Multimodal Fusion of Text and Image Data using Cross-Attention Mechanism

Presenting by:

A. Chaitanya Sai
S. Vamsi Krishna
G. Sai Prabhath
K.N.L. Rekha

Overview

- Problem Statement
- Proposed Method
- Expermentation
- Improved Methodology
- Results
- Future Work

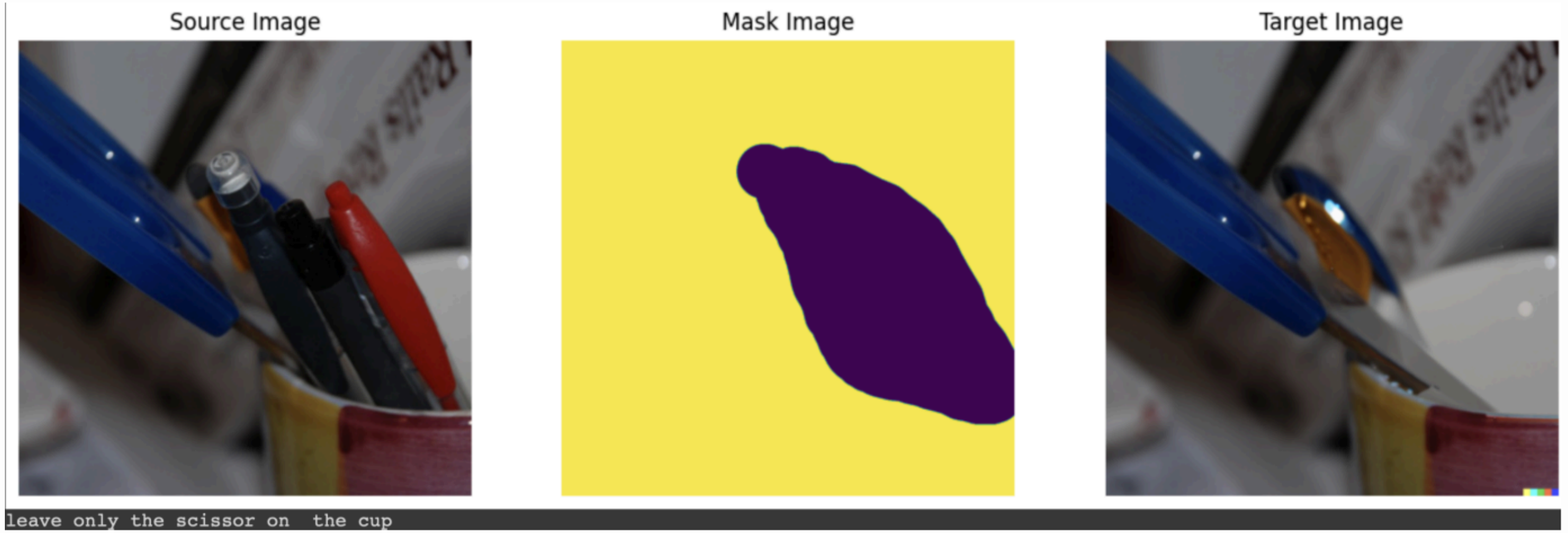
PROBLEM STATEMENT

Integrating image and text data in multimodal machine learning is challenging due to differences in data structure and encoding. Methods like concatenation and CLIP often lose positional information from images, which is crucial for accurate alignment.

While cross-attention mechanisms show promise by preserving positional and semantic details, challenges like data inequality and modality interaction persist. Addressing these issues is essential for achieving seamless and human-like multimodal data fusion in AI systems.

Dataset

We have used MagicBrush dataset provided by HuggingFace API



train data size : 8,807

test data size: 528

*because of time constraints we have used only 50% of train data in all out experiments

Code

- We have written the code from scratch
- We have used pytorch library for training

Model Architecture

Image Encoders:

1. CLIP - gives embedding of dimension 512
2. Resnet50 - gives embedding of dimension 2048

Text Encoders:

1. CLIP - gives embedding of dimension 512
2. Bert - gives embedding of dimension 768

- To match the dimensions of Resnet50 to Bert we have added an additional learnable adapter block which is a linear layer to map the dimension from 2048 to 768

Model Architecture

Mask Prediction Model:

- When input is a 768 x 768 or 512x512 dim vector

```
self.mask_prediction_network = nn.Sequential(
    nn.Conv2d(1, 16, kernel_size=3, stride=1, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Conv2d(16, 32, kernel_size=3, stride=1, padding=1),
    nn.ReLU(),
    nn.AdaptiveAvgPool2d((224, 224)),
    nn.Conv2d(32, 1, kernel_size=3, stride=1, padding=1),
    nn.Sigmoid()
)
```

Model Architecture

Mask Prediction Model:

