Image-Sentence Pair Matching

-Project Report-

Ionescu Ariana-Georgiana
Group 511

1. Introduction

This project is designed to determine whether a given text description aligns with an associated image. By combining advanced techniques in natural language processing (NLP) and computer vision, the system evaluates the semantic similarity between the textual content and visual data. The integration of textual and visual data analysis is a growing need in modern AI applications. Whether for improving user experiences, automating content validation, or building smarter AI systems, bridging the gap between text and images is critical. This project serves as a foundation for exploring and implementing cutting-edge techniques in multimodal machine learning.

The structure of the project is:

- Data analysis, preprocessing the inputs, using tokenization for the text input
- Feature extraction using neural network models for each input type
- Cross-modal comparison using cosine similarity
- Training the models on the training data and using the validation data to check the model's ability to generalize
- Predicting whether or not the image and the text match

2. Data preprocessing

1. Text input:

+picture with the csv first 5 - 10 lines from train

Using a function called *text_preprocessing* which takes the text from the .csv files from the column 'caption', removes all characters that are not letters, removes the stopwords, and applies lemmatization. Using a new column 'clean_text' the output of the function is saved in each csv file, and the 'caption' column is removed since it is no longer needed.

2. Image input:

+pictures with the first 5 - 10 pictures like the csv (names)

Using a function called *preprocess_image* which takes the image paths, reads the image using the cv2 library, and transforms it from BGR input to RGB input, the image size is 100x100 with 3 color channels and to use them in the models later we divide each value with 255.0 to map the values from [0; 255] to [0; 1].

Using a function called *get_image_paths_clean_captions_id* going through the csv files row by row it extracts the clean text, image path, ID, and label to make sure that the data is correctly matched. The *preprocess_image* function is applied to the image path

output to get the numeric data, into numpy arrays called *x_train_image, x_test_image,* and *x_val_image* for each dataset. To finish the text preprocessing, on the clean text, we apply tokenization. The tokenizer object is fitted on the train text, we then apply *text_to_sequence* to transform to numerical data and apply *sequence.pad_sequence* so each text input is of the same size. The final numpy arrays are called *x_train_text, x_test_text,* and *x_val_text*.

3. Models

1. Convolutional Neural Network (CNN) for Image inputs and Gated Recurrent Unit Network (GRU) for Text inputs

Convolutional Neural Networks are a type of deep learning architecture that use data to learn patterns and classify objects. The layers used in a CNN architecture are: **the convolutional layer** (extracts features from the image inputs), **the pooling layer** (reduces dimensionality of the input image, which reduces the computational load), **the fully connected layer** (classifies the image based on the extracted features).

GRU networks are a type of RNN that can process sequential data. It uses a gating mechanism to selectively update the hidden state of the model at each step. The mechanism is used to control the flow of information at each step. GRU uses two mechanisms called **reset gate** and **update gate**.

2. Vision Transformer (ViT) for Image inputs and Long Short-Term Memory Network (LSTM) for Text inputs

The Vision Transformer is a type of transformer that was designed for computer vision. The ViT decomposes the image input into patches, flattens them, applies linear projection, and applies position embedding to each patch(the order is important). The embedded patches need to go through a transformer encoder a set amount of times (the L in the picture below). The architecture for the transformer encoder is: a normalization layer, a multi-head attention layer, a sum layer (between the embedded patches and the output of the multi-head attention layer), a normalization layer, a fully connected layer, a sum layer (between the output of the previous sum and the output of the fully connected layer). The last part of the architecture is the MLP head which is a fully connected layer.

