CS6301 Final Project: Referring Expression Comprehension

https://github.com/jxu9001/CS6301-Final-Project.git

Cheung, Kevin Tak Hay (kxc220019) Pingili, Nithin (nxp162430) Xu, Jerry (zxx200003)

Introduction

In this project, we explored the multi-modality field in Deep Learning. Specifically, we are trying to perform a visual-language task called Referring Expression Comprehension (REC). The goal is to localize a target object in the image described by the referring expression phrased in natural language. It is very similar to the visual grounding and object detection tasks. The difference is that the textual query in REC task would provide more details to the model to identify a particular object in the image.

The main challenge of this project is to find an appropriate method to fuse the visual and textual inputs into one embedding and perform bounding box coordinate regression. We have researched on different ideas and built a encoder-decoder model and an encoder-only model. On top of that, we looked into different Visual-language pre-trained models and wanted to see if these models can improve the performance. Eventually, we decided to go with FLAVA (Foundational Language and Vision Alignment Model) [1] due to its simplicity compared to other methods.

Due to the limited resources, we are not able to perform larger scale training. However, we still ran our three candidate models on a subset of RefCOCO data and performed bounding box prediction. With 10 epochs of fine-tuning/training, our models achieved IoU@0.5 of 15% (encoder-decoder model), 23% (encoder-only model) and 32% (FLAVA pre-trained model).

Task

Referring Expression Comprehension - Given an image and a set of text captions, the goal is to localize a target object in the image using a bounding box described by the referring expression phrased in natural language. To accomplish the task, we will be building models that take a pair of image and natural language expression as input and return four coordinates - the location of the bounding box.

The evaluation metric for this task would be Intersection over Union (IoU). It specifies the amount of overlapping area between the predicted and ground truth bounding boxes. A widely used standard would be IoU@0.5, which means the prediction is correct if the IoU is larger than 50%.

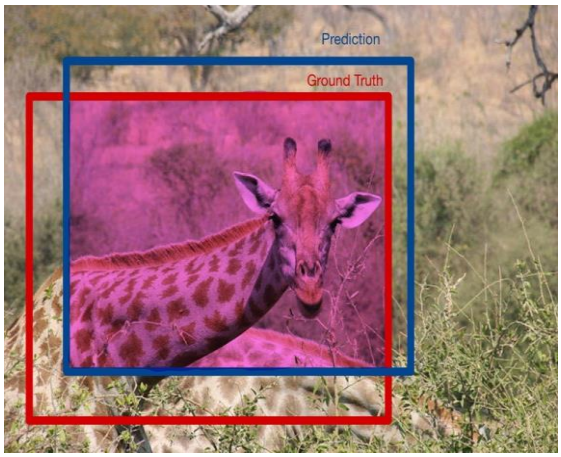

Figure 1 IoU illustration

Data

Throughout the research, we mainly utilized the RefCOCO family of datasets, including RefCOCO and RefCOCO+ from Kazemzadeh et al. 2014 [2] and RefCOCOg from Mao et al. 2016 [3] referring expression dataset. These datasets are the annotations on top of the MSCOCO (Microsoft Common Objects in Context) dataset. The average RefCOCOg expression has 8.4 words while the average RefCOCO and RefCOCO+ expressions have 3.5 words. The dataset statistics are as follows.

Dataset	Train	Development	Test
RefCOCO	40,000 expressions + 19,213 images	5,000 expressions + 4,000 images	5,000 expressions + 4,000 images
RefCOCO+	42,278 expressions + 16,992 images	3,805 expressions + 1,500 images	1,975 expressions + 750 images
RefCOCOg	42,226 expressions + 1,300 images	2,573 expressions + 1,300 images	5,023 expressions + 2,600 images

Figure 2 Statistics of RefCOCO Family

Due to limited resources, we combined these datasets and randomly selected 50,000 image-expression pairs. Our training set contains 40,000 samples and the testing set contains 10,000 samples. Unfortunately, we cannot perform any hyper-parameter fine-tuning due to the high computation cost. Therefore, we did not create any development set.

Methodology

We have used three models throughout this project. We trained all these three models via back-propagation. We selected L1 Loss function as the training objective, which will be used to minimize the predicted and actual coordinates of the bounding box. This metric is chosen since it is the most popular metric used in the visual grounding task. The details of the three models are as follows.

