

---

# PairUp: Detecting Congruous Image and Text Pairings

---

**Maitree Samanta**

California State University East Bay, Hayward

Department of Computer Science

May 6<sup>th</sup>, 2022

## 1. ABSTRACT

Vision and language are two important aspects of human intelligence in understanding the world. Much research has been done separately in the domains of computer vision and language understanding. Less work has been done on integration of these two modalities. Most of the previous work is focused on the task of automatic image captioning, where computer algorithms create written descriptions for images. Recent progress in this field has come through the use of deep learning methods. This work focuses on the topic of Image-text pairing which attempts to measure the visual semantic similarity between a sentence and an image. The resulting PairUp system takes a deep learning generative approach to the task of measuring the similarity between an image and a text pairing. The PairUp system could serve as a verification or validation system for image captioning systems.

## 2. ACKNOWLEDGEMENTS

I would like to thank and express my gratitude to Professor **Lynne Grewe** for her continuous support and guidance. Her valuable and constructive suggestions and advice have helped me through all the stages of this project.

## 3. INTRODUCTION & MOTIVATION

The PairUp system addresses the issue of detecting congruent text and image pairings in a proof-of-concept system that uses computer vision and machine learning that applies semantic analysis in domain translation to subsequently perform kind-to-kind semantic analysis. Congruous image and text pairings are defined as text and images that are semantically similar and by this definition are inversely related to incongruous pairings.

One use of measuring the congruence of image and text pairings is for a verification system of an automatic captioning system or an automatic report creation system. The creation of good automatic report generation systems is of interest to the medical field. Having a validation system, like PairUp, would be invaluable to trigger potentially hazardous image/ text generation pairs.

This work describes the design and implementation of the PairUp system. It is a bi-modal deep learning based system that accepts an image and a text pair and determines if the image and text are congruous or not.

---

# PairUp: Detecting Congruous Image and Text Pairings

---

## 4. RELATED WORK

There are a number of related works in the areas of domain translation, hate speech/ message detection and sentiment analysis. This section focuses on the areas of domain translation and the textual semantic analysis as these relate most closely to the approach taken up by PairUp system.

### 4.1. Image to Text Synthesis/ Image Captioning

Image to Text synthesis or Image Captioning is the creation of text that describes an image. The authors of “Show and Tell” system [1] proposed a neural and probabilistic framework to generate descriptions from images using CNN and LSTM networks. The input image is fed to a CNN which generates a feature vector which is then provided as an input to the LSTM that outputs the probability for the next word. The main goal here is to maximize the probability of the correct description of the image. This paper sets the stage for the image captioning task with a visual encoder and language decoder framework. One major reason image caption generation is well suited to the encoder-decoder framework of machine translation is because it is analogous to “translating” an image to a sentence.

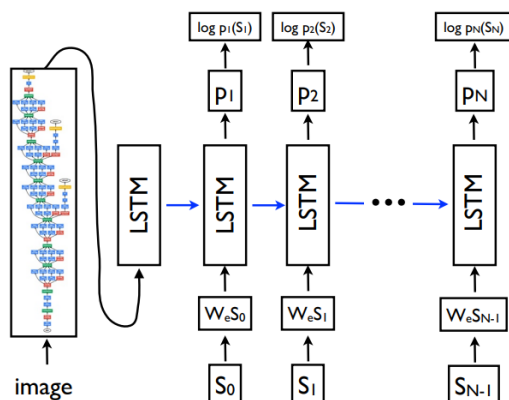


Figure 1: Performing image captioning task with a visual encoder and language decoder [1]

However, we see some criticism associated with the performance of a basic encoder-decoder architecture; mainly the potential loss of information due to the use of fixed-length vectors at the boundary of the encoder-decoder. This was first addressed in the paper [2] where the model had the flexibility to dynamically attend to the input content based on the query. In this approach, the basic encoder-decoder is extended by letting a model (soft-) search for a set of input words, or their annotations computed by an encoder, when generating each target word. This means the model does not have to encode the

---

# PairUp: Detecting Congruous Image and Text Pairings

---

sentence into a fixed-length vector, and also lets the model focus only on information relevant to the generation of the next target word.

Similar work followed from the domain of machine translation to image caption generation where the authors of “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention” [3] incorporated a form of attention mechanism that focused on relevant portions of the image to generate image captions.

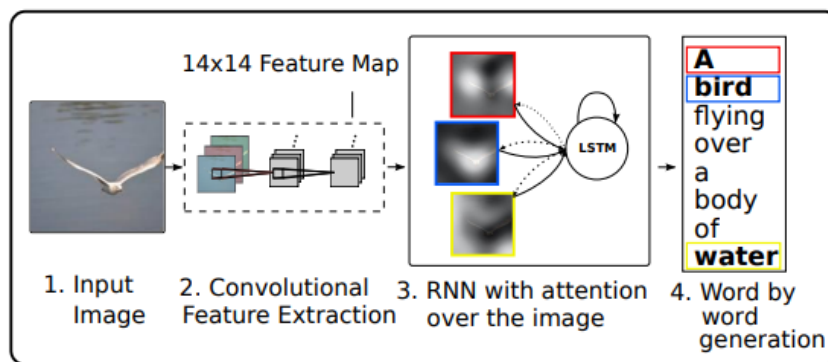


Figure 2: Model from [3] learns a words/image alignment

In the “Image Captioning with Semantic Attention” system [4], the authors use a CNN to extract a top-down visual feature and at the same time detect visual concepts (regions, objects, attributes, etc.). They employ a semantic attention model to combine the visual feature with visual concepts in a recurrent neural network that generates the image caption.

In paper [5], the authors proposed a novel adaptive attention model with a visual sentinel. At each time step, the model decides whether to attend to the image (and if so, to which regions) or to the visual sentinel. The model decides whether to attend to the image and where, in order to extract meaningful information for sequential word generation.

As will be discussed in section 7, the approach taken by PairUp system closely follows as what is described in the paper [3] i.e. a CNN based encoder is used followed by a GRU decoder with local attention mechanism that places an emphasis on the most important pixels in the image to carry out the corresponding caption generation.

## 4.2. Semantic Textual Similarity

Semantic textual similarity determines how similar two pieces of texts are. Broadly text similarity can be broken into two definitions: semantic similarity (meaning) and lexical similarity [6]. Lexical similarity is a measure of the degree to which the sets of words from the two text passages are similar. Two sentences that would be lexically similar are “the dog ate the cat food” and “the cat ate the dog food”. These sentences obviously have very different meanings or semantic similarity. While research is active in both areas, and it is true that often semantically similar text passages are lexically similar it is not necessary to

---

## PairUp: Detecting Congruous Image and Text Pairings

---

be highly lexically similar. Consider the following two sentences “How old are you?” and “What is your age?” Both of these phrases are close semantically but are very different lexically.

It is also possible to consider semantic similarity relative to a task. Consider the need to create an automated Q&A system. Conceptually, one might consider sentences as semantically similar if they can serve as the same question or as meaningful responses to the same question. Figure 3 illustrates the idea.



Figure 3: Semantic Textual Similarity. Taken from Google AI Blog [8]

One generic metric for basic semantic similarity can be estimated by taking the average of all the word embeddings between the two sentences being compared and calculating the cosine between the resulting embeddings [7]. First words are mapped into a dimensional space with similar words mapped nearby as shown in Figure 4. There are a number of existing frameworks that have been created by Natural Language Processing experts including Word2Vec [9]. Words frequently found close together in a collection of training documents will be mapped to nearby locations.

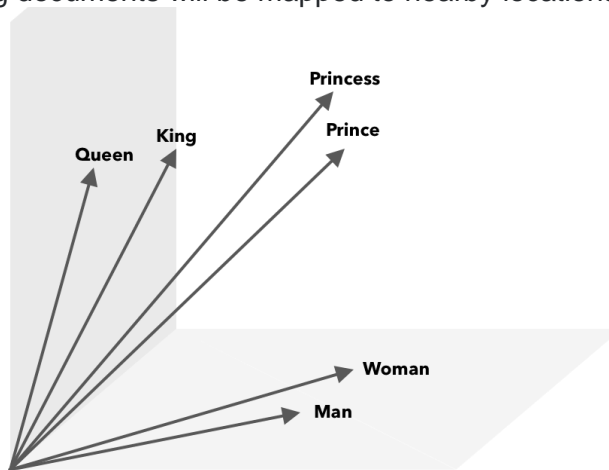


Figure 4: Words mapped to a dimensional space, similar words mapped nearby. (2-D here). Taken from <https://blog.marketmuse.com/topic-modeling-with-word2vec/>

---

## PairUp: Detecting Congruous Image and Text Pairings

---

Then metrics like the cosine of the angle between two vectors can be used to determine how similar or dissimilar the two vectors are. A maximum value of 1 indicates the angle is zero or the vectors are identical. Now this cosine so far is only between two words, but, in most cases two sentences or text passages are being compared. In this case, each sentence must be converted to a hyper dimensional vector and then the same cosine type metric could be applied. Many techniques have been proposed. Word2Vec accomplishes this with a combination of two techniques: CBOW (Continuous bag of words) and Skip-gram model. Details are given in [9].

Models like Word2vec [9] and GloVe [10] do not take word order into account as they are based on the bag-of-words method. In this sense they are closer to measuring lexical similarity rather than real meaning similarity. This is a major drawback since differences in word order can often completely alter the meaning of a sentence and sentence embeddings must capture these variations.

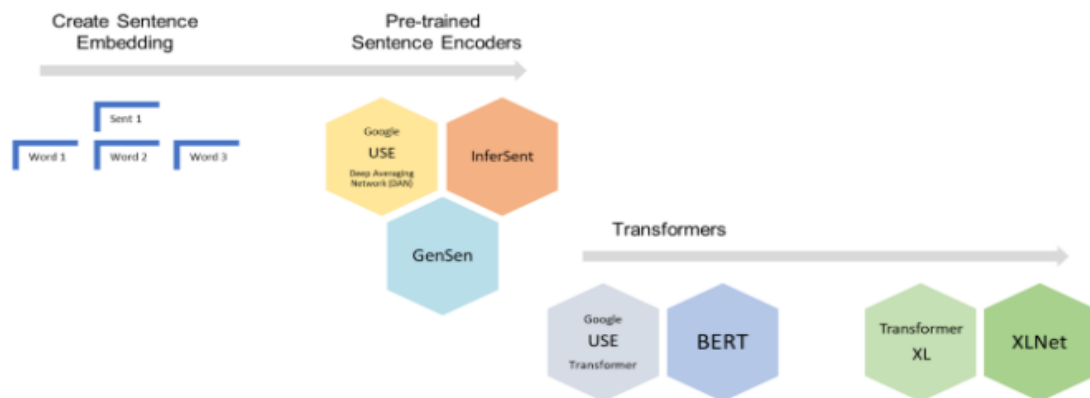


Figure 5: Evolution of sentence similarity models. Taken from [12]

Pre-trained sentence encoders are trained on a range of supervised and unsupervised tasks to capture as much universal semantic information as possible. However, they still face issues dealing with long-range dependencies and result in longer training times due to the sequential nature of the model architecture. Google's Universal Sentence Encoder [11] is a good example, which is available in a simpler version that uses a Deep Averaging Network (DAN) where input embeddings for words and bigrams are averaged together and passed through a feed-forward deep neural network. Figure 6 shows results of the semantic similarity scores given by this encoder for different example sentence pairings.

---

## PairUp: Detecting Congruous Image and Text Pairings

---

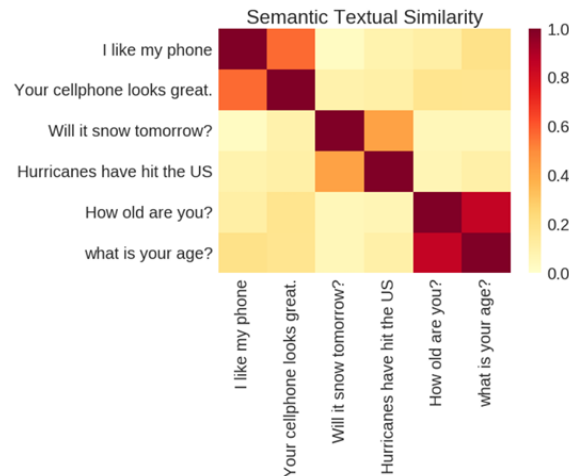


Figure 6: Example of Sentence similarity scores using embeddings from the universal sentence encoder [8]

In [13] the research addresses the potential deficiency of RNN models in which they cannot capture long-range dependencies and are also slow in operation due to sequential processing. This is solved through the creation of the Transformer architecture offering parallelism (see Figure 7). The concept of self-attention allows the model to look at other words in the input sequence to get a better understanding of a certain word in the sequence. The first use of the transformer [13] removed the need for RNNs through the use of three key components: Positional Encoding, Self-attention and Multi-head attention. Positional encoding replaced the key advantage of RNNs in NLP — the ability to consider the order of a sequence (they were recurrent). It worked by adding a set of varying sine wave activations to each input embedding based on position. Self-attention is where the attention mechanism is applied between a word and all of the other words in its own context (sentence/paragraph). This is different from vanilla attention which specifically focused on attention between encoders and decoders. Multi-head attention can be seen as several parallel attention mechanisms working together. Using several attention heads allowed the representation of several sets of relationships (rather than a single set).

# PairUp: Detecting Congruous Image and Text Pairings

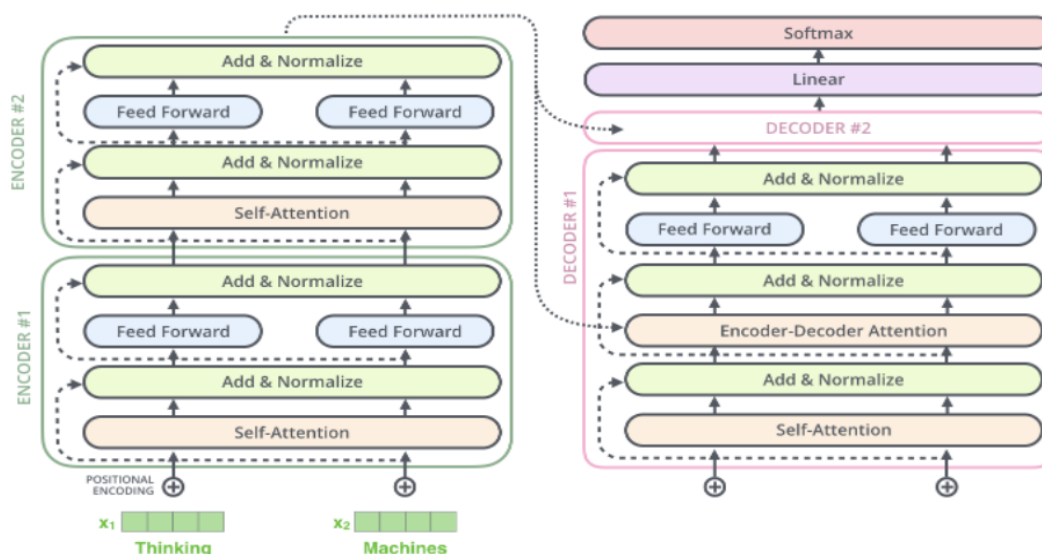


Figure 7: The Transformer architecture. Taken from “The Illustrated Transformer” by Jay Alammar [14]

With transformer models, it is possible to use the same ‘core’ of a model and simply swap the last few layers for different use cases (without retraining the core). This new property resulted in the rise of pre-trained models for NLP. Pre-trained transformer models are trained on vast amounts of training data. One of the most widely used of these pre-trained models is Bidirectional Encoder Representations from Transformers (BERT) [15]. It is a transformer-based model based on the encoder portion which can be trained bi-directionally and can have a deeper sense of language context and flow than single-direction language models. BERT allows the same pre-trained model to successfully tackle a broad set of NLP tasks including semantic textual similarity operations.

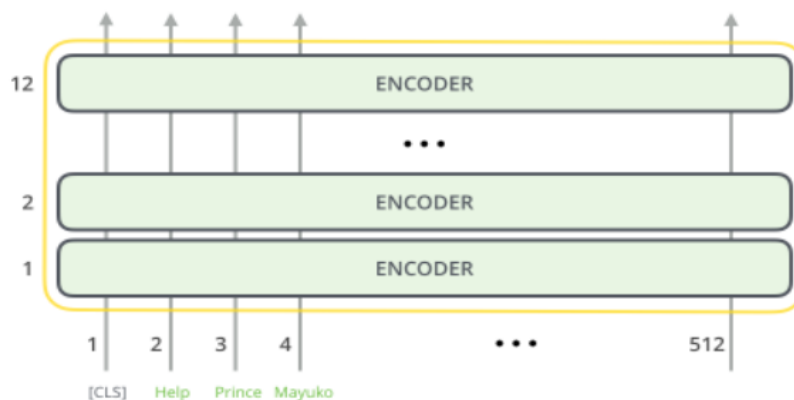


Figure 8: BERT Architecture, Taken from “The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)” by Jay Alammar [16]

# PairUp: Detecting Congruous Image and Text Pairings

There are two steps in the BERT framework: pre-training and fine-tuning. During pre-training, the model is trained on unlabeled data over different pre-training tasks. For fine-tuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks. Each downstream task has separate fine-tuned models, even though they are initialized with the same pre-trained parameters.

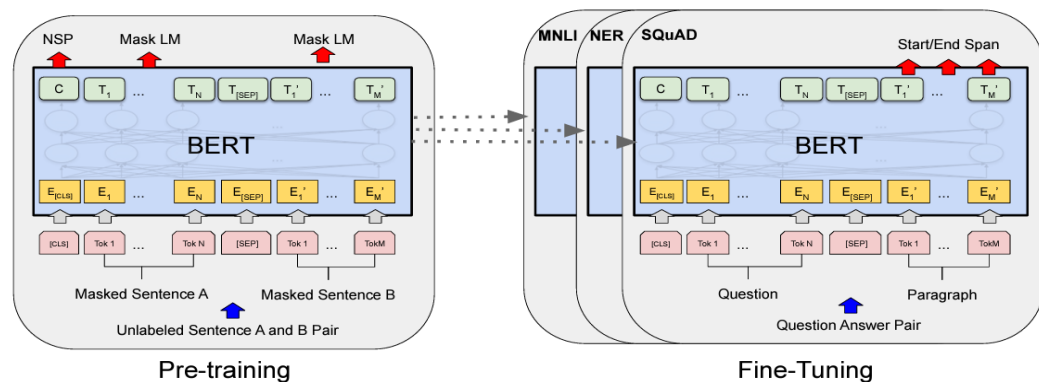


Figure 9: Pre-training and fine-tuning procedures for BERT [15]

When BERT is used for sentence similarity tasks, it suffers from a fundamental issue when building sentence vectors since Transformers work using word or token-level embeddings, not sentence-level embeddings. BERT uses a cross-encoder structure which requires that both sentences are fed into the network; a classification head is added to the top of BERT which is used to output a similarity score as shown in Figure 10.

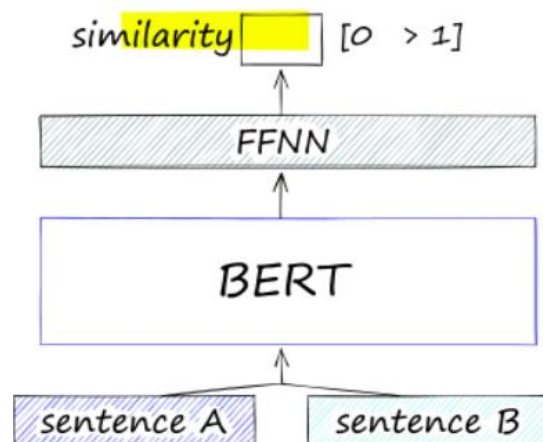


Figure 10: The BERT cross-encoder architecture consists of a BERT model which consumes sentences A and B. Both are processed in the same sequence, separated by a [SEP] token. All of this is followed by a feedforward NN classifier that outputs a similarity score. Taken from NLP for Semantic Search [17]



---

## PairUp: Detecting Congruous Image and Text Pairings

---

The cross-encoder network does produce very accurate similarity scores, but it's not scalable. If we wanted to perform a similarity search through a small 100K sentence dataset, we would need to complete the cross-encoder inference computation 100K times. Ideally, we need to pre-compute sentence vectors that can be stored and then used whenever required. If these vector representations are good, all we need to do is calculate the cosine similarity between each. With the original BERT (and other transformers), we can build a sentence embedding by averaging the values across all token embeddings output by BERT (if we input 512 tokens, we output 512 embeddings). These sentence embeddings can be stored and compared much faster, shifting search times from 65 hours to around 5 seconds. However, the accuracy is not good, and is worse than using averaged GloVe embeddings. The solution to this lack of an accurate model with reasonable latency was designed by Nils Reimers and Iryna Gurevych in 2019 with the introduction of **sentence-BERT (SBERT)** [18] and the sentence-transformers library. Reimers and Gurevych demonstrated the dramatic speed increase in 2019. Finding the most similar sentence pair from 10K sentences took 65 hours with BERT. With SBERT, embeddings are created in ~5 seconds and compared with cosine similarity in ~0.01 seconds. Sentence-BERT (SBERT) is a modification of the pre-trained BERT network that uses Siamese and triplet network structures to derive semantically meaningful sentence embeddings that can be compared using cosine-similarity.

Unlike BERT, SBERT is fine-tuned on sentence pairs using a Siamese architecture. We can think of this as having two identical BERTs in parallel that share the exact same network weights as shown in Figure 11. In reality, we are using a single BERT model. However, because we process sentence A followed by sentence B as pairs during training, it is easier to think of this as two models with tied weights.

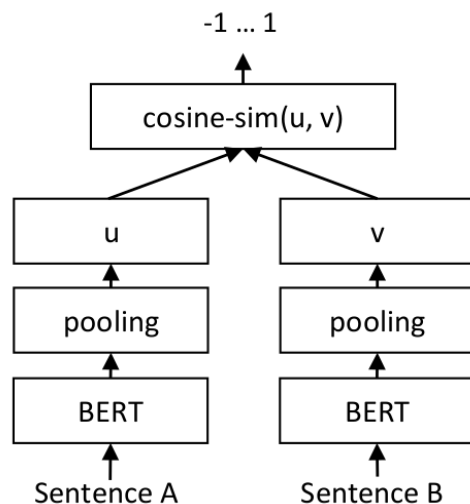


Figure 11: SBERT architecture at inference, for example, to compute similarity scores. This architecture is also used with the regression objective function. Taken NLP for Semantic Search [17]

---

## PairUp: Detecting Congruous Image and Text Pairings

---

As will be discussed in section 7, the approach taken by PairUp system is mainly based on the sentence-BERT (SBERT) [18] where one of the input sentences would come from the output of the image captioning system module and the other one would come from the user input. Cosine similarity metrics would be used to determine the similarity scores of the two sentences.

