COMP5625M Assessment 2 - Image Caption Generation [100 marks]

The maximum marks for each part are shown in the section headers. The overall assessment carries a total of 100 marks.

This assessment is weighted 25% of the final grade for the module.

Motivation

Through this assessment, you will:

1. Understand the principles of text pre-processing and vocabulary building.
2. Gain experience working with an image-to-text model.
3. Use and compare two text similarity metrics for evaluating an image-to-text model, and understand evaluation challenges.

Setup and resources

Having a GPU will speed up the image feature extraction process. If you want to use a GPU, please refer to the module website for recommended working environments with GPUs.

Please implement the coursework using PyTorch and Python-based libraries, and refer to the notebooks and exercises provided.

This assessment will use a subset of the COCO "Common Objects in Context" dataset for image caption generation. COCO contains 330K images of 80 object categories, and at least five textual reference captions per image. Our subset consists of nearly 5070 of these images, each with five or more different descriptions of the salient entities and activities, and we will refer to it as COCO_5070.

To download the data:

1. **Images and annotations**: download the zipped file provided in the link here as `COMP5625M_data_assessment_2.zip`.

`Info only:` To understand more about the COCO dataset, you can look at the download page. We have already provided you with the "2017 Train/Val annotations (241MB)", but our image subset consists of fewer images than the original COCO dataset. **So, no need to download anything from here!**

2. **Image metadata**: as our set is a subset of the full COCO dataset, we have created a CSV file containing relevant metadata for our particular subset of images. You can also download it from Drive, "coco_subset_meta.csv", at the same link as 1.

Submission

Please submit the following:

1. Your completed Jupyter notebook file, in .ipynb format. **Do not change the file name.**
2. The .html version of your notebook; File > Download as > HTML (.html). Check that all cells have been run and all outputs (including all graphs you would like to be marked) are displayed in the .html for marking.

Final note:

> **Please include everything you would like to be marked in this notebook, including figures. Under each section, put the relevant code containing your solution. You may re-use functions you defined previously, but any new code must be in the appropriate section.** Feel free to add as many code cells as you need under each section.

Your student username (for example, `sc15jb`):

mm22mp

Your full name:

Meghana Puttaswamy

Imports

Feel free to add to this section as needed.

```
import torch
import torch.nn as nn
from torchvision import transforms
import torchvision.models as models
from torch.utils.data import Dataset
from torch.nn.utils.rnn import pack_padded_sequence, pad_packed_sequence
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter
from torch.utils.data import DataLoader
from PIL import Image
import os
import numpy as np
```

Detect which device (CPU/GPU) to use.

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print('Using device:', device) #Printing the device used
```

```
Using device: cpu
```

```
from google.colab import drive
drive.mount('/content/drive') # mounting the drive for data set
```

```
Mounted at /content/drive
```

The basic principle of our image-to-text model is as pictured in the diagram below, where an Encoder network encodes the input image as a feature vector by providing the outputs of the last convolutional layer of a pre-trained CNN (we use ResNet50). This pretrained network has been trained on the complete ImageNet dataset and is thus able to recognise common objects.

(Hint) You can alternatively use the COCO trained pretrained weights from PyTorch. One way to do this is use the "FasterRCNN_ResNet50_FPN_V2_Weights.COCO_V1" but use e.g., "resnet_model = model.backbone.body". Alternatively, you can use the checkpoint from your previous coursework where you finetuned to COCO dataset.

These features are then fed into a Decoder network along with the reference captions. As the image feature dimensions are large and sparse, the Decoder network includes a linear layer which downsizes them, followed by *a batch normalisation layer* to speed up training. Those resulting features, as well as the reference text captions, are then passed into a recurrent network (we will use **RNN** in this assessment).

The reference captions used to compute loss are represented as numerical vectors via an **embedding layer** whose weights are learned during training.

The Encoder-Decoder network could be coupled and trained end-to-end, without saving features to disk; however, this requires iterating through the entire image training set during training. We can make the **training more efficient by decoupling the networks**. Thus, we will:

> First extract the feature representations of the images from the Encoder

> Save these features (Part 1) such that during the training of the Decoder (Part 3), we only need to iterate over the image feature data and the reference captions.

Hint Try commenting out the feature extraction part once you have saved the embeddings. This way if you have to re-run the entire codes for some reason then you can only load these features.

Overview

> 1. Extracting image features
> 2. Text preparation of training and validation data
> 3. Training the decoder
> 4. Generating predictions on test data
> 5. Caption evaluation via BLEU score
> 6. Caption evaluation via Cosine similarity
> 7. Comparing BLEU and Cosine similarity

1 Extracting image features [11 marks]

> 1.1 Design a encoder layer with pretrained ResNet50 (4 marks)

> 1.2 Image feature extraction step (7 marks)

1.1 Design a encoder layer with pretrained ResNet50 (4 marks)

> Read through the template EncoderCNN class below and complete the class.

> You are expected to use ResNet50 pretrained on imageNet provided in the Pytorch library (torchvision.models)

```
class EncoderCNN(nn.Module):
    def __init__(self):
        """Load the pretrained ResNet-50 and replace top fc layer."""
        super(EncoderCNN, self).__init__()
        resnet = models.resnet50(pretrained=True)#Defining the pre-trained Resnet-50
        for param in resnet.parameters():
            param.requires_grad_(False) # Freezing the trained weights
        modules = list(resnet.children())[:-1] # delete the last fully connected layer.
        self.resnet = nn.Sequential(*modules) ## Create the sequential model and add the modules in order

    def forward(self, images):
        """Extract feature vectors from input images."""
        with torch.no_grad():
            features = self.resnet(images) #Using Resnet for the forward navigation
        return features


# instantiate encoder and put into evaluation mode.
encoder = EncoderCNN().eval()
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /root/.cache/torch/hub/checkpoints/resnet50-0676ba61.pth
100%|██████████| 97.8M/97.8M [00:01<00:00, 75.5MB/s]
```

1.2 Image feature extraction step (7 marks)

Pass the images through the `Encoder` model, saving the resulting features for each image. You may like to use a `Dataset` and `DataLoader` to load the data in batches for faster processing, or you may choose to simply read in one image at a time from disk without any loaders.

Note that as this is a forward pass only, no gradients are needed. You will need to be able to match each image ID (the image name without file extension) with its features later, so we suggest either saving a dictionary of image ID: image features, or keeping a separate ordered list of image IDs.

Use this ImageNet transform provided.

```
transform = transforms.Compose([
        transforms.Resize(256),
        transforms.RandomCrop(224),
        #transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(degrees=45),
        transforms.ToTensor(),
        transforms.Normalize((0.485, 0.456, 0.406),
                             (0.229, 0.224, 0.225))])


# Get unique images from the csv for extracting features
imageList = pd.read_csv("/content/drive/MyDrive/Assesment_2_DL/Assesment2/COMP5625M_data_assessment_2/coco_subset_meta.csv")
imagesUnique = sorted(imageList['file_name'].unique()) #Get sorted list of unique images
df_unique_files =  pd.DataFrame.from_dict(imagesUnique)
df_unique_files.columns = ['file_name']
df_unique_files #Print unique file names
```

	file_name
0	000000000009.jpg
1	000000000025.jpg
2	000000000030.jpg
3	000000000034.jpg
4	000000000036.jpg
...	...
5063	000000581906.jpg

```
# Define a class COCOImagesDataset(Dataset) function that takes the
# image file names and reads the image and apply transform to it
IMAGE_DIR = "/content/drive/MyDrive/Assesment_2_DL/Assesment2/COMP5625M_data_assessment_2/coco/images/"

class COCOImagesDataset(Dataset):
    def __init__(self, df, image_dir, transform=None):
        self.df = df
        self.image_dir = image_dir
        self.transform = transform #Image transform

    def __getitem__(self, index):
        filename = self.df.iloc[index]['file_name']
        image = Image.open(os.path.join(self.image_dir,filename)).convert("RGB")
        if self.transform:
            image = self.transform(image)
        return image, filename

    def __len__(self):
        return len(self.df)



#Loading the data
dataset = COCOImagesDataset(df_unique_files,IMAGE_DIR, transform=transform)
dataloader = DataLoader(dataset, batch_size=1, shuffle=False)


# Apply encoder to extract featues and save them (e.g., you can save it using image_ids)
# Hint - make sure to save your features after running this - you can use torch.save to do this

features_map = dict()
from tqdm.notebook import tqdm
from PIL import Image

encoder.resnet.to(device)

for images,filename in tqdm(dataloader):
    images = images.to(device)
    with torch.no_grad():
        features = encoder.resnet(images)
    features = features.squeeze()
    features_map[filename[0]] = features
torch.save(features_map, "features_map.pt")
```

```
100%  5068/5068 [39:26<00:00, 2.33it/s]
```

2 Text preparation [23 marks]

> 2.1 Build the caption dataset (3 Marks)

> 2.2 Clean the captions (3 marks)

> 2.3 Split the data (3 marks)

> 2.4 Building the vocabulary (10 marks)

> 2.5 Prepare dataset using dataloader (4 marks)

2.1 Build the caption dataset (3 Marks)

All our selected COCO_5029 images are from the official 2017 train set.

The `coco_subset_meta.csv` file includes the image filenames and unique IDs of all the images in our subset. The `id` column corresponds to each unique image ID.

The COCO dataset includes many different types of annotations: bounding boxes, keypoints, reference captions, and more. We are interested in the captioning labels. Open `captions_train2017.json` from the zip file downloaded from the COCO website. You are welcome to come up with your own way of doing it, but we recommend using the `json` package to initially inspect the data, then the `pandas` package to look at the annotations (if you read in the file as `data`, then you can access the annotations dictionary as `data['annotations']`).

Use `coco_subset_meta.csv` to cross-reference with the annotations from `captions_train2017.json` to get all the reference captions for each image in COCO_5029.

For example, you may end up with data looking like this (this is a `pandas` DataFrame, but it could also be several lists, or some other data structure/s):

```
features_map = torch.load("features_map.pt")
import json

# Load Captions
with open('/content/drive/MyDrive/Assesment_2_DL/Assesment2/COMP5625M_data_assessment_2/coco/annotations2017/captions_train2017.json', 'r') a
    data = json.load(json_file)
df = pd.DataFrame.from_dict(data["annotations"])
df.head()
```

	image_id	id	caption
0	203564	37	A bicycle replica with a clock as the front wh...
1	322141	49	A room with blue walls and a white sink and door.
2	16977	89	A car that seems to be parked illegally behind...
3	106140	98	A large passenger airplane flying through the ...
4	106140	101	There is a GOL plane taking off in a partly cl...

```
#print((features_map['000000335914.jpg']))


# get the filename matching id from coco_subset_meta.csv
coco_subset = pd.read_csv("/content/drive/MyDrive/Assesment_2_DL/Assesment2/COMP5625M_data_assessment_2/coco_subset_meta.csv")
data_coco = coco_subset[['id', 'file_name']]
new_file = pd.merge(data_coco, df.drop('id', axis=1), left_on='id', right_on='image_id')
new_file = new_file.drop_duplicates()[['image_id','id','caption','file_name']]# For each id, adding image and filename
new_file #print the new_file
```

	image_id	id	caption	file_name
0	262145	262145	People shopping in an open market for vegetables.	000000262145.jpg
1	262145	262145	An open market full of people and piles of veg...	000000262145.jpg
2	262145	262145	People are shopping at an open air produce mar...	000000262145.jpg
3	262145	262145	Large piles of carrots and potatoes at a crowd...	000000262145.jpg
4	262145	262145	People shop for vegetables like carrots and po...	000000262145.jpg
...	...	...	...	...
40008	80067	80067	a dog that has a tooth brush in his mouth	000000080067.jpg
40009	80067	80067	Dog with a tooth brush in mouth in floor	000000080067.jpg
40010	80067	80067	Small gray and black dog holding a toothbrush ...	000000080067.jpg
40011	80067	80067	a close up o a dog with a tooth brush in its m...	000000080067.jpg
40012	80067	80067	A dog with a tooth brush in his mouth.	000000080067.jpg

25342 rows × 4 columns

2.2 Clean the captions (3 marks)

Create a cleaned version of each caption. If using dataframes, we suggest saving the cleaned captions in a new column; otherwise, if you are storing your data in some other way, create data structures as needed.

A cleaned caption should be all lowercase, and consist of only alphabet characters.

Print out 10 original captions next to their cleaned versions to facilitate marking.

 images matched to caption

```
new_file["clean_caption"] = "" # add a new column to the dataframe for the cleaned captions
import re
def gen_clean_captions_df(df):

    # Remove spaces in the beginning and at the end
    # Convert to lower case
    # Replace all non-alphabet characters with space
    # Replace all continuous spaces with a single space

    # -->your code here
    clean_captions = []
    for caption in df['caption']:
        cleaned_caption = re.sub('[^a-zA-Z]', ' ', caption.lower()).strip()
        cleaned_caption = re.sub('\s+', ' ', cleaned_caption)
        clean_captions.append(cleaned_caption)

    # add to dataframe
    df["clean_caption"] = clean_captions

    return df


# clean and print 10 of these
new_file = gen_clean_captions_df(new_file)
new_file.head(10)
```

	image_id	id	caption	file_name	clean_caption
0	262145	262145	People shopping in an open market for vegetables.	000000262145.jpg	people shopping in an open market for vegetables
1	262145	262145	An open market full of people and piles of veg...	000000262145.jpg	an open market full of people and piles of veg...
2	262145	262145	People are shopping at an open air produce mar...	000000262145.jpg	people are shopping at an open air produce market
3	262145	262145	Large piles of carrots and potatoes at a crowd...	000000262145.jpg	large piles of carrots and potatoes at a crowd...
4	262145	262145	People shop for vegetables like carrots and po...	000000262145.jpg	people shop for vegetables like carrots and po...
15	262146	262146	a person skiing down a steep hill	000000262146.jpg	a person skiing down a steep hill
16	262146	262146	A person skiing down a steep snowy hill.	000000262146.jpg	a person skiing down a steep snowy hill
17	262146	262146	A person on snow skis going down a steep slope.	000000262146.jpg	a person on snow skis going down a steep slope
18	262146	262146	A skier is skiing down a down hill slope.	000000262146.jpg	a skier is skiing down a down hill slope
19	262146	262146	A skier is shown taking on a very steep slope.	000000262146.jpg	a skier is shown taking on a very steep slope

2.3 Split the data (3 marks)

Split the data 70/10/20% into train/validation/test sets. **Be sure that each unique image (and all corresponding captions) only appear in a single set.**

We provide the function below which, given a list of unique image IDs and a 3-split ratio, shuffles and returns a split of the image IDs.

If using a dataframe, `df['image_id'].unique()` will return the list of unique image IDs.

```
import random
import math

def split_ids(image_id_list, train=.7, valid=0.1, test=0.2):
    """
    Args:
        image_id_list (int list): list of unique image ids
        train (float): train split size (between 0 - 1)
        valid (float): valid split size (between 0 - 1)
        test (float): test split size (between 0 - 1)
    """
```

```
    list_copy = image_id_list.copy()
    random.shuffle(list_copy)

    train_size = math.floor(len(list_copy) * train)
    valid_size = math.floor(len(list_copy) * valid)

    return list_copy[:train_size], list_copy[train_size:(train_size + valid_size)], list_copy[(train_size + valid_size):]


unique_ids = list(new_file['image_id'].unique())
train,val,test = split_ids(unique_ids)


print ("The number of images in the training set is: ",len(train))
print ("The number of images in the validation set is: ",len(val))
print ("The number of images in the testing set is: ",len(test))
```

```
 The number of images in the training set is:  3547
 The number of images in the validation set is:  506
 The number of images in the testing set is:  1015
```

2.4 Building the vocabulary (10 marks)

The vocabulary consists of all the possible words which can be used - both as input into the model, and as output predictions, and we will build it using the cleaned words found in the reference captions from the training set. In the vocabulary each unique word is mapped to a unique integer (a Python `dictionary` object).

A `Vocabulary` object is provided for you below to use.

```
class Vocabulary(object):
    """ Simple vocabulary wrapper which maps every unique word to an integer ID. """
    def __init__(self):
        # intially, set both the IDs and words to dictionaries with special tokens
        self.word2idx = {'<pad>': 0, '<unk>': 1, '<end>': 2}
        self.idx2word = {0: '<pad>', 1: '<unk>', 2: '<end>'}
        self.idx = 3

    def add_word(self, word):
        # if the word does not already exist in the dictionary, add it
        if not word in self.word2idx:
            # this will convert each word to index and index to word as you saw in the tutorials
            self.word2idx[word] = self.idx
            self.idx2word[self.idx] = word
            # increment the ID for the next word
            self.idx += 1

    def __call__(self, word):
        # if we try to access a word not in the dictionary, return the id for <unk>
        if not word in self.word2idx:
            return self.word2idx['<unk>']
        return self.word2idx[word]

    def __len__(self):
        return len(self.word2idx)
```

Collect all words from the cleaned captions in the **training and validation sets**, ignoring any words which appear 3 times or less; this should leave you with roughly 2200 words (plus or minus is fine). As the vocabulary size affects the embedding layer dimensions, it is better not to add the very infrequently used words to the vocabulary.

Create an instance of the `Vocabulary()` object and add all your words to it.

```
# [Hint] building a vocab function such with frequent words e.g., setting MIN_FREQUENCY = 3
MIN_FREQUENCY = 4

def build_vocab(df_ids, new_file):
    """
    Parses training set token file captions and builds a Vocabulary object and dataframe for
    the image and caption data

    Returns:
        vocab (Vocabulary): Vocabulary object containing all words appearing more than min_frequency
    """
    word_mapping = Counter()
```

```
    for index, id in enumerate(df_ids):
        caption_list = list(new_file.loc[new_file['image_id']==id]['clean_caption'])
        for caption in caption_list:
            for word in caption.split():
                if word in word_mapping:
                    word_mapping[word] += 1
                else:
                    word_mapping[word] = 1

    # create a vocab instance
    vocab = Vocabulary()

    # add the words to the vocabulary
    for word in word_mapping:
        # Ignore infrequent words to reduce the embedding size
        if word_mapping[word]>=MIN_FREQUENCY:
            vocab.add_word(word)
    return vocab


# build your vocabulary for train, valid and test sets
# ---> your code here!
train_vocab = build_vocab(train,new_file)
val_vocab = build_vocab(val,new_file)
trainval_vocab = build_vocab((train+val),new_file)
test_vocab = build_vocab(test,new_file)


print('The number of words in training vocabulary is:',len(train_vocab))
print('The number of words in validation vocabulary is:',len(val_vocab))
print('The number of words in testing vocabulary is:',len(test_vocab))
print('The number of words in combined training and validation vocabulary is:',len(trainval_vocab))
```

```
 The number of words in training vocabulary is: 2272
 The number of words in validation vocabulary is: 751
 The number of words in testing vocabulary is: 1129
 The number of words in combined training and validation vocabulary is: 2429
```

```
#Printing the max caption length
max_cap_len = 0
for caption in new_file['clean_caption']:
    word_len = len(caption.split())
    if max_cap_len<word_len:
        max_cap_len = word_len
print (max_cap_len)
```

```
 46
```

2.5 Prepare dataset using dataloader (4 marks)

Create a PyTorch `Dataset` class and a corresponding `DataLoader` for the inputs to the decoder. Create three sets: one each for training, validation, and test. Set `shuffle=True` for the training set DataLoader.

The `Dataset` function `__getitem__(self, index)` should return three Tensors:

> 1. A Tensor of image features, dimension (1, 2048).
> 2. A Tensor of integer word ids representing the reference caption; use your `Vocabulary` object to convert each word in the caption to a word ID. Be sure to add the word ID for the `<end>` token at the end of each caption, then fill in the the rest of the caption with the `<pad>` token so that each caption has uniform lenth (max sequence length) of **47**.
> 3. A Tensor of integers representing the true lengths of every caption in the batch (include the `<end>` token in the count).

Note that as each unique image has five or more (say, `n`) reference captions, each image feature will appear `n` times, once in each unique (feature, caption) pair.

```
class COCO_Features(Dataset):
    """ COCO subset custom dataset, compatible with torch.utils.data.DataLoader. """

    def __init__(self, df, features, vocab):
        """
        Args:
            df: (dataframe or some other data structure/s you may prefer to use)
            features: image features
            vocab: vocabulary wrapper
```

```
        """
        self.df = df
        self.features = features
        self.vocab = vocab
        self.MAX_SEQ_LEN = 47

    def __getitem__(self, index):
        """ Returns one data tuple (feature [1, 2048], target caption of word IDs [1, 47], and integer true caption length) """
        filename = self.df['file_name'].iloc[index]
        feature_image = self.features[filename]
        caption_ids = []
        caption = self.df['clean_caption'].iloc[index]
        #Plus one for <end> token
        caption_len = len(caption.split())+1
        for word in caption.split():
            caption_ids.append(self.vocab(word))
        caption_ids.append(self.vocab('<end>'))
        if len(caption_ids)<self.MAX_SEQ_LEN:
            for pads in range(0,self.MAX_SEQ_LEN-len(caption_ids)):
                caption_ids.append(self.vocab('<pad>'))
        caption_ids = torch.as_tensor(caption_ids)
        return (feature_image.unsqueeze(0),caption_ids,caption_len)


    def __len__(self):
        return len(self.features)


def caption_collate_fn(data):
    """ Creates mini-batch tensors from the list of tuples (image, caption).
    Args:
        data: list of tuple (image, caption).
            - image: torch tensor of shape (3, 224, 224).
            - caption: torch tensor of shape (?); variable length.
    Returns:
        images: torch tensor of shape (batch_size, 3, 224, 224).
        targets: torch tensor of shape (batch_size, padded_length).
        lengths: list; valid length for each padded caption.
    """
    # Sort the data by caption length (descending order) to use pack_padded_sequence
    data.sort(key=lambda x: x[2], reverse=True)
    images, captions, lengths = zip(*data)

    # Merge images (from tuple of 1D tensor to 2D tensor).
    images = torch.cat(images, 0).unsqueeze(1)

    targets = torch.stack(captions)

    return images, targets, torch.tensor(lengths)




def caption_collate_fn(data):
    """ Creates mini-batch tensors from the list of tuples (image, caption).
    Args:
        data: list of tuple (image, caption).
            - image: torch tensor of shape (3, 224, 224).
            - caption: torch tensor of shape (?); variable length.
    Returns:
        images: torch tensor of shape (batch_size, 3, 224, 224).
        targets: torch tensor of shape (batch_size, padded_length).
        lengths: list; valid length for each padded caption.
    """
    # Sort the data by caption length (descending order) to use pack_padded_sequence
    data.sort(key=lambda x: x[2], reverse=True)
    images, captions, lengths = zip(*data)

    # Merge images (from tuple of 1D tensor to 2D tensor).
    images = torch.cat(images, 0).unsqueeze(1)

    targets = torch.stack(captions)

    return images, targets, torch.tensor(lengths)
```

```
train_df = new_file[new_file['image_id'].isin(train)]
val_df = new_file[new_file['image_id'].isin(val)]
test_df = new_file[new_file['image_id'].isin(test)]

train_df_file_names = list(train_df['file_name'].unique())
val_df_file_names = list(val_df['file_name'].unique())
test_df_file_names = list(test_df['file_name'].unique())

train_feature_map = {key: features_map[key] for key in train_df_file_names if key in features_map}
val_feature_map = {key: features_map[key] for key in val_df_file_names if key in features_map}
test_feature_map = {key: features_map[key] for key in test_df_file_names if key in features_map}


dataset_train = COCO_Features(
    df=train_df,
    vocab=trainval_vocab,
    features=train_feature_map,
)

#  your dataloader here (make shuffle true as you will be training RNN)
train_loader = DataLoader(dataset_train, batch_size=64, shuffle=True, collate_fn=caption_collate_fn)

# Do the same as above for your validation set
dataset_val = COCO_Features(
    df=val_df,
    vocab=trainval_vocab,
    features=val_feature_map
)

val_loader = DataLoader(dataset_val, batch_size=64, shuffle=False, collate_fn=caption_collate_fn)
```

Load one batch of the training set and print out the shape of each returned Tensor.

```
tensors = list(next(iter(val_loader)))
for i in range(len(tensors)):
    print ("The shape of tensor-"+str(i+1)+" is: ",tensors[i].shape)
```

```
 The shape of tensor-1 is:  torch.Size([64, 1, 2048])
 The shape of tensor-2 is:  torch.Size([64, 47])
 The shape of tensor-3 is:  torch.Size([64])
```

3 Train DecoderRNN [20 marks]

> 3.1 Design RNN-based decoder (10 marks)

> 3.2 Train your model with precomputed features (10 Marks)

3.1 Design a RNN-based decoder (10 marks)

Read through the `DecoderRNN` model below. First, complete the decoder by adding an `RNN` layer to the decoder where indicated, using the PyTorch API as reference.

Keep all the default parameters except for `batch_first`, which you may set to True.

In particular, understand the meaning of `pack_padded_sequence()` as used in `forward()`. Refer to the PyTorch `pack_padded_sequence()` documentation.

```
from torch.nn.modules import dropout
from torch.nn.utils.rnn import pack_padded_sequence

class DecoderRNN(nn.Module):
    def __init__(self, vocab_size, embed_size=256, hidden_size=512, num_layers=1, max_seq_length=47):
        super(DecoderRNN, self).__init__()

        # Linear layer to resize the input features
        self.resize = nn.Linear(2048, embed_size).to(device)

        # Batch normalization layer
        self.bn = nn.BatchNorm1d(embed_size, momentum = 0.01).to(device)

        # Embedding layer
        self.embed = nn.Embedding(vocab_size, embed_size).to(device)
```

```
        # RNN layer
        self.rnn = nn.RNN(embed_size, hidden_size, num_layers, batch_first=True).to(device)

        # Linear layer to generate the output
        self.linear = nn.Linear(hidden_size, vocab_size).to(device)

        # Maximum sequence length
        self.max_seq_length = max_seq_length
        self.drop = nn.Dropout(0.5)
    def forward(self, features, captions, lengths):

        # Move tensors to the same device as the model
        features = features.to(self.embed.weight.device)
        captions = captions.to(self.embed.weight.device)
        lengths = lengths.to(self.embed.weight.device)

        # Resize the input features
        im_features = self.resize(features).squeeze(1)

        # Apply batch normalization
        im_features = self.bn(im_features)

        # Embed the captions
        embeddings = self.embed(captions)

        # Concatenate the image features and caption embeddings
        inputs = torch.cat((im_features.unsqueeze(1), embeddings), 1)

        # Pack the padded sequence
        packed = pack_padded_sequence(inputs, lengths, batch_first=True)

        # Pass the packed sequence through the RNN layer
        outputs, _ = self.rnn(packed)

        # Unpack the padded sequence
        outputs, _ = torch.nn.utils.rnn.pad_packed_sequence(outputs, batch_first=True)

        # Generate the output
        outputs = self.linear(self.drop(outputs))

        return outputs

    def sample(self, features, states=None):
        """Generate captions for given image features using greedy search."""
        sampled_ids = []

        inputs = self.bn(self.resize(features).squeeze(1)).unsqueeze(1)
        for i in range(self.max_seq_length):
            hiddens, states = self.rnn(inputs, states)  # hiddens: (batch_size, 1, hidden_size)
            outputs = self.linear((hiddens.squeeze(1)))   # outputs:  (batch_size, vocab_size)
            _, predicted = outputs.max(1)               # predicted: (batch_size)
            sampled_ids.append(predicted)
            inputs = self.embed(predicted)              # inputs: (batch_size, embed_size)
            inputs = inputs.unsqueeze(1)                # inputs: (batch_size, 1, embed_size)
        sampled_ids = torch.stack(sampled_ids, 1)       # sampled_ids: (batch_size, max_seq_length)
        return sampled_ids


decoder = DecoderRNN(len(trainval_vocab))
decoder
```

```
DecoderRNN(
  (resize): Linear(in_features=2048, out_features=256, bias=True)
  (bn): BatchNorm1d(256, eps=1e-05, momentum=0.01, affine=True, track_running_stats=True)
  (embed): Embedding(2429, 256)
  (rnn): RNN(256, 512, batch_first=True)
  (linear): Linear(in_features=512, out_features=2429, bias=True)
  (drop): Dropout(p=0.5, inplace=False)
)
```

3.2 Train your model with precomputed features (10 marks)

Train the decoder by passing the features, reference captions, and targets to the decoder, then computing loss based on the outputs and the targets. Note that when passing the targets and model outputs to the loss function, the targets will also need to be formatted using `pack_padded_sequence()`.

We recommend a batch size of around 64 (though feel free to adjust as necessary for your hardware).

We strongly recommend saving a checkpoint of your trained model after training so you don't need to re-train multiple times.

Display a graph of training and validation loss over epochs to justify your stopping point.

```
from tqdm import tqdm
import torch.optim as optim
from tqdm import tqdm
criterion = nn.CrossEntropyLoss()

#optimizer = optim.Adam(decoder.parameters(), lr=0.0001)
optimizer = optim.SGD(decoder.parameters(), lr=0.05)
num_epochs = 50
```

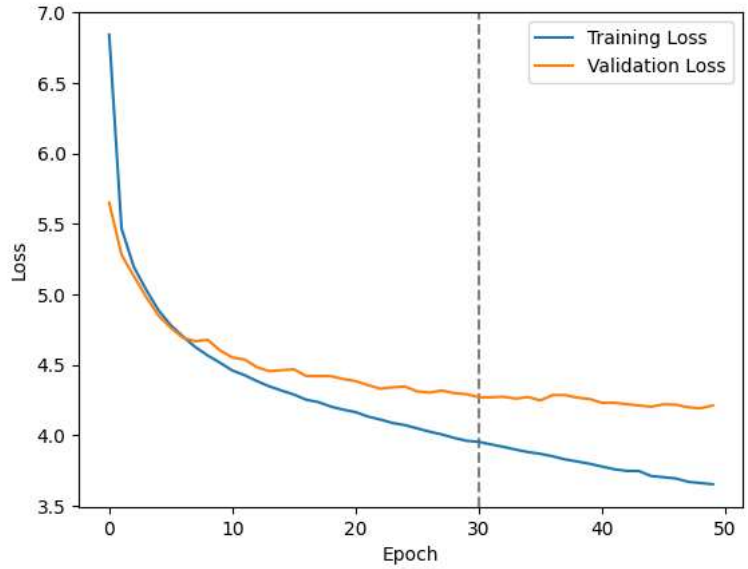
```
# Define empty lists to store the training and validation losses
train_losses = []
val_losses = []

# Loop over the epochs
for epoch in range(num_epochs):
    # Set the decoder to training mode
    decoder.train()

    # Initialize the running loss for this epoch
    running_loss = 0.0

    # Loop over the training set in batches
    for i, (features, captions, lengths) in enumerate(tqdm(train_loader)):
        # Move the inputs and targets to the device
        targets = pack_padded_sequence(captions[:, 1:], lengths, batch_first=True)[0]

        # Zero the gradients
        decoder.zero_grad()

        # Forward pass through the decoder
        outputs = decoder(features, captions, lengths)
        outputs = pack_padded_sequence(outputs, lengths, batch_first=True)[0]

        # Compute the loss
        loss = criterion(outputs, targets)

        # Backward pass and optimization step
        loss.backward()
        optimizer.step()

        # Update the running loss
        running_loss += loss.item()

    # Set the decoder to evaluation mode
    decoder.eval()

    # Initialize the running validation loss for this epoch
    running_val_loss = 0.0

    # Loop over the validation set in batches
    with torch.no_grad():
        for i, (features, captions, lengths) in enumerate(val_loader):
            # Move the inputs and targets to the device
            features = features
            captions = captions
            lengths = lengths
            targets = pack_padded_sequence(captions[:, 1:], lengths, batch_first=True)[0]

            # Forward pass through the decoder
            outputs = decoder(features, captions, lengths)
            outputs = pack_padded_sequence(outputs, lengths, batch_first=True)[0]

            # Compute the loss
            loss = criterion(outputs, targets)

            # Update the running validation loss
            running_val_loss += loss.item()
```

```
    # Print the epoch status
    print(f"Epoch [{epoch+1}/{num_epochs}], Training Loss: {running_loss/len(train_loader):.4f}, Validation Loss: {running_val_loss/len(val_l

    # Append the training and validation losses to the corresponding lists
    train_losses.append(running_loss/len(train_loader))
    val_losses.append(running_val_loss/len(val_loader))

    # Save a checkpoint of the model
    torch.save({'decoder_state_dict':decoder.state_dict(),'train_losses': train_losses,'val_losses': val_losses}, f"decoder-{epoch+1}.ckpt")
```

```
100%|██████████| 56/56 [00:17<00:00,  3.29it/s]
Epoch [22/50], Training Loss: 4.1336, Validation Loss: 4.3570
100%|██████████| 56/56 [00:16<00:00,  3.31it/s]
Epoch [23/50], Training Loss: 4.1127, Validation Loss: 4.3308
100%|██████████| 56/56 [00:17<00:00,  3.29it/s]
Epoch [24/50], Training Loss: 4.0879, Validation Loss: 4.3409
100%|██████████| 56/56 [00:16<00:00,  3.37it/s]
Epoch [25/50], Training Loss: 4.0728, Validation Loss: 4.3464
100%|██████████| 56/56 [00:16<00:00,  3.30it/s]
Epoch [26/50], Training Loss: 4.0498, Validation Loss: 4.3117
100%|██████████| 56/56 [00:16<00:00,  3.35it/s]
Epoch [27/50], Training Loss: 4.0263, Validation Loss: 4.3037
100%|██████████| 56/56 [00:16<00:00,  3.31it/s]
Epoch [28/50], Training Loss: 4.0065, Validation Loss: 4.3180
100%|██████████| 56/56 [00:17<00:00,  3.28it/s]
Epoch [29/50], Training Loss: 3.9824, Validation Loss: 4.2996
100%|██████████| 56/56 [00:16<00:00,  3.33it/s]
Epoch [30/50], Training Loss: 3.9618, Validation Loss: 4.2924
100%|██████████| 56/56 [00:16<00:00,  3.29it/s]
Epoch [31/50], Training Loss: 3.9542, Validation Loss: 4.2705
100%|██████████| 56/56 [00:16<00:00,  3.34it/s]
Epoch [32/50], Training Loss: 3.9358, Validation Loss: 4.2705
100%|██████████| 56/56 [00:17<00:00,  3.27it/s]
Epoch [33/50], Training Loss: 3.9184, Validation Loss: 4.2740
100%|██████████| 56/56 [00:16<00:00,  3.33it/s]
Epoch [34/50], Training Loss: 3.8989, Validation Loss: 4.2611
100%|██████████| 56/56 [00:17<00:00,  3.29it/s]
Epoch [35/50], Training Loss: 3.8814, Validation Loss: 4.2719
100%|██████████| 56/56 [00:17<00:00,  3.26it/s]
Epoch [36/50], Training Loss: 3.8692, Validation Loss: 4.2481
100%|██████████| 56/56 [00:16<00:00,  3.31it/s]
Epoch [37/50], Training Loss: 3.8515, Validation Loss: 4.2858
100%|██████████| 56/56 [00:17<00:00,  3.28it/s]
Epoch [38/50], Training Loss: 3.8296, Validation Loss: 4.2859
100%|██████████| 56/56 [00:16<00:00,  3.32it/s]
Epoch [39/50], Training Loss: 3.8147, Validation Loss: 4.2677
100%|██████████| 56/56 [00:16<00:00,  3.35it/s]
Epoch [40/50], Training Loss: 3.7985, Validation Loss: 4.2578
100%|██████████| 56/56 [00:17<00:00,  3.29it/s]
Epoch [41/50], Training Loss: 3.7788, Validation Loss: 4.2309
100%|██████████| 56/56 [00:16<00:00,  3.29it/s]
Epoch [42/50], Training Loss: 3.7597, Validation Loss: 4.2310
100%|██████████| 56/56 [00:16<00:00,  3.31it/s]
Epoch [43/50], Training Loss: 3.7481, Validation Loss: 4.2214
100%|██████████| 56/56 [00:16<00:00,  3.32it/s]
Epoch [44/50], Training Loss: 3.7472, Validation Loss: 4.2120
100%|██████████| 56/56 [00:16<00:00,  3.42it/s]
Epoch [45/50], Training Loss: 3.7112, Validation Loss: 4.2030
100%|██████████| 56/56 [00:16<00:00,  3.34it/s]
Epoch [46/50], Training Loss: 3.7033, Validation Loss: 4.2204
100%|██████████| 56/56 [00:16<00:00,  3.29it/s]
Epoch [47/50], Training Loss: 3.6943, Validation Loss: 4.2176
100%|██████████| 56/56 [00:16<00:00,  3.34it/s]
Epoch [48/50], Training Loss: 3.6706, Validation Loss: 4.1995
100%|██████████| 56/56 [00:16<00:00,  3.30it/s]
Epoch [49/50], Training Loss: 3.6620, Validation Loss: 4.1919
100%|██████████| 56/56 [00:17<00:00,  3.29it/s]
Epoch [50/50], Training Loss: 3.6533, Validation Loss: 4.2123
```

```
checkpoint = torch.load('decoder-50.ckpt')
# Extract the training and validation losses
train_losses = checkpoint['train_losses']
val_losses = checkpoint['val_losses']

# Draw straight line across best model
plt.axvline(x=30, color='gray', linestyle='--')

# Plot the training and validation losses
plt.plot(train_losses, label='Training Loss')
plt.plot(val_losses, label='Validation Loss')
plt.xlabel('Epoch')
```

```
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
def load_image(filename, transform=None):
    img_path = os.path.join(IMAGE_DIR, filename)
    with open(img_path, 'rb') as f:
            image = Image.open(f).convert('RGB')
    if transform is not None:
        image = transform(image).unsqueeze(0)
    return image
```

4 Generate predictions on test data [8 marks]

Display 5 sample test images containing different objects, along with your model's generated captions and all the reference captions for each.

> Remember that everything **displayed** in the submitted notebook and .html file will be marked, so be sure to run all relevant cells.

```
def decode_caption(caption, vocab):
    """
    Decode a caption from a list of integers to a sentence.
    """
    sentence = []
    for word_id in caption:
        word = vocab.idx2word[word_id]
        if word == '<end>':
            break
        sentence.append(word)
    return ' '.join(sentence)

dataset_test = COCO_Features(
    df=test_df,
    vocab=trainval_vocab,
    features=test_feature_map,
)
# Create a data loader for the testing data
test_loader = DataLoader(dataset_test, batch_size=1, shuffle=False, collate_fn=caption_collate_fn)
checkpoint = torch.load('decoder-40.ckpt')
decoder.load_state_dict(checkpoint["decoder_state_dict"])
```

```
<All keys matched successfully>
```

```
decoder.eval()

img_counter=0
# Loop over the testing data and generate captions for each image
for i, (features, captions, lengths) in enumerate(test_loader):
```

```
    if (i+1) % 150==0:
        img_counter+=1

        # Generate captions using the sample function
        generated_captions = decoder.sample(features)

        # Convert the generated captions and reference captions to strings
        generated_captions = [decode_caption(cap.squeeze().tolist(), trainval_vocab) for cap in generated_captions]
        reference_captions = [decode_caption(cap.tolist(), trainval_vocab) for cap in captions]

        # Display the image and captions
        img = Image.open(IMAGE_DIR+test_df['file_name'].iloc[i])
        plt.imshow(np.asarray(img))
        plt.axis('off')
        plt.show()

        print("Generated caption:")
        for gen in generated_captions:
            print(gen)

        print("\nReference captions:")
        ref_caps = list(new_file.loc[new_file['file_name'] == test_df['file_name'].iloc[i]]['clean_caption'])
        for ref in ref_caps:
            print(ref)

        if img_counter==5:
            break
```

```
Generated caption:
street with sign the of <unk> <unk> a

Reference captions:
busy street at night with cars and pedestrians
a person pushing a baby carriage and walking a dog on a street full of cars
a street at night with a grey dog and vehicles
a person is pushing a stroller across the street with a dog
a car stopped with a person walking across the street
```

```
Generated caption:
young woman a on of in of with and on of

Reference captions:
a man in a suit riding a motorcycle
a man on a red motorcycle driving down the street
a person riding a red scooter on a narrow path
and elderly man riding a motorcycle in the street wearing a helmet
a man in a blue suit and helmet rides a motorcycle with his bag hanging on the front
```


```
Generated caption:
```

```
bus down street a on street a street <pad>
```

5 Caption evaluation using BLEU score [10 marks]

There are different methods for measuring the performance of image to text models. We will evaluate our model by measuring the text similarity between the generated caption and the reference captions, using two commonly used methods. Ther first method is known as *Bilingual Evaluation Understudy (BLEU)*.

> 5.1 Average BLEU score on all data (5 marks)

> 5.2 Examplaire high and low score BLEU score samples (5 marks, at least two)

5.1 Average BLEU score on all data (5 marks)

One common way of comparing a generated text to a reference text is using BLEU. This article gives a good intuition to how the BLEU score is computed: https://machinelearningmastery.com/calculate-bleu-score-for-text-python/, and you may find an implementation online to use. One option is the NLTK implementation `nltk.translate.bleu_score` here: https://www.nltk.org/api/nltk.translate.bleu_score.html

> **Tip:** BLEU scores can be weighted by ith-gram. Check that your scores make sense; and feel free to use a weighting that best matches the data. We will not be looking for specific score ranges; rather we will check that the scores are reasonable and meaningful given the captions.

Write the code to evaluate the trained model on the complete test set and calculate the BLEU score using the predictions, compared against all five references captions.

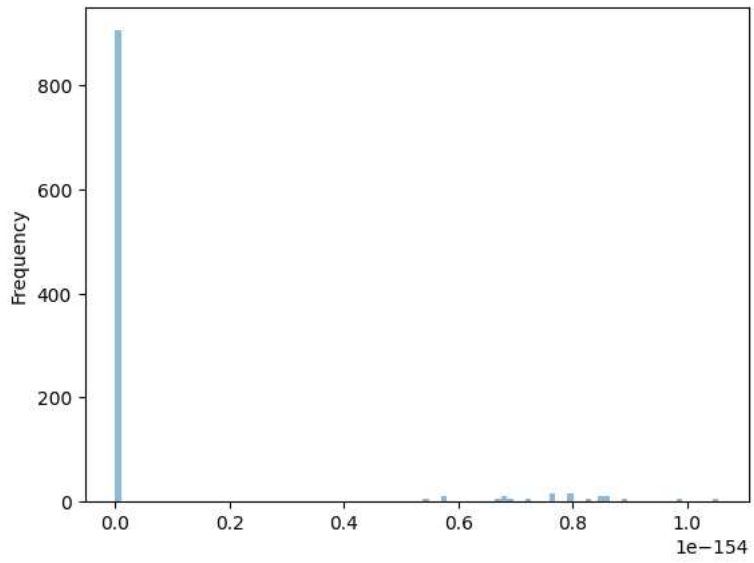
Display a histogram of the distribution of scores over the test set.

```
from nltk.translate.bleu_score import sentence_bleu
stats = pd.DataFrame(columns=['ref','preds','bleu','cos_sim'])

# Loop over the testing data and generate captions for each image
for i, (features, captions, lengths) in enumerate(tqdm(test_loader)):
      # Generate captions using the sample function
    generated_captions = decoder.sample(features)

    # Convert the generated captions and reference captions to strings
    generated_captions = [decode_caption(cap.squeeze().tolist(), trainval_vocab) for cap in generated_captions]
    generated_captions = generated_captions[0].split()
    ref_caps = list(new_file.loc[new_file['file_name'] == test_df['file_name'].iloc[i]]['clean_caption'])
    ref_caps = [sentence.split() for sentence in ref_caps]

    bleu_score = sentence_bleu(ref_caps, generated_captions)

    stats.loc[len(stats)] = [[' '.join(words) for words in ref_caps], ' '.join(generated_captions), bleu_score, np.nan]
```

```
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
 90%|[redacted] | 915/1015 [00:50<00:05, 19.98it/s]/usr/local/lib/python3.10/dist-packages/nltk/translate/bleu_score.py:552: UserWarn
The hypothesis contains 0 counts of 2-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
/usr/local/lib/python3.10/dist-packages/nltk/translate/bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 3-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
/usr/local/lib/python3.10/dist-packages/nltk/translate/bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 4-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
/usr/local/lib/python3.10/dist-packages/nltk/translate/bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 2-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
```

```
print("Average Cosine similarity score:", stats['bleu'].mean())
ax = stats['bleu'].plot.hist(bins=100, alpha=0.5)
```

```
Average Cosine similarity score: 8.390708471563783e-156
```



5.2 Examplaire high and low score BLEU score samples (5 marks)

Find one sample with high BLEU score and one with a low score, and display the model's predicted sentences, the BLEU scores, and the 5 reference captions.

```
best_score = stats.sort_values('bleu', ascending=False).iloc[0]
print ("Predicted caption: ",best_score['preds'])
best_score_ref = best_score['ref']
print ()
print ("Reference captions : -")
for sentence in best_score_ref:
    print (sentence)
```

```
Predicted caption:  man on motorcycle a on of street a

Reference captions : -
person wearing helmet sitting on motorcycle stopped in roadway
a man on a motorcycle stopped in traffic
a male on an orange red black and white motorcycle
there is a man riding a orange motorcycle on the street
man on the back of a black and orange motorcycle in a city
```

```
worst_score = stats.sort_values('bleu').iloc[0]
```

```
worst_score = stats.sort_values( 'bleu' ).iloc[0]
print ("Predicted caption: ",worst_score['preds'])
worst_score_ref = worst_score['ref']
print ()
print ("Reference captions : -")
for sentence in worst_score_ref:
    print (sentence)
```

```
 Predicted caption:  large of are in of with and <unk>

 Reference captions : -
 a train sitting on tracks next to a loading station
 a transit train is on the tracks directly outside the building
 a light blue train is pulled up beside a building
 a blue passenger train traveling through a train station
 a blue train sitting at a train station
```

6 Caption evaluation using cosine similarity [12 marks]

> 6.1 Cosine similarity (6 marks)

> 6.2 Cosine similarity examples (6 marks)

6.1 Cosine similarity (6 marks)

The cosine similarity measures the cosine of the angle between two vectors in n-dimensional space. The smaller the angle, the greater the similarity.

To use the cosine similarity to measure the similarity between the generated caption and the reference captions:

- Find the embedding vector of each word in the caption
- Compute the average vector for each caption
- Compute the cosine similarity score between the average vector of the generated caption and average vector of each reference caption
- Compute the average of these scores

Calculate the cosine similarity using the model's predictions over the whole test set.

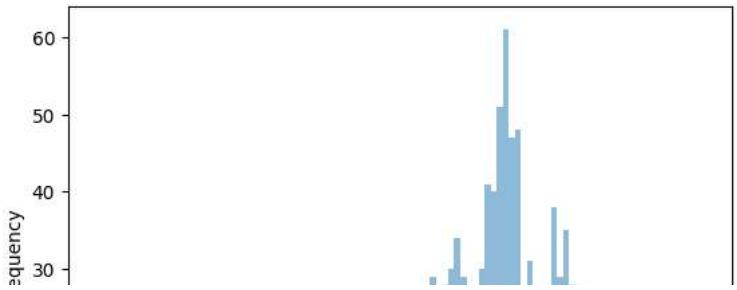
Display a histogram of the distribution of scores over the test set.

```
from numpy.linalg import norm

cos = torch.nn.CosineSimilarity(dim=0)
for i, (features, captions, lengths) in enumerate(tqdm(test_loader)):

    # Generate captions using the sample function
    generated_captions = decoder.sample(features)

    # Obtain the embedding vector for each word in the generated caption
    generated_caption = generated_captions[0].unsqueeze(0)
    generated_embedding_matrix = decoder.embed(generated_caption)
    generated_embedding = generated_embedding_matrix.detach().squeeze(0).mean(dim=0)

    # Obtain the embedding vector for each word in the reference caption
    reference_caption = captions[0].unsqueeze(0)
    reference_embedding_matrix = decoder.embed(reference_caption)
    reference_embedding = reference_embedding_matrix.detach().squeeze(0).mean(dim=0)
    cosine = cos(generated_embedding,reference_embedding)
    stats.at[i, 'cos_sim'] = cosine.item()
```

```
 100%|██████████| 1015/1015 [00:41<00:00, 24.51it/s]
```

```
print("Average Cosine similarity score:", stats['cos_sim'].mean())
ax = stats['cos_sim'].plot.hist(bins=100, alpha=0.5)
```

```
Average Cosine similarity score: 0.693587259706018
```



6.2 Cosine similarity examples (6 marks)

Find one sample with high cosine similarity score and one with a low score, and display the model's predicted sentences, the cosine similarity scores, and the 5 reference captions.

```
best_score = stats.sort_values('cos_sim', ascending=False).iloc[0]
print ("Predicted caption: ",best_score['preds'])
best_score_ref = best_score['ref']
print ()
print ("Reference captions : -")
for sentence in best_score_ref:
    print (sentence)
```

```
 Predicted caption:  city with and <unk> on <unk> a

 Reference captions : -
 a traffic light on a busy urban street in the snow
 a large building and some cars on a street
 cars are traveling on a road where snow has cleared to the side
 a car driving in the city street at night
 a street with cars on it and snow and night
```

```
worst_score = stats.sort_values('cos_sim').iloc[0]
print ("Predicted caption: ",worst_score['preds'])
worst_score_ref = worst_score['ref']
print ()
print ("Reference captions : -")
for sentence in worst_score_ref:
    print (sentence)
```

```
 Predicted caption:  woman a on of in of with and on

 Reference captions : -
 woman in swim suit holding parasol on sunny day
 a woman posing for the camera holding a pink open umbrella and wearing a bright floral ruched bathing suit by a life guard stand with la
 a woman in a floral swimsuit holds a pink umbrella
 a woman with an umbrella near the sea
 a girl in a bathing suit with a pink umbrella
```

7 Comparing BLEU and Cosine similarity [16 marks]

> 7.1 Test set distribution of scores (6 marks)

> 7.2 Analysis of individual examples (10 marks)

7.1 Test set distribution of scores (6 marks)

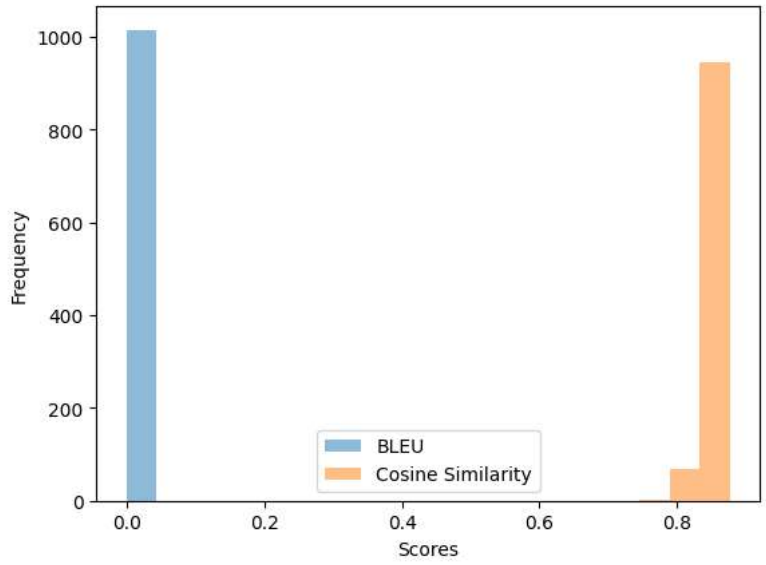
Compare the model's performance on the test set evaluated using BLEU and cosine similarity and discuss some weaknesses and strengths of each method (explain in words, in a text box below).

Please note, to compare the average test scores, you need to rescale the Cosine similarity scores [-1 to 1] to match the range of BLEU method [0.0 - 1.0].

```
rescaled_cos_sim = []
i = 0
for score in stats['cos_sim']:
  rescaled_cos_sim.append((score + 1) / 2)
  i = i +1
stats['rescaled_cos_sim'] = rescaled_cos_sim
```

```
scores_df = pd.DataFrame({
    'BLEU': stats['bleu'],
    'Cosine Similarity': stats['rescaled_cos_sim']
})


# Plot histogram of scores
ax = scores_df.plot.hist(bins=20, alpha=0.5)
ax.set_xlabel('Scores')
ax.set_ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```



7.2 Analysis of individual examples (10 marks)

Find and display one example where both methods give similar scores and another example where they do not and discuss. Include both scores, predicted captions, and reference captions.

```
# Find example with similar scores
similar_scores_idx = (stats['bleu'] - stats['cos_sim']).abs().idxmin()
similar_scores_pred = stats['preds'].iloc[similar_scores_idx]
similar_scores_refs = stats['ref'].iloc[similar_scores_idx]
similar_scores_bleu = stats['bleu'].iloc[similar_scores_idx]
similar_scores_cos_sim = stats['cos_sim'].iloc[similar_scores_idx]

# Find example with different scores
different_scores_idx = (stats['bleu'] - stats['cos_sim']).abs().idxmax()
different_scores_pred = stats['preds'].iloc[different_scores_idx]
different_scores_refs = stats['ref'].iloc[different_scores_idx]
different_scores_bleu = stats['bleu'].iloc[different_scores_idx]
different_scores_cos_sim = stats['cos_sim'].iloc[different_scores_idx]

# Display the examples
print("Example with similar scores:")
print("Predicted Caption:", similar_scores_pred)
print("Reference Captions:")
for ref in similar_scores_refs:
    print(ref)
print("BLEU Score:", similar_scores_bleu)
print("Cosine Similarity:", similar_scores_cos_sim)

print("Example with different scores:")
print("Predicted Caption:", different_scores_pred)
print("Reference Captions:")
for ref in different_scores_refs:
    print(ref)
print("BLEU Score:", different_scores_bleu)
print("Cosine Similarity:", different_scores_cos_sim)
```

```
Example with similar scores:
Predicted Caption: woman a on of in of with and on
Reference Captions:
```