In the Name of God

University of Tehran
School of Electrical and Computer Engineering

Neural Networks and Deep Learning
Fourth Assignment

Question 1	
Name:	Mohammad Taha Majlesi
Student ID:	810101504
Submission Deadline:	2025-05-25

Contents

List of Figures

List of Tables

1 Question 1: Image Captioning with a Hybrid ResNet50 + LSTM-GRU Network

1.1 Dataset Preparation

1.1.1 Dataset Analysis

In this assignment, we aim to build a model for image captioning, which involves generating a brief description of an image in one or more sentences. This is achieved by combining Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). This technology can enhance accessibility for visually impaired individuals, enable intelligent image search, and facilitate content analysis on social media platforms.

The dataset used is **Flickr8k**, which consists of 8,000 images, each accompanied by five captions that clearly describe the prominent events in the image. The images were selected from six different groups on the Flickr website and do not include famous people or places. However, they were manually chosen to ensure a diversity of locations and situations.

Figure 2: Sample images from the dataset with one of their corresponding captions.

Upon downloading the images, we find a total of 8,091 images in JPG format, in color, and with varying dimensions. There is also a text file where each line contains a description for one of the images.

First, we preprocess the caption file. For each caption, we convert all letters to lowercase, then remove punctuation and numbers. We store all five captions for each image. Next, we split the captions into an 80% training set, a 10% validation set, and a 10% test set. Consequently, we can read the images corresponding to these caption splits. This results in 6,473 training samples, 809 validation samples, and 809 test samples. The preprocessing function for the captions is implemented as follows.

1.2 Tokenizer Creation

1.2.1 Tokenizer Analysis

In natural language processing, a **tokenizer** is a module that converts a given text into a list of numbers. These numbers represent the indices of embeddings that the model can understand. To do this, the text must be broken down into tokens. There are several methods to accomplish this.

One approach is to consider individual characters or sets of bytes that form characters as tokens. This results in a small dictionary, but the semantic meaning of individual characters is not very distinct, making it difficult for the model to understand the text. Additionally, for each input, the number of tokens is very large, causing the context window to fill up quickly, and thus the model cannot process long inputs.

Another approach is to consider a few words or an entire sentence as a single token. Due to the variety and numerous forms of phrases, the number of dictionary elements becomes very large. Since each token encompasses a larger portion of the input, the context window is consumed more slowly, allowing for longer inputs. However, the model needs to see more data to learn all these different tokens.

Today, large language models use a combination of the above methods. For example, a tokenizer might break words into prefixes and suffixes. If it encounters a new word, it breaks it down into smaller components like characters or bytes, enabling it to correctly tokenize most inputs.

1.2.2 Special Tokens

In tokenizers, there are usually some special tokens that are inserted into the input text for specific purposes by the tokenizer. The model learns these tokens just like any other token, and because they appear in various contexts, they can store meaningful information from all texts.

In this project, we have four special tokens. The two tokens, ‘sos‘ (start of sentence) and ‘eos‘ (end of sentence), mark the beginning and end of each input sentence, respectively. This way, the model can learn that texts have a start and an end, and the sentences it generates will also have a beginning and an end.

As mentioned, we set the number of tokens for all model inputs to 37. If a text has more tokens, we only consider the first 37 tokens. If it has fewer than 37 tokens, we need to bring the token count up to 37. For this, we use the ‘pad‘ token. After the sentence ends and the ‘eos‘ token is placed, we add as many ‘pad‘ tokens as needed to reach the maximum possible number of tokens, which is 37.

The use of ‘padding‘ is essential in natural language processing because most modern text processing models, from the transformer family, can only accept a fixed number of tokens as input. Although models from the RNN family can accept variable-length inputs, a fixed token count reduces computational complexity and simplifies the model, making it easier to train.

The last token is ‘unk‘, which stands for unknown tokens. It is possible that during evaluation, a token appears in the input that was not present in the vocabulary created from the training data. For the model to understand the meaning of these tokens to some extent, we use the ‘unk‘ token. We replace tokens that are not in the vocabulary with ‘unk‘ during inference. For this token to have a meaning similar to undefined tokens, we must use it in the training data so the model can learn it. To do this, we can consider a certain number or percentage of the least frequent tokens in the training data as ‘unk‘. In our case, we take the captions, count the frequency of each token, and then consider the 10% least frequent tokens and all tokens that appeared with the same or fewer frequency as ‘unk‘. This method works well when new tokens in the evaluation data are part of the low-frequency tokens in the general population, which is usually the case.

1.2.3 Creating the Tokenizer Class

The ‘Tokenizer‘ class receives the training captions and, by analyzing the frequency of tokens in the captions as explained above, creates a vocabulary for tokens with a certain minimum frequency. For each token, it assigns a unique number from 0 to the total number of vocabulary tokens.

This class, upon receiving text, tokenizes it based on spaces, as we discussed in section 1-2-1, and assigns the corresponding number from the dictionary to each token. It can also receive the model's output, which is in the form of indices for the vocabulary tokens, and return the corresponding text output.

Thus, using the training data, we create a vocabulary and save it in JSON format to have it available for subsequent runs. The vocabulary created from the five captions of the training images contains a total of 8,536 unique words. 41% of them, or 3,519, appeared only once and were not stored in the vocabulary. The other 5,017 words, which appeared more than once in the training captions, were stored. In total, among all training captions, there were 349,331 tokens, of which only one percent were replaced by the unknown token. By examining the low-frequency words that we did not store in the vocabulary and considered as 'unk', some are genuinely rare words like "seahorse," while many others seem to be due to spelling mistakes, such as "playgrou" which is incompletely written. Others are specific forms of high-frequency words, like "visits." The last two categories are mistakenly considered low-frequency. With sub-word tokenization, these errors could be covered, although it would increase the training complexity, which is not necessary for this assignment's objective and image captioning. Here is an example of the input and output text of the model with special tokens:

```
Text:
a blackandwhite dog bounds off the ground all feet in the air of a yellow
    field

Output text with special tokens:
<sos> a blackandwhite dog bounds off the ground all feet in the air of a
    yellow field <eos> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
```

Listing 1: Example of tokenized text

We then create a dataset class that, given an index, returns the corresponding image and tokenized captions. We also use the transformations used in ResNet for the images. These transformations first reduce the image dimensions to 232, then crop it to a size of 224 pixels. After that, the pixel values are scaled to be between zero and one, and finally, the values are normalized using the mean and standard deviation of the ImageNet images. ImageNet contains millions of images with various classes, and normalizing with its mean and standard deviation is common in computer vision and has been shown to help with model stability and convergence. This way, the mean of the images approaches zero, and their standard deviation approaches one.

```
class Flickr8kDataset(Dataset):
    def __init__(self, image_dir, captions, tokenizer, transform=None):
        self.image_dir = image_dir
        self.transform = transform
        self.captions = captions
        self.ids = defaultdict(list)
        for img in captions.keys():
            self.ids[img] = np.array([tokenizer.text_to_ids(caption) for
    caption in captions[img]])
        self.tokenizer = tokenizer
        self.image_files = list(captions.keys())

    def __getitem__(self, index):
        image_file = self.image_files[index]
        image_path = os.path.join(self.image_dir, image_file)
        image = Image.open(image_path).convert('RGB')
        if self.transform:
            image = self.transform(image)
        ids = self.ids[image_file]
        return image, ids
```

Listing 2: Dataset Class

1.3 Model Implementation

1.3.1 Encoder Implementation

For the encoder part, we use a pre-trained CNN model, **ResNet50**, and the transfer learning method. Let's first examine this model.

ResNet50 is a 50-layer deep residual network that uses skip connections to solve the vanishing gradient problem. This problem occurs in deep networks where, due to the depth of the network, the gradient values become very small in the initial layers, and the impact of these layers on the model's performance diminishes, even though they can have a very significant effect. Adding the input of a convolutional layer to its output is called a skip connection.

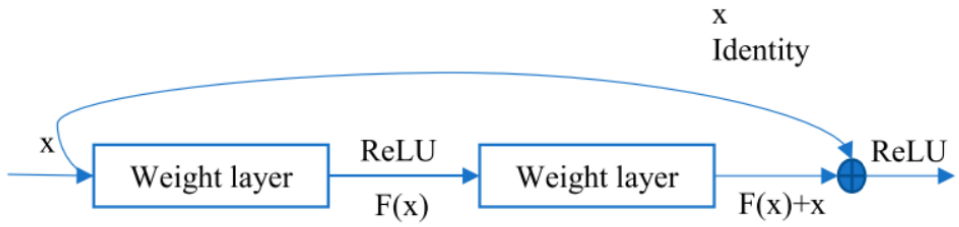


Figure 3: ResNet50 skip connection to solve the vanishing gradient problem.

A downsampling connection is one that bypasses some layers of the model, and with it, the output is as follows: $Y = F(X) + X$

The ResNet50 model is composed of two types of blocks: the identity block and the convolutional block. The identity block is used when the input and output dimensions are the same, and the input is added to the output.

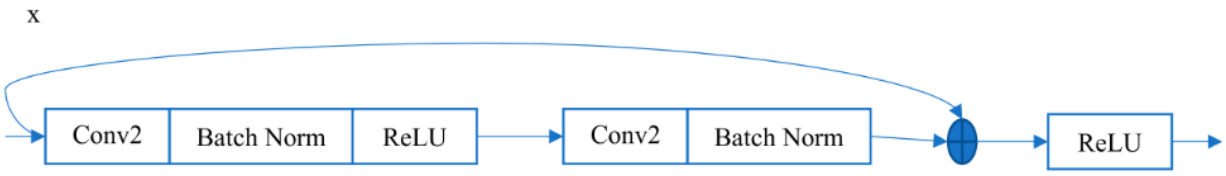


Figure 4: ResNet50 identity block.

If the input and output dimensions are not equal, another block, the convolutional block, is used, where a convolutional layer is added to the output layer.

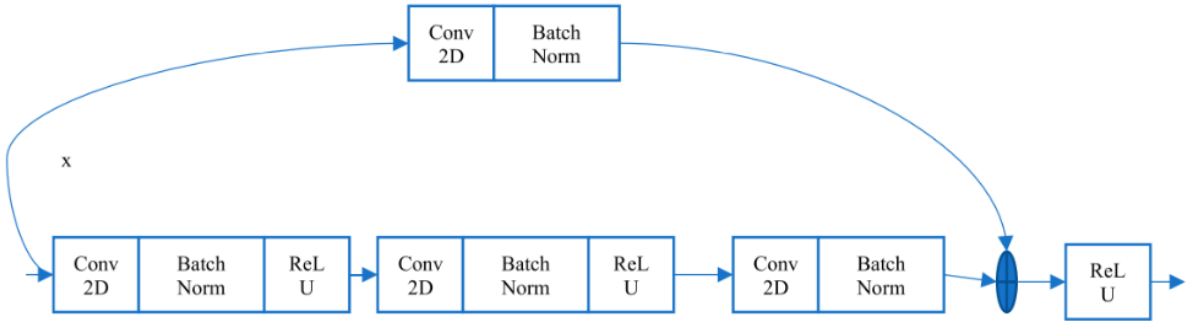


Figure 5: ResNet50 convolutional block.

The ResNet50 model takes color images with dimensions (3, 224, 224) as input, and the fully connected layer of the model is removed to use the pre-trained model for feature extraction.

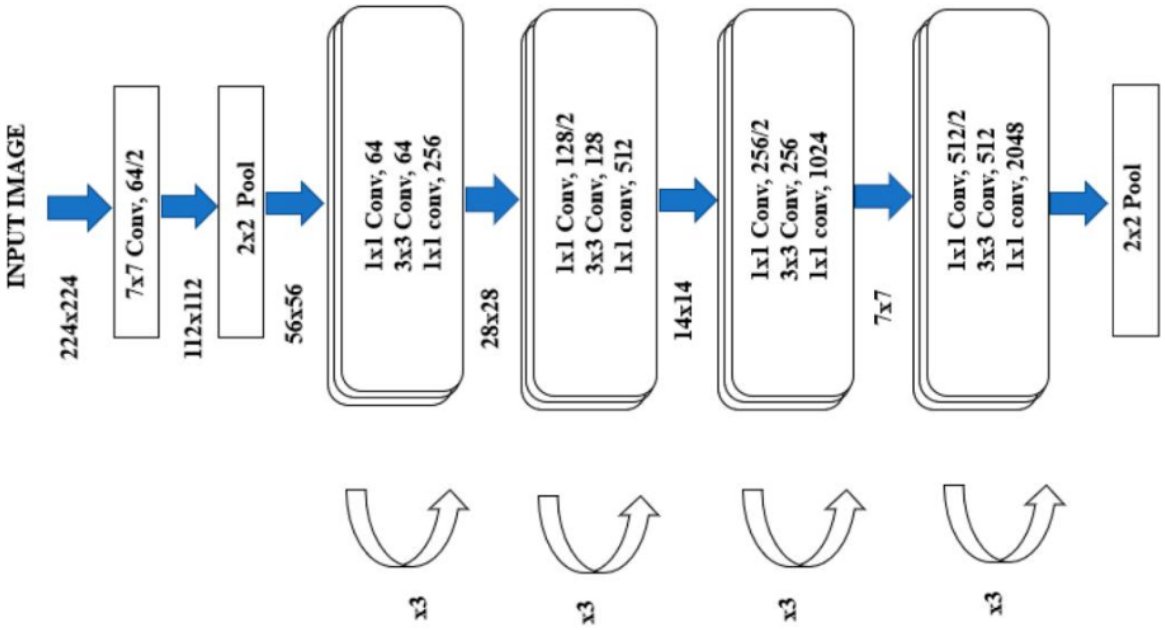


Figure 6: ResNet50 architecture used as an encoder.

The PyTorch framework has the ResNet50 model readily available. After removing its final layers, we freeze all the layers. Then, by passing a sample input of size 224x224, we observe that the output vector has a size of 2048.

```
class Encoder(nn.Module):
    def __init__(self):
        super(Encoder, self).__init__()
        resnet = models.resnet50(weights='IMAGENET1K_V1')
        modules = list(resnet.children())[:-1]
        self.encoder = nn.Sequential(*modules)
        for param in self.encoder.parameters():
            param.requires_grad = False

    def forward(self, x):
        x = self.encoder(x)
        x = x.view(x.size(0), -1)
        return x
```

Listing 3: Encoder Implementation

1.3.2 Decoder Implementation

After creating the token vector, we can represent their indices in one-hot format and feed them to the model for learning. This method has several drawbacks. First, due to the use of one-hot representation, a vector with all zero elements is created, with only the house corresponding to the token being one. This results in a vector with a length equal to the vocabulary size, which is 5017, while the only information stored in it is the token's index. This also means that the entire burden of learning the meaning of these words is placed on the LSTM model, making the learning process almost impossible.

For this reason, another method is used in language models to convert the input into numbers, which is called **embedding**. In this method, a vector is assigned to each word, and the vectors corresponding to all words together form the embedding layer. Then, the model, using the backpropagation algorithm, adjusts the weights of the embedding vector for each word to capture the different meanings of each word within its vector. Some of these meanings are understandable to us, but some are only for improving performance and we cannot provide a semantic justification for them. However, it has been shown that the difference between the embedding vectors of words with similar relationships has a constant value. For example, the difference between the vectors for "father" and "mother" is very close to the difference between the vectors for "brother" and "sister." One of the problems with this method is that it tries to fit all the meanings of a word into its corresponding vector, whereas the main meaning of a word is understood based on the other words in the sentence. The attention mechanism was created to solve this problem, but we do not use it in this project. Thus, by using word embedding, we convert words into vectors and give them as the initial state to the LSTM. A vector length of 300 is a common value, and we will investigate the effect of different embedding dimensions on the model's performance with the help of the validation set.

The **LSTM** model is from the RNN family of models, which was introduced to address the vanishing gradient problem of RNNs. It consists of three gates: the forget gate, the input gate, and the output gate. The cell state refers to the transient information from the model. The gates change the information present in the cell state. Various types of LSTMs are created by stacking several LSTMs in series or making them bidirectional, but in the paper, a simple model is used.

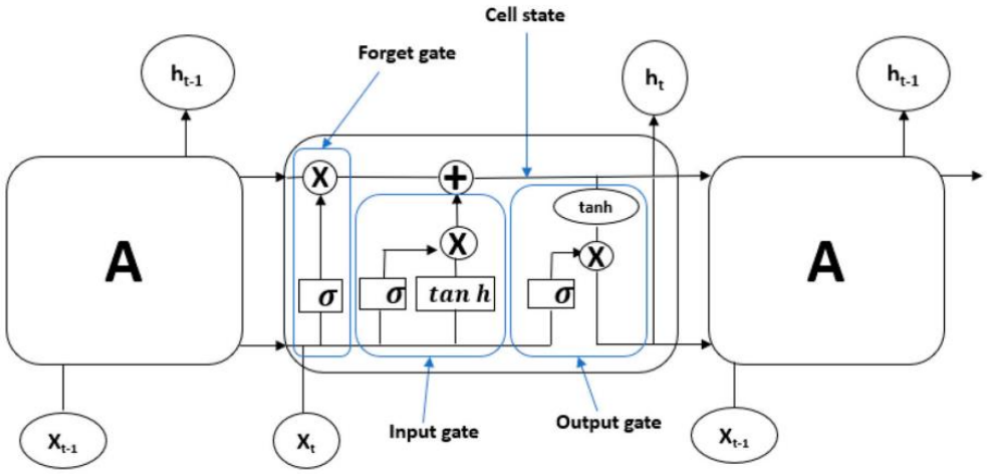


Figure 7: LSTM model structure.

The **GRU** model consists of two gates: the update gate and the reset gate. This model is also similar to LSTM and was introduced to solve the vanishing gradient problem of RNNs. Its advantage over LSTM is that it has fewer parameters and is capable of understanding longer dependencies in the text.

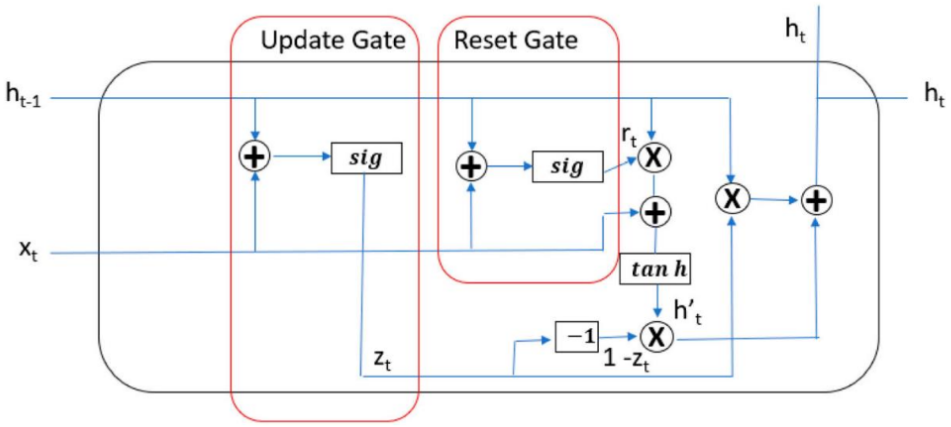


Figure 8: GRU model structure.

In this paper, a combination of LSTM and GRU models is presented and used for the decoder. The idea behind this combination is to leverage the strengths of both models and cover their weaknesses.

Both LSTM and GRU are types of RNNs that aim to solve the vanishing gradient problem and better understand long-term dependencies in text. The LSTM model has three gates: input (i_t), forget (f_t), and output (o_t), while the GRU has two gates: reset (r_t) and update (z_t), resulting in lower computational cost. The gates store information in memory.

The output of the LSTM is given as input to the GRU. x_t is the input, which includes the hidden state h_{t-1} of the LSTM. The gates are defined as follows:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$
$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

The LSTM values are obtained by multiplying x_t and h_{t-1} with W, adding the bias b_i, and passing through a sigmoid function. The cell input C_t is defined as follows: $C_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$ The output is also defined as follows: $C'_t = f_t \times C'_{t-1} + i_t \times C_t$ C'_t goes to the GRU layer, i.e., z_t. z_t and h_{t-1} are multiplied by weights and go to the reset gate r_t.

$$z_t = \sigma(W_z[C'_t] + W_z[h_{t-1}])$$
$$r_t = \sigma(W_r[C'_t] + W_r[h_{t-1}])$$

The output layer determines what information to keep.

$$h'_t = \tanh(W'_c + r_t \odot W_{ct}[h_{t-1}])$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t$$
$$h_t = o_t \times \tanh(h_t)$$

In the above relations, h_t and h'_t represent the hidden states, and W_o, W_f, W_i, W_c represent the weights, and b_i, b_f, b_o, b_c represent the biases.

In summary, the decoder receives the feature vector from the encoder. The features are given as the initial hidden state and cell state to the LSTM. The embedding of the previous token is also given as input to the LSTM. We always give the start-of-sentence token to the model initially, and the model predicts the next tokens. According to the relations written in the paper, we give the cell state at each time step from the LSTM as input to the GRU at the same time step. We also give the feature vector as the initial hidden state to the GRU model. Finally, we pass the output of the GRU through linear layers and, by increasing the dimension to the number of words in the vocabulary, we get the logits as output.

1.3.3 End-to-End Model Integration

We need to combine the encoder and decoder parts so that the images pass through the convolutional part of ResNet50, and the resulting features, after passing through the LSTM and then the GRU, produce the caption.

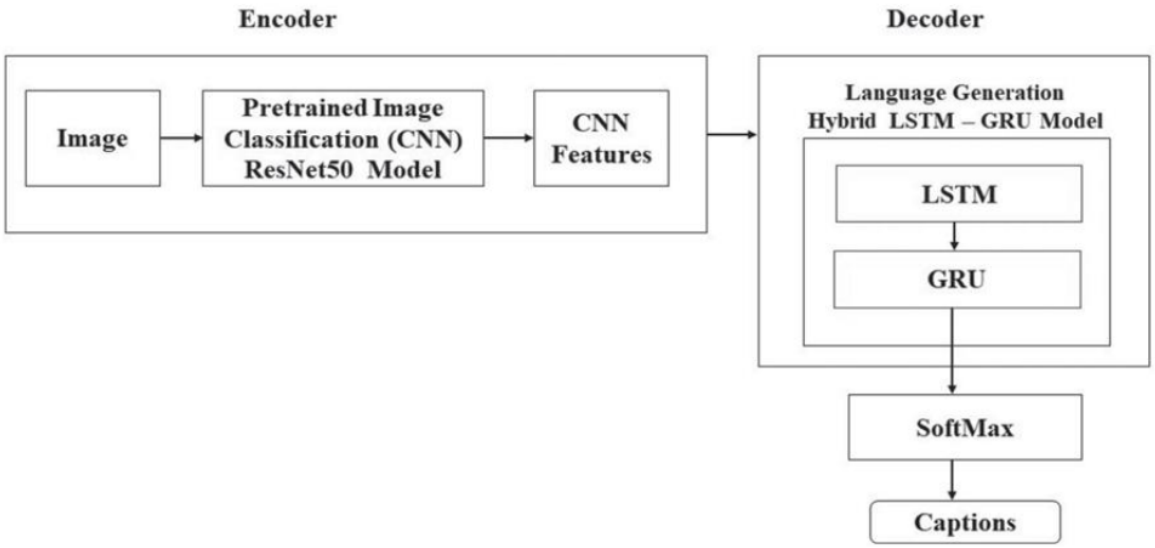


Figure 9: Architecture of the model used.

Now we form the 'HybridModel' class, which includes the encoder and decoder parts. First, we pass the images through the encoder to get the features. To give the images and captions to the decoder, we face a problem. For each image, there are five captions, while the decoder expects one caption per image. To solve this problem, we convert the three-dimensional caption matrix into a two-dimensional matrix, so that it has five times the batch size of captions. We also repeat the feature vector 5 times to match the number of images and captions. This is also efficient in terms of time because each image passes through the encoder only once, and the model works correctly because we provide one caption per image to the decoder.

```
class HybridModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_size, device):
        super(HybridModel, self).__init__()
        self.encoder = Encoder()
        self.decoder = Decoder(vocab_size, embedding_dim, hidden_size,
    device)

    def forward(self, images, captions):
        features = self.encoder(images)
        features = features.unsqueeze(1).repeat(1, captions.shape[1], 1)
        features = features.view(-1, features.shape[-1])
        captions = captions.view(-1, captions.shape[-1])
        return self.decoder(features, captions)
```

Listing 4: End-to-End Model

With an embedding size of 300, the model will have about 56 million parameters, and with an embedding size of 50, it will have about 53 million parameters.

1.4 Model Training and Evaluation

1.4.1 Text Generation Methods

To evaluate the models, besides checking the loss on the validation data during epochs, we also use other methods. But before examining the evaluation methods, we must determine the text generation method. We implement and examine two common methods.

The first algorithm for text generation is the **greedy** method. In this method, at each step, we select the most probable token. This method is very simple to implement and justify, but it may not produce the most probable overall text. By choosing a token with high probability, and then choosing subsequent tokens in the same way, the total probability of the generated text might be low. Thus, this method does not produce the most probable text but a suboptimal yet reasonably good text. The details of this algorithm are as follows:

1. Create an empty array to store the output.
2. Add the ‘¡sos¿‘ token to the array as the first token.
3. Generate tokens until either the ‘¡eos¿‘ token is produced or the maximum number of desired tokens is reached.
4. Use a decoder to get the probability of the next token based on the previous tokens.
5. Select the most probable token. At each step, this is the best local choice.
6. Consider the selected token as the current input to generate the next tokens.
7. Add the selected token to the list of generated tokens.
8. Stop the process after generating ‘¡eos¿‘ or enough tokens.

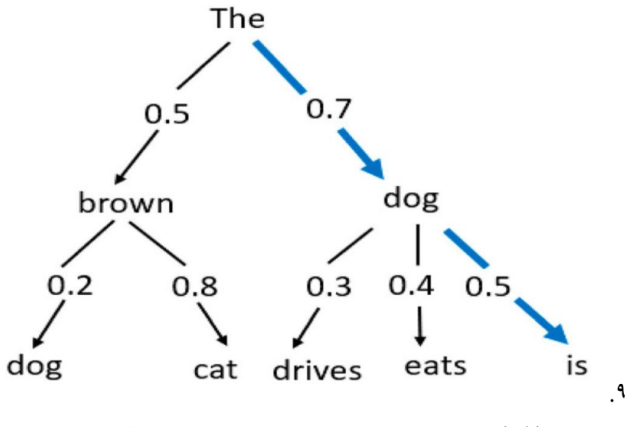


Figure 10: How the greedy method works for text generation.

The next common method is **beam search**, which usually produces more accurate and diverse captions. A beam is a sequence of n consecutive tokens. If the number of beams is considered to be one, we get the greedy algorithm. The larger the number of beams, the higher the probability of finding the most probable sequence. This method solves the problems of the greedy algorithm by selecting the most probable beams at each step, and finally, the most probable beam is considered as the output. The details of this algorithm are as follows:

1. Create an initial beam containing the start-of-sentence token.

2. For each beam, form a set of probabilities for the next token. This is done with the help of the model and by giving the existing tokens in each beam.

3. For each next token in each beam, we assign a score. This score is equal to the sum of the log probabilities of each token in that beam plus the log probability of the new token.

4. We consider K probable beams to use as the beam in the next step.

5. We repeat steps 2 to 4 until we reach a sufficient number or the ‘¡eos¿‘ token is produced.

6. Finally, we consider the most probable beam as the generated text.

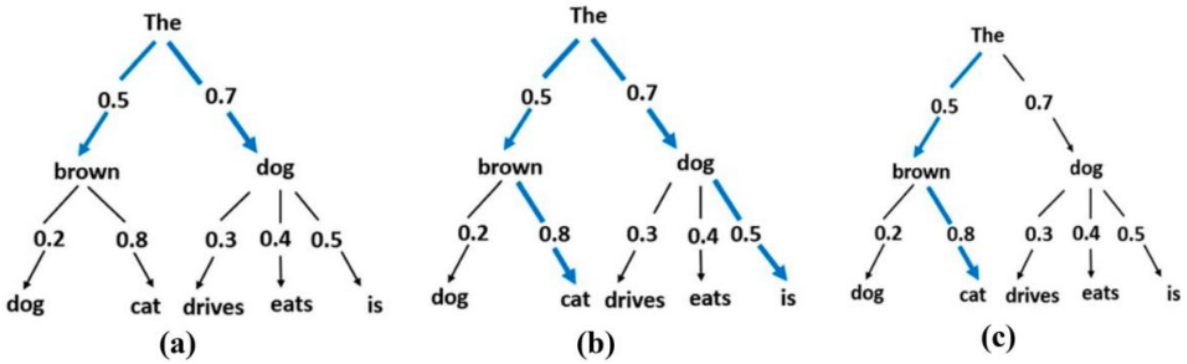


Figure 11: How beam search works for text generation.

1.4.2 Evaluation Metrics (Scoring)

Like other machine learning domains, using numerical metrics can help us compare models more accurately based on fixed mathematical criteria. In general, numerical evaluation methods have advantages over human evaluation. For instance, evaluating a large number of captions by a human is a very slow process, and usually, after evaluating a few captions, the quality of comparison decreases due to fatigue. Also, human comparisons can be subjective, and each person's opinion on comparing captions can differ, whereas numerical metrics have a fixed and specific value.

Of course, any numerical metric can only evaluate a caption from a specific aspect, and there is no metric that is flawless and correctly compares all aspects of the text. Therefore, usually for evaluating metrics, they are compared with human evaluations. Also, in generating captions for images, it is possible to generate captions that, although not present in the target captions, are logically related to the image. For example, the model might generate a caption about objects in the background, and we expect this caption to get a better score than a caption that is completely unrelated to the image, but due to not considering the images, both get low scores.

Usually, metrics used in natural language processing and machine translation are used for image captioning. The first metric, which we also use, is **BLEU** (Bilingual Evaluation Understudy). It measures the precision of n-gram models between the target text and the generated text. BLEU-1 only uses 1-grams, and 1-grams also show the number of common words between two captions without considering their position. Thus, BLEU-1 measures the number of words in the generated text that are also in the target text.

1.5 Evaluation of Different Models

1.5.1 Embedding Vector with Length 50

Initially, we use the described model with embedding vectors of length 50 and train a model for 10 epochs on the training data, then evaluate it on the validation data.

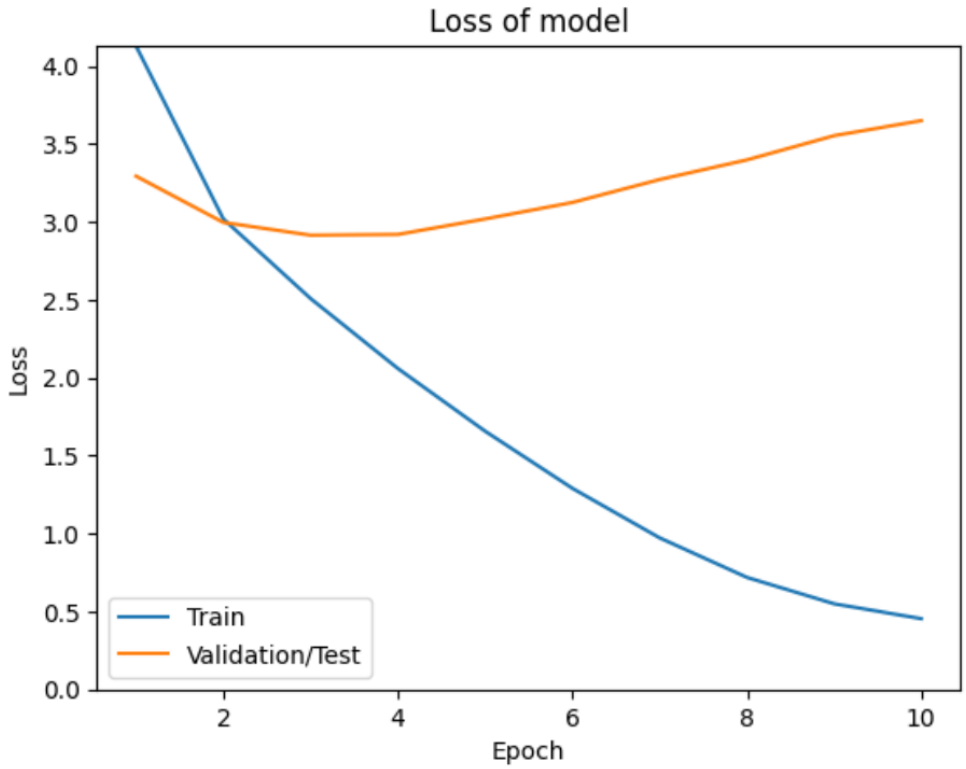


Figure 12: Loss changes during training by epoch.

As is clear from the graph above, the model is overfitting. Although the training data loss is decreasing rapidly, the validation data loss starts to increase after the fourth epoch. Thus, the model's loss on the validation data after 10 epochs reaches 3.6487, which is very high.

True: a person in the distance hikes among with stars visible in the sky

Greedy: a man stands on the sand with his arms in the air and looks down the landscape

Beam (K=3): a man feels on top of a mountain with other people in the background

Beam (K=5): a man feels on top of a mountain with arms in the background

Beam (K=10): a man feels on top of a mountain with arms in the background

True: a man holding the hand of another man who has a tattoo on his arm

Greedy: a man with a blue hat is sleeping against a column in a little girl in front of a pile of other children

Beam (K=3): a man in a blue hat is sleeping against a red building with people children children

Beam (K=5): a man with a blue hat sits on the ground against a building with like a

Beam (K=10): a man with a blue hat sits on the ground against a building with like a

True: one very fat sumo wrestler wrestles with a wrestler while an umpire dressed in green watches

Greedy: two men are wrestling in a competition metal a metal suit white and black colored

Beam (K=3): a man and a woman are dressed in black and white costumes in a parade while the black and white shirts

Beam (K=5): two men are wrestling in the sand metal are white white and one black

Beam (K=10): two men are wrestling in a competition room in a park with like children

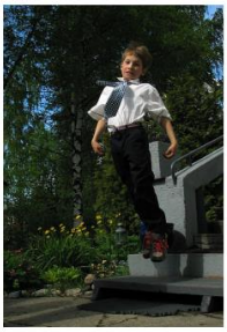
True: a black dog in the middle of running or leaping

Greedy: two dogs are running on the grass with two red dogs with one black one

Beam (K=3): two dogs are standing in the snow with two with

Beam (K=5): two dogs are running side by side in the snow

Beam (K=10): a black dog and a black dog are running in the snow



True: a boy in a shirt and tie jumps off a staircase

Greedy: a girl is jumping a rail with a white wall behind it in the background in a like children

Beam (K=3): a woman in a white shirt is standing outside of a plastic bag with a bicycle children children

Beam (K=5): the girl is wearing a white shirt and jeans and walking down the street white she can

Beam (K=10): the woman is performing a trick on a skateboard outside in a park white sign

Figure 13: Captions generated for a few sample images for the model without dropout.

Table 1: Evaluation metrics for the model without dropout.

	MicroAcc	MacroAcc	BLEU-1	BLEU-2	BLEU-3	BLEU-4
Greedy	0.208325	0.214487	0.434335	0.254904	0.146316	0.087936
Beam K=3	0.218159	0.218958	0.442262	0.266012	0.157038	0.096279
Beam K=5	0.210859	0.211694	0.450165	0.272289	0.161610	0.098869
Beam K=10	0.210317	0.211630	0.451148	0.275044	0.163723	0.100048

In the paper, it is stated that the model's loss decreases to 0.4013 and its accuracy reaches 0.8932. Also, in the paper, it is stated that the BLEU-1 score for the greedy algorithm is 0.5142 and for the Beam search algorithm is 0.6034.

1.5.2 Embedding Vector with Length 50 and Dropout

Observing the overfitting in the loss graph, to prevent it, we decided to add a dropout layer with a rate of 50% to the hidden state layers of the LSTM and GRU, the cell state layer of the LSTM, and the embedding layers, and train the model for 10 epochs.

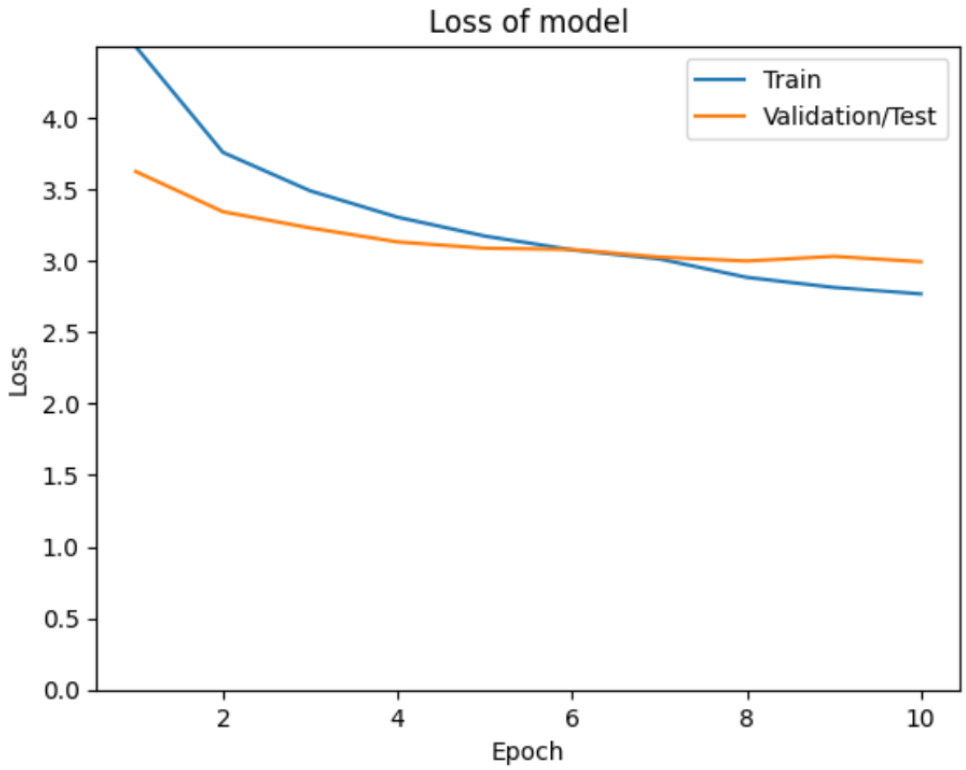


Figure 14: Loss changes of the model with dropout during training by epoch.

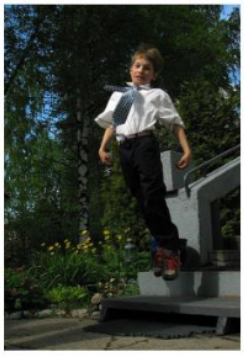
It can be seen that with this, overfitting does not occur. Of course, the training speed decreases, but it seems to be compensated for in more epochs. Also, finally, after 10 epochs, the model's loss reaches 2.9930, which is better than the previous model.

True: a person in the distance hikes among with stars visible in the sky
Greedy: a man is standing on a rock overlooking a valley
Beam (K=3): a man stands on a cliff overlooking the ocean
Beam (K=5): a man stands on a cliff overlooking the ocean mountains
Beam (K=10): a man is standing on top of a cliff overlooking the ocean

True: a man holding the hand of another man who has a tattoo on his arm
Greedy: a man in a blue shirt is standing in front of a large building
Beam (K=3): a man in a blue shirt is standing in front of a large building
Beam (K=5): a man in a blue shirt is standing in front of a wooden building
Beam (K=10): a man in a blue shirt is standing in front of a wooden building

True: one very fat sumo wrestler wrestles with a wrestler while an umpire dressed in green watches
Greedy: two men are wrestling on a trampoline
Beam (K=3): a man in a blue shirt is jumping into the air to catch a ball ball
Beam (K=5): a man in a blue shirt is jumping into the air to catch a ball ball
Beam (K=10): a man in a black shirt is jumping into the air to catch a ball ball

True: a black dog in the middle of running or leaping
Greedy: two dogs run through the grass
Beam (K=3): two dogs run through the grass
Beam (K=5): two dogs run through the grass
Beam (K=10): two dogs run through the grass



True: a boy in a shirt and tie jumps off a staircase
Greedy: a woman in a white dress is walking through a field of flowers
Beam (K=3): a woman in a white dress is walking down a sidewalk
Beam (K=5): a woman in a white dress is walking down a sidewalk
Beam (K=10): a woman in a white dress is walking down a sidewalk

Figure 15: Captions generated for a few sample images for the model with dropout.

Table 2: Evaluation metrics for the model with dropout.

	MicroAcc	MacroAcc	BLEU-1	BLEU-2	BLEU-3
Greedy	0.232908	0.243318	0.570553	0.381970	0.245657
Beam K=3	0.238769	0.247726	0.570572	0.382528	0.248707
Beam K=5	0.237850	0.247339	0.568306	0.380303	0.246676
Beam K=10	0.235818	0.244766	0.558266	0.369942	0.237431

From the table above, it is clear that the evaluation metrics have improved, and the BLEU-1 value has become closer to the paper's value, although the accuracy and loss are still far apart. Due to the increase in the metric values and preventing overfitting, we will use dropout in the rest of the models as well.

1.5.3 Not Using Teacher Forcing

Many text generation models use the teacher forcing method. In the paper, although not explicitly stated, it seems from the model's image that this method is used. But we train the model once without this method to examine its effect.

In the teacher forcing method, we give the model the true token at time t and, with the help of the model, generate the token for the next moment. But without using teacher forcing, at each moment, we give it the token predicted by the model at the previous moment.

Using teacher forcing speeds up convergence. Also, because in the initial moments of training, the model's predictions have a large error, the model's hidden state is updated by incorrect predictions, and by accumulating errors, training becomes difficult. Of course, at inference time, the model does not have the true tokens and does not use teacher forcing. Therefore, there is a difference between the training and evaluation time, and this may cause a decrease in quality and instability. For this reason, we examine both methods.

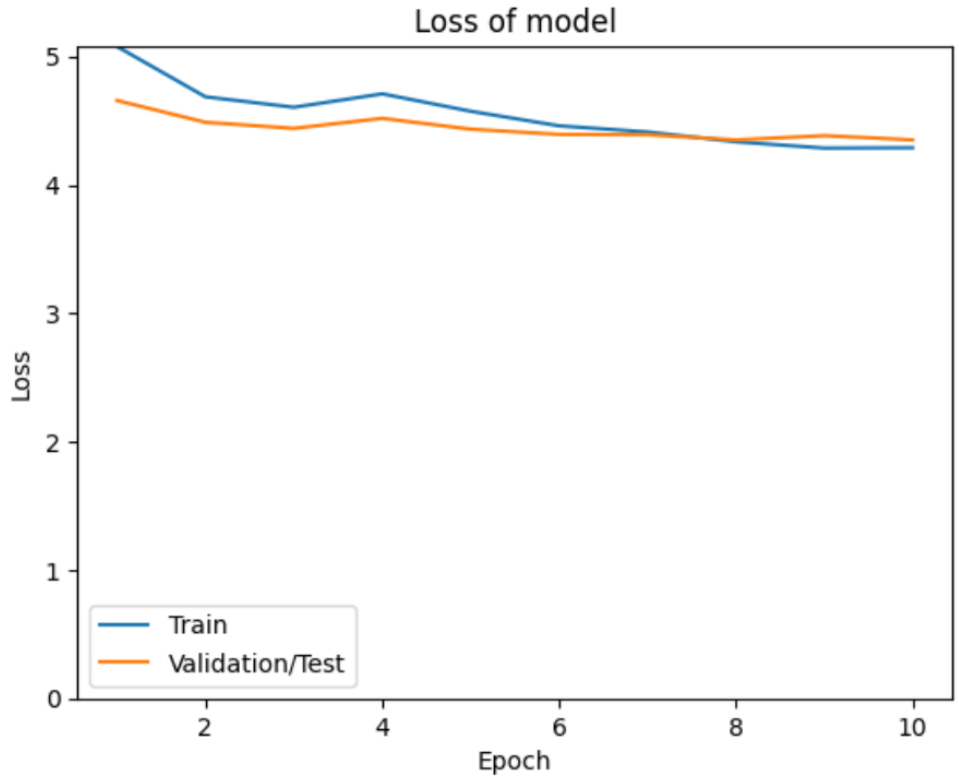


Figure 16: Loss changes of the model without teacher forcing during training by epoch.

You could see that without using teacher forcing, the training speed decreases, and the model starts training with a higher loss of 5. In the end, the amount of loss is also higher than the previous methods and reaches 4.35. Also, the generated captions are of very poor quality, and many repeated tokens are seen in them. For this reason, we will use teacher forcing in the continuation.

1.5.4 Reducing Dictionary Size

During some runs, the model sometimes generated words with spelling errors. We think that if we reduce the dictionary size, that problem might be solved, and due to fewer parameters, the model's performance might improve. Thus, we create a tokenizer that also includes words with two repetitions in the captions. This means that a total of 3847 words are in the vocabulary, and thus 55 percent of the dictionary words and 1.68 percent of the caption tokens are considered as unknown words. Now we train a new model with the new tokenizer for 10 epochs.

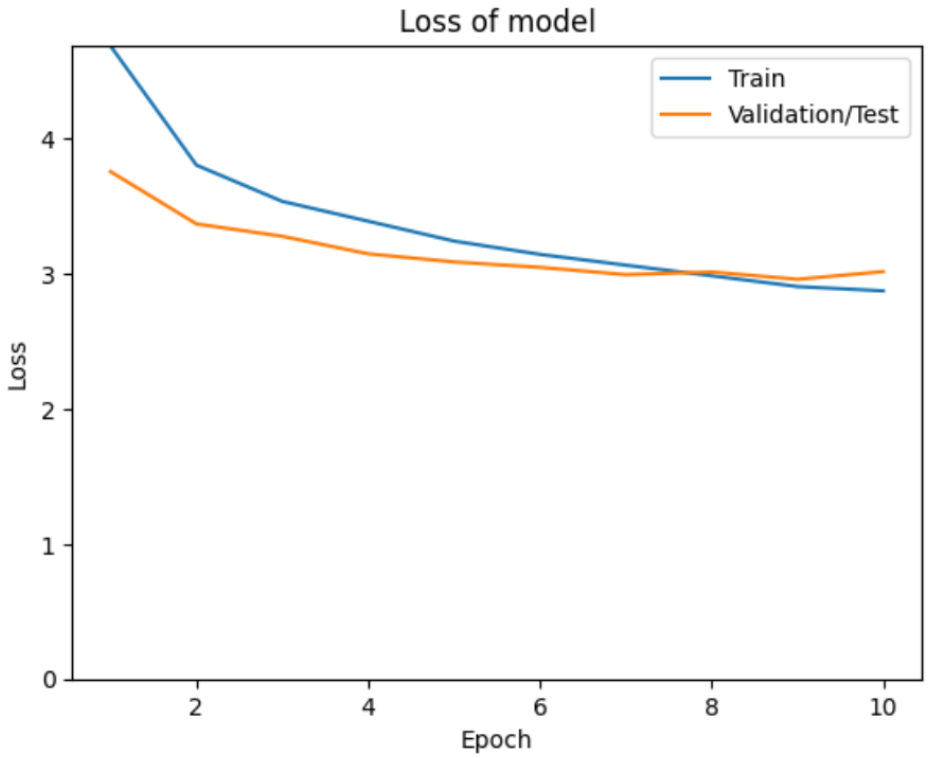


Figure 17: Loss changes of the model with a small dictionary during training by epoch.

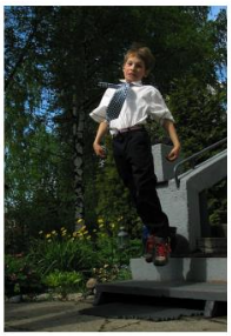
It can be seen that, similar to the previous model, due to the presence of dropout, overfitting is prevented. Also, the validation data loss after 10 epochs is 3.0165, which is similar to the previous model.

True: a person in the distance hikes among with stars visible in the sky
Greedy: a man is climbing a rock wall a mountain
Beam (K=3): a man is climbing a rock wall a mountain
Beam (K=5): a man is climbing a rock wall a mountain
Beam (K=10): a man in a red shirt is climbing a rock wall a tree

True: a man holding the hand of another man who has a tattoo on his arm
Greedy: a man in a white shirt is sitting on a bench with a white dog
Beam (K=3): a man in a red shirt is sitting on a bench a book
Beam (K=5): a man in a red shirt is sitting on a bench a book
Beam (K=10): a man in a red shirt is sitting on a bench in front of a crowd of people

True: one very fat sumo wrestler wrestles with a wrestler while an dressed in green watches
Greedy: two men are playing with a ball
Beam (K=3): a man in a white shirt is standing in front of a crowd of people
Beam (K=5): a man in a white shirt is standing in front of a crowd of people
Beam (K=10): a man in a white shirt is standing in front of a crowd of people

True: a black dog in the middle of running or leaping
Greedy: two dogs are running in the snow
Beam (K=3): two dogs are running in the snow
Beam (K=5): two dogs are running in the snow
Beam (K=10): two dogs are running in the snow



True: a boy in a shirt and tie jumps off a staircase
Greedy: a woman in a red shirt is jumping on a trampoline
Beam (K=3): a man in a red shirt is skateboarding on a skateboard in a skate park
Beam (K=5): a man in a red shirt is doing a trick on a skateboard
Beam (K=10): a man in a red shirt is doing a trick on a skateboard

Figure 18: Captions generated for a few sample images for the model with a small dictionary.

Table 3: Evaluation metrics for the model with a small dictionary.

	MicroAcc	MacroAcc	BLEU-1	BLEU-2	BLEU-3
Greedy	0.221242	0.229228	0.551296	0.365600	0.229846
Beam K=3	0.231269	0.238014	0.567192	0.378439	0.245444
Beam K=5	0.230941	0.237559	0.568355	0.382278	0.251422
Beam K=10	0.228382	0.231685	0.560551	0.374397	0.247933

As is clear from the values in the table above, the quality has dropped compared to the previous metrics. Thus, we will continue with the same previous tokenizer.

1.5.5 Embedding Vector with Length 150

By increasing the length of the token embedding vector, if it helps the model to learn a richer meaning for each token, the model's performance may improve. So, we create a model with dropout and the original tokenizer with an embedding length of 150 and train it for 10 epochs.

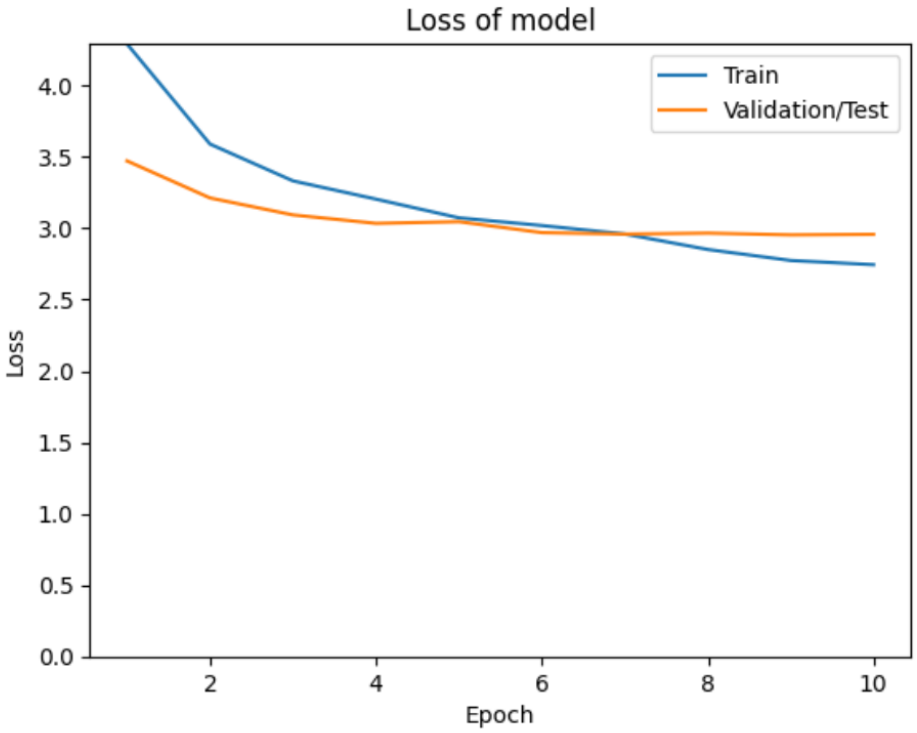


Figure 19: Loss changes of the model with embedding size 150 during training by epoch.

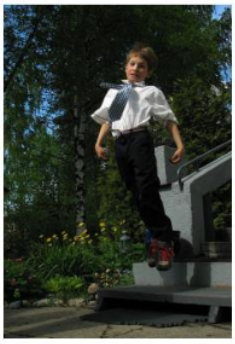
It is clear that again, overfitting is prevented, and the model's loss on the validation data after 10 epochs reaches 2.9578, which is better than the previous models.

True: a person in the distance hikes among with stars visible in the sky
Greedy: a man is standing on a rocky mountain
Beam (K=3): a man is standing in front of a mountain
Beam (K=5): a man is standing in front of a mountain
Beam (K=10): a man is standing in front of a mountain

True: a man holding the hand of another man who has a tattoo on his arm
Greedy: a man in a blue shirt and a black hat is sitting on a bench a cigarette
Beam (K=3): a man in a white shirt and blue jeans is sitting on a bench a cigarette
Beam (K=5): a man in a white shirt and blue jeans is sitting on a bench reading a newspaper
Beam (K=10): a man in a red shirt is standing in front of a brick building

True: one very fat sumo wrestler wrestles with a wrestler while an umpire dressed in green watches
Greedy: two men are wrestling on a trampoline to catch a ball
Beam (K=3): a man in a white shirt is jumping into the air to catch a tennis ball in its mouth
Beam (K=5): a man in a white shirt is jumping into the air to catch a tennis ball in its mouth
Beam (K=10): a man in a white shirt is jumping into the air to catch a tennis ball in its mouth

True: a black dog in the middle of running or leaping
Greedy: a brown dog runs through the grass
Beam (K=3): two dogs run through the grass
Beam (K=5): two dogs run through the grass
Beam (K=10): two dogs run through the grass



True: a boy in a shirt and tie jumps off a staircase
Greedy: a woman in a white dress walks down a sidewalk a parking meter
Beam (K=3): a woman in a white dress walks down a sidewalk
Beam (K=5): a woman in a white dress walks down a sidewalk
Beam (K=10): a woman in a white dress walks down a sidewalk

Figure 20: Captions generated for a few sample images for the model with embedding size 150.

Table 4: Evaluation metrics for the model with embedding size 150.

	MicroAcc	MacroAcc	BLEU-1	BLEU-2	BLEU-3
Greedy	0.229487	0.237735	0.536161	0.348303	0.218271
Beam K=3	0.231479	0.240317	0.557379	0.370610	0.238766
Beam K=5	0.229925	0.238243	0.558502	0.370435	0.238130
Beam K=10	0.229560	0.237379	0.553082	0.369200	0.238274

From the evaluation metrics, it is clear that the performance of this model has dropped compared to the models with an embedding size of 50. One of the possible reasons for this is the increase in the number of model parameters and the need for more data and time to learn the new parameters.

1.5.6 Embedding Vector with Length 300

Similar to before, we increase the embedding vector length to 300, which is a common value in language models, and train a model for 10 epochs.

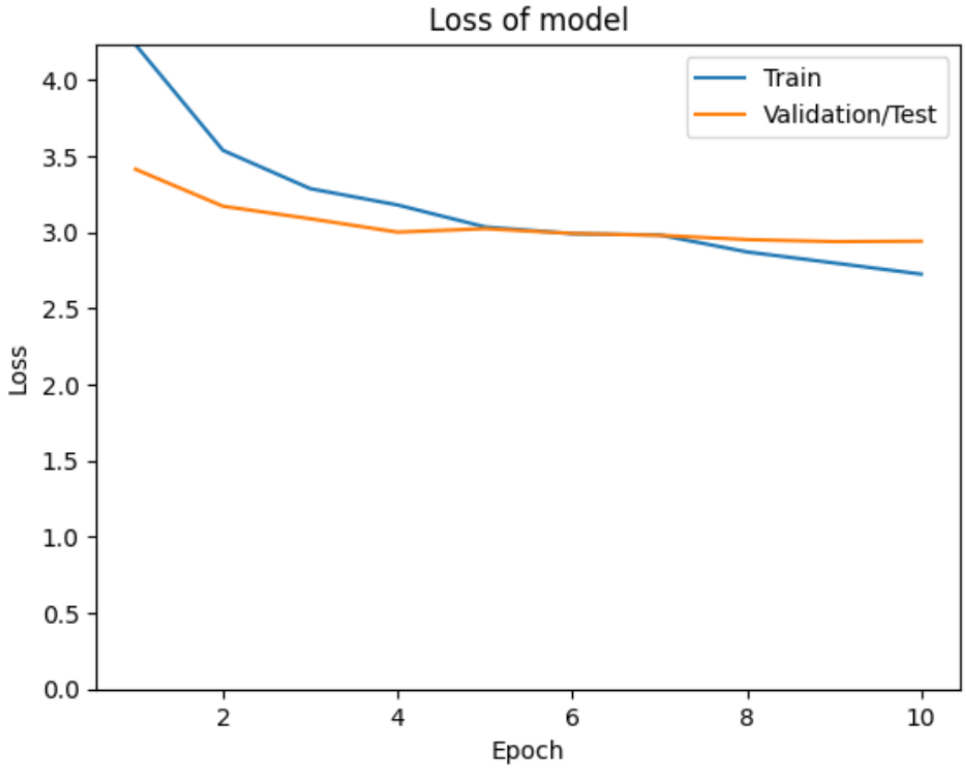


Figure 21: Loss changes of the model with embedding size 300 during training by epoch.

The model's loss on the validation data after 10 epochs reaches 2.9421, which is better than the previous models.

True: a person in the distance hikes among with stars visible in the sky

Greedy: a person is climbing a rock face a rock

Beam (K=3): a man is climbing up a rock face a rock to his left his racket

Beam (K=5): a man is climbing up a steep rock face a rock to his left into the air his racket

Beam (K=10): a man is climbing up a steep rock face a rock into the air above his head to his left

True: a man holding the hand of another man who has a tattoo on his arm

Greedy: a man in a blue shirt is sitting on a bench a cigarette his hand his arm around a man in a white shirt and black pants is standing

Beam (K=3): a man in a blue shirt is sitting on a bench a cigarette a cigarette to his left his racket

Beam (K=5): a man in a blue shirt and jeans is sitting on a bench in front of a brick building his hand to his left

Beam (K=10): a man in a blue shirt is sitting on a bench in front of a brick building a cigarette to his left into the air

True: one very fat sumo wrestler wrestles with a wrestler while an umpire dressed in green watches

Greedy: a man in a white shirt and jeans is jumping over a hurdle his hand to his mouth his hand to his left

Beam (K=3): a man in a white shirt and jeans is jumping over a hurdle a ramp to his left to his left

Beam (K=5): a man in a white shirt and jeans is doing a trick on a ramp to his left to his left his racket

Beam (K=10): a man in a white shirt and jeans is doing a back flip on a ramp into the air to his left

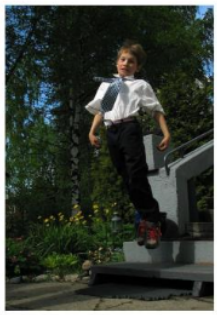
True: a black dog in the middle of running or leaping

Greedy: a black dog is running in the snow a tennis ball

Beam (K=3): a black dog is running in the snow a tennis ball

Beam (K=5): a black dog is running in the snow a tennis ball

Beam (K=10): a black dog is running through the snow



True: a boy in a shirt and tie jumps off a staircase

Greedy: a woman in a black hoodie and jeans is walking on a sidewalk a city street his hand a cigarette to a man who is

Beam (K=3): a woman in a blue shirt is walking on a sidewalk a cigarette his hand while another man watches

Beam (K=5): a woman in a blue shirt and jeans is walking on a sidewalk a city street his hand while another man watches

Beam (K=10): a woman in a white shirt and jeans is riding a bicycle down a ramp into the air his racket into the air

Figure 22: Captions generated for a few sample images for the model with embedding size 300.

Table 5: Evaluation metrics for the model with embedding size 300.

	MicroAcc	MacroAcc	BLEU-1	BLEU-2	BLEU-3
Greedy	0.226392	0.236276	0.461385	0.295351	0.182981
Beam K=3	0.231373	0.240192	0.458475	0.293568	0.188023
Beam K=5	0.228925	0.238368	0.454495	0.289496	0.185276
Beam K=10	0.225481	0.233207	0.457976	0.291854	0.183927

It can be seen that the evaluation metrics have decreased, and thus, by examining larger embedding sizes, the model's performance decreased. Therefore, we will use an embedding size of 50 in the continuation.

1.5.7 Training the Best Model for 40 Epochs

Now that we have evaluated different models on the validation data, we consider the best model, which is the model with an embedding size of 50 and dropout, and train it for 40 epochs. We examine the loss at different epochs to find the appropriate number of epochs for training on the test data and perform more detailed analyses.

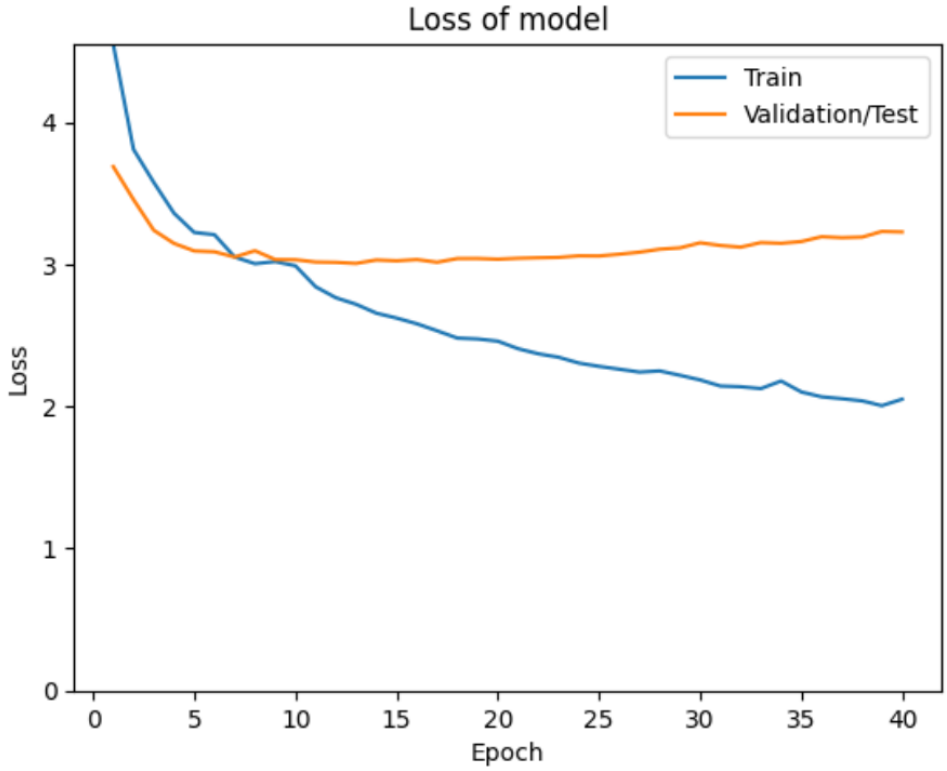


Figure 23: Loss changes of the model in 40 epochs.

From the graph above, it is clear that despite using dropout, overfitting still occurs, and the validation data loss increases after a while. Also, the lowest model loss on the validation

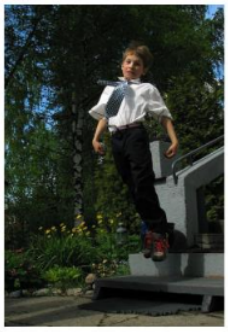
data is 3.0079, which occurs at epoch 13.

True: a person in the distance hikes among with stars visible in the sky
Greedy: a person standing on a mountaintop with a mountain in the background
Beam (K=3): a person standing on top of a mountain the ocean the horizon
Beam (K=5): a person is standing on top of a mountain with his arms in the air the beach
Beam (K=10): a person is standing on top of a mountain with his arms in the air the beach

True: a man holding the hand of another man who has a tattoo on his arm
Greedy: a man in a kilt is holding a sword a tripod a tripod
Beam (K=3): a man in a cowboy hat uses a tool box to a group of people
Beam (K=5): a man in a cowboy hat uses a tool box to a group of people
Beam (K=10): a man in a cowboy hat uses a tool box to a large crowd people

True: one very fat sumo wrestler wrestles with a wrestler while an umpire dressed in green watches
Greedy: a man in a red shirt and black shorts is doing a handstand on a wakeboard
Beam (K=3): a group of men are playing with a fire baton water
Beam (K=5): a group of men are playing with a fire baton water
Beam (K=10): two men are wrestling in a ring boxing

True: a black dog in the middle of running or leaping
Greedy: a brown dog is running through the sand the beach the beach
Beam (K=3): a brown dog is running through the sand the beach the beach
Beam (K=5): a brown dog is running through the sand the beach the beach
Beam (K=10): a brown dog is running through the sand the beach a stick its mouth



True: a boy in a shirt and tie jumps off a staircase
Greedy: a man in a black shirt and jeans is jumping on a trampoline a skateboard
Beam (K=3): a man in a white shirt and blue jeans is doing a back flip in the park a fence
Beam (K=5): the man is wearing a black shirt and blue jeans is running on a concrete surface a leafy day
Beam (K=10): the man is wearing a black shirt and blue jeans is running on a concrete surface the street

Figure 24: Captions generated for a few sample images after training for 40 epochs.

Table 6: Evaluation metrics of the model after training for 40 epochs.

	MicroAcc	MacroAcc	BLEU-1	BLEU-2	BLEU-3
Greedy	0.225810	0.235044	0.538645	0.341239	0.208188
Beam K=3	0.233323	0.242256	0.522502	0.334616	0.210917
Beam K=5	0.229185	0.238734	0.512771	0.326633	0.203052
Beam K=10	0.229237	0.236271	0.500319	0.318688	0.200355

The evaluation metrics have also slightly decreased compared to before. So, with long-term training, the model's performance can decrease. Therefore, for training the final model on the test data, we will use fewer epochs.

1.5.8 Other Investigations

We not only trained the mentioned models several times but also investigated the effect of other methods. Among the investigations we conducted, we can mention the use of a variable learning rate, using a model with larger embedding sizes with a smaller dictionary, giving zero vectors to the hidden state of GRU and the cell state of LSTM, using two dropout layers, and other methods. But the model's loss did not improve much. In the end, the model with an embedding size of 50, with three dropout layers, and by giving the feature vector as the initial hidden state of LSTM and GRU and the cell state of LSTM, was considered as the final model for evaluation on the test data.

1.6 Final Evaluation on Test Data

Given the previous section, we train the final model for 13 epochs on the combination of training and validation data.

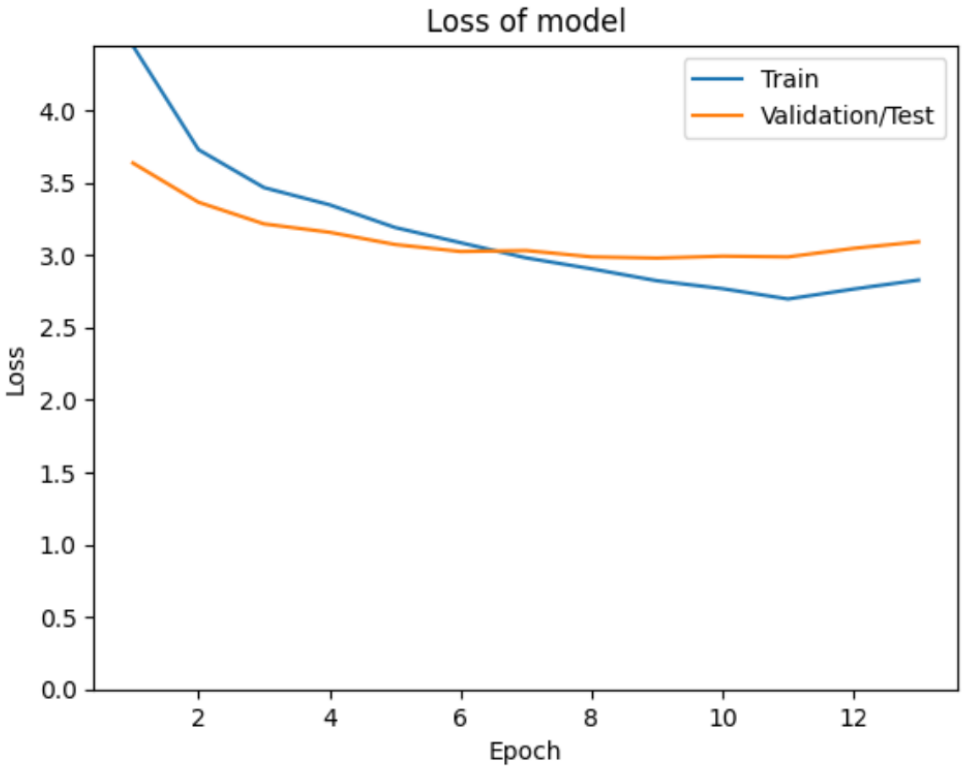
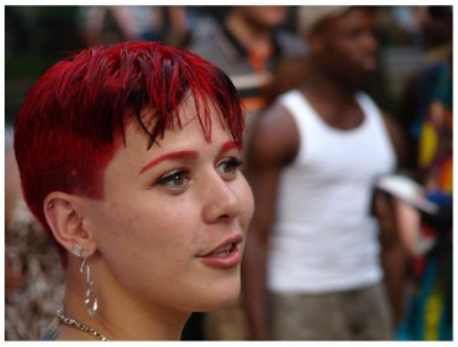


Figure 25: Graph of the model's loss changes during training.

From the loss graph, it can be seen that overfitting occurs, and the lowest model loss at the ninth epoch is 2.9783, and finally, after 13 epochs, the loss reaches 3.09. But to avoid leaking test information into the training, we consider the final model.

True: a blonde woman is next to a man in a red shirt
Greedy: a woman and a woman are standing in front of a large statue
Beam (K=3): a group of people are posing for a picture
Beam (K=5): a group of people are posing for a picture
Beam (K=10): a group of people are posing for a picture

True: a black dog running through the snow
Greedy: a black dog with a red collar is running through the snow
Beam (K=3): a black dog is running through the snow with a stick in its mouth
Beam (K=5): a black dog is running through the snow with a stick in its mouth
Beam (K=10): a black dog is running through the snow with a stick in its mouth



True: a closeup of a woman with short red hair
Greedy: a woman wearing a white shirt and a hat is standing in front of a large rock
Beam (K=3): a woman wearing a white shirt is standing in front of a crowd of people
Beam (K=5): a young woman wearing a red shirt is standing in front of a crowd of people
Beam (K=10): a woman wearing a red shirt is standing in front of a crowd of people

True: a lone surfboarder on a white surfboard flying through the air over a wav
Greedy: a surfer is riding a wave in the air
Beam (K=3): a surfer is riding a wave in the air
Beam (K=5): a surfer is riding a wave in the air
Beam (K=10): a surfer is riding a wave in the air

True: a man dressed in an indian costume plays the drums
Greedy: a man in a red costume and a mask is standing in front of a crowd of people
Beam (K=3): a man in a white costume is holding a sign that says free hugs
Beam (K=5): a man in a white costume is standing in front of a large crowd
Beam (K=10): a man in a white costume is standing in front of a large crowd

Figure 26: Captions generated by the final model for a few sample test images.

Table 7: Final evaluation metrics of the model on the evaluation data.

	MicroAcc	MacroAcc	BLEU-1	BLEU-2	BLEU-3
Greedy	0.225810	0.235044	0.538645	0.341239	0.208188
Beam K=3	0.233323	0.242256	0.522502	0.334616	0.210917
Beam K=5	0.229185	0.238734	0.512771	0.326633	0.203052
Beam K=10	0.229237	0.236271	0.500319	0.318688	0.200355

1.7 Conclusion

From the table above, it is clear that the model has reached a similar performance to that observed on the validation data. As we stated, the accuracy and loss are different from the paper, while the BLEU is close to the value reported in the paper.

In this exercise, we tried to build a model by implementing the paper's method, which, by combining LSTM and GRU and using ResNet50 as a feature extractor, can generate a suitable caption for an image.

For this, we first wrote a tokenizer and preprocessed the images of the Flickr8k dataset along with the five captions of each image. Then we implemented the model, and finally, by examining the details that were not mentioned in the paper, we tried to create the best possible model. With the help of accuracy and BLEU-1 to BLEU-4 metrics, we evaluated the models and chose the best model.

The final model on the test data reached the performance stated in the paper in terms of BLEU-1, but the model's loss was higher than the paper's loss, and the accuracy was also lower. Regarding accuracy, since it is not a common metric and given the presence of five captions for each image, it is possible that our implementation differs from the paper's, but the difference in the loss function is strange given the similarity in BLEU-1. Unfortunately, the paper has omitted many important details and the presentation of results, and for this reason, we cannot be sure that we have gone wrong somewhere. Visually, the generated captions correctly describe the generalities of the images, but they have errors in expressing some details such as colors, numbers, and gender.

Also, for generating text, like the paper, we used both greedy and Beam Search methods, but we did not observe the significant difference between the greedy and beam search methods that was stated in the paper. Although the captions of some images of some models had fewer incorrect details in beam search with a size of 10, we did not observe much difference in terms of the metric values.

Overall, this exercise led to gaining skills in implementing recurrent models and tokenizers and the necessary preprocessing, as well as theoretical knowledge such as the metrics used in image captioning and text generation methods.