import kerastuner as kt

import math

import numpy as np

import pandas as pd

import random

import sklearn.metrics

import tensorflow as tf

import os

import glob

from tqdm import tqdm

from matplotlib import pyplot as plt

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from google.cloud import storage

physical\_devices = tf.config.list\_physical\_devices('GPU')

physical\_devices

tf.config.set\_visible\_devices([tf.config.PhysicalDevice(name='/physical\_device:GPU:1', device\_type='GPU')], 'GPU')

tf.config.get\_visible\_devices('GPU')

def decode\_fn\_embedding(example\_proto):

feature\_description = {

"embedding": tf.io.FixedLenFeature([256], dtype=tf.float32),

"labels": tf.io.FixedLenFeature([], dtype=tf.int64),

}

example = tf.io.parse\_single\_example(

example\_proto,

feature\_description

)

return example

def preprocess\_embedding\_example(example\_dict, positive\_label=1, features\_as\_dict=False):

labels = example\_dict["labels"]

label = tf.math.reduce\_any(labels == positive\_label)

label = tf.cast(label, tf.int32)

embedding = example\_dict["embedding"]

if features\_as\_dict:

features = {"embedding": embedding}

else:

features = embedding

return features, label

input\_root = ...

sens\_prev\_input\_root = ...

use\_sens\_prev\_data = True

has\_validation\_data = True

positive\_label = 1

train\_batch\_size = 256

test\_batch\_size = 256

validation\_batch\_size = 256

do\_resample = False

def class\_func(features, label):

return label

resample\_fn = tf.data.experimental.rejection\_resample(

class\_func, target\_dist = [0.5, 0.5], seed=0

)

train\_glob = f"{input\_root}/train/tfrecord/\*.tfrecord"

train\_files = tf.io.gfile.glob(train\_glob)

if use\_sens\_prev\_data:

train\_sens\_prev\_glob = f"{sens\_prev\_input\_root}/train/tfrecord/\*.tfrecord"

train\_sens\_prev\_files = tf.io.gfile.glob(train\_sens\_prev\_glob)

train\_files = train\_files + train\_sens\_prev\_files

random.shuffle(train\_files)

if not len(train\_files):

raise ValueError(f"Did not find any train files matching {train\_glob}")

test\_glob = f"{input\_root}/test/tfrecord/\*.tfrecord"

test\_files = tf.io.gfile.glob(test\_glob)

if not len(test\_files):

raise ValueError(f"Did not find any eval files matching {test\_glob}")

test\_ds = tf.data.TFRecordDataset(test\_files).map(decode\_fn\_embedding)

test\_ds = test\_ds.map(lambda x: preprocess\_embedding\_example(x, positive\_label=positive\_label)).batch(batch\_size=test\_batch\_size)

if use\_sens\_prev\_data:

test\_sens\_prev\_glob = f"{sens\_prev\_input\_root}/test/tfrecord/\*.tfrecord"

test\_sens\_prev\_files = tf.io.gfile.glob(test\_sens\_prev\_glob)

if not len(test\_sens\_prev\_files):

raise ValueError(f"Did not find any eval files matching {test\_sens\_prev\_glob}")

test\_sens\_prev\_ds = tf.data.TFRecordDataset(test\_sens\_prev\_files).map(decode\_fn\_embedding)

test\_sens\_prev\_ds = test\_sens\_prev\_ds.map(lambda x: preprocess\_embedding\_example(x, positive\_label=positive\_label)).batch(batch\_size=test\_batch\_size)

train\_ds = tf.data.TFRecordDataset(train\_files).map(decode\_fn\_embedding)

train\_ds = train\_ds.map(lambda x: preprocess\_embedding\_example(x, positive\_label=positive\_label))

if do\_resample:

train\_ds = train\_ds.apply(resample\_fn).map(lambda \_,b:(b))

train\_ds = train\_ds.batch(batch\_size=256).shuffle(buffer\_size=10)

train\_ds = train\_ds.repeat()

if has\_validation\_data:

eval\_glob = f"{input\_root}/validation/tfrecord/\*.tfrecord"

eval\_files = tf.io.gfile.glob(eval\_glob)

if use\_sens\_prev\_data:

eval\_sens\_prev\_glob = f"{sens\_prev\_input\_root}/validation/tfrecord/\*.tfrecord"

eval\_sens\_prev\_files = tf.io.gfile.glob(eval\_sens\_prev\_glob)

eval\_files = eval\_files + eval\_sens\_prev\_files

if not len(eval\_files):

raise ValueError(f"Did not find any eval files matching {eval\_glob}")

eval\_ds = tf.data.TFRecordDataset(eval\_files).map(decode\_fn\_embedding)

eval\_ds = eval\_ds.map(lambda x: preprocess\_embedding\_example(x, positive\_label=positive\_label)).batch(batch\_size=validation\_batch\_size)

else:

eval\_ds = tf.data.TFRecordDataset(test\_files).map(decode\_fn\_embedding)

eval\_ds = eval\_ds.map(lambda x: preprocess\_embedding\_example(x, positive\_label=positive\_label)).batch(batch\_size=validation\_batch\_size)

check\_ds = tf.data.TFRecordDataset(train\_files).map(decode\_fn\_embedding)

cnt = 0

pos\_cnt = 0

for example in tqdm(check\_ds):

label = example['labels']

if label == 1:

pos\_cnt += 1

cnt += 1

print(f'{cnt} train entries with {pos\_cnt} positive')

metrics = []

metrics.append(

tf.keras.metrics.PrecisionAtRecall(

recall=0.9, num\_thresholds=200, class\_id=None, name=None, dtype=None

)

)

metrics.append(

tf.keras.metrics.AUC(

num\_thresholds=200,

curve="PR",

)

)

def build\_model(hp):

model = Sequential()

optimizer = tf.keras.optimizers.Adam(

learning\_rate=0.001,

beta\_1=0.9,

beta\_2=0.999,

epsilon=1e-08,

amsgrad=False,

name="Adam",

)

activation=hp.Choice("activation", ["tanh", "gelu"])

kernel\_initializer=hp.Choice("kernel\_initializer", ["he\_uniform", "glorot\_uniform"])

for i in range(hp.Int("num\_layers", 1, 2)):

model.add(tf.keras.layers.BatchNormalization())

units=hp.Int("units", min\_value=128, max\_value=256, step=128)

if i == 0:

model.add(

Dense(

units=units,

activation=activation,

kernel\_initializer=kernel\_initializer,

input\_shape=(None, 256)

)

)

else:

model.add(

Dense(

units=units,

activation=activation,

kernel\_initializer=kernel\_initializer,

)

)

model.add(Dense(1, activation='sigmoid', kernel\_initializer=kernel\_initializer))

model.compile(optimizer=optimizer, loss='binary\_crossentropy', metrics=metrics)

return model

tuner = kt.tuners.BayesianOptimization(

build\_model,

objective=kt.Objective('val\_loss', direction="min"),

max\_trials=30,

directory='tuner\_dir',

project\_name='with\_twitter\_clip')

callbacks = [tf.keras.callbacks.EarlyStopping(

monitor='val\_loss', min\_delta=0, patience=5, verbose=0,

mode='auto', baseline=None, restore\_best\_weights=True

)]

steps\_per\_epoch = 400

tuner.search(train\_ds,

epochs=100,

batch\_size=256,

steps\_per\_epoch=steps\_per\_epoch,

verbose=2,

validation\_data=eval\_ds,

callbacks=callbacks)

tuner.results\_summary()

models = tuner.get\_best\_models(num\_models=2)

best\_model = models[0]

best\_model.build(input\_shape=(None, 256))

best\_model.summary()

tuner.get\_best\_hyperparameters()[0].values

optimizer = tf.keras.optimizers.Adam(

learning\_rate=0.001,

beta\_1=0.9,

beta\_2=0.999,

epsilon=1e-08,

amsgrad=False,

name="Adam",

)

best\_model.compile(optimizer=optimizer, loss='binary\_crossentropy', metrics=metrics)

best\_model.summary()

callbacks = [tf.keras.callbacks.EarlyStopping(

monitor='val\_loss', min\_delta=0, patience=10, verbose=0,

mode='auto', baseline=None, restore\_best\_weights=True

)]

history = best\_model.fit(train\_ds, epochs=100, validation\_data=eval\_ds, steps\_per\_epoch=steps\_per\_epoch, callbacks=callbacks)

model\_name = 'twitter\_hypertuned'

model\_path = f'models/nsfw\_Keras\_with\_CLIP\_{model\_name}'

tf.keras.models.save\_model(best\_model, model\_path)

def copy\_local\_directory\_to\_gcs(local\_path, bucket, gcs\_path):

"""Recursively copy a directory of files to GCS.

local\_path should be a directory and not have a trailing slash.

"""

assert os.path.isdir(local\_path)

for local\_file in glob.glob(local\_path + '/\*\*'):

if not os.path.isfile(local\_file):

dir\_name = os.path.basename(os.path.normpath(local\_file))

copy\_local\_directory\_to\_gcs(local\_file, bucket, f"{gcs\_path}/{dir\_name}")

else:

remote\_path = os.path.join(gcs\_path, local\_file[1 + len(local\_path) :])

blob = bucket.blob(remote\_path)

blob.upload\_from\_filename(local\_file)

client = storage.Client(project=...)

bucket = client.get\_bucket(...)

copy\_local\_directory\_to\_gcs(model\_path, bucket, model\_path)

copy\_local\_directory\_to\_gcs('tuner\_dir', bucket, 'tuner\_dir')

loaded\_model = tf.keras.models.load\_model(model\_path)

print(history.history.keys())

plt.figure(figsize = (20, 5))

plt.subplot(1, 3, 1)

plt.plot(history.history['auc'])

plt.plot(history.history['val\_auc'])

plt.title('model auc')

plt.ylabel('auc')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.subplot(1, 3, 2)

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.subplot(1, 3, 3)

plt.plot(history.history['precision\_at\_recall'])

plt.plot(history.history['val\_precision\_at\_recall'])

plt.title('model precision at 0.9 recall')

plt.ylabel('precision\_at\_recall')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.savefig('history\_with\_twitter\_clip.pdf')

test\_labels = []

test\_preds = []

for batch\_features, batch\_labels in tqdm(test\_ds):

test\_preds.extend(loaded\_model.predict\_proba(batch\_features))

test\_labels.extend(batch\_labels.numpy())

test\_sens\_prev\_labels = []

test\_sens\_prev\_preds = []

for batch\_features, batch\_labels in tqdm(test\_sens\_prev\_ds):

test\_sens\_prev\_preds.extend(loaded\_model.predict\_proba(batch\_features))

test\_sens\_prev\_labels.extend(batch\_labels.numpy())

n\_test\_pos = 0

n\_test\_neg = 0

n\_test = 0

for label in test\_labels:

n\_test +=1

if label == 1:

n\_test\_pos +=1

else:

n\_test\_neg +=1

print(f'n\_test = {n\_test}, n\_pos = {n\_test\_pos}, n\_neg = {n\_test\_neg}')

n\_test\_sens\_prev\_pos = 0

n\_test\_sens\_prev\_neg = 0

n\_test\_sens\_prev = 0

for label in test\_sens\_prev\_labels:

n\_test\_sens\_prev +=1

if label == 1:

n\_test\_sens\_prev\_pos +=1

else:

n\_test\_sens\_prev\_neg +=1

print(f'n\_test\_sens\_prev = {n\_test\_sens\_prev}, n\_pos\_sens\_prev = {n\_test\_sens\_prev\_pos}, n\_neg = {n\_test\_sens\_prev\_neg}')

test\_weights = np.ones(np.asarray(test\_preds).shape)

test\_labels = np.asarray(test\_labels)

test\_preds = np.asarray(test\_preds)

test\_weights = np.asarray(test\_weights)

pr = sklearn.metrics.precision\_recall\_curve(

test\_labels,

test\_preds)

auc = sklearn.metrics.auc(pr[1], pr[0])

plt.plot(pr[1], pr[0])

plt.title("nsfw (MU test set)")

test\_sens\_prev\_weights = np.ones(np.asarray(test\_sens\_prev\_preds).shape)

test\_sens\_prev\_labels = np.asarray(test\_sens\_prev\_labels)

test\_sens\_prev\_preds = np.asarray(test\_sens\_prev\_preds)

test\_sens\_prev\_weights = np.asarray(test\_sens\_prev\_weights)

pr\_sens\_prev = sklearn.metrics.precision\_recall\_curve(

test\_sens\_prev\_labels,

test\_sens\_prev\_preds)

auc\_sens\_prev = sklearn.metrics.auc(pr\_sens\_prev[1], pr\_sens\_prev[0])

plt.plot(pr\_sens\_prev[1], pr\_sens\_prev[0])

plt.title("nsfw (sens prev test set)")

df = pd.DataFrame(

{

"label": test\_labels.squeeze(),

"preds\_keras": np.asarray(test\_preds).flatten(),

})

plt.figure(figsize=(15, 10))

df["preds\_keras"].hist()

plt.title("Keras predictions", size=20)

plt.xlabel('score')

plt.ylabel("freq")

plt.figure(figsize = (20, 5))

plt.subplot(1, 3, 1)

plt.plot(pr[2], pr[0][0:-1])

plt.xlabel("threshold")

plt.ylabel("precision")

plt.subplot(1, 3, 2)

plt.plot(pr[2], pr[1][0:-1])

plt.xlabel("threshold")

plt.ylabel("recall")

plt.title("Keras", size=20)

plt.subplot(1, 3, 3)

plt.plot(pr[1], pr[0])

plt.xlabel("recall")

plt.ylabel("precision")

plt.savefig('with\_twitter\_clip.pdf')

def get\_point\_for\_recall(recall\_value, recall, precision):

idx = np.argmin(np.abs(recall - recall\_value))

return (recall[idx], precision[idx])

def get\_point\_for\_precision(precision\_value, recall, precision):

idx = np.argmin(np.abs(precision - precision\_value))

return (recall[idx], precision[idx])

precision, recall, thresholds = pr

auc\_precision\_recall = sklearn.metrics.auc(recall, precision)

print(auc\_precision\_recall)

plt.figure(figsize=(15, 10))

plt.plot(recall, precision)

plt.xlabel("recall")

plt.ylabel("precision")

ptAt50 = get\_point\_for\_recall(0.5, recall, precision)

print(ptAt50)

plt.plot( [ptAt50[0],ptAt50[0]], [0,ptAt50[1]], 'r')

plt.plot([0, ptAt50[0]], [ptAt50[1], ptAt50[1]], 'r')

ptAt90 = get\_point\_for\_recall(0.9, recall, precision)

print(ptAt90)

plt.plot( [ptAt90[0],ptAt90[0]], [0,ptAt90[1]], 'b')

plt.plot([0, ptAt90[0]], [ptAt90[1], ptAt90[1]], 'b')

ptAt50fmt = "%.4f" % ptAt50[1]

ptAt90fmt = "%.4f" % ptAt90[1]

aucFmt = "%.4f" % auc\_precision\_recall

plt.title(

f"Keras (nsfw MU test)\nAUC={aucFmt}\np={ptAt50fmt} @ r=0.5\np={ptAt90fmt} @ r=0.9\nN\_train={...}} ({...} pos), N\_test={n\_test} ({n\_test\_pos} pos)",

size=20

)

plt.subplots\_adjust(top=0.72)

plt.savefig('recall\_precision\_nsfw\_Keras\_with\_twitter\_CLIP\_MU\_test.pdf')

precision, recall, thresholds = pr\_sens\_prev

auc\_precision\_recall = sklearn.metrics.auc(recall, precision)

print(auc\_precision\_recall)

plt.figure(figsize=(15, 10))

plt.plot(recall, precision)

plt.xlabel("recall")

plt.ylabel("precision")

ptAt50 = get\_point\_for\_recall(0.5, recall, precision)

print(ptAt50)

plt.plot( [ptAt50[0],ptAt50[0]], [0,ptAt50[1]], 'r')

plt.plot([0, ptAt50[0]], [ptAt50[1], ptAt50[1]], 'r')

ptAt90 = get\_point\_for\_recall(0.9, recall, precision)

print(ptAt90)

plt.plot( [ptAt90[0],ptAt90[0]], [0,ptAt90[1]], 'b')

plt.plot([0, ptAt90[0]], [ptAt90[1], ptAt90[1]], 'b')

ptAt50fmt = "%.4f" % ptAt50[1]

ptAt90fmt = "%.4f" % ptAt90[1]

aucFmt = "%.4f" % auc\_precision\_recall

plt.title(

f"Keras (nsfw sens prev test)\nAUC={aucFmt}\np={ptAt50fmt} @ r=0.5\np={ptAt90fmt} @ r=0.9\nN\_train={...} ({...} pos), N\_test={n\_test\_sens\_prev} ({n\_test\_sens\_prev\_pos} pos)",

size=20

)

plt.subplots\_adjust(top=0.72)

plt.savefig('recall\_precision\_nsfw\_Keras\_with\_twitter\_CLIP\_sens\_prev\_test.pdf')