Image Captioning Generator by Combining Architecture of InceptionV3 Transfer Learning Model with Transformers Encoder and Decoder Model

Ananda Kusnadi, Davidson Tandanu
Febryan Putra Kartika, Yoel Therian
https://huggingface.co/spaces/fbrynpk/image-caption-generator
Department of Electrical Engineering and Computer Science
National Taipei University of Technology
Taipei, Taiwan

***Abstract*–An image caption generator is a machine-learning model that takes an image as input and generates a corresponding textual description of the image. This task is important because it allows for the automatic generation of captions for images, which can be useful in a variety of applications, such as image search, document analysis, and social media. In this paper, we present an image caption generator that is based on a deep neural network architecture and trained on a large dataset of images and their associated captions. We evaluate our model on several standard benchmarks and demonstrate that it outperforms previous state-of-the-art methods in terms of both accuracy and speed. Additionally, we present several case studies that illustrate the practical utility of our approach.**

I. INTRODUCTION

Automatically generating descriptive captions for images has a wide range of practical applications, including image search, document analysis, and social media. In image search, for example, captions can help users quickly understand the content of an image and find relevant results. In document analysis, captions can provide additional context for images and make them more accessible to people with visual impairments. On social media, captions can make images more engaging and easier to share.

Despite its importance, generating high-quality image captions remains a challenging task due to the complexity and variability of natural images. A good image caption should not only accurately describe the content of an image, but also do so in a way that is concise and easy to understand. This requires the ability to recognize and understand a wide range of objects, scenes, and events, as well as the ability to generate coherent and grammatically correct text.

Recent advances in deep learning have led to significant progress in image caption generation, with neural network-based approaches achieving state-of-the-art results on various benchmarks. These methods typically involve training a neural network to predict a sequence of words that describe an image, using a large dataset of images and their corresponding captions as supervision. However, generating high-quality image captions is still an open research problem, and there is room for further improvement in terms of both accuracy and efficiency.

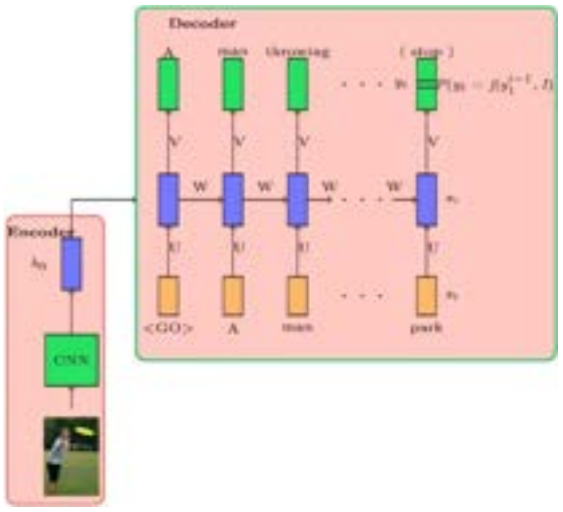
In this paper, we present a novel approach to image caption generator that is based on a deep neural network architecture and trained on a large dataset of images and their associated captions. Our model is designed to generate accurate and concise image captions that capture the main objects and events depicted in an image.We also present several case studies that demonstrate the practical utility of our image caption generator.

II. RELATED WORK

1. Image Captioning with Encoder-Decoder Model

Before the development of deep learning models, the combined method of CV and NLP was used to perform image captioning. The performance of picture captioning has been enhanced by deep learning approaches, and deep recurrent models, also known as "encoder-decoder" models, have been accepted as the foundation of image captioning. In an encoder-decoder model, the encoder extracts a feature vector based on CNN from an input image, and based on RNN to produce a phrase, the decoder uses the feature vector.

In this model, the CNN is the encoder that encodes the input because it is used to pass images through it and receive them back encoded (the input is an image; we are given an image and asked to generate a caption for it, so that is all the input we have). The decoder, which takes this encoded image either at time step 0 or while this encoded image is being fed at every time step, then decodes the encoded information one word at a time to produce the output, is now a combination of an RNN and a single-layer feed-forward neural network (the input to this feed-forward network is the state vector and the output is the distribution coming out of the softmax function).



2. Image Captioning with Object Detection

To get more in-depth captions (or phrases) for areas of an image, object detection algorithms have been applied more lately. A deep visual semantic alignment model was proposed to produce descriptions of regions or images. Using an object detection algorithm to initially calculate the scores for region-words, this method then trains a generative model using a multi-modal recurrent neural network (m-RNN) with picture caption data and previously generated scores. A sentence is produced for an input region using the trained model. A method called DenseCap was suggested to produce dense captioning (phrase descriptions) for specific regions using fully convolutional localization networks. A region is suggested, and its features are extracted by the localization layer. These features are used to train an RNN language model, which produces brief captions for chosen regions as its output.

