dso569-final-project-bonev4

May 6, 2024

1 I. DNN and CNN

[3]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.layers import Dropout


import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os
from PIL import Image, ImageFile
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten, BatchNormalization
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.layers import Dropout
print(tf.__version__)
import PIL

import time
import numpy as np
import os
import pydot
from typing import List, Tuple
```

```
from matplotlib.pyplot import imshow
%matplotlib inline
import matplotlib.pyplot as plt
import PIL.Image
import pathlib
import shutil

import tensorflow as tf
from tensorflow import keras

from tensorflow.keras.preprocessing import image
from tensorflow.keras import layers
from tensorflow.keras.layers import Input, Add, Dense, Activation,␣
↪ZeroPadding2D, BatchNormalization, Flatten, Conv2D, AveragePooling2D,␣
↪MaxPooling2D, GlobalMaxPooling2D
from tensorflow.keras.initializers import glorot_uniform
from tensorflow.keras.models import Model, load_model

from tensorflow.python.keras.utils import layer_utils
#from tensorflow.keras.utils.vis_utils import model_to_dot
from tensorflow.keras.utils import model_to_dot
from tensorflow.keras.utils import plot_model

from tensorflow.keras.applications.imagenet_utils import preprocess_input

from IPython.display import SVG

import scipy.misc

import tensorflow.keras.backend as K
K.set_image_data_format('channels_last') # can be channels_first or␣
↪channels_last.
#K.set_learning_phase(1) # 1 stands for learning phase
print(tf.__version__)
```

```
2.16.1
2.16.1
```

[4]:
```
!pip install Pillow numpy
```

```
Requirement already satisfied: Pillow in
/Users/mingentsai/miniconda3/envs/py3k/lib/python3.11/site-packages (9.4.0)
Requirement already satisfied: numpy in
/Users/mingentsai/miniconda3/envs/py3k/lib/python3.11/site-packages (1.26.4)
```

1.1 1. Data Preprocessing

224*224, greyscale

[5]:
```
folder_path = '/Users/mingentsai/Desktop/USC/Courses/DSO 569/Homework/Group␣
↪Project/Bone_Fracture_Binary_Classification'
contents = os.listdir(folder_path)
print(contents)
```

```
['.DS_Store', 'test', 'train', 'val']
```

[6]:
```
ImageFile.LOAD_TRUNCATED_IMAGES = True # Allow loading of truncated images

def load_images_from_folder(base_folder):
  data = []
  labels = []
  categories = {'fractured': 1, 'not fractured': 0}
  for category, label in categories.items():
    folder_path = os.path.join(base_folder, category)
    if os.path.isdir(folder_path):
      for filename in os.listdir(folder_path):
        if filename.lower().endswith(('.jpeg', '.jpg')): # Handle both .jpeg␣
↪and .jpg fi
          img_path = os.path.join(folder_path, filename)
          try:
            with Image.open(img_path) as img:
              img = img.convert('L')
              img = img.resize((224, 224),PIL.Image.Resampling.LANCZOS)
              data.append(np.array(img))
              labels.append(label)
          except IOError as e:
            print(f"Error opening or processing image {img_path}: {e}")
        else:
          print(f"Skipped non-JPEG file {filename}")
    else:
      print(f"Directory {folder_path} does not exist")

  # Convert to numpy arrays
  data = np.array(data)
  labels = np.array(labels)

  # Shuffle the data
  idx = np.arange(len(data))
  np.random.shuffle(idx)
  data = data[idx]
  labels = labels[idx]

  return data, labels
```

[7]:
```
train_data, train_labels = load_images_from_folder(os.path.join(folder_path,␣
↪'train'))
```

```
val_data, val_labels = load_images_from_folder(os.path.join(folder_path, 'val'))
test_data, test_labels = load_images_from_folder(os.path.join(folder_path,␣
↪'test'))
```

```
Skipped non-JPEG file .DS_Store
Skipped non-JPEG file .DS_Store
```

```
/Users/mingentsai/miniconda3/envs/py3k/lib/python3.11/site-
packages/PIL/Image.py:996: UserWarning: Palette images with Transparency
expressed in bytes should be converted to RGBA images
  warnings.warn(
```

```
Skipped non-JPEG file .DS_Store
Skipped non-JPEG file .DS_Store
Skipped non-JPEG file .DS_Store
```

[34]:
```
train_data
```

[34]:
```
array([[[0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        …,
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0]],

       [[0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        …,
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0]],

       [[0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        …,
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0]],

       …,

       [[0, 0, 0, …, 0, 0, 0],
        [1, 1, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        …,
```

```
        [0, 1, 0, …, 0, 0, 0],
        [0, 1, 0, …, 0, 0, 0],
        [0, 1, 0, …, 1, 0, 0]],

       [[0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        …,
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0]],

       [[0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        …,
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0]]], dtype=uint8)
```

[20]:
```
train_labels[:100]
```

[20]:
```
array([1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0])
```

[10]:
```
train_data.shape
```

[10]:
```
(9200, 224, 224)
```

[11]:
```
len(train_labels)
```

[11]:
```
9200
```

[29]:
```
test_data[:10]
```

[29]:
```
array([[[  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        …,
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0]],

       [[  0,   0,   0, …,   0,   0,   0],
```

```
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        …,
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0]],

       [[  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        …,
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0]],

       …,

       [[  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        …,
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0]],

       [[194,  29,   4, …, 109, 119, 191],
        [ 63,   0,  10, …,   0,   0,  16],
        [ 19,   9,  11, …,  12,  20,  21],
        …,
        [  0,   2,  25, …,  12,   3,  31],
        [ 78,  22,  33, …,   8,   0, 106],
        [237, 217, 219, …,  19, 113, 223]],

       [[  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        …,
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0]]], dtype=uint8)
```

[12]:
```
test_data.shape
```

[12]:
```
(491, 224, 224)
```

[13]:
```
len(test_labels)
```

[13]: 491

[35]:
```
test_labels
```

[35]:
```
array([1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1,
       0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
       1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
       1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1,
       0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 1, 0])
```

[14]:
```
val_data
```

[14]:
```
array([[[ 32,  30,  29, …,  35,  36,  36],
        [ 31,  29,  28, …,  33,  36,  37],
        [ 33,  32,  29, …,  35,  36,  36],
        …,
        [ 17,  18,  17, …,  18,  18,  18],
        [ 16,  18,  17, …,  17,  17,  18],
        [ 16,  18,  17, …,  17,  18,  21]],

       [[  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        …,
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0]],

       [[  0,   0,   0, …,   0,   0,   0],
```

```
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        …,
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0]],

       …,

       [[  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        …,
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0]],

       [[  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        …,
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0],
        [  0,   0,   0, …,   0,   0,   0]],

       [[ 95,  97, 101, …,  28,  27,  13],
        [ 97,  96,  95, …,  27,  22,  14],
        [100,  95,  93, …,  23,  22,  15],
        …,
        [ 90,  92,  97, …,  13,  11,  13],
        [ 90,  93,  94, …,  12,  10,  13],
        [ 89,  98,  95, …,  12,   9,  13]]], dtype=uint8)
```

[15]:
```
val_data.shape
```

[15]:
```
(827, 224, 224)
```

[16]:
```
len(val_labels)
```

[16]:
```
827
```

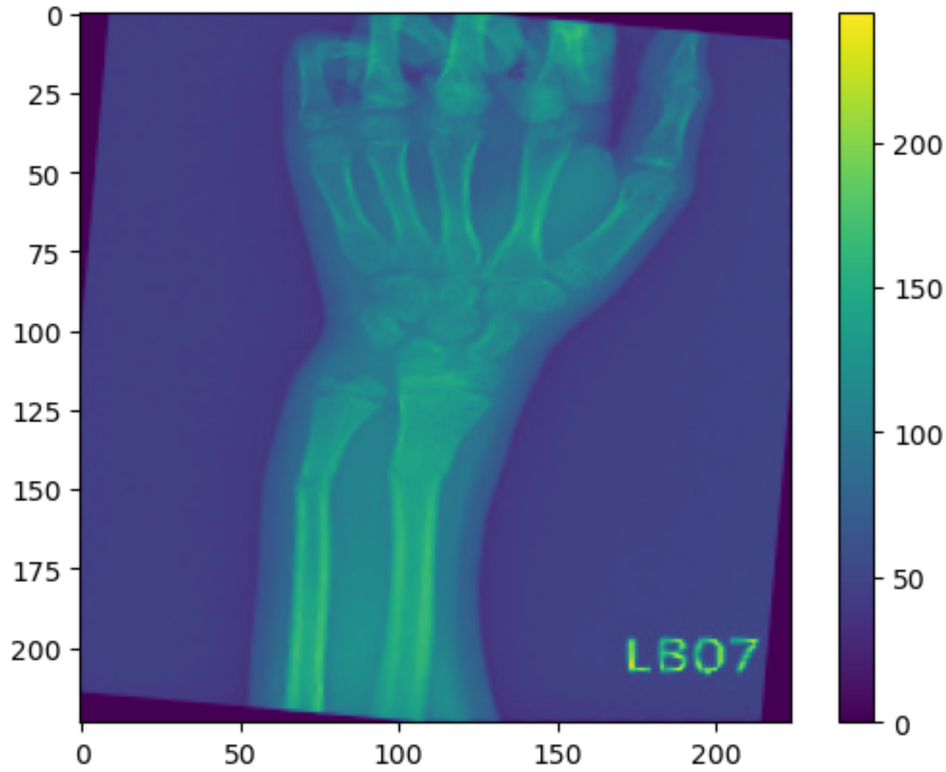
[36]:
```
val_labels
```

[36]:
```
array([1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0,
```

