**Project Report: Image Captioning using CNN and Transformer**

**Objective**

To build and fine tune an image captioning system that automatically generates meaningful textual descriptions for input images using a CNN-based encoder and Transformer-based decoder architecture.

**Dataset Used:** Flickr8k

* Each image has five descriptive captions.
* Images and captions are processed to build an image-caption mapping.

**Preprocessing Pipeline**

**1. Caption Preprocessing**

* Converted to lowercase.
* Removed punctuation and extra whitespaces.
* Appended special tokens [start] and [end].

def preprocess(text):

...

return '[start] ' + text + ' [end]'

**2. Vectorization**

* Uses TextVectorization from TensorFlow to tokenize and convert captions into integer sequences.
* Limit vocabulary to 10,000 most common words.
* Pad/truncate captions to a fixed MAX\_LENGTH (e.g., 40).
* Built vocabulary using tokenizer.adapt.

**3. Lookup Layers**

* word2idx: Converts words to integer IDs.
* idx2word: Converts integer IDs back to words (used for generating text from predictions).

**4. Data Preparation**

* Create a dictionary mapping each image to its list of captions.
* Split image keys into training (80%) and validation (20%) sets.
* Flatten these mappings into parallel lists of image paths and their respective captions.
* Batched and shuffled using TensorFlow's pipeline.

### 5. Data Pipeline Creation

* tf.data.Dataset is used to pair images with captions.
* Each image is loaded and resized to 299x299 and normalized to [0,1] range.
* Dataset is shuffled and batched for training.

**Image Processing**

* Images resized to 299x299.
* Normalized to [0, 1] range.
* Used InceptionV3 from Keras as feature extractor (encoder).
* Extracted feature maps reshaped to (-1, 2048).

def CNN\_Encoder():

...

return cnn\_model

**Data Augmentation**

* Random horizontal flip.
* Random rotation (0.2 radians).
* Random contrast adjustment (0.3).

image\_augmentation = tf.keras.Sequential([

tf.keras.layers.RandomFlip("horizontal"),

tf.keras.layers.RandomRotation(0.2),

tf.keras.layers.RandomContrast(0.3),

])

Helps improve model generalization by diversifying training data.

**Model Architecture**

**1. Encoder**

* Uses pre-trained InceptionV3 (excluding top classification layer).
* Output feature map is reshaped to 2D tensor (num\_patches, 2048).
* Fully based on a Transformer encoder block.
* Applies dense layer followed by self-attention and residual connection.
* **Purpose:** Extract rich features from input images

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

...

**2. Decoder**

* Applies self-attention over the image feature patches.
* Includes LayerNorm, Dense transformation, and multi-head attention.
* Uses two attention blocks:
  + First: self-attention for generated captions.
  + Second: cross-attention to attend image features.
* Followed by feed-forward layers and output projection.

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

...

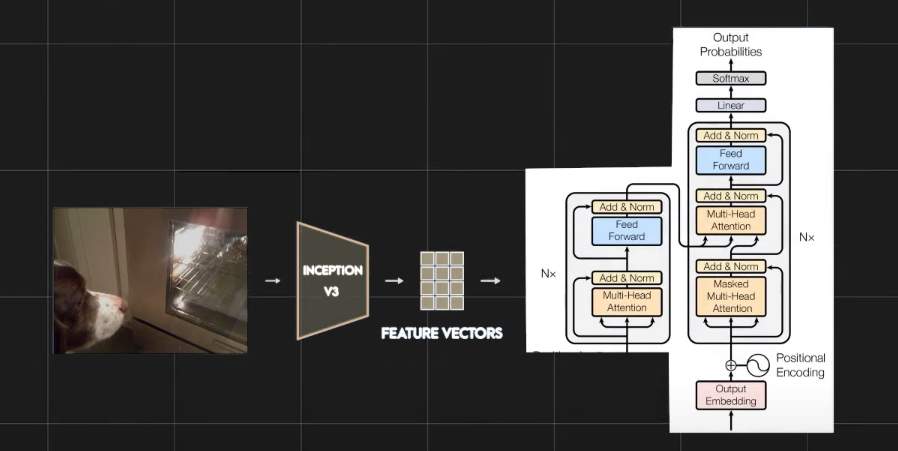
**3. Embeddings Layer**

* Converts token IDs to dense vectors (token embeddings).
* Adds positional embeddings to encode token positions.
* Combines token embeddings and positional embeddings.
* Essential for Transformer to retain word order.

