Week 2 :

Resources

- **Code and Tutorials:**
    - **Pre-trained CNN Models:**
        - **PyTorch Models:** https://pytorch.org/vision/stable/models.html
        - **TensorFlow Models:** https://www.tensorflow.org/api_docs/python/tf/keras/applications
    - **Transformer Tutorials:**
        - **Hugging Face Transformers:** https://huggingface.co/transformers/

Tasks:

1. Data Preparation

- **Identify and Download a Dataset:** download dataset given in link below
- **Preprocessing Captions:**
    - Tokenize captions using NLP libraries (e.g., NLTK, SpaCy, Hugging Face).
    - Build a vocabulary and encode captions into sequences.
    - Add padding/truncation for captions to ensure uniform length.
- **Preprocessing Images:**
    - Use a pre-trained CNN (e.g., ResNet, EfficientNet) to extract visual features from images.
    - Normalize and resize images to a fixed size (e.g., 224x224 for most CNNs).
    - Store extracted features for efficient training.

2. Model Prototyping

- **CNN Feature Extractor:**
    - Implement a CNN (like ResNet-50) to process images and extract visual embeddings.
- **Text Encoder (Transformer or LSTM):**
    - Use an encoder to process tokenized captions, experimenting with LSTMs or transformer-based models (e.g., Hugging Face's BERT).
- **Combine Image and Text Features:**
    - Add a mechanism to concatenate or fuse image embeddings and encoded captions for the decoder to process.
- **Initial Decoder Setup:**
    - Use a transformer decoder or an attention mechanism to generate the captions sequentially.
    - Begin by building a minimal model pipeline and confirm it can process dummy data (input image + captions) without errors.

Assignment dataset : https://www.kaggle.com/datasets/adityajn105/flickr8k

Expected Outcomes:

- Preprocessed dataset with tokenized captions and extracted image features.

- A working model prototype combining CNNs and transformers for caption generation.

HINTS

Preprocessing Captions:

Tokenization: Use a tokenizer to split sentences into words:

```
from nltk.tokenize import word_tokenize

captions = ["A man riding a horse.", "A dog playing in the yard."]

tokenized_captions = [word_tokenize(caption.lower()) for caption in captions
```

- **Build Vocabulary:** Assign unique indices to words. Include special tokens like `<start>`, `<end>`, and `<pad>`.

Encode Captions: Convert tokens into sequences of integers:

```
word_to_index = {"<start>": 1, "a": 2, "man": 3, ...}  # Example vocabulary

encoded_caption = [word_to_index[word] for word in tokenized_caption]
```

-

Pad Captions: Use libraries like TensorFlow/Keras for padding:

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

padded_captions = pad_sequences(encoded_captions, maxlen=20, padding="post")
```

-

Preprocessing Images:

Use pre-trained CNN models to extract features:

```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array, load_img


# Load and preprocess image
image = load_img('image.jpg', target_size=(224, 224))
image = preprocess_input(img_to_array(image))


# Extract features
model = ResNet50(weights='imagenet', include_top=False, pooling='avg')
features = model.predict(image.reshape(1, 224, 224, 3))
```

-

Save the extracted features for faster processing during training:

```
import pickle
with open('features.pkl', 'wb') as f:
    pickle.dump(features, f)
```

-

---

2. Model Prototyping

CNN Feature Extractor:

- Use a CNN to encode image features. You can experiment with pre-trained networks like ResNet, EfficientNet, or MobileNet.
- The output of the CNN should be a fixed-length feature vector for each image.

Text Encoder:

Use a transformer-based encoder or LSTM for captions:

```
from tensorflow.keras.layers import Embedding, LSTM

vocab_size = 5000  # Adjust based on your dataset

embedding_dim = 256

max_length = 20



text_input = Input(shape=(max_length,))

embedded_text = Embedding(vocab_size, embedding_dim)(text_input)

encoded_text = LSTM(512, return_sequences=True)(embedded_text)
```

-

Combine Image and Text Features:

Concatenate image features with encoded text:

```
from tensorflow.keras.layers import Concatenate



combined_features = Concatenate()([features, encoded_text])
```

-

Initial Decoder Setup:

- Use a transformer decoder or attention mechanism:
    - Implement cross-attention to align image features with text.

Keras has built-in layers for attention in TensorFlow 2.6+:

```
from tensorflow.keras.layers import Attention
```

```
attention = Attention()([combined_features, encoded_text])
```

○