Lab_4

October 26, 2022

[1]:
```
# Set up the dataset
img_dir = "https://vizwiz.cs.colorado.edu//VizWiz_visualization_img/"
train_split = "train"
val_split = "val"
test_split = "test"
train_annotation_file = "https://vizwiz.cs.colorado.edu/VizWiz_final/vqa_data/
 ↪Annotations/%s.json" %train_split
val_annotation_file = "https://vizwiz.cs.colorado.edu/VizWiz_final/vqa_data/
 ↪Annotations/%s.json" %val_split
test_annotation_file = "https://vizwiz.cs.colorado.edu/VizWiz_final/vqa_data/
 ↪Annotations/%s.json" %test_split
print(train_annotation_file)
print(val_annotation_file)
print(test_annotation_file)
```

```
https://vizwiz.cs.colorado.edu/VizWiz_final/vqa_data/Annotations/train.json
https://vizwiz.cs.colorado.edu/VizWiz_final/vqa_data/Annotations/val.json
https://vizwiz.cs.colorado.edu/VizWiz_final/vqa_data/Annotations/test.json
```

[2]:
```
#Read the file to extract each dataset example with label

import requests
import numpy as np
#extract training dataset with label
train_split_data = requests.get(train_annotation_file, allow_redirects=True)
train_data = train_split_data.json()
#extract validation dataset with label
val_split_data = requests.get(val_annotation_file, allow_redirects=True)
val_data = val_split_data.json()
#extract test dataset with label
test_split_data = requests.get(test_annotation_file, allow_redirects=True)
test_data = test_split_data.json()
```

[3]:
```
#randomly select samples
import random
np.random.seed(123)
train_ind = np.random.choice(len(train_data), 1500, replace=False)
```

```
print("Selected", train_ind.shape[0], "training samples out of",␣
 ↪len(train_data))
val_ind = np.random.choice(len(val_data), 600, replace=False)
print("Selected", val_ind.shape[0], "validation samples out of", len(val_data))
test_ind = np.random.choice(len(test_data), 1000, replace=False)
print("Selected", test_ind.shape[0], "test samples out of", len(test_data))
large_train_ind = np.random.choice(len(train_data), 4000, replace=False)
print("Selected", large_train_ind.shape[0], "training samples out of",␣
 ↪len(train_data), "for the large set")
```

```
Selected 1500 training samples out of 20523
Selected 600 validation samples out of 4319
Selected 1000 test samples out of 8000
Selected 4000 training samples out of 20523 for the large set
```

[4]:

```
#training lists of desired features
train_question_list = []
train_label_list = []
train_image_list = []
for i in range(train_ind.shape[0]):
    vq = train_data[train_ind[i]]
    question = vq['question']
    train_question_list.append(question)
    label = vq['answerable']
    train_label_list.append(label)
    image = vq['image']
    train_image_list.append(image)
#validation lists of desired features
val_question_list = []
val_label_list = []
val_image_list = []
for i in range(val_ind.shape[0]):
    vq = val_data[val_ind[i]]
    question = vq['question']
    val_question_list.append(question)
    label = vq['answerable']
    val_label_list.append(label)
    image = vq['image']
    val_image_list.append(image)
#test lists of desired features
test_question_list = []
test_image_list = []
for i in range(test_ind.shape[0]):
    vq = test_data[test_ind[i]]
    question = vq['question']
    test_question_list.append(question)
    image = vq['image']
```

```
    test_image_list.append(image)
#training lists of desired features for large set
large_train_question_list = []
large_train_label_list = []
large_train_image_list = []
for i in range(large_train_ind.shape[0]):
    vq = train_data[large_train_ind[i]]
    question = vq['question']
    large_train_question_list.append(question)
    label = vq['answerable']
    large_train_label_list.append(label)
    image = vq['image']
    large_train_image_list.append(image)
```

[5]:
```
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from PIL import Image
from urllib import request
from io import BytesIO
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
vgg_model = VGG16(weights='imagenet', include_top = False,␣
 ↪input_shape=(224,224,3))
vgg_model.summary()
```

```
2022-10-26 10:18:44.003328: I tensorflow/core/platform/cpu_feature_guard.cc:151]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584
```

```
 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

_________________________________________________________________
```

[6]:

```
import pandas as pd
url = img_dir + train_image_list[0]
res = request.urlopen(url).read()
img = Image.open(BytesIO(res)).resize((224,224))
img_data = image.img_to_array(img)
img_data = np.expand_dims(img_data, axis=0)
img_data = preprocess_input(img_data)
img_data = np.array(img_data)
print(img_data.shape)
vgg16_feature = vgg_model.predict(img_data)
```

```
(1, 224, 224, 3)
```