### 4. Transformer Decoder

* Two attention layers:
  + **Self-attention:** For previously generated words.
  + **Cross-attention:** Attends to image features.
* Followed by feed-forward neural layers.
* Output projected to vocabulary size using a Dense + Softmax layer.

## Model Class (ImageCaptioningModel)

* Combines CNN, encoder, and decoder.
* Handles:
  + Training: computes loss and gradients.
  + Testing: evaluates on validation data.
  + Metrics tracking: accuracy and loss.

### Training Logic:

1. Extract image features via CNN.
2. Encode features with Transformer encoder.
3. Prepare y\_input (caption tokens excluding last) and y\_true (caption tokens excluding first).
4. Decode using Transformer decoder.
5. Compute masked loss and accuracy.

**Training Setup**

* **Loss Function:** SparseCategoricalCrossentropy with masking.
* **Optimizer:** Adam
* **Callback:** EarlyStopping with patience of 3 epochs.

loss = tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=False, reduction="none")

optimizer = tf.keras.optimizers.Adam()

**Evaluation**

**Metrics Used:**

* **Loss:** Measures token-level prediction error.
* **Accuracy:** Word-level accuracy considering padding mask.

**Visualization:**

* Loss and accuracy plotted over training epochs for model monitoring.
* Sample predictions evaluated qualitatively.

**Caption Generation**

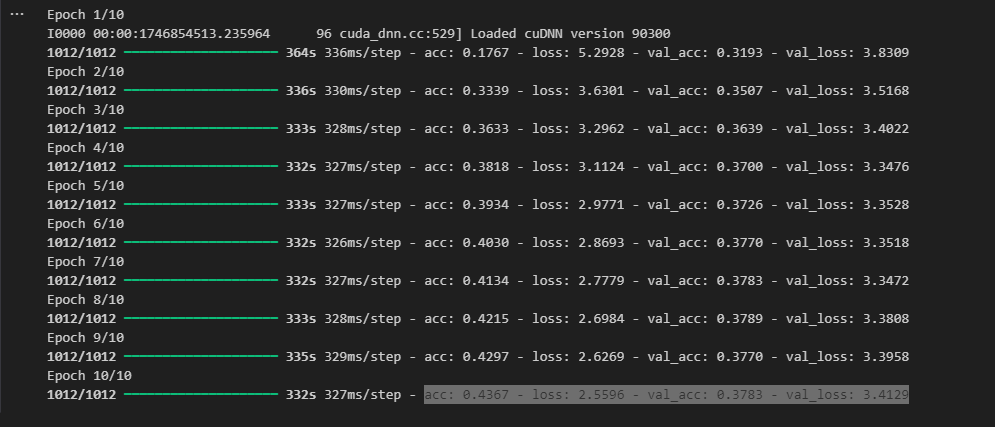
1. Load and preprocess input image.
2. Extract features via CNN and encode.
3. Start with [start] token.
4. Iteratively predict next word using greedy decoding (argmax).
5. Stop if [end] token is predicted or max length reached.
6. Return joined words as final caption.

def generate\_caption(img\_path):

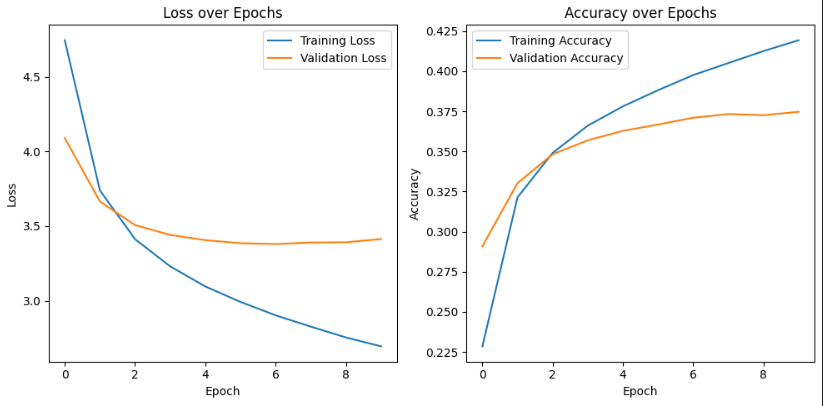
...

return y\_inp.replace('[start] ', '')

