ps2

April 6, 2024

[ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import seaborn as sns
import tensorflow as tf
from tensorflow.keras import layers, Model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input
from sklearn.model_selection import train_test_split
```

[ ]:

```
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).
```

[ ]:

```
# Load text data from CSV
df = pd.read_csv('/content/drive/MyDrive/hateful_memes/hateful_memes_original.
 ↪csv')

# Adjust image directory path for Colab
image_dir = '/content/drive/MyDrive/hateful_memes/img'
```

[ ]:

```
df.head()
```

[ ]:

```
                                                 text            img  label
0           a school bus that was engulfed in flames  img/32674.png      0
1  when you ask your dad who is a retired drill s…  img/10246.png      1
2    how i see kim burell everytime she grabs a mic!  img/14570.png      1
3                            doing o's with the smoke  img/05316.png      0
4                  im gettin white girl wasted tonight  img/20936.png      0
```

[ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   text    10000 non-null  object
 1   img     10000 non-null  object
 2   label   10000 non-null  int64
dtypes: int64(1), object(2)
memory usage: 234.5+ KB
```

[ ]:
```
df.drop_duplicates(inplace=True)
```

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   text    10000 non-null  object
 1   img     10000 non-null  object
 2   label   10000 non-null  int64
dtypes: int64(1), object(2)
memory usage: 234.5+ KB
```

[ ]:
```
# Preprocess text data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(df['text'])
sequences = tokenizer.texts_to_sequences(df['text'])
word_index = tokenizer.word_index
max_sequence_length = max([len(seq) for seq in sequences])
text_data = pad_sequences(sequences, maxlen=max_sequence_length)
```

[ ]:
```
text_data
```

[ ]:
```
array([[   0,    0,    0, …, 5586,   10, 5587],
       [   0,    0,    0, …, 3091, 1316,  287],
       [   0,    0,    0, …, 3092,    2, 3880],
       …,
       [   0,    0,    0, …,    0,  296,  727],
       [   0,    0,    0, …,   80,  363,  333],
       [   0,    0,    0, …, 3874,    9,   63]], dtype=int32)
```

[ ]:
```
image_data = []
valid_indices = []  # Keep track of valid indices for images
for idx, image_file in enumerate(os.listdir(image_dir)):
    if image_file.endswith('.png'):
        img_path = os.path.join(image_dir, image_file)
        img = tf.keras.preprocessing.image.load_img(img_path, target_size=(224,␣
↪224))
        img_array = tf.keras.preprocessing.image.img_to_array(img)
        img_array = preprocess_input(img_array)
        image_data.append(img_array)
        valid_indices.append(idx)
image_data = np.array(image_data)
```

[ ]:
```
valid_indices
```

[ ]:
```
[0,
 1,
 2,
 3,
 4,
 5,
 6,
 7,
 8,
 9,
 10,
 11,
 12,
 13,
 14,
 15,
 16,
 17,
 18,
 19,
 20,
 21,
 22,
 23,
 24,
 25,
 26,
 27,
 28,
 29,
 30,
 31,
 32,
 33,
 34,
 35,
36,
37,
38,
39,
40,
41,
42,
43,
44,
45,
46,
47,
48,
49,
50,
51,
52,
53,
54,
55,
56,
57,
58,
59,
60,
61,
62,
63,
64,
65,
66,
67,
68,
69,
70,
71,
72,
73,
74,
75,
76,
77,
78,
79,
80,
81,
82,
83,
84,
85,
86,
87,
88,
89,
90,
91,
92,
93,
94,
95,
96,
97,
98,
99,
100,
101,
102,
103,
104,
105,
106,
107,
108,
109,
110,
111,
112,
113,
114,
115,
116,
117,
118,
119,
120,
121,
122,
123,
124,
125,
126,
127,
128,
129,
130,
131,
132,
133,
134,
135,
136,
137,
138,
139,
140,
141,
142,
143,
144,
145,
146,
147,
148,
149,
150,
151,
152,
153,
154,
155,
156,
157,
158,
159,
160,
161,
162,
163,
164,
165,
166,
167,
168,
169,
170,
171,
172,
173,
174,
175,
176,
177,
178,
179,
180,
181,
182,
183,
184,
185,
186,
187,
188,
189,
190,
191,
192,
193,
194,
195,
196,
197,
198,
199,
200,
201,
202,
203,
204,
205,
206,
207,
208,
209,
210,
211,
212,
213,
214,
215,
216,
217,
218,
219,
220,
221,
222,
223,
224,
225,
226,
227,
228,
229,