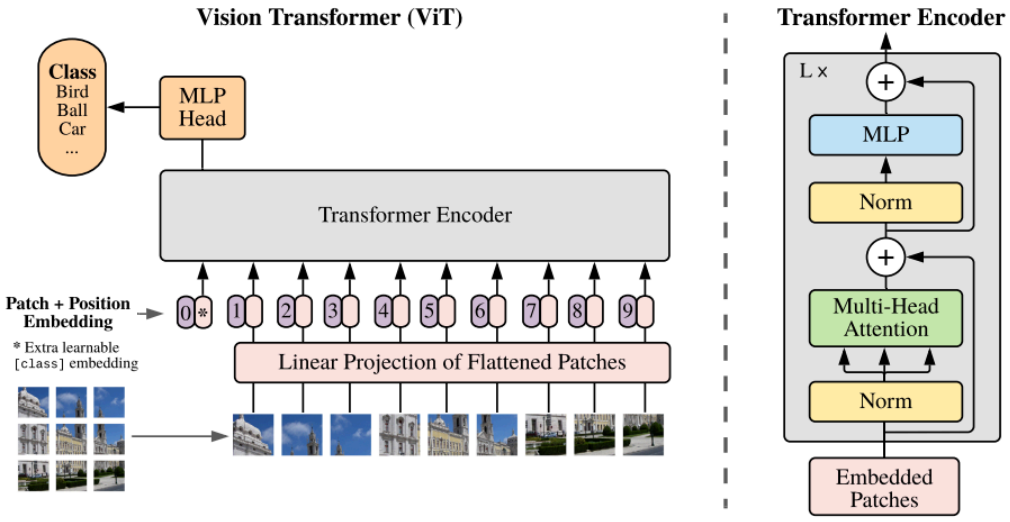


Image from An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale by A. Dosovitskiy, L. Beyer, etc.

For the implementation, using TensorFlow, I defined a custom layer ImagePatches using a class, that takes the size of one embedding *patch_size,* and a method *call* that takes the image*,* extracts the patches using *extract_patches* from *tensorflow.image,* and returns the flattened patches. I defined the ViT using a function *get_vision_transformer* that follows the architecture described earlier.

- Input layer - using Input with the shape (100,100,3) and dtype float32
- Patch Extraction layer - using ImagePatches
- Position Embedding - using Embedding
- Transformer Encoder - using a for to go through it as many times as needed
    - Normalization layer - using LayerNormalization
    - Multi-Head Attention Layer - using MultiHeadAttention
    - Normalization layer - using LayerNormalization
    - Fully connected layer - using Dense
- Normalization layer - using LayerNormalization
- Flatten layer - using Flatten
- Output layer/MLP Head - using Dense, with the same units as the last dense layer of the text encoder

LSTM networks are a type of RNN that can process sequential data. In addition to the RNN, the LSTM has a memory cell, a container that holds information for a longer period. The memory cell is controlled by three gates: **an input gate** (adding information), **a forget gate** (removing information), and **an output gate,** which allow the model to retain and remove information as it flows through it, learning dependencies.

3. Match Model

To determine if the caption and the image match, using the model *get_match_model* we calculate the cosine similarity using the formula *Dot(axes= -1, normalize=True),* and then map the output from [-1; 1] to [0; 1] using a dense layer and the activation function *sigmoid*. It is important that the output of the image and text models have the same shape.

4. Results

1. CNN + GRU

From CNN+GRU_sub6.py file
Using the following structure and parameters for CNN:

- Sequential layer to instantiate the model - using Sequential()
- 2D Convolutional layer - using Conv2D(32, 3, activation='relu', input_shape=(100, 100, 3))
- Apply batch normalization
- Dropout layer - using Dropout(0.25)
- 2D Max Pooling layer - using MaxPooling2D(2)
- 2D Convolutional layer - using Conv2D(32, 3, activation='relu')
- Apply batch normalization
- Dropout layer - using Dropout(0.25)
- 2D Max Pooling layer - using MaxPooling2D(2)
- 2D Convolutional layer - using Conv2D(64, 3, activation='relu')
- Dropout layer - using Dropout(0.5)
- Flatten layer
- Fully connected layer - using Dense(100) for the output (the same number of units as in the text model)

Using the following structure and parameters for GRU:

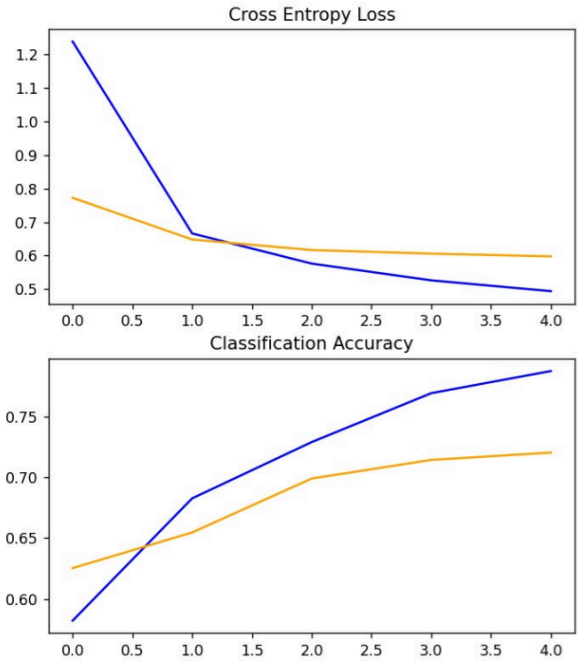
- Sequential layer to instantiate the model - using Sequential()
- Embedding layer with input dimension the same size as the dictionary size and the output dimension of 128
- GRU layer with 64 units
- Dropout layer - using Dropout(0.5)
- Fully connected layer - using Dense(100) for the output (the same number of units as in the text model)

The match model is the same as the one described previously.

The results for the validation data are:
Accuracy: 0.69
Loss: 0.6359
The confusion matrix is:

	0	1
0	0.72	0.32
1	0.28	0.68

The loss and the accuracy for the train and validation data (for batch size 32, 4 epochs and learning rate of 0.0001):



From CNN+GRU_sub3.py file.

Using the following structure and parameters for CNN:

- Sequential layer to instantiate the model - using Sequential()
- 2D Convolutional layer - using Conv2D(32, 3, activation='relu', input_shape=(100, 100, 3))
- 2D Max Pooling layer - using MaxPooling2D(2)
- 2D Convolutional layer - using Conv2D(64, 3, activation='relu')
- 2D Max Pooling layer - using MaxPooling2D(2)
- 2D Convolutional layer - using Conv2D(64, 3, activation='relu')
- Flatten layer
- Dropout layer - using Dropout(0.4)
- Fully connected layer - using Dense(64) for the output (the same number of units as in the text model)

Using the following structure and parameters for GRU:

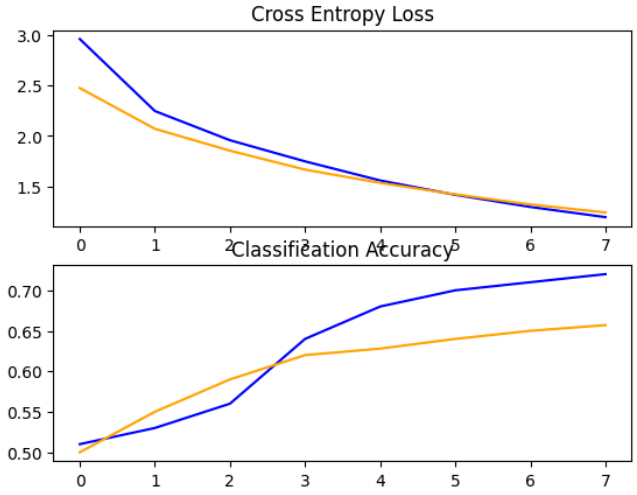
- Sequential layer to instantiate the model - using Sequential()
- Embedding layer with input dimension the same size as the dictionary size and the output dimension of 128
- GRU layer with 64 units
- GRU layer with 64 units
- Dropout layer - using Dropout(0.4)
- Fully connected layer - using Dense(64) for the output (the same number of units as in the text model)

The match model does not use cosine similarity in this case. It takes the outputs of the two models, applies concatenation, applies Dense(64, activation='relu'), and the final output, using Dense(1, activation='sigmoid').

The results for the validation data are:
Accuracy: 0.657
Loss: 1.24
The confusion matrix is:

	0	1
0	0.7	0.32
1	0.3	0.68

The loss and the accuracy for the train and validation data(using a batch size of 32, 8 epochs, and learning rate of 0.00005):



2. ViT + LSTM

From ViT+LSTM_sub10.py file
Using a ViT that extracts 16 patches (of size 25x25), 64 units for the linear projection layer (Dense(64)), going through the Transformer Encoder component 5 times, with 3 attention heads, and the MLP component having two fully connected layers of 256 as 64 units, the MLP Head layer of 512 units with the final output layer of 128 units.