### 5. SYSTEM OVERVIEW

There can be different approaches taken to do any kind of multi-modal analysis, in PairUp case the comparison of an image and text for semantic similarity. There are three basic approaches, the first two, Visual Driver Based and Textual Driver Based work by performing a modal transformation, meaning one of the modes is transformed into the other's domain. In the case of a **Visual Driver Based Semantic Similarity** the input text is used to synthesize a related image and then compared to the input image for similarity. In the case of **Textual Driver Based Semantic Similarity** the input image is converted to a textual representation and then compared to the input text for semantic similarity. The third kind of multi-modal similarity approach does not explicitly involve any modal transformation.

The modal transformation approaches yield information that is more easily explainable. ML explainability is a desirable feature in systems and is the reason why PairUp takes this approach. Specifically a Textual Driver based approach is taken as shown in Figure 12. The **Image to Text Synthesis** module accepts the input image and generates a relevant text caption. The output caption text is then passed to the next stage which comprises a **Textual Similarity Analyzer** module that compares the caption and input text to determine if the texts are congruous or not. The details of these modules are described in the following sections.

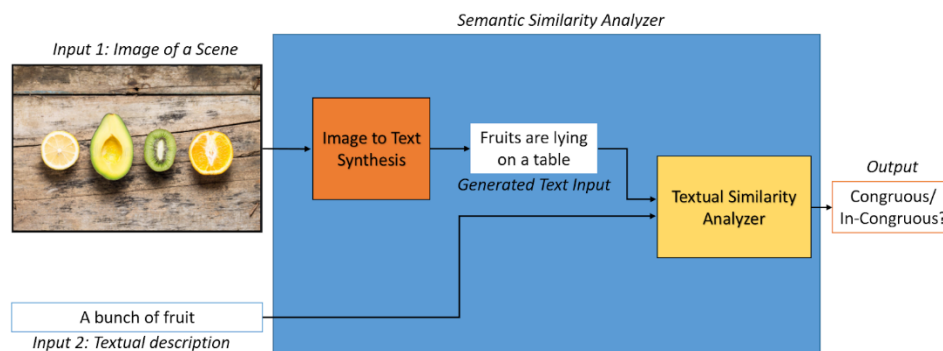


Figure 12: PairUp implementation featuring Textual Driver Based Semantic Similarity

---

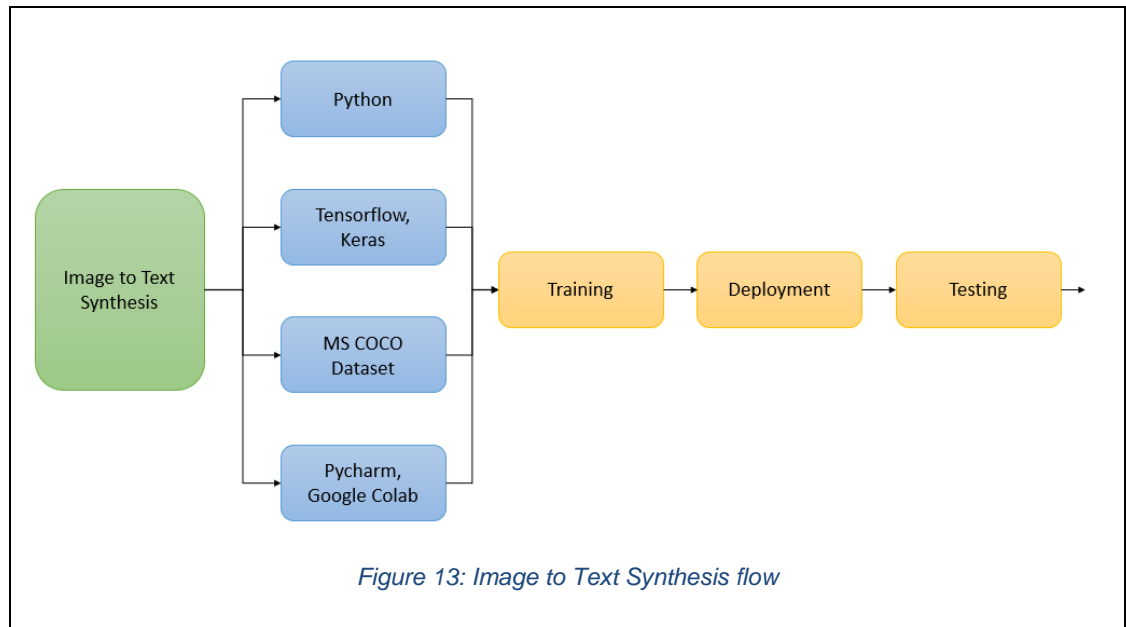
# PairUp: Detecting Congruous Image and Text Pairings

---

## 6. SOFTWARE & TOOLS

The following frameworks and software engineering tools were used in the design and implementation of PairUp. Figures 13 and 14 show where some of these frameworks are incorporated in PairUp.

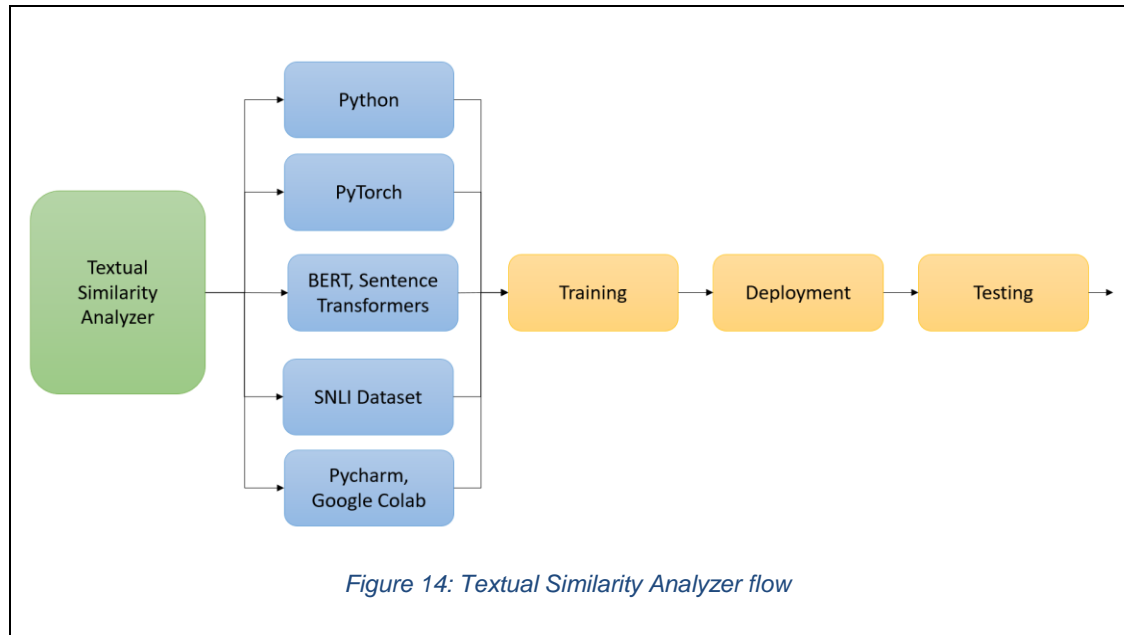
- TensorFlow, Keras framework, PyTorch
- Hugging Face library
- Sentence Transformer library
- Various Python libraries
- GitHub
- Discord
- Google Drive
- Google Colab



---

## PairUp: Detecting Congruous Image and Text Pairings

---



### 7. IMAGE TO TEXT SYNTHESIS

This section covers the **Image to Text Synthesis** subsystem of PairUp (see Figure 12). Image to text synthesis or conversion popularly known as image captioning task involves generating a textual description for a given image. This requires methods from the computer vision field to understand the image content and a language model from the field of natural language processing to turn the understanding of the image into a natural language text phrase(s) or sentence(s).

Figure 15 shows a possible Image to Text Synthesis (captioning) system that uses an encoder-decoder scheme. The encoding is implemented with some form of a Convolutional Neural Network. As the decoder must produce a sequence of words one type of architecture suited to sequences are Recurrent Neural Networks (RNN). In Figure 15, a Long Term Short Term (LSTM) RNN architecture is utilized. The encoder produces a hidden state "h" which is used by the decoder (LSTM) to recursively generate each word of the caption (synthesized text).

There lies a fundamental problem with this classical approach which is when the model is trying to generate the next word of the caption, this word is usually describing only a part of the image. It is unable to capture the essence of the entire input image. Using the whole representation of the image hidden state h to condition the generation of each word cannot efficiently produce different words for different parts of the image. This is exactly where an Attention mechanism is helpful. Moreover LSTMs tend to be computationally expensive to train and evaluate, so in practice, memory is limited to just

---

## PairUp: Detecting Congruous Image and Text Pairings

---

a few elements. Attention models can help address this problem by selecting the most relevant elements from an input image.

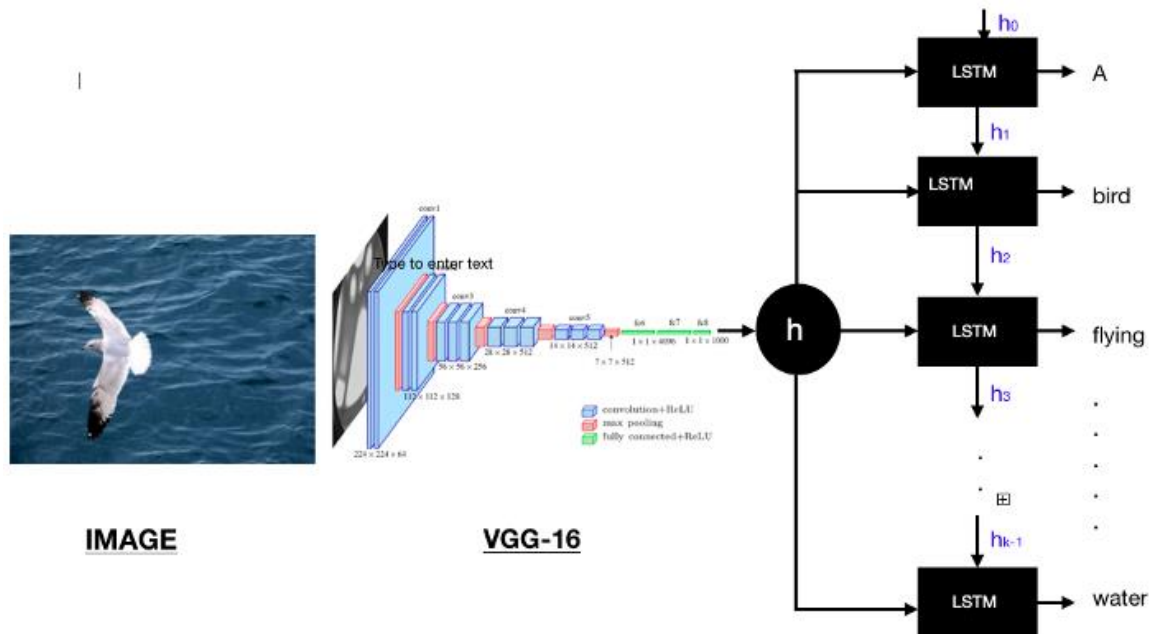


Figure 15: A classic image captioning model [19]

With an Attention mechanism, the image is first divided into  $n$  parts, and then computation of an image representation of each part  $h_1, \dots, h_n$  takes place. When the LSTM (or RNN) is generating a new word, the attention mechanism is focusing on the relevant part of the image, so the decoder only uses specific parts of the image. Figure 16 shows this modification of the “classic” model for image captioning that now has a new attention layer. The prediction of the new word of the caption works as follows: If the system has predicted  $i$  words, the hidden state of the LSTM is  $h_i$ . The system selects the “relevant” part of the image by using  $h_i$  as the context. Then, the output of the attention model  $z_{i+1}$ , which is the representation of the image filtered such that only the relevant parts of the image remains, is used as an input for the LSTM. Then, the LSTM predicts a new word and returns a new hidden state  $h_{i+1}$ .

# PairUp: Detecting Congruous Image and Text Pairings

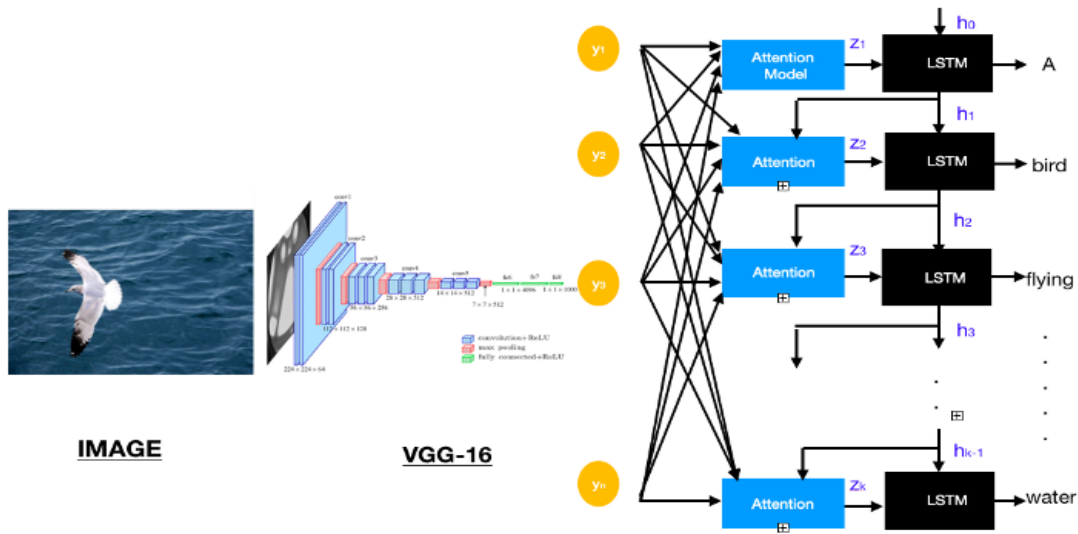


Figure 16: Image captioning model with LSTMs using Attention mechanism [19]

To understand what attention semantically means consider Figure 17 where a dog stands on a colorful skateboard with its tongue out. So, when we say the word “dog” it focuses only on the dog in the image, and when we say the word “tongue” the focus narrows in on the dog’s tongue in the image. Similarly, when we say “skateboard” the focus should be only on the skateboard in the image. This means “dog”, “tongue” and “skateboard” come from the different pixels in the image.

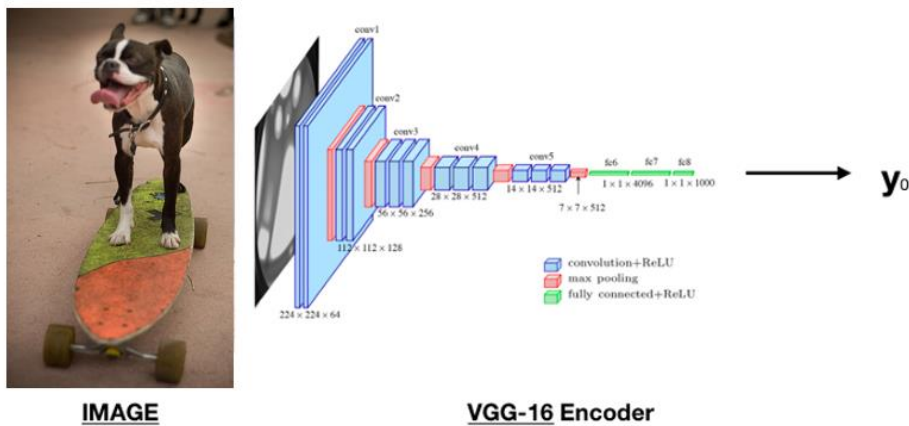


Figure 17: Encoder showing the encoded operation of input image [19]

---

## PairUp: Detecting Congruous Image and Text Pairings

---

The “pixel values” in different layers of a CNN (remember this serves as our encoder) do correlate to regions in the original image as illustrated in Figure 18. For example, the output of the 5<sup>th</sup> convolution layer of VGGNet is a  $14 \times 14 \times 512$  size feature map. This 5<sup>th</sup> convolution layer has  $14 \times 14$  pixels, and each pixel corresponding to processed information from the original image in an area of  $16 \times 16$  pixels. So information from any pixel (data point) in a CNN layer can be traced back to some sub-region of the original image.

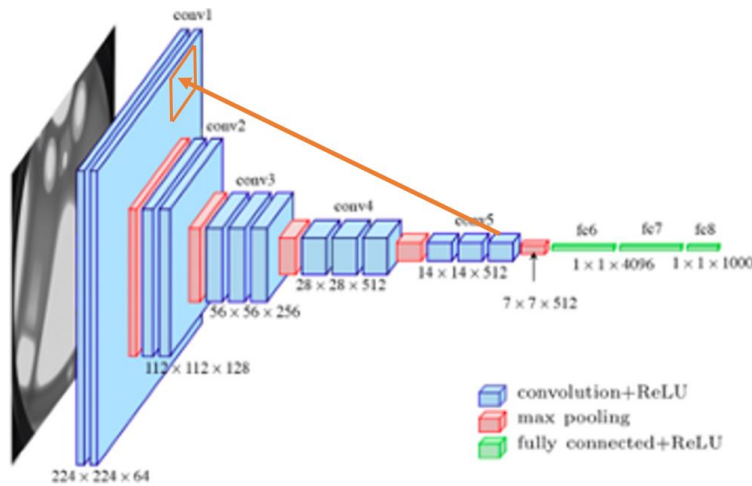


Figure 18: 5<sup>th</sup> Convolution layer showing the correspondence to the portion of the input image [19]

Figure 18 shows the 5<sup>th</sup> convolution block with  $14 \times 14$  (196) values that can be passed in at different time steps for the Attention mechanism to process as shown in Figure 19.

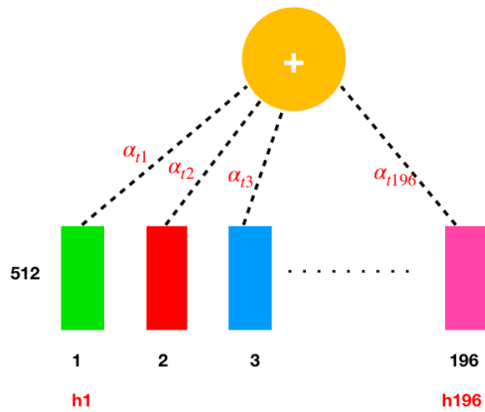


Figure 19: Model learning an attention over the various channels [19]

---

# PairUp: Detecting Congruous Image and Text Pairings

---

Attention mechanisms can be broadly classified into 2 types (see Figure 20):

1. **Global Attention** (Luong's Attention) [20]: Attention is placed on all source positions. It takes into consideration all encoder hidden states to derive the context vector ( $c(t)$ ). In order to calculate  $c(t)$ , we compute  $a(t)$  which is a variable length alignment vector. The alignment vector is derived by computing a similarity measure between  $h(t)$  and  $\bar{h}(s)$  where  $h(t)$  is the source hidden state while  $\bar{h}(s)$  is the target hidden state. Similar states in encoder and decoder are actually referring to the same meaning.
2. **Local Attention** (Bahdanau Attention) [21]: Attention is placed only on a few source positions. It focuses only on a small subset of the hidden states of the encoder per target word.

Both attention based models differ from the normal encoder-decoder architecture only in the decoding phase. These attention based methods differ in the way that they compute context vectors ( $c(t)$ ) as shown in Figure 20. Global Attention is computationally very expensive and is impractical when translating for long sentences. In section 7.3, the discussion of the PairUp's use of a RNN Decoder with Gated Recurrent Units with local attention is described.

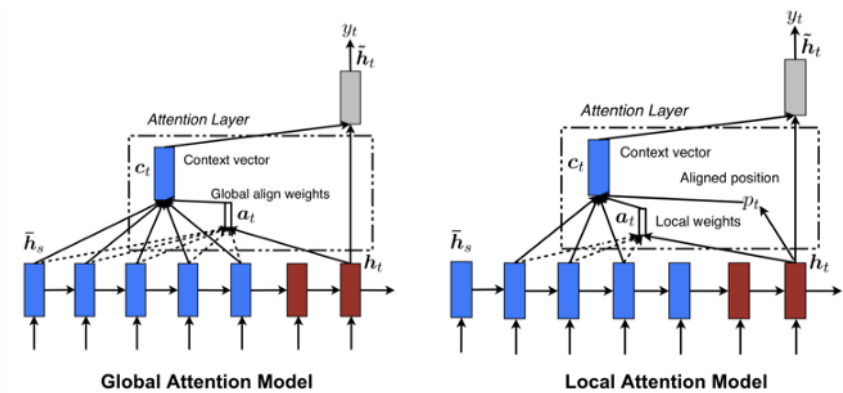


Figure 20: Global vs Local Attention model [19]

## 7.1. DATASET

There are many open source datasets available for the image captioning problem, like Flickr 8k (containing 8k images), MS COCO (containing 165k images) [22], etc. For our task, we have selected 20k images from the MS COCO dataset. This has been done so that the dataset size is manageable and can be saved and accessed easily on Google Drive without running into I/O issues. There is a Google Drive timeout issue that happens when a large number of files are placed inside a directory and accessed frequently. PairUp used 16k images for training and the remaining 4k for validation.



---

## PairUp: Detecting Congruous Image and Text Pairings

---

The captioning JSON file contains five annotations for each image. An example image from the dataset is shown in Figure 21 along with the JSON containing the five corresponding captions. While five captions is an arbitrary set number, it indicates that for an image there are many possible captions. What is important is that these captions represent the main meaning or semantic description of what is occurring in the image. In this sense the five captions are semantically similar.

The PairUp code in Table 1 downloads the entire dataset of images and annotations/ captions. Then it iterates over the annotations JSON file to group all captions together having the same image ID in preparation for PairUp training/validation.