```
0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0,
1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1,
0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0,
0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0,
1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1,
0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0,
0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0,
0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0,
1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0,
0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0,
0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1,
1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0,
0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1,
0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0,
0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1,
0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0,
0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1,
1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1,
1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1,
1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1,
1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1,
0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1,
0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1,
1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1])
```

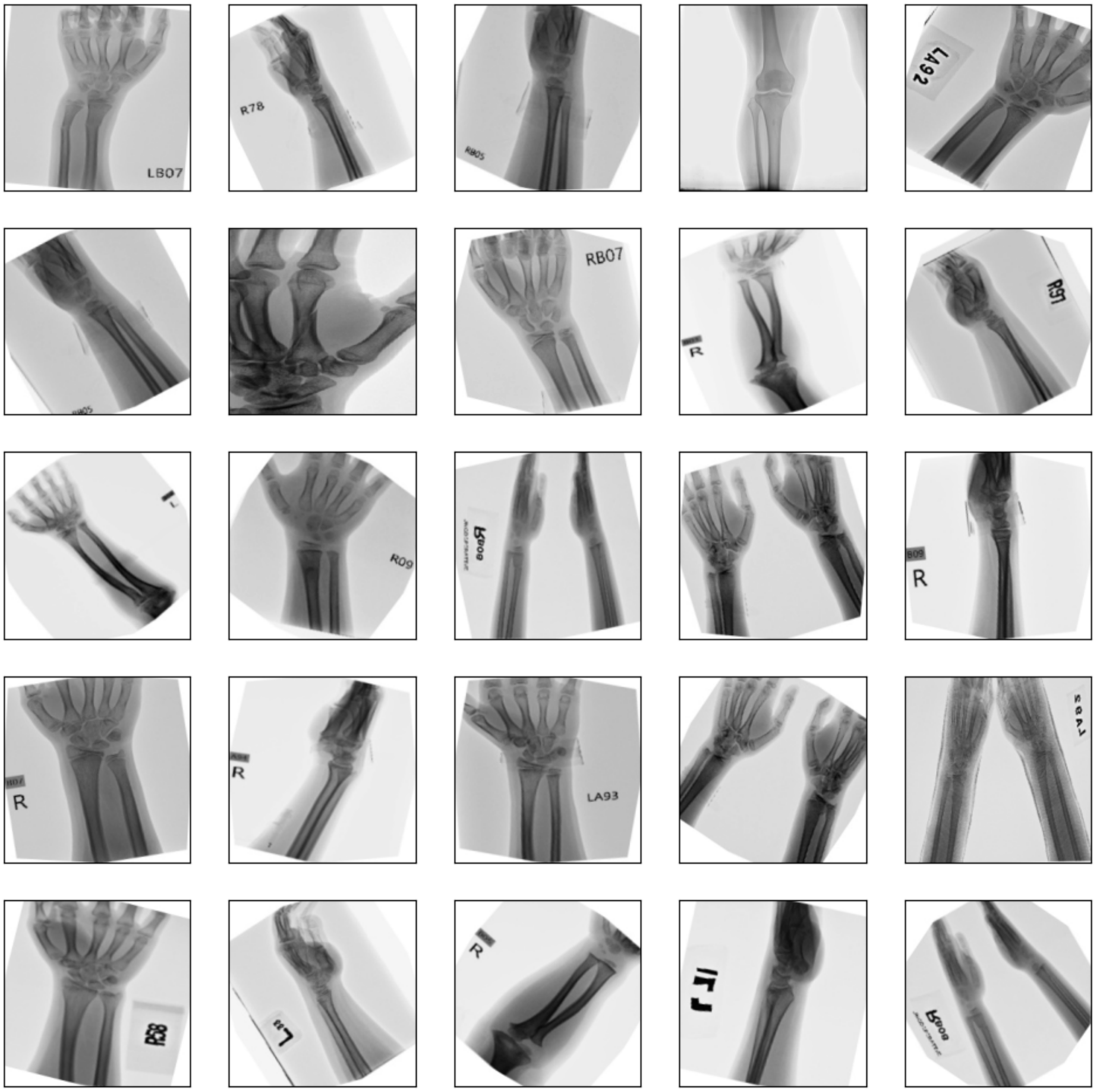
[37]:

```
plt.figure()
plt.imshow(train_data[0])
plt.colorbar()
plt.grid(False)
plt.show()
```



[38]:
```
# normalize to range 0-1
train_images = train_data / 255.0
test_images = test_data / 255.0
val_images = val_data / 255.0
```

[39]:
```
plt.figure(figsize=(12,12))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)

plt.show()
```



1.2 2. DNN

three hidden layers 128,256,128, three dropout layers after each hidden layer, sigmoid for binary classification

[40]:
```
#Model 1: a simple neural network model

def create_model():
    model = Sequential([
        # Flatten the input images to a vector
        Flatten(input_shape=(224, 224, 1)),

        # First Dense layer with ReLU activation
```

```
        Dense(128, activation='relu'),
        BatchNormalization(),  # Normalize the activations of the first layer
        Dropout(0.5),  # Dropout 50% of the nodes of the previous layer during␣
↪training

        # Second Dense layer with more neurons
        Dense(256, activation='relu'),
        BatchNormalization(),  # Normalize the activations of the second layer
        Dropout(0.5),  # Dropout 50% of the nodes

        # Third Dense layer
        Dense(128, activation='relu'),
        BatchNormalization(),  # Normalize the activations of the third layer
        Dropout(0.5),  # Dropout 50% of the nodes

        # Output layer with 1 unit for binary classification
        Dense(1, activation='sigmoid')
    ])

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

# Create the model
model_dnn = create_model()

# Display the model summary to understand its structure
model_dnn.summary()
```

```
/Users/mingentsai/miniconda3/envs/py3k/lib/python3.11/site-
packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 50176)	0
dense (Dense)	(None, 128)	6,422,656
batch_normalization	(None, 128)	512

```
(BatchNormalization)

dropout (Dropout)               (None, 128)                      0

dense_1 (Dense)                 (None, 256)                 33,024

batch_normalization_1           (None, 256)                  1,024
(BatchNormalization)

dropout_1 (Dropout)             (None, 256)                      0

dense_2 (Dense)                 (None, 128)                 32,896

batch_normalization_2           (None, 128)                    512
(BatchNormalization)

dropout_2 (Dropout)             (None, 128)                      0

dense_3 (Dense)                 (None, 1)                      129
```

Total params: 6,490,753 (24.76 MB)

Trainable params: 6,489,729 (24.76 MB)

Non-trainable params: 1,024 (4.00 KB)

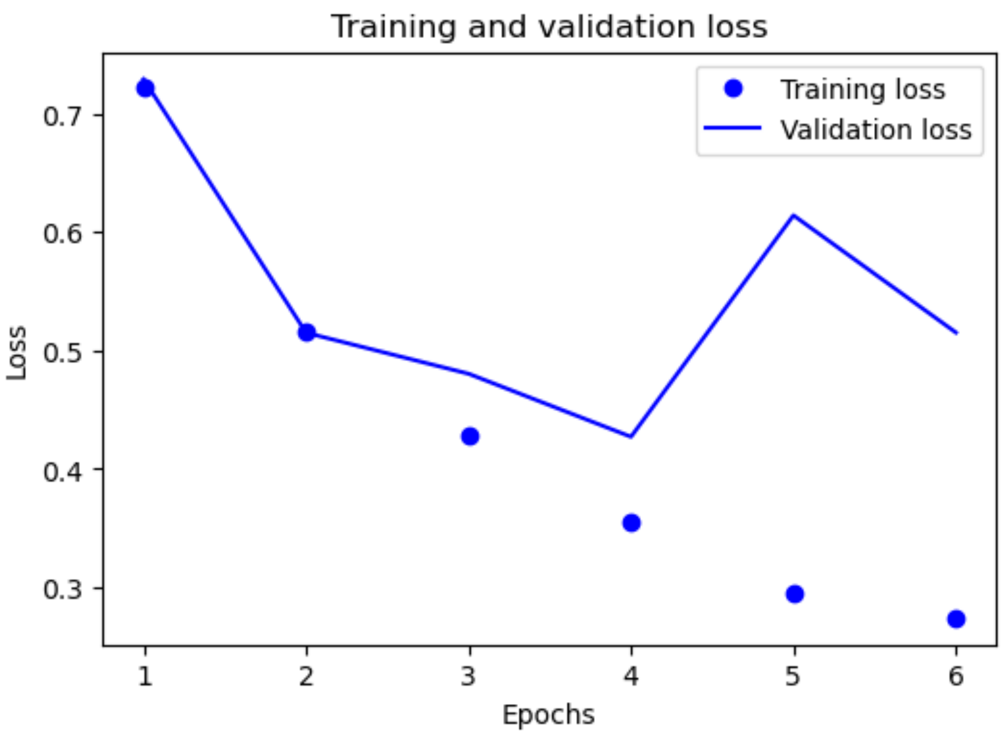
[41]:

```python
from tensorflow import keras

# Define the callback list with early stopping and model checkpoint
callbacks_list = [
    keras.callbacks.EarlyStopping(monitor="val_loss", patience=2),
    keras.callbacks.ModelCheckpoint(filepath="model_dnn_best.keras",␣
 ↪monitor="val_loss", save_best_only=True)
]

# Train the model
history_dnn = model_dnn.fit(
    train_images,
    train_labels,
    epochs=100,
    batch_size=64,
    validation_data=(val_images, val_labels), # Using explicit validation set
    callbacks=callbacks_list)
```

```
Epoch 1/100
144/144              3s 16ms/step -
accuracy: 0.5930 - loss: 0.8238 - val_accuracy: 0.6929 - val_loss: 0.7289
Epoch 2/100
144/144              2s 14ms/step -
accuracy: 0.7350 - loss: 0.5500 - val_accuracy: 0.7630 - val_loss: 0.5149
Epoch 3/100
144/144              2s 14ms/step -
accuracy: 0.7986 - loss: 0.4569 - val_accuracy: 0.7860 - val_loss: 0.4804
Epoch 4/100
144/144              2s 15ms/step -
accuracy: 0.8438 - loss: 0.3649 - val_accuracy: 0.8404 - val_loss: 0.4272
Epoch 5/100
144/144              2s 14ms/step -
accuracy: 0.8683 - loss: 0.3084 - val_accuracy: 0.8356 - val_loss: 0.6140
Epoch 6/100
144/144              2s 14ms/step -
accuracy: 0.8952 - loss: 0.2709 - val_accuracy: 0.8307 - val_loss: 0.5151
```

[42]:

```python
import matplotlib.pyplot as plt

history_dict = history_dnn.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

# Set up the range of epochs
epochs = range(1, len(loss_values) + 1)

# Create a plot of the training and validation loss
plt.figure(figsize=(6, 4))
plt.plot(epochs, loss_values, 'bo', label='Training loss')  # 'bo' for blue dot
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')  # 'b' for
 ↪solid blue line
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



[43]:
```
# Load the best model and evaluate it on the test set
model_best = keras.models.load_model("model_dnn_best.keras")
test_loss, test_acc = model_best.evaluate(test_images, test_labels)
print("Test accuracy:", test_acc)
```

```
16/16              0s 2ms/step -
accuracy: 0.8498 - loss: 0.3726
Test accuracy: 0.8309572339057922
```

1.3 3. CNN

two councolutional layers, sigmoid for binary classification

[46]:
```
def create_model_cnn():
    model02 = Sequential([
        # First convolutional layer with 32 filters, kernel size of 3x3, ReLU␣
↪activation, and same padding
        Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(224,␣
↪224, 1)),
        # First pooling layer which reduces image size by half
        MaxPooling2D((2, 2)),

