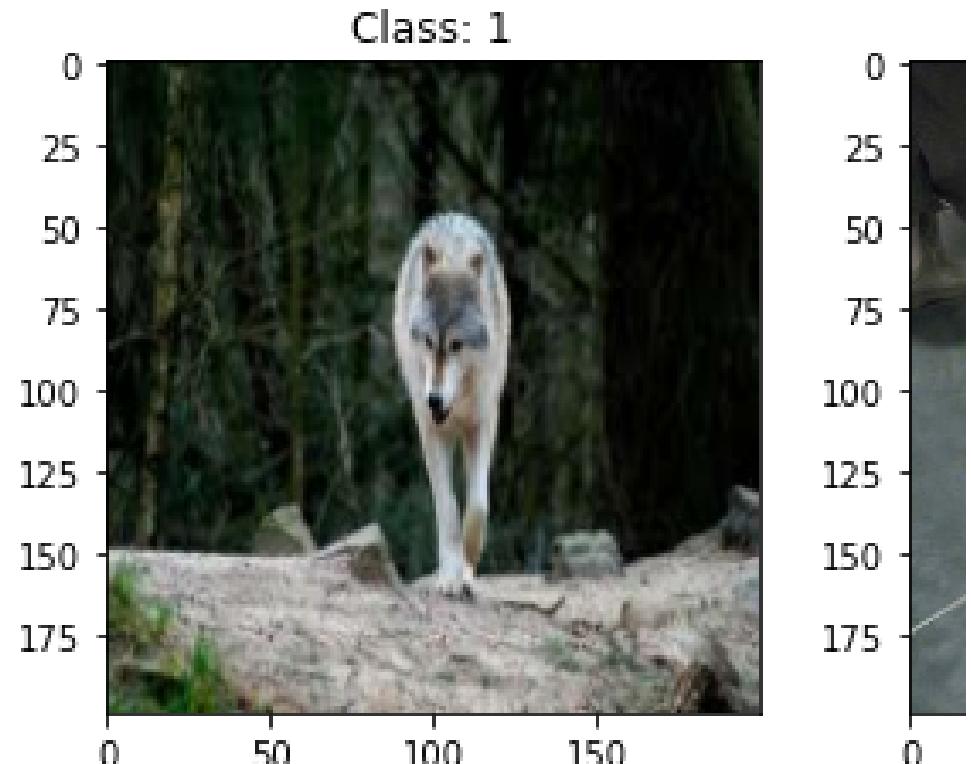
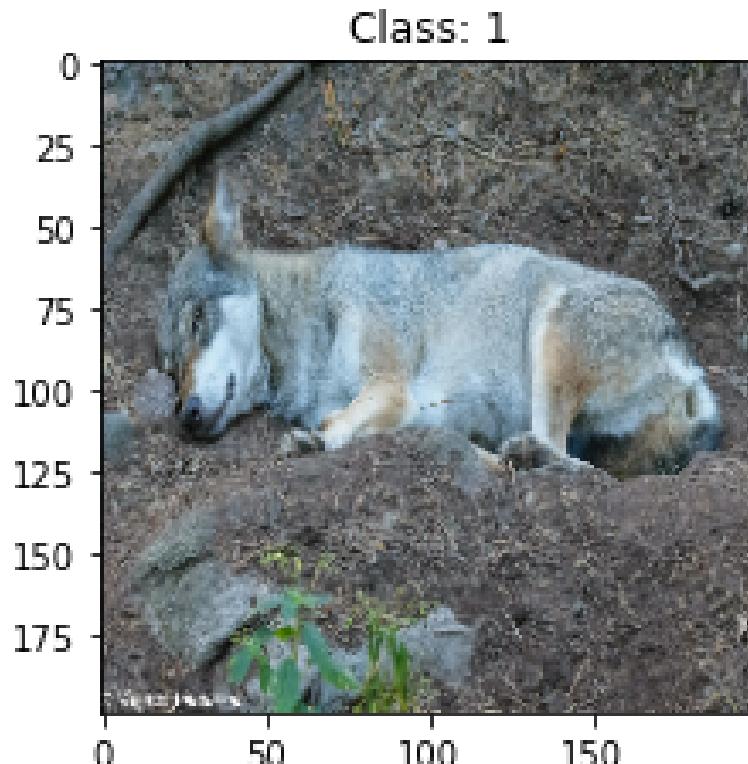
In [... # Visualize some sample images
sample_images, sample_labels = next(train_dataset.as_numpy_iterator()) #
plt.figure(figsize=(12, 12))
for i in range(9):
 plt.subplot(3, 3, i+1)
 plt.imshow(sample_images[i].astype("uint8")) # Convert the images to
 plt.title("Class: " + str(int(sample_labels[i])))
plt.show()