Figure 21: An example image from COCO dataset and corresponding 5 annotations

```
"annotations": [  
  {"image_id": 203564,"id": 37,"caption": "A bicycle replica with a  
clock as the front wheel."},  
  {"image_id": 203564,"id": 181,"caption": "The bike has a clock as a  
tire."},  
  {"image_id": 203564,"id": 478,"caption": "A black metal bicycle with  
a clock inside the front wheel."},  
  {"image_id": 203564,"id": 6637,"caption": "A bicycle figurine in  
which the front wheel is replaced with a clock\n"},  
  {"image_id": 203564,"id": 6802,"caption": "A clock with the  
appearance of the wheel of a bicycle "}]
```

```
01 # Download caption annotation files  
02 annotation_folder = '/annotations/'  
03 if not os.path.exists(os.path.abspath('.') + annotation_folder):
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
04  annotation_zip = tf.keras.utils.get_file('captions.zip',
05
cache_subdir=os.path.abspath('.'),
06
origin='http://images.cocodataset.org/annotations/annotations_trainval
2014.zip',
07                                     extract=True)
08  annotation_file =
09  os.path.dirname(annotation_zip)+'annotations/captions_train2014.json'
10  os.remove(annotation_zip)
11  annotation_file = os.path.abspath('.') +
12  '/annotations/captions_train2014.json'
13  # Download image files
14  image_folder = '/train2014_20k/'
15  print(f"image_folder: {image_folder}")
16
17  if not os.path.exists(os.path.abspath('.') + image_folder):
18      image_zip = tf.keras.utils.get_file('train2014.zip',
19
cache_subdir=os.path.abspath('.'),
20
origin='http://images.cocodataset.org/zips/train2014.zip',
21                                     extract=True)
22      PATH = os.path.dirname(image_zip) + image_folder
23      os.remove(image_zip)
24  else:
25      PATH = os.path.abspath('.') + image_folder
```

```
01  with open(annotation_file, 'r') as f:
02      annotations = json.load(f)
03
04  # Group all captions together having the same image ID.
05  image_path_to_caption = collections.defaultdict(list)
06  for val in annotations['annotations']:
07      caption = f"<start> {val['caption']} <end>"
08      image_path = PATH + 'COCO_train2014_' + '%012d.jpg' %
09      (val['image_id'])
10      image_path_to_caption[image_path].append(caption)
11
12  image_paths = list(image_path_to_caption.keys())
13  random.shuffle(image_paths)
14  # Select the first 20000 image_paths from the shuffled set.
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
15 # train_image_paths = image_paths[:6000]
16 train_image_paths = image_paths[:20_000]
17 print(len(train_image_paths))
18
19
20 from pathlib import Path
21
22 train_demo_image_paths = []
23 basepath =
Path('/content/drive/MyDrive/Capstone_Project/train2014_20k/')
24 print(f"base_path: {basepath}")
25
26 files_in_basepath = basepath.iterdir()
27 for item in files_in_basepath:
28     if item.is_file():
29         # print(item.name)
30         train_demo_image_paths.append(PATH + item.name)
31
32 print(len(train_demo_image_paths))
33 print(train_demo_image_paths[0])
34
35 train_captions = []
36 img_name_vector = []
37
38 # for image_path in train_image_paths:
39 for image_path in train_demo_image_paths:
40     caption_list = image_path_to_caption[image_path]
41     train_captions.extend(caption_list)
42     img_name_vector.extend([image_path] * len(caption_list))
```

Table 1: PairUp code to read in and organize data for training/ validation.

### 7.2. PREPROCESSING

PairUp uses InceptionV3 Convolutional Neural Network [\[23\]](#) to process each image translating it into features. The architecture of InceptionV3 is shown in Figure 22 and begins like most CNNs with a number of convolutional oriented layers followed by fully-connected (or dense) layers.

---

# PairUp: Detecting Congruous Image and Text Pairings

---

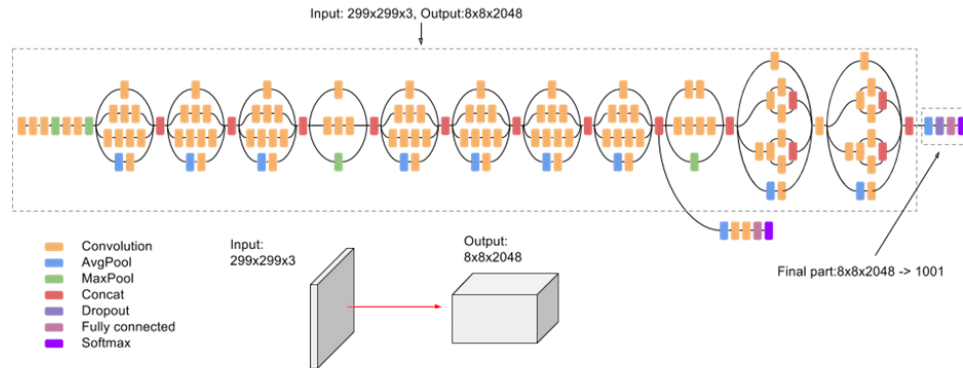


Figure 22: Inception V3 from <https://cloud.google.com/tpu/docs/inception-v3-advanced>

InceptionV3 is known to have relatively high accuracy rates for relatively fast performance. InceptionV3 is compared to other well-known architectures as shown in Figure 23.

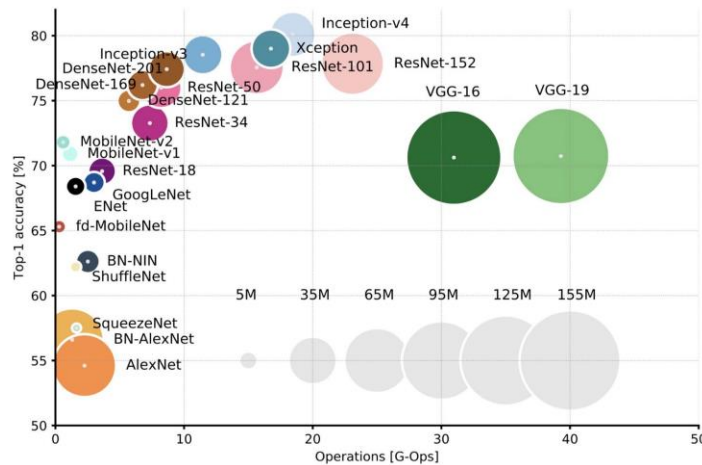


Figure 23: Comparison of various well known ML model architectures from <https://towardsdatascience.com/neural-network-architectures-156e5bad51ba>

While InceptionV3 rates a little lower in accuracy than InceptionV4, it is a smaller model as shown in Figure 24 and as a result and evidenced in Figure 23 runs faster, requiring less operations. For these reasons, InceptionV3 was selected as a good general purpose feature extraction model architecture.

---

## PairUp: Detecting Congruous Image and Text Pairings

---

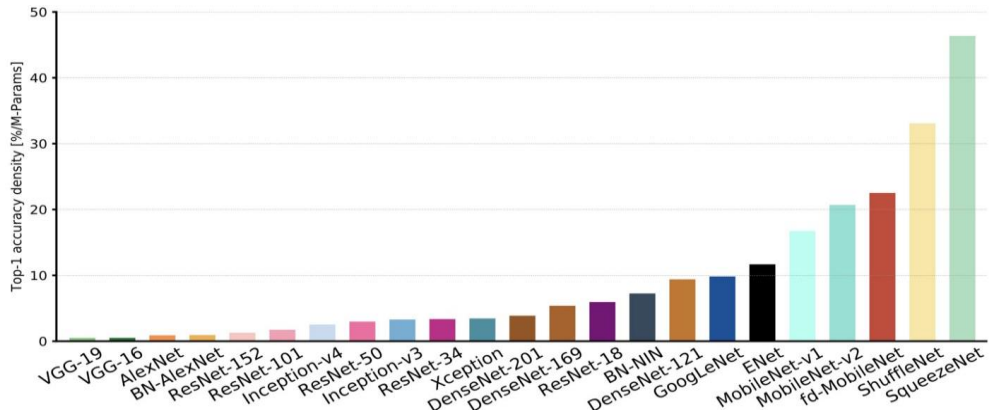


Figure 24: Comparison of %/M-Parameters of various well known ML model architectures from <https://culurciello.medium.com/analysis-of-deep-neural-networks-dcf398e71aee>

Table 2 shows the code using the Keras API to create PairUp's instance of the InceptionV3 model. The InceptionV3 model was pre-trained on the [ImageNet dataset](#) for classifying images. As shown in Table 2 after creating the model, an input image is passed through this feature extractor. The corresponding captions are transformed into integer sequences using the TextVectorization layer of PairUp via the steps described in Table 2.

After this preprocessing on the data, it is split in an 80/20 split randomly for training and testing.

```
# If include_top=True then the whole InceptionV3 model is downloaded.
# If
# include_top=False then only the convolutional part of the model is
# downloaded.

01 image_model = tf.keras.applications.InceptionV3(include_top=False,
weights='imagenet')
02
03 new_input = image_model.input
04 hidden_layer = image_model.layers[-1].output
05
06 image_features_extract_model = tf.keras.Model(new_input,
hidden_layer)

# We pre-process each image with InceptionV3 and cache the output to
disk.

07 from tqdm import tqdm
08
09 # Get unique images
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
10 encode_train = sorted(set(img_name_vector))
11
12 # Feel free to change batch_size according to your system
configuration
13 image_dataset = tf.data.Dataset.from_tensor_slices(encode_train)
14 image_dataset = image_dataset.map(
15     load_image, num_parallel_calls=tf.data.AUTOTUNE).batch(16)
16
17 # for img, path in image_dataset:
18 for img, path in tqdm(image_dataset):
19     batch_features = image_features_extract_model(img)
20     batch_features = tf.reshape(batch_features,
21                                (batch_features.shape[0], -1,
batch_features.shape[3]))
22
23     for bf, p in zip(batch_features, path):
24         path_of_feature = p.numpy().decode("utf-8")
25         np.save(path_of_feature, bf.numpy())
26
27 # We transform the text captions into integer sequences using the
28 # TextVectorization layer, with the following steps:
29 # We use "adapt" to iterate over all captions, split the captions into
30 # words, and compute a vocabulary of the top 5,000 words (to save
31 # memory).
32 # Tokenize all captions by mapping each word to its index in the
33 # vocabulary. # All output sequences will be padded to length 50.
34 # Create word-to-index and index-to-word mappings to display results.
35
36 caption_dataset =
37 tf.data.Dataset.from_tensor_slices(train_captions)
38
39 # We will override the default standardization of TextVectorization
40 # to preserve
41 # "<>" characters, so we preserve the tokens for the <start> and
42 # <end>.
43 def standardize(inputs):
44     inputs = tf.strings.lower(inputs)
45     return tf.strings.regex_replace(inputs,
46                                     r"!\"#$%&\(\)\*\+\.,-
47 /:;=?@\\[\\"^_`{|}~", "")
48
49 # Max word count for a caption.
50 max_length = 50
51 # Use the top 5000 words for a vocabulary.
52 vocabulary_size = 5000
53 tokenizer = tf.keras.layers.TextVectorization(
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
40     max_tokens=vocabulary_size,
41     standardize=standardize,
42     output_sequence_length=max_length)
43 # Learn the vocabulary from the caption data.
44 tokenizer.adapt(caption_dataset)
45
46
47 # Create the tokenized vectors
48
49 cap_vector = caption_dataset.map(lambda x: tokenizer(x))
50
51 # Create mappings for words to indices and indices to words.
52 word_to_index = tf.keras.layers.StringLookup(
53     mask_token="",
54     vocabulary=tokenizer.get_vocabulary())
55 index_to_word = tf.keras.layers.StringLookup(
56     mask_token="",
57     vocabulary=tokenizer.get_vocabulary(),
58     invert=True)
59
60 # Next, we split the dataset (image and captions) into 80:20 ratio
61 # i.e. [train:test]
62
63 img_to_cap_vector = collections.defaultdict(list)
64 for img, cap in zip(img_name_vector, cap_vector):
65     img_to_cap_vector[img].append(cap)
66
67 # Create training and validation sets using an 80-20 split
68 # randomly.
69 img_keys = list(img_to_cap_vector.keys())
70 random.shuffle(img_keys)
71 slice_index = int(len(img_keys)*0.8)
72 img_name_train_keys, img_name_val_keys = img_keys[:slice_index],
73 img_keys[slice_index:]
74
75 img_name_train = []
76 cap_train = []
77 for imgt in img_name_train_keys:
78     capt_len = len(img_to_cap_vector[imgt])
79     img_name_train.extend([imgt] * capt_len)
80     cap_train.extend(img_to_cap_vector[imgt])
81
82 img_name_val = []
83 cap_val = []
84 for imgv in img_name_val_keys:
```

# PairUp: Detecting Congruous Image and Text Pairings

```
80 capv_len = len(img_to_cap_vector[imgv])
81 img_name_val.extend([imgv] * capv_len)
82 cap_val.extend(img_to_cap_vector[imgv])
```

Table 2: PairUp Pre-processing code to create InceptionV3 feature extractor, process the images from dataset with it as well as process the corresponding captions into integer sequences, finally ending in the split of data from training & validation.

## 7.3. ENCODER-DECODER MODEL

A detailed diagram of PairUp's Image to Text synthesizer is shown in Figure 25 involving an Encoder-Decoder with Attention mechanism scheme. The Encoder consists of the InceptionV3 feature extractor used in conjunction with an attention mechanism which feeds into a decoder RNN made up of GRU.

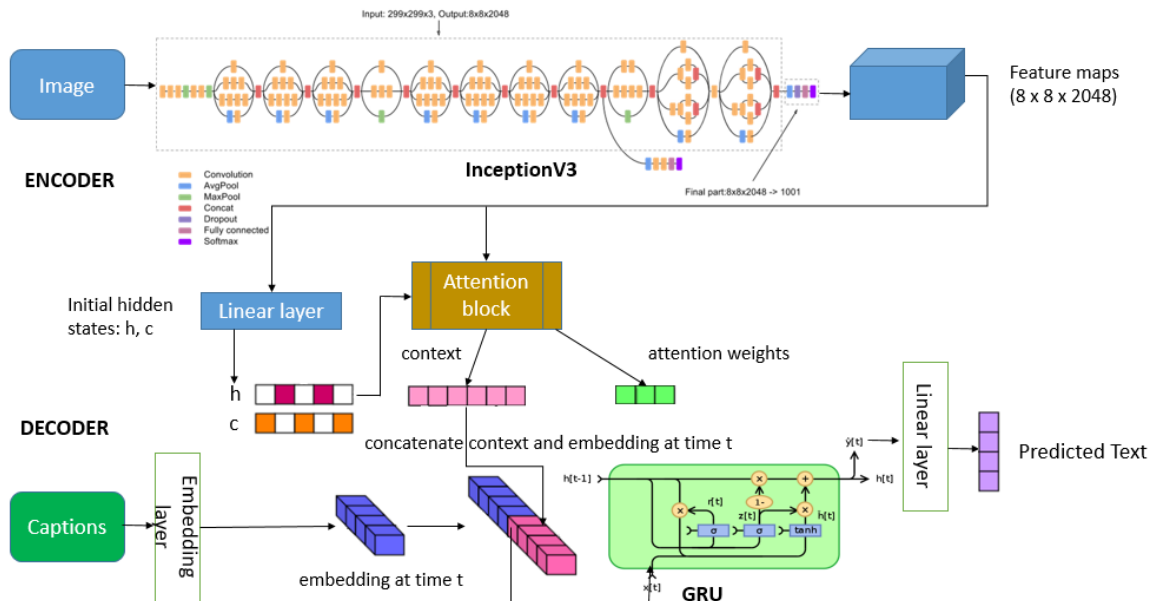


Figure 25: PairUp Encoder Decoder model with Attention mechanism

The CNN Encoder is defined as a single fully connected layer followed by ReLU activation. PairUp extracts the features from the lower convolutional layer of InceptionV3 giving us a vector of shape (8, 8, 2048) and then squashes it to (64, 2048). When the batch size is considered, the shape becomes (BATCH\_SIZE, 64, 2048). (Here, the embedding dimension is 2048). This process is achieved with the PairUp code shown in Table 3.



---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
01 class CNN_Encoder(tf.keras.Model):
02     # Since you have already extracted the features and dumped it
03     # This encoder passes those features through a Fully connected
layer
04     def __init__(self, embedding_dim):
05         super(CNN_Encoder, self).__init__()
06         # shape after fc == (batch_size, 64, embedding_dim)
07         self.fc = tf.keras.layers.Dense(embedding_dim)
08
09         @tf.function(input_signature=[tf.TensorSpec(shape=[None, 64,
features_shape], dtype=tf.float32)])
10         def call(self, x):
11             # print(x.shape)
12
13             x = self.fc(x)
14             x = tf.nn.relu(x)
15             return x
```

Table 3: PairUp code to flatten the output of the feature extraction preprocessing stage using a dense layer

An important piece of PairUp Image to Text Synthesizer is the use of an Attention mechanism (see Figure 25) and the Bahdanau attention mechanism [2] is applied. A key idea behind the Bahdanau mechanism is that it does not require a fixed-length vector for the text representation but instead uses a variable-length one. The **Bahdanau/ Local Attention** is applied as follows:

1. **Producing the Encoder Hidden States** - Encoder produces hidden states of each element in the input sequence.
2. **Calculating Attention Scores** between the previous decoder hidden state and each of the encoder's hidden states are calculated (Note: The last encoder hidden state can be used as the first hidden state in the decoder). The results of this step are what are called attention scores.
3. **Softmaxing the Attention Scores** - the attention scores for each encoder hidden state are combined and represented in a single vector and subsequently softmaxed (ranging from 0 to 1 in value)
4. **Calculating the Context Vector** - the encoder hidden states and their respective attention scores are multiplied to form what is called the context vector.
5. **Decoding the Output** - the context vector is concatenated with the previous decoder output and fed into the Decoder RNN for that time step along with the previous decoder hidden state to produce a new output.
6. The process (steps 2–5) **repeats** itself for each time step of the decoder until a token is produced or output is past the specified maximum length.

## PairUp: Detecting Congruous Image and Text Pairings

Figure 26 illustrates visually where the Bahdanau mechanism fits in the encoder/decoder scheme and illustrates the steps involved. Table 4 shows the basic code for the calculation of the context vector and attention weights using the Bahdanau mechanism.

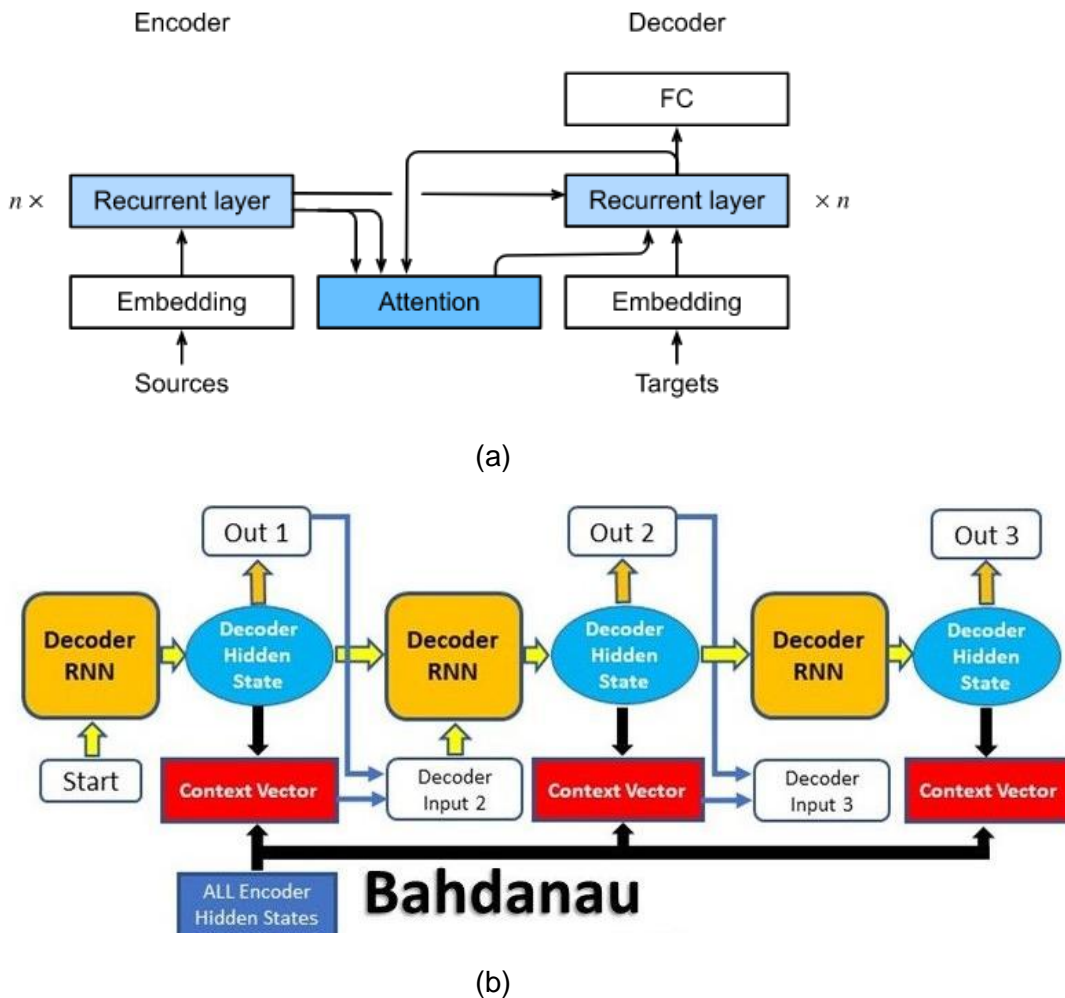


Figure 26: (a) Layers in an RNN encoder-decoder model with Bahdanau attention. Taken from [https://d2l.ai/chapter\\_attention-mechanisms/bahdanau-attention.html](https://d2l.ai/chapter_attention-mechanisms/bahdanau-attention.html) (b) steps in Bahdanau attention mechanism <https://blog.floydhub.com/attention-mechanism/>

```
01 class BahdanauAttention(tf.keras.Model):
02     def __init__(self, units):
03         super(BahdanauAttention, self).__init__()
04         self.W1 = tf.keras.layers.Dense(units)
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
05     self.W2 = tf.keras.layers.Dense(units)
06     self.V = tf.keras.layers.Dense(1)
07
08     def call(self, features, hidden):
09         # features (CNN_encoder output) shape == (batch_size, 64,
10         embedding_dim)
11         # hidden shape == (batch_size, hidden_size)
12         # hidden_with_time_axis shape == (batch_size, 1, hidden_size)
13         hidden_with_time_axis = tf.expand_dims(hidden, 1)
14
15         # attention_hidden_layer shape == (batch_size, 64, units)
16         attention_hidden_layer = (tf.nn.tanh(self.W1(features) +
17         self.W2(hidden_with_time_axis)))
18
19         # score shape == (batch_size, 64, 1)
20         # This gives you an unnormalized score for each image feature.
21         score = self.V(attention_hidden_layer)
22
23         # attention_weights shape == (batch_size, 64, 1)
24         attention_weights = tf.nn.softmax(score, axis=1)
25
26         # context_vector shape after sum == (batch_size, hidden_size)
27         context_vector = attention_weights * features
28         context_vector = tf.reduce_sum(context_vector, axis=1)
29
30         return context_vector, attention_weights
```

Table 4: PairUp's Bahdanau Attention code implementation

The **RNN Decoder** in PairUp's Image to Text Synthesizer is implemented as a **Gated Recurrent Unit (GRU)** [15] with **Bahdanau Attention**. GRUs are a gating mechanism in recurrent neural networks, introduced in 2014 by Kyunghyun Cho et al. The GRU (see Figure 27) is like a long short-term memory (LSTM) with a forget gate, but has fewer parameters than LSTM, as it lacks an output gate.

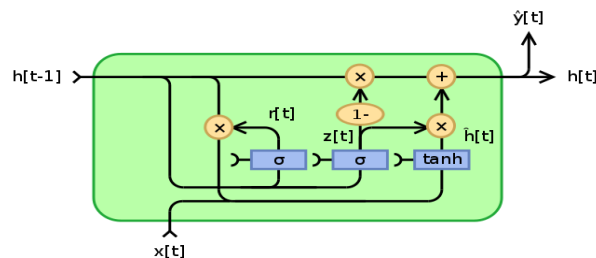


Figure 27: Gated Recurrent Unit (GRU)

---

## PairUp: Detecting Congruous Image and Text Pairings

---

The decoder takes as input the encoded features, the hidden state and the decoder's own output (being fed to itself) and returns the predictions, its own hidden state and attention weights. Table 5 shows the Keras code created to implement this RNN.