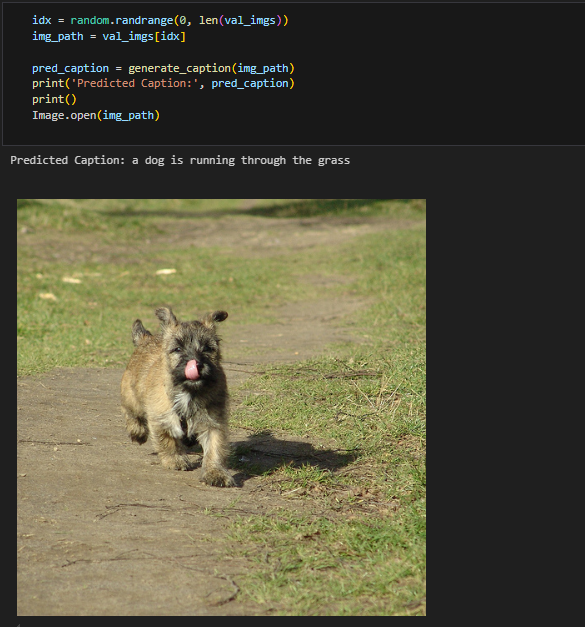
**Model Performance:** -

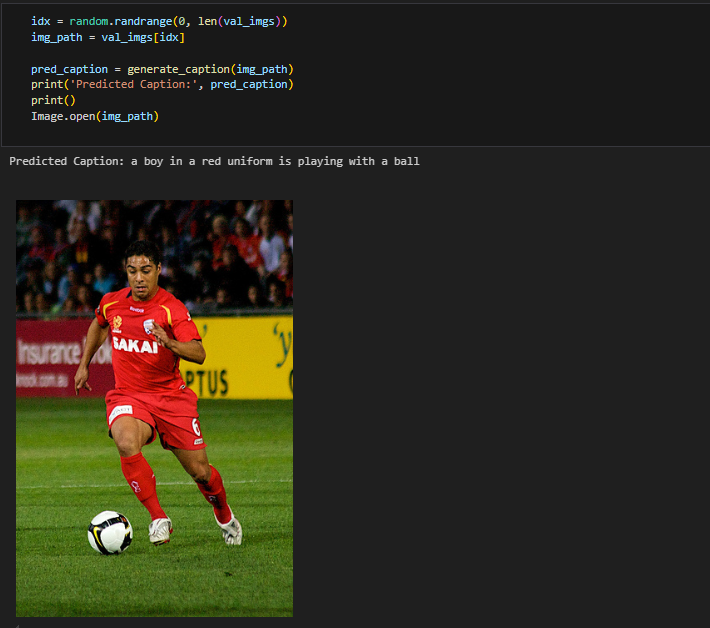
* Final Accuracy: 0.4172
* Final Loss: 2.7210
* Final Validation Accuracy: 0.3747
* Final Validation Loss : 3.4137
* Consistently generates fluent and relevant captions
* It can more if we run this on more epochs.

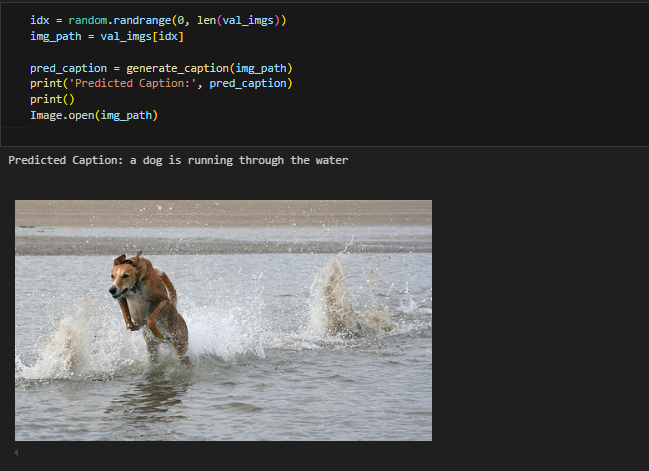
**Graphs:**

****

**Results:**

****

****

****

**Model Infrastructure**

* **Encoder:** CNN (InceptionV3) + Transformer Encoder
* **Decoder:** Transformer Decoder
* **Training Framework:** TensorFlow 2.x + Keras Subclassing API
* **Hardware:** Kaggle GPU / Colab GPU (recommended)

**Conclusion**

This project demonstrates a powerful hybrid model that merges convolutional feature extraction and sequence modeling using Transformers for automatic image description. It performs competitively with strong qualitative results and can be further improved using:

* Beam Search
* Attention Visualization
* Fine-tuning CNN layers
* Larger datasets (e.g., MSCOCO)

**Group Members:**

22K-8729 **Sajad Ali**  22K-4084 **Umar Orakzai**

22k-8727 **Hasan Abdul Rehman** 22K-4044 **Muhammad Umar**