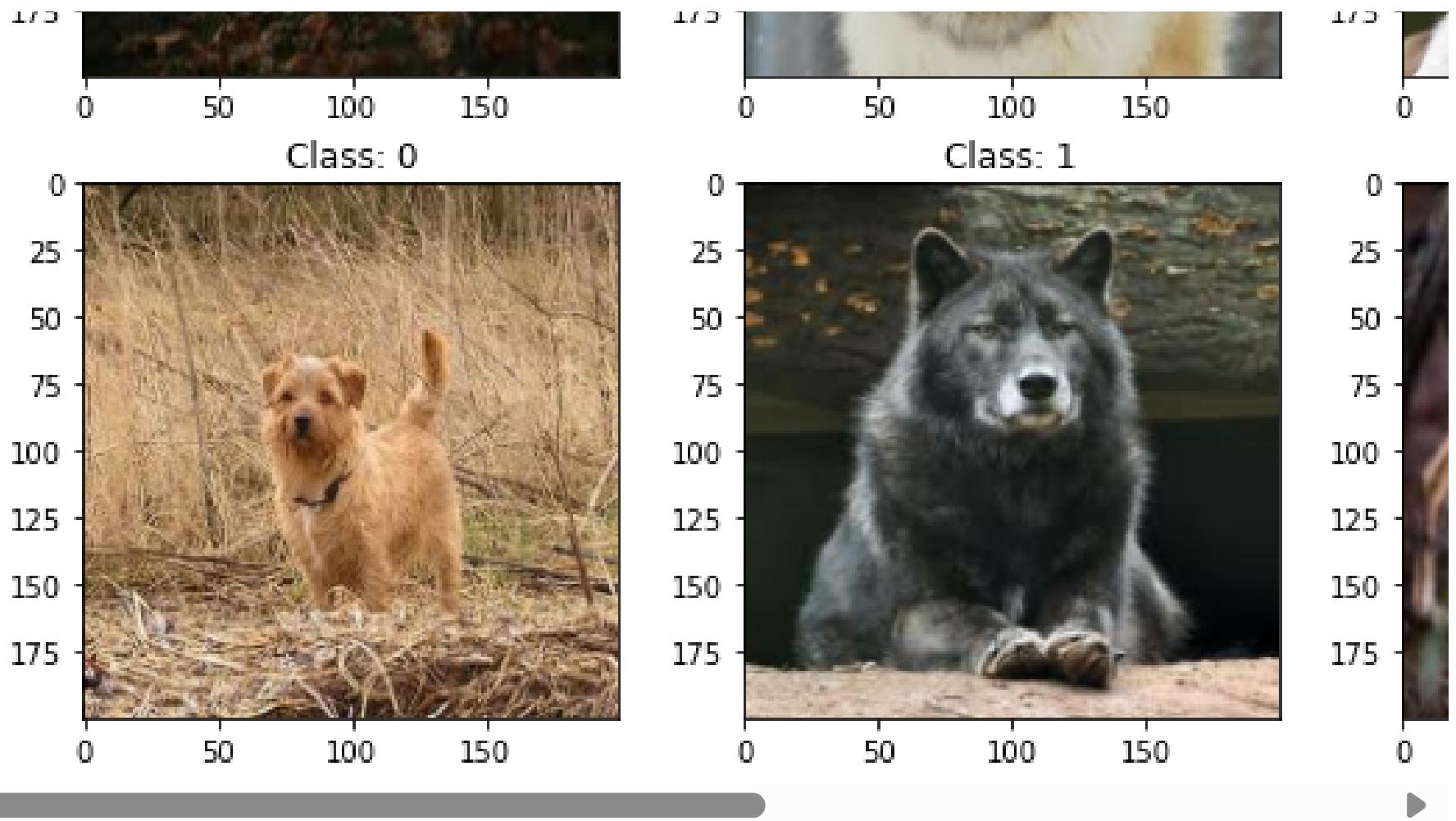




```
In [...] # Visualize some sample images
sample_images, sample_labels = next(train_dataset.as_numpy_iterator()) #
plt.figure(figsize=(12, 12))
for i in range(9):
    plt.subplot(3, 3, i+1)
    plt.imshow(sample_images[i].astype("uint8")) # Convert the images to
```

```
plt.title("Class: " + str(int(sample_labels[i])))  
plt.show()
```