```
01 class RNN_Decoder(tf.keras.Model):
02     def __init__(self, embedding_dim, units, vocab_size):
03         super(RNN_Decoder, self).__init__()
04         self.units = units
05
06         self.embedding = tf.keras.layers.Embedding(vocab_size,
embedding_dim)
07         self.gru = tf.keras.layers.GRU(self.units,
08                                         return_sequences=True,
09                                         return_state=True,
10 recurrent_initializer='glorot_uniform')
11         self.fc1 = tf.keras.layers.Dense(self.units)
12         self.fc2 = tf.keras.layers.Dense(vocab_size)
13
14         self.attention = BahdanauAttention(self.units)
15
16         # https://stackoverflow.com/questions/62250441/saving-a-
tensorflow-keras-model-encoder-decoder-to-savedmodel-format
17         @tf.function(input_signature=[tf.TensorSpec(shape=(None, 1),
dtype=tf.int64),
18                                     tf.TensorSpec(shape=(None, 64,
embedding_dim), dtype=tf.float32),
19                                     tf.TensorSpec(shape=(None, units),
dtype=tf.float32)])
20     def call(self, x, features, hidden):
21         # print(x.shape)
22         # print(features.shape)
23         # print(hidden.shape)
24
25         # defining attention as a separate model
26         context_vector, attention_weights = self.attention(features,
hidden)
27
28         # x shape after passing through embedding == (batch_size, 1,
embedding_dim)
29         x = self.embedding(x)
30
31         # x shape after concatenation == (batch_size, 1, embedding_dim
+ hidden_size)
32         x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)
33
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
34     # passing the concatenated vector to the GRU
35     output, state = self.gru(x)
36
37     # shape == (batch_size, max_length, hidden_size)
38     x = self.fc1(output)
39
40     # x shape == (batch_size * max_length, hidden_size)
41     x = tf.reshape(x, (-1, x.shape[2]))
42
43     # output shape == (batch_size * max_length, vocab)
44     x = self.fc2(x)
45     out = x
46
47     # return x, state, attention_weights
48     return out, state, attention_weights
49
50 def reset_state(self, batch_size):
51     # print(batch_size)
52     return tf.zeros((batch_size, self.units))
```

Table 5: PairUp's Image to Text Synthesizer's Decoder implementation

PairUp instantiates both encoder and decoder model as follows:

```
01 encoder = CNN_Encoder(embedding_dim)
02 decoder = RNN_Decoder(embedding_dim, units,
tokenizer.vocabulary_size())
```

### 7.4. TRAINING THE MODEL

For training the PairUp's image captioning model both an optimizer and loss function are needed. It uses the **Adam optimizer** since it combines the advantages of two other extensions of stochastic gradient descent namely **AdaGrad** which maintains a per-parameter learning rate that improves performance on problems with sparse gradients and **RMSProp** which also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). For loss, PairUp computes the **Sparse categorical cross entropy** loss between the labels and predictions. Table 6 shows the related code for loss calculation and optimizer definition.

```
01 optimizer = tf.keras.optimizers.Adam()
02 loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
03     from_logits=True, reduction='none')
04
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
05
06 def loss_function(real, pred):
07     mask = tf.math.logical_not(tf.math.equal(real, 0))
08     loss_ = loss_object(real, pred)
09
10     mask = tf.cast(mask, dtype=loss_.dtype)
11     loss_ *= mask
12
13     return tf.reduce_mean(loss_)
```

Table 6: PairUp optimizer function definition (Adam) and loss function calculation.

During training checkpoints are created and saved for easy stop and restart of the training process and also for use of models at any saved checkpoint. Table 7 sets up this code.

```
01 checkpoint_path =
"/content/drive/MyDrive/Capstone_Project/checkpoints/checkpoints_20k/t
rain"
02
03
04 ckpt = tf.train.Checkpoint(encoder=encoder,
05                             decoder=decoder,
06                             optimizer=optimizer)
07 ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path,
max_to_keep=2)
08
09 start_epoch = 1
10 if ckpt_manager.latest_checkpoint:
11     start_epoch = int(ckpt_manager.latest_checkpoint.split('-')[-1])
12     # restoring the latest checkpoint in checkpoint_path
13     ckpt.restore(ckpt_manager.latest_checkpoint)
```

```
01 train_writer =
tf.summary.create_file_writer("output_20k/logs/train/")
```

Table 7: Check point saving setup in PairUp. PairUp creates a summary writer object so that the training log details can be saved

PairUp training is accomplished using the following steps which are shown in the code in Table 8.

- Loop that iterates over epochs
- For each epoch, we open a for loop that iterates over the dataset, in batches

---

## PairUp: Detecting Congruous Image and Text Pairings

---

- For each batch, we open a GradientTape() scope defined inside **train\_step()** function
- Inside this scope, we call the model (forward pass) and compute the loss. The ENCODER output, hidden state (initialized to 0) and the DECODER input (which is the <start> token) are passed to the DECODER. The DECODER returns the predictions and the DECODER hidden state. The DECODER hidden state is then passed back into the model and the predictions are used to calculate the loss.

While training, we use the **Teacher Forcing** technique, to decide the next input of the DECODER. It is a technique where the target word is passed as the next input to the decoder. This technique helps to learn the correct sequence or correct statistical properties for the sequence, quickly.

- Outside the scope, we retrieve the gradients of the weights of the model with regard to the loss
- Finally, we use the optimizer to update the weights of the model based on the gradients
- We save into a checkpoint after each epoch and output the training loss.

```
01 EPOCHS = 200
02
03 for epoch in range(start_epoch, EPOCHS):
04     start = time.time()
05     total_loss = 0
06
07     for (batch, (img_tensor, target)) in enumerate(dataset):
08         batch_loss, t_loss = train_step(img_tensor, target)
09         total_loss += t_loss
10
11         if batch % 100 == 0:
12             average_batch_loss =
batch_loss.numpy()/int(target.shape[1])
13             print(f'Epoch {epoch} Batch {batch} Loss
{average_batch_loss:.4f}')
14
15         print(f'Epoch {epoch} Loss {total_loss/num_steps:.6f}')
16         print(f'Time taken for 1 epoch {time.time()-start:.2f} sec\n')
17
18     ckpt_manager.save()
19
20     with train_writer.as_default():
21         tf.summary.scalar("Loss", total_loss/num_steps, step=epoch)
22
23 @tf.function
24 def train_step(img_tensor, target):
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
24 # print(f'\nimg_tensor.shape: {img_tensor.shape}')
25 # print(f'target.shape: {target.shape}')
26
27 loss = 0
28
29 # initializing the hidden state for each batch
30 # because the captions are not related from image to image
31 hidden = decoder.reset_state(batch_size=target.shape[0])
32 # print(f'hidden.shape: {hidden.shape}')
33
34 dec_input = tf.expand_dims([word_to_index('<start>')] *
target.shape[0], 1)
35 # print(f'dec_input.shape: {dec_input.shape}')
36
37 with tf.GradientTape() as tape:
38     features = encoder(img_tensor)
39     # print(f'features.shape: {features.shape}')
40
41     for i in range(1, target.shape[1]):
42         # passing the features through the decoder
43         predictions, hidden, _ = decoder(dec_input, features,
hidden)
44
45         loss += loss_function(target[:, i], predictions)
46
47         # using teacher forcing
48         dec_input = tf.expand_dims(target[:, i], 1)
49
50         # print(f'predictions.shape: {predictions.shape}')
51
52 total_loss = (loss / int(target.shape[1]))
53
54 trainable_variables = encoder.trainable_variables +
decoder.trainable_variables
55
56 gradients = tape.gradient(loss, trainable_variables)
57
58 optimizer.apply_gradients(zip(gradients, trainable_variables))
59
60 return loss, total_loss
```

Table 8: Check point saving setup in PairUp. PairUp creates a summary writer object so that the training log details can be saved.

We use **TensorBoard** for visualizing the loss metrics and save the final encoder and decoder models as below:



---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
01 %load_ext tensorboard
02 %tensorboard --logdir output_20k/logs
03
04 tf.saved_model.save(encoder, 'output_20k/encoder_model')
05 tf.saved_model.save(decoder, 'output_20k/decoder_model')
```

### 7.5. SELECTION OF HYPER-PARAMETERS

PairUp uses a tf.data dataset for training our model and the following configuration settings.

```
01 BATCH_SIZE = 64
02 BUFFER_SIZE = 1000
03 embedding_dim = 256
04 units = 512
05 num_steps = len(img_name_train) // BATCH_SIZE
06 # Shape of the vector extracted from InceptionV3 is (64, 2048)
07 # These two variables represent that vector shape
08 features_shape = 2048
09 attention_features_shape = 64
10
11
12 # Load the numpy files
13 def map_func(img_name, cap):
14     img_tensor = np.load(img_name.decode('utf-8')+'.npy')
15     return img_tensor, cap
16
17
18 dataset = tf.data.Dataset.from_tensor_slices((img_name_train,
19 cap_train))
20
21 # Use map to load the numpy files in parallel
22 dataset = dataset.map(lambda item1, item2: tf.numpy_function(
23     map_func, [item1, item2], [tf.float32,tf.int64]),
24     num_parallel_calls=tf.data.AUTOTUNE)
25
26 # Shuffle and batch
27 dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
28 dataset = dataset.prefetch(buffer_size=tf.data.AUTOTUNE)
```

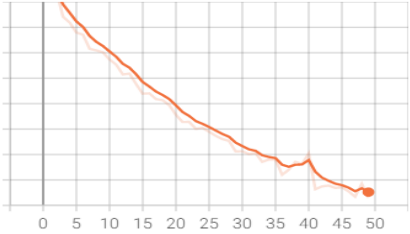
Table 9 shows the results of different training runs with a different number of epochs. As can be seen, the best training is experienced with the model trained to 150 epochs. While there is some variance in these 4 training runs for loss, it is minimal ending in losses near 0.2. However, we additionally applied different metrics related to text similarity of the synthesized text test dataset (see section 7.6) as it compares to the

---

## PairUp: Detecting Congruous Image and Text Pairings

---

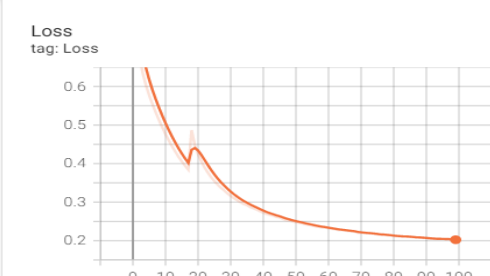
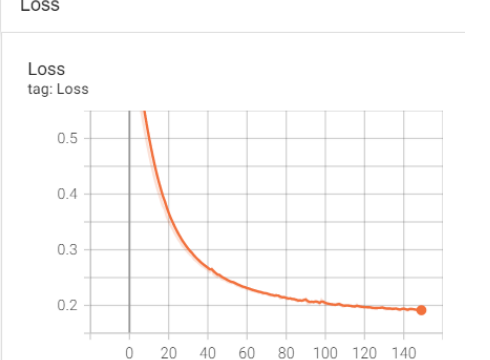
ground truth including SPICE, Bleu\_1, Bleu\_2, Bleu\_3, Bleu\_4, METEOR, Rouge\_L, and CIDEr. Section 7.7 discusses each of these metrics. Testing results for these metrics over a test dataset are also presented in Table 9, the average value of each metric is given. Training session number 3 is superior in terms of the SPICE metric and this model is selected as the best performing model for the image captioning task in PairUp.

Models	Hyper-parameters	Summary & Plots
1	<pre>EPOCHS = 50 BATCH_SIZE = 64 BUFFER_SIZE = 1000 embedding_dim = 256 units = 512 features_shape = 2048 attention_features_shape = 64</pre>	<div><div>Loss</div><div><div>Loss</div><div>tag: Loss</div></div></div> <div><pre>SPICE: 0.094 Bleu_1: 0.406 Bleu_2: 0.240 Bleu_3: 0.136 Bleu_4: 0.076 METEOR: 0.151 ROUGE_L: 0.334 CIDEr: 0.279</pre></div> <div><p>Observations: We see that the loss curve decreases gradually with each epoch but around 40 epochs it tends to go a bit high before going down further. The SPICE and Bleu score shows very low values.</p></div>

---

## PairUp: Detecting Congruous Image and Text Pairings

---

2	<pre>EPOCHS = 100 BATCH_SIZE = 64 BUFFER_SIZE = 1000 embedding_dim = 256 units = 512 features_shape = 2048 attention_features_shape = 64</pre>	<div><p>Loss</p><p>Loss tag: Loss</p><table><tr><th>Epoch</th><th>Loss</th></tr><tr><td>0</td><td>0.65</td></tr><tr><td>10</td><td>0.55</td></tr><tr><td>20</td><td>0.45</td></tr><tr><td>30</td><td>0.35</td></tr><tr><td>40</td><td>0.30</td></tr><tr><td>50</td><td>0.28</td></tr><tr><td>60</td><td>0.26</td></tr><tr><td>70</td><td>0.25</td></tr><tr><td>80</td><td>0.24</td></tr><tr><td>90</td><td>0.23</td></tr><tr><td>100</td><td>0.22</td></tr></table></div> <p>SPICE: 0.106 Bleu_1: 0.433 Bleu_2: 0.270 Bleu_3: 0.163 Bleu_4: 0.097 METEOR: 0.162 ROUGE_L: 0.351 CIDEr: 0.322</p> <p>Observations: We see that with a higher epoch of 100, SPICE and Bleu scores are higher indicating that the model's performance improves.</p>	Epoch	Loss	0	0.65	10	0.55	20	0.45	30	0.35	40	0.30	50	0.28	60	0.26	70	0.25	80	0.24	90	0.23	100	0.22
Epoch	Loss																									
0	0.65																									
10	0.55																									
20	0.45																									
30	0.35																									
40	0.30																									
50	0.28																									
60	0.26																									
70	0.25																									
80	0.24																									
90	0.23																									
100	0.22																									
3	<pre>EPOCHS = 150 BATCH_SIZE = 64 BUFFER_SIZE = 1000 embedding_dim = 256 units = 512 features_shape = 2048 attention_features_shape = 64</pre>	<div><p>Loss</p><p>Loss tag: Loss</p><table><tr><th>Epoch</th><th>Loss</th></tr><tr><td>0</td><td>0.55</td></tr><tr><td>20</td><td>0.35</td></tr><tr><td>40</td><td>0.28</td></tr><tr><td>60</td><td>0.25</td></tr><tr><td>80</td><td>0.23</td></tr><tr><td>100</td><td>0.22</td></tr><tr><td>120</td><td>0.21</td></tr><tr><td>140</td><td>0.20</td></tr><tr><td>150</td><td>0.20</td></tr></table></div> <p>SPICE: 0.148 Bleu_1: 0.514 Bleu_2: 0.378 Bleu_3: 0.278 Bleu_4: 0.205 METEOR: 0.211 ROUGE_L: 0.430</p>	Epoch	Loss	0	0.55	20	0.35	40	0.28	60	0.25	80	0.23	100	0.22	120	0.21	140	0.20	150	0.20				
Epoch	Loss																									
0	0.55																									
20	0.35																									
40	0.28																									
60	0.25																									
80	0.23																									
100	0.22																									
120	0.21																									
140	0.20																									
150	0.20																									

## PairUp: Detecting Congruous Image and Text Pairings

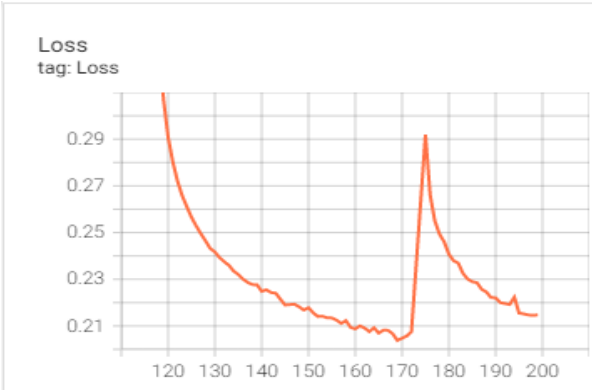
		<p>CIDEr: 0.596</p> <p>Observations: With epoch of 150, we see even higher SPICE and Bleu scores indicating that the model has room for even better performance.</p>
4	<pre> EPOCHS = 200 BATCH_SIZE = 64 BUFFER_SIZE = 1000 embedding_dim = 256 units = 512 features_shape = 2048 attention_features_shape = 64 </pre>	 <p>SPICE: 0.126  Bleu_1: 0.479  Bleu_2: 0.331  Bleu_3: 0.222  Bleu_4: 0.147  METEOR: 0.188  ROUGE_L: 0.397  CIDEr: 0.455</p> <p>Observations: When we try even higher epochs, the loss starts to increase beyond epoch 170 and starts decreasing from 175 onwards. The spike in loss curve may be attributed to unavoidable consequences of Mini-Batch Gradient Descent in Adam. Some mini-batches have 'by chance' unlucky data for the optimization, inducing those spikes. We don't see overfitting of the model but still see lower SPICE score for the validation data.</p>

Table 9: PairUp Image to Text Synthesizer model training runs, 4 different training runs took place with varying number of epochs

### 7.6. TESTING THE MODEL

In order to generate the caption for an unseen image, we have created an evaluate function that looks similar to the training loop, except that we don't use teacher forcing here. This means the input to the decoder at each time step is its previous predictions

---

## PairUp: Detecting Congruous Image and Text Pairings

---

along with the hidden state and the encoder output. The model stops predicting when it encounters an “end” token. It also stores the attention weights for every time step. We make use of this same function to evaluate the performance of the model on the validation set. The results code is shown in Table 10.

```
01 def evaluate(image):
02     attention_plot = np.zeros((max_length,
03                                attention_features_shape))
04     hidden = decoder.reset_state(batch_size=1)
05     # print(f'hidden.shape: {hidden.shape}')
06
07     temp_input = tf.expand_dims(load_image(image)[0], 0)
08     img_tensor_val = image_features_extract_model(temp_input)
09     img_tensor_val = tf.reshape(img_tensor_val,
10                                (img_tensor_val.shape[0],
11                                 -1,
12                                img_tensor_val.shape[3]))
13     features = encoder(img_tensor_val)
14     # print(f'features.shape: {features.shape}')
15
16     dec_input = tf.expand_dims([word_to_index('<start>')], 0)
17     # print(f'dec_input.shape: {dec_input.shape}')
18     result = []
19
20     for i in range(max_length):
21         predictions, hidden, attention_weights = decoder(dec_input,
22                                                         features,
23                                                         hidden)
24         # print(f'predictions.shape: {predictions.shape}')
25
26         attention_plot[i] = tf.reshape(attention_weights, (-1,
27                                                         ))).numpy()
28
29         predicted_id = tf.random.categorical(predictions,
30                                              1)[0][0].numpy()
31         predicted_word =
32         tf.compat.as_text(index_to_word(predicted_id).numpy())
33         result.append(predicted_word)
34
35         # print(tf.random.categorical(predictions, 1))
36         # print(f'predicted_id: {predicted_id}')
37         # print(f'predicted_word: {predicted_word}\n')
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
36         if predicted_word == '<end>':
37             return result, attention_plot
38
39         dec_input = tf.expand_dims([predicted_id], 0)
40         dec_input = tf.cast(dec_input, tf.int64)
41
42         attention_plot = attention_plot[:len(result), :]
43         return result, attention_plot
```

Table 10: PairUp evaluation code for the Image to Text Synthesizer.

The following code iterates over the validation set and dumps the generated captions for the images into a JSON file.

```
01 results_val = []
02 image_ids = []
03
04 for id in range(0, len(img_name_val), 5):
05     image = img_name_val[id]
06     real_caption = '
07     '.join([tf.compat.as_text(index_to_word(i).numpy())
08             for i in cap_val[id] if i not in [0]])
09     result, attention_plot = evaluate(image)
10
11     out_dict = {}
12     out_dict["image_id"] = int(image[-10:-4])
13     out_dict["caption"] = '
14     '.join(result[:-1])
15
16     if out_dict["image_id"] not in image_ids:
17         results_val.append(out_dict)
18     else:
19         print(f"image_id: {out_dict['image_id']} is duplicate")
20         image_ids.append(int(image[-10:-4]))
21
22 import json
23 with open('output_20k/output_val.json', 'w') as fout:
24     json.dump(results_val, fout)
```

### 7.7. METRICS

Metrics can be used for evaluating the goodness of the synthesized text by PairUp's Image to Text Synthesizer. Common metrics used for this are posed as similarity measures that would compare PairUp's synthesized text/ sentence to the set of ground-truth sentences (captions in dataset corresponding to the image). Most proposed metrics perform n-gram matching and these include BLEU [24] (and its variations), ROGUE [25],

---

## PairUp: Detecting Congruous Image and Text Pairings

---

CIDEr [26], and METEOR [27]. A different approach is taken by the SPICE [28] metric which uses scene graphs which allows for noun / object matching between the texts being compared and in that sense can be considered more “intelligent”. Each of these metrics is discussed in this section and defined using the original publication material.