        # Second convolutional layer with 64 filters
        Conv2D(64, (3, 3), activation='relu', padding='same'),
```

```python
        # Second pooling layer
        MaxPooling2D((2, 2)),

        # Flattening the output of the convolutional layers to feed into the␣
↪dense layer
        Flatten(),

        # Densely connected layer with 64 neurons
        Dense(64, activation='relu'),

        Dense(1, activation='sigmoid')])

    # Compiling the model
    model02.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model02

# Create the CNN model instance
model_cnn = create_model_cnn()

# Display the model summary to understand its structure
model_cnn.summary()
```

```
Model: "sequential_2"


 Layer (type)                     Output Shape                  Param #

 conv2d_2 (Conv2D)                (None, 224, 224, 32)              320

 max_pooling2d_2 (MaxPooling2D)   (None, 112, 112, 32)                0

 conv2d_3 (Conv2D)                (None, 112, 112, 64)           18,496

 max_pooling2d_3 (MaxPooling2D)   (None, 56, 56, 64)                  0

 flatten_2 (Flatten)              (None, 200704)                      0

 dense_6 (Dense)                  (None, 64)                 12,845,120

 dense_7 (Dense)                  (None, 1)                          65


 Total params: 12,864,001 (49.07 MB)
```

Trainable params: 12,864,001 (49.07 MB)

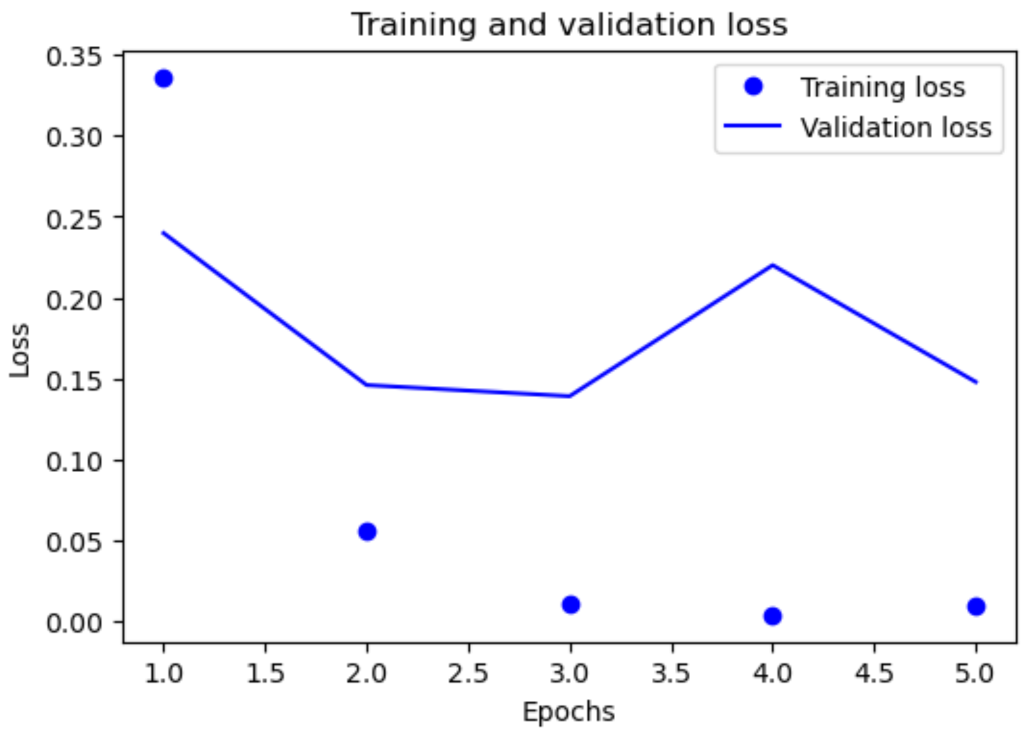
Non-trainable params: 0 (0.00 B)

[47]:
```python
# Define the callback list with early stopping and model checkpoint
callbacks_list = [
    keras.callbacks.EarlyStopping(monitor="val_loss", patience=2),
    keras.callbacks.ModelCheckpoint(filepath="model_cnn_best.keras",␣
 ↪monitor="val_loss", save_best_only=True)
]

# Train the model
history_cnn = model_cnn.fit(
    train_images,
    train_labels,
    epochs=100,
    batch_size=64,
    validation_data=(val_images, val_labels),  # Using explicit validation set
    callbacks=callbacks_list)
```

```
Epoch 1/100
144/144              75s 517ms/step -
accuracy: 0.7183 - loss: 0.5430 - val_accuracy: 0.9045 - val_loss: 0.2399
Epoch 2/100
144/144              73s 506ms/step -
accuracy: 0.9809 - loss: 0.0645 - val_accuracy: 0.9504 - val_loss: 0.1461
Epoch 3/100
144/144              74s 512ms/step -
accuracy: 0.9980 - loss: 0.0109 - val_accuracy: 0.9601 - val_loss: 0.1392
Epoch 4/100
144/144              74s 514ms/step -
accuracy: 0.9982 - loss: 0.0060 - val_accuracy: 0.9541 - val_loss: 0.2201
Epoch 5/100
144/144              73s 510ms/step -
accuracy: 0.9973 - loss: 0.0103 - val_accuracy: 0.9589 - val_loss: 0.1480
```

[49]:
```python
import matplotlib.pyplot as plt

history_dict = history_cnn.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

# Set up the range of epochs
epochs = range(1, len(loss_values) + 1)

# Create a plot of the training and validation loss
```

```
plt.figure(figsize=(6, 4))
plt.plot(epochs, loss_values, 'bo', label='Training loss')  # 'bo' for blue dot
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')  # 'b' for␣
 ↪solid blue line
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



[50]:
```
model_best = keras.models.load_model("model_cnn_best.keras")
test_loss, test_acc = model_best.evaluate(test_images, test_labels)
print("Test accuracy:", test_acc)
```

```
16/16               1s 62ms/step -
accuracy: 0.9677 - loss: 0.0715
Test accuracy: 0.9674134254455566
```

9-final-project-resnet-pre-trained

May 6, 2024

1 II. Pretrained ResNet

[2]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os
from PIL import Image, ImageFile
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten, BatchNormalization
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.layers import Dropout
print(tf.__version__)
import PIL

import time
import numpy as np
import os
import pydot
from typing import List, Tuple
from matplotlib.pyplot import imshow
%matplotlib inline
import matplotlib.pyplot as plt
import PIL.Image
import pathlib
import shutil

import tensorflow as tf
from tensorflow import keras

from tensorflow.keras.preprocessing import image
from tensorflow.keras import layers
```

```
from tensorflow.keras.layers import Input, Add, Dense, Activation,␣
↪ZeroPadding2D, BatchNormalization, Flatten, Conv2D, AveragePooling2D,␣
↪MaxPooling2D, GlobalMaxPooling2D
from tensorflow.keras.initializers import glorot_uniform
from tensorflow.keras.models import Model, load_model

from tensorflow.python.keras.utils import layer_utils
#from tensorflow.keras.utils.vis_utils import model_to_dot
from tensorflow.keras.utils import model_to_dot
from tensorflow.keras.utils import plot_model

from tensorflow.keras.applications.imagenet_utils import preprocess_input

from IPython.display import SVG

import scipy.misc

import tensorflow.keras.backend as K
K.set_image_data_format('channels_last') # can be channels_first or␣
↪channels_last.
#K.set_learning_phase(1) # 1 stands for learning phase
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
```

```
2.16.1
```

[21]:
```
folder_path = '/Users/mingentsai/Desktop/USC/Courses/DSO 569/Homework/Group␣
↪Project/Bone_Fracture_Binary_Classification'
contents = os.listdir(folder_path)
print(contents)
```

```
['.DS_Store', 'test', 'train', 'val']
```

1.1 1. Data Preprocessing

224*224, RGB

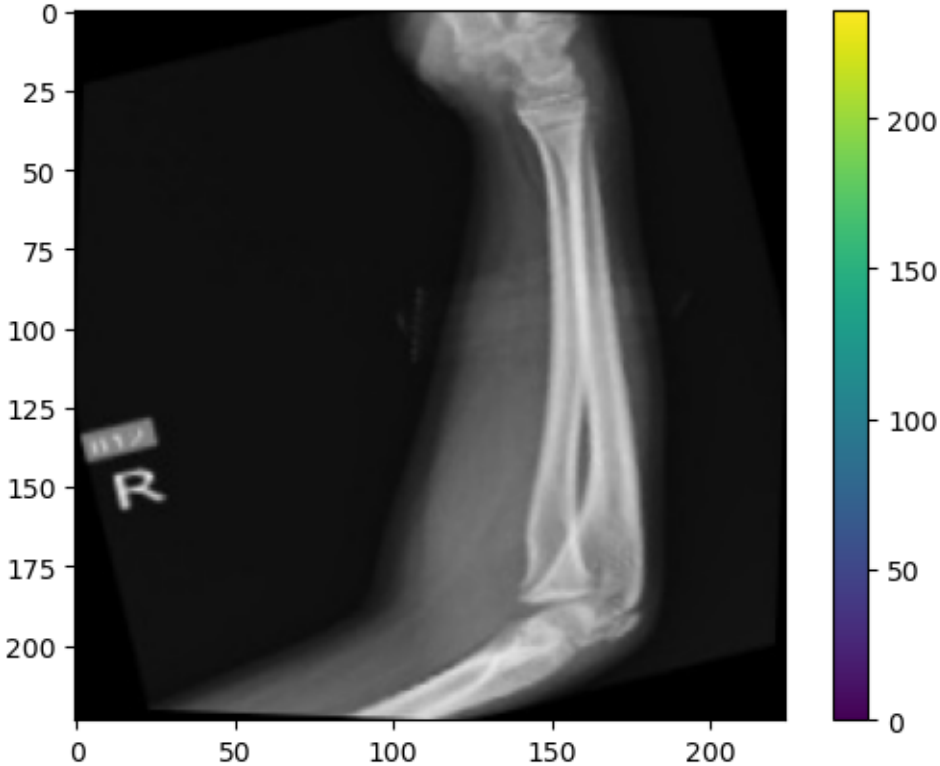
[22]:
```
ImageFile.LOAD_TRUNCATED_IMAGES = True # Allow loading of truncated images

def load_images_from_folder(base_folder):
  data = []
  labels = []
  categories = {'fractured': 1, 'not fractured': 0}
  for category, label in categories.items():
    folder_path = os.path.join(base_folder, category)
    if os.path.isdir(folder_path):
      for filename in os.listdir(folder_path):
```

```
        if filename.lower().endswith(('.jpeg', '.jpg')): # Handle both .jpeg␣
↪and .jpg fi
          img_path = os.path.join(folder_path, filename)
          try:
            with Image.open(img_path) as img:
              img = img.convert('RGB')
              img = img.resize((224, 224),PIL.Image.Resampling.LANCZOS)
              data.append(np.array(img))
              labels.append(label)
          except IOError as e:
            print(f"Error opening or processing image {img_path}: {e}")
        else:
          print(f"Skipped non-JPEG file {filename}")
    else:
      print(f"Directory {folder_path} does not exist")

  # Convert to numpy arrays
  data = np.array(data)
  labels = np.array(labels)

  # Shuffle the data
  idx = np.arange(len(data))
  np.random.shuffle(idx)
  data = data[idx]
  labels = labels[idx]

  return data, labels
```

[23]:
```
train_data, train_labels = load_images_from_folder(os.path.join(folder_path,␣
↪'train'))
val_data, val_labels = load_images_from_folder(os.path.join(folder_path, 'val'))
test_data, test_labels = load_images_from_folder(os.path.join(folder_path,␣
↪'test'))
```

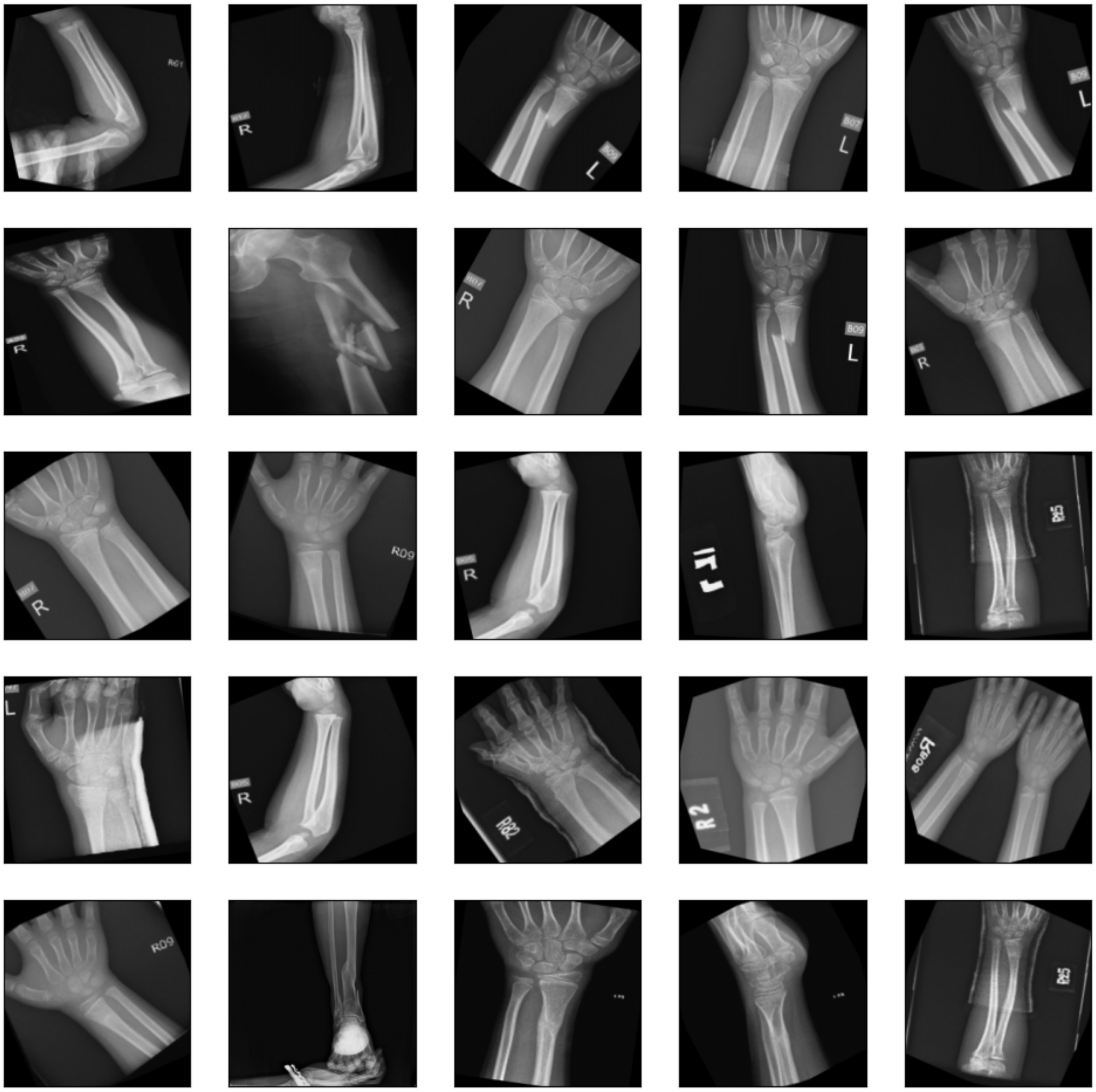
```
Skipped non-JPEG file .DS_Store
Skipped non-JPEG file .DS_Store
Skipped non-JPEG file .DS_Store
Skipped non-JPEG file .DS_Store
Skipped non-JPEG file .DS_Store
```

[24]:
```
plt.figure()
plt.imshow(train_data[1])
plt.colorbar()
plt.grid(False)
plt.show()
```



[25]:
```
# normalize to range 0-1
train_images = train_data / 255.0
test_images = test_data / 255.0
val_images = val_data / 255.0
```

[26]:
```
plt.figure(figsize=(12,12))
for i in range(25):
  plt.subplot(5,5,i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(train_images[i], cmap=plt.cm.binary)
plt.show()
```



```
[ ]: from tensorflow.keras.utils import to_categorical
     # Assuming you have two classes, and train_labels are integer labels like [0, 1]
     # train_labels= to_categorical(train_labels, num_classes=2)
     # test_labels= to_categorical(test_labels, num_classes=2)
     # val_labels= to_categorical(val_labels, num_classes=2)
```

1.2 2. Model fitting

```
[27]: base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224,␣
       ↪224, 3))
```

[10]:
```
#train_images_RGB = np.stack((train_images,train_images,train_images),axis=3)
#test_images_RGB = np.stack((test_images,test_images,test_images),axis=3)
#val_images_RGB = np.stack((val_images,val_images,val_images),axis=3)
#test_images_RGB.shape
```

[10]: (491, 224, 224, 3)

[28]:
```
x = base_model.output
x = GlobalAveragePooling2D()(x)
predictions = Dense(1, activation='sigmoid')(x)
# For binary classification, use ‘sigmoid’; for multi-class, use ‘softmax’
```

[29]:
```
model = Model(inputs=base_model.input, outputs=predictions)
```

[30]:
```
model.compile(
    optimizer='adam', # optimizer
    loss='binary_crossentropy', # loss function to optimize
    metrics=['accuracy'] # metrics to monitor
)
```

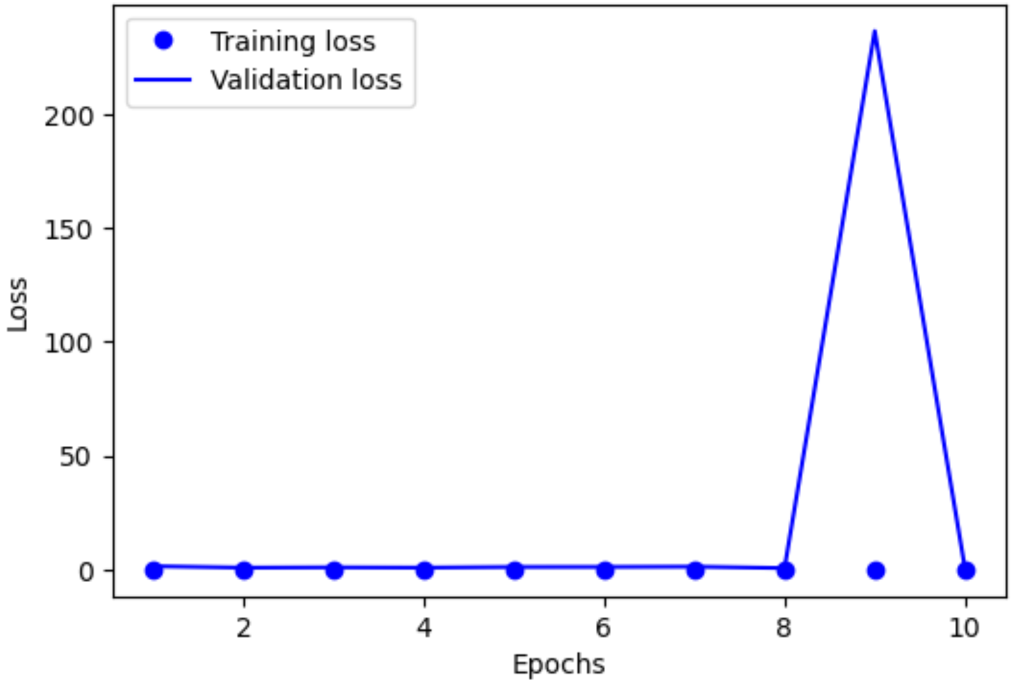
[31]:
```
callbacks_list = [
    #tf.keras.callbacks.EarlyStopping(
        #monitor="val_accuracy",
        #patience=2,),
    tf.keras.callbacks.ModelCheckpoint(
        filepath="checkpoint_path.keras",
        monitor="val_accuracy",
        save_best_only=True,
    )
]



history = model.fit(train_images,  train_labels, epochs=10,
          batch_size=64, validation_data=(val_images, val_labels),
          callbacks=callbacks_list)
```

```
Epoch 1/10
144/144              843s 6s/step -
accuracy: 0.8662 - loss: 0.3728 - val_accuracy: 0.5925 - val_loss: 1.3040
Epoch 2/10
144/144              820s 6s/step -
accuracy: 0.9739 - loss: 0.0850 - val_accuracy: 0.5925 - val_loss: 0.6833
Epoch 3/10
144/144              814s 6s/step -
accuracy: 0.9895 - loss: 0.0345 - val_accuracy: 0.5925 - val_loss: 0.8066
Epoch 4/10
```

```
144/144               799s 6s/step -
accuracy: 0.9972 - loss: 0.0097 - val_accuracy: 0.3736 - val_loss: 0.6957
Epoch 5/10
144/144               820s 6s/step -
accuracy: 0.9884 - loss: 0.0337 - val_accuracy: 0.4208 - val_loss: 0.9673
Epoch 6/10
144/144               807s 6s/step -
accuracy: 0.9966 - loss: 0.0091 - val_accuracy: 0.6288 - val_loss: 0.9960
Epoch 7/10
144/144               802s 6s/step -
accuracy: 1.0000 - loss: 8.0597e-04 - val_accuracy: 0.7497 - val_loss: 1.1062
Epoch 8/10
144/144               802s 6s/step -
accuracy: 1.0000 - loss: 3.4829e-04 - val_accuracy: 0.8924 - val_loss: 0.4970
Epoch 9/10
144/144               804s 6s/step -
accuracy: 0.9933 - loss: 0.0217 - val_accuracy: 0.4099 - val_loss: 236.7034
Epoch 10/10
144/144               795s 6s/step -
accuracy: 0.9881 - loss: 0.0455 - val_accuracy: 0.8525 - val_loss: 0.5135
```

[32]:

```
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
# Set up the range of epochs
epochs = range(1, len(loss_values) + 1)
# Create a plot of the training and validation loss
plt.figure(figsize=(6, 4))
plt.plot(epochs, loss_values, 'bo', label='Training loss') # 'bo' for blue dot
plt.plot(epochs, val_loss_values, 'b', label='Validation loss') # 'b' for solid␣
↪blue line plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



[33]:
```
model_best = keras.models.load_model("checkpoint_path.keras")
test_loss, test_acc = model_best.evaluate(test_images_RGB, test_labels)
print("Test accuracy:", test_acc)
```

```
16/16            10s 584ms/step -
accuracy: 0.4553 - loss: 4.4207
Test accuracy: 0.48472505807876587
```

dso569-final-project-resnet-custom

May 6, 2024

1 III. Custom ResNet50

[35]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os
#from google.colab import drive
from PIL import Image, ImageFile
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten, BatchNormalization
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.layers import Dropout
print(tf.__version__)
import PIL

import time
import numpy as np
import os
import pydot
from typing import List, Tuple
from matplotlib.pyplot import imshow
%matplotlib inline
import matplotlib.pyplot as plt
import PIL.Image
import pathlib
import shutil

import tensorflow as tf
from tensorflow import keras

from tensorflow.keras.preprocessing import image
from tensorflow.keras import layers
```

```
from tensorflow.keras.layers import Input, Add, Dense, Activation,␣
↪ZeroPadding2D, BatchNormalization, Flatten, Conv2D, AveragePooling2D,␣
↪MaxPooling2D, GlobalMaxPooling2D
from tensorflow.keras.initializers import glorot_uniform
from tensorflow.keras.models import Model, load_model

from tensorflow.python.keras.utils import layer_utils
#from tensorflow.keras.utils.vis_utils import model_to_dot
from tensorflow.keras.utils import model_to_dot
from tensorflow.keras.utils import plot_model

from tensorflow.keras.applications.imagenet_utils import preprocess_input

from IPython.display import SVG

import scipy.misc

import tensorflow.keras.backend as K
K.set_image_data_format('channels_last') # can be channels_first or␣
↪channels_last.
#K.set_learning_phase(1) # 1 stands for learning phase
```

```
2.16.1
```

1.1 1. Data Preprocessing

224*224, RGB

[36]:
```
#drive.mount('/content/drive')

# List contents of Google Drive root directory
folder_path = '/Users/mingentsai/Desktop/USC/Courses/DSO 569/Homework/Group␣
↪Project/Bone_Fracture_Binary_Classification'
contents = os.listdir(folder_path)
print(contents)
```

```
['.DS_Store', 'test', 'train', 'val']
```

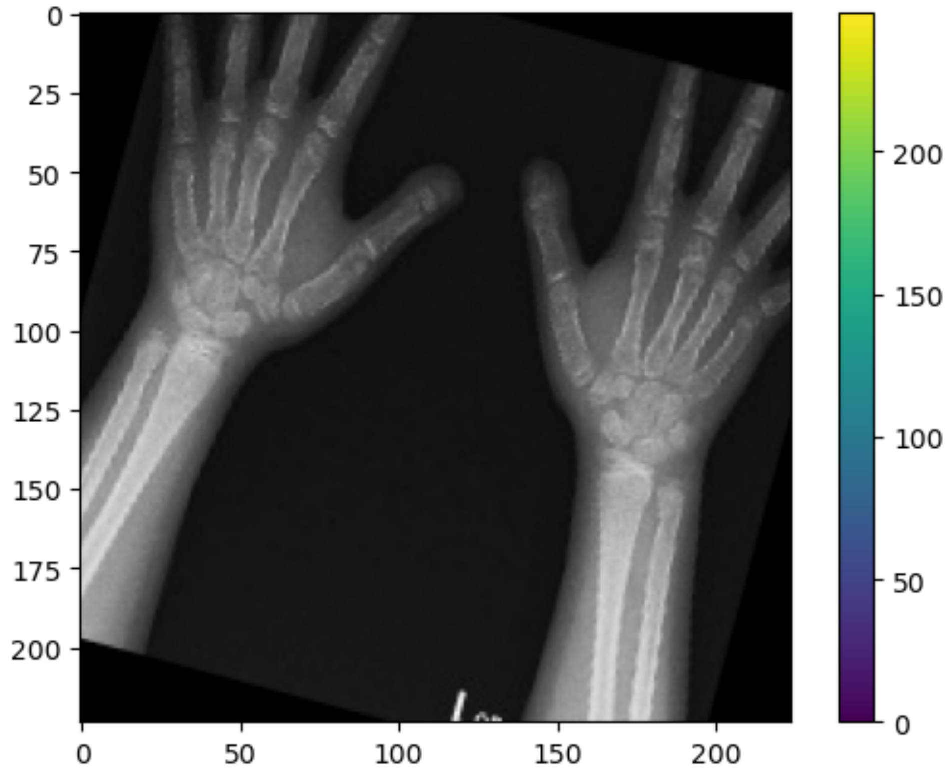
[74]:
```
ImageFile.LOAD_TRUNCATED_IMAGES = True # Allow loading of truncated images

def load_images_from_folder(base_folder):
  data = []
  labels = []
  categories = {'fractured': 1, 'not fractured': 0}
  for category, label in categories.items():
    folder_path = os.path.join(base_folder, category)
    if os.path.isdir(folder_path):
      for filename in os.listdir(folder_path):
```

```
    if filename.lower().endswith(('.jpeg', '.jpg')): # Handle both .jpeg␣
↪and .jpg fi
      img_path = os.path.join(folder_path, filename)
      try:
        with Image.open(img_path) as img:
          img = img.convert('RGB')
          img = img.resize((224, 224),PIL.Image.Resampling.LANCZOS)
          data.append(np.array(img))
          labels.append(label)
      except IOError as e:
        print(f"Error opening or processing image {img_path}: {e}")
    else:
      print(f"Skipped non-JPEG file {filename}")
  else:
    print(f"Directory {folder_path} does not exist")

# Convert to numpy arrays
data = np.array(data)
labels = np.array(labels)

# Shuffle the data
idx = np.arange(len(data))
np.random.shuffle(idx)
data = data[idx]
labels = labels[idx]

return data, labels
```

[75]:
```
train_data, train_labels = load_images_from_folder(os.path.join(folder_path,␣
↪'train'))
val_data, val_labels = load_images_from_folder(os.path.join(folder_path, 'val'))
test_data, test_labels = load_images_from_folder(os.path.join(folder_path,␣
↪'test'))
```

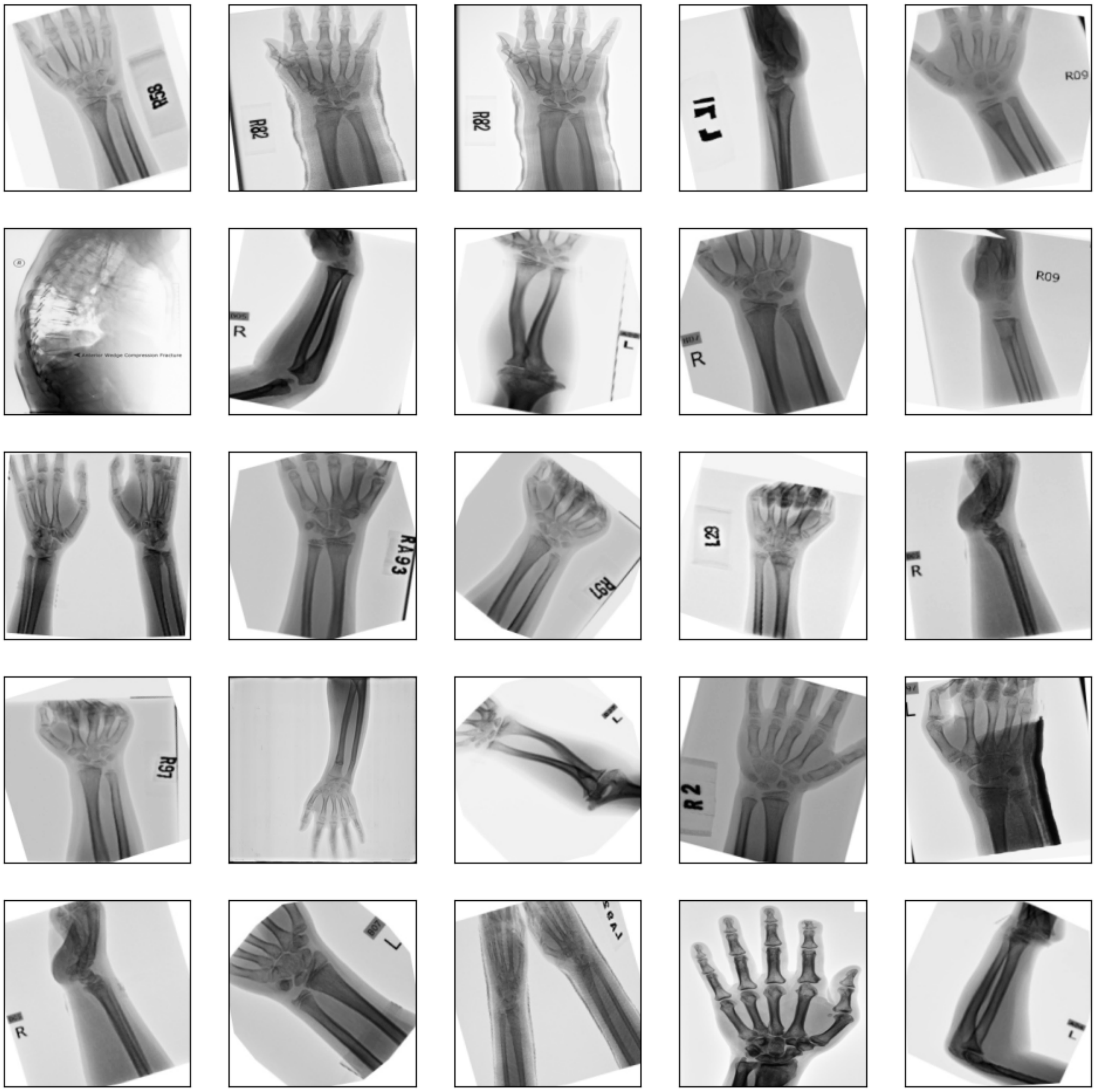
```
Skipped non-JPEG file .DS_Store
Skipped non-JPEG file .DS_Store
Skipped non-JPEG file .DS_Store
Skipped non-JPEG file .DS_Store
Skipped non-JPEG file .DS_Store
```

[76]:
```
plt.figure()
plt.imshow(train_data[1])
plt.colorbar()
plt.grid(False)
plt.show()
```



[77]:
```
# normalize to range 0-1
train_images = train_data / 255.0
test_images = test_data / 255.0
val_images = val_data / 255.0
```

[78]:
```
plt.figure(figsize=(12,12))
for i in range(25):
  plt.subplot(5,5,i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(train[i], cmap=plt.cm.binary)
plt.show()
```



```
[80]: train_images.shape
```

```
[80]: (9200, 224, 224, 3)
```

```
[81]: from tensorflow.keras.utils import to_categorical

      # Assuming you have two classes, and train_labels are integer labels like [0, 1]
      train_labels= to_categorical(train_labels, num_classes=2)
      test_labels= to_categorical(test_labels, num_classes=2)
      val_labels= to_categorical(val_labels, num_classes=2)
```

```
[102]: train_labels[:20]
```

[102]:
```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.]])
```

[103]:
```
val_labels[:20]
```

[103]:
```
array([[0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.]])
```

1.2 2. Model fitting

[88]:
```python
def identity_block(X: tf.Tensor, level: int, block: int, filters: list[int]) ->␣
↪tf.Tensor:
    """
    Creates an identity block (see figure 3.1 from readme)

    Input:
        X - input tensor of shape (m, height_prev, width_prev, chan_prev)
        level - integer, one of the 5 levels that our networks is conceptually␣
↪divided into (see figure 3.1 in the readme file)
              - level names have the form: conv2_x, conv3_x ... conv5_x
        block - each conceptual level has multiple blocks (1 identity and␣
↪several convolutional blocks)
                block is the number of this block within its conceptual layer
                i.e. first block from level 2 will be named conv2_1
        filters - a list on integers, each of them defining the number of␣
↪filters in each convolutional layer

    Output:
        X - tensor (m, height, width, chan)
    """

    # layers will be called␣
↪conv{level}_iden{block}_{convlayer_number_within_block}'
    conv_name = f'conv{level}_{block}' + '_{layer}_{type}'

    # unpack number of filters to be used for each conv layer
    f1, f2, f3 = filters

    # the shortcut branch of the identity block
    # takes the value of the block input
    X_shortcut = X

    # first convolutional layer (plus batch norm & relu activation, of course)
    X = Conv2D(filters=f1, kernel_size=(1, 1), strides=(1, 1),
               padding='valid', name=conv_name.format(layer=1, type='conv'),
               kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=conv_name.format(layer=1, type='bn'))(X)
    X = Activation('relu', name=conv_name.format(layer=1, type='relu'))(X)

    # second convolutional layer
    X = Conv2D(filters=f2, kernel_size=(3, 3), strides=(1, 1),
               padding='same', name=conv_name.format(layer=2, type='conv'),
               kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=conv_name.format(layer=2, type='bn'))(X)
    X = Activation('relu')(X)
```

```
    # third convolutional layer
    X = Conv2D(filters=f3, kernel_size=(1, 1), strides=(1, 1),
               padding='valid', name=conv_name.format(layer=3, type='conv'),
               kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=conv_name.format(layer=3, type='bn'))(X)

    # add shortcut branch to main path
    X = Add()([X, X_shortcut])

    # relu activation at the end of the block
    X = Activation('relu', name=conv_name.format(layer=3, type='relu'))(X)

    return X
```

[89]:
```
def convolutional_block(X: tf.Tensor, level: int, block: int, filters:␣
↪list[int], s: tuple[int,int,int]=(2, 2)) -> tf.Tensor:
    """
    Creates a convolutional block (see figure 3.1 from readme)

    Input:
        X - input tensor of shape (m, height_prev, width_prev, chan_prev)
        level - integer, one of the 5 levels that our networks is conceptually␣
↪divided into (see figure 3.1 in the readme file)
              - level names have the form: conv2_x, conv3_x ... conv5_x
        block - each conceptual level has multiple blocks (1 identity and␣
↪several convolutional blocks)
                block is the number of this block within its conceptual layer
                i.e. first block from level 2 will be named conv2_1
        filters - a list on integers, each of them defining the number of␣
↪filters in each convolutional layer
        s   - stride of the first layer;
            - a conv layer with a filter that has a stride of 2 will reduce the␣
↪width and height of its input by half

    Output:
        X - tensor (m, height, width, chan)
    """

    # layers will be called conv{level}_{block}_{convlayer_number_within_block}'
    conv_name = f'conv{level}_{block}' + '_{layer}_{type}'

    # unpack number of filters to be used for each conv layer
    f1, f2, f3 = filters

    # the shortcut branch of the convolutional block
    X_shortcut = X
```

```
    # first convolutional layer
    X = Conv2D(filters=f1, kernel_size=(1, 1), strides=s, padding='valid',
               name=conv_name.format(layer=1, type='conv'),
               kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=conv_name.format(layer=1, type='bn'))(X)
    X = Activation('relu', name=conv_name.format(layer=1, type='relu'))(X)

    # second convolutional layer
    X = Conv2D(filters=f2, kernel_size=(3, 3), strides=(1, 1), padding='same',
               name=conv_name.format(layer=2, type='conv'),
               kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=conv_name.format(layer=2, type='bn'))(X)
    X = Activation('relu', name=conv_name.format(layer=2, type='relu'))(X)

    # third convolutional layer
    X = Conv2D(filters=f3, kernel_size=(1, 1), strides=(1, 1), padding='valid',
               name=conv_name.format(layer=3, type='conv'),
               kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=conv_name.format(layer=3, type='bn'))(X)

    # shortcut path
    X_shortcut = Conv2D(filters=f3, kernel_size=(1, 1), strides=s,␣
↪padding='valid',
                        name=conv_name.format(layer='short', type='conv'),
                        kernel_initializer=glorot_uniform(seed=0))(X_shortcut)
    X_shortcut = BatchNormalization(axis=3, name=conv_name.
↪format(layer='short', type='bn'))(X_shortcut)

    # add shortcut branch to main path
    X = Add()([X, X_shortcut])

    # nonlinearity
    X = Activation('relu', name=conv_name.format(layer=3, type='relu'))(X)

    return X
```

[92]:
```
def ResNet50(input_size: tuple[int,int,int], classes: int) -> Model:
    """
        Builds the ResNet50 model (see figure 4.2 from readme)

        Input:
            - input_size - a (height, width, chan) tuple, the shape of the␣
↪input images
            - classes - number of classes the model must learn

        Output:
```

```
            model - a Keras Model() instance
    """

    # tensor placeholder for the model's input
    X_input = Input(input_size)

    ### Level 1 ###

    # padding
    X = ZeroPadding2D((3, 3))(X_input)

    # convolutional layer, followed by batch normalization and relu activation
    X = Conv2D(filters=64, kernel_size=(7, 7), strides=(2, 2),
               name='conv1_1_1_conv',
               kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name='conv1_1_1_nb')(X)
    X = Activation('relu')(X)

    ### Level 2 ###

    # max pooling layer to halve the size coming from the previous layer
    X = MaxPooling2D((3, 3), strides=(2, 2))(X)

    # 1x convolutional block
    X = convolutional_block(X, level=2, block=1, filters=[64, 64, 256], s=(1,␣
↪1))

    # 2x identity blocks
    X = identity_block(X, level=2, block=2, filters=[64, 64, 256])
    X = identity_block(X, level=2, block=3, filters=[64, 64, 256])

    ### Level 3 ###

    # 1x convolutional block
    X = convolutional_block(X, level=3, block=1, filters=[128, 128, 512], s=(2,␣
↪2))

    # 3x identity blocks
    X = identity_block(X, level=3, block=2, filters=[128, 128, 512])
    X = identity_block(X, level=3, block=3, filters=[128, 128, 512])
    X = identity_block(X, level=3, block=4, filters=[128, 128, 512])

    ### Level 4 ###
    # 1x convolutional block
    X = convolutional_block(X, level=4, block=1, filters=[256, 256, 1024],␣
↪s=(2, 2))
    # 5x identity blocks
```

```
    X = identity_block(X, level=4, block=2, filters=[256, 256, 1024])
    X = identity_block(X, level=4, block=3, filters=[256, 256, 1024])
    X = identity_block(X, level=4, block=4, filters=[256, 256, 1024])
    X = identity_block(X, level=4, block=5, filters=[256, 256, 1024])
    X = identity_block(X, level=4, block=6, filters=[256, 256, 1024])

    ### Level 5 ###
    # 1x convolutional block
    X = convolutional_block(X, level=5, block=1, filters=[512, 512, 2048],␣
↪s=(2, 2))
    # 2x identity blocks
    X = identity_block(X, level=5, block=2, filters=[512, 512, 2048])
    X = identity_block(X, level=5, block=3, filters=[512, 512, 2048])

    # Pooling layers
    X = AveragePooling2D(pool_size=(2, 2), name='avg_pool')(X)

    # Output layer
    X = Flatten()(X)
    X = Dense(classes, activation='softmax', name='fc_' + str(classes),
              kernel_initializer=glorot_uniform(seed=0))(X)

    # Create model
    model = Model(inputs=X_input, outputs=X, name='ResNet50')

    return model
```

[91]:
```
image_size = (224, 224)
channels = 3
num_classes = 2
```

[93]:
```
model = ResNet50(input_size = (image_size[1], image_size[0], channels), classes␣
↪= num_classes)
```

[94]:
```
model.summary()
```

```
Model: "ResNet50"


 Layer (type)          Output Shape           Param #   Connected to

 input_layer_4         (None, 224, 224,             0   -
 (InputLayer)          3)

 zero_padding2d_4      (None, 230, 230,             0   input_layer_4[0]…
 (ZeroPadding2D)       3)
```

conv1_1_1_conv (Conv2D)	(None, 112, 112, 64)	9,472	zero_padding2d_4…
conv1_1_1_nb (BatchNormalizatio…	(None, 112, 112, 64)	256	conv1_1_1_conv[0…
activation_52 (Activation)	(None, 112, 112, 64)	0	conv1_1_1_nb[0][…
max_pooling2d_4 (MaxPooling2D)	(None, 55, 55, 64)	0	activation_52[0]…
conv2_1_1_conv (Conv2D)	(None, 55, 55, 64)	4,160	max_pooling2d_4[…
conv2_1_1_bn (BatchNormalizatio…	(None, 55, 55, 64)	256	conv2_1_1_conv[0…
conv2_1_1_relu (Activation)	(None, 55, 55, 64)	0	conv2_1_1_bn[0][…
conv2_1_2_conv (Conv2D)	(None, 55, 55, 64)	36,928	conv2_1_1_relu[0…
conv2_1_2_bn (BatchNormalizatio…	(None, 55, 55, 64)	256	conv2_1_2_conv[0…
conv2_1_2_relu (Activation)	(None, 55, 55, 64)	0	conv2_1_2_bn[0][…
conv2_1_3_conv (Conv2D)	(None, 55, 55, 256)	16,640	conv2_1_2_relu[0…
conv2_1_short_conv (Conv2D)	(None, 55, 55, 256)	16,640	max_pooling2d_4[…
conv2_1_3_bn (BatchNormalizatio…	(None, 55, 55, 256)	1,024	conv2_1_3_conv[0…
conv2_1_short_bn (BatchNormalizatio…	(None, 55, 55, 256)	1,024	conv2_1_short_co…
add_64 (Add)	(None, 55, 55, 256)	0	conv2_1_3_bn[0][… conv2_1_short_bn…
conv2_1_3_relu (Activation)	(None, 55, 55, 256)	0	add_64[0][0]

conv2_2_1_conv (Conv2D)	(None, 55, 55, 64)	16,448	conv2_1_3_relu[0…
conv2_2_1_bn (BatchNormalizatio…	(None, 55, 55, 64)	256	conv2_2_1_conv[0…
conv2_2_1_relu (Activation)	(None, 55, 55, 64)	0	conv2_2_1_bn[0][…
conv2_2_2_conv (Conv2D)	(None, 55, 55, 64)	36,928	conv2_2_1_relu[0…
conv2_2_2_bn (BatchNormalizatio…	(None, 55, 55, 64)	256	conv2_2_2_conv[0…
activation_53 (Activation)	(None, 55, 55, 64)	0	conv2_2_2_bn[0][…
conv2_2_3_conv (Conv2D)	(None, 55, 55, 256)	16,640	activation_53[0]…
conv2_2_3_bn (BatchNormalizatio…	(None, 55, 55, 256)	1,024	conv2_2_3_conv[0…
add_65 (Add)	(None, 55, 55, 256)	0	conv2_2_3_bn[0][… conv2_1_3_relu[0…
conv2_2_3_relu (Activation)	(None, 55, 55, 256)	0	add_65[0][0]
conv2_3_1_conv (Conv2D)	(None, 55, 55, 64)	16,448	conv2_2_3_relu[0…
conv2_3_1_bn (BatchNormalizatio…	(None, 55, 55, 64)	256	conv2_3_1_conv[0…
conv2_3_1_relu (Activation)	(None, 55, 55, 64)	0	conv2_3_1_bn[0][…
conv2_3_2_conv (Conv2D)	(None, 55, 55, 64)	36,928	conv2_3_1_relu[0…
conv2_3_2_bn (BatchNormalizatio…	(None, 55, 55, 64)	256	conv2_3_2_conv[0…
activation_54 (Activation)	(None, 55, 55, 64)	0	conv2_3_2_bn[0][…

conv2_3_3_conv (Conv2D)	(None, 55, 55, 256)	16,640	activation_54[0]…
conv2_3_3_bn (BatchNormalizatio…	(None, 55, 55, 256)	1,024	conv2_3_3_conv[0…
add_66 (Add)	(None, 55, 55, 256)	0	conv2_3_3_bn[0][… conv2_2_3_relu[0…
conv2_3_3_relu (Activation)	(None, 55, 55, 256)	0	add_66[0][0]
conv3_1_1_conv (Conv2D)	(None, 28, 28, 128)	32,896	conv2_3_3_relu[0…
conv3_1_1_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_1_1_conv[0…
conv3_1_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_1_1_bn[0][…
conv3_1_2_conv (Conv2D)	(None, 28, 28, 128)	147,584	conv3_1_1_relu[0…
conv3_1_2_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_1_2_conv[0…
conv3_1_2_relu (Activation)	(None, 28, 28, 128)	0	conv3_1_2_bn[0][…
conv3_1_3_conv (Conv2D)	(None, 28, 28, 512)	66,048	conv3_1_2_relu[0…
conv3_1_short_conv (Conv2D)	(None, 28, 28, 512)	131,584	conv2_3_3_relu[0…
conv3_1_3_bn (BatchNormalizatio…	(None, 28, 28, 512)	2,048	conv3_1_3_conv[0…
conv3_1_short_bn (BatchNormalizatio…	(None, 28, 28, 512)	2,048	conv3_1_short_co…
add_67 (Add)	(None, 28, 28, 512)	0	conv3_1_3_bn[0][… conv3_1_short_bn…
conv3_1_3_relu (Activation)	(None, 28, 28, 512)	0	add_67[0][0]

conv3_2_1_conv (Conv2D)	(None, 28, 28, 128)	65,664	conv3_1_3_relu[0…
conv3_2_1_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_2_1_conv[0…
conv3_2_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_2_1_bn[0][…
conv3_2_2_conv (Conv2D)	(None, 28, 28, 128)	147,584	conv3_2_1_relu[0…
conv3_2_2_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_2_2_conv[0…
activation_55 (Activation)	(None, 28, 28, 128)	0	conv3_2_2_bn[0][…
conv3_2_3_conv (Conv2D)	(None, 28, 28, 512)	66,048	activation_55[0]…
conv3_2_3_bn (BatchNormalizatio…	(None, 28, 28, 512)	2,048	conv3_2_3_conv[0…
add_68 (Add)	(None, 28, 28, 512)	0	conv3_2_3_bn[0][… conv3_1_3_relu[0…
conv3_2_3_relu (Activation)	(None, 28, 28, 512)	0	add_68[0][0]
conv3_3_1_conv (Conv2D)	(None, 28, 28, 128)	65,664	conv3_2_3_relu[0…
conv3_3_1_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_3_1_conv[0…
conv3_3_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_3_1_bn[0][…
conv3_3_2_conv (Conv2D)	(None, 28, 28, 128)	147,584	conv3_3_1_relu[0…
conv3_3_2_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_3_2_conv[0…
activation_56 (Activation)	(None, 28, 28, 128)	0	conv3_3_2_bn[0][…

conv3_3_3_conv (Conv2D)	(None, 28, 28, 512)	66,048	activation_56[0]…
conv3_3_3_bn (BatchNormalizatio…	(None, 28, 28, 512)	2,048	conv3_3_3_conv[0…
add_69 (Add)	(None, 28, 28, 512)	0	conv3_3_3_bn[0][… conv3_2_3_relu[0…
conv3_3_3_relu (Activation)	(None, 28, 28, 512)	0	add_69[0][0]
conv3_4_1_conv (Conv2D)	(None, 28, 28, 128)	65,664	conv3_3_3_relu[0…
conv3_4_1_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_4_1_conv[0…
conv3_4_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_4_1_bn[0][…
conv3_4_2_conv (Conv2D)	(None, 28, 28, 128)	147,584	conv3_4_1_relu[0…
conv3_4_2_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_4_2_conv[0…
activation_57 (Activation)	(None, 28, 28, 128)	0	conv3_4_2_bn[0][…
conv3_4_3_conv (Conv2D)	(None, 28, 28, 512)	66,048	activation_57[0]…
conv3_4_3_bn (BatchNormalizatio…	(None, 28, 28, 512)	2,048	conv3_4_3_conv[0…
add_70 (Add)	(None, 28, 28, 512)	0	conv3_4_3_bn[0][… conv3_3_3_relu[0…
conv3_4_3_relu (Activation)	(None, 28, 28, 512)	0	add_70[0][0]
conv4_1_1_conv (Conv2D)	(None, 14, 14, 256)	131,328	conv3_4_3_relu[0…
conv4_1_1_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_1_1_conv[0…

conv4_1_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_1_1_bn[0][…
conv4_1_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_1_1_relu[0…
conv4_1_2_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_1_2_conv[0…
conv4_1_2_relu (Activation)	(None, 14, 14, 256)	0	conv4_1_2_bn[0][…
conv4_1_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	conv4_1_2_relu[0…
conv4_1_short_conv (Conv2D)	(None, 14, 14, 1024)	525,312	conv3_4_3_relu[0…
conv4_1_3_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_1_3_conv[0…
conv4_1_short_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_1_short_co…
add_71 (Add)	(None, 14, 14, 1024)	0	conv4_1_3_bn[0][… conv4_1_short_bn…
conv4_1_3_relu (Activation)	(None, 14, 14, 1024)	0	add_71[0][0]
conv4_2_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_1_3_relu[0…
conv4_2_1_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_2_1_conv[0…
conv4_2_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_2_1_bn[0][…
conv4_2_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_2_1_relu[0…
conv4_2_2_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_2_2_conv[0…
activation_58 (Activation)	(None, 14, 14, 256)	0	conv4_2_2_bn[0][…

conv4_2_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	activation_58[0]…
conv4_2_3_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_2_3_conv[0…
add_72 (Add)	(None, 14, 14, 1024)	0	conv4_2_3_bn[0][… conv4_1_3_relu[0…
conv4_2_3_relu (Activation)	(None, 14, 14, 1024)	0	add_72[0][0]
conv4_3_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_2_3_relu[0…
conv4_3_1_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_3_1_conv[0…
conv4_3_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_3_1_bn[0][…
conv4_3_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_3_1_relu[0…
conv4_3_2_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_3_2_conv[0…
activation_59 (Activation)	(None, 14, 14, 256)	0	conv4_3_2_bn[0][…
conv4_3_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	activation_59[0]…
conv4_3_3_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_3_3_conv[0…
add_73 (Add)	(None, 14, 14, 1024)	0	conv4_3_3_bn[0][… conv4_2_3_relu[0…
conv4_3_3_relu (Activation)	(None, 14, 14, 1024)	0	add_73[0][0]
conv4_4_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_3_3_relu[0…
conv4_4_1_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_4_1_conv[0…

conv4_4_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_4_1_bn[0][…
conv4_4_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_4_1_relu[0…
conv4_4_2_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_4_2_conv[0…
activation_60 (Activation)	(None, 14, 14, 256)	0	conv4_4_2_bn[0][…
conv4_4_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	activation_60[0]…
conv4_4_3_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_4_3_conv[0…
add_74 (Add)	(None, 14, 14, 1024)	0	conv4_4_3_bn[0][… conv4_3_3_relu[0…
conv4_4_3_relu (Activation)	(None, 14, 14, 1024)	0	add_74[0][0]
conv4_5_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_4_3_relu[0…
conv4_5_1_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_5_1_conv[0…
conv4_5_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_5_1_bn[0][…
conv4_5_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_5_1_relu[0…
conv4_5_2_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_5_2_conv[0…
activation_61 (Activation)	(None, 14, 14, 256)	0	conv4_5_2_bn[0][…
conv4_5_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	activation_61[0]…
conv4_5_3_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_5_3_conv[0…

add_75 (Add)	(None, 14, 14, 1024)	0	conv4_5_3_bn[0][… conv4_4_3_relu[0…
conv4_5_3_relu (Activation)	(None, 14, 14, 1024)	0	add_75[0][0]
conv4_6_1_conv (Conv2D)	(None, 14, 14, 256)	262,400	conv4_5_3_relu[0…
conv4_6_1_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_6_1_conv[0…
conv4_6_1_relu (Activation)	(None, 14, 14, 256)	0	conv4_6_1_bn[0][…
conv4_6_2_conv (Conv2D)	(None, 14, 14, 256)	590,080	conv4_6_1_relu[0…
conv4_6_2_bn (BatchNormalizatio…	(None, 14, 14, 256)	1,024	conv4_6_2_conv[0…
activation_62 (Activation)	(None, 14, 14, 256)	0	conv4_6_2_bn[0][…
conv4_6_3_conv (Conv2D)	(None, 14, 14, 1024)	263,168	activation_62[0]…
conv4_6_3_bn (BatchNormalizatio…	(None, 14, 14, 1024)	4,096	conv4_6_3_conv[0…
add_76 (Add)	(None, 14, 14, 1024)	0	conv4_6_3_bn[0][… conv4_5_3_relu[0…
conv4_6_3_relu (Activation)	(None, 14, 14, 1024)	0	add_76[0][0]
conv5_1_1_conv (Conv2D)	(None, 7, 7, 512)	524,800	conv4_6_3_relu[0…
conv5_1_1_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_1_1_conv[0…
conv5_1_1_relu (Activation)	(None, 7, 7, 512)	0	conv5_1_1_bn[0][…
conv5_1_2_conv (Conv2D)	(None, 7, 7, 512)	2,359,808	conv5_1_1_relu[0…

conv5_1_2_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_1_2_conv[0…
conv5_1_2_relu (Activation)	(None, 7, 7, 512)	0	conv5_1_2_bn[0][…
conv5_1_3_conv (Conv2D)	(None, 7, 7, 2048)	1,050,624	conv5_1_2_relu[0…
conv5_1_short_conv (Conv2D)	(None, 7, 7, 2048)	2,099,200	conv4_6_3_relu[0…
conv5_1_3_bn (BatchNormalizatio…	(None, 7, 7, 2048)	8,192	conv5_1_3_conv[0…
conv5_1_short_bn (BatchNormalizatio…	(None, 7, 7, 2048)	8,192	conv5_1_short_co…
add_77 (Add)	(None, 7, 7, 2048)	0	conv5_1_3_bn[0][… conv5_1_short_bn…
conv5_1_3_relu (Activation)	(None, 7, 7, 2048)	0	add_77[0][0]
conv5_2_1_conv (Conv2D)	(None, 7, 7, 512)	1,049,088	conv5_1_3_relu[0…
conv5_2_1_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_2_1_conv[0…
conv5_2_1_relu (Activation)	(None, 7, 7, 512)	0	conv5_2_1_bn[0][…
conv5_2_2_conv (Conv2D)	(None, 7, 7, 512)	2,359,808	conv5_2_1_relu[0…
conv5_2_2_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_2_2_conv[0…
activation_63 (Activation)	(None, 7, 7, 512)	0	conv5_2_2_bn[0][…
conv5_2_3_conv (Conv2D)	(None, 7, 7, 2048)	1,050,624	activation_63[0]…
conv5_2_3_bn (BatchNormalizatio…	(None, 7, 7, 2048)	8,192	conv5_2_3_conv[0…

add_78 (Add)	(None, 7, 7, 2048)	0	conv5_2_3_bn[0][… conv5_1_3_relu[0…
conv5_2_3_relu (Activation)	(None, 7, 7, 2048)	0	add_78[0][0]
conv5_3_1_conv (Conv2D)	(None, 7, 7, 512)	1,049,088	conv5_2_3_relu[0…
conv5_3_1_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_3_1_conv[0…
conv5_3_1_relu (Activation)	(None, 7, 7, 512)	0	conv5_3_1_bn[0][…
conv5_3_2_conv (Conv2D)	(None, 7, 7, 512)	2,359,808	conv5_3_1_relu[0…
conv5_3_2_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_3_2_conv[0…
activation_64 (Activation)	(None, 7, 7, 512)	0	conv5_3_2_bn[0][…
conv5_3_3_conv (Conv2D)	(None, 7, 7, 2048)	1,050,624	activation_64[0]…
conv5_3_3_bn (BatchNormalizatio…	(None, 7, 7, 2048)	8,192	conv5_3_3_conv[0…
add_79 (Add)	(None, 7, 7, 2048)	0	conv5_3_3_bn[0][… conv5_2_3_relu[0…
conv5_3_3_relu (Activation)	(None, 7, 7, 2048)	0	add_79[0][0]
avg_pool (AveragePooling2D)	(None, 3, 3, 2048)	0	conv5_3_3_relu[0…
flatten_4 (Flatten)	(None, 18432)	0	avg_pool[0][0]
fc_2 (Dense)	(None, 2)	36,866	flatten_4[0][0]

Total params: 23,624,578 (90.12 MB)

Trainable params: 23,571,458 (89.92 MB)

```
Non-trainable params: 53,120 (207.50 KB)
```

[95]:
```
model.compile(
    optimizer='adam', # optimizer
    loss='categorical_crossentropy', # loss function to optimize
    metrics=['accuracy'] # metrics to monitor
)
```

[96]:
```
print("Train images shape:", train_images.shape)
print("Train labels shape:", train_labels.shape)
print("Validation images shape:", val_images.shape)
print("Validation labels shape:", val_labels.shape)
```

```
Train images shape: (9200, 224, 224, 3)
Train labels shape: (9200, 2)
Validation images shape: (827, 224, 224, 3)
Validation labels shape: (827, 2)
```

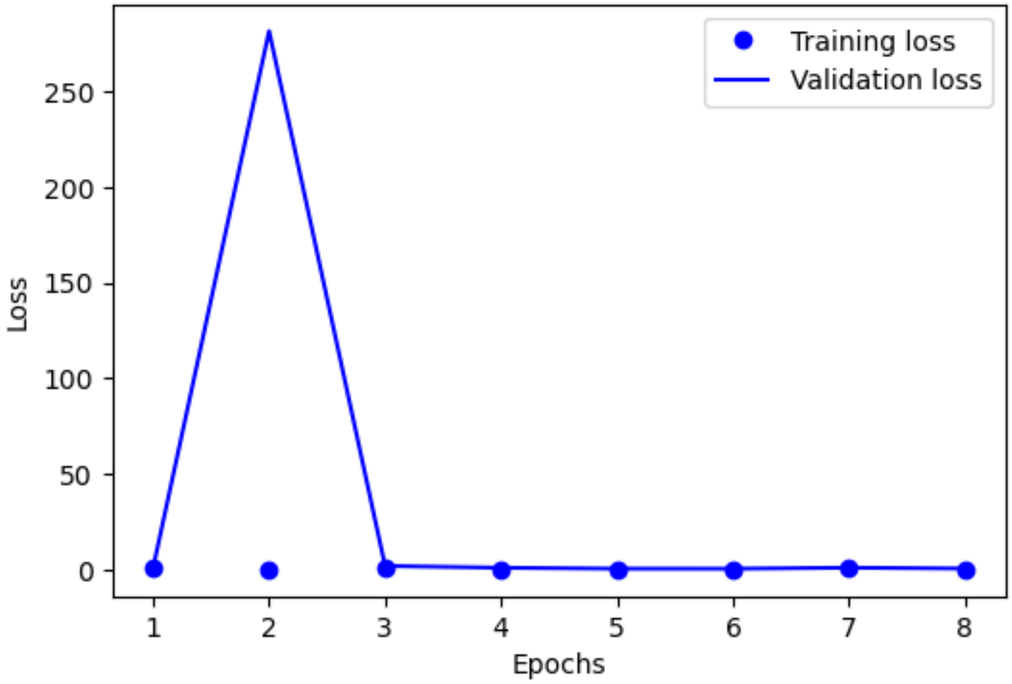
[97]:
```
callbacks_list = [
    tf.keras.callbacks.EarlyStopping(
        monitor="val_accuracy",
        patience=2,
    ),
    tf.keras.callbacks.ModelCheckpoint(
        filepath="checkpoint_path.keras",
        monitor="val_accuracy",
        save_best_only=True,
    )
]



history = model.fit(train_images,  train_labels, epochs=100,
          batch_size=64, validation_data=(val_images, val_labels),
          callbacks=callbacks_list)
```

```
Epoch 1/100
144/144             816s 6s/step -
accuracy: 0.6599 - loss: 1.9708 - val_accuracy: 0.6179 - val_loss: 0.7685
Epoch 2/100
144/144             809s 6s/step -
accuracy: 0.9193 - loss: 0.2482 - val_accuracy: 0.6264 - val_loss: 281.7943
Epoch 3/100
144/144             805s 6s/step -
accuracy: 0.9233 - loss: 0.6854 - val_accuracy: 0.6239 - val_loss: 1.7360
```

```
Epoch 4/100
144/144              955s 7s/step -
accuracy: 0.9640 - loss: 0.1195 - val_accuracy: 0.8017 - val_loss: 0.7777
Epoch 5/100
144/144              816s 6s/step -
accuracy: 0.9816 - loss: 0.0519 - val_accuracy: 0.9238 - val_loss: 0.3030
Epoch 6/100
144/144              818s 6s/step -
accuracy: 0.9954 - loss: 0.0202 - val_accuracy: 0.9262 - val_loss: 0.2753
Epoch 7/100
144/144              832s 6s/step -
accuracy: 0.8946 - loss: 0.4906 - val_accuracy: 0.7848 - val_loss: 0.8124
Epoch 8/100
144/144              807s 6s/step -
accuracy: 0.9882 - loss: 0.0440 - val_accuracy: 0.9214 - val_loss: 0.3822
```

[98]:

```python
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
# Set up the range of epochs
epochs = range(1, len(loss_values) + 1)
# Create a plot of the training and validation loss
plt.figure(figsize=(6, 4))
plt.plot(epochs, loss_values, 'bo', label='Training loss') # 'bo' for blue dot
plt.plot(epochs, val_loss_values, 'b', label='Validation loss') # 'b' for solid
 ↪blue line plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



[101]:
```
model_best = keras.models.load_model("checkpoint_path.keras")
test_loss, test_acc = model_best.evaluate(test_images, test_labels)
print("Test accuracy:", test_acc)
```

```
16/16              10s 580ms/step -
accuracy: 0.9199 - loss: 0.2041
Test accuracy: 0.9226069450378418
```