In ... `from tensorflow.keras import layers`

```
# Create a Sequential model for classification
model = tf.keras.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(200, 200, 3),
    layers.MaxPooling2D(2, 2),
    layers.Conv2D(64, (3, 3), activation='relu'),
```

```
layers.MaxPooling2D(2, 2),  
layers.Conv2D(64, (3, 3), activation='relu'),  
layers.MaxPooling2D(2, 2),  
layers.Conv2D(64, (3, 3), activation='relu'),  
layers.MaxPooling2D(2, 2),  
  
layers.Flatten(),  
layers.Dense(512, activation='relu'),  
layers.BatchNormalization(),  
layers.Dense(512, activation='relu'),  
layers.Dropout(0.1),  
layers.BatchNormalization(),  
layers.Dense(512, activation='relu'),  
layers.Dropout(0.2),  
layers.BatchNormalization(),  
layers.Dense(1, activation='sigmoid') # Use 1 neuron with sigmoid act  
])  
  
# Compile the model  
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])  
  
# Display model summary  
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
conv2d (Conv2D)	(None, 198, 198, 32)	896
max_pooling2d (MaxPooling2D)	(None, 99, 99, 32)	0
conv2d_1 (Conv2D)	(None, 97, 97, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 48, 48, 64)	0
conv2d_2 (Conv2D)	(None, 46, 46, 64)	36928
max_pooling2d_2 (MaxPooling2D)	(None, 23, 23, 64)	0
conv2d_3 (Conv2D)	(None, 21, 21, 64)	36928
max_pooling2d_3 (MaxPooling2D)	(None, 10, 10, 64)	0
flatten (Flatten)	(None, 6400)	0

dense (Dense)	(None, 512)	3277312
batch_normalization (Batch Normalization)	(None, 512)	2048
dense_1 (Dense)	(None, 512)	262656
dropout (Dropout)	(None, 512)	0
batch_normalization_1 (BatchNormalization)	(None, 512)	2048
dense_2 (Dense)	(None, 512)	262656
dropout_1 (Dropout)	(None, 512)	0
batch_normalization_2 (BatchNormalization)	(None, 512)	2048
dense_3 (Dense)	(None, 1)	513

=====

Total params: 3902529 (14.89 MB)

Trainable params: 3899457 (14.88 MB)

Non-trainable params: 3072 (12.00 KB)

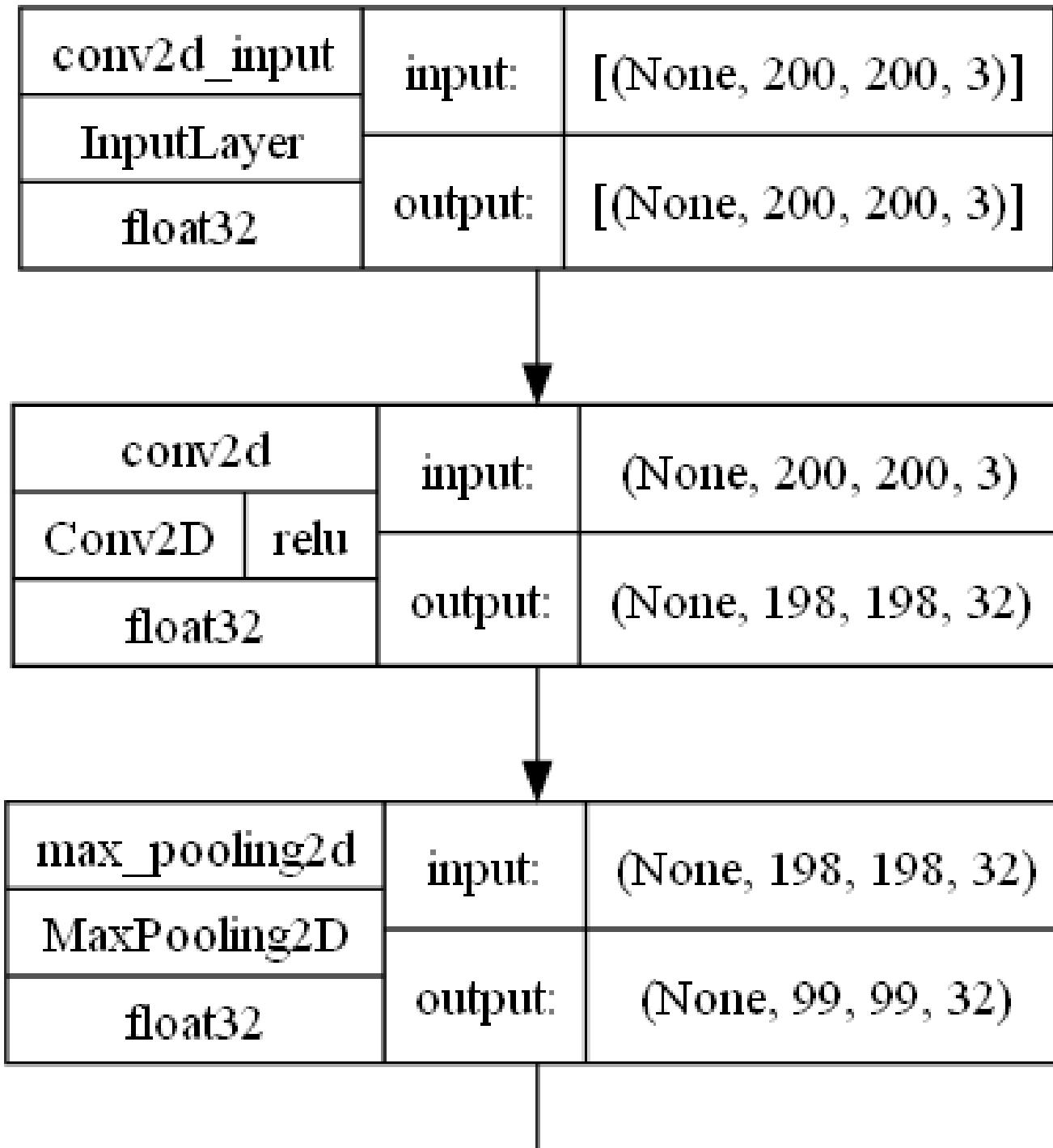
In [17]: !pip install pydot

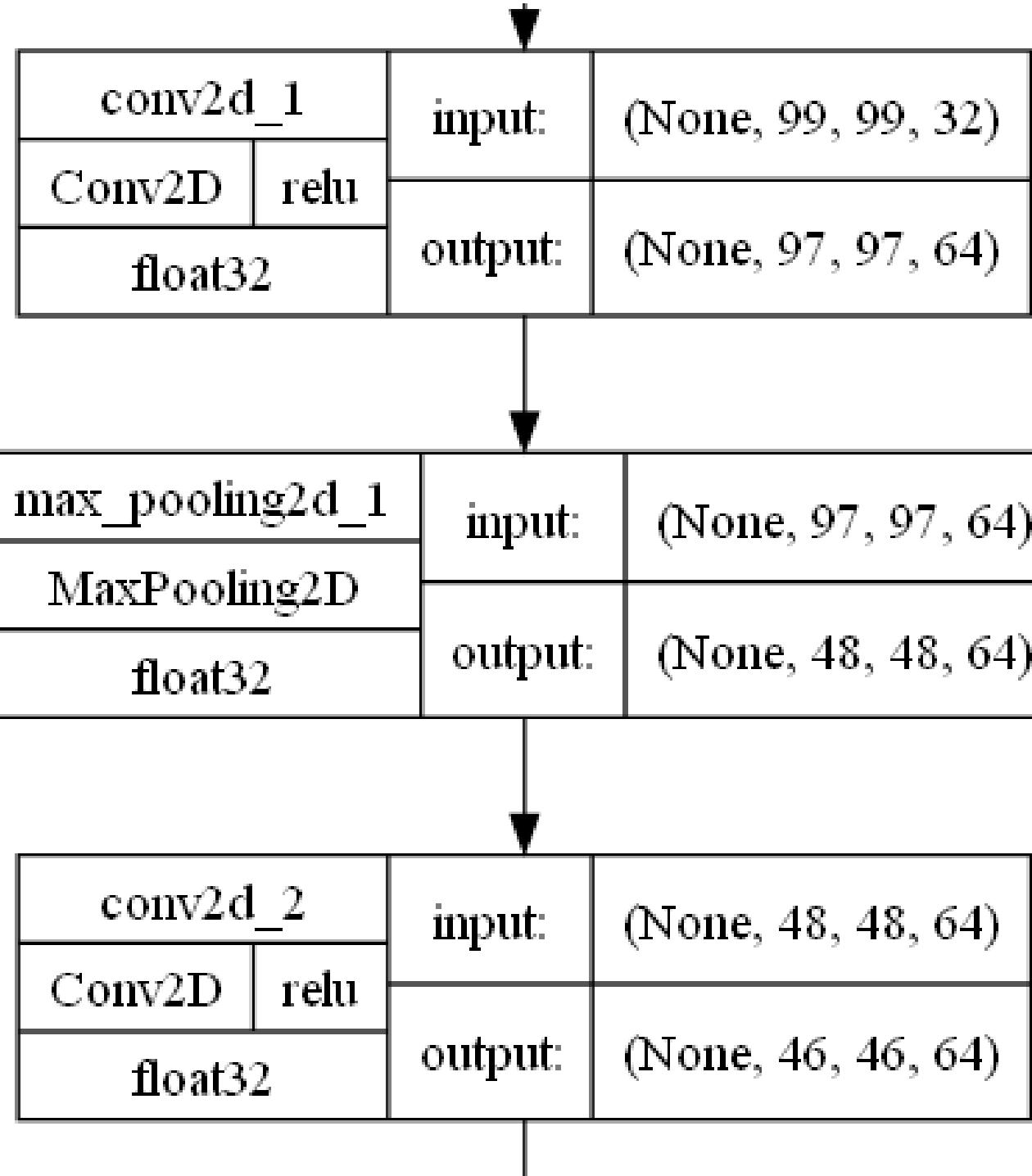
```
Requirement already satisfied: pydot in c:\users\pakbo\anaconda3\lib\site-packages  
Requirement already satisfied: pyparsing>=2.1.4 in c:\users\pakbo\anaconda3\lib\site-packages
```

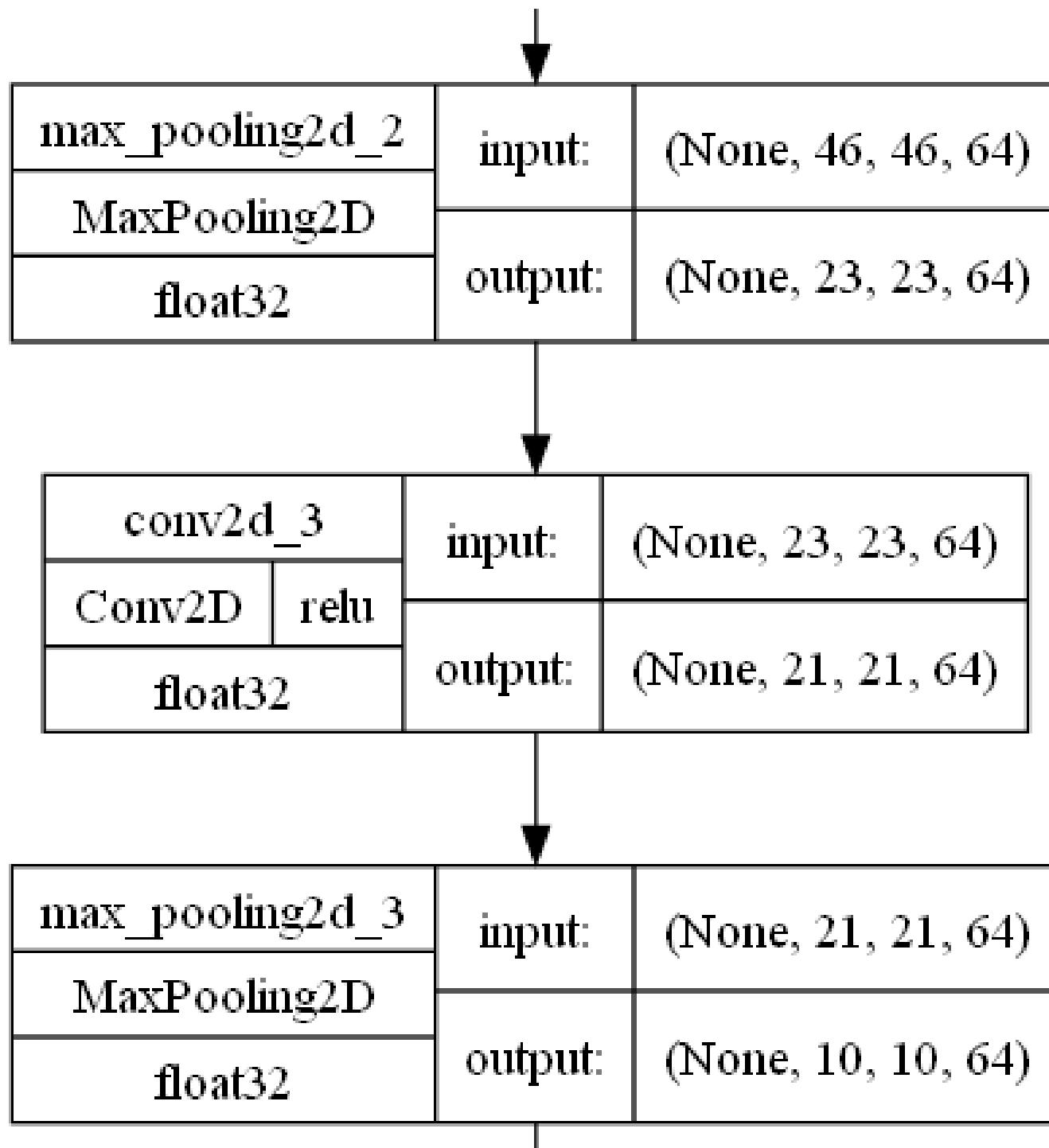
In [... from tensorflow.keras.utils import plot_model

```
# Visualize the model with show_shapes, show_dtype, and show_layer_activations
plot_model(
    model,
    show_shapes=True,
    show_dtype=True,
    show_layer_activations=True
)
```

Out[18]:



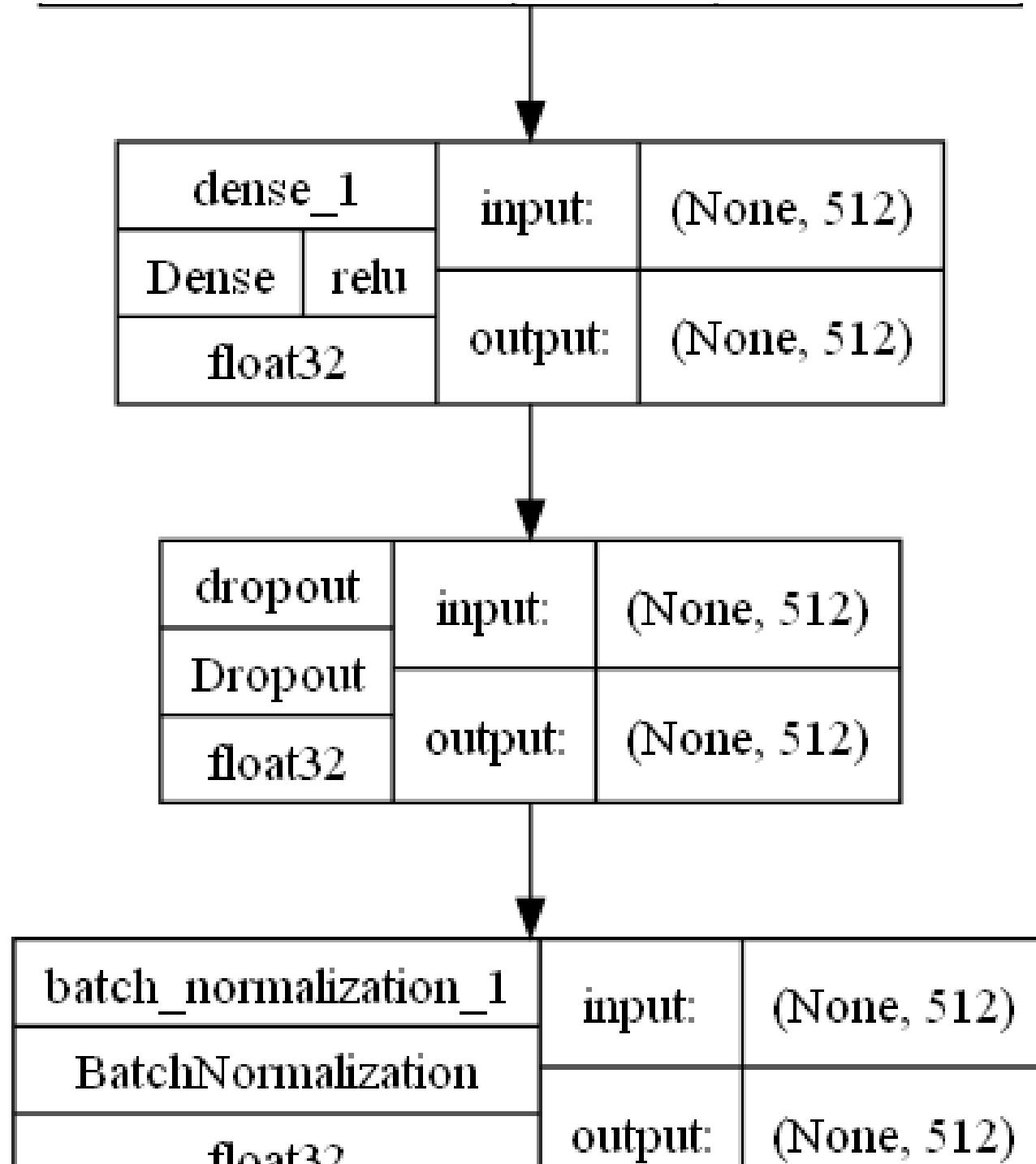


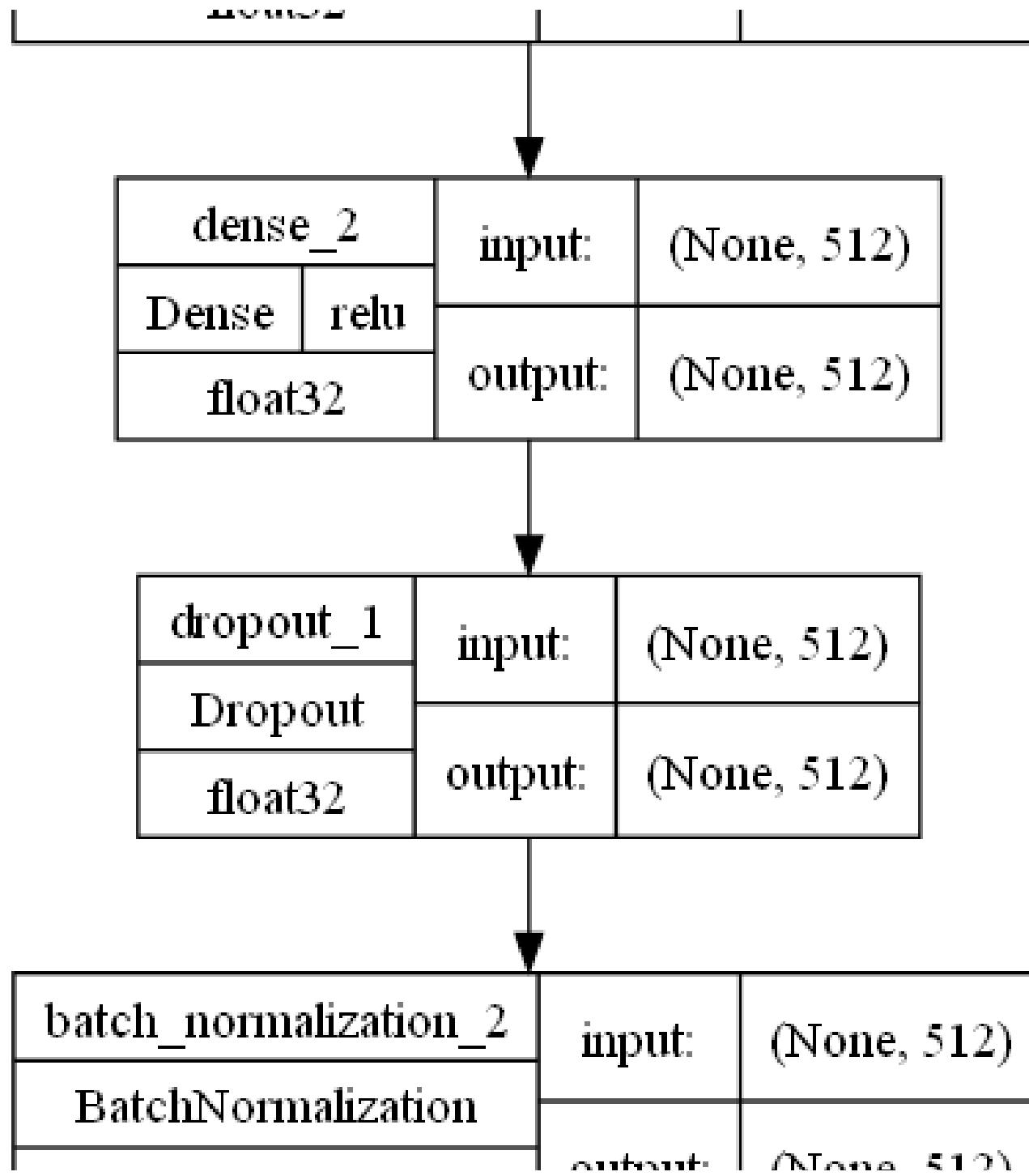


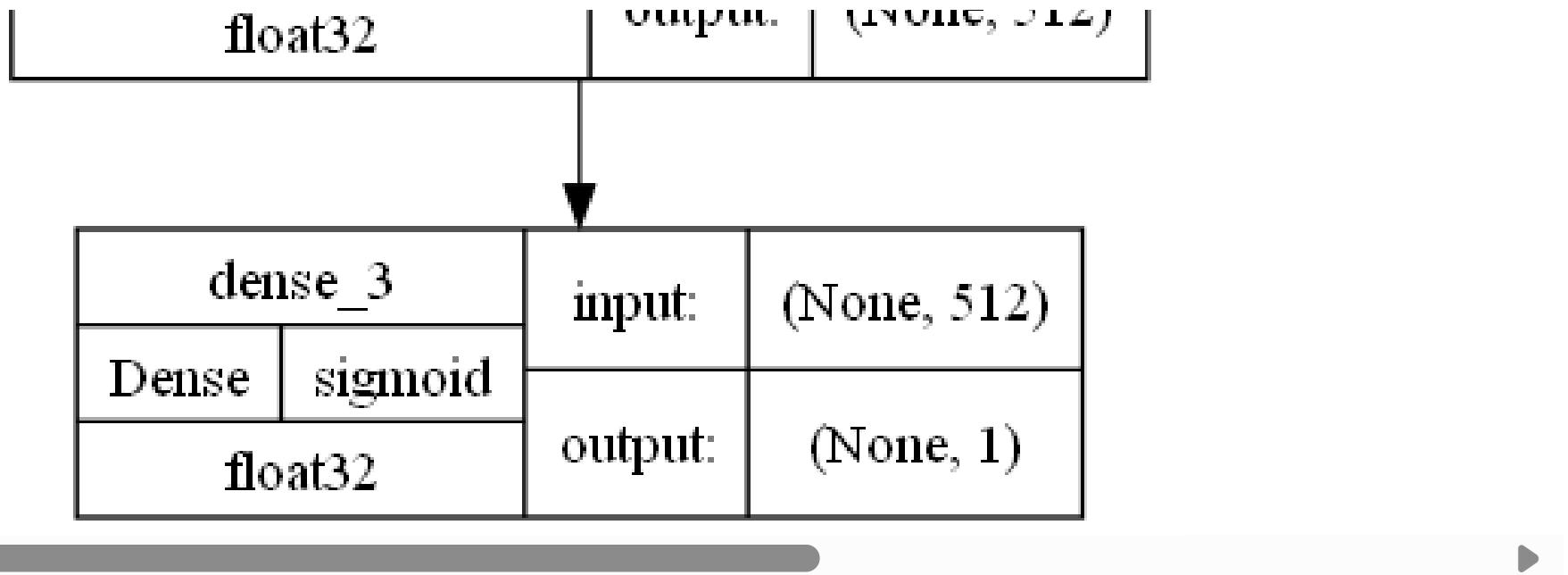
flatten	input:	(None, 10, 10, 64)
Flatten		
float32	output:	(None, 6400)

dense	input:	(None, 6400)
Dense	relu	
float32	output:	(None, 512)

batch_normalization	input:	(None, 512)
BatchNormalization		
float32	output:	(None, 512)







```
In [... model.compile(  
    loss='binary_crossentropy', # Use binary cross-entropy for binary cl  
    optimizer='adam', # You can adjust the optimizer as needed  
    metrics=['accuracy']) # You can include additional metrics if desired  
)
```

```
In [20]: # Fit the model using the improved training and validation datasets  
history = model.fit(  
    train_dataset,  
    epochs=10,  
    validation_data=validation_dataset  
)
```

Epoch 1/10
27/27 [=====] - 27s 913ms/step - loss: 0.81
accuracy: 1.0000

Epoch 2/10
27/27 [=====] - 23s 852ms/step - loss: 0.68
accuracy: 0.9681

Epoch 3/10
27/27 [=====] - 24s 883ms/step - loss: 0.61
accuracy: 0.5745

Epoch 4/10
27/27 [=====] - 25s 926ms/step - loss: 0.56
accuracy: 0.8191

Epoch 5/10
27/27 [=====] - 24s 870ms/step - loss: 0.51
accuracy: 0.9468

Epoch 6/10
27/27 [=====] - 27s 970ms/step - loss: 0.56
accuracy: 0.6489

Epoch 7/10
27/27 [=====] - 29s 1s/step - loss: 0.4809
accuracy: 1.0000

Epoch 8/10
27/27 [=====] - 25s 880ms/step - loss: 0.41
accuracy: 1.0000

Epoch 9/10