- When input is a 768 dim vector

```
self.mask_prediction_network =  nn.Sequential(
    nn.ConvTranspose2d(768, 512, kernel_size=3, stride=2, padding=0),  # [32, 512, 3, 3]
    nn.ReLU(inplace=True),
    nn.ConvTranspose2d(512, 256, kernel_size=5, stride=2, padding=1),  # [32, 256, 7, 7]
    nn.ReLU(inplace=True),
    nn.ConvTranspose2d(256, 128, kernel_size=5, stride=2, padding=2),  # [32, 128, 14, 14]
    nn.ReLU(inplace=True),
    nn.ConvTranspose2d(128, 64, kernel_size=5, stride=2, padding=1),   # [32, 64, 28, 28]
    nn.ReLU(inplace=True),
    nn.ConvTranspose2d(64, 32, kernel_size=5, stride=2, padding=1),    # [32, 32, 56, 56]
    nn.ReLU(inplace=True),
    nn.ConvTranspose2d(32, 16, kernel_size=5, stride=2, padding=0),    # [32, 16, 112, 112]
    nn.ReLU(inplace=True),
    nn.ConvTranspose2d(16, 1, kernel_size=4, stride=2, padding=2),     # [32, 1, 224, 224]
    nn.Sigmoid()  # Assuming output needs to be normalized to [0, 1]
)
```

Base Lines

Concatenation

- **Process: Combines image and text encoded features into a single vector.**
- **Advantages:**
  - **Simple and computationally lightweight.**
  - **Effective for general relationships.**
- **Limitations:**
  - **Loses spatial information.**
  - **Limited feature interaction.**

E1 - ImageEncoder ∈ R^d
E2 - TextEncoder ∈ R^d
Resultant ∈ R^(2d)

Base Lines

Concatenation

Concatenation	show the cows tongue	
	Make the man smile.	
	Put some fries on the plate.	

Base Lines

Cross Product:

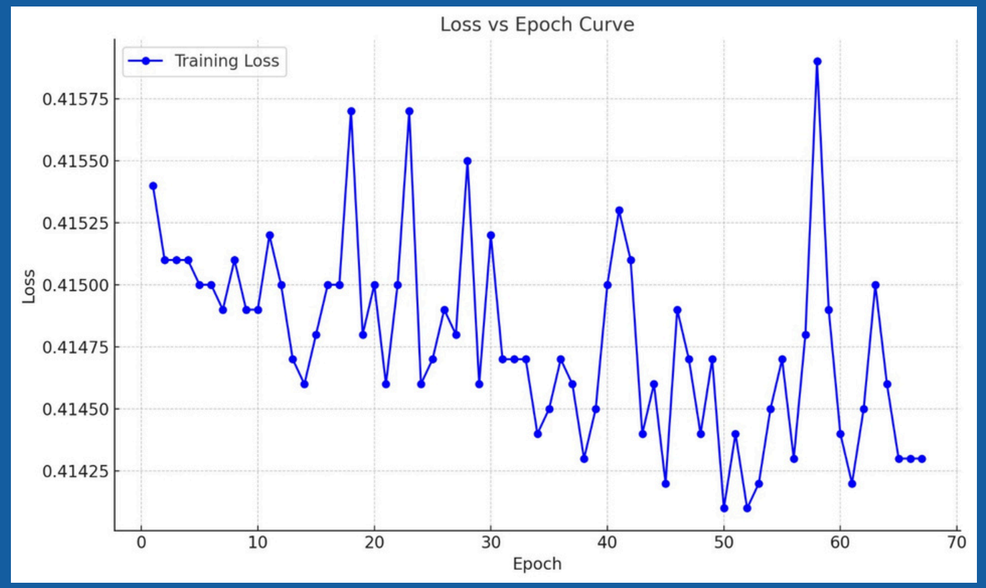
E1 - ImageEncoder $\in R^d$
E2 - TextEncoder $\in R^d$
Resultant $\in R^{(dxd)}$