Encoder-Decoder Model

This is a model inspired by the VGTR [4] paper. We followed the author's idea and developed a simplified version. We used a visual backbone (ResNet [5]) and a textual backbone (uncased BERT [6]) to process the images and expressions. Next, these inputs will be imported to visual and textual encoder branches separately. The entire encoder structures can be stacked for N times, depending on the user choices.

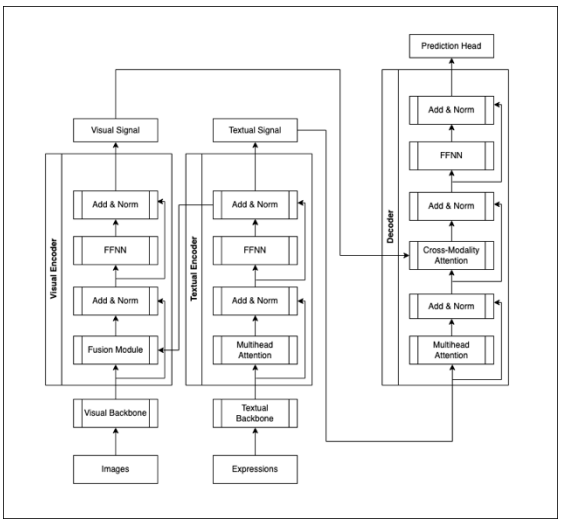

Figure 3 Decoder-Encoder Architecture

Text Encoder

The processed text embeddings will go through Multihead attention layers, Add & Norm layers and FFNN layers. The resulting text embeddings will be used in two ways: 1) an input to the decoder and 2) input to the fusion module in the visual encoder branch.

Visual Encoder

The processed visual embeddings from ResNet will fuse with the post-encoded textual embeddings within the fusion module. The fusion module is a simple Multihead attention mechanism using the visual embeddings as query and textual embeddings as key and value. The joint-embeddings will then go through Add & Norm, FFNN layers and form the multimodal embeddings. The new embedding will be used as input to the decoder.

Decoder

In the decoding stage, the textual embeddings will go through the Multihead attention, Add & Norm layers. Next, the embeddings will be jointly trained with the visual embeddings from the encoders with a Multihead attention layer. As usual, the new embeddings will go through another round of FFNN and Add & Norm operations. Like the encoder, the decoder structure can be stacked for N times depending on users' choice.

Prediction Head

After receiving the output from the decoder, a shallow network (three FFNN layers with ReLU and Sigmoid activations) will be used to perform bounding box regression.

Encoder-only Model

This is a simplified version of the Encoder-Decoder experiment. In this model, we also pass the image and text into the backbones and produce the visual and textual embeddings. Next, we concatenate the two and form a joint embedding. This joint embedding will be passed to the encoder and go through Multihead attention, Add & Norm and FFNN layers. We are hoping these operations can help train a multimodal representation. Similar to the previous model, we can stack the entire structure N times. At the end, the multimodal embedding will be used in the prediction head for bounding box regression.

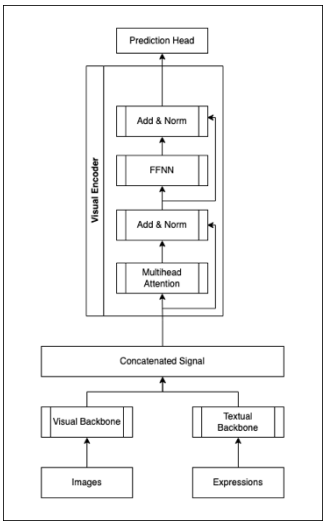

Figure 4 Encoder-only Architecture

FLAVA

FLAVA (Foundational Language and Vision Alignment) [1] is a model that aligns textual and visual information by jointly learning representations of text and image features. FLAVA uses ViT [5] to extract unimodal image representations and unimodal text representations.

The model then aligns the text and image embeddings and creates a combined representation of the input text and image using ViT. This joint representation is then used to predict a set of semantic labels, which describe the relationship between the input text and image.

The model also makes use of a multi-level attention mechanism, which focuses on various granularities in the input text and image, to enhance the alignment between text and image. Additionally, FLAVA is trained using a novel contrastive loss function that encourages the model to learn more discriminative representations of the input text and image.

At the end, a linear classifier will be built on top of FLAVA to predict the four parameters of the bounding box.

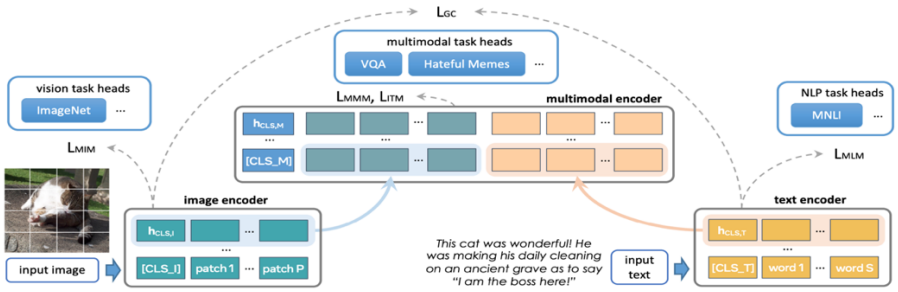

Figure 5 FLAVA illustration

Implementations

Preprocessing

Before we feed the image and text into the networks, we perform several pre-processing steps. For the models requires backbones - We have chosen the ResNet as the visual backbone. It is a pre-trained convolutional network trained on a large image database ImageNet. It helps with extracting the essential features from the images. To process the image via ResNet, we have to resize each image to 224 X 224. Next, we pick uncased BERT as the textual backbone to extract the textual embeddings due to its simplicity. Lastly, for all ground truth bounding boxes, we normalize the coordinates to the range from 0 to 1. With this implementation, we can use the sigmoid activated output layer in the prediction head to perform prediction without worrying about how resizing the image would affect the bounding box. Note that the bounding box is structured as (x, y, w, h), where (x, y) are the coordinates of the top left corner of the bounding box and (w, h)

Encoder-Decoder Model

We built the entire architecture using Pytorch. The following snippets show how both the Encoder and Decoder classes are structured.

```
for i in range(self.n):
    # Encoder Structure in the Encoder Class
   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))

for i in range(self.n):
    # Decoder Structure in the Decoder Class
   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))
```

After these two classes are defined, we will pass the image and text embeddings (processed by the backbones) to the encoder. The forward function for the encoder is as follow:

```
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 all_layers:

      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
```

The function takes the embeddings and processes as described in the previous section. Eventually, it will return a pair of image and text embeddings. These embeddings will be passed along to the decoder forward function, which is shown below:

```
def forward(self, image, expr):

    for attention, EDattention, FFNN, norm1, norm2, norm3 in all_layers:

        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
```

The decoder will again handle the two embeddings produced from the encoder and go through the process described in the previous section. Eventually, it will produce a joint embedding. This embedding will then be passed to the prediction head, which is a simple bounding box regression network:

```
self.hidden_layer_1 = nn.Linear(768, 512)
self.hidden_layer_2 = nn.Linear(512, 512)
self.output_layer = nn.Linear(512, 4)
```

Encoder-only Model

Like the previous model, we defined a new class using Pytorch:

```
for i in range(6):
    # Encoder Structure within the Encoder class
    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)
```

Using this model class, we can pass the text and image into the network and perform training. Note that the text input will be processed within this class, while the image will be handled by the backbone before being passed into the network. The forward function is defined as follow:

```
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):
        attn_output, _ = attention(x, x, x)
        attn_output = self.dropout(attn_output)
        x = norm(x + attn_output)
        x = FFNN(x)

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

The text and image embeddings will be processed as the way described in the previous section. However, the prediction head in this model is slightly different from the ones in the encoder-decoder model. It is because we

are trying to build a light-weighted model and we do not want to have too much training on the prediction head. Therefore, we only utilized one output layer activated with sigmoid function to transform the embedding to four coordinates.

FLAVA

First, we froze the parameters in FLAVA to prevent them from updating during training. Then, we built a linear classifier on top of FLAVA to predict the four parameters of the bounding box. The following code snippet shows how FLAVA was constructed.

```
class FLAVAModel(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
```

Similar to previous models, we used a sigmoid activation function when predicting the four parameters of the bounding box to make the predicted parameters scale-invariant.

Experiments and Results

Test Results

The models were evaluated using the intersection over union metric. As suggested in the previous section, we will be evaluating the models using IoU@0.5.

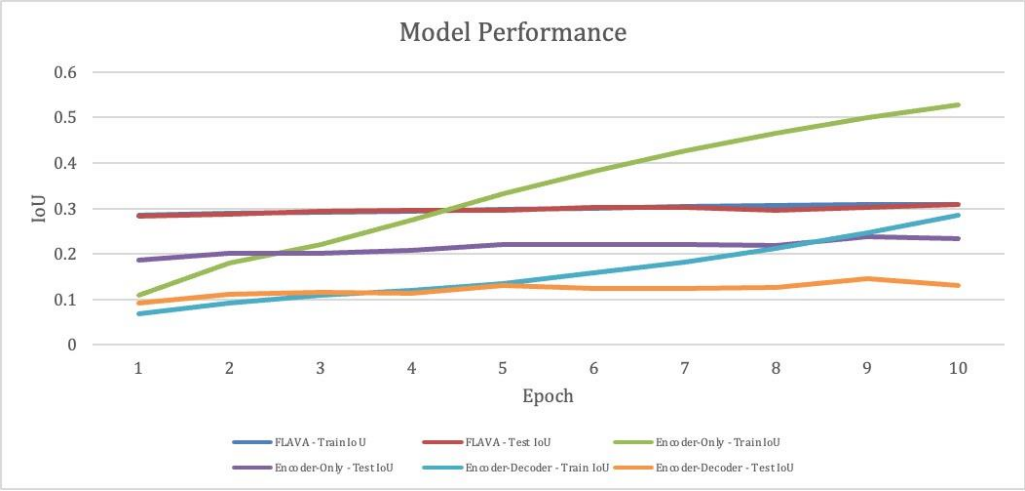

Figure 6 Model Performance

Model	Train IoU	Test IoU
Encoder-Decoder	28.6%	13.0%
Encoder-Only	52.8%	23.4%
FLAVA	31.0%	30.9%

Figure 7 Train and Test IoUs

From the graph, we can see that the encoder-decoder model and encoder-only model suffer from overfitting. Also, the test IoUs are much worse than the train IoUs for both the models. We believe the bad performance from the two models are due to insufficient training. The 40,000 image and expression pairs are far from enough for the network to distinguish the objects in the images and spatial relationship between the objects.

Interestingly, the train and test IoU for the FLAVA model are very similar. The FLAVA model has better test IoU than the other models. Although we fine-tuned the models with 10 epoches, we can see the train and test IoUs are still improving, and it does not look like there exists any overfitting. We believe this is because this pre-trained model has been trained on a large dataset of images and expressions. It contains much more knowledge than the other two models.

Error Analysis

In this section, we will look into some incorrect samples and see how we can improve the performance of the model.

FLAVA

In the left image of figure 8, the expression (the whole pizza with fresh greens) asks for the whole pizza. However, the FLAVA model output draws a bounding box for a small part of the pizza. We think the model needs to be trained for more epochs to better understand the visual features and text features, so it is able to recognize the entire pizza and draw a bounding box.

Encoder-decoder model

In the middle image of figure 8, the expression is “man in back of surfboard”. However, the wrong person was identified as the man behind the surfboard. The model recognizes that there is a person behind the surfboard but could not differentiate between the man and woman resulting in the model choosing the wrong person referred to in the expression. We think training on more examples of male and female may improve the model performance.

Encoder-only model

In the right image of figure 8, the expression is “a man wearing black t shirt and holding a tennis ball in his hand ”. However, a woman is predicted who does not match the description of a man wearing black t-shirt and holding a tennis ball in his hand. Despite the detailed description of the person, the model failed because it has not seen these kinds of image-text pairs in the training phase. Therefore, we think we need to train on more image-expression pairs. Also, we need to train the model for more epochs.

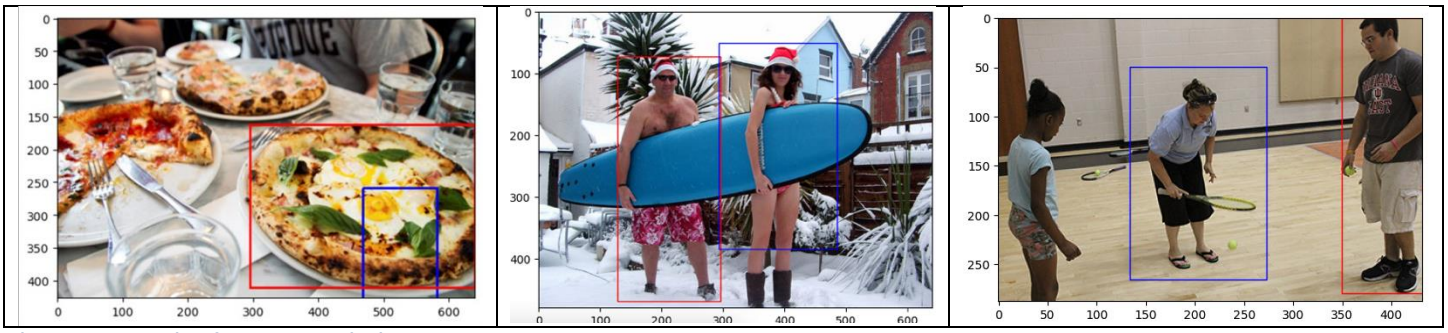

Figure 8 Examples for Error Analysis

Speed Analysis

The following chart shows the speed for training and testing:

Model	GPU	Training Time	Testing Time	GPU Cost
Encoder-Decoder	A100	15 mins / epoch	0.6ms / example	$0.78 per epoch
Encoder-only	A100	15 mins / epoch	0.2ms / example	$0.78 per epoch
FLAVA	A100/V100	25 mins / epoch	89.2ms / example	$0.50 per epoch

We observe that FLAVA took a lot longer in both training and inference. It makes sense because there are more layers in FLAVA, and the hidden sizes are a lot bigger than our Encoder-Decoder and Encoder-only models.

Conclusion

In this paper, we have demonstrated the use of encoder-decoder model, encoder-only model and FLAVA pretrained model to localize a target object in an image described by a referring expression phrased in natural language. Unsurprisingly, the FLAVA model performed the best, with IoU = 32% on the test set because it was pretrained on a large dataset of image-text pairs. It was interesting to see that the encoder-only model performed better than the encoder-decoder model even though it is a more simplified version of the encoder-decoder model. However, both models suffer from overfitting. Overall, all three models need to be trained for more epochs and more image-expression pairs to achieve better performance.

Discussion

1. What is the difference between Object Detection and Referring Expression Comprehension?
   For object detection task, the goal is to identify and classify all objects that are present in an image without any linguistic input. Most state-of-the-art object detection models detect at most 100 objects in each image. Each detected object will be assigned to a class and the confidence of the prediction. However, in REC, the goal is to interpret the meaning of the given natural language expression and produce a bounding box around the referred object in an image. Even if multiple objects of the same class exist in the image, only one of them should be chosen in the bounding box.

   We considered using these object detection models, such as YOLO, as our visual backbone since they provide a lot more information than ResNet. However, the inference costs for these models are always very high and we cannot afford this extra computation requirement.

2. What is the difference between Visual Grounding and Referring Expression Comprehension?
   Visual grounding is the broad concept of identifying several object regions in an image that correlate to multiple noun phrases from a sentence that describes the scene. While REC focuses on understanding natural language expressions and localizing objects in images.

References

1. A. Singh, R. Hu, V. Goswami, G. Couairon, W. Galuba, M. Rohrbach, and D. Kiela, “Flava: A foundational language and vision alignment model,” 2022.
2. S. Kazemzadeh, V. Ordonez, M. Matten, and T. Berg, “ReferItGame: Referring to objects in photographs of natural scenes,” in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 787–798.
3. J. Mao, J. Huang, A. Toshev, O. Camburu, A. Yuille, and K. Murphy, “Generation and comprehension of unambiguous object descriptions,” in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 11–20.
4. Y. Du, Z. Fu, Q. Liu, and Y. Wang, “Visual grounding with transformers,” 2022.
5. K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015.
6. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” 2019.
7. A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, 2 M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An image is worth 16x16 words: Transformers for image recognition at scale,” 2021.