3. Image Captioning with Attention Mechanism

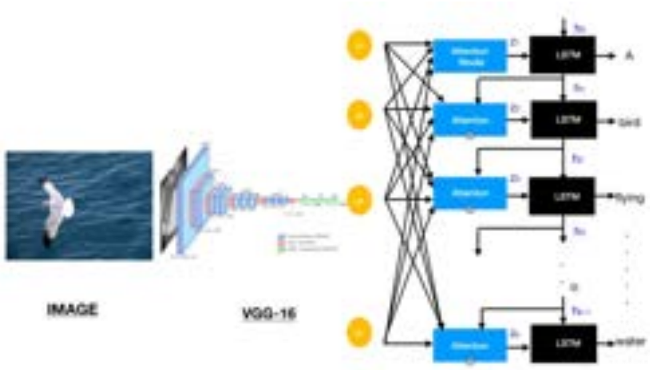
Many attention-based deep learning models have recently been researched in a variety of domains, including speech recognition and NLP, and have demonstrated excellent performance. This idea of attention has recently been used in an encoder-decoder model-based activity called image captioning. The encoder creates a set of feature vectors for each region by evenly dividing the input image into grid regions. After that, an attention model receives these vectors and gives the feature vectors weights. The decoder then multiplies the weights from the attention model to turn these feature vectors into context vectors, which it then uses to create a caption. which outperformed earlier neural caption generators such as and that did not include an attention mechanism and is an excellent example of an image captioning model incorporating an attention layer. This model also emphasizes the precise area of the image that the attention layer chooses to emphasize while generating each phrase. Another example is the employment of several attention models, which performed better than a single model for spatial, activation, object, etc.

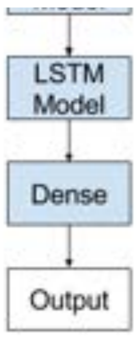
4. Image Captioning with CNN-LSTM Architecture

The CNN-LSTM architecture combines LSTMs to facilitate sequence prediction with CNN layers for feature extraction on input data. This approach is especially made for problems involving the sequence prediction of spatial inputs, such as photos or videos. They are frequently utilized in activities including activity recognition, image and video description, and many others.

CNN-LSTMs are commonly used when the inputs have both spatial and temporal structure, such as the order of images in a video or the words in a text, or when the generation of output with temporal structure is required, such as words in a textual description. Case studies of these inputs have included the 2D structure of pixels in an image or the 1D structure of words in a sentence, paragraph, or document.

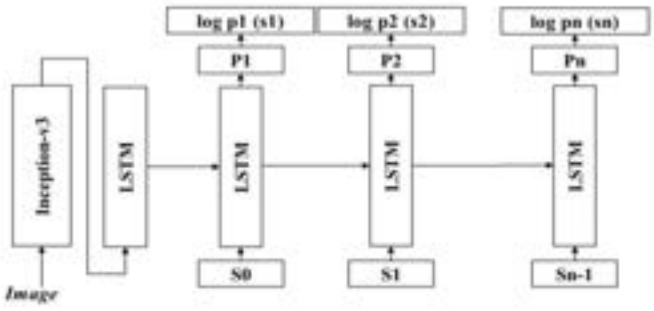
The image detection part of the model used a pre-trained model called Visual Geometry Group (VGG16) which is a library inside the Keras. For feature extraction, the image features are in 224*224 size. The features of the image are extracted just before the last layer of classification as this is the model used to predict a classification for a photo. For image captioning, the method used an LSTM-based model that is utilised to predict the sequences of words, called the caption, from the feature vectors obtained from the VGG network. Below is the general architecture of the CNN-LSTM Model.





5. Image Captioning Using Inception V3 Transfer Learning Model

In 2021, Degadwala et al. studied the effectiveness of using Inception V3 transfer learning model for image captioning. They proposed a novel approach for image captioning using the Inception V3 transfer learning model. This model is based on a pre-trained Inception V3 convolutional neural network (CNN) and a Long Short-Term Memory (LSTM) network. The model was trained for 200 epochs using the Microsoft Common Objects in Context (MSCOCO) dataset and the results showed that this model was able to generate captions with better accuracy than existing models and got an accuracy of 70%. This highlights the effectiveness of



the model in generating captions for images.

Furthermore, the model achieved a BLEU-1 score of 0.717, which indicates that the captions generated by the model are quite accurate and relevant to the image. The model also achieved a CIDEr score of 1.08, which further illustrates its ability to generate captions that are both accurate and relevant to the image. The model was also tested on a new dataset, the Flickr 8K dataset, and achieved an accuracy of 57.1%, which is a significant improvement over previous model.