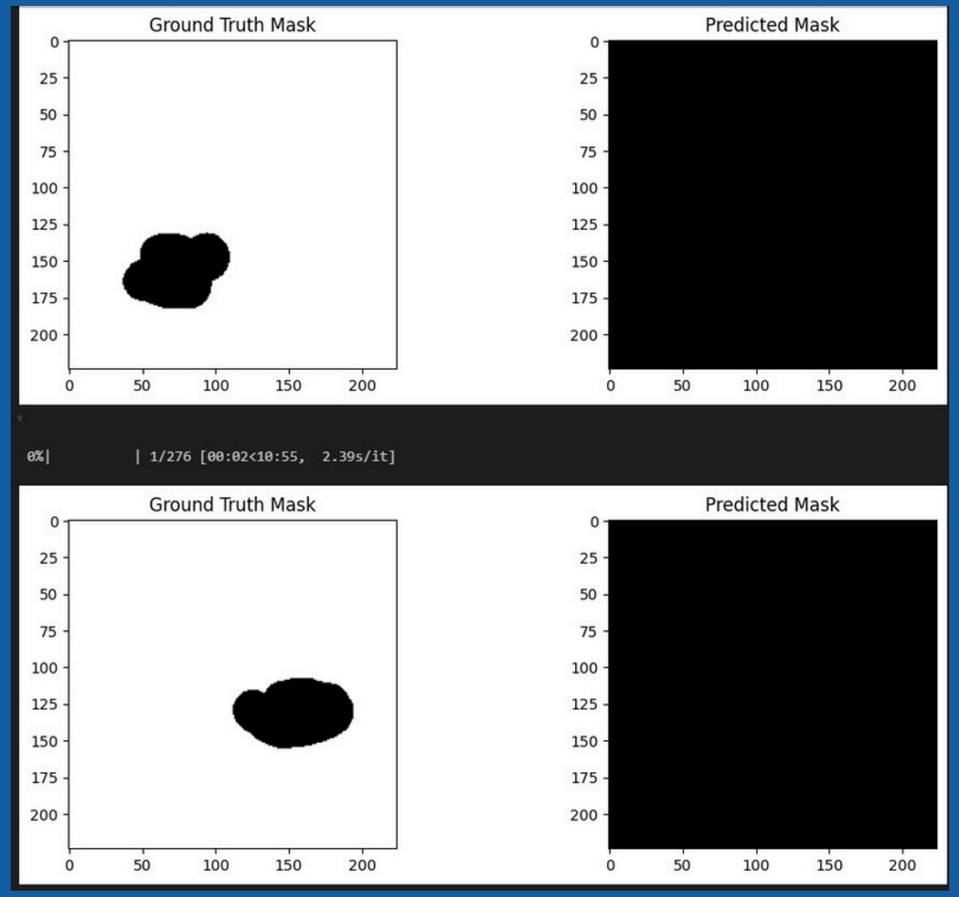
Experiment using CLIP:

- **Process: Aligns image and text features into a shared embedding space.**
- **Advantages:**
  - **Great for retrieval tasks (image-to-text, text-to-image).**
  - **Learns semantic relationships effectively.**
- **Limitations:**
  - **Lacks positional data (relies on pooled features).**
  - **Not ideal for spatially detailed tasks.**

RESULTS

CLIP used as image and text encoder:

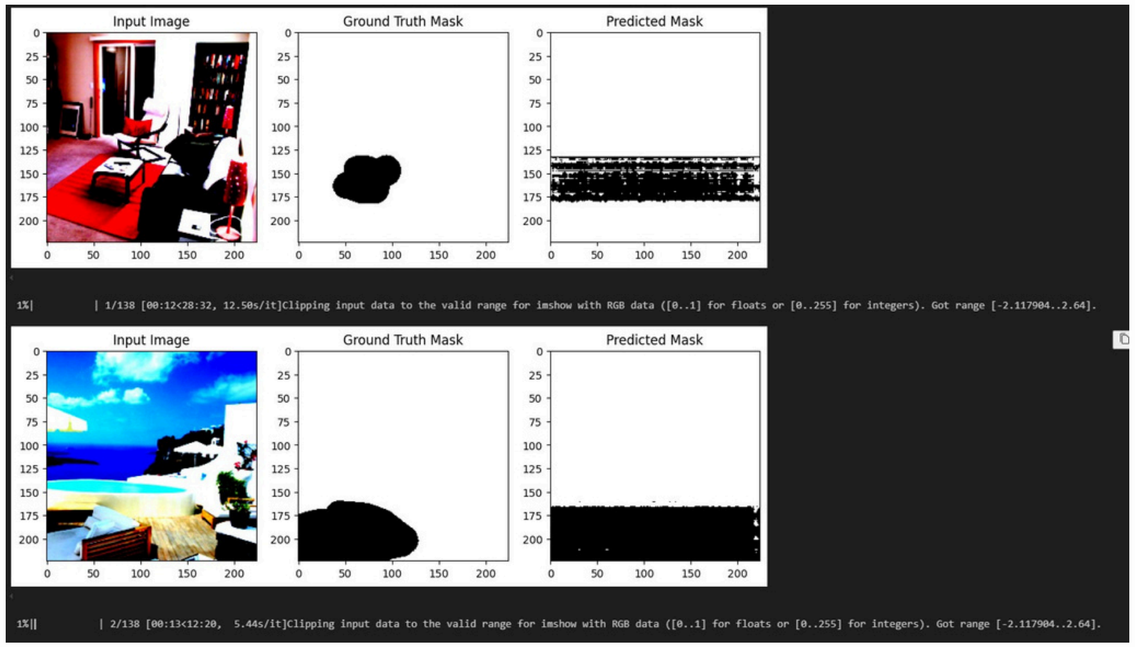


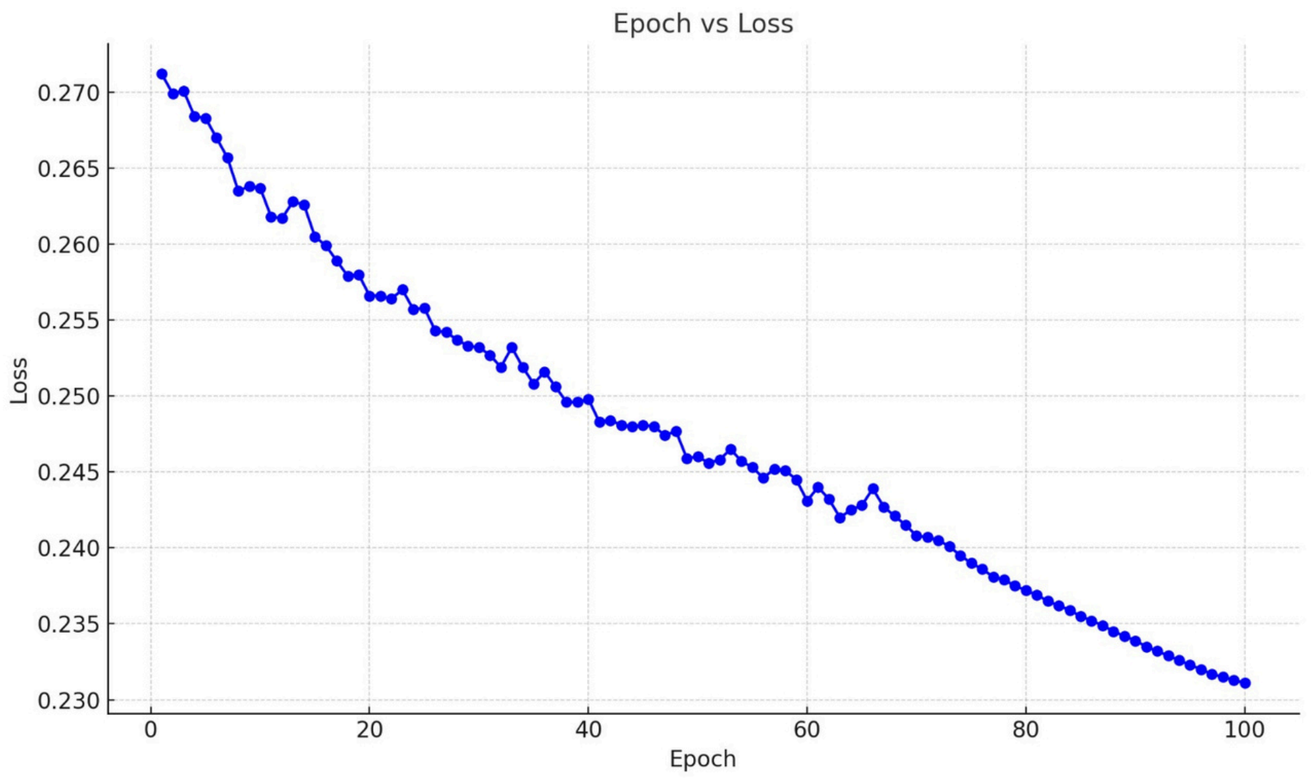


- IOU was nearly zero

The results of cross-product approach:

- here resnet50 is used as image encoder and bert as text encoder





Average IoU: 0.1991

Base Lines

Cross-Attention

- **Process: Text queries attend to image pixels via attention mechanisms.**
- **Advantages:**
  - **Preserves spatial and semantic data.**
  - **Ideal for detailed alignment tasks (e.g., image annotation).**
- **Limitations:**
  - **Computationally intensive.**
  - **Best for tasks requiring high precision.**