Bleu is a modified precision metric with a sentence-brevity penalty, calculated as a weighted geometric mean over different length n-grams. METEOR uses exact, stem, synonym and paraphrase matches between n-grams to align sentences, before computing a weighted F-score with an alignment fragmentation penalty. ROUGE is a package of measures for automatic evaluation of text summaries using F-measures. CIDEr applies term frequency-inverse document frequency (tf-idf) weights to n-grams in the candidate and reference sentences, which are then compared by summing their cosine similarity across n-grams.

The metric BLEU (BiLingual Evaluation Understudy) is a geometric mean of n-gram scores from 1 to 4. It is based on n-gram based precision. Although it is easy to compute, BLEU has some weaknesses since it is mostly a measure of fluency rather than semantic similarity.

$$\text{BLEU}_n(a, b) = \frac{\sum_{w_n \in a} \min \left( c_a(w_n), \max_{j=1, \dots, |b|} c_{b_j}(w_n) \right)}{\sum_{w_n \in a} c_a(w_n)}$$

Where, a: candidate sentence, b: set of reference sentences,  $w_n$ : n-gram,  $c_x(y_n)$ : count of n-gram  $y_n$  in sentence x.

The metric ROGUE (Recall Oriented Understudy of Gisting Evaluation) is similar to BLEU but is based on n-gram based recall. It shares similar strengths and weaknesses as BLEU.

$$\text{ROUGE}_n(a, b) = \frac{\sum_{j=1}^{|b|} \sum_{w_n \in b_j} \min \left( c_a(w_n), c_{b_j}(w_n) \right)}{\sum_{j=1}^{|b|} \sum_{w_n \in b_j} c_{b_j}(w_n)}$$

Where, a: candidate sentence, b: set of reference sentences,  $w_n$ : n-gram,  $c_x(y_n)$ : count of n-gram  $y_n$  in sentence x.

---

## PairUp: Detecting Congruous Image and Text Pairings

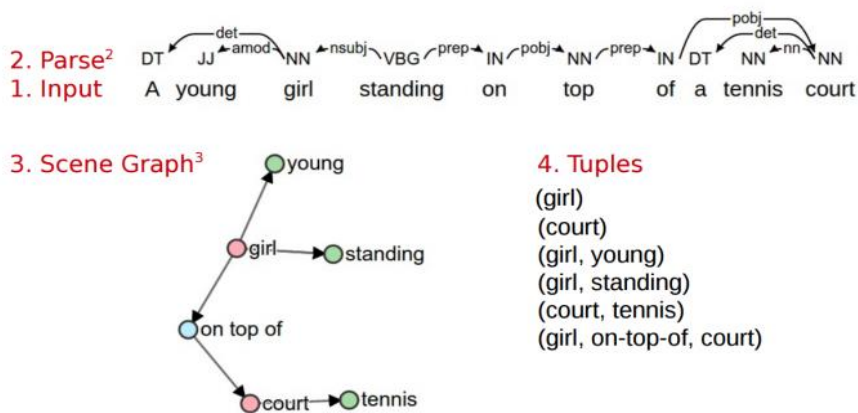
---

The metric METEOR (Metric for Evaluation of Translation with Explicit ORdering) provides smoother penalization of different ordering of chunks and higher correlation with human consensus scores. The metric CIDEr (Consensus-based Image Description Evaluation) gives more weight-age to important n-grams and higher correlation with human consensus scores as compared to the previously discussed metrics.

$$CIDEr_n(a, b) = \frac{1}{|b|} \sum_{j=1}^{|b|} \frac{\mathbf{g}^n(a) \cdot \mathbf{g}^n(b_j)}{\|\mathbf{g}^n(a)\| \|\mathbf{g}^n(b_j)\|}$$

Where, a: candidate sentence, b: set of reference sentences,  $g^n(x)$ : vector formed by TF-IDF scores of all n-grams in x.

The metric SPICE (Semantic Propositional Image Caption Evaluation) is based on the key idea of creation of scene graphs and is calculated as F-score over tuples with: merging of synonymous nodes, and wordnet, synsets used for tuple matching and merging.



<sup>1</sup> Johnson et. al. Image Retrieval Using Scene Graphs, CVPR 2015

<sup>2</sup> Klein & Manning: Accurate Unlexicalized Parsing, ACL 2003

<sup>3</sup> Schuster et. al: Generating semantically precise scene graphs from textual descriptions for improved image retrieval, EMNLP 2015

Figure 28: Scene graph. Taken from [28]

Given candidate caption c, a set of reference caption S, the mapping T from captions to tuples:



---

## PairUp: Detecting Congruous Image and Text Pairings

---

$$P(c, S) = \frac{|T(c) \otimes T(S)|}{|T(c)|}$$
$$R(c, S) = \frac{|T(c) \otimes T(S)|}{|T(S)|}$$
$$SPICE(c, S) = F_1(c, S) = \frac{2 \cdot P(c, S) \cdot R(c, S)}{P(c, S) + R(c, S)}$$

The SPICE metric places importance on capturing details about objects, attributes and relationships and higher correlation with humans compared to n-gram based metrics. But the metric does not check whether the grammar is correct. It depends on semantic parsers, which might not always be correct.

We make use of the **pycocoevalcap** repository that provides Python 3 support for the caption evaluation metrics used for the MS COCO dataset. As inputs we provide the original captions file and the generated caption file in JSON format to create the coco original and coco results object. We then create a coco evaluation object with both and evaluate it to generate the results using the metrics discussed above. The code is shown in Table 11. The test metric values on this data for different training model runs is shown in Table 9 which is run over 4K samples reserved for testing from the MS COCO dataset.

```
01 # https://github.com/salaniz/pycocoevalcap
02 !pip install pycocoevalcap
03 !pip install pycocotools
04 # !pip install "git+https://github.com/salaniz/pycocoevalcap.git"
05
06 from pycocotools.coco import COCO
07 from pycocoevalcap.eval import COCOEvalCap
08
09 annotation_file =
10 '/content/drive/MyDrive/Capstone_Project/annotations/captions_train201
11 4.json'
12 results_file =
13 '/content/drive/MyDrive/Capstone_Project/output_20k/output_val.json'
14
15 # create coco object and coco_result object
16 coco = COCO(annotation_file)
17 coco_result = coco.loadRes(results_file)
18
19 # create coco_eval object by taking coco and coco_result
20 coco_eval = COCOEvalCap(coco, coco_result)
21
22 # evaluate on a subset of images by setting
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
20 # coco_eval.params['image_id'] = coco_result.getImgIds()
21 # please remove this line when evaluating the full validation set
22 coco_eval.params['image_id'] = coco_result.getImgIds()
23
24 # evaluate results
25 # SPICE will take a few minutes the first time, but speeds up due
  to caching
26 coco_eval.evaluate()
27
28 # print output evaluation scores
29 for metric, score in coco_eval.eval.items():
30     print(f'{metric}: {score:.3f}')
```

*Table 11: PairUp code to calculate similarity metrics for the Image to Text Synthesizer which uses the text generated by the PairUp test evaluation code shown in Table 10*

### 8. TEXTUAL SIMILARITY ANALYZER

PairUp uses an ensemble of seven NLP models in a voting scheme as shown in Figure 29. Five of which are pre-trained sentence-transformer models from Hugging face library and two were custom built. The cosine similarity scores from these seven models is used to determine if the sentence pairs are congruous or not.

In section 8.1, the dataset used for the two custom models is discussed followed by the preprocessing stage needed for all models in section 8.2. In section 8.3 the creation and training of the two custom models is covered. The remaining 5 pre-trained models are described in section 8.4. Finally in section 8.5, we present a comparison of all the models.

The voting scheme takes as output from each model a cosine similarity score. A majority rule vote is taken as follows:

- Synthesized Text, Input Text is Congruous if the majority of models have a cosine similarity score  $\geq 0.5$
- Otherwise it is In-Congruous (not congruous).

The threshold value of 0.5 was chosen empirically after examining numerous pairing results. The good thing about having a majority rules multi-model Textual Similarity Analyzer is that it makes PairUp more robust than using a single model because oftentimes a failure in one model is not necessarily repeated in the other models.

---

# PairUp: Detecting Congruous Image and Text Pairings

---

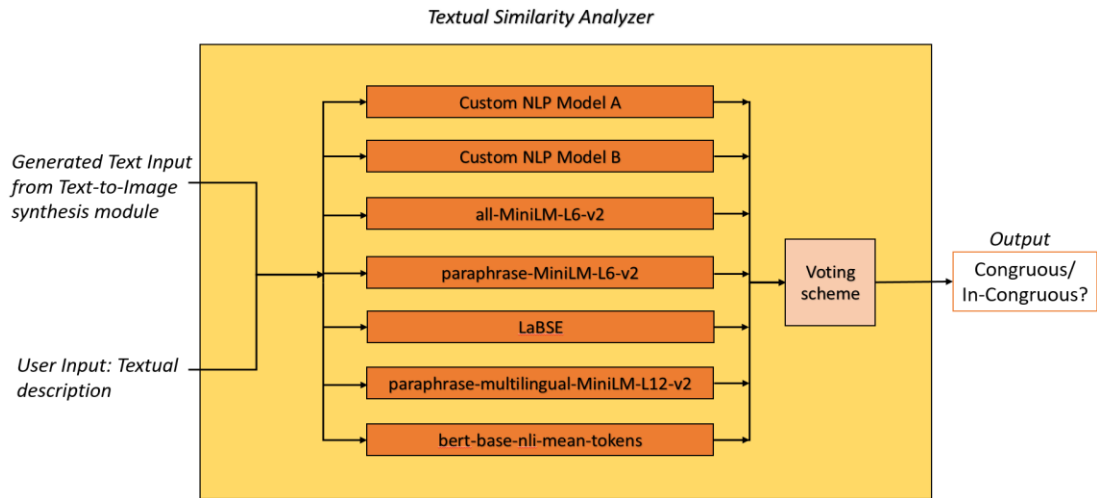


Figure 29: An overview of Textual Similarity Analyzer module

## 8.1. DATASET

In order to train the two custom models, we make use of the **Stanford Natural Language Inference (SNLI)** dataset [29]. It is a collection of 570k human-written English sentence pairs manually labeled for balanced classification with the labels entailment, contradiction, and neutral. Roughly entailment can be considered as semantically similar text pairs, contradiction as not similar and neutral as uncertain or neither similar or not similar. Figure 30 shows a few examples from this dataset. Neutral is not something that PairUp will classify but, it describes that transition between similar and not similar. All pairs include contain one of the following labels:

- 0 - Entailment,
- 1 - Neutral
- 2 - Contradiction

Text	Judgments	Hypothesis
A man inspects the uniform of a figure in some East Asian country.	contradiction C C C C C	The man is sleeping
An older and younger man smiling.	neutral N N E N N	Two men are smiling and laughing at the cats playing on the floor.
A black race car starts up in front of a crowd of people.	contradiction C C C C C	A man is driving down a lonely road.
A soccer game with multiple males playing.	entailment E E E E E	Some men are playing a sport.
A smiling costumed woman is holding an umbrella.	neutral N N E C N	A happy woman in a fairy costume holds an umbrella.

Figure 30: Examples from SNLI dataset from <https://nlp.stanford.edu/projects/snli/>

PairUp uses **SentenceTransformers** which is a Python framework that is used for state-of-the-art sentence, text and image embeddings. The Hugging Face Hub [30] hosts Git-based repositories which are storage spaces including the hosting of the

---

# PairUp: Detecting Congruous Image and Text Pairings

---

SNLI dataset. The following shows the installation of required software and loading of the SNLI dataset.

```
01 !pip install -q sentence_transformers
02 !pip install -q huggingface-hub
03 !pip install datasets
```

```
01 from datasets import load_dataset
02
03 snli = load_dataset("snli", split="train")
04 dataset = snli
05 snli
```

## 8.2. PREPROCESSING

The SNLI dataset contains -1 values in the label feature where no confident class could be assigned. These samples are removed from the SNLI dataset using the following code.

```
01 print(f"Number of entries in dataset BEFORE removal of invalid
entries: {len(dataset)}")
02 dataset = dataset.filter(
03     lambda x: False if x['label'] == -1 else True
04 )
05 print(f"Number of entries in dataset AFTER removal of invalid
entries: {len(dataset)}")
```

PairUp uses a pre-trained uncased **BERT (Bidirectional Encoder Representations from Transformer)** to create contextualized word embeddings as depicted in Figure 31. It was pre-trained on the English language using a masked language modeling (MLM) objective. The model used here is uncased: it does not make a difference between “english” and “English”.

---

# PairUp: Detecting Congruous Image and Text Pairings

---

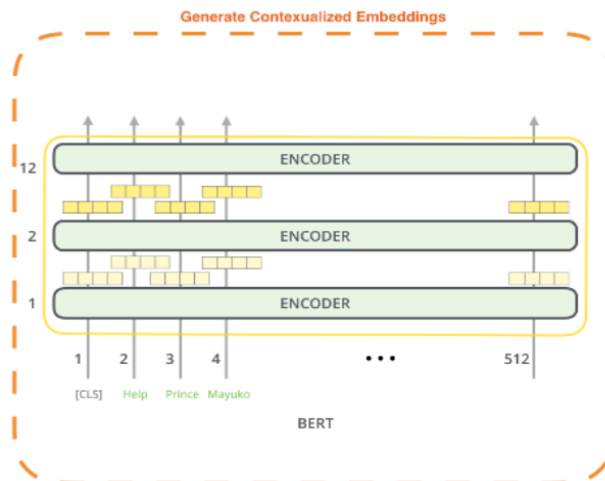


Figure 31: Creating contextualized embeddings using pre-trained BERT [16]

Originally, this BERT model was pre-trained with two objectives:

- **Masked language modeling (MLM):** taking a sentence, the model randomly masks 15% of the words in the input then runs the entire masked sentence through the model and has to predict the masked words (see Figure 32). This is different from traditional recurrent neural networks (RNNs) that usually see the words one after the other, or from autoregressive models like GPT which internally mask the future tokens. It allows the model to learn a bidirectional representation of the sentence.

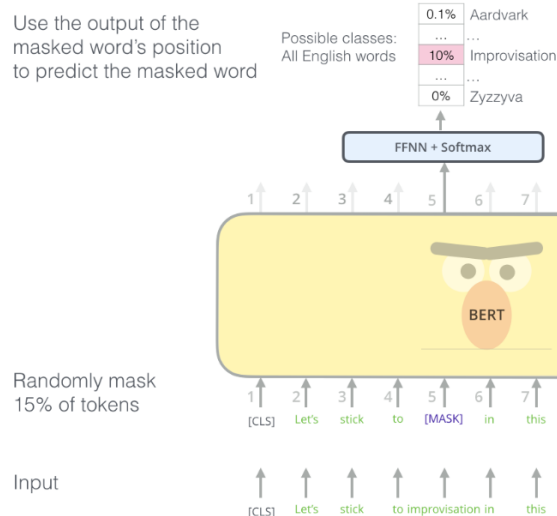


Figure 32: BERT trained with masked language modelling objective [16]

---

# PairUp: Detecting Congruous Image and Text Pairings

---

- **Next sentence prediction (NSP):** the models concatenate two masked sentences as inputs during pre-training (see Figure 33). Sometimes they correspond to sentences that were next to each other in the original text, sometimes not. The model then has to predict if the two sentences were following each other or not.

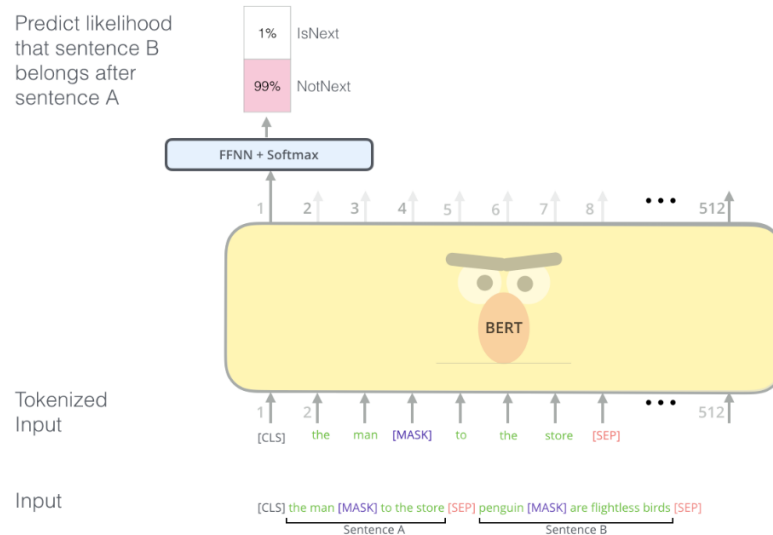


Figure 33: BERT trained with next sentence prediction objective [16]

## 8.2.1. Pre-processing for Custom NLP Model A

For training we convert the human-readable sentences into transformer-readable tokens, so we go ahead and tokenize our sentences using **BertTokenizer**.

```
01 import torch
02 from transformers import BertTokenizer
03
04
05 tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

Both premise and hypothesis features must be split into their own `input_ids` and `attention_mask` tensors.

```
01 all_cols = ['label']
02
03 for part in ['premise', 'hypothesis']:
04     dataset = dataset.map(
05         lambda x: tokenizer(
06             x[part], max_length=128, padding='max_length',
07             truncation=True
08         ), batched=True
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
09     )
10     for col in ['input_ids', 'attention_mask']:
11         dataset = dataset.rename_column(
12             col, part+'_'+col
13         )
14         all_cols.append(part+'_'+col)
15 print(all_cols)
```

Now we prepare the data to be read into the model. To do this, we first convert the dataset features into PyTorch tensors and then initialize a data loader which will feed data into our model during training.

```
01 # convert dataset features to PyTorch tensors
02 dataset.set_format(type='torch', columns=all_cols)
03
04 # initialize the dataloader
05 batch_size = 16
06 loader = torch.utils.data.DataLoader(dataset, batch_size=batch_size,
07                                       shuffle=True)
```

### 8.2.2. Pre-processing for Custom NLP Model B

We make use of the same SNLI dataset but they need to be transformed into the format required by sentence-transformers using their **InputExample** class. So, we convert the current premise, hypothesis, and label format into an almost matching format with the InputExample class.

```
01 from sentence_transformers import InputExample
02 from tqdm.auto import tqdm
03
04
05 train_samples = []
06 for row in tqdm(dataset):
07     train_samples.append(InputExample(
08         texts=[row['premise'], row['hypothesis']],
09         label=row['label']
10     ))
```

We also initialized a DataLoader to load the dataset.

```
01 from torch.utils.data import DataLoader
02
03
04 batch_size = 16
05 loader = DataLoader(train_samples, batch_size=batch_size,
06                     shuffle=True)
```

---

# PairUp: Detecting Congruous Image and Text Pairings

---

## 8.3. CUSTOM MODEL CREATION AND TRAINING

In this section, we discuss the various NLP models that form the core logic to determine sentence similarity.

### 8.3.1. Custom NLP Model A

To train our custom SBERT model, we don't start from scratch. We begin with an already pre-trained BERT model (and tokenizer).

```
01 import torch
02 from transformers import BertModel
03
04
05 # start from a pretrained bert-base-uncased model
06 model = BertModel.from_pretrained('bert-base-uncased')
07 # model
```

We create a “Siamese”-BERT architecture during training. All this means is that given a sentence pair, we feed sentence A into BERT first, then feed sentence B once BERT has finished processing the first.

This has the effect of creating a Siamese-like network where we can imagine two identical BERTs are being trained in parallel on sentence pairs. In reality, there is just a single model processing two sentences one after the other.

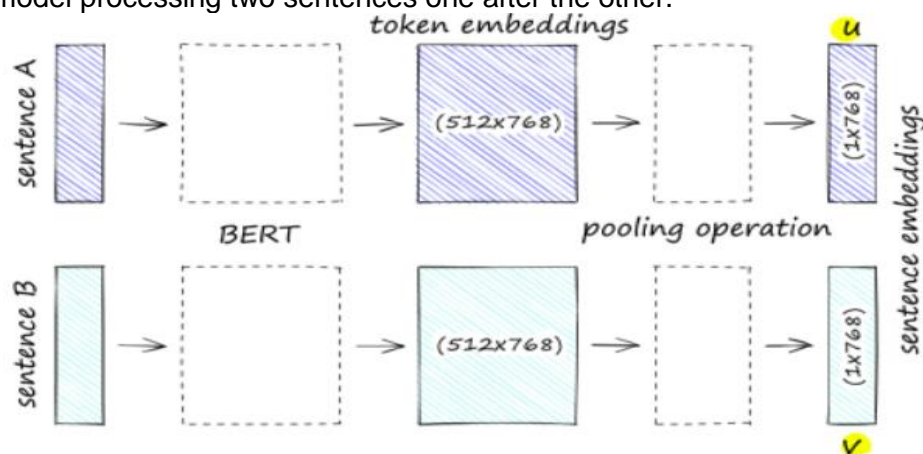


Figure 34: An SBERT model applied to a sentence pair sentence A and sentence B. Note that the BERT model outputs token embeddings (consisting of 512 768-dimensional vectors). We then compress that data into a single 768-dimensional sentence vector using a pooling [17]



---

## PairUp: Detecting Congruous Image and Text Pairings

---

BERT outputs 512 X 768-dimensional embeddings. We convert these into an average embedding using mean-pooling. This pooled output is our sentence embedding. We will have two per step - one for sentence “A” that we call  $u$ , and one for sentence “B”, called  $v$ .

To perform this mean pooling operation, we will define a function called `mean_pool`.