Using the following structure and parameters for the LSTM:

- Embedding layer with input dimension the same size as the dictionary size and the output dimension of 128
- LSTM layer with 64 units
- Dropout layer of 0.2
- LSTM layer with 128 units
- Applying batch normalization to reduce overfitting
- Dropout layer of 0.3
- Dense layer of 128 units

The match model is the same as the one described previously.

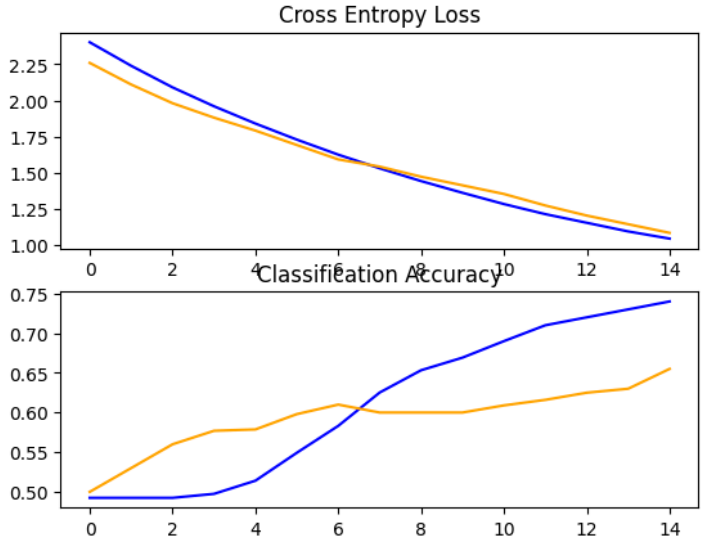
The results for the validation data are:
Accuracy: 0.652

Loss: 1.08

The confusion matrix is:

	0	1
0	0.67	0.36
1	0.33	0.64

The loss and the accuracy for the train and validation data: (for 15 epochs and 128 batch size, and learning rate 0.00005)



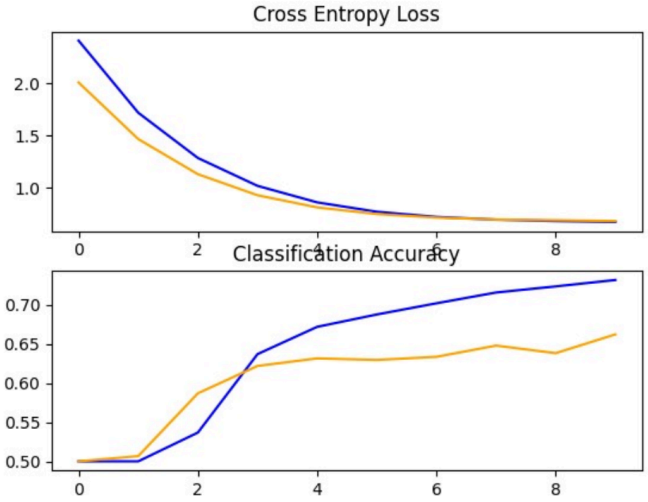
From ViT+LSTM_sub7.py file
Using different parameters for ViT: extracting patches of size 33x33, going through the Transformer Encoder 3 times, having 3 heads for the multi-head attention layer, and the MLP component having one fully connected layer 64 units, the MLP Head layer with the final output layer of 128 units.

The LSTM model is the same as the one described previously.
The match model is the same as the one described previously.

The results for the validation data are:
Accuracy: 0.65
Loss: 0.7138
The confusion matrix is:

	0	1
0	0.64	0.35
1	0.36	0.65

The loss and the accuracy for the train and validation data(during 10 epochs with a batch size of 32 and learning rate of 0.00005):



5. Conclusion:

In conclusion, this report explores two multimodal TensorFlow models to determine whether a given text description matches an associated image. The first version uses a CNN and GRU architecture, where the CNN extracts image features, and the text input is then processed sequentially by a GRU network. The second version adopts a ViT and LSTM approach, where a Vision Transformer extracts image patches, and an LSTM processes the textual information. Both models aim to capture the similarities between text and image but use different strategies for feature extraction and sequence modeling.