```
27/27 [=====] - 26s 967ms/step - loss: 0.31  
racy: 0.9043  
Epoch 10/10  
27/27 [=====] - 24s 883ms/step - loss: 0.31  
racy: 0.0426
```

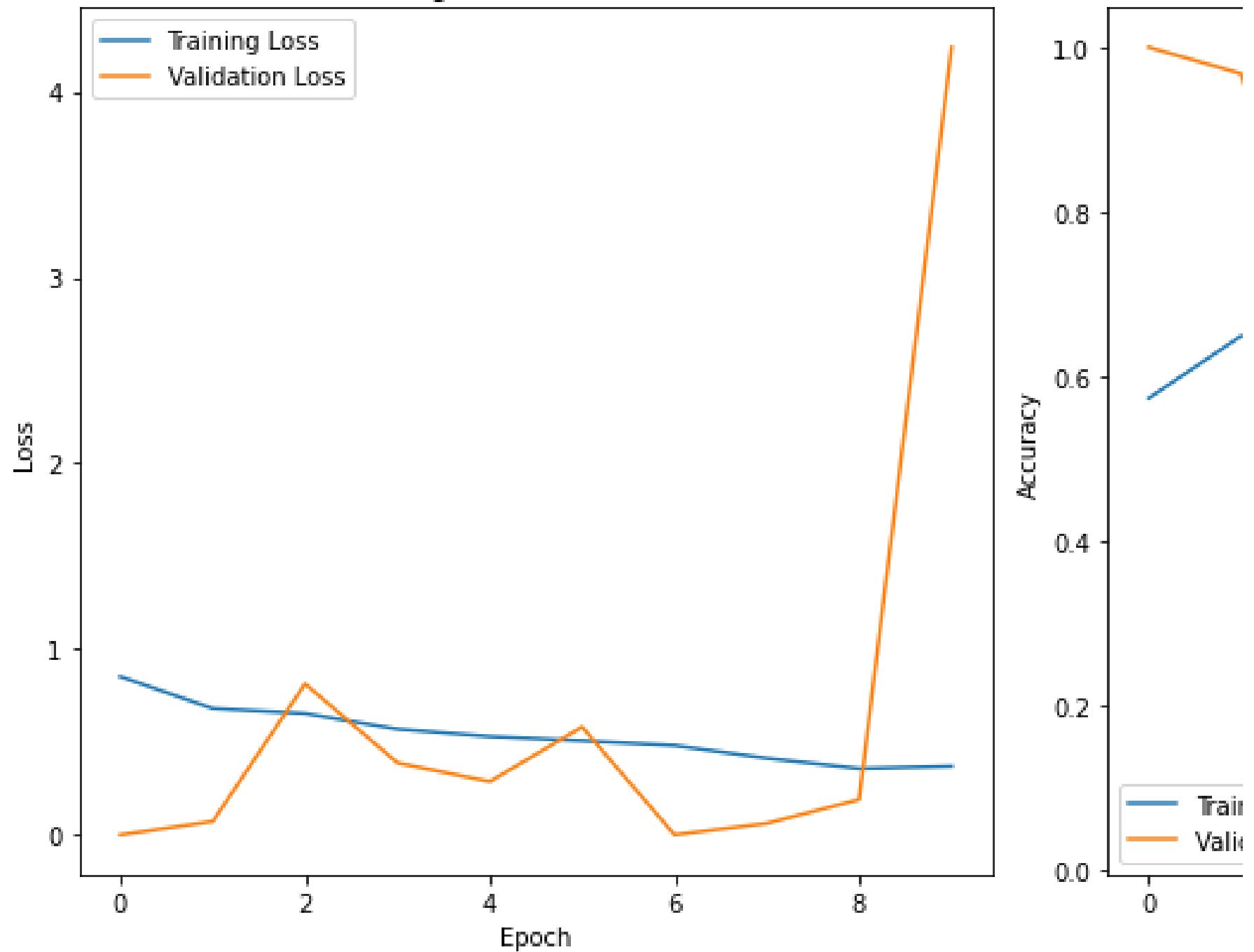
```
In [21]: # Create a DataFrame from the history object  
history_df = pd.DataFrame(history.history)
```

```
# Plot the training and validation loss  
plt.figure(figsize=(12, 6))  
plt.subplot(1, 2, 1)  
plt.plot(history_df['loss'], label='Training Loss')  
plt.plot(history_df['val_loss'], label='Validation Loss')  
plt.xlabel('Epoch')  
plt.ylabel('Loss')  
plt.legend()  
plt.title('Training and Validation Loss')
```

```
# Plot the training and validation accuracy  
plt.subplot(1, 2, 2)  
plt.plot(history_df['accuracy'], label='Training Accuracy')  
plt.plot(history_df['val_accuracy'], label='Validation Accuracy')  
plt.xlabel('Epoch')  
plt.ylabel('Accuracy')
```

```
plt.legend()  
plt.title('Training and Validation Accuracy')  
  