```
01 # define mean pooling function
02 def mean_pool(token_embeds, attention_mask):
03     # reshape attention_mask to cover 768-dimension embeddings
04     in_mask = attention_mask.unsqueeze(-1).expand(
05         token_embeds.size()
06     ).float()
07     # perform mean-pooling but exclude padding tokens (specified by
08     # in_mask)
09     pool = torch.sum(token_embeds * in_mask, 1) / torch.clamp(
10         in_mask.sum(1), min=1e-9
11     )
12     return pool
```

We take BERT’s token embeddings output and the sentence’s `attention_mask` tensor. We then resize the `attention_mask` to align to the higher 768-dimensionality of the token embeddings. We apply this resized mask `in_mask` to those token embeddings to exclude padding tokens from the mean pooling operation.

Our mean pooling takes the average activation of values across each dimension to produce a single value [Lines: 28-29]. This brings our tensor sizes from (512\*768) to (1\*768). The next step is to concatenate these embeddings.

Concatenation of  $u$ ,  $v$ , and  $|u-v|$  produces a vector three times the length of each original vector [Line: 36]. We perform this concatenation operation using PyTorch.

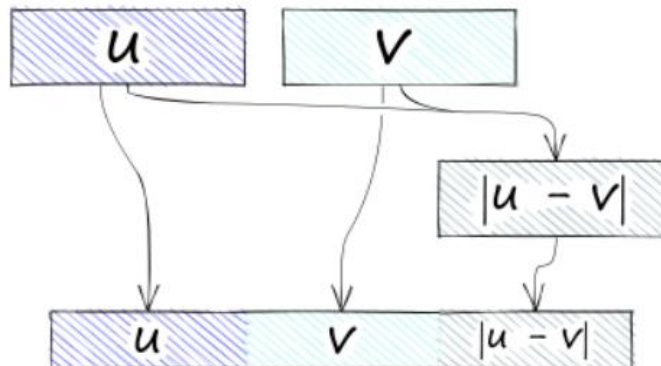


Figure 35: Concatenation of the embeddings  $u$ ,  $v$   $|u - v|$  [17]

---

## PairUp: Detecting Congruous Image and Text Pairings

---

Vector  $(u, v, |u-v|)$  is then fed into a feed-forward neural network (FFNN). The FFNN processes the vector and outputs three activation values. One for each of our label classes; entailment, neutral, and contradiction [Line: 39].

```
01 ffnn = torch.nn.Linear(768*3, 3)
```

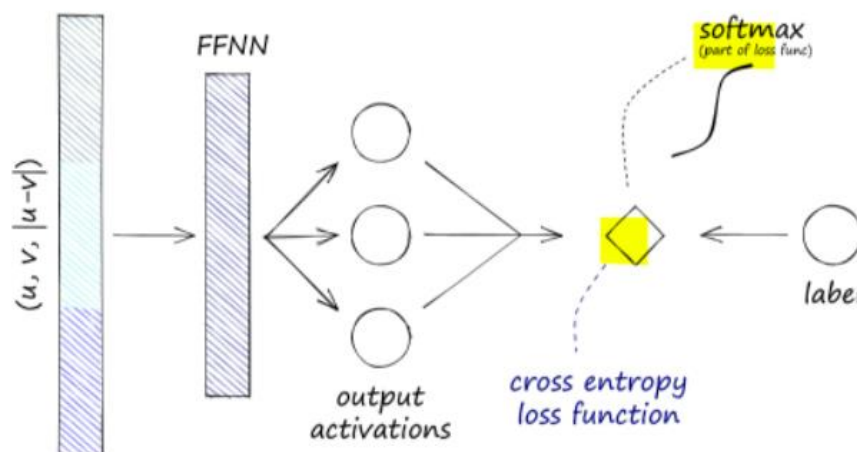


Figure 36: The operations were performed during training on two sentence embeddings,  $u$  and  $v$ . Note that softmax-loss refers to cross-entropy loss (which contains a softmax function by default) [17]

Softmax loss is calculated by applying a softmax function across the three activation values (or nodes), producing a predicted label. We then use cross-entropy loss to calculate the difference between our predicted label and true label [Line: 42].

```
01 loss_func = torch.nn.CrossEntropyLoss()
```

Using the loss, we calculate the gradients and then optimize [Lines: 45-46]. We use an Adam optimizer with a learning rate of  $2e-5$  and a linear warmup period of 10% of the total training data for the optimization function. To set that up, we use the standard PyTorch Adam optimizer alongside a learning rate scheduler provided by HF transformers.

```
01 from transformers.optimization import  
get_linear_schedule_with_warmup  
02  
03 # we would initialize everything first  
04 optim = torch.optim.Adam(model.parameters(), lr=1e-5)  
05 # and setup a warmup for the first ~10% steps  
06 total_steps = int(len(dataset) / batch_size)  
07 warmup_steps = int(0.1 * total_steps)  
08 scheduler = get_linear_schedule_with_warmup(
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
09     optim, num_warmup_steps=warmup_steps,  
10     num_training_steps=total_steps - warmup_steps  
11 )
```

We repeat all the steps discussed above for training with 5 epochs.

```
01 from tqdm.auto import tqdm  
02  
03  
04 TRAINING_MODE = True  
05 epochs = 5  
06  
07 if TRAINING_MODE:  
08     for epoch in range(epochs):  
09         model.train() # make sure model is in training mode  
10         # initialize the dataloader loop with tqdm (tqdm ==  
progress bar)  
11         loop = tqdm(loader, leave=True)  
12         for batch in loop:  
13             # zero all gradients on each new step  
14             optim.zero_grad()  
15  
16             # prepare batches and move all to the active device  
17             inputs_ids_a = batch['premise_input_ids'].to(device)  
18             inputs_ids_b = batch['hypothesis_input_ids'].to(device)  
19             attention_a =  
batch['premise_attention_mask'].to(device)  
20             attention_b =  
batch['hypothesis_attention_mask'].to(device)  
21             label = batch['label'].to(device)  
22  
23             # extract token embeddings from BERT  
24             u = model(inputs_ids_a, attention_mask=attention_a)[0]  
# all token embeddings A  
25             v = model(inputs_ids_b, attention_mask=attention_b)[0]  
# all token embeddings B  
26  
27             # get the mean pooled vectors  
28             u = mean_pool(u, attention_a)  
29             v = mean_pool(v, attention_b)  
30  
31             # build the |u-v| tensor  
32             uv = torch.sub(u, v)  
33             uv_abs = torch.abs(uv)  
34  
35             # concatenate u, v, |u-v|  
36             x = torch.cat([u, v, uv_abs], dim=-1)
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
37
38         # process concatenated tensor through FFNN
39         x = ffnn(x)
40
41         # calculate the 'softmax-loss' between predicted and
true label
42         loss = loss_func(x, label)
43
44         # using loss, calculate gradients and then optimize
45         loss.backward()
46         optim.step()
47
48         # update learning rate scheduler
49         scheduler.step()
50         # update the TDQM progress bar
51         loop.set_description(f'Epoch {epoch}')
52         loop.set_postfix(loss=loss.item())
```

We observed that it takes around 7 hours of Google Colab Pro's time to go over one epoch of the dataset using GPU runtime. The training took almost close to 33 hours for completion. For this reason we only performed minimal training.

### 8.3.2. Custom NLP Model B

For this custom model, we only make use of the transformer model module, followed by a mean pooling module. The custom Sentence Transformer is constructed using a pre-trained uncased BERT model followed by a pooling layer. Here, we perform the fine-tuning using the sentence-transformers approach. It hides a lot of underlying mechanisms and is much easier to handle compared to the standard PyTorch approach as followed for the custom NLP model A.

```
01 from sentence_transformers import SentenceTransformer, models
02
03
04 bert = models.Transformer('bert-base-uncased')
05 pooler = models.Pooling(
06     bert.get_word_embedding_dimension(),
07     pooling_mode_mean_tokens=True
08 )
09
10 model = SentenceTransformer(modules=[bert, pooler])
11 model
```

We define the Softmax loss as below to be used for optimization of our model.

---

# PairUp: Detecting Congruous Image and Text Pairings

---

```
01 from sentence_transformers import losses
02
03
04 loss = losses.SoftmaxLoss(
05     model=model,
06     sentence_embedding_dimension=model.get_sentence_embedding_dimension(),
07     num_labels=3
08 )
```

We then train for 5 epochs and warm up for 10% of training and also save the trained model.

```
01 epochs = 5
02 warmup_steps = int(0.1 * len(dataset))
03 TRAINING_MODE = True
04
05 if TRAINING_MODE:
06     model.fit(
07         train_objectives=[(loader, loss)],
08         epochs=epochs,
09         warmup_steps=warmup_steps,
10         output_path="./output_nlp/sbert_test_b",
11         show_progress_bar=True
12     )
```

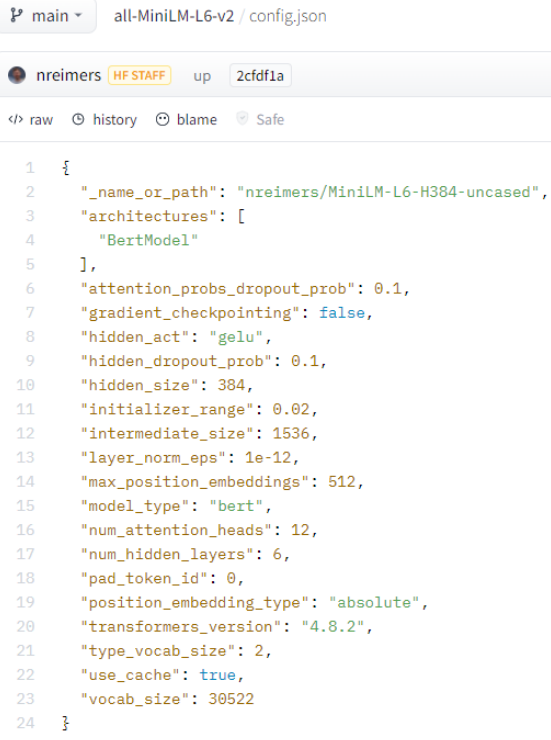
We observed that it takes around 5 hours of Google Colab Pro's time to go over one epoch of the dataset using GPU runtime. The training took almost close to 26 hours for completion. For this reason we performed only minimal training.

## 8.4. PRE-TRAINED SENTENCE SIMILARITY NLP MODELS

PairUp uses in addition to the custom models discussed previously, five popular pre-trained Sentence Similarity NLP Models from Hugging Face (a model repository) [31]. Table 12 describes each of these models and the associated repository on Hugging Face.

<a href="https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2">sentence-transformers/all-MiniLM-L6-v2</a>	It is a sentence-transformers model that maps sentences & paragraphs to a 384 dimensional dense vector space and can be used for tasks like clustering or semantic search.
--	--

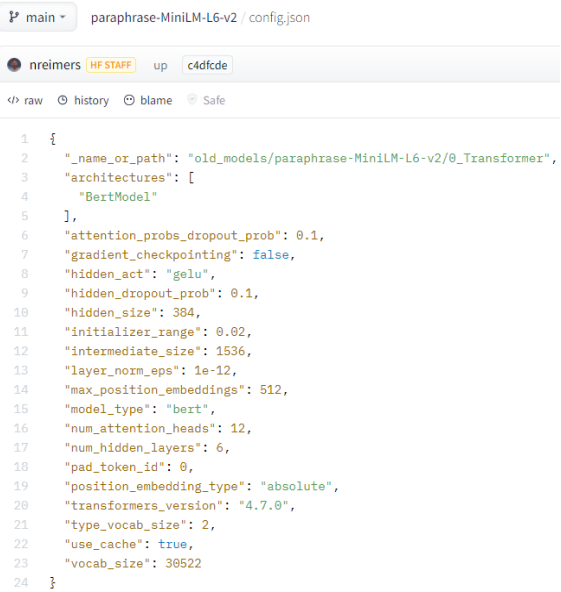
# PairUp: Detecting Congruous Image and Text Pairings

 <pre>1 { 2   "_name_or_path": "nreimers/MiniLM-L6-H384-uncased", 3   "architectures": [ 4     "BertModel" 5   ], 6   "attention_probs_dropout_prob": 0.1, 7   "gradient_checkpointing": false, 8   "hidden_act": "gelu", 9   "hidden_dropout_prob": 0.1, 10  "hidden_size": 384, 11  "initializer_range": 0.02, 12  "intermediate_size": 1536, 13  "layer_norm_eps": 1e-12, 14  "max_position_embeddings": 512, 15  "model_type": "bert", 16  "num_attention_heads": 12, 17  "num_hidden_layers": 6, 18  "pad_token_id": 0, 19  "position_embedding_type": "absolute", 20  "transformers_version": "4.8.2", 21  "type_vocab_size": 2, 22  "use_cache": true, 23  "vocab_size": 30522 24 }</pre>	<p>The model uses the pre-trained “nreimers/MiniLM-L6-H384-uncased” model and is fine-tuned on a 1B sentence pair’s dataset (data from multiple datasets is concatenated to form the 1B pairs). It uses a contrastive learning objective: given a sentence from the pair, the model should predict which out of a set of randomly sampled other sentences, was actually paired with it in our dataset.</p> <p>The model is intended to be used as a sentence and short paragraph encoder. Given an input text, it outputs a vector which captures the semantic information. The sentence vector may be used for information retrieval, clustering or sentence similarity tasks.</p> <p>By default, input text longer than 256 word pieces is truncated.</p>
<p><a href="#">sentence-transformers/paraphrase-MiniLM-L6-v2</a></p>	<p>It is also a sentence-transformers model and is similar to all-MiniLM-L6-v2 but uses the pre-trained “old_models/paraphrase-MiniLM-L6-v2/0_Transformer” BERT model.</p> <p>By default, input text longer than 128 word pieces is truncated.</p>

---

# PairUp: Detecting Congruous Image and Text Pairings


---

 <pre>1 { 2   "_name_or_path": "old_models/paraphrase-MiniLM-L6-v2/0_Transformer", 3   "architectures": [ 4     "BertModel" 5   ], 6   "attention_probs_dropout_prob": 0.1, 7   "gradient_checkpointing": false, 8   "hidden_act": "gelu", 9   "hidden_dropout_prob": 0.1, 10  "hidden_size": 384, 11  "initializer_range": 0.02, 12  "intermediate_size": 1536, 13  "layer_norm_eps": 1e-12, 14  "max_position_embeddings": 512, 15  "model_type": "bert", 16  "num_attention_heads": 12, 17  "num_hidden_layers": 6, 18  "pad_token_id": 0, 19  "position_embedding_type": "absolute", 20  "transformers_version": "4.7.0", 21  "type_vocab_size": 2, 22  "use_cache": true, 23  "vocab_size": 38522 24 }</pre>	
<a href="https://huggingface.co/sentence-transformers/LaBSE">sentence-transformers/LaBSE</a>	<p>The language-agnostic BERT sentence embedding encodes text into high dimensional vectors. The model is trained and optimized to produce similar representations exclusively for bilingual sentence pairs that are translations of each other. So it can be used for mining for translations of a sentence in a larger corpus.</p>

---

# PairUp: Detecting Congruous Image and Text Pairings

---

 <pre>1 { 2   "_name_or_path": "old_models/LaBSE/0_Transformer", 3   "architectures": [ 4     "BertModel" 5   ], 6   "attention_probs_dropout_prob": 0.1, 7   "directionality": "bidi", 8   "gradient_checkpointing": false, 9   "hidden_act": "gelu", 10  "hidden_dropout_prob": 0.1, 11  "hidden_size": 768, 12  "initializer_range": 0.02, 13  "intermediate_size": 3072, 14  "layer_norm_eps": 1e-12, 15  "max_position_embeddings": 512, 16  "model_type": "bert", 17  "num_attention_heads": 12, 18  "num_hidden_layers": 12, 19  "pad_token_id": 0, 20  "pooler_fc_size": 768, 21  "pooler_num_attention_heads": 12, 22  "pooler_num_fc_layers": 3, 23  "pooler_size_per_head": 128, 24  "pooler_type": "first_token_transform", 25  "position_embedding_type": "absolute", 26  "transformers_version": "4.7.0", 27  "type_vocab_size": 2, 28  "use_cache": true, 29  "vocab_size": 501153 30 }</pre>	
<a href="https://huggingface.co/sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2">sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2</a>	It is a sentence-transformers model that maps sentences & paragraphs to a 384 dimensional dense vector space and uses 12 hidden layers.



# PairUp: Detecting Congruous Image and Text Pairings

<div><div>main ▾paraphrase-multilingual-MiniLM-L12-v2 / config.json</div><div>nreimersAdd new SentenceTransformer model. e625097</div><div>raw history blame Safe</div><div><pre>1 { 2   "_name_or_path": "old_models/paraphrase-multilingual-MiniLM-L12-v2/0_Transformer", 3   "architectures": [ 4     "BertModel" 5   ], 6   "attention_probs_dropout_prob": 0.1, 7   "gradient_checkpointing": false, 8   "hidden_act": "gelu", 9   "hidden_dropout_prob": 0.1, 10  "hidden_size": 384, 11  "initializer_range": 0.02, 12  "intermediate_size": 1536, 13  "layer_norm_eps": 1e-12, 14  "max_position_embeddings": 512, 15  "model_type": "bert", 16  "num_attention_heads": 12, 17  "num_hidden_layers": 12, 18  "pad_token_id": 0, 19  "position_embedding_type": "absolute", 20  "transformers_version": "4.7.0", 21  "type_vocab_size": 2, 22  "use_cache": true, 23  "vocab_size": 250037 24 }</pre></div></div>	
<div><div><a href="#">sentence-transformers/bert-base-nli-mean-tokens</a></div><div>main ▾bert-base-nli-mean-tokens / config.json</div><div>nreimersAdd new SentenceTransformer model. 2739dcf</div><div>raw history blame Safe</div><div><pre>1 { 2   "_name_or_path": "old_models/bert-base-nli-mean-tokens/0_BERT", 3   "architectures": [ 4     "BertModel" 5   ], 6   "attention_probs_dropout_prob": 0.1, 7   "gradient_checkpointing": false, 8   "hidden_act": "gelu", 9   "hidden_dropout_prob": 0.1, 10  "hidden_size": 768, 11  "initializer_range": 0.02, 12  "intermediate_size": 3072, 13  "layer_norm_eps": 1e-12, 14  "max_position_embeddings": 512, 15  "model_type": "bert", 16  "num_attention_heads": 12, 17  "num_hidden_layers": 12, 18  "pad_token_id": 0, 19  "position_embedding_type": "absolute", 20  "transformers_version": "4.7.0", 21  "type_vocab_size": 2, 22  "use_cache": true, 23  "vocab_size": 30522 24 }</pre></div></div>	<p>It is a sentence-transformers model that maps sentences &amp; paragraphs to a 768 dimensional dense vector space and can be used for tasks like clustering or semantic search.</p>

Table 12: PairUp uses 5 popular pre-trained models for Text similarity in addition to the 2 custom models developed

---

# PairUp: Detecting Congruous Image and Text Pairings

---

## 8.5. COMPARISON OF MODELS

In order to evaluate the various NLP models, we make use of the validation set from the SNLI dataset which is loaded with the following script.

```
01 from datasets import load_dataset
02
03 snli_val = load_dataset("snli", split="validation")
```

We create a collection of pairs of similar sentences by filtering out the entries with label = 0 and another collection of dissimilar sentences by filtering out the entries with label = 2.

```
01 # print(type(snli_val))
02
03 sentences_similar = []
04 sentences_dissimilar = []
05
06 NUM_ENTRIES = 100
07
08 for i in range(3 * NUM_ENTRIES):
09     # print(snli_val[i])
10     if snli_val[i]['label'] == 0:
11         sentences_similar.append([snli_val[i]['premise'],
snli_val[i]['hypothesis']])
12     elif snli_val[i]['label'] == 2:
13         sentences_dissimilar.append([snli_val[i]['premise'],
snli_val[i]['hypothesis']])
14
15 print(sentences_similar)
16 print(sentences_dissimilar)
17 print(len(sentences_similar))
18 print(len(sentences_dissimilar))
```

For each of the individual models, we create embeddings for each of the sentences.

```
01 sentence_embeddings_similar_dict = {}
02 sentence_embeddings_dissimilar_dict = {}
03
04 for i in range(len(model_names)-1):
05     temp = []
06     for j in range(len(sentences_similar)):
07         temp.append(models[i].encode(sentences_similar[j]))
08     sentence_embeddings_similar_dict[model_names[i]] = temp
09
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
10 for i in range(len(model_names)-1):
11     temp = []
12     for j in range(len(sentences_dissimilar)):
13         temp.append(models[i].encode(sentences_dissimilar[j]))
14     sentence_embeddings_dissimilar_dict[model_names[i]] = temp
15
16 print(f"Number of models: {len(model_names)}")
17 print(len(sentence_embeddings_similar_dict[model_names[0]]))
18 print(len(sentence_embeddings_dissimilar_dict[model_names[0]]))
```

To evaluate our custom model B, we define the mean pooling function and build embeddings and calculate cosine similarity using the below functions:

```
01 cos_sim = torch.nn.CosineSimilarity()
02
03
04 def mean_pool(token_embeds, attention_mask):
05     # reshape attention_mask to cover 768-dimension embeddings
06     in_mask = attention_mask.unsqueeze(-1).expand(
07         token_embeds.size()
08     ).float()
09     # perform mean-pooling but exclude padding tokens (specified by
10     in_mask)
11     pool = torch.sum(token_embeds * in_mask, 1) / torch.clamp(
12         in_mask.sum(1), min=1e-9
13     )
14     return pool
15
16 # build embeddings and calculate cosine similarity
17 def sts_process(sentence_a, sentence_b, model):
18     vecs = [] # init list of sentence vecs
19     for sentence in [sentence_a, sentence_b]:
20         # build input_ids and attention_mask tensors with tokenizer
21         input_ids = tokenizer(
22             sentence, max_length=512, padding='max_length',
23             truncation=True, return_tensors='pt'
24         )
25         # process tokens through model and extract token embeddings
26         token_embeds = model(**input_ids).last_hidden_state
27         # mean-pool token embeddings to create sentence embeddings
28         sentence_embeds = mean_pool(token_embeds,
29             input_ids['attention_mask'])
30     vecs.append(sentence_embeds)
31     # calculate cosine similarity between pairs and return numpy
32     array
33     return cos_sim(vecs[0], vecs[1]).detach().numpy()
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

We then compute the sentence similarity and dissimilarity scores for each of the similar and dissimilar pairs using cosine similarity.

```
01 similarity_score_dict = {}
02
03 for i in range(len(model_names)):
04     similarity_score_list = []
05
06     for j in range(len(sentences_similar)):
07
08         if i == (len(model_names) - 1):
09             similarity_score = sts_process(sentences_similar[j][0],
10 sentences_similar[j][1], model)[0]
11         else:
12             similarity_score =
13 cosine_similarity(sentence_embeddings_similar_dict[model_names[i]][j][
14 0].reshape(1, -1),
15 sentence_embeddings_similar_dict[model_names[i]][j][1].reshape(1, -
16 1))[0][0]
17             similarity_score_list.append(similarity_score)
18             # pprint('Similarity between "{}\n" and "{}\n" is {} using
19 model: {}'.format(sentences_similar[j][0],
20 #     sentences_similar[j][1],
21 #     similarity_score,
22 #     model_names[i]))
23             similarity_score_dict[model_names[i]] = similarity_score_list
24
25 print(similarity_score_dict)
26
27 # -----
28
29 dissimilarity_score_dict = {}
30
31 for i in range(len(model_names)):
32     dissimilarity_score_list = []
33
34     for j in range(len(sentences_dissimilar)):
35
36         if i == (len(model_names) - 1):
37             dissimilarity_score = sts_process(sentences_dissimilar[j][0],
38 sentences_dissimilar[j][1], model)[0]
39         else:
40             dissimilarity_score =
41 cosine_similarity(sentence_embeddings_dissimilar_dict[model_names[i]][
42 j][0].reshape(1, -1),
43 sentence_embeddings_dissimilar_dict[model_names[i]][j][1].reshape(1, -
44 1))[0][0]
45             dissimilarity_score_list.append(dissimilarity_score)
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
33     # pprint('Similarity between "{}\n" and "{}\n" is {} using
model: {}'.format(sentences_dissimilar[j][0],
34     #     sentences_dissimilar[j][1],
35     #     dissimilarity_score,
36     #     model_names[i]))
37     dissimilarity_score_dict[model_names[i]] =
dissimilarity_score_list
38
39 print(dissimilarity_score_dict)
```

We use Pandas's DataFrame data structure to organize the similarity and dissimilarity scores.

```
01 import pandas as pd
02
03 df_similar = pd.DataFrame(similarity_score_dict)
04 df_dissimilar = pd.DataFrame(dissimilarity_score_dict)
```

Table 13 shows the similarity scores obtained for the different models. The higher the score is better since it means the model is able to assign a higher cosine similarity score to the sentences that are similar.

