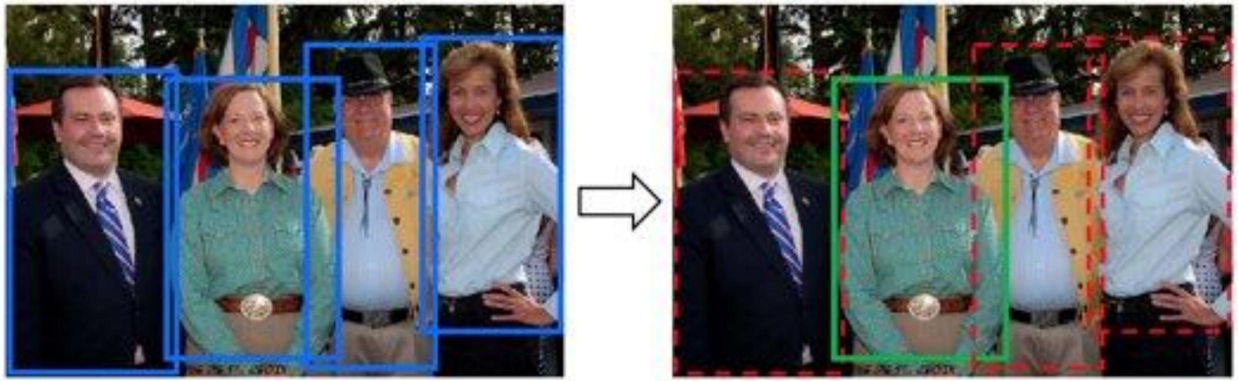
A Study of Referring Expression Comprehension

Group 37: Kevin Tak Hay Cheung, Nithin Pingili, Jerry Xu

Overview of REC

- Given an image and a text caption or captions, we aim to return a bounding box surrounding the target described by the caption(s).
- Also known as Object Grounding.
- REC is an important component of other vision-language tasks such as Visual Question Answering and Visual Dialogue.
- Challenging because it combines CV with NLP.

Expression: a lady standing next to a man wearing a blue suit and tie



Whole image and region proposals

Chosen region in green

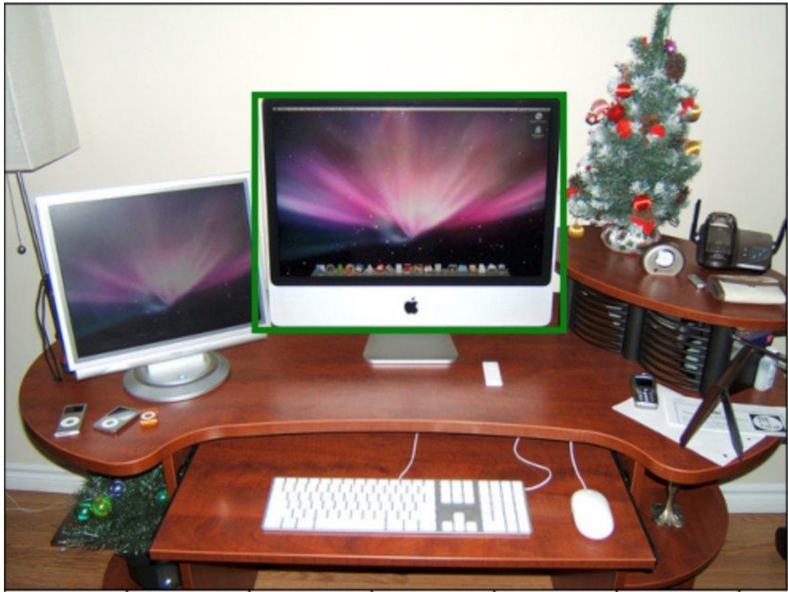
Dataset

- The most common dataset for REC is the RefCOCO family of datasets (RefCOCO, RefCOCO+, RefCOCOg).
- An image may have multiple bounding boxes, and each bounding box may have multiple expressions. Due to limited resources, we only kept one expression per bounding box.
- We ended up with ~180,000 image-text pairs.
- We only used a subset of 50,000 image-text pairs.

E1: biggest monitor

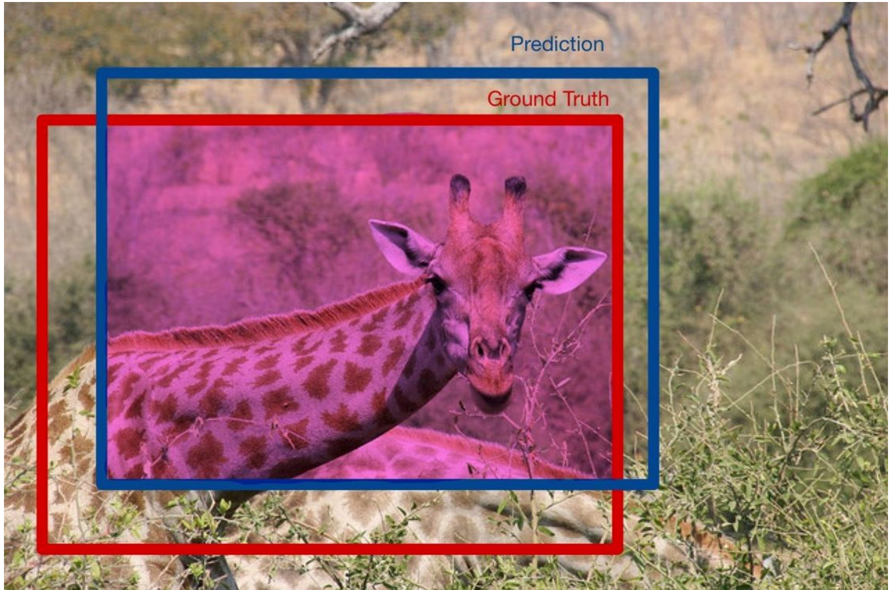
E2: front monitor

E3: the monitor dead center apple logo



Evaluation Metric (IoU)

- IoU defined as the area of overlap between the predicted bounding box and the ground truth bounding box divided by the area enclosed by the predicted bounding box and the ground truth bounding box.
- IoU >= 0.5 is usually OK, but in other applications (e.g. self-driving cars) we may need a higher threshold.



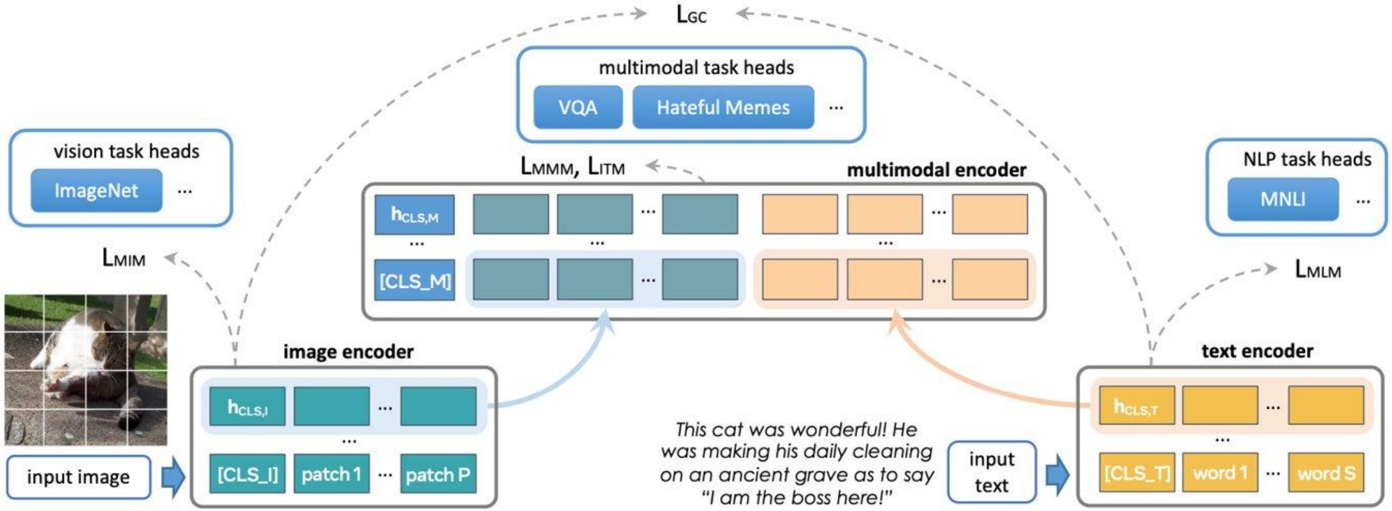
Method

We are assessing the performance of 3 models on the REC task.

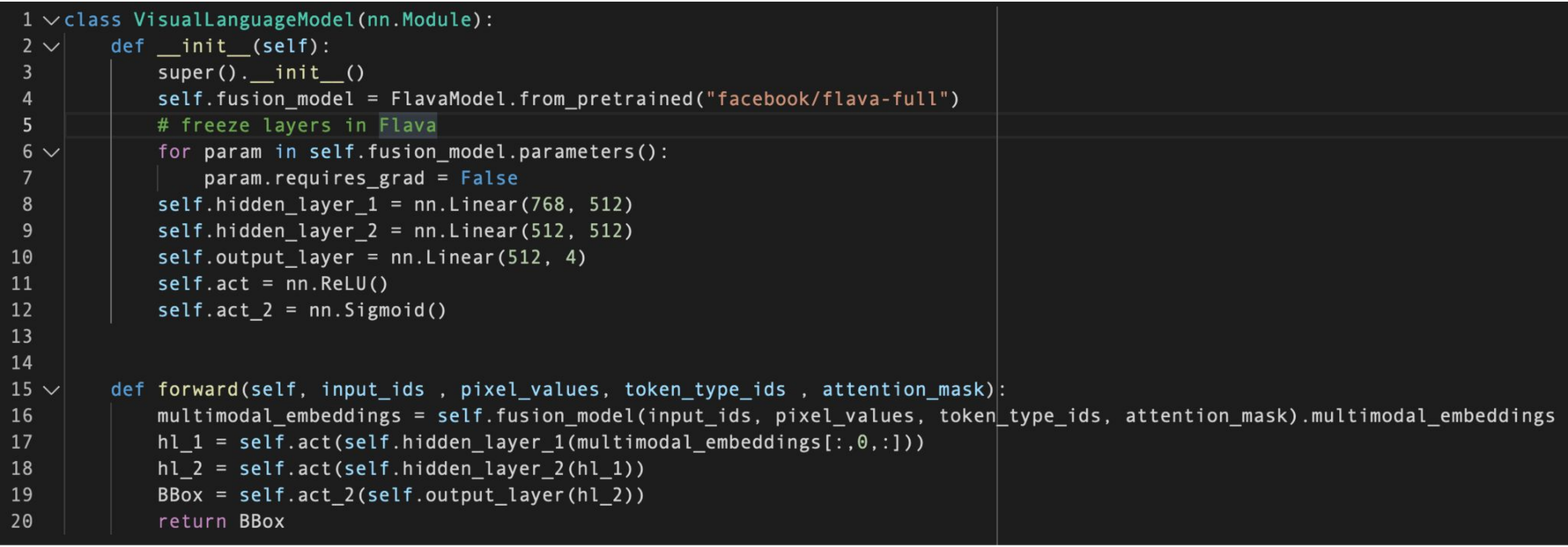
- FLAVA
- Encoder-Decoder Model
- Decoder-only Model

FLAVA (Foundational Language and Vision Alignment Model)

- Authors aimed to create a single universal model that can handle vision tasks, language tasks, and cross- and multi-modal vision and language tasks.
- FLAVA uses ViT to extract unimodal image representations, unimodal text representations, and fuse and align the image and text representations.
- FLAVA does not have an object detection head or a visual grounding module.
- Therefore we are training our own prediction head for the REC task.

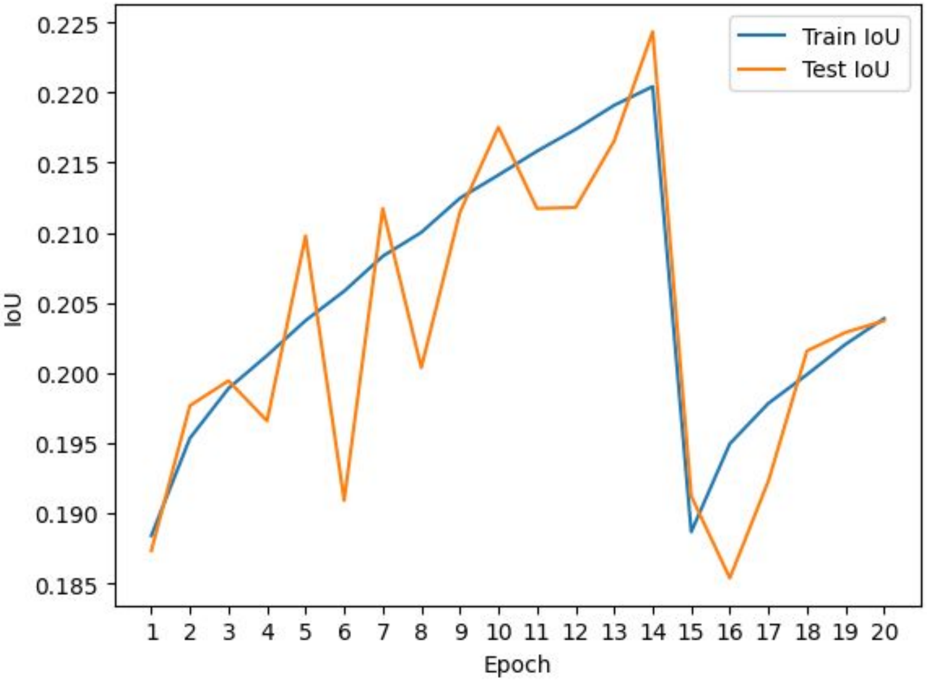


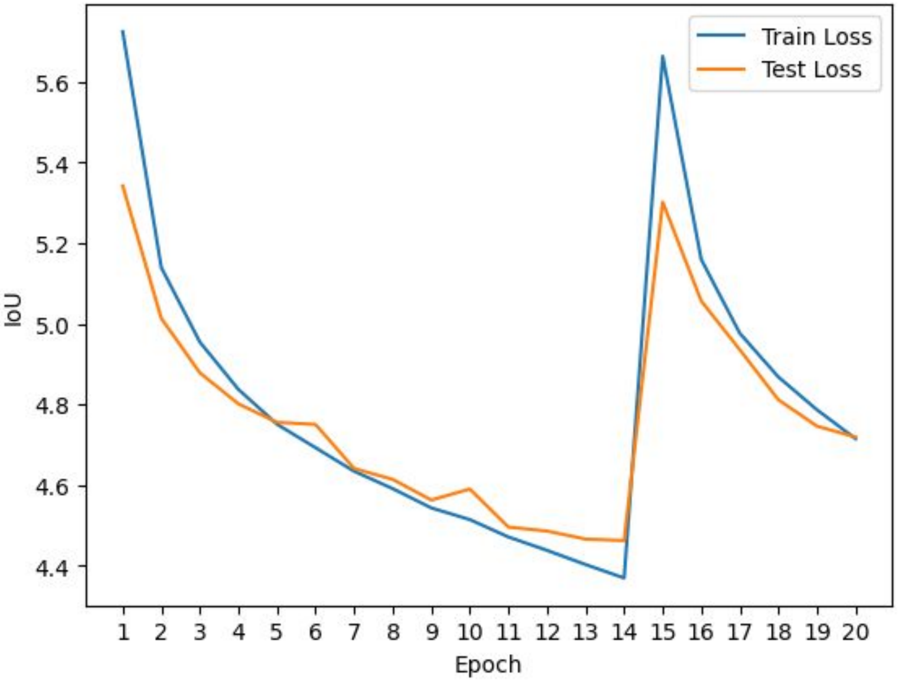
FLAVA



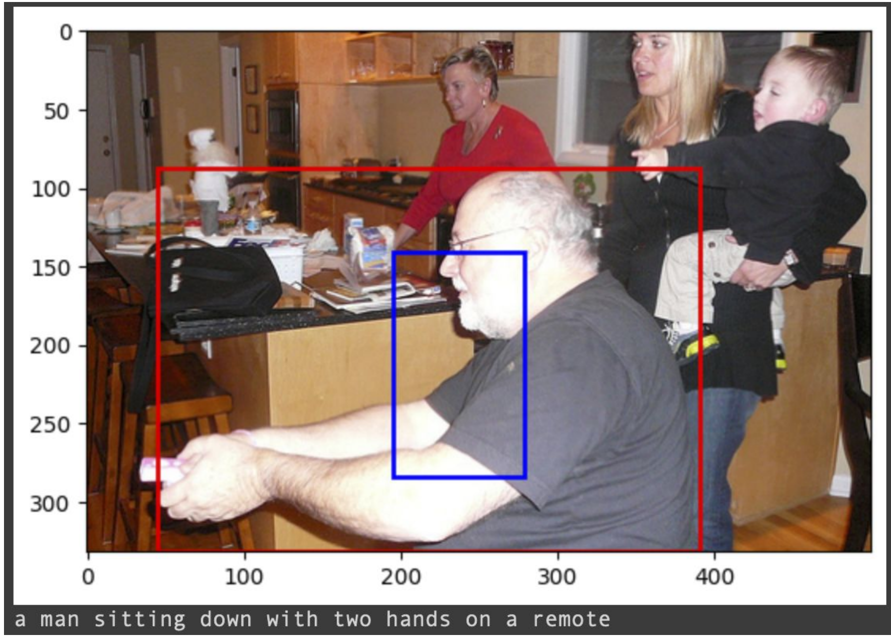
```python
class VisualLanguageModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.fusion_model = FlavaModel.from_pretrained("facebook/flava-full")
        # freeze layers in Flava
        for param in self.fusion_model.parameters():
            param.requires_grad = False
        self.hidden_layer_1 = nn.Linear(768, 512)
        self.hidden_layer_2 = nn.Linear(512, 512)
        self.output_layer = nn.Linear(512, 4)
        self.act = nn.ReLU()
        self.act_2 = nn.Sigmoid()


    def forward(self, input_ids , pixel_values, token_type_ids , attention_mask):
        multimodal_embeddings = self.fusion_model(input_ids, pixel_values, token_type_ids, attention_mask).multimodal_embeddings
        hl_1 = self.act(self.hidden_layer_1(multimodal_embeddings[:,0,:]))
        hl_2 = self.act(self.hidden_layer_2(hl_1))
        BBox = self.act_2(self.output_layer(hl_2))
        return BBox
```

FLAVA Results

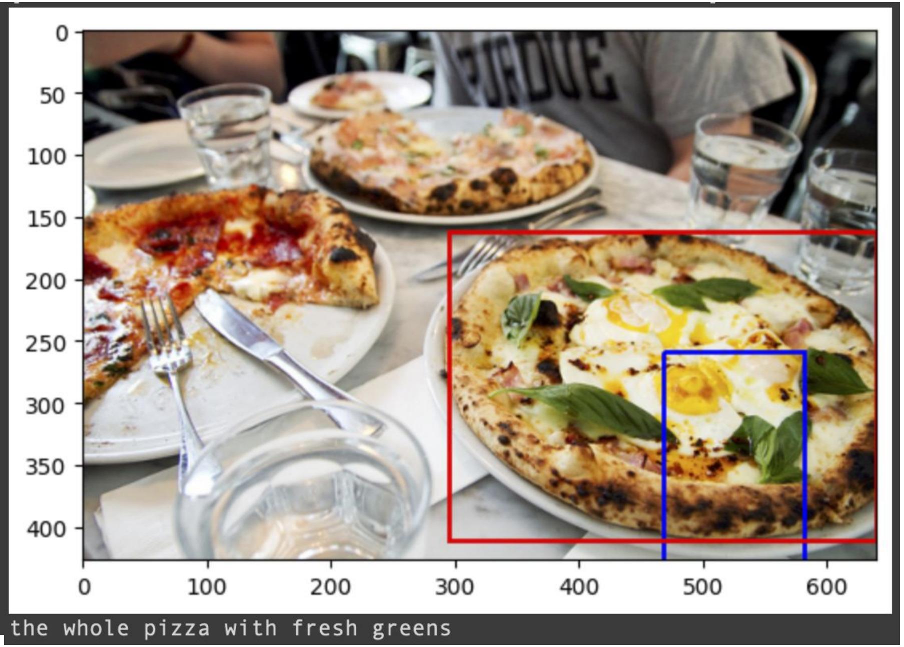




FLAVA Examples

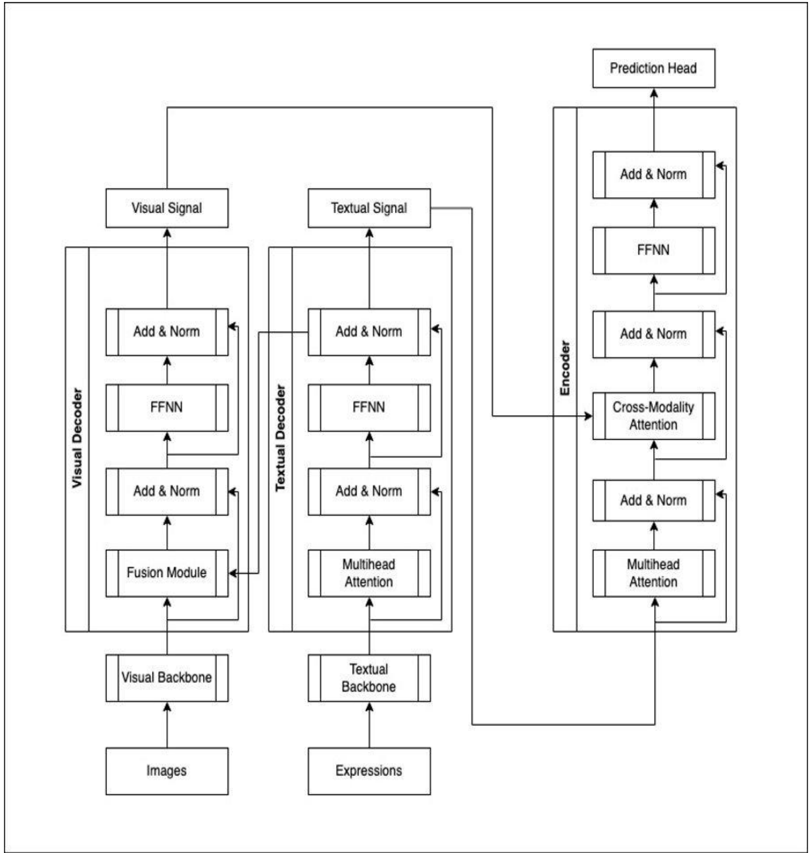


```
a man sitting down with two hands on a remote
```

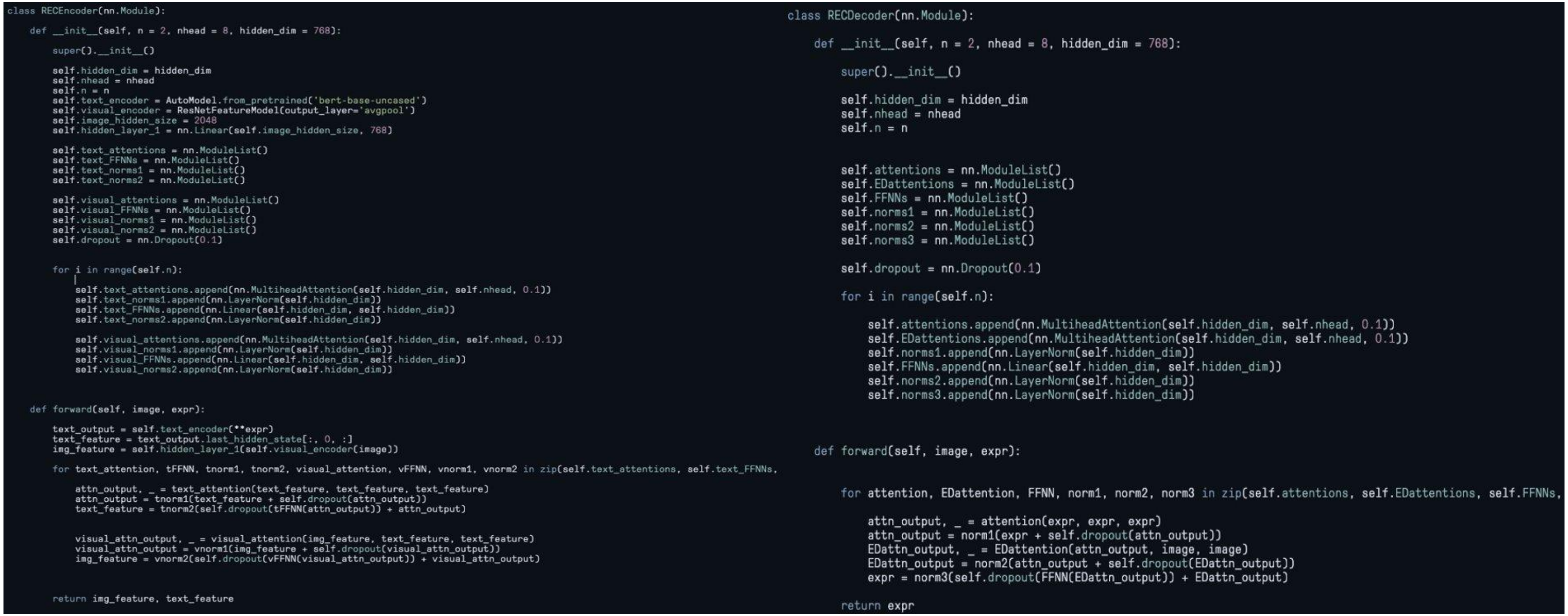


```
the whole pizza with fresh greens
```

Encoder-Decoder Model



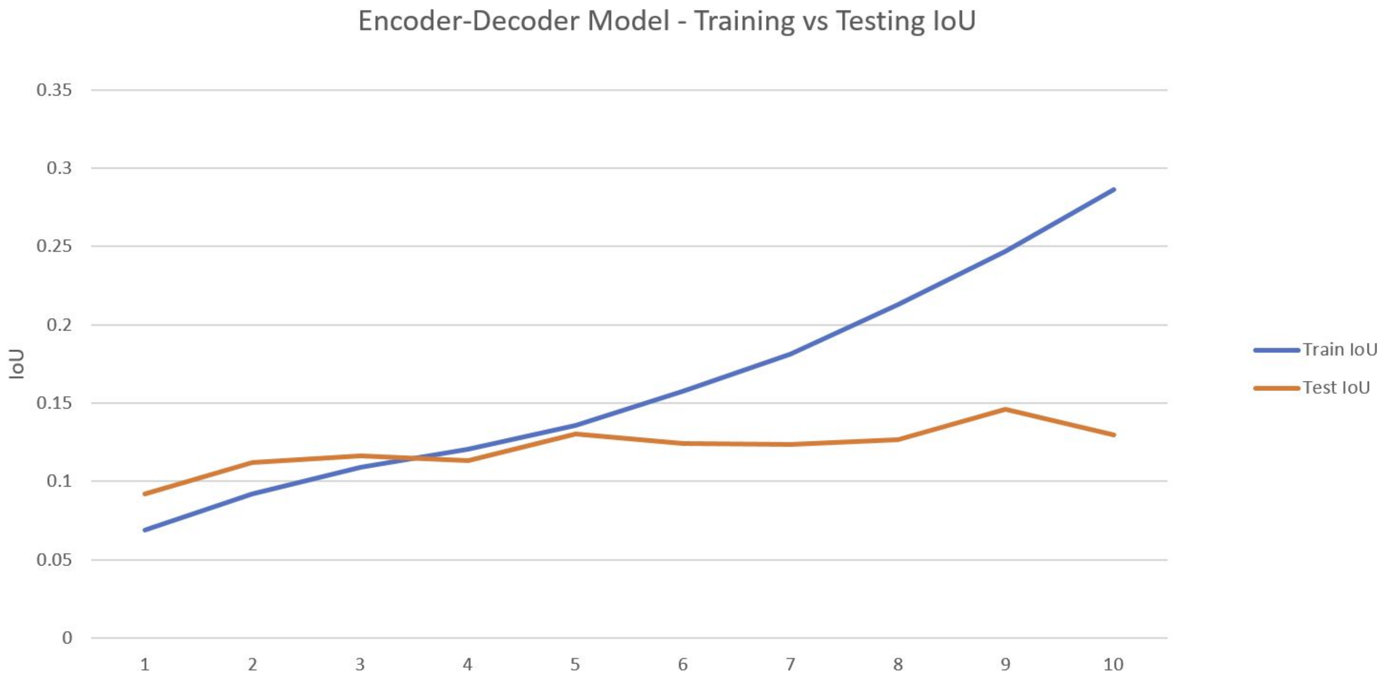
Encoder-Decoder Model



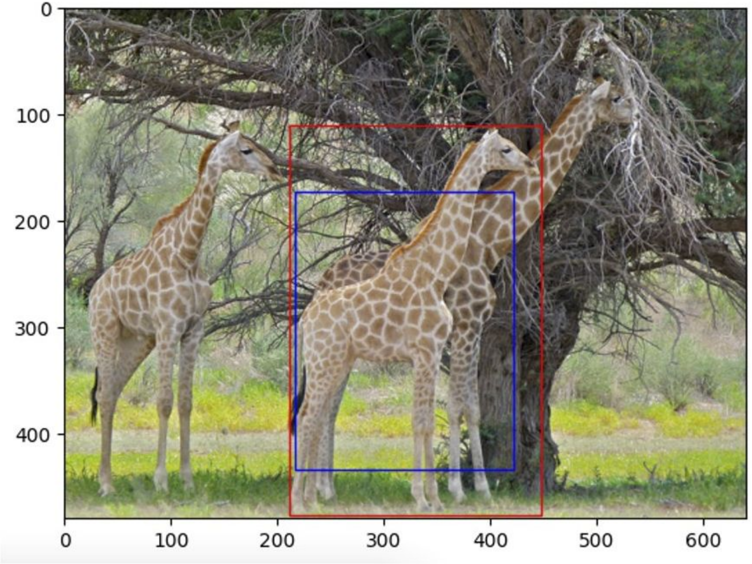
```
class RECEncoder(nn.Module):

    def __init__(self, n = 2, nhead = 8, hidden_dim = 768):

        super().__init__()

        self.hidden_dim = hidden_dim
        self.nhead = nhead
        self.n = n
        self.text_encoder = AutoModel.from_pretrained('bert-base-uncased')
        self.visual_encoder = ResNetFeatureModel(output_layer='avgpool')
        self.image_hidden_size = 2048
        self.hidden_layer_1 = nn.Linear(self.image_hidden_size, 768)

        self.text_attentions = nn.ModuleList()
        self.text_FFNNs = nn.ModuleList()
        self.text_norms1 = nn.ModuleList()
        self.text_norms2 = nn.ModuleList()

        self.visual_attentions = nn.ModuleList()
        self.visual_FFNNs = nn.ModuleList()
        self.visual_norms1 = nn.ModuleList()
        self.visual_norms2 = nn.ModuleList()
        self.dropout = nn.Dropout(0.1)


        for i in range(self.n):

            self.text_attentions.append(nn.MultiheadAttention(self.hidden_dim, self.nhead, 0.1))
            self.text_norms1.append(nn.LayerNorm(self.hidden_dim))
            self.text_FFNNs.append(nn.Linear(self.hidden_dim, self.hidden_dim))
            self.text_norms2.append(nn.LayerNorm(self.hidden_dim))

            self.visual_attentions.append(nn.MultiheadAttention(self.hidden_dim, self.nhead, 0.1))
            self.visual_norms1.append(nn.LayerNorm(self.hidden_dim))
            self.visual_FFNNs.append(nn.Linear(self.hidden_dim, self.hidden_dim))
            self.visual_norms2.append(nn.LayerNorm(self.hidden_dim))



    def forward(self, image, expr):

        text_output = self.text_encoder(**expr)
        text_feature = text_output.last_hidden_state[:, 0, :]
        img_feature = self.hidden_layer_1(self.visual_encoder(image))

        for text_attention, tFFNN, tnorm1, tnorm2, visual_attention, vFFNN, vnorm1, vnorm2 in zip(self.text_attentions, self.text_FFNNs,

            attn_output, _ = text_attention(text_feature, text_feature, text_feature)
            attn_output = tnorm1(text_feature + self.dropout(attn_output))
            text_feature = tnorm2(self.dropout(tFFNN(attn_output)) + attn_output)


            visual_attn_output, _ = visual_attention(img_feature, text_feature, text_feature)
            visual_attn_output = vnorm1(img_feature + self.dropout(visual_attn_output))
            img_feature = vnorm2(self.dropout(vFFNN(visual_attn_output)) + visual_attn_output)


        return img_feature, text_feature
```

```
class RECDecoder(nn.Module):

    def __init__(self, n = 2, nhead = 8, hidden_dim = 768):

        super().__init__()

        self.hidden_dim = hidden_dim
        self.nhead = nhead
        self.n = n


        self.attentions = nn.ModuleList()
        self.EDattentions = nn.ModuleList()
        self.FFNNs = nn.ModuleList()
        self.norms1 = nn.ModuleList()
        self.norms2 = nn.ModuleList()
        self.norms3 = nn.ModuleList()

        self.dropout = nn.Dropout(0.1)

        for i in range(self.n):

            self.attentions.append(nn.MultiheadAttention(self.hidden_dim, self.nhead, 0.1))
            self.EDattentions.append(nn.MultiheadAttention(self.hidden_dim, self.nhead, 0.1))
            self.norms1.append(nn.LayerNorm(self.hidden_dim))
            self.FFNNs.append(nn.Linear(self.hidden_dim, self.hidden_dim))
            self.norms2.append(nn.LayerNorm(self.hidden_dim))
            self.norms3.append(nn.LayerNorm(self.hidden_dim))



    def forward(self, image, expr):


        for attention, EDattention, FFNN, norm1, norm2, norm3 in zip(self.attentions, self.EDattentions, self.FFNNs,

            attn_output, _ = attention(expr, expr, expr)
            attn_output = norm1(expr + self.dropout(attn_output))
            EDattn_output, _ = EDattention(attn_output, image, image)
            EDattn_output = norm2(attn_output + self.dropout(EDattn_output))
            expr = norm3(self.dropout(FFNN(EDattn_output)) + EDattn_output)

        return expr
```

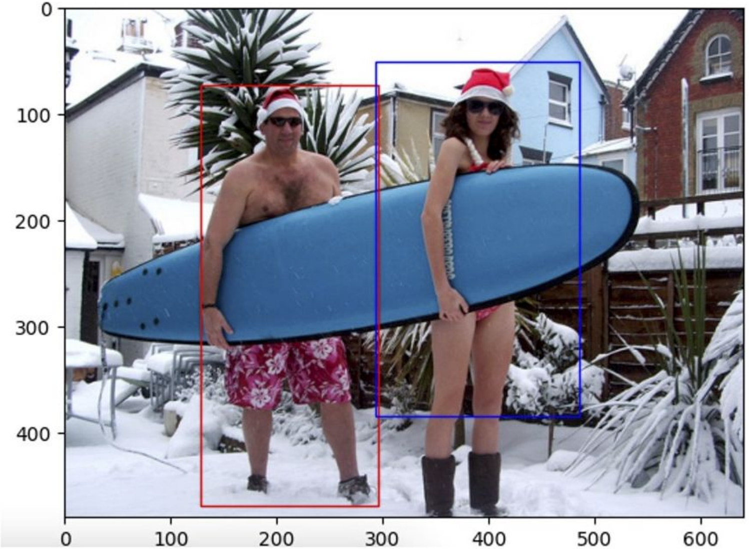
Encoder-Decoder Performance



Encoder-Decoder Examples

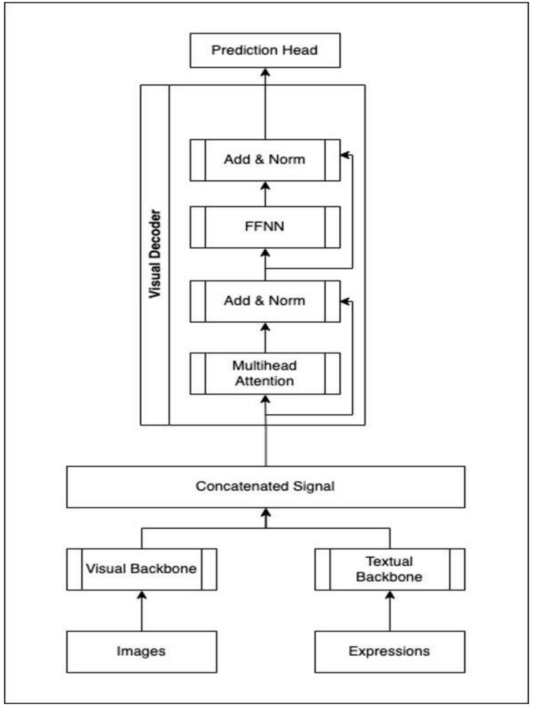


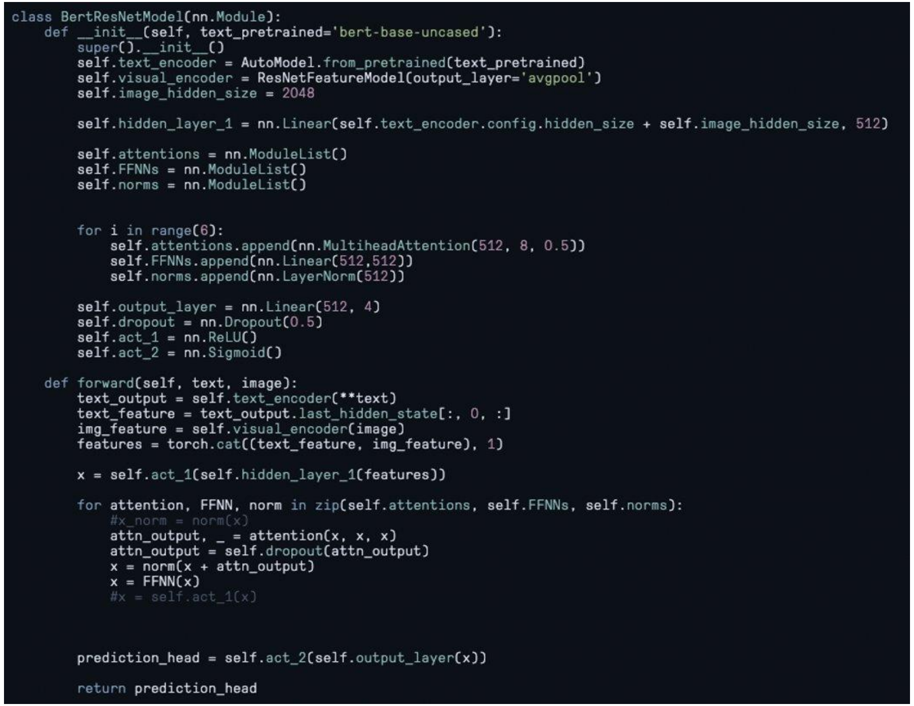
Expression: small giraffe in the middle first to us



Expression: man in back of surfboard

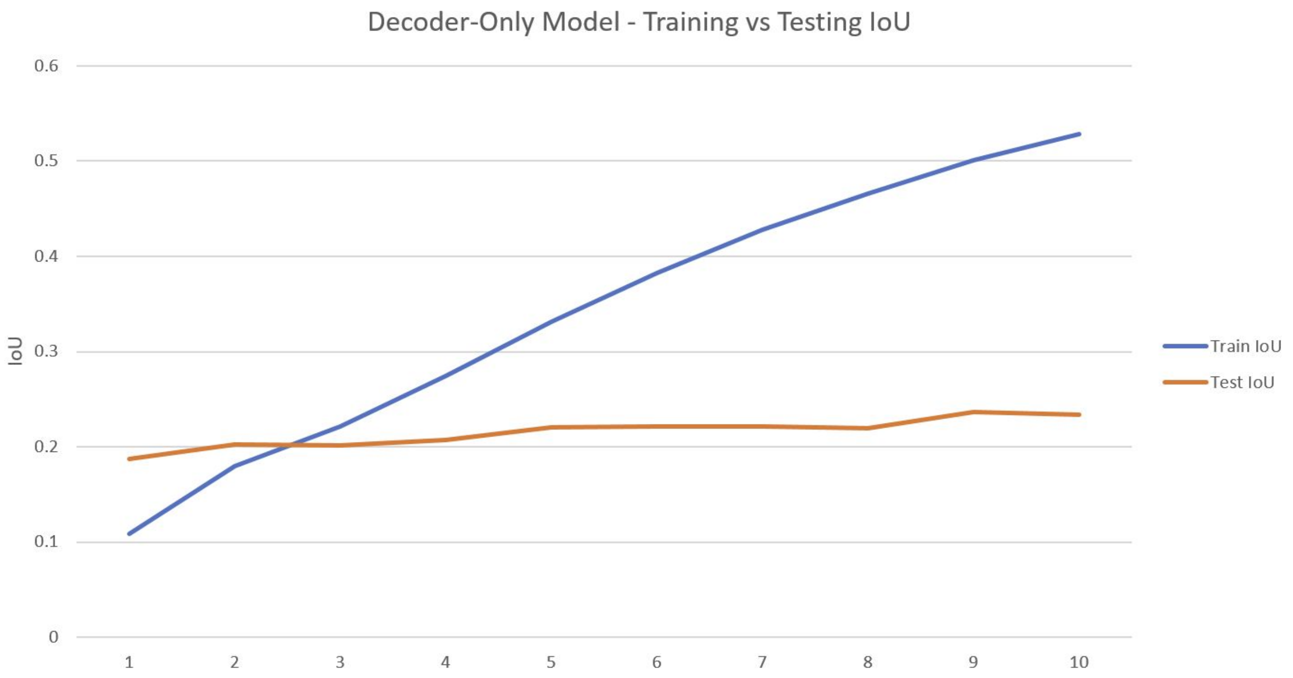
Decoder-Only Model



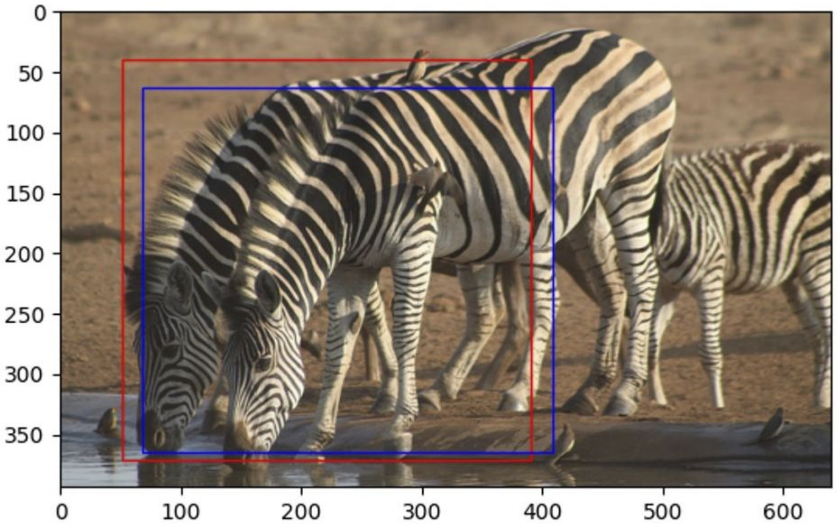


```python
class BertResNetModel(nn.Module):
    def __init__(self, text_pretrained='bert-base-uncased'):
        super().__init__()
        self.text_encoder = AutoModel.from_pretrained(text_pretrained)
        self.visual_encoder = ResNetFeatureModel(output_layer='avgpool')
        self.image_hidden_size = 2048

        self.hidden_layer_1 = nn.Linear(self.text_encoder.config.hidden_size + self.image_hidden_size, 512)

        self.attentions = nn.ModuleList()
        self.FFNNs = nn.ModuleList()
        self.norms = nn.ModuleList()


        for i in range(6):
            self.attentions.append(nn.MultiheadAttention(512, 8, 0.5))
            self.FFNNs.append(nn.Linear(512,512))
            self.norms.append(nn.LayerNorm(512))

        self.output_layer = nn.Linear(512, 4)
        self.dropout = nn.Dropout(0.5)
        self.act_1 = nn.ReLU()
        self.act_2 = nn.Sigmoid()

    def forward(self, text, image):
        text_output = self.text_encoder(**text)
        text_feature = text_output.last_hidden_state[:, 0, :]
        img_feature = self.visual_encoder(image)
        features = torch.cat((text_feature, img_feature), 1)

        x = self.act_1(self.hidden_layer_1(features))

        for attention, FFNN, norm in zip(self.attentions, self.FFNNs, self.norms):
            #x_norm = norm(x)
            attn_output, _ = attention(x, x, x)
            attn_output = self.dropout(attn_output)
            x = norm(x + attn_output)
            x = FFNN(x)
            #x = self.act_1(x)



        prediction_head = self.act_2(self.output_layer(x))

        return prediction_head
```

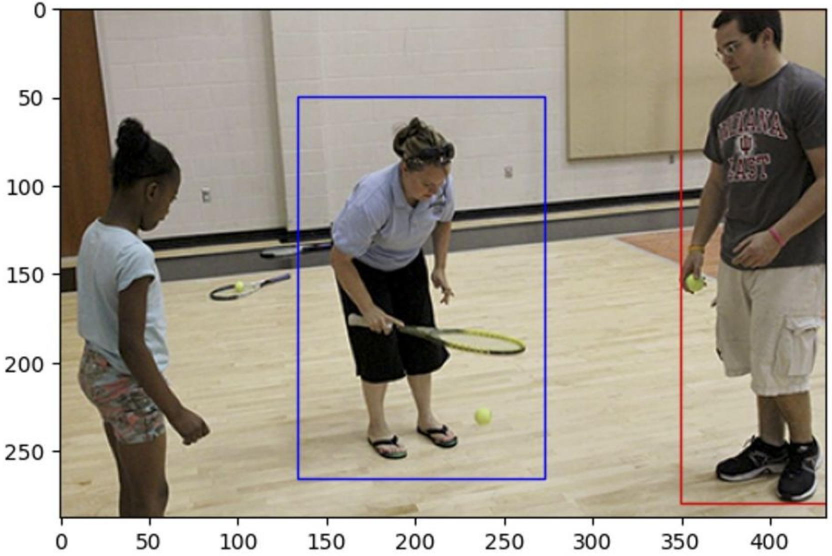
Decoder-Only Performance



Decoder-Only Examples



Expression: drinking zebra on the left



Expression: a man wearing black t shirt and holding a tennis ball in his hand

Further Work

- Train on more data and for more epochs.
  - Fine-tuning pre-trained models (e.g. FLAVA) requires hundreds of epochs. Since one epoch on Google Colab costs around $0.50, we were limited by money.
  - Training models from scratch (e.g. our Encoder-Decoder and Decoder-Only models) requires millions of text-image pairs.
- Tune hyperparameters (e.g., LR, batch size, LR Decay, etc.).
  - We could not do this due to financial constraints.

Questions?