ARCHITECTURE

RESNET ENCODER

Linear Layer

self attention

leave only the scissor on the cup

BERT ENCODER

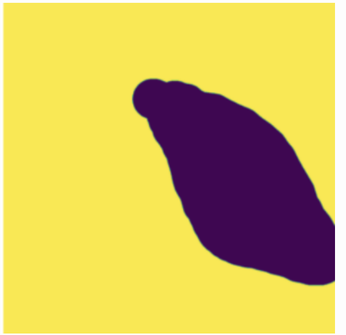
self attention

V

K

Q

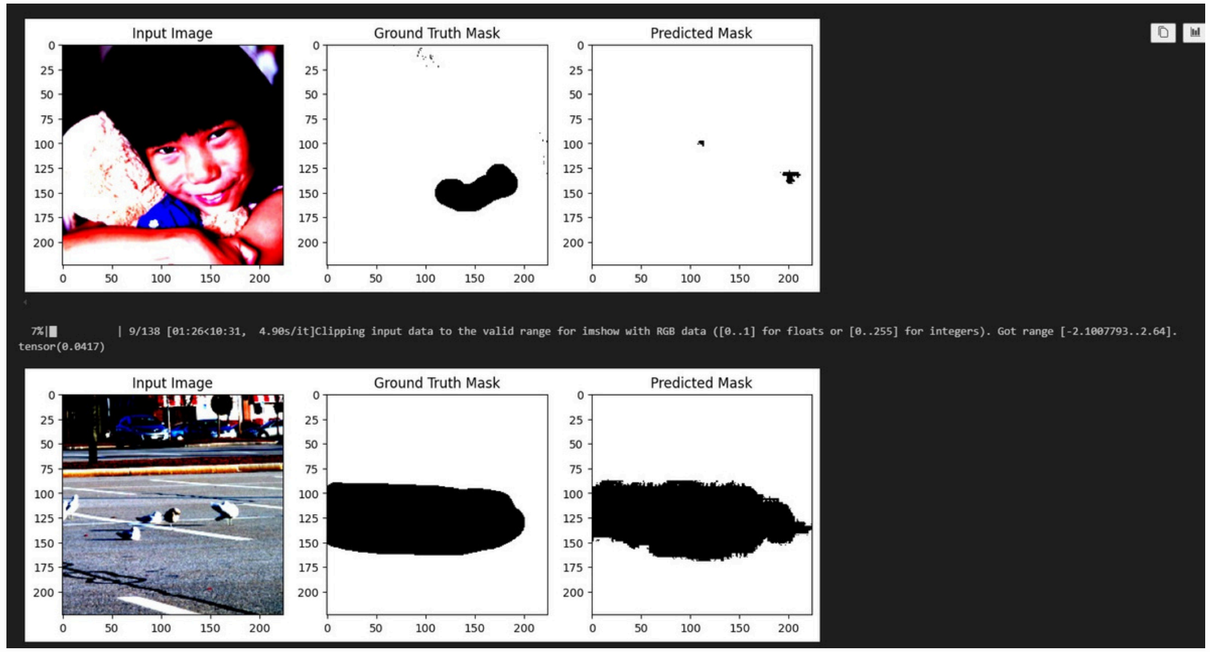
conv layers



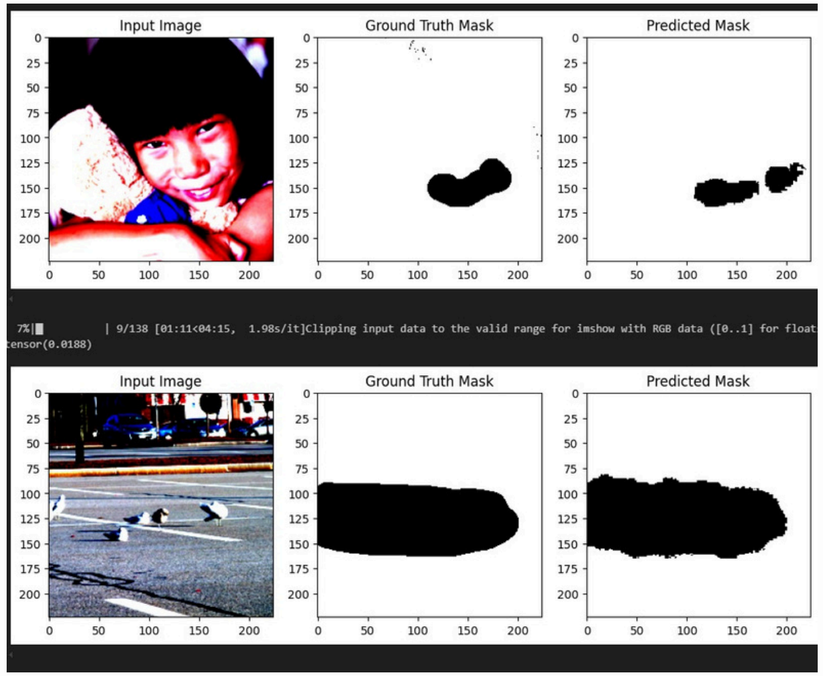
cross attention multihead

$$\mathbf{A} = softmax\left(\frac{\mathbf{Q_{text}}{\mathbf{K_{image}}}^{T}}{\sqrt{d_k}}\right) \in \mathbb{R}^{n \times n}$$

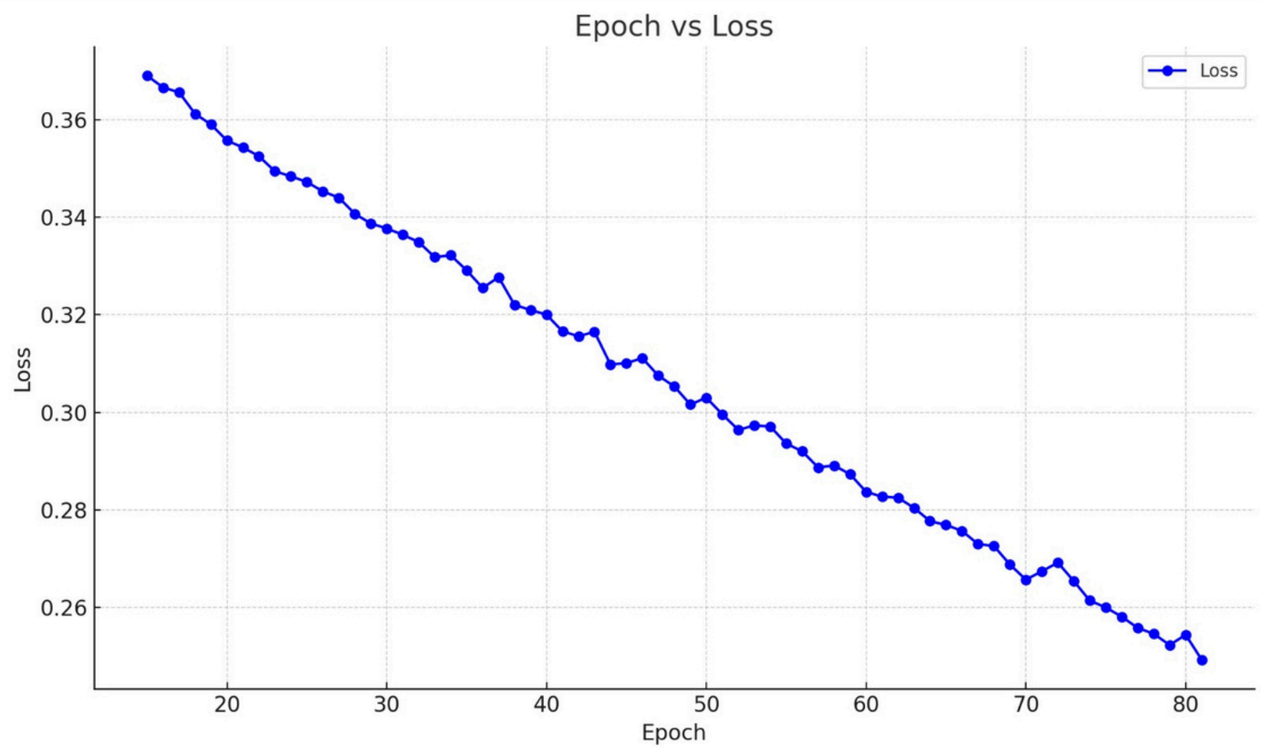
$$\mathbf{O_{C-Attention}} = \mathbf{A}\mathbf{V_{image}} \in \mathbb{R}^{n \times d_V}$$

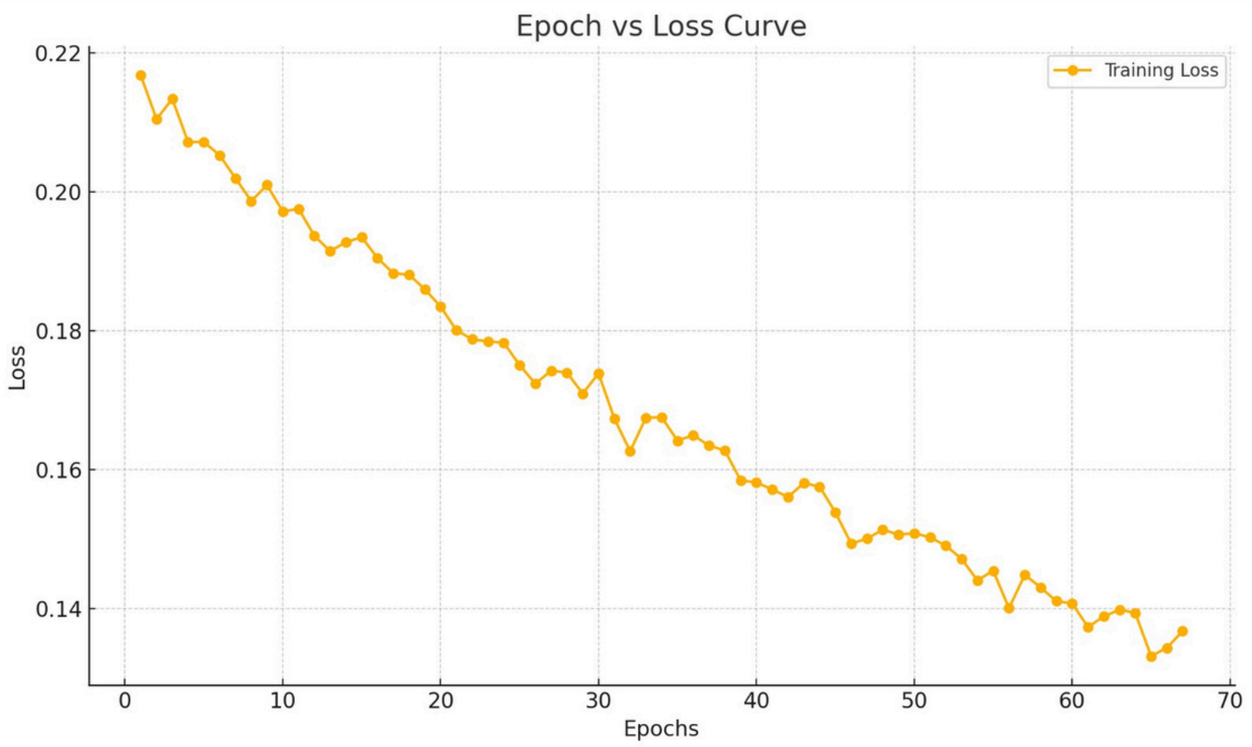


BCE with 100 epochs



BCE with 200 epochs

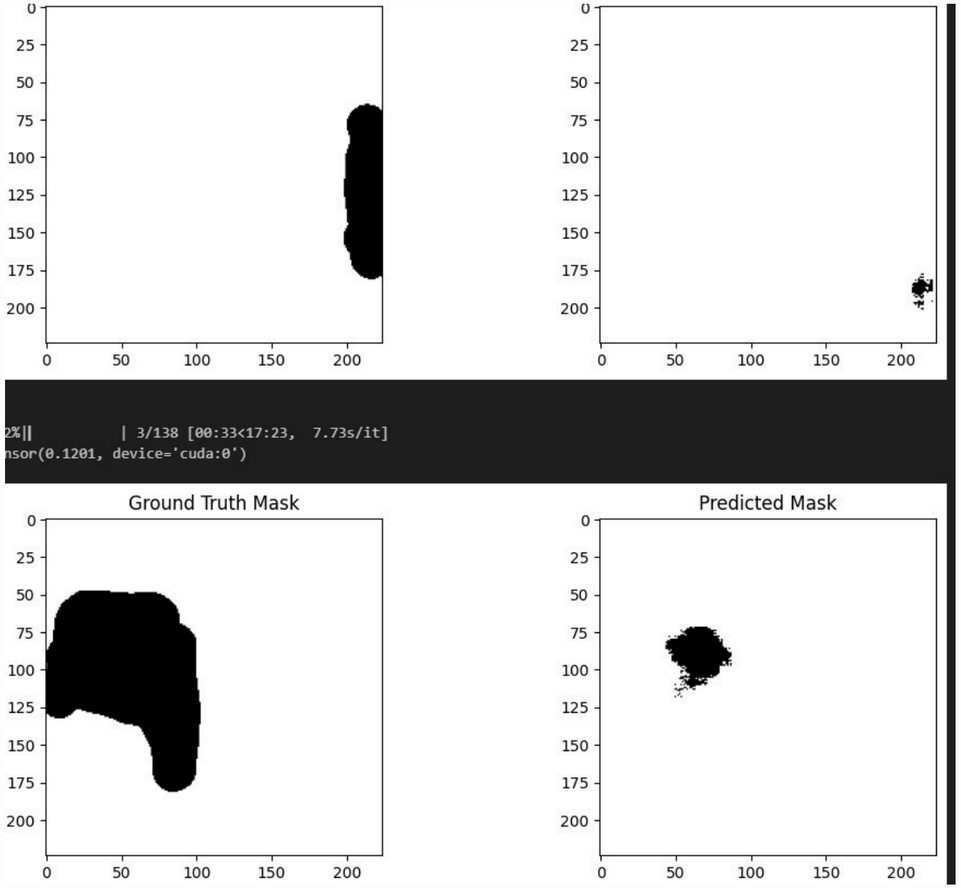




CROSS ATTENTION + BCE with 100 epochs
results in Average IoU: 0.9166

CROSS ATTENTION + BCE with 200 epochs
results in Average IoU: 0.9617

The results with Multi-Head Cross Attention using BCE+IOU(Jaccard distance) Loss :

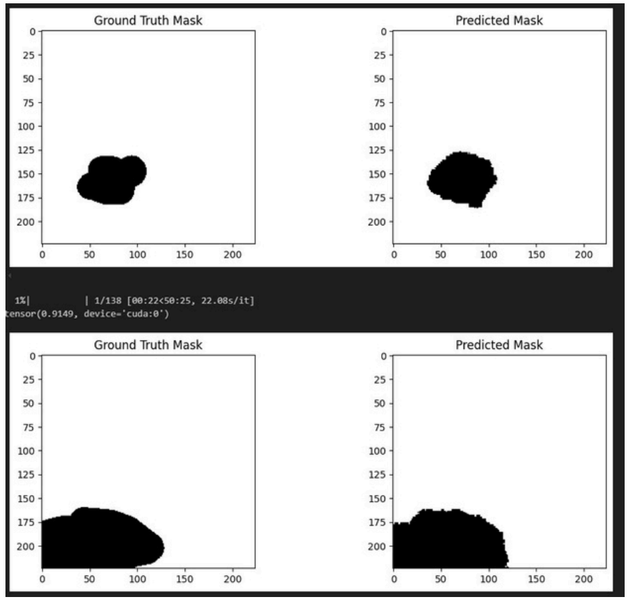


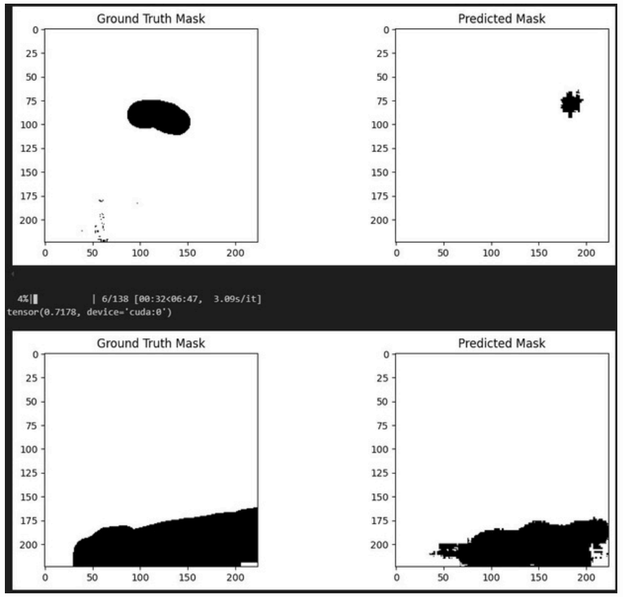
$$IOU = \frac{\text{Area of overlap}}{\text{Area of union}} = \frac{|T \cap P|}{|T \cup P|}$$

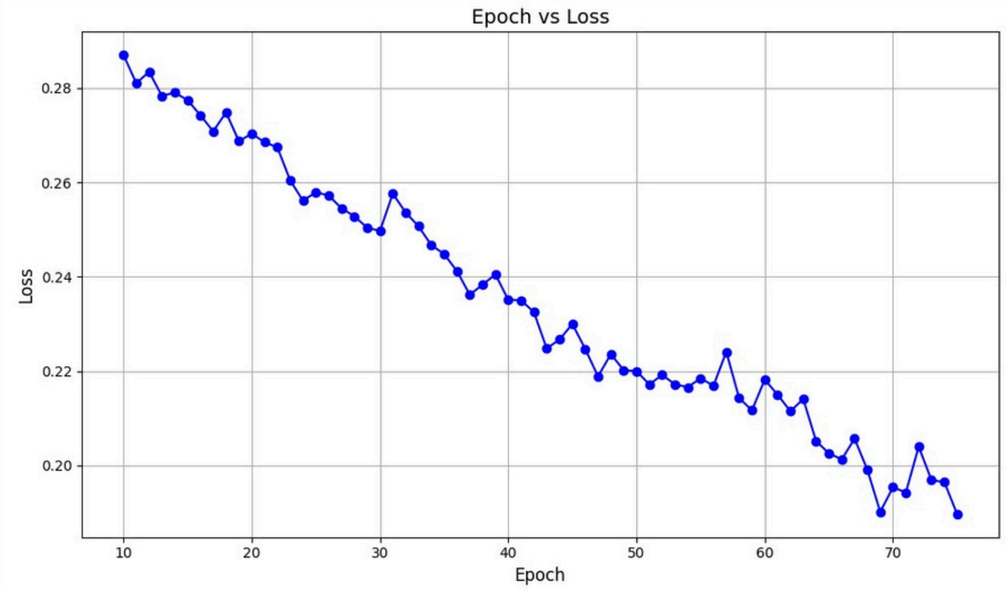
T → True
P → Prediction

BCE+IOU with a result of Average IoU: 0.2604

The results with Cross Attention with BCE+DICE Loss:





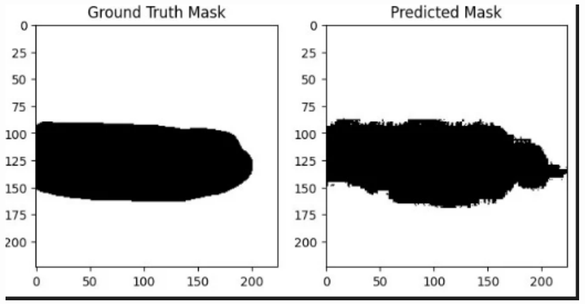


DICE + BCE with a result of Average IoU: 0.6186

$$DC = \frac{2TP}{2TP + FP + FN},$$

FUTURE WORK

- THE BACKBONE USED FOR GENERATING IMAGING EMBEDINGS IS RESNET 50
- WE CAN ALSO TRY WITH BETTER MODELS TO GENERATE IMAGE EMBEDINGS.
- OUR BEST RESULTS SUFFER WITH IRREGULAR BOUNDARIES WHICH MAY BE CRUCIAL FOR SOME SPECIFIC OFFLOADING TASKS , SO WE WOULD LIKE TO TRY BCE+CONTOUR LOSS WHICH FORCES MATCHING OF BOUNDARIES BY EDGE ENFORCEMENT.
- FOR GENERATING MASK WE USED FROM CROSS ATTENTION OUTPUTS WE USED CONVO TRANSPOSE 2D LAYERS FOR UPSCALING WE MAY GET BETTER GENERATION BY USING BETTER UPSCALING NETWORKS LIKE UNETS WITH ATTENTION.



THANK YOU!