# Show the plots  
plt.tight_layout()  
plt.show()
```

Training and Validation Loss



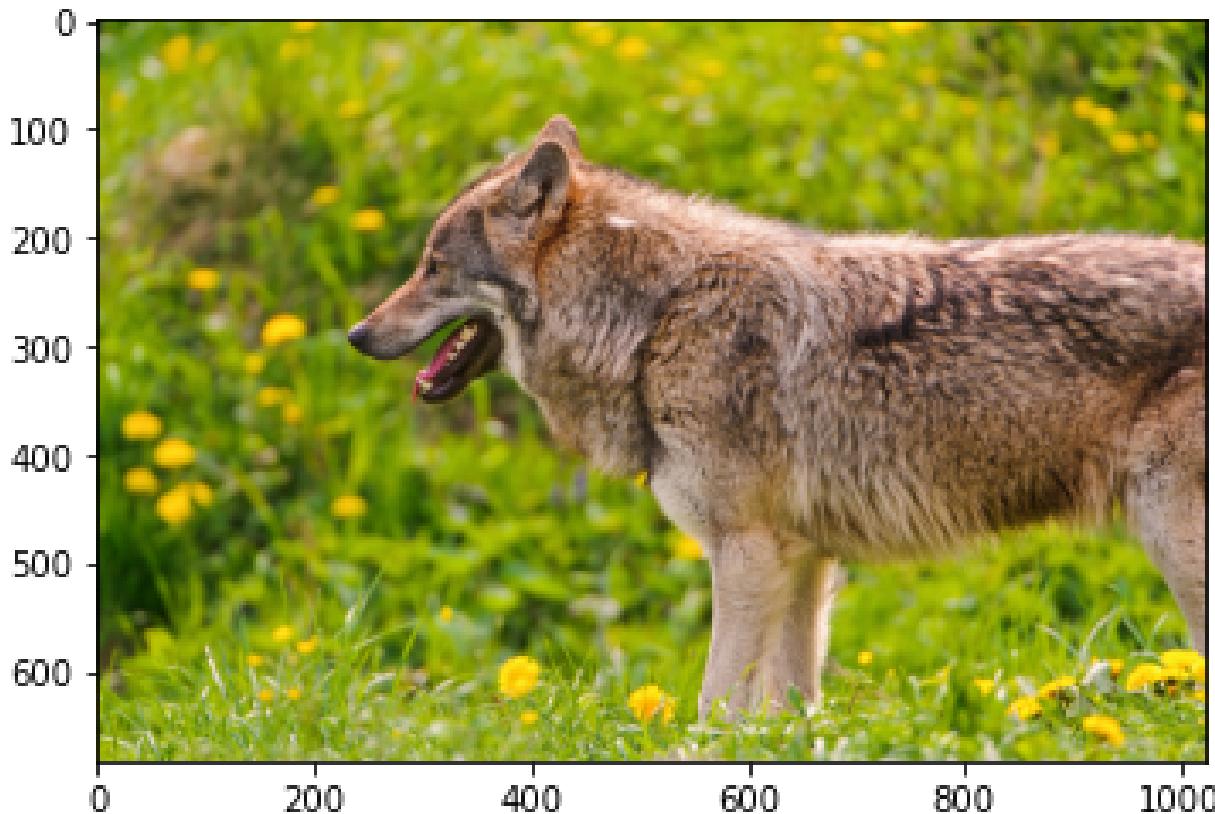
```
In [...] # Load and preprocess the image
img_path = 'Dataset/Valid/wolves/Img-5142.jpg' # Provide the correct path
img = image.load_img(img_path, target_size=(200, 200))
img = image.img_to_array(img)
img = img / 255.0 # Normalize the image
img = img.reshape((1,) + img.shape) # Reshape for model input

# Make a prediction
prediction = model.predict(img)

# Interpret the prediction
if prediction[0][0] >= 0.5:
    print("It's a Wolf")
else:
    print("It's a Dog")

# Display the image
plt.imshow(image.load_img(img_path))
plt.show()

1/1 [=====] - 0s 177ms/step
It's a Wolf
```



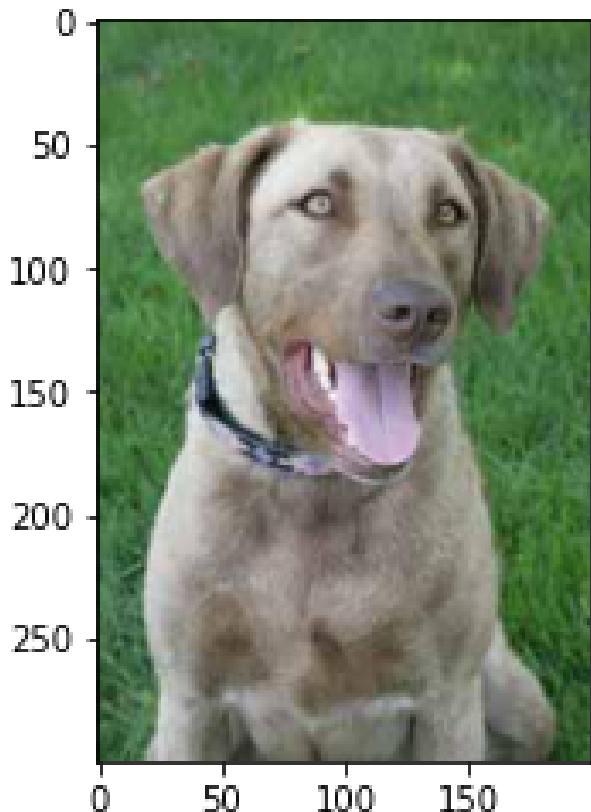
```
In [...] # Load and preprocess the dog image
    dog_img_path = 'Dataset/Valid/dogs/n02099849_4498.jpg' # Provide the cor
    img = image.load_img(dog_img_path, target_size=(200, 200))
    img = image.img_to_array(img)
    img = img / 255.0 # Normalize the image
    img = img.reshape((1,) + img.shape) # Reshape for model input

    # Make a prediction
    prediction = model.predict(img)
```

```
# Interpret the prediction
if prediction[0][0] >= 0.5:
    print("It's a Dog")
else:
    print("It's a Wolf")

# Display the dog image
plt.imshow(image.load_img(dog_img_path))
plt.show()

1/1 [=====] - 0s 35ms/step
It's a Dog
```



I... Inferences:

The dataset provided **for** the classification task consists of images of dogs

The images were preprocessed, normalized, **and** reshaped to suit the neural n

The model, designed using TensorFlow, employs a Convolutional Neural Networ

The model was compiled using the Adam optimizer **and** binary cross-entropy lo

During the training process, the model displayed varying levels of accuracy

The model successfully classified sample images of both dogs **and** wolves, de

The dataset provided **for** the classification task consists of images of dogs

The images were preprocessed, normalized, **and** reshaped to suit the neural n

The model, designed using TensorFlow, employs a Convolutional Neural Networ

The model was compiled using the Adam optimizer **and** binary cross-entropy lo

During the training process, the model displayed varying levels of accuracy

The model successfully classified sample images of both dogs **and** wolves, de