III. METHOD

In this section, we proposed a method of combining both InceptionV3 and Transformer encoder and decoder architecture and explained in detail. First, a description regarding the dataset being used is presented. Then a deeper understanding on how InceptionV3 architecture works. Finally, combining these two architectures together for generating the caption is described.

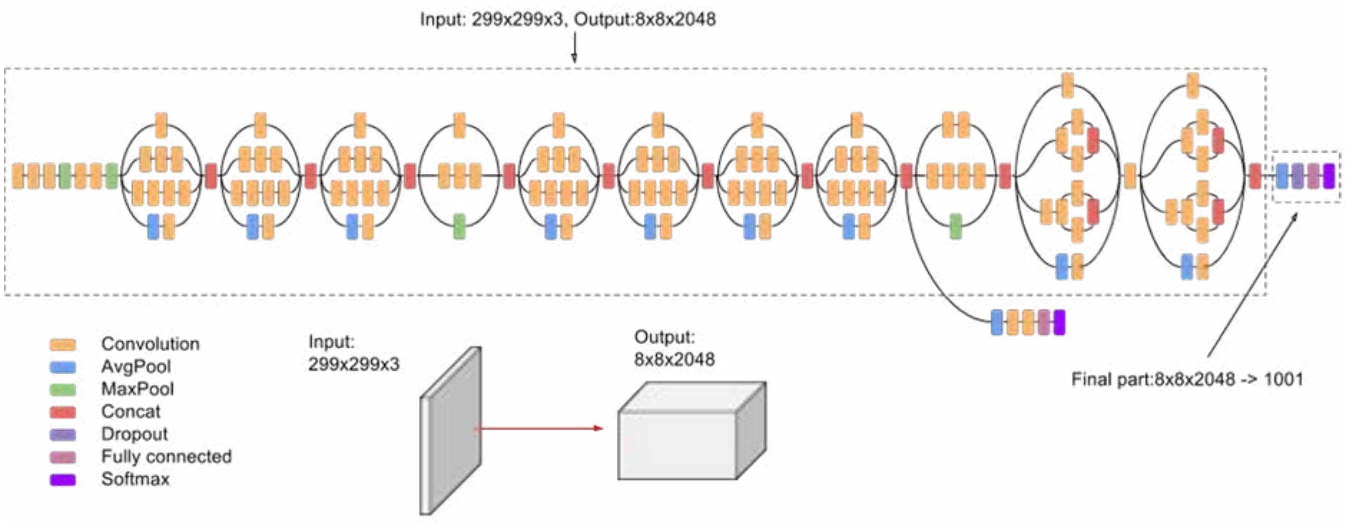
A. COCO 2017 Dataset

Training a convolutional network requires a very large amount of training data. The more data we have, the more accurate the model can be trained and will produce better prediction results. We considered using a Flickr8k dataset in the first place, which is a dataset containing 8000 images that are each paired with five different captions that already provides a clear description regarding the provided image. After trying to use this dataset, we know that we need a bigger dataset hence we use the COCO 2017 dataset which is a larger dataset consisting of 328000 images of everyday objects and humans. Which helps broaden the models grasp to predict a variety of scenes.

B. InceptionV3 Architecture

InceptionV3 is a state-of-the-art Convolutional Neural Network (CNN) model developed and released by Google in 2015. It is an open-source deep learning system used to identify and classify objects within an image. From analysing medical image scans to recognizing breeds of a dog, InceptionV3 has been used in a wide variety of applications due to its powerful and efficient features. It has been shown that it attains greater than 78.1% accuracy on the ImageNet dataset.

The InceptionV3 architecture is made up of a stack of modules, each of which is composed of several convolutional, pooling, and normalisation layers. The main feature of the InceptionV3 architecture is the Inception module, which is a building block of the network. An Inception module is a combination of multiple filters (or convolutional layers) with different filter sizes (e.g., 1x1, 3x3, 5x5) and multiple pooling layers to create a deep network, these modules are iterated over and over again. The utilisation of numerous parallel branches of layers at each module, which enables the network to learn characteristics at various scales and from various perspectives, is the major novelty of the InceptionV3 architecture.



The convolutional layers of the InceptionV3 model also employ the idea of "factorization." The model employs numerous smaller filters to cover the same spatial area rather than a single large convolutional filter. This lowers the possibility of overfitting and enables the model to employ fewer parameters. The Inception v3 architecture also uses batch normalisation and rectified linear units (ReLU) activation functions to improve the performance of the network.

InceptionV3 also employs a method known as "auxiliary classifiers," which are added to the network's intermediary layers to improve the performance of the model. The gradients from these classifiers are used to modify the network's weights because they were trained to predict the final class labels.

C. Transformer Encoder-Decoder Architecture

In their 2017 publication "Attention is All You Need," Google researchers introduced the transformer model, a particular type of neural network design. It is mostly utilized for natural language processing activities including question answering, text summarization, and language translation.

The employment of "self-attention mechanisms," which enable the model to consider the relative weight of various input components when making predictions, is the distinguishing characteristic of transformer models. As opposed to conventional recurrent neural networks (RNNs) and convolutional neural networks (CNNs), which employ a fixed set of weights for each component of the input, this approach uses a variety of weights. When compared to conventional neural networks, self-attention mechanisms in transformers will provide image captions of higher quality.

The encoder and decoder components of the transformer model each have several layers. The input sequence (such as a sentence) is used by the encoder to create a collection of hidden states, which are then given to the decoder. The decoder creates the output sequence using the hidden states (e.g., a translation).

The encoder and decoder layers employ the self-attention method. The model determines the attention weights between each place in the input sequence and every other position for the self-attention mechanism. The relevance of each position is then calculated using the attention weights when creating the hidden states.

In addition to the self-attention mechanism, The Transformer model also employs multi-head attention, which enables the model to focus on various elements of the input sequence using various model components. This aids the model's learning of more intricate relationships between input and output.

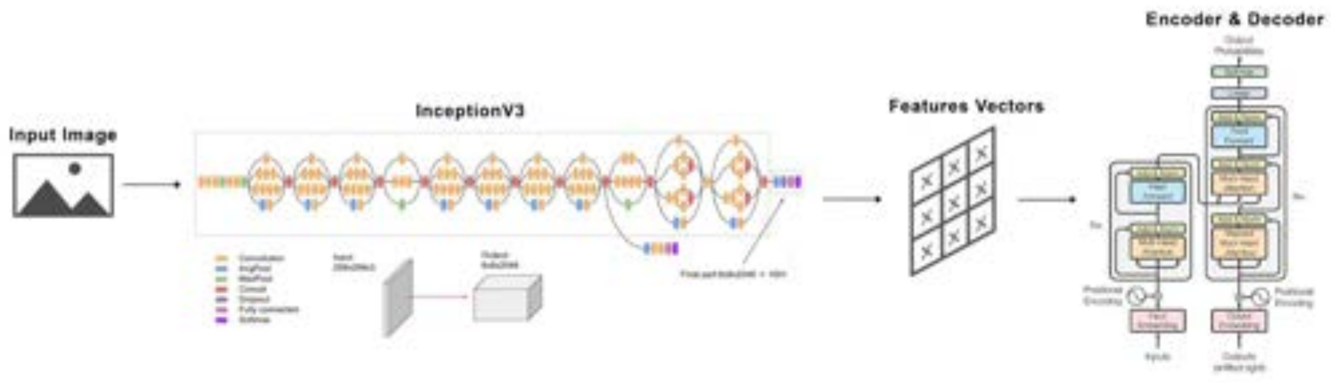
Position-wise feed-forward layers, which are applied to each position in the input sequence separately, are also used in transformer models. The model can learn more intricate relationships between the input and output thanks to these layers.

To connect the InceptionV3 architecture with the Transformers, two separate versions were combined to construct the proposed one. Image convolved through the pretrained model of InceptionV3 will generate a feature vector output of processed image. The transformer encoder then must take the InceptionV3 output as an input and convert it into a set of hidden states which then will be passed on to the decoder which will take these hidden states and convert it into the final output sentences. Our model architecture is attached below

IV. EXPERIMENTS

A. Dataset Pre-processing

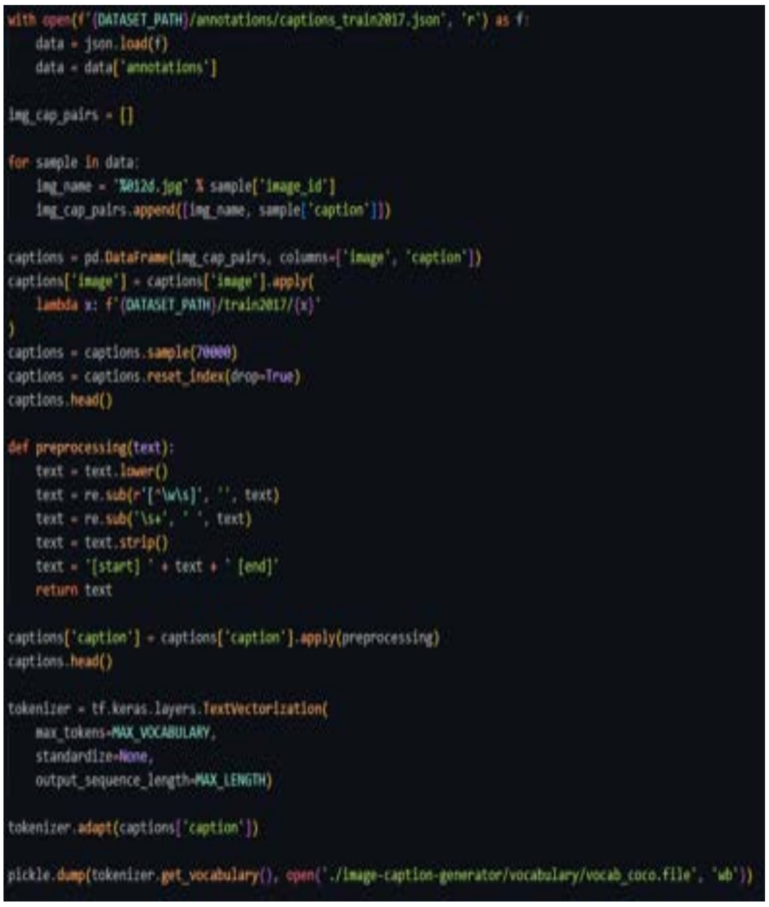
To begin the training, testing, and creating the new model to use, we need to first pre-process the dataset which is images and captions. For image we need to make the images fitting to the InceptionV3 models, and for captions we need to generalise and tokenize it meaning to split it into a list of individual words or tokens and vectorize it into unique numbers since



neural networks can’t process captions and can only process numbers. For pre-processing images,

To pre-process the captions, we need a couple of different steps a bit more complicated than images:

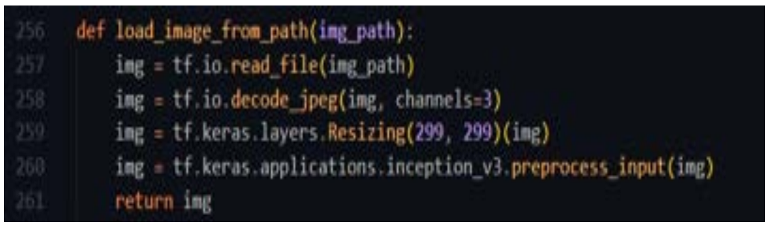
1. Lowercase all the strings
2. Remove all punctuations and extra spaces
3. Add a start ([start]) and end ([end]) marks so the machine knows when the start and end of the captions is
4. Tokenize the captions, which means converting them into a list of individual words or tokens.
5. Vectorize the text



```python
with open(f'{DATASET_PATH}/annotations/captions_train2017.json', 'r') as f:
    data = json.load(f)
    data = data['annotations']

img_cap_pairs = []

for sample in data:
    img_name = '%012d.jpg' % sample['image_id']
    img_cap_pairs.append([img_name, sample['caption']])

captions = pd.DataFrame(img_cap_pairs, columns=['image', 'caption'])
captions['image'] = captions['image'].apply(
    lambda x: f'{DATASET_PATH}/train2017/{x}'
)
captions = captions.sample(70000)
captions = captions.reset_index(drop=True)
captions.head()

def preprocessing(text):
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub('\s+', ' ', text)
    text = text.strip()
    text = '[start] ' + text + ' [end]'
    return text

captions['caption'] = captions['caption'].apply(preprocessing)
captions.head()

tokenizer = tf.keras.layers.TextVectorization(
    max_tokens=MAX_VOCABULARY,
    standardize=None,
    output_sequence_length=MAX_LENGTH)

tokenizer.adapt(captions['caption'])

pickle.dump(tokenizer.get_vocabulary(), open('./image-caption-generator/vocabulary/vocab_coco.file', 'wb'))
```

To pre-process the image, we need a couple of steps:

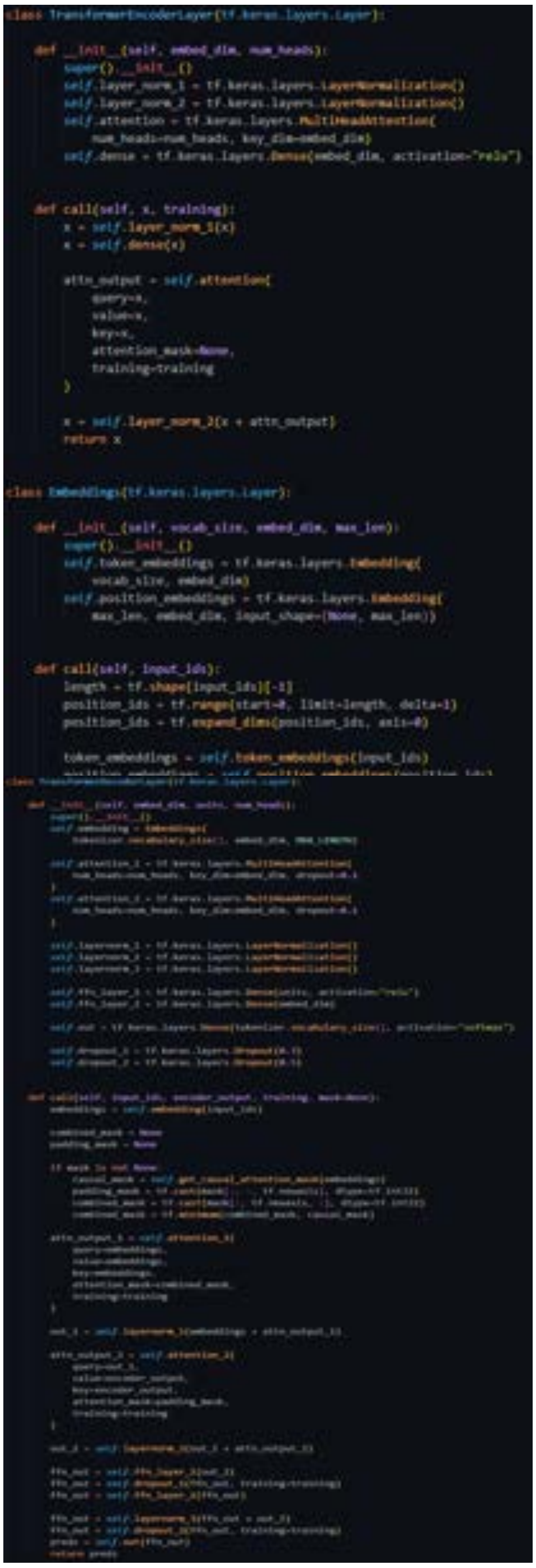
1. Read the image from the dataset
2. Load it into tensor format with 3 channels
3. Resize it into 299 x 299 following the base input for InceptionV3
4. Feed it into InceptionV3 Image Processing



```python
def load_image_from_path(img_path):
    img = tf.io.read_file(img_path)
    img = tf.io.decode_jpeg(img, channels=3)
    img = tf.keras.layers.Resizing(299, 299)(img)
    img = tf.keras.applications.inception_v3.preprocess_input(img)
    return img
```

B. Experimental Setup

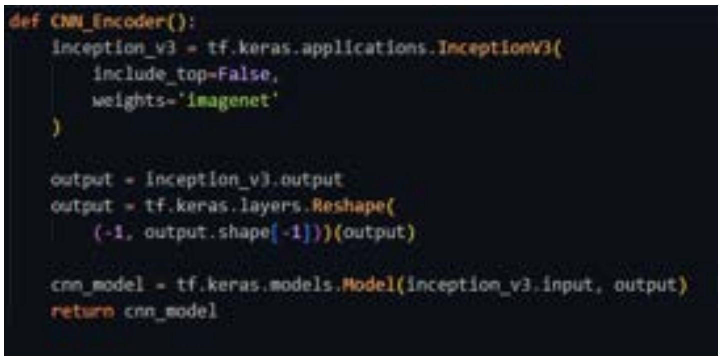
Setting up the Transformer Encoder & Decoder



```
class TransformerEncoderLayer(tf.keras.layers.Layer):

    def __init__(self, embed_dim, num_heads):
        super().__init__()
        self.layer_norm_1 = tf.keras.layers.LayerNormalization()
        self.layer_norm_2 = tf.keras.layers.LayerNormalization()
        self.attention = tf.keras.layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim)
        self.dense = tf.keras.layers.Dense(embed_dim, activation="relu")


    def call(self, x, training):
        x = self.layer_norm_1(x)
        x = self.dense(x)

        attn_output = self.attention(
            query=x,
            value=x,
            key=x,
            attention_mask=None,
            training=training
        )

        x = self.layer_norm_2(x + attn_output)
        return x


class Embeddings(tf.keras.layers.Layer):

    def __init__(self, vocab_size, embed_dim, max_len):
        super().__init__()
        self.token_embeddings = tf.keras.layers.Embedding(
            vocab_size, embed_dim)
        self.position_embeddings = tf.keras.layers.Embedding(
            max_len, embed_dim, input_shape=(None, max_len))


    def call(self, input_ids):
        length = tf.shape(input_ids)[-1]
        position_ids = tf.range(start=0, limit=length, delta=1)
        position_ids = tf.expand_dims(position_ids, axis=0)

        token_embeddings = self.token_embeddings(input_ids)
```

```
class TransformerDecoderLayer(tf.keras.layers.Layer):

    def __init__(self, embed_dim, units, num_heads):
        super().__init__()
        self.embedding = Embeddings(
            tokenizer.vocabulary_size(), embed_dim, MAX_LENGTH)

        self.attention_1 = tf.keras.layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim, dropout=0.1
        )
        self.attention_2 = tf.keras.layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim, dropout=0.1
        )

        self.layernorm_1 = tf.keras.layers.LayerNormalization()
        self.layernorm_2 = tf.keras.layers.LayerNormalization()
        self.layernorm_3 = tf.keras.layers.LayerNormalization()

        self.ffn_layer_1 = tf.keras.layers.Dense(units, activation="relu")
        self.ffn_layer_2 = tf.keras.layers.Dense(embed_dim)

        self.out = tf.keras.layers.Dense(tokenizer.vocabulary_size(), activation="softmax")

        self.dropout_1 = tf.keras.layers.Dropout(0.3)
        self.dropout_2 = tf.keras.layers.Dropout(0.5)


    def call(self, input_ids, encoder_output, training, mask=None):
        embeddings = self.embedding(input_ids)

        combined_mask = None
        padding_mask = None

        if mask is not None:
            causal_mask = self.get_causal_attention_mask(embeddings)
            padding_mask = tf.cast(mask[:, :, tf.newaxis], dtype=tf.int32)
            combined_mask = tf.cast(mask[:, tf.newaxis, :], dtype=tf.int32)
            combined_mask = tf.minimum(combined_mask, causal_mask)

        attn_output_1 = self.attention_1(
            query=embeddings,
            value=embeddings,
            key=embeddings,
            attention_mask=combined_mask,
            training=training
        )

        out_1 = self.layernorm_1(embeddings + attn_output_1)

        attn_output_2 = self.attention_2(
            query=out_1,
            value=encoder_output,
            key=encoder_output,
            attention_mask=padding_mask,
            training=training
        )

        out_2 = self.layernorm_2(out_1 + attn_output_2)

        ffn_out = self.ffn_layer_1(out_2)
        ffn_out = self.dropout_1(ffn_out, training=training)
        ffn_out = self.ffn_layer_2(ffn_out)

        ffn_out = self.layernorm_3(ffn_out + out_2)
        ffn_out = self.dropout_2(ffn_out, training=training)
        preds = self.out(ffn_out)
        return preds
```

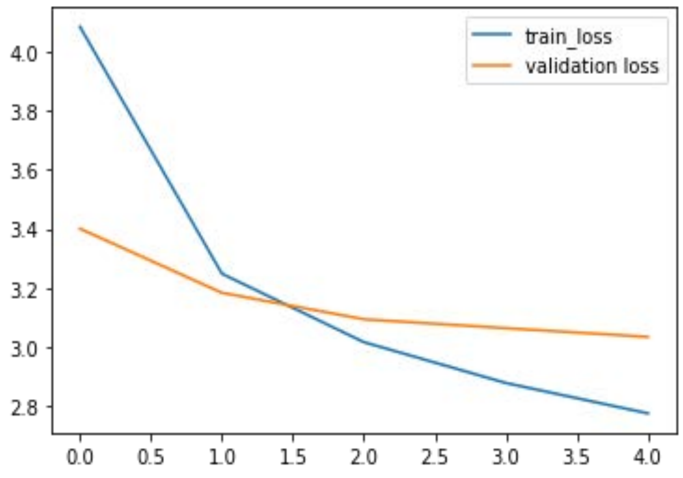
Setting up the InceptionV3 CNN Layer



```
def CNN_Encoder():
    inception_v3 = tf.keras.applications.InceptionV3(
        include_top=False,
        weights='imagenet'
    )

    output = inception_v3.output
    output = tf.keras.layers.Reshape(
        (-1, output.shape[-1]))(output)

    cnn_model = tf.keras.models.Model(inception_v3.input, output)
    return cnn_model
```

C. Experimental Results

The training time of the model took about one and a half hour for each epoch, since our machine can't handle the training process really well, we decided to train the model for 5 epoch so a total of about seven and a half hour to train the whole model, as a result we got around 43% of accuracy by only training this model for 5 epochs, for each epochs the loss also gradually decreasing which means that the

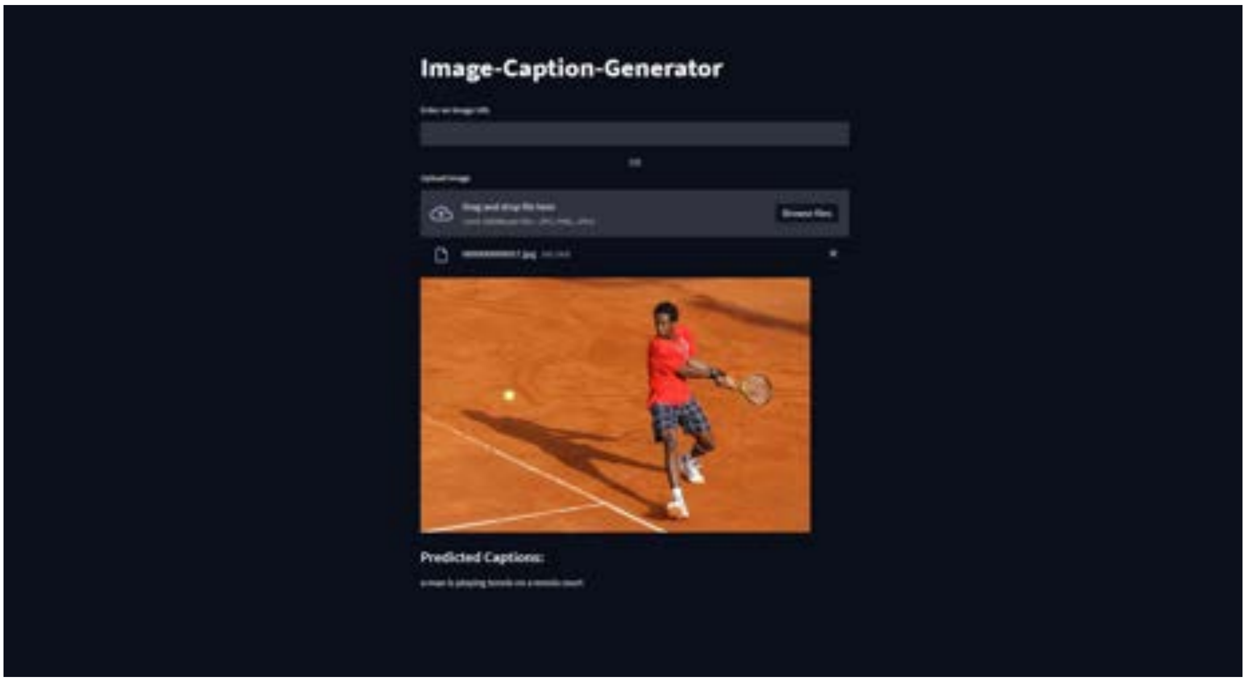


machine would predict a more accurate output overtime

At the end of our experiment, we deployed our model for testing purposes in public through streamlit and can be accessed through "Image-Caption-Generator" so that people can try our model and have fun with it when possible, the results might not be accurate at all time but for the limited and minimum testing time the model got, an accuracy of 43% is a moderately good accuracy especially since this is a generative model that will predict new captions for each image, and the output will be predicted word by word, even a minor change in the initial condition can affect the whole model and either boost or lower the accuracy.

V. CONCLUSION

In this paper, we propose a novel approach using the method of combining both InceptionV3 and Transformer encoder and decoder architecture. By using the pretrained model of InceptionV3 to process the image and then from the InceptionV3 output passing through as an input to the encoder and decoder to predict the generated captions output. The transformer encoder decoder architecture works better compared to conventional neural networks because the self-attention mechanisms, because of these captions inputted into the transformer doesn't need to be process word by word like the attention mechanism provided by conventional RNN but rather it processes the word as a whole sentence, hence time wise it will work faster because it will avoid sequential processing,



Predicted Captions:

a man is playing tennis on a tennis court

REFERENCES

[1] S. Degadwala, D. Vyas, H. Biswas, U. Chakraborty and S. Saha, "*Image Captioning Using Inception V3 Transfer Learning Model*," 2021 6th International Conference on Communication and Electronics Systems (ICCES), Coimbatre, India, 2021, pp. 1103-1108, doi: 10.1109/ICCES51350.2021.9489111.

[2] Vaswani, A. et al. (2017) *Attention is all you need*, arXiv.org. Available at: https://arxiv.org/abs/1706.03762 (Accessed: January 13, 2023).

[3] Image caption generator (no date) Clairvoyant. Available at: https://www.clairvoyant.ai/blog/image-caption-generator (Accessed: January 13, 2023).

[4] Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. "*Show, attend and tell: neural image caption generation with visual attention"*. In Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37 (ICML'15). JMLR.org, 2048–2057.

[5] S. -H. Han and H. -J. Choi, "Explainable Image Caption Generator Using Attention and Bayesian Inference," 2018 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 2018, pp. 478-481, doi: 10.1109/CSCI46756.2018.00098.