	./nlp_models/all-MiniLM-l6-v2	./nlp_models/paraphrase-MiniLM-l6-v2	./nlp_models/LaBSE	./nlp_models/paraphrase-multilingual-MiniLM-l12-v2	./nlp_models/bert-base-mlm-mean-tokens	./output_nlp/sbert_test_b	./output_nlp/sbert_test_a
0	0.761516	0.802448	0.819890	0.775430	0.864785	0.771428	0.858764
1	0.727639	0.787500	0.602852	0.840839	0.678445	0.762457	0.792599
2	0.783048	0.854760	0.598355	0.927094	0.845174	0.825220	0.886017
3	0.803669	0.777568	0.500152	0.787946	0.855469	0.819836	0.868879
4	0.599873	0.684842	0.500475	0.674292	0.673711	0.650059	0.532525
...	...	...	...	...	...	...	...
91	0.704088	0.733395	0.621464	0.792933	0.685196	0.626790	0.837693
92	0.571842	0.556694	0.623473	0.502408	0.418696	0.401750	0.394299
93	0.614979	0.581915	0.705548	0.625909	0.673605	0.566768	0.888389
94	0.584636	0.467268	0.588640	0.475142	0.676764	0.785215	0.712294
95	0.778371	0.760970	0.525313	0.736112	0.727106	0.762494	0.909202
96 rows x 7 columns							

Table 13: model comparison on sentences with high similarity

Table 14 shows the dissimilarity scores obtained for the sentences with different models. The lower the score is better since it means the model is able to assign a lower cosine similarity score to the sentences that are dissimilar. The results are also presented in graph form in Figure 37.

Figure 37 shows a model comparison chart plotting the 7 NLP similarity models used in PairUp against their mean cosine similarity scores. This was generated using the following code.

```
01 ax = df_comb.plot(kind='barh', figsize=(10, 10), title="NLP Model
Performance vs Cosine Similarity Scores", ylabel="Cosine Similarity
Score", fontsize=14)
02 ax.title.set_size(20)
03 ax.set_xlabel("Cosine Similarity Scores", fontsize=20)
```

# PairUp: Detecting Congruous Image and Text Pairings

```
04 ax.set_ylabel("NLP Models", fontsize=20)
```

The blue bar in Figure 37 shows similarity scores for similar sentences, the higher the score indicates that the model is able to identify that the sentences are similar. Conversely, the orange bar shows similarity scores for dissimilar sentences, the lower the score indicates that the model is able to identify that the sentences are dissimilar.

Figure 37 indicates that custom model A (sbert\_test\_a) performs the best among all models since it assigns a high score for similar sentences while a low one for dissimilar ones. We believe this behavior stems from the model's expressive architecture since it make use of vectors ( $u$ ,  $v$ ,  $|u-v|$ ) which then passed through a feed forward network allowing for better understanding of the context of sentence similarity.

Among the pre-trained models, the "paraphrase-multilingual-MiniLM-L12-v2" performs better than the others, most likely attributed to its use of 12 hidden layers which is able to capture the underlying sentence similarity context better than the rest.

Table 15 shows the mean cosine similarity score of similar and dissimilar sentences for each of the models.

As discussed at the start of section 8, a majority rule voting scheme using the cosine similarity score output by each of PairUp's seven models is employed. The empirically chosen threshold of 0.5 is used to separate congruous and non-congruous text pairs.

	./nlp_models/all-MiniLM-L6-v2	./nlp_models/paraphrase-MiniLM-L6-v2	./nlp_models/LaBSE	./nlp_models/paraphrase-multilingual-MiniLM-L12-v2	./nlp_models/bert-base-nli-mean-tokens	./output_nlp/sbert_test_b	./output_nlp/sbert_test_a
0	0.074186	0.004630	0.238242	-0.033536	0.007217	-0.054761	-0.485240
1	0.330629	0.390224	0.516300	0.381650	0.238097	0.282341	-0.024343
2	-0.018999	0.042277	0.209647	0.054110	0.373043	0.453697	0.366036
3	-0.003865	-0.051531	0.152060	-0.055300	0.091823	0.132740	-0.006588
4	0.299173	0.395348	0.420800	0.446505	0.550699	0.389955	0.645899
...	...	...	...	...	...	...	...
98	0.119102	0.164458	0.401571	0.162299	0.137026	0.163437	0.055310
99	0.420295	0.485038	0.605058	0.452142	0.528520	0.497263	0.718917
100	0.319567	0.361291	0.444456	0.164982	0.287144	0.342377	0.182060
101	0.588081	0.702202	0.666743	0.703631	0.467337	0.620370	0.780477
102	0.035823	-0.083932	0.275417	-0.058952	0.190753	0.186204	-0.300915

103 rows x 7 columns

Table 14: model comparison on sentences with low similarity

```
01 comb_data = {"Mean Cosine similarity score of similar sentences" :  
df_similar.mean(),  
02               "Mean Cosine similarity score of dissimilar sentences"  
: df_dissimilar.mean()}  
03 df_comb = pd.DataFrame(comb_data)  
04 df_comb
```

# PairUp: Detecting Congruous Image and Text Pairings

	Mean Cosine similarity score of similar sentences	Mean Cosine similarity score of dissimilar sentences
./nlp_models/all-MiniLM-L6-v2	0.668136	0.303063
./nlp_models/paraphrase-MiniLM-L6-v2	0.694677	0.304168
./nlp_models/LaBSE	0.572388	0.445516
./nlp_models/paraphrase-multilingual-MiniLM-L12-v2	0.711657	0.311254
./nlp_models/bert-base-nli-mean-tokens	0.695206	0.331151
./output_nlp/sbert_test_b	0.668554	0.348047
./output_nlp/sbert_test_a	0.715974	0.231565

Table 15: model comparison for both similar and dissimilar sentence pairings using mean cosine similarity metric

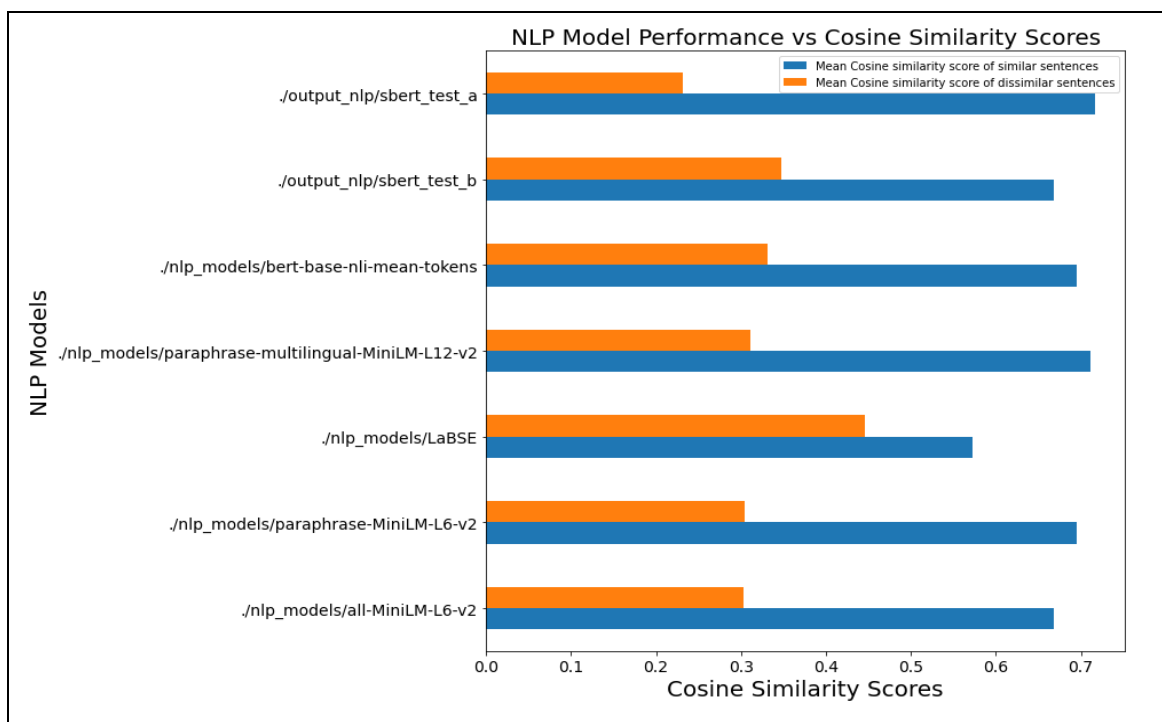


Figure 37: Model comparison chart for all 7 models used in PairUp.

## 9. INTEGRATION OF IMAGE-TO-TEXT SYNTHESIS MODULE WITH TEXTUAL SIMILARITY ANALYZER

This section describes the steps involved in combining the two main components of the PairUp system: the Image to Text Synthesizer and the Text Similarity Analyzer. The output from image to text synthesis module i.e. the predicted caption is passed onto the text similarity analyzer block which also accepts the user input text and determines the congruousness of the image-text pairing.

---

## PairUp: Detecting Congruous Image and Text Pairings

---

We import the below Python packages.

```
01 import tensorflow as tf
02 import matplotlib.pyplot as plt
03 import collections
04 import random
05 import numpy as np
06 import os
07 import time
08 import json
09 from PIL import Image
10
11 print(np.__version__)
12 print(tf.version.VERSION)
```

We load the best encoder and decoder models obtained from Section 7.1 to perform the image to text synthesis operation.

```
01 encoder_loaded =
tf.saved_model.load('output_20k_150_epoch/encoder_model')
02 decoder_loaded =
tf.saved_model.load('output_20k_150_epoch/decoder_model')
```

We also make use of additional helper functions below to load the image and standardize the inputs such that the input image can be processed and passed onto the encoder model. The decoder model needs access to the tokenizer to construct words from the encoder predictions.

```
01 max_length = 50
02 attention_features_shape = 64
03
04 def load_image(image_path):
05     img = tf.io.read_file(image_path)
06     img = tf.io.decode_jpeg(img, channels=3)
07     img = tf.keras.layers.Resizing(299, 299)(img)
08     img = tf.keras.applications.inception_v3.preprocess_input(img)
09     return img, image_path
10
11 image_model = tf.keras.applications.InceptionV3(include_top=False,
weights='imagenet')
12
13 new_input = image_model.input
14 hidden_layer = image_model.layers[-1].output
15
```



---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
16 image_features_extract_model = tf.keras.Model(new_input,
hidden_layer)
17
18
19 def standardize(inputs):
20     inputs = tf.strings.lower(inputs)
21     return tf.strings.regex_replace(inputs,
22                                     r"!\"#$%&\(\)\*\+\.,-
/;:=?@[\\]\^_`{|}~", "")
23
24 # https://stackoverflow.com/questions/65103526/how-to-save-
textvectorization-to-disk-in-tensorflow
25 import pickle
26
27 from_disk =
pickle.load(open("output_20k_150_epoch/tokenizer.pickle", "rb"))
28 tokenizer =
tf.keras.layers.TextVectorization.from_config(from_disk['config'])
29 # You have to call `adapt` with some dummy data (BUG in Keras)
30 tokenizer.adapt(tf.data.Dataset.from_tensor_slices(["xyz"]))
31 tokenizer.set_weights(from_disk['weights'])
32
33 # # Create mappings for words to indices and indicies to words.
34 word_to_index = tf.keras.layers.StringLookup(
35     mask_token="",
36     vocabulary=tokenizer.get_vocabulary())
37 index_to_word = tf.keras.layers.StringLookup(
38     mask_token="",
39     vocabulary=tokenizer.get_vocabulary(),
40     invert=True)
```

In order to generate the caption for the new input image, we have created an `evaluate_loaded` function as below. The input to the decoder at each time step is its previous predictions along with the hidden state and the encoder output. The model stops predicting when it encounters an “end” token. It also stores the attention weights for every time step.

```
01 def evaluate_loaded(image):
02     attention_plot = np.zeros((max_length,
attention_features_shape))
03
04     # hidden = decoder_loaded.reset_state(batch_size=1)
05     hidden = tf.zeros((1, 512))
06
07     temp_input = tf.expand_dims(load_image(image)[0], 0)
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
08     img_tensor_val = image_features_extract_model(temp_input)
09     img_tensor_val = tf.reshape(img_tensor_val,
10                                (img_tensor_val.shape[0],
11                                 -1,
12                                 img_tensor_val.shape[3]))
13     features = encoder_loaded(img_tensor_val)
14     dec_input = tf.expand_dims([word_to_index('<start>')], 0)
15     result = []
16
17     for i in range(max_length):
18         predictions, hidden, attention_weights =
decoder_loaded(dec_input,
19                                                         features,
20                                                         hidden)
21
22         attention_plot[i] = tf.reshape(attention_weights, (-1,
23 ))).numpy()
24         predicted_id = tf.random.categorical(predictions,
25 1)[0][0].numpy()
26         predicted_word =
tf.compat.as_text(index_to_word(predicted_id).numpy())
27         result.append(predicted_word)
28
29         if predicted_word == '<end>':
30             return result, attention_plot
31
32         dec_input = tf.expand_dims([predicted_id], 0)
33         dec_input = tf.cast(dec_input, tf.int64)
34
35     attention_plot = attention_plot[:len(result), :]
36     return result, attention_plot
```

We also use a `plot_attention` function to generate a plot of the generated words and identify its corresponding portion from the image.

```
01 def plot_attention(image, result, attention_plot):
02     temp_image = np.array(Image.open(image))
03
04     fig = plt.figure(figsize=(10, 10))
05
06     len_result = len(result)
07     for i in range(len_result):
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
08     temp_att = np.resize(attention_plot[i], (8, 8))
09     grid_size = max(int(np.ceil(len_result/2)), 2)
10     ax = fig.add_subplot(grid_size, grid_size, i+1)
11     ax.set_title(result[i])
12     img = ax.imshow(temp_image)
13     ax.imshow(temp_att, cmap='gray', alpha=0.6,
extent=img.get_extent())
14
15     plt.tight_layout()
16     plt.show()
```

We create a widget “Choose Files” which allows a user to upload an image for which the caption would be generated.

```
01 # https://medium.com/@rk.sarthak01/how-to-import-files-images-in-
google-colab-from-your-local-system-46a801b1e568
02 from google.colab import files
03 import cv2
04 import matplotlib.pyplot as plt
05 import matplotlib.image as mpimg
06 import os
07
08 uploaded = files.upload()
09
10 # print(list(uploaded.keys()))
11
12 image_file = list(uploaded.keys())[0]
13 image_path = os.path.abspath('.') + "/" + image_file
14 print(image_path)
```

The code below helps to generate the caption of the inputted image and generate the attention plot.

```
01 result, attention_plot = evaluate_loaded(image_path)
02 plot_attention(image_path, result, attention_plot)
03 output = ' '.join(result[:-1])
04 print(f"Predicted Caption: {output}")
05
06 # opening the image
07 Image.open(image_path)
```

We install the sentence\_transformers framework to work with the various sentence transformer models.

```
01 !pip install -q sentence_transformers
02
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
03 from sentence_transformers import SentenceTransformer
04 from sklearn.metrics.pairwise import cosine_similarity
05 from pprint import pprint
```

We load the various NLP models (pre-defined ones and custom NLP model B) obtained from section 7.2

```
01 # https://huggingface.co/models?pipeline_tag=sentence-
similarity&sort=downloads
02
03 nlp_models_path = "./nlp_models/"
04
05 # model_names = ["all-MiniLM-L6-v2",
06 #               "paraphrase-MiniLM-L6-v2",
07 #               "LaBSE",
08 #               "paraphrase-multilingual-MiniLM-L12-v2",
09 #               "bert-base-nli-mean-tokens"]
10
11 model_names = [nlp_models_path + "all-MiniLM-L6-v2",
12               nlp_models_path + "paraphrase-MiniLM-L6-v2",
13               nlp_models_path + "LaBSE",
14               nlp_models_path + "paraphrase-multilingual-MiniLM-
L12-v2",
15               nlp_models_path + "bert-base-nli-mean-tokens",
16               "./output_nlp/sbert_test_b"]
17 models = []
18
19 for i in range(len(model_names)):
20     model = SentenceTransformer(model_names[i])
21     # model.save(nlp_models_path + model_names[i])
22     models.append(model)
```

Since, the custom NLP model A relies on using the tokenizer and model separately, we use the below to load it. To evaluate the model, we also make use of the same mean\_pool and sts\_process functions discussed earlier to perform pooling operations and calculate cosine similarity respectively.

```
01 from transformers import BertTokenizer, BertModel
02
03 model_path = './output_nlp/sbert_test_a'
04
05 tokenizer = BertTokenizer.from_pretrained(model_path)
06 model = BertModel.from_pretrained(model_path)
07
08
09 import numpy as np
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
10 import torch
11
12 cos_sim = torch.nn.CosineSimilarity()
13
14
15 def mean_pool(token_embeds, attention_mask):
16     # reshape attention_mask to cover 768-dimension embeddings
17     in_mask = attention_mask.unsqueeze(-1).expand(
18         token_embeds.size())
19     ).float()
20     # perform mean-pooling but exclude padding tokens (specified by
21     in_mask)
22     pool = torch.sum(token_embeds * in_mask, 1) / torch.clamp(
23         in_mask.sum(1), min=1e-9
24     )
25     return pool
26
27 # build embeddings and calculate cosine similarity
28 def sts_process(sentence_a, sentence_b, model):
29     vecs = [] # init list of sentence vecs
30     for sentence in [sentence_a, sentence_b]:
31         # build input_ids and attention_mask tensors with tokenizer
32         input_ids = tokenizer(
33             sentence, max_length=512, padding='max_length',
34             truncation=True, return_tensors='pt'
35         )
36         # process tokens through model and extract token embeddings
37         token_embeds = model(**input_ids).last_hidden_state
38         # mean-pool token embeddings to create sentence embeddings
39         sentence_embeds = mean_pool(token_embeds,
40             input_ids['attention_mask'])
41         vecs.append(sentence_embeds)
42     # calculate cosine similarity between pairs and return numpy
43     array
44     return cos_sim(vecs[0], vecs[1]).detach().numpy()
```

We take input from the user about the textual description of the scene. This is the text which would be used to decide whether it can be paired with the input image.

```
01 input_text = input("Enter the sentence: ")
02 print(f"You have entered: {input_text}")
```

The generated caption of the input image (say: output) from section 7.1 and the input text from the previous step (say: input\_text) is provided as inputs to the various NLP models.

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
01 sentences = [output, input_text]
02 print(sentences)
```

We create sentence embeddings for each of the NLP models with the pair of sentences from the last step.

```
01 sentence_embeddings_list = []
02
03 for i in range(len(model_names)):
04     sentence_embeddings = models[i].encode(sentences)
05     sentence_embeddings_list.append(sentence_embeddings)
```

We calculate the cosine similarity scores for the various models.

```
01 similarity_score_list = []
02
03 for i in range(len(model_names)):
04     similarity_score =
cosine_similarity(sentence_embeddings_list[i][0].reshape(1, -1),
sentence_embeddings_list[i][1].reshape(1, -1))[0][0]
05     similarity_score_list.append(similarity_score)
06     pprint('Similarity between {} and {} is {} using model:
{}'.format(sentences[0],
07           sentences[1],
08           similarity_score,
09           model_names[i]))
10
11 similarity_score_list.append(sts_process(sentences[0], sentences[1],
model)[0])
12 pprint('Similarity between {} and {} is {} using model:
{}'.format(sentences[0],
13           sentences[1],
14           similarity_score_list[-1],
15           model_path))
```

Based on the obtained cosine similarity scores, this PairUp system determines if the generated caption and the input text are congruous or not. This is done through a Voting mechanism i.e. If the majority of the models assigns a score  $\geq 0.5$ , then it is deemed as “congruous” else the image and text pairings are deemed as “incongruous”.

```
01 congruent_count = 0
02
03 for i in range(len(similarity_score_list)):
04     if similarity_score_list[i] >= 0.5:
05         congruent_count += 1
06
```

---

## PairUp: Detecting Congruous Image and Text Pairings

---

```
07 # print(congruent_count)
08 # print(len(similarity_score_list) - congruent_count)
09 print("===== FINAL RESULTS
=====")
10 if congruent_count > (len(similarity_score_list) -
congruent_count):
11     print("OUTPUT: Image and Text pairing are congruous")
12 else:
13     print("OUTPUT: Image and Text pairing are NOT congruous")
14
15 print("=====
=====")
16 print(f"Predicted caption: {output}")
17 print(f"INPUT text: {input_text}")
18 print("INPUT Image: ")
19 Image.open(image_path)
```

### 10. RESULTS

Our PairUp system as shown in Figure 12 consists of the major components of the Image to Text Synthesizer and the Text Similarity Analyzer. Table 10 showed the metrics/ results for the Image to Text Synthesizer and Figure 37 showed the metrics/ results for the Text Similarity Analyzer. In this section, typical results obtained from the entire PairUp system are demonstrated. Ten novel proposed image and text pairings are presented to the PairUp system to determine if they are congruous (matching in similarity) or not. In Table 16 each of the 10 pairs are shown followed by the determination of the multiple models implemented in the Text Similarity Analyzer. Discussion of each of the ten pairings is presented in Table 16. While these are anecdotal findings, there is no approved metric for PairUp beyond what was presented on the individual components in Table 10 and Figure 37. Typically results are given as comparison to human perceived ratings on the pairings and while we have not given numerical values to this, the comments in each of the ten pairings serve as both human perceived ratings as well as comments on the performance of PairUp.

