d. What method are you using? You should provide an explicit formal description of the method.

In this project, what we try to accomplish is use the Raster2Seq methodology to predict pose estimation on the MP-100 dataset. This means that the pose estimation problem will be treated as a sequence generation problem. In more detail traditional approaches, given an input image, they would predict (x,y) coordinates for each keypoint that describe the pose, whereas with the use of Raster2Seq we will generate a sequence of tokes that describe the keypoints. The methodology followed consists of 8 parts:

- Input and Output
- Sequence Representation
- Coordinate Tokenization
- Model Architecture
- Autoregressive Generation Process
- Loss Functions
- Training Procedure

Input and Output

The Input to our model will be and RGB image that will contain one object of the 84 categories (out of 100 we managed to obtain 84 of them). Some examples are a photo of cat, human, locust, alpaca etc. The output of the model will be a sequence of N keypoints. Each keypoint has coordinates and a semantic meaning and the number really depends on the category since a cat could have 20 keypoints and a car only 10.

Sequence Representation

Instead of predicting coordinates directly, Raster2Seq converts the problem into predicting a Sequence of Tokens. The sequence looks like this:

[<coord>, x_1_token, y_1_token, <sep>, <coord>, x_2_token, y_2_token, <sep>, ..., <eos>]

Where:

- <coord> = special token meaning "a keypoint is coming"

- x_1_token = discrete token representing the x-coordinate of keypoint 1

- y_1_token = discrete token representing the y-coordinate of keypoint 1

- <sep> = separator token between keypoints

- <eos> = end-of-sequence token (signals we're done)

The choice of using Sequence of Tokens rather than directly number of the coordinates of the keypoints allows us to use powerful transformer models designed for sequences transforming the selection of coordinates from a regression problem to a classification problem (which token?).

Coordinate Tokenization

Following the aforementioned steps, an issue arises as to "how to convert coordinates to tokens since coordinates are continuous and tokens are distinct. To address this, we use Bilinear Interpolation over Learned Embeddings. More specifically we first learn a table of vectors where each row is an embedding for an integer coordinate, like 0, 1, 2, ..., up to V = vocabulary size (number of possible coordinate values). When we get a coordinate that might not be an integer (for example 12.7), we find the two closest integers around it (12 and 13), look up their embeddings, and then take a weighted average of those two vectors: more weight on 13's embedding if the value is closer to 13, more weight on 12's if it's closer to 12. This lets the model treat coordinates as continuous values while still using a fixed set of discrete learned embeddings.

Model Architecture

The Raster2Seq model has three main components:

- Visual Encoder
- Transformer
- Prediction heads

The visual encoder is a ResNet-50 convolutional neural network pretrained on ImageNet that extracts hierarchical visual features from the input image. It takes a 512×512×3 RGB image and produces a 2048×16×16 feature map by downsampling through successive convolutional layers, creating a rich spatial representation that captures both low-level details and high-level semantic information about object structure and keypoint locations.

The transformer decoder consists of 6 stacked layers, each containing multi-head self-attention, cross-attention with deformable sampling, and feed-forward networks with layer normalization. The self-attention mechanism models dependencies between previously

generated keypoint tokens in the sequence, while the cross-attention mechanism attends to relevant spatial locations in the image feature map to determine the next token, enabling the model to autoregressively generate keypoint coordinates conditioned on both the image content and prior keypoint predictions.

The prediction heads consist of two parallel linear layers that operate on the final decoder embeddings:

- a classification head that predicts discrete token IDs over a vocabulary of coordinate values and special tokens,
- and a regression head that directly outputs continuous (x,y) coordinate values.

The classification head enables the model to select discrete coordinate tokens through cross-entropy optimization, while the regression head provides fine-grained coordinate adjustments through L1 loss, with both outputs combined during training to leverage the benefits of both discrete and continuous representations.

Autoregressive Generation Process

The model generates the sequence one token at a time, like writing a sentence and continues until <eos> token is generated. A key element of this process is self-attention where also masking is applied to ensure that at each position can only attend to previous positions. This prevents "cheating" by looking at future tokens.

Loss Functions

In Raster2Seq, we train the model using two loss functions combined:

- Classification Loss (L_ce): This is the cross-entropy loss for predicting the correct token. For each position t in the sequence the model predicts probability distribution over all tokens: p(token_t | token_{1:t-1}). The ground truth is the actual token at position t. At the end, we take the average over all positions and all samples in the batch.
- Coordinate Regression Loss (L_coords): This is the L1 loss for predicting exact coordinates. This is L1 distance (Manhattan distance) averaged over all keypoints.

Thus the Total Loss is computed through this formula

$L\_total = \lambda_1 \times L\_ce + \lambda_2 \times L\_coords$

Where $\lambda_1 = 1.0$ and $\lambda_2 = 5.0$

Training Procedure

The details of the training procedure we followed can be summarized to this:

- Optimizer: AdamW with learning rate 10−4
- Batch size: 2 (limited by GPU memory)
- Learning rate schedule: Step decay, reducing by 0.1 at epoch 200
- Total epochs: 300 however to get preliminary results we conducted training with just 5 epochs to get a good look of the results since training is time consuming
- Data augmentation: Random horizontal flip, random crop, color jittering
- Hardware: Initially we used Apple M-series GPU with MPS backend and then we continued using CUDA

At each epoch, we loaded the images and the ground truth keypoint sequences of a batch and then forwarded pass through encoder to get the image features. After that, a forward pass through decoder was implemented to get token predictions. Using these predictions and the ground truth, we computed the total loss as the sum of the cross-entropy loss over the predicted tokens (L_ce) and the coordinate loss measuring how close the predicted keypoint coordinates were to the true ones (L_coords). Each iteration, completes with a backward pass by computing the gradients and updating the weights through the Optimizer.