[7]:

```
def extract_image_features(image_url):
```

```
    res = request.urlopen(url).read()
    img = Image.open(BytesIO(res)).resize((224,224))
    img_data = image.img_to_array(img)
    img_data = np.expand_dims(img_data, axis=0)
    img_data = preprocess_input(img_data)
    img_data = np.array(img_data)
    #print(img_data.shape)
    vgg16_feature = vgg_model.predict(img_data)
    vgg16_feature = np.array(vgg16_feature)
    image_feature_vector = vgg16_feature.ravel()
    #print(image_feature_vector)
    #print(len(image_feature_vector))
    return image_feature_vector
```

[8]:

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
```

[9]:

```
def extract_question_features(question):
    #print(question)
    tokenized_question = tokenizer(question, padding="max_length",␣
 ↪truncation=True)
    #print(tokenized_question['input_ids'])
    return tokenized_question['input_ids']
```

[12]:

```
X_train = [] ## features
y_train = [] ## target labels
train_image_features = []
for i, fname in enumerate(train_image_list):
    print(i)
    url = img_dir + train_image_list[i]
    image_features = extract_image_features(url)
    question = train_question_list[i]
    question_features = extract_question_features(question)
    multimodal_features = np.concatenate((question_features, image_features))
    X_train.append(multimodal_features)
    y_train.append(train_label_list[i])
X_train = np.vstack(X_train)
y_train = np.array(y_train)
```

```
0
1
2
3
4
5
6
7
8
```

```
1497
1498
1499
```

[11]:

```
X_val = [] ## features
y_val = [] ## target labels
train_image_features = []
for i, fname in enumerate(val_image_list):
    print(i)
    url = img_dir + val_image_list[i]
    image_features = extract_image_features(url)
    question = val_question_list[i]
    question_features = extract_question_features(question)
    multimodal_features = np.concatenate((question_features, image_features))
    X_val.append(multimodal_features)
    y_val.append(val_label_list[i])
X_val = np.vstack(X_val)
y_val = np.array(y_val)
```

```
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
```

```
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
```

[17]:

```python
X_test = [] ## features
test_image_features = []
for i, fname in enumerate(test_image_list):
```

```
    print(i)
    url = img_dir + test_image_list[i]
    image_features = extract_image_features(url)
    question = test_question_list[i]
    question_features = extract_question_features(question)
    multimodal_features = np.concatenate((question_features, image_features))
    X_test.append(multimodal_features)
X_test = np.vstack(X_test)
```

```
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
```

```
999
```

[10]:
```python
large_X_train = [] ## features
large_y_train = [] ## target labels
train_image_features = []
for i, fname in enumerate(large_train_image_list):
    print(i)
    url = img_dir + large_train_image_list[i]
    image_features = extract_image_features(url)
    question = large_train_question_list[i]
    question_features = extract_question_features(question)
    multimodal_features = np.concatenate((question_features, image_features))
    large_X_train.append(multimodal_features)
    large_y_train.append(large_train_label_list[i])
large_X_train = np.vstack(large_X_train)
large_y_train = np.array(large_y_train)
```

```
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
```

```
3967
3968
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3989
3990
3991
3992
3993
3994
3995
3996
3997
3998
3999
```

[13]:

```
print(X_train.shape)
print(y_train.shape)
print(X_val.shape)
print(y_val.shape)
```

```
(1500, 25600)
(1500,)
(600, 25600)
(600,)
```

[14]:

```
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Activation, Dense
from tensorflow.keras import activations, optimizers
```

```python
from keras import optimizers
from keras.layers import Embedding
from keras.initializers import Constant
from keras import layers, Input, Model

model_one = Sequential()
model_one.add(Dense(200, activation='sigmoid'))
model_one.add(Dense(2, activation='sigmoid'))
optimizer = keras.optimizers.Adam(learning_rate=0.0005)
model_one.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model_one.fit(X_train, y_train, epochs=20)
model_one.summary()
predicted_values = model_one.predict(X_val)
y_pred = []
for i in range(predicted_values.shape[0]):
    pred = np.argmax(predicted_values[i,:])
    y_pred.append(pred)
y_pred = np.array(y_pred)
from sklearn.metrics import average_precision_score
average_precision = average_precision_score(y_val, y_pred)
print("Average Precision for model 1:", average_precision)
```

```
Epoch 1/20
47/47 [==============================] - 2s 32ms/step - loss: 0.6098 - accuracy:
0.7293
Epoch 2/20
47/47 [==============================] - 1s 30ms/step - loss: 0.4813 - accuracy:
0.7720
Epoch 3/20
47/47 [==============================] - 2s 38ms/step - loss: 0.4389 - accuracy:
0.7867
Epoch 4/20
47/47 [==============================] - 2s 39ms/step - loss: 0.4146 - accuracy:
0.7920
Epoch 5/20
47/47 [==============================] - 2s 34ms/step - loss: 0.3939 - accuracy:
0.8107
Epoch 6/20
47/47 [==============================] - 2s 32ms/step - loss: 0.3675 - accuracy:
0.8313
Epoch 7/20
47/47 [==============================] - 1s 30ms/step - loss: 0.3430 - accuracy:
0.8440
Epoch 8/20
47/47 [==============================] - 1s 30ms/step - loss: 0.3194 - accuracy:
0.8520
```

```
Epoch 9/20
47/47 [==============================] - 2s 33ms/step - loss: 0.3113 - accuracy:
0.8600
Epoch 10/20
47/47 [==============================] - 1s 30ms/step - loss: 0.2876 - accuracy:
0.8820
Epoch 11/20
47/47 [==============================] - 1s 30ms/step - loss: 0.2781 - accuracy:
0.8880
Epoch 12/20
47/47 [==============================] - 1s 31ms/step - loss: 0.2601 - accuracy:
0.8973
Epoch 13/20
47/47 [==============================] - 2s 32ms/step - loss: 0.2522 - accuracy:
0.8973
Epoch 14/20
47/47 [==============================] - 2s 33ms/step - loss: 0.2313 - accuracy:
0.9180
Epoch 15/20
47/47 [==============================] - 2s 32ms/step - loss: 0.2155 - accuracy:
0.9233
Epoch 16/20
47/47 [==============================] - 2s 35ms/step - loss: 0.2055 - accuracy:
0.9347
Epoch 17/20
47/47 [==============================] - 2s 34ms/step - loss: 0.2008 - accuracy:
0.9393
Epoch 18/20
47/47 [==============================] - 2s 33ms/step - loss: 0.1868 - accuracy:
0.9340
Epoch 19/20
47/47 [==============================] - 1s 31ms/step - loss: 0.1799 - accuracy:
0.9353
Epoch 20/20
47/47 [==============================] - 2s 38ms/step - loss: 0.1676 - accuracy:
0.9453
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 200)               5120200

 dense_1 (Dense)             (None, 2)                 402

=================================================================
Total params: 5,120,602
Trainable params: 5,120,602
Non-trainable params: 0
```

```
-----------------------------------------------------------------
Average Precision for model 1: 0.7362696385161593
```

[15]:

```
model_two = Sequential()
model_two.add(Dense(200, activation='relu'))
model_two.add(Dense(2, activation='softmax'))
optimizer = keras.optimizers.Adam(learning_rate=0.0001)
model_two.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model_two.fit(X_train, y_train, epochs=20)
model_two.summary()
predicted_values = model_two.predict(X_val)
y_pred = []
for i in range(predicted_values.shape[0]):
    pred = np.argmax(predicted_values[i,:])
    y_pred.append(pred)
y_pred = np.array(y_pred)
average_precision = average_precision_score(y_val, y_pred)
print("Average Precision for model 2:", average_precision)
```

```
Epoch 1/20
47/47 [==============================] - 2s 31ms/step - loss: 13.9571 -
accuracy: 0.6620
Epoch 2/20
47/47 [==============================] - 1s 30ms/step - loss: 2.6413 - accuracy:
0.8873
Epoch 3/20
47/47 [==============================] - 2s 34ms/step - loss: 0.6967 - accuracy:
0.9467
Epoch 4/20
47/47 [==============================] - 1s 32ms/step - loss: 0.1873 - accuracy:
0.9753
Epoch 5/20
47/47 [==============================] - 1s 28ms/step - loss: 0.0588 - accuracy:
0.9920
Epoch 6/20
47/47 [==============================] - 1s 29ms/step - loss: 0.0303 - accuracy:
0.9947
Epoch 7/20
47/47 [==============================] - 1s 30ms/step - loss: 0.0075 - accuracy:
0.9967
Epoch 8/20
47/47 [==============================] - 1s 30ms/step - loss: 0.0395 - accuracy:
0.9947
Epoch 9/20
47/47 [==============================] - 1s 30ms/step - loss: 0.1195 - accuracy:
0.9860
```

```
Epoch 10/20
47/47 [==============================] - 1s 30ms/step - loss: 0.1335 - accuracy:
0.9867
Epoch 11/20
47/47 [==============================] - 1s 29ms/step - loss: 0.0723 - accuracy:
0.9873
Epoch 12/20
47/47 [==============================] - 1s 29ms/step - loss: 0.1046 - accuracy:
0.9880
Epoch 13/20
47/47 [==============================] - 1s 30ms/step - loss: 0.2310 - accuracy:
0.9853
Epoch 14/20
47/47 [==============================] - 2s 37ms/step - loss: 0.2056 - accuracy:
0.9753
Epoch 15/20
47/47 [==============================] - 2s 35ms/step - loss: 0.2077 - accuracy:
0.9740
Epoch 16/20
47/47 [==============================] - 2s 50ms/step - loss: 0.1410 - accuracy:
0.9867
Epoch 17/20
47/47 [==============================] - 2s 37ms/step - loss: 0.0583 - accuracy:
0.9920
Epoch 18/20
47/47 [==============================] - 2s 34ms/step - loss: 0.1010 - accuracy:
0.9880
Epoch 19/20
47/47 [==============================] - 1s 31ms/step - loss: 0.0872 - accuracy:
0.9887
Epoch 20/20
47/47 [==============================] - 2s 32ms/step - loss: 0.3084 - accuracy:
0.9853
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 200)               5120200

 dense_3 (Dense)             (None, 2)                 402

=================================================================
Total params: 5,120,602
Trainable params: 5,120,602
Non-trainable params: 0
_________________________________________________________________
Average Precision for model 2: 0.7290812147955006
```

[16]:
```
from tensorflow.keras.layers import Dropout


model_three = Sequential()
model_three.add(Dense(1000, activation='relu'))
model_three.add(Dropout(0.1))
model_three.add(Dense(200, activation='sigmoid'))
model_three.add(Dense(2, activation='softmax'))
optimizer = keras.optimizers.Adam(learning_rate=0.0001)
model_three.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model_three.fit(X_train, y_train, epochs=20)
model_three.summary()
predicted_values = model_three.predict(X_val)
y_pred = []
for i in range(predicted_values.shape[0]):
    pred = np.argmax(predicted_values[i,:])
    y_pred.append(pred)
y_pred = np.array(y_pred)
average_precision = average_precision_score(y_val, y_pred)
print("Average Precision for model 3:", average_precision)
```

```
Epoch 1/20
47/47 [==============================] - 8s 149ms/step - loss: 0.5920 -
accuracy: 0.7080
Epoch 2/20
47/47 [==============================] - 6s 138ms/step - loss: 0.4825 -
accuracy: 0.7527
Epoch 3/20
47/47 [==============================] - 7s 144ms/step - loss: 0.4295 -
accuracy: 0.7847
Epoch 4/20
47/47 [==============================] - 8s 180ms/step - loss: 0.3865 -
accuracy: 0.8133
Epoch 5/20
47/47 [==============================] - 7s 149ms/step - loss: 0.3390 -
accuracy: 0.8507
Epoch 6/20
47/47 [==============================] - 7s 149ms/step - loss: 0.2979 -
accuracy: 0.8820
Epoch 7/20
47/47 [==============================] - 7s 156ms/step - loss: 0.2674 -
accuracy: 0.9040
Epoch 8/20
47/47 [==============================] - 7s 148ms/step - loss: 0.2485 -
accuracy: 0.9040
Epoch 9/20
```

```
47/47 [==============================] - 7s 157ms/step - loss: 0.2261 -
accuracy: 0.9307
Epoch 10/20
47/47 [==============================] - 7s 152ms/step - loss: 0.2069 -
accuracy: 0.9307
Epoch 11/20
47/47 [==============================] - 7s 147ms/step - loss: 0.1996 -
accuracy: 0.9340
Epoch 12/20
47/47 [==============================] - 7s 147ms/step - loss: 0.1773 -
accuracy: 0.9460
Epoch 13/20
47/47 [==============================] - 7s 139ms/step - loss: 0.1609 -
accuracy: 0.9627
Epoch 14/20
47/47 [==============================] - 7s 150ms/step - loss: 0.1537 -
accuracy: 0.9507
Epoch 15/20
47/47 [==============================] - 6s 136ms/step - loss: 0.1462 -
accuracy: 0.9593
Epoch 16/20
47/47 [==============================] - 7s 143ms/step - loss: 0.1274 -
accuracy: 0.9720
Epoch 17/20
47/47 [==============================] - 7s 139ms/step - loss: 0.1234 -
accuracy: 0.9720
Epoch 18/20
47/47 [==============================] - 7s 138ms/step - loss: 0.1154 -
accuracy: 0.9713
Epoch 19/20
47/47 [==============================] - 7s 142ms/step - loss: 0.1099 -
accuracy: 0.9740
Epoch 20/20
47/47 [==============================] - 6s 137ms/step - loss: 0.0992 -
accuracy: 0.9767
Model: "sequential_2"
-----------------------------------------------------------------
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 1000)              25601000

 dropout (Dropout)           (None, 1000)              0

 dense_5 (Dense)             (None, 200)               200200

 dense_6 (Dense)             (None, 2)                 402

=================================================================
```

```
Total params: 25,801,602
Trainable params: 25,801,602
Non-trainable params: 0
-----------------------------------------------------------------
Average Precision for model 3: 0.7295143526125121
```

[34]:

```
model_test = Sequential()
model_test.add(Dense(200, activation='sigmoid'))
model_test.add(Dense(2, activation='sigmoid'))
optimizer = keras.optimizers.Adam(learning_rate=0.0005)
model_test.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model_test.fit(large_X_train, large_y_train, epochs=20,␣
 ↪validation_data=(X_val, y_val))
model_test.summary()
predicted_values = model_test.predict(X_test)
y_pred = []
for i in range(predicted_values.shape[0]):
    pred = np.argmax(predicted_values[i,:])
    y_pred.append(pred)
y_pred = np.array(y_pred)

from sklearn.metrics import average_precision_score
```

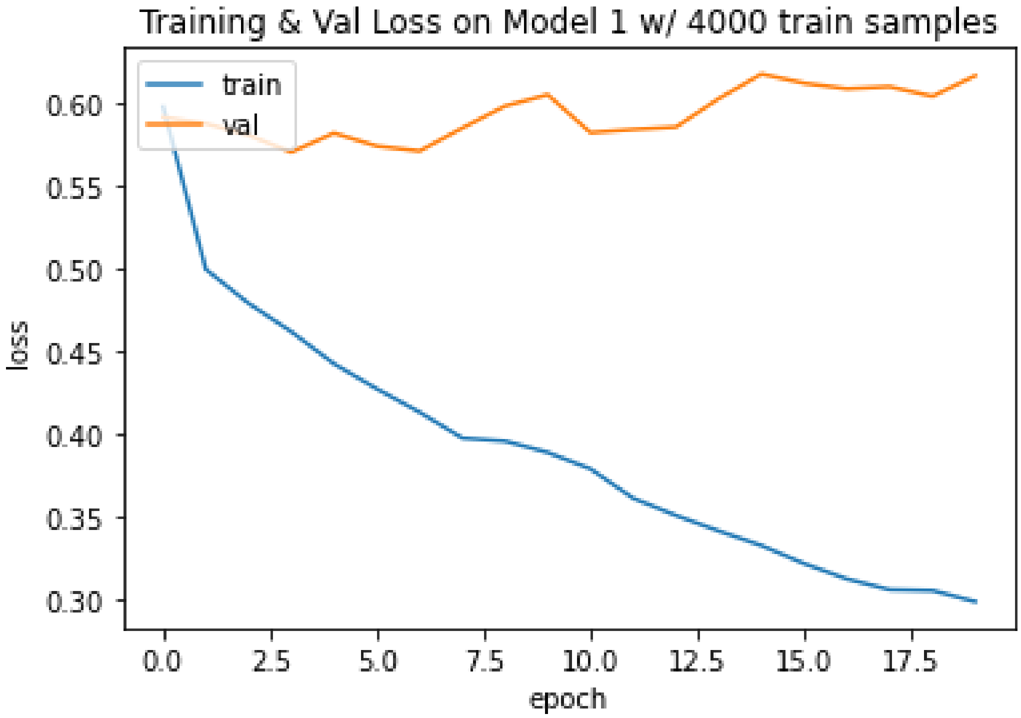
```
Epoch 1/20
125/125 [==============================] - 5s 38ms/step - loss: 0.5978 -
accuracy: 0.7235 - val_loss: 0.5915 - val_accuracy: 0.7017
Epoch 2/20
125/125 [==============================] - 4s 34ms/step - loss: 0.4996 -
accuracy: 0.7570 - val_loss: 0.5876 - val_accuracy: 0.6833
Epoch 3/20
125/125 [==============================] - 5s 39ms/step - loss: 0.4791 -
accuracy: 0.7648 - val_loss: 0.5806 - val_accuracy: 0.6933
Epoch 4/20
125/125 [==============================] - 4s 35ms/step - loss: 0.4620 -
accuracy: 0.7788 - val_loss: 0.5704 - val_accuracy: 0.6967
Epoch 5/20
125/125 [==============================] - 4s 35ms/step - loss: 0.4426 -
accuracy: 0.7872 - val_loss: 0.5819 - val_accuracy: 0.6917
Epoch 6/20
125/125 [==============================] - 4s 32ms/step - loss: 0.4274 -
accuracy: 0.7977 - val_loss: 0.5741 - val_accuracy: 0.7000
Epoch 7/20
125/125 [==============================] - 4s 30ms/step - loss: 0.4133 -
accuracy: 0.8112 - val_loss: 0.5710 - val_accuracy: 0.7083
Epoch 8/20
125/125 [==============================] - 4s 30ms/step - loss: 0.3976 -
```

```
accuracy: 0.8165 - val_loss: 0.5849 - val_accuracy: 0.7100
Epoch 9/20
125/125 [==============================] - 4s 30ms/step - loss: 0.3958 -
accuracy: 0.8175 - val_loss: 0.5982 - val_accuracy: 0.7067
Epoch 10/20
125/125 [==============================] - 4s 34ms/step - loss: 0.3890 -
accuracy: 0.8207 - val_loss: 0.6051 - val_accuracy: 0.7017
Epoch 11/20
125/125 [==============================] - 4s 31ms/step - loss: 0.3792 -
accuracy: 0.8270 - val_loss: 0.5822 - val_accuracy: 0.7200
Epoch 12/20
125/125 [==============================] - 4s 30ms/step - loss: 0.3614 -
accuracy: 0.8415 - val_loss: 0.5841 - val_accuracy: 0.7217
Epoch 13/20
125/125 [==============================] - 4s 31ms/step - loss: 0.3510 -
accuracy: 0.8455 - val_loss: 0.5855 - val_accuracy: 0.7050
Epoch 14/20
125/125 [==============================] - 4s 32ms/step - loss: 0.3416 -
accuracy: 0.8447 - val_loss: 0.6027 - val_accuracy: 0.7233
Epoch 15/20
125/125 [==============================] - 5s 37ms/step - loss: 0.3328 -
accuracy: 0.8515 - val_loss: 0.6175 - val_accuracy: 0.7083
Epoch 16/20
125/125 [==============================] - 4s 33ms/step - loss: 0.3219 -
accuracy: 0.8540 - val_loss: 0.6120 - val_accuracy: 0.7350
Epoch 17/20
125/125 [==============================] - 4s 33ms/step - loss: 0.3126 -
accuracy: 0.8633 - val_loss: 0.6087 - val_accuracy: 0.7017
Epoch 18/20
125/125 [==============================] - 4s 34ms/step - loss: 0.3061 -
accuracy: 0.8725 - val_loss: 0.6099 - val_accuracy: 0.6933
Epoch 19/20
125/125 [==============================] - 4s 30ms/step - loss: 0.3056 -
accuracy: 0.8658 - val_loss: 0.6041 - val_accuracy: 0.7100
Epoch 20/20
125/125 [==============================] - 4s 31ms/step - loss: 0.2989 -
accuracy: 0.8702 - val_loss: 0.6165 - val_accuracy: 0.7133
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_13 (Dense)            (None, 200)               5120200

 dense_14 (Dense)            (None, 2)                 402

=================================================================
Total params: 5,120,602
Trainable params: 5,120,602
```

```
Non-trainable params: 0
_________________________________________________________________
```

[35]:

```
loss = history.history['loss']
val_loss = history.history['val_loss']
loss = np.array(loss)
val_loss = np.array(val_loss)

import matplotlib.pyplot as plt
plt.plot(loss)
plt.plot(val_loss)
plt.title('Training & Val Loss on Model 1 w/ 4000 train samples')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



[49]:

```
## save results to results.csv
import pandas as pd
df = pd.DataFrame(y_pred)

#.to_csv("results.csv")
df.to_csv("results.csv", header = None, index = None)
```

[ ]:

[ ]:

[ ]: