Deep Learning Home Assignment

Alireza Bolhassani

Download data and images from google drive

```
!wget "https://drive.google.com/u/0/uc?id=1RwL4NeJuw9jm                                          d'
!unzip imgs.zip

!wget "https://drive.google.com/u/0/uc?id=131r_ha4-2_g-                                          d'

from IPython.display import clear_output
clear_output(wait=False)
```

Introduction

In this project, I have developed a machine learning model to classify the Real Sense depth image dataset into two classes.

The model was trained on the Real Sense depth image dataset using a deep learning framework, and its performance was evaluated using various metrics such as training accuracy and loss, confusion matrix, macro-averaged precision and recall, and Precision-Recall curve. These evaluations provide insights into the model's classification performance, convergence behavior during training, and trade-off between precision and recall.

```
import pandas as pd

groundtruth = pd.read_csv("groundtruth.tsv", delimiter="\t")
groundtruth.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2909 entries, 0 to 2908
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   user_id     2909 non-null   object
 1   ad_clicked  2909 non-null   int64 
 2   attention   2909 non-null   int64 
 3   log_id      2909 non-null   int64 
dtypes: int64(3), object(1)
memory usage: 91.0+ KB
```

```
groundtruth.head(3)
```

Zsófia Gyarmathy 12:45 PM Today

Overall a decent implemenation of data preparation, and subsequently one convolutional neural network.

The model should have been optimised though, and not just one setup trained. Also, some more motivation of hyperparameter choices and more reflection on the results would have been appropriate.

The code is mostly functional with a few errors.

The level of explanation of the code is admirable, but sometimes the explanations (or the code) has not been updated to match the task at hand.

The use of sections is helpful, although more levels would have aided navigation of the notebook even more (e.g., 1) Data Preparation; 2) Model; 3) Evaluation).

Zsófia Gyarmathy 12:34 PM Today

It's good if you have an introduction, but please make sure you adapt any old notebook or copy-and-pasted text to the current usecase. (You are not working with the Real Sense depth image dataset any more.)

	user_id	ad_clicked
0	5npsk114ba8hfbj4jr3lt8jhf5	0
1	5o9js8slc8rg2a8mo5p3r93qm0	1
2	[illegible]	0

Generating lists to store the filenames.

Classes includes:

- list_of_class1 is related to clicked ads.
- list_of_class0 is related to not click on ads.

All photos are added to the dataset after resizing.

```
import os

image_list = os.listdir("imgs")

list_of_class1 = []
list_of_class0 = []

for _, row in groundtruth.iterrows():
    if str(row["log_id"]) + ".png" in image_list:
        if row["ad_clicked"]:
            list_of_class1.append(row["log_id"])
        else:
            list_of_class0.append(row["log_id"])

len(list_of_class1), len(list_of_class0)
```

```
(494, 1270)
```

> **Zsófia Gyarmathy** 12:35 PM Today
>
> Correct preparation of positive and negative class image file names.
>
> Small coding hint: a for-cycle is always the slowest; e.g., list comprehension tends to be much faster, but you can use pandas-specific vectorized methods, too, to speed things up. On such a small dataset, it's not relevant, but on a larger one, it can speed things up considerably. E.g, from 319 ms ± 118 ms per loop to 75.8 ms ± 1.55 ms per loop by using, e.g.:
> list_of_class1 = [x for x in groundtruth[groundtruth["ad_clicked"]==1]["log_id"].astype("str") if x + ".png" in image_list]
> list_of_class0 = [x for x in groundtruth[groundtruth["ad_clicked"]==0]["log_id"].astype("str") if x + ".png" in image_list]

Creating labels and assign to lists

I create label arrays labels1, and labels0 by using np.ones() and np.zeros() functions to assign labels to the corresponding lists list_of_class1, and list_of_class0. I also concatenate these label arrays using np.concatenate() function along the 0-axis to create a single labels array that represents the labels for all the image files.

> **Zsófia Gyarmathy** 12:35 PM Today
>
> Correct preparation of labels.

```
import numpy as np

labels1 = np.ones(len(list_of_class1))
labels0 = np.zeros(len(list_of_class0))
labels = np.concatenate((labels1, labels0), axis=0)

len(labels)
```

```
1764
```

> **Zsófia Gyarmathy** 12:35 PM Today
> (edited 12:35 PM Today)
>
> It's very good that you explain the steps you take, but again you should make sure that your explanations match your actual code (in your case, you choose 100x100 as dimension, not 500x500).

Dimensions of images

I create an empty list called dataset to store the data from the images. I also define the desired dimensions of the images as 500 x 500 pixels and store it in a tuple called dim.

```
dataset = []
dim = (100, 100)
```

Zsófia Gyarmathy
12:35 PM Today

Correct preparation of the dataset list; the images in dataset correspond indeed to the labels.

Iterating through the images in lists

```
import cv2

for img in list_of_class1:
  image = cv2.imread(f"imgs/{img}.png")
  resized = cv2.resize(image, dim, interpolation = c
  dataset.append(resized)
for img in list_of_class0:
  image = cv2.imread(f"imgs/{img}.png")
  resized = cv2.resize(image, dim, interpolation = c
  dataset.append(resized)
```

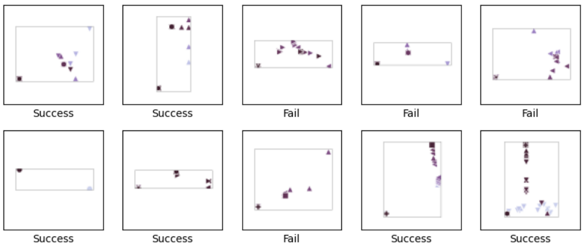
Zsófia Gyarmathy
12:35 PM Today

(Small comment: again, good implementation and code explanation, but please keep it up-to-date: you use 2x5 subplots).

Displaying some pictures

In the below cell, I generate a grid of 3x3 subplots using matplotlib. Each subplot displays an image from the dataset using plt.imshow. The plot is then displayed using plt.show().

```
import matplotlib.pyplot as plt

names = ["Success", "Fail"]
random = np.random.randint(len(dataset), size=10)
plt.figure(figsize=(10,4))
i = 1
for n in random:
    plt.subplot(2, 5, i)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(dataset[n], cmap=plt.cm.binary)
    plt.xlabel(names[int(labels[n])])
    i += 1
plt.show()
```



Zsófia Gyarmathy
12:36 PM Today

It's good again that you explain your code, but this time, your bullet points contradict the text above them. You use 20% of test, 75% of 80% as train,and 25% of 80% a validation.

The implementation of the split is good (default shuffled; and reproducible via random_state), although given the high class imbalance, it would be even better to stratify your splits by the labels.

Converting to NumPy array, Split to training, validation and test sets

In the following, It converts the dataset and labels into NumPy arrays and then partitions them into training, validation, and test sets using the train_test_split function. The test set constitutes 20% of the original dataset, while the training and validation sets are split into 75% and 25% portions, respectively. The code concludes by displaying the shapes of the training, validation, and test sets, providing a concise overview of their dimensions.

- 80 percent of data for training.
- 75 percentage of the 20 percentage of remaining data is used as validation data.
- The rest of data is considered as a test data set

Zsófia Gyarmathy
12:37 PM Today

It's good that you implement scaling, but you want to scale the X_test data as well (and you don't need to scale the X_trainvald one).

```
from sklearn.model_selection import train_test_split

dataset = np.array(dataset)
labels = np.array(labels)

X_trainvald, X_test, y_trainvald, y_test = train_test_s
X_train, X_vald, y_train, y_vald = train_test_split(X_t

# Data scaling
X_trainvald = X_trainvald / 255
X_train = X_train / 255
X_vald = X_vald / 255

print("Training shape:", X_train.shape)
print("Validation shape:", X_vald.shape)
print("Test shape:", X_test.shape)
```

```
Training shape: (1058, 100, 100, 3)
Validation shape: (353, 100, 100, 3)
Test shape: (353, 100, 100, 3)
```

Zsófia Gyarmathy
12:36 PM Today

It's good that you check the X, y shapes, but you made a mistake in the code: you want to check train, valid, test splits, but you check train+valid, train, test.

```
# display the shapes
shapes = [
    [X_trainvald.shape, y_trainvald.shape],
    [X_train.shape, y_train.shape],
    [X_test.shape, y_test.shape],
```

Zsófia Gyarmathy
12:36 PM Today

```
]
shape = pd.DataFrame(shapes, columns=["X", "y"], index=
shape
```

	X	y
Train	(1411, 100, 100, 3)	(1411,)
Validation	(1058, 100, 100, 3)	(1058,)
Test	(353, 100, 100, 3)	(353,)

It's good that you clear session -- hopefully you don't forget to run this cell, too, if you train your model with different hyperparameter setups.

Clear Session

```
from tensorflow.keras import backend
import tensorflow as tf

tf.random.set_seed(42)
backend.clear_session()
```

Zsófia Gyarmathy
12:36 PM Today

Reasonable architecture up until the final layer.

It would be nice if you included some motivation for the hidden layers, as well as for the output layer: how did you arrive at the chosen hyperparameters (even if it's just via a first guess), e.g., number of filters, kernel size, regularizer, number of layers, number of neurons etc.

As for the outer layer, correct choice and explanation of its architecture in the text, but your code unfortunately does not implement that: you use n_classes number of neurons (2 in this case), which should not go with sigmoid, but with softmax activation. (You still can get good results with sigmoid, but nothing precludes the scenario where your loss is getting smaller and smaller while the winning class is still the wrong one.)

Create Convolutional Neural Network Model

layers.Dense(1, activation='sigmoid')

- 1: Single neuron because it's a binary classification task.
- activation='sigmoid': Sigmoid activation function is used for binary classification, mapping the output to a value between 0 and 1.

```
from tensorflow.keras import layers, models, regular

input_tensor = layers.Input(shape=(dim[0], dim[1], 3

################### Conv layer one #################
conv1 = layers.Conv2D(32, (3, 3), activation='relu',
maxpool1 = layers.MaxPooling2D((2, 2))(conv1)
dropout1 = layers.Dropout(0.25)(maxpool1)

################### Conv layer two #################
conv2 = layers.Conv2D(64, (3, 3), activation='relu',
maxpool2 = layers.MaxPooling2D((2, 2))(conv2)
dropout2 = layers.Dropout(0.25)(maxpool2)

conv3 = layers.Conv2D(128, (3, 3), activation='relu'
maxpool3 = layers.MaxPooling2D((2, 2))(conv3)

################### Convert tensors to vectors #####
flatten = layers.Flatten()(maxpool3)

dense1 = layers.Dense(128, activation='relu', kernel
dropout3 = layers.Dropout(0.5)(dense1)

n_classes = len(np.unique(labels))
output_tensor = layers.Dense(n_classes, activation='
modeling = models.Model(inputs=input_tensor, outputs


modeling.summary()
```

```
    Model: "model"
    _________________________________________________
     Layer (type)                Output Shape
    =================================================
     input_1 (InputLayer)        [(None, 100, 100, 3

     conv2d (Conv2D)             (None, 98, 98, 32)

     max_pooling2d (MaxPooling2  (None, 49, 49, 32)
     D)

     dropout (Dropout)           (None, 49, 49, 32)

     conv2d_1 (Conv2D)           (None, 47, 47, 64)

     max_pooling2d_1 (MaxPoolin  (None, 23, 23, 64)
     g2D)

     dropout_1 (Dropout)         (None, 23, 23, 64)

     conv2d_2 (Conv2D)           (None, 21, 21, 128)

     max_pooling2d_2 (MaxPoolin  (None, 10, 10, 128)
     g2D)

     flatten (Flatten)           (None, 12800)

     dense (Dense)               (None, 128)

     dropout_2 (Dropout)         (None, 128)

     dense_1 (Dense)             (None, 2)

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

Zsófia Gyarmathy
12:37 PM Today

Correct choice of loss for the 2-neuron outer layer. (As mentioned, the loss should be softmax in the outer layer, though, to ensure a probability distribution and not "independent probabilities" for the two classes.)

Zsófia Gyarmathy
12:37 PM Today

Correct callbacks, although a patience of 3 epochs for early stopping is probably very much on the low side (especially before you've seen anything of the training curve). Here, again, some motivation of your hyperparameter choices would be good.

Also, if you intend to use both ModelCheckpoint and EarlyStopping callbacks, you will have to make sure that you load the saved model from the checkpoint and use that for evaluation, otherwise you might be dealing with different models! (The ES callback will load the minimum-loss model, which may not correspond to the saved max-accuracy model!)

Zsófia Gyarmathy
12:37 PM Today

Especially with a 100-long maximal epoch training, a 3-epoch patience is very small.

```
Total params: 1732034 (6.61 MB)
Trainable params: 1732034 (6.61 MB)
Non-trainable params: 0 (0.00 Byte)
```

You should make sure your hyperparameter choices as a whole are reasonable.

Compiling

In the following cell, the code prepares the model for training by compiling it with the modeling variable.

```
modeling.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accura
```

In the next cell, with the ModelCheckpoint function, it makes a model checkpoint callback that saves the best model weights to a file called 'my_model.keras' during the training process.

```
from tensorflow.keras.callbacks import ModelCheckpoint

checking = ModelCheckpoint('my_model.keras', save_best_only=True, monitor='val_accuracy', mo

from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

Fitting the model

The epoch parameter in neural network training is a hyperparameter that represents the number of times the entire training dataset is used to update the model weights. Each epoch consists of one or more batches, which are subsets of the training data that are processed in each iteration.

```
epochs = 100
batch_size = 64
history = modeling.fit(X_train, y_train, batch_size=
```

```
Epoch 9/100
17/17 [==============================] - ETA:
Epoch 9: val_accuracy did not improve from 0.7
17/17 [==============================] - 12s 7
Epoch 10/100
17/17 [==============================] - ETA:
Epoch 10: val_accuracy did not improve from 0.
17/17 [==============================] - 12s 7
Epoch 11/100
17/17 [==============================] - ETA:
Epoch 11: val_accuracy did not improve from 0.
17/17 [==============================] - 12s 7
Epoch 12/100
17/17 [==============================] - ETA:
Epoch 12: val_accuracy did not improve from 0.
17/17 [==============================] - 12s 7
Epoch 13/100
17/17 [==============================] - ETA:
Epoch 13: val_accuracy did not improve from 0.
17/17 [==============================] - 12s 6
Epoch 14/100
17/17 [==============================] - ETA:
Epoch 14: val_accuracy did not improve from 0.
17/17 [==============================] - 12s 7
Epoch 15/100
17/17 [==============================] - ETA:
Epoch 15: val_accuracy did not improve from 0.
17/17 [==============================] - 12s 6
Epoch 16/100
17/17 [==============================] - ETA:
Epoch 16: val_accuracy did not improve from 0.
17/17 [==============================] - 12s 6
Epoch 17/100
17/17 [==============================] - ETA:
Epoch 17: val_accuracy did not improve from 0.
17/17 [==============================] - 12s 6
Epoch 18/100
17/17 [==============================] - ETA:
Epoch 18: val_accuracy did not improve from 0.
17/17 [==============================] - 12s 7
Epoch 19/100
17/17 [==============================] - ETA:
Epoch 19: val_accuracy did not improve from 0.
17/17 [==============================] - 12s 7
```

Zsófia Gyarmathy
12:38 PM Today

You shouldn't just print out evaluations and training histories: you should be reflecting on it. What do you conclude from it? How would you update your hyperparameters on this basis?

Printing accuracy and loss by evaluate method

In the following cell, I print the accuracy and loss values obtained from evaluating the model on the test dataset.

```
loss, accuracy = modeling.evaluate(X_test, y_test)

print("Accuracy:", accuracy, "Loss:", loss)
```

```
12/12 [==============================] - 1s 70ms
Accuracy: 0.7110481858253479 Loss: 2.68505239486
```

Model Accuracy and Loss plot

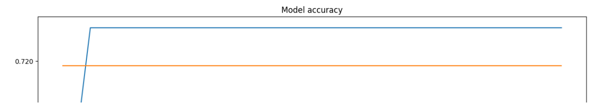
```
fig, axis = plt.subplots(2, 1, figsize=(15, 15))

axis[0].plot(history.history['accuracy'])
axis[0].plot(history.history['val_accuracy'])
axis[0].set_title('Model accuracy')
axis[0].set_ylabel('Accuracy')
axis[0].set_xlabel('Epoch')
axis[0].legend(['Train', 'Validation'], loc='center

axis[1].plot(history.history['loss'])
axis[1].plot(history.history['val_loss'])
axis[1].set_title('Model loss')
axis[1].set_ylabel('Loss')
axis[1].set_xlabel('Epoch')
axis[1].legend(['Train', 'Validation'], loc='center

plt.show()
```

Zsófia Gyarmathy
12:39 PM Today

It's good that you check the confusion matrix and the precision-recall curve, well done! However, again you shouldn't just display them, but also interpret them.

Zsófia Gyarmathy
12:41 PM Today

Again, you should make sure you adapt any copy-and-pasted code to your task: you are

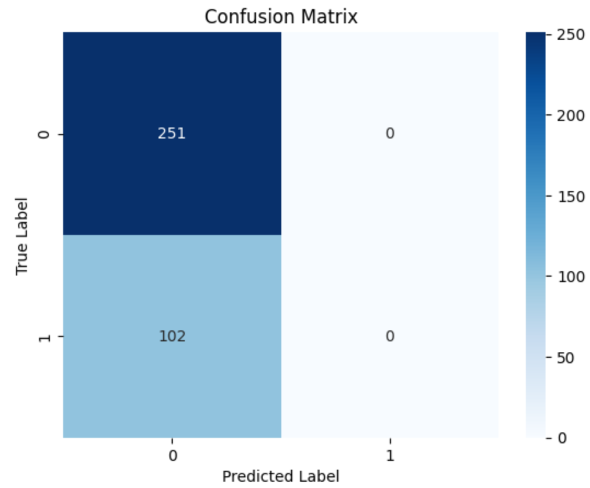


Evaluation using Confusion Matrix

The following code generates a confusion matrix using the seaborn library to evaluate the performance of a trained model.

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

y_prob = modeling.predict(X_test)
y_pred = np.argmax(y_prob, axis=1)
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
12/12 [==============================]
```



Precision Recall Curve

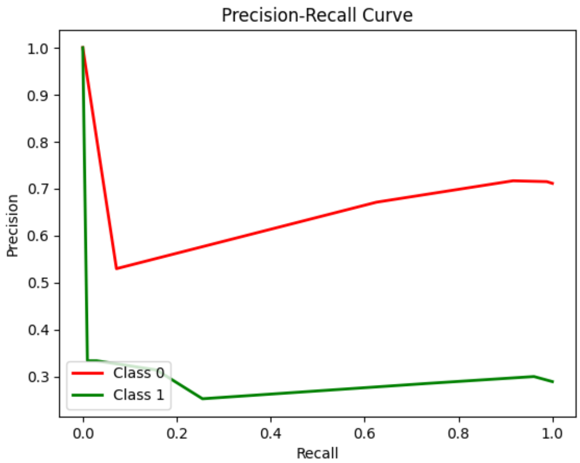
dealing with a 2-way, not 3-way classification.

Zsófia Gyarmathy
12:42 PM Today

It's good that you try to form some conclusion, but it should be more specific. Why are the results not acceptable? (You could, e.g., say that the network simply learned to predict the majority class.) I suppose you indicate that an important reason behind the performance you see is class imbalance. In that case, how would you change your hyperparameters? E.g., you could employ class weights etc. If the results are not acceptable, you should ideally take that as an indication that further exploration of the hyperparameter space is needed (provided you know that the data is prepared appropriately).

```
from sklearn.metrics import precision_recall_curve,
from sklearn.preprocessing import label_binarize

y_test_bin = label_binarize(y_test, classes=[0, 1, 2
y_prob = modeling.predict(X_test)
precision = dict()
recall = dict()
for i in range(2):
    precision[i], recall[i], _ = precision_recall_cu
plt.figure()
for i, color in zip(range(2), ['red', 'green']):
    plt.plot(recall[i], precision[i], color=color, l
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="lower left")
plt.show()
```

```
12/12 [==============================]
```



Conclusion

Unfortunately, the results are not acceptable, even though the parameters of the model have been studied a lot, but the results are still weak. According to the investigation carried out, I consider two possible reasons for it.

First, the amount of information for an artificial intelligence network is not only not enough, but it is not even symmetrical.