230,
231,
232,
233,
234,
235,
236,
237,
238,
239,
240,
241,
242,
243,
244,
245,
246,
247,
248,
249,
250,
251,
252,
253,
254,
255,
256,
257,
258,
259,
260,
261,
262,
263,
264,
265,
266,
267,
268,
269,
270,
271,
272,
273,
274,
275,
276,
277,
278,
279,
280,
281,
282,
283,
284,
285,
286,
287,
288,
289,
290,
291,
292,
293,
294,
295,
296,
297,
298,
299,
300,
301,
302,
303,
304,
305,
306,
307,
308,
309,
310,
311,
312,
313,
314,
315,
316,
317,
318,
319,
320,
321,
322,
323,
324,
325,
326,
327,
328,
329,
330,
331,
332,
333,
334,
335,
336,
337,
338,
339,
340,
341,
342,
343,
344,
345,
346,
347,
348,
349,
350,
351,
352,
353,
354,
355,
356,
357,
358,
359,
360,
361,
362,
363,
364,
365,
366,
367,
368,
369,
370,
371,
372,
373,
374,
375,
376,
377,
378,
379,
380,
381,
382,
383,
384,
385,
386,
387,
388,
389,
390,
391,
392,
393,
394,
395,
396,
397,
398,
399,
400,
401,
402,
403,
404,
405,
406,
407,
408,
409,
410,
411,
412,
413,
414,
415,
416,
417,
418,
419,
420,
421,
422,
423,
424,
425,
426,
427,
428,
429,
430,
431,
432,
433,
434,
435,
436,
437,
438,
439,
440,
441,
442,
443,
444,
445,
446,
447,
448,
449,
450,
451,
452,
453,
454,
455,
456,
457,
458,
459,
460,
461,
462,
463,
464,
465,
466,
467,
468,
469,
470,
471,
472,
473,
474,
475,
476,
477,
478,
479,
480,
481,
482,
483,
484,
485,
486,
487,
488,
489,
490,
491,
492,
493,
494,
495,
496,
497,
498,
499,
500,
501,
502,
503,
504,
505,
506,
507,
508,
509,
510,
511,
512,
513,
514,
515,
516,
517,
518,
519,
520,
521,
522,
523,
524,
525,
526,
527,
528,
529,
530,
531,
532,
533,
534,
535,
536,
537,
538,
539,
540,
541,
542,
543,
544,
545,
546,
547,
548,
549,
550,
551,
552,
553,
554,
555,
556,
557,
558,
559,
560,
561,
562,
563,
564,
565,
566,
567,
568,
569,
570,
571,
572,
573,
574,
575,
576,
577,
578,
579,
580,
581,
582,
583,
584,
585,
586,
587,
588,
589,
590,
591,
592,
593,
594,
595,
596,
597,
598,
599,
600,
601,
602,
603,
604,
605,
606,
607,
608,
609,
610,
611,
612,
613,
614,
615,
616,
617,
618,
619,
620,
621,
622,
623,
624,
625,
626,
627,
628,
629,
630,
631,
632,
633,
634,
635,
636,
637,
638,
639,
640,
641,
642,
643,
644,
645,
646,
647,
648,
649,
650,
651,
652,
653,
654,
655,
656,
657,
658,
659,
660,
661,
662,
663,
664,
665,
666,
667,
668,
669,
670,
671,
672,
673,
674,
675,
676,
677,
678,
679,
680,
681,
682,
683,
684,
685,
686,
687,
688,
689,
690,
691,
692,
693,
694,
695,
696,
697,
698,
699,
700,
701,
702,
703,
704,
705,
706,
707,
708,
709,
710,
711,
712,
713,
714,
715,
716,
717,
718,
719,
720,
721,
722,
723,
724,
725,
726,
727,
728,
729,
730,
731,
732,
733,
734,
735,
736,
737,
738,
739,
740,
741,
742,
743,
744,
745,
746,
747,
748,
749,
750,
751,
752,
753,
754,
755,
756,
757,
758,
759,
760,
761,
762,
763,
764,
765,
766,
767,
768,
769,
770,
771,
772,
773,
774,
775,
776,
777,
778,
779,
780,
781,
782,
783,
784,
785,
786,
787,
788,
789,
790,
791,
792,
793,
794,
795,
796,
797,
798,
799,
800,
801,
802,
803,
804,
805,
806,
807,
808,
809,
810,
811,
812,
813,
814,
815,
816,
817,
818,
819,
820,
821,
822,
823,
824,
825,
826,
827,
828,
829,
830,
831,
832,
833,
834,
835,
836,
837,
838,
839,
840,
841,
842,
843,
844,
845,
846,
847,
848,
849,
850,
851,
852,
853,
854,
855,
856,
857,
858,
859,
860,
861,
862,
863,
864,
865,
866,
867,
868,
869,
870,
871,
872,
873,
874,
875,
876,
877,
878,
879,
880,
881,
882,
883,
884,
885,
886,
887,
888,
889,
890,
891,
892,
893,
894,
895,
896,
897,
898,
899,
900,
901,
902,
903,
904,
905,
906,
907,
908,
909,
910,
911,
912,
913,
914,
915,
916,
917,
918,
919,
920,
921,
922,
923,
924,
925,
926,
927,
928,
929,
930,
931,
932,
933,
934,
935,
936,
937,
938,
939,
940,
941,
942,
943,
944,
945,
946,
947,
948,
949,
950,
951,
952,
953,
954,
955,
956,
957,
958,
959,
960,
961,
962,
963,
964,
965,
966,
967,
968,
969,
970,
971,
972,
973,
974,
975,
976,
977,
978,
979,
980,
981,
982,
983,
984,
985,
986,
987,
988,
989,
990,
991,
992,
993,
994,
995,
996,
997,
998,
999,
…]
```

[ ]:

```
text_data_filtered = text_data[valid_indices]
y_filtered = df['label'].iloc[valid_indices]
```

[ ]:

```
X_text = text_data_filtered
X_image = image_data
y = y_filtered.values
```

[ ]:

```
X_text_train, X_text_test, X_image_train, X_image_test, y_train, y_test =␣
 ↪train_test_split(
    X_text, X_image, y, test_size=0.2, random_state=42)
X_text_train, X_text_val, X_image_train, X_image_val, y_train, y_val =␣
 ↪train_test_split(
    X_text_train, X_image_train, y_train, test_size=0.1, random_state=42)
```

[ ]:

```
text_input = layers.Input(shape=(max_sequence_length,), name='text_input')
text_embedding = layers.Embedding(len(word_index) + 1, 100,␣
 ↪input_length=max_sequence_length)(text_input)
text_flatten = layers.Flatten()(text_embedding)

image_input = layers.Input(shape=(224, 224, 3), name='image_input')
```

```
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224,␣
 ↪224, 3))
image_output = base_model(image_input)
image_flatten = layers.Flatten()(image_output)

concatenated = layers.concatenate([text_flatten, image_flatten])
output = layers.Dense(1, activation='sigmoid')(concatenated)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 1s 0us/step
```

[ ]:
```
model = Model(inputs=[text_input, image_input], outputs=output)
```

[ ]:
```
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
```

[ ]:
```
model.fit([X_text_train, X_image_train], y_train, validation_data=([X_text_val,␣
 ↪X_image_val], y_val), epochs=10, batch_size=32)
```

```
Epoch 1/10
69/69 [==============================] - 1865s 27s/step - loss: 2.3896 -
accuracy: 0.5852 - val_loss: 5406.7153 - val_accuracy: 0.6122
Epoch 2/10
69/69 [==============================] - 1834s 27s/step - loss: 1.0522 -
accuracy: 0.6280 - val_loss: 22.0468 - val_accuracy: 0.6082
Epoch 3/10
69/69 [==============================] - 1784s 26s/step - loss: 0.6331 -
accuracy: 0.7391 - val_loss: 2.1582 - val_accuracy: 0.5265
Epoch 4/10
69/69 [==============================] - 1715s 25s/step - loss: 0.5437 -
accuracy: 0.8506 - val_loss: 0.8670 - val_accuracy: 0.5633
Epoch 5/10
69/69 [==============================] - 1694s 25s/step - loss: 0.3645 -
accuracy: 0.9030 - val_loss: 2.1191 - val_accuracy: 0.4816
Epoch 6/10
69/69 [==============================] - 1695s 25s/step - loss: 0.2139 -
accuracy: 0.9317 - val_loss: 0.8870 - val_accuracy: 0.5959
Epoch 7/10
69/69 [==============================] - 1674s 24s/step - loss: 0.1662 -
accuracy: 0.9545 - val_loss: 1.1787 - val_accuracy: 0.4980
Epoch 8/10
69/69 [==============================] - 1712s 25s/step - loss: 0.1234 -
accuracy: 0.9604 - val_loss: 1.0449 - val_accuracy: 0.5918
Epoch 9/10
69/69 [==============================] - 1686s 24s/step - loss: 0.0557 -
accuracy: 0.9891 - val_loss: 1.0527 - val_accuracy: 0.5878
```

```
Epoch 10/10
69/69 [==============================] - 1934s 28s/step - loss: 0.0297 -
accuracy: 0.9977 - val_loss: 1.1396 - val_accuracy: 0.5959
```

```
[ ]: <keras.src.callbacks.History at 0x7c016cb8e980>
```

```
[ ]: loss, accuracy = model.evaluate([X_text_test, X_image_test], y_test)
     print(f'Test Loss: {loss}, Test Accuracy: {accuracy}')
```

```
20/20 [==============================] - 128s 6s/step - loss: 0.9612 - accuracy:
0.6498
Test Loss: 0.9611522555351257, Test Accuracy: 0.649754524230957
```

```
[ ]: y_pred_prob = model.predict([X_text_test, X_image_test])
```

```
20/20 [==============================] - 129s 6s/step
```

```
[ ]: y_pred = (y_pred_prob > 0.5).astype(int)
```

```
[ ]: y_pred
```

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

[ ]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, roc_auc_score, confusion_matrix
```

[ ]:

```
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred_prob)

print(f'Precision: {precision}, Recall: {recall}, F1-score: {f1}, AUC: {auc}')
```

```
Precision: 0.4973821989528796, Recall: 0.4460093896713615, F1-score:
0.4702970297029703, AUC: 0.6200721919456437
```

[ ]:

```
conf_matrix = confusion_matrix(y_test, y_pred)
```

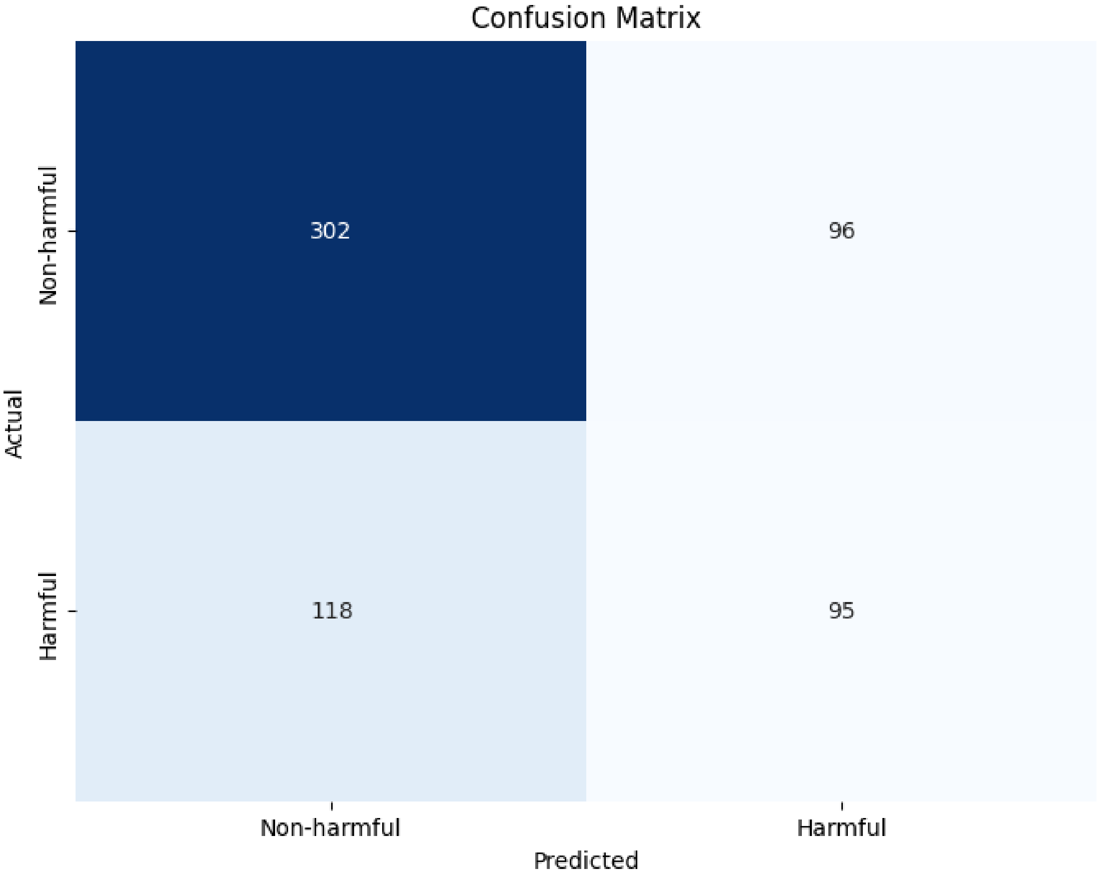
[ ]:

```
print(conf_matrix)
```

```
[[302  96]
 [118  95]]
```

[ ]:

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g', cbar=False,
            xticklabels=['Non-harmful', 'Harmful'], yticklabels=['Non-harmful',␣
 ↪'Harmful'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```


Confusion Matrix
Actual
Non-harmful
Harmful
Non-harmful
Harmful
Predicted