What is interesting is that even if there are wrong elements in some of the synthesized text, the main semantic concept may be correct enough to be considered a congruent pair. Consider pair number 1 where the PairUp says the input text “I see a fire hydrant” is close enough in meaning to the input image which PairUp translates it to “a red and yellow fire hydrant on it and against a fence”. The fence in the background is actually a building with rows of windows at ground level that repeat across that visually could be construed to be a fence. As the fence is not the main object but rather the fire hydrant, the pairing is “fairly” congruent.


---

## PairUp: Detecting Congruous Image and Text Pairings

---

Pairs 1-3 and 5-10 are successfully labeled by PairUp as either congruous or not congruous. However, pair number 4 is a total failure given the output from PairUp's Image to Text Synthesizer. These are good results yielding 90% correct results.

First the number of test cases, 10 is way too little to generalize how well PairUp will perform and more evaluation needs to be performed. Also even in the correct result cases, there were cases of synthesized text that are not ideal and this suggests that this Image to Text Synthesizer component should be further developed.

Serial No.	1
Scenario	Provide a text that is semantically similar to the input image
File:	<code>./results/image_captioning_load_model_text_similarity_results1.ipynb</code>
Input Image	<div>image1.jpg</div> 
Input Text	I see a fire hydrant
Predicted caption from Image captioning module	a red and yellow fire hydrant on it and against a fence



---

## PairUp: Detecting Congruous Image and Text Pairings

---

Sentence similarity scores	'Similarity score: 0.685952365398407 using model: ./nlp_models/all-MiniLM-L6-v2' ( 'Similarity score: 0.6085982322692871 using model: ' ./nlp_models/paraphrase-MiniLM-L6-v2') 'Similarity score: 0.3544763922691345 using model: ./nlp_models/LaBSE' ( 'Similarity score: 0.7212551236152649 using model: ' ./nlp_models/paraphrase-multilingual-MiniLM-L12-v2') ( 'Similarity score: 0.7824613451957703 using model: ' ./nlp_models/bert-base-nli-mean-tokens') 'Similarity score: 0.7693963050842285 using model: ./output_nlp/sbert_test_b' 'Similarity score: 0.7527866959571838 using model: ./output_nlp/sbert_test_a'
Final Output	===== FINAL RESULTS ===== OUTPUT: Image and Text pairing are congruous =====
Comments	The predicted caption confuses the background of the building with the fence but is still able to identify the fire hydrant in the foreground. Since, the input text is similar to the predicted caption, all the NLP models except LaBSE outputs a high similarity score.

Serial No.	2
Scenario	Provide an input image that is somewhat noisy but the input text clearly explains the image
File:	./results/image_captioning_load_model_text_similarity_results2.ipynb
Input Image	image2.jpg

---

## PairUp: Detecting Congruous Image and Text Pairings

---


	
Input Text	a dog is standing on the floor and looking at me
Predicted caption from Image captioning module	a dog walking on a white and white object.
Sentence similarity scores	'Similarity score: 0.37771075963974 using model: ./nlp_models/all-MiniLM-L6-v2' ( 'Similarity score: 0.4452994465827942 using model: ' ./nlp_models/paraphrase-MiniLM-L6-v2') 'Similarity score: 0.4699344038963318 using model: ./nlp_models/LaBSE' ( 'Similarity score: 0.5165355205535889 using model: ' ./nlp_models/paraphrase-multilingual-MiniLM-L12-v2') ( 'Similarity score: 0.5982468724250793 using model: ' ./nlp_models/bert-base-nli-mean-tokens') 'Similarity score: 0.5994710922241211 using model: ./output_nlp/sbert_test_b' 'Similarity score: 0.5388370156288147 using model: ./output_nlp/sbert_test_a'

---

## PairUp: Detecting Congruous Image and Text Pairings

---

Final Output	<pre>===== FINAL RESULTS ===== OUTPUT: Image and Text pairing are congruous ===== =====</pre>
Comments	Even though the input image is somewhat noisy, the predicted caption still identifies it as a dog even though it fails to recognize the background as a floor. Most of the NLP models assign a similarity score $\geq 0.5$ but since the input text is not an exact match of the predicted caption, the scores tend to be slightly on the lower-highs.

Serial No.	3
Scenario	Provide a text that is semantically different from the input image
File:	<code>./results/image_captioning_load_model_text_similarity_results3.ipynb</code>
Input Image	<p>image2.jpg</p> 

---

## PairUp: Detecting Congruous Image and Text Pairings

---


Input Text	I see a cat
Predicted caption from Image captioning module	a dog standing on a white and silver plate on a metal surface
Sentence similarity scores	('Similarity score: 0.23163682222366333 using model: './nlp_models/all-MiniLM-L6-v2') ( 'Similarity score: 0.1715337187051773 using model: './nlp_models/paraphrase-MiniLM-L6-v2') 'Similarity score: 0.3256340026855469 using model: ./nlp_models/LaBSE' ( 'Similarity score: 0.10926523059606552 using model: './nlp_models/paraphrase-multilingual-MiniLM-L12-v2') ( 'Similarity score: -0.02365039475262165 using model: './nlp_models/bert-base-nli-mean-tokens') 'Similarity score: 0.13404245674610138 using model: ./output_nlp/sbert_test_b' 'Similarity score: -0.22792141139507294 using model: ./output_nlp/sbert_test_a'
Final Output	===== FINAL RESULTS ===== OUTPUT: Image and Text pairing are <b>NOT congruous</b> =====
Comments	Since the input text is completely different to the image, the NLP models assign very low similarity scores and the final output of the PairUp system correctly determines the image and text pairing as "incongruous".

Serial No.	4
Scenario	Provide an input image that is suffering from occlusion i.e. the subject is getting hidden by something
File:	./results/image_captioning_load_model_text_similarity_results4.ipynb
Input Image	image3.jpg

---

## PairUp: Detecting Congruous Image and Text Pairings

---


	
Input Text	a man is putting a luggage on a girl's head
Predicted caption from Image captioning module	[UNK] is helping banana hand out on their head.
Sentence similarity scores	('Similarity score: 0.16791044175624847 using model: ' ./nlp_models/all-MiniLM-L6-v2') ( 'Similarity score: 0.41926953196525574 using model: ' ./nlp_models/paraphrase-MiniLM-L6-v2') 'Similarity score: 0.2949058711528778 using model: ./nlp_models/LaBSE' ( 'Similarity score: 0.2729232907295227 using model: ' ./nlp_models/paraphrase-multilingual-MiniLM-L12-v2') ( 'Similarity score: 0.429276704788208 using model: ' ./nlp_models/bert-base-nli-mean-tokens') 'Similarity score: 0.3201252222061157 using model: ./output_nlp/sbert_test_b' 'Similarity score: 0.28150954842567444 using model: ./output_nlp/sbert_test_a'
Final Output	===== FINAL RESULTS ===== OUTPUT: Image and Text pairing are <b>NOT congruous</b>

---

## PairUp: Detecting Congruous Image and Text Pairings

---

	=====
Comments	The predicted caption generates incorrect results. We see the presence of [UNK] tokens. Now, this may be due to the fact that the input image is too difficult for the captioning model to predict the correct text. The input image suffers from occlusion since the hand of the man hides the face of the girl. Even though we provide a good description of the image, the system still flags it as incongruous. This behavior exposes the current limitation of our model that even though the downstream text is correct, if the predicted caption is incorrect, the overall system fails and generates incorrect results.

Serial No.	5
File:	./results/image_captioning_load_model_text_similarity_results5.ipynb
Input Image	
Input Text	I see a baseball player
Predicted caption from Image captioning module	a boy wearing a black baseball uniform are standing in a baseball game.
Sentence similarity scores	('Similarity score: 0.5450605154037476 using model: '

---

## PairUp: Detecting Congruous Image and Text Pairings

---

	<pre>'./nlp_models/all-MiniLM-L6-v2') ('Similarity score: 0.7109076976776123 using model: ' './nlp_models/paraphrase-MiniLM-L6-v2') 'Similarity score: 0.42547422647476196 using model: ./nlp_models/LaBSE' ('Similarity score: 0.6771198511123657 using model: ' './nlp_models/paraphrase-multilingual-MiniLM- L12-v2') ('Similarity score: 0.6399741768836975 using model: ' './nlp_models/bert-base-nli-mean-tokens') 'Similarity score: 0.5699881315231323 using model: ./output_nlp/sbert_test_b' 'Similarity score: 0.7610739469528198 using model: ./output_nlp/sbert_test_a'</pre>
Final Output	<pre>===== FINAL RESULTS ===== OUTPUT: Image and Text pairing are congruous ===== ===</pre>
Comments	<p>The predicted caption generated for this case correctly captures the image and when we provide a much simpler text input, the PairUp system generates fairly high scores, and hence determines the image-text pairing is congruent.</p>


Serial No.	6
File:	<pre>./results/image_captioning_load_model_text_simila rity_results6.ipynb</pre>



---

## PairUp: Detecting Congruous Image and Text Pairings

---

Input Image	
Input Text	a person is skiing on snow
Predicted caption from Image captioning module	a person on skis heading to snow covered
Sentence similarity scores	('Similarity score: 0.7561790943145752 using model: ' ./nlp_models/all-MiniLM-L6-v2') ( 'Similarity score: 0.7207747101783752 using model: ' ./nlp_models/paraphrase-MiniLM-L6-v2') 'Similarity score: 0.7275044918060303 using model: ./nlp_models/LaBSE' ( 'Similarity score: 0.7639342546463013 using model: ' ./nlp_models/paraphrase-multilingual-MiniLM-L12-v2') ( 'Similarity score: 0.861535906791687 using model: ' ./nlp_models/bert-base-nli-mean-tokens') 'Similarity score: 0.8027735352516174 using model: ./output_nlp/sbert_test_b' 'Similarity score: 0.9115321636199951 using model: ./output_nlp/sbert_test_a'




---

## PairUp: Detecting Congruous Image and Text Pairings

---

Final Output	<pre>===== FINAL RESULTS ===== OUTPUT: Image and Text pairing are congruous ===== =====</pre>
Comments	The predicted caption correctly describes the image. It identifies that there is a person who is skiing and when we provide a text that is almost the same as what is being predicted, we see very high cosine similarity scores from all the models.

Serial No.	7
File:	<code>./results/image_captioning_load_model_text_similarity_results7.ipynb</code>
Input Image	
Input Text	I see a bed
Predicted caption from Image captioning module	this is a bed, bed near lamps on the bed.
Sentence similarity scores	<pre>'Similarity score: 0.669435977935791 using model: ./nlp_models/all-MiniLM-L6-v2' ('Similarity score: 0.6811484098434448 using model: ' './nlp_models/paraphrase-MiniLM-L6-v2') 'Similarity score: 0.3005426526069641 using model: ./nlp_models/LaBSE'</pre>

---

## PairUp: Detecting Congruous Image and Text Pairings

---


	<pre>('Similarity score: 0.6992238759994507 using model: ' './nlp_models/paraphrase-multilingual-MiniLM- L12-v2') ('Similarity score: 0.8993604183197021 using model: ' './nlp_models/bert-base-nli-mean-tokens') 'Similarity score: 0.87471604347229 using model: ./output_nlp/sbert_test_b' 'Similarity score: 0.8453609347343445 using model: ./output_nlp/sbert_test_a'</pre>
Final Output	<pre>===== FINAL RESULTS ===== OUTPUT: Image and Text pairing are congruous ===== ===</pre>
Comments	<p>From the predicted caption, we see that it identifies there is a bed in the picture along with lamps but the generated text is not entirely grammatically correct. However, when we input a bed, which is a key component of the image, the PairUp system is able to understand that it is semantically similar and classifies the image-text pairing as congruous.</p>

Serial No.	8
File:	./results/image_captioning_load_model_text_similarity_results8.ipynb

---

## PairUp: Detecting Congruous Image and Text Pairings

---


Input Image	
Input Text	A man is playing baseball
Predicted caption from Image captioning module	a pitcher during a baseball game.
Sentence similarity scores	('Similarity score: 0.7083694338798523 using model: './nlp_models/all-MiniLM-L6-v2') ( 'Similarity score: 0.669160008430481 using model: './nlp_models/paraphrase-MiniLM-L6-v2') 'Similarity score: 0.5182166695594788 using model: ./nlp_models/LaBSE' ( 'Similarity score: 0.7986025214195251 using model: './nlp_models/paraphrase-multilingual-MiniLM-L12-v2') ( 'Similarity score: 0.8292828798294067 using model: './nlp_models/bert-base-nli-mean-tokens') 'Similarity score: 0.861363410949707 using model: ./output_nlp/sbert_test_b' 'Similarity score: 0.9066307544708252 using model: ./output_nlp/sbert_test_a'
Final Output	===== FINAL RESULTS ===== OUTPUT: Image and Text pairing are congruous

---

## PairUp: Detecting Congruous Image and Text Pairings

---

	===== ===
Comments	The generated caption is really interesting in this case since it identifies that there is a baseball game going on. It says that there is a “pitcher” i.e. player who delivers the ball to the batter but this is not entirely true since the batter is trying to hit the ball in the game. When we input that there is a man playing baseball, the PairUp system says it's congruous which of course is true.

Serial No.	9
File:	./results/image_captioning_load_model_text_similarity_results9.ipynb
Input Image	
Input Text	I see lots of umbrellas
Predicted caption from Image captioning module	yellow umbrellas are shown on top of it.
Sentence similarity scores	('Similarity score: 0.7032688856124878 using model: ' ./nlp_models/all-MiniLM-L6-v2')

---

## PairUp: Detecting Congruous Image and Text Pairings

---


	<pre>('Similarity score: 0.6618703603744507 using model: '  './nlp_models/paraphrase-MiniLM-L6-v2') 'Similarity score: 0.32168811559677124 using model: ./nlp_models/LaBSE' ('Similarity score: 0.6921645998954773 using model: '  './nlp_models/paraphrase-multilingual-MiniLM- L12-v2') ('Similarity score: 0.6807048916816711 using model: '  './nlp_models/bert-base-nli-mean-tokens') 'Similarity score: 0.7595049142837524 using model: ./output_nlp/sbert_test_b' 'Similarity score: 0.8085371851921082 using model: ./output_nlp/sbert_test_a'</pre>
Final Output	<pre>===== FINAL RESULTS ===== OUTPUT: Image and Text pairing are congruous ===== ===</pre>
Comments	The PairUp system works fairly well and determines that the input text and input image are congruous.

Serial No.	10
File:	./results/image_captioning_load_model_text_similarity_results10.ipynb

---

## PairUp: Detecting Congruous Image and Text Pairings

---

Input Image	
Input Text	a full plate of food
Predicted caption from Image captioning module	a pile of different vegetables on top of a wooden table.
Sentence similarity scores	('Similarity score: 0.4230615496635437 using model: ' ./nlp_models/all-MiniLM-L6-v2') ( 'Similarity score: 0.3836595416069031 using model: ' ./nlp_models/paraphrase-MiniLM-L6-v2') 'Similarity score: 0.501850962638855 using model: ./nlp_models/LaBSE' ( 'Similarity score: 0.40308666229248047 using model: ' ./nlp_models/paraphrase-multilingual-MiniLM-L12-v2') ( 'Similarity score: 0.6528552174568176 using model: ' ./nlp_models/bert-base-nli-mean-tokens') 'Similarity score: 0.632498025894165 using model: ./output_nlp/sbert_test_b' 'Similarity score: 0.6960119009017944 using model: ./output_nlp/sbert_test_a'
Final Output	===== FINAL RESULTS =====



---

# PairUp: Detecting Congruous Image and Text Pairings

---

	OUTPUT: Image and Text pairing are congruous =====
Comments	The generated caption captures some level of detail with respect to identifying that there are vegetables and it's on a table but it does not identify them as carrots. But even if we provide the input text as a generic plate of food, which is semantically similar to vegetables, the PairUp system determines the pair as congruous.

Table 16: PairUp's evaluation of novel Image and text pairs

## 11. CONCLUSION AND FUTURE WORK

In this capstone, the design, implementation and testing of PairUp for determining congruity between Image and Text pairings was demonstrated. As discussed in section 10, with a small pairing set, PairUp achieved 90% accuracy. PairUp takes a **Textual Driver Based Semantic Similarity** approach whereby the input image in the image-text pair is converted to a textual representation by an image captioning technique. Next, a textual semantic similarity evaluation of this synthesized text and the original input text is performed.

One area of weakness is that sometimes the Image to Text Synthesizer sub-system generates less than ideal text even if overall the pairing congruency determination succeeded. If PairUp does not generate a good enough caption PairUp's congruence detection will fail and improvement of this module is left as future work. Some ideas to improve PairUp include:

- Combine multiple image captioning datasets (such as: MS COCO, Flickr30k, VizWiz [31], Hateful Memes [32] etc.) to create a huge dataset and then train the text-to-image synthesis/ image captioning model. Allowing the model to train on a bigger dataset would create a more robust image captioning model.
- Vision Transformer (ViT) [33] which are emerging as a competitive alternative to convolutional neural networks (CNNs) and they can be explored in the encoder architecture of the image captioning model. In many computer vision tasks, ViT models are seen to outperform the current state-of-the-art (CNN) by almost x4 in terms of computational efficiency and accuracy. ViTs can be explored to see if the image captioning tasks can be improved further.
- Create a combined end-to-end architecture by fusing together the functionalities of image to text generation and semantic similarity comparison. This approach would need additional work to modify the image captioning datasets such that each training example will have an image, a caption of the text, a similar/ dissimilar/ neutral sentence along with the similarity score.

It would also be interesting to compare the performance of the current PairUp system with a Visual Driver Based Semantic Similarity approach i.e., the input text is used to synthesize a relevant image and then a visual semantic similarity evaluation is carried out

---

# PairUp: Detecting Congruous Image and Text Pairings

---

between this newly generated image and the original image to determine if they are semantically similar or not.

## 12. DELIVERABLES

**GitHub:** <https://github.com/maitreeedu92/capstone-project>

**Demo of our Pair-up system:** <https://youtu.be/T8ERQSHr8LQ>

## 13. REFERENCES

1. Oriol Vinyals, Alexander Toshev, Samy Bengio, Dumitru Erhan, "Show and Tell: A Neural Image Caption Generator" on CVPR 2015.
2. Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate" on ICLR0215.
3. Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, Yoshua Bengio, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention" on CVPR2016.
4. Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang, Jiebo Luo, "Image Captioning with Semantic Attention" on CVPR2016.
5. Jiasen Lu, Caiming Xiong, Devi Parikh, Richard Socher, "Knowing When to Look: Adaptive Attention via A Visual Sentinel for Image Captioning" on ICCV2017.
6. K. Ganesan, "What is text similarity?" <https://kavita-ganesan.com/what-is-text-similarity/#.YmcvC8jMJPY>, 2022.
7. Pinky Sitikhu, Kritish Pahi, Pujan Thapa, Subarna Shakya; "A Comparison of Semantic Similarity Methods for Maximum Human Interpretability" <https://arxiv.org/abs/1910.09129>
8. <https://ai.googleblog.com/2018/05/advances-in-semantic-textual-similarity.html>
9. Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, "Efficient Estimation of Word Representations in Vector Space" on cs.CL2013. <https://arxiv.org/pdf/1301.3781.pdf>
10. Jeffrey Pennington, Richard Socher, and Christopher D. Manning, "GloVe: Global Vectors for Word Representation", 2014.
11. Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, Ray Kurzweil, "Universal Sentence Encoder" on cs.CL2018. <https://arxiv.org/abs/1803.11175>
12. <https://medium.com/analytics-vidhya/semantic-similarity-in-sentences-and-bert-e8d34f5a4677>
13. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, "Attention Is All You Need" on cs.CL2017.
14. <https://jalammar.github.io/illustrated-transformer/>



---

## PairUp: Detecting Congruous Image and Text Pairings

---

15. Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" on cs.CL2019. <https://arxiv.org/abs/1810.04805>
16. <https://jalammar.github.io/illustrated-bert/>
17. <https://www.pinecone.io/learn/sentence-embeddings/>
18. Nils Reimers, Iryna Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks" at EMNLP 2019.
19. <https://medium.com/swlh/image-captioning-using-attention-mechanism-f3d7fc96eb0e>
20. Minh-Thang Luong, Hieu Pham, Christopher D. Manning, "Effective Approaches to Attention-based Neural Machine Translation" on cs.CL2015.
21. Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate" on ICLR2015.
22. Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollár, "Microsoft COCO: Common Objects in Context" on CVPR.
23. Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna, "Rethinking the Inception Architecture for Computer Vision" on CVPR2015.
24. Papineni, Kishore, et al. "BLEU: a method for automatic evaluation of machine translation." Proceedings of the 40th annual meeting on association for computational linguistics. Association for Computational Linguistics, 2002.
25. Lin, Chin-Yew. "Rouge: A package for automatic evaluation of summaries." Text Summarization Branches Out (2004).
26. Vedantam, Ramakrishna, C. Lawrence Zitnick, and Devi Parikh. "Cider: Consensus-based image description evaluation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.
27. Banerjee, Satanjeev, and Alon Lavie. "METEOR: An automatic metric for MT evaluation with improved correlation with human judgments." Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization. 2005.
28. Peter Anderson, Basura Fernando, Mark Johnson, Stephen Gould, SPICE: Semantic Propositional Image Caption Evaluation.
29. Samuel R. Bowman, Gabor Angeli, Christopher Potts, Christopher D. Manning; "A large annotated corpus for learning natural language inference".
30. <https://huggingface.co/models>
31. <https://vizwiz.org/tasks-and-datasets/image-captioning/>
32. <https://ai.facebook.com/blog/hateful-memes-challenge-and-data-set/>
33. Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale"