Approaching Object detection and Panoptic segmenation problem by DETR

Viet-Hoang Tran
Faculity of Computer Science, University of Information Technology, HCM.C.
Email: 18520785@gm.uit.edu.vn

Abstract. In this article, we introduce the panoptic segmentation problem of image, its evaluation metric, and provide the 2020 S.O.T.A Detection Transformer (DETR) as the optimal solution. Then explain explicitly how does DETR performs object detection: the Transformer architecture, the process of extracted image's feature transformation to Transformer's digestible, the queries for detecting an questioning object and the loss computation, also the optimization alogrithm. In order to deploy panoptic segmentation, the panoptic head is added to notice each detected object and render the segmentation version of image. Finally, clarify its powerful properties by implement COCO-2017 (cocodataset.org) pretrained model and applying it to Wheat Head dataset [11]. The code and outperform result are available at: thoconvuive(github)

1. Introduction

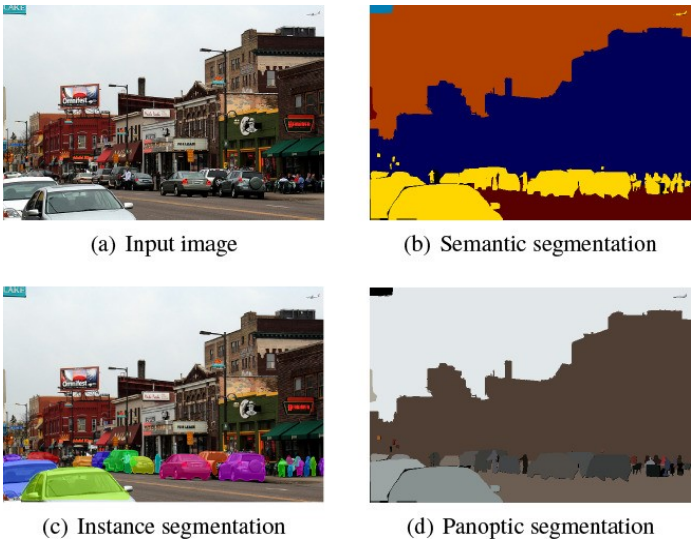
Transformers-the advanced seq2seq architeture, relies on a simple yet powerful mechanism called attention. Have been widely applied to solve problems with sequential data, in particular in natural language processing (NLP) tasks such as language modeling and machine translation, and have also been extended to tasks as diverse as speech recognition, symbolic mathematics, and reinforcement learning.

Following this archivements, Carion et al. [1] release Detection Transformer (DETR), an important new approach to object detection and panoptic segmentation problem in computer vision feld. It removes many hand-crafted process based on optimization/penalty system. Receiving a set of images as input, DETR can automatically extract features and gives each pixel in image an opportunity to know how they intereact with each others. On the other hands, it creates queries, these queries have a chance to *look* at the pixels's attention. Their duty is asking a seperate and good question to the image and help it detect object, as well as bounding/segmenting their location.

2. Problem Approaching

2.1. Objective

As we know it, the *semantic segmentation* is a long fundamental computer vision tasks which has many research works. The goal of this problem is classifing objects features in the image and comprised of sets of pixels into meaningful classes that correspond with real-world categories. On the other hand, *instance segmentation* identifies each instance of each object featured in the image instead of categorizing each pixel like in semantic segmentation.



(a) Input image (b) Semantic segmentation

(c) Instance segmentation (d) Panoptic segmentation

Figure 1: the comparison of three segmentaton forms.(Source: [2]*)*

Panoptic segmentation

Inherit two tasks above, *panoptic segmentation* (PS) is the combination of both types segmentation above. Its

work is not also classifying all the pixels in the image as belonging to a class label, yet also identify what instance of that class they belong to. The word *panoptic* means considering all parts or elements all inclusive, and indeed, it segments things, class of things including *stuff*. PS makes a lot of sense in the real world, such as identifying cars, trucks, walker, pavement,... on the road (category of things having instance-level annotation that our driving car must not cross over) and stuff (category of things without instance-level annotation that we can cross: road, rain cover, pond, etc).

2.2. Evaluation Metric

From the panopticapi, the COCO dataset provides us the evaluation script. Which is called PQ (Panoptic Quality) metric. PQ has two steps, segment matching and PQ computation.

Let us find out the first step. For each image, we determine to locate a predicted segment and ground truth satisfied IoU metric with the matching threshold 0.5. Along with the non-overlapping property of a panoptic segmentation, gives an *unique matching*: as most one predicted segmentation can be found for each ground truth. Here is the theorem which can be found with its explanation in [3].

For another part, with each matching, given the p as a predicted segment, g as a ground truth and each segment falls into one of three sets TP, FP and FN as respectively are true positive, false positive and false negative. We computate the PQ independently for each class and average over for all. PQ computation recipe can be found below:

$$\mathrm{PQ} = \frac{\sum_{(p,g)\in TP} \mathrm{IoU}(p,g)}{|TP| + \frac{1}{2}|FP| + \frac{1}{2}|FN|}.$$

It is quite complexity to digest, so break it into two parts for better approaching, also for facilitating interpretation of results.

The first computation session is the segmentation quality (SQ), this metric evaluates how closely matched with a predicted segment with corresponding ground truth. But what about non confident cases? It does not take any account of the bad predictions.

The second session: region quality (RQ) is generated to solve the problem. RQ is familiar to the F1 score of precision and recall, which is usually applied for quality estimation in detection settings. Similar to mAP, F1-score requires both components to be high concurrently. Its role is attemping to identify how affective of the system is getting a prediction right. Another approaching of PQ metric can be found by assembling two composition above:

$$\mathrm{PQ} = \underbrace{\frac{\sum_{(p,g)\in TP} \mathrm{IoU}(p,g)}{|TP|}}_{\text{segmentation quality (SQ)}} \times \underbrace{\frac{|TP|}{|TP| + \frac{1}{2}|FP| + \frac{1}{2}|FN|}}_{\text{recognition quality (RQ)}}.$$

Why PQ metric? The standard metric for instance segmentation is mean Average Precision (mAP) metric. In order to calculate mAP metric, confident scores is required so that we can rank predictions from highest to lowest for producing the precision/recall curve. It is quite natural for object detection (or instance segmentation) but it has shown no basis to semantic segmentation (which requires uniformity).

Basic evaluation metric IoU for semantic segmentation also causes issues. This metric ignores instance labels, i.e we just have one segment to compare which one ground truth for each class. So it is not suitable for evaluating things belong to one class. This metric is distinct from our segmentation quality (SQ), the one calculate all pixel labels and ignore object-label levels, and another computates the average IoU of matched segment.

2.3. Transformer for object detection

The paper *Attention is all you need* [4], publishing *Transformer* architecture, changed the NLP game and has achieved great hieghts. It is relied on a simple yet powerful mechanism called *Self-attention.* Attention not only enables deep learning model to selectively focus on certain parts of their input and thus reason more effectively, but aslo transmit and process our data to train parallelly (that creates a huge advantage due to ultilise GPU's power).

Following this achievement, DETR (Detection Tra nsformer) is realised, it produces a new important approach for panoptic segmentation.

Advantages

With both two-stage detector predicts proposal boxes containing R.O.I by hand-crafting selective search or proposal network, whereas single-state detector predicts the bounding boxes and evaluates class probabilities for each. Their problem arises when too many anchor boxes are created, meanwhile a few

number of classes that an image contains. In addition to performance issues, this may cause the difficulty to evaluate the objective function, when a majority of anchor boxes that not contain things and the rest are vice versa. DETR doesn't need to manually set the number of anchor box, aspect ratio, default cordinates of bounding boxes, even threshold for non-maximum surpression. DETR entrusts all those task to encoder-decoder Transformer and bipartite matching.

3. The DETR model

DETR predictor has two essential ingredients: an DETR architecture that predicts (in a single pass) a set of objects and models their relation (3.2); and a set prediction loss that forces unique matching between predicted and ground truth boxes (4.1). In this article, we will introduce DETR architecture first, due to necessarily clarify the Transformer.

3.1. Transformer interpretation

The Transformer architecture also uses 2 parts Encoder and Decoder (shown left and right halves in the Figure 2 respectively). The Encoder converts features which extracted by the backbone into its latent representation in the form of hidden state vectors. From Encoder's output, the Decoder tries to predict the output sequence using this mentioned representation. Both Encoder and Decoder are composed of modules that can be stacked on top of each other multiple times.

3.1.1. Encoder and Decoder

For Encoder, each layer has two sub-layers, it has a *multi-head self-attention* mechanism and position-wise fully connected *feed-forward network*. It employs a *residual connection* around each of the two sub-layers, followed *by layer normalization*.

For Decoder, in addition to sublayers above, performing multi-head attention over the output of the Encoder stack. Similar to the Encoder, it emloys residual connections around each of the sub-layers, followed by layer normalization. The first sub-layer is modified to be *masked* in order to ensures that the predictions for position *i* can depend only on the known outputs at positions less than *i.*

Positional encoding

Since the Transformer architecture is permutation in-

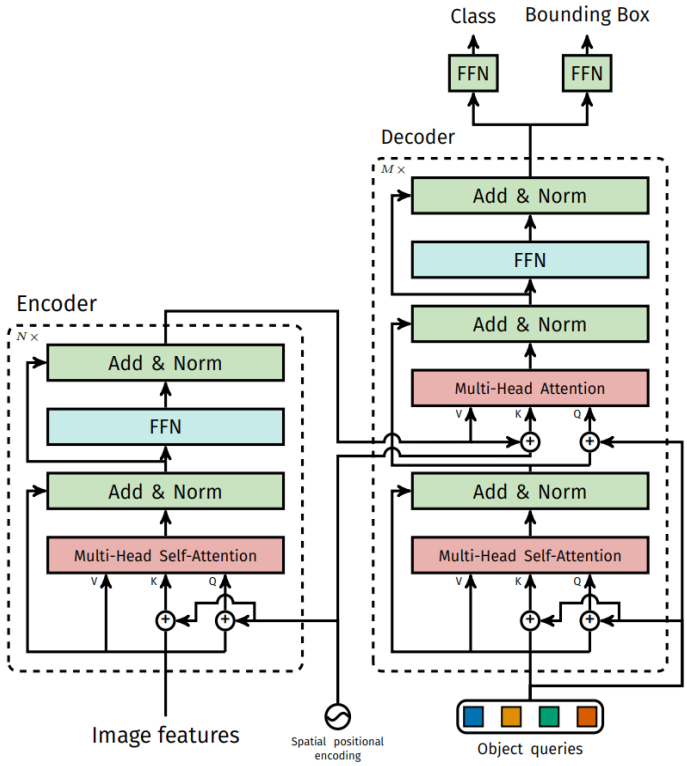


Figure 2: The model architecture of DETR Transformer [1]

variant. In order to assure the order of feature in the sequence, we need to inject some information about the relative position of the tokens in it. This is the role of *positional encoding* module. It locates a bottom of each Encoder and Decoder's stack. There are various positional encodings for the Transformer, the standard mathematic formulation is presented as follow:

Given frequencies:

$$w_i = \frac{1}{10000^{2i/d}}$$

The positional encoding (PE) calculation for feature at position *pos* of a certain dimenstion *i*:

$$\mathrm{PE}_{(pos,i)} = \sin(w_i.pos) \quad \text{if } i = 2k$$
$$\mathrm{PE}_{(pos,i)} = \cos(w_i.pos) \quad \text{if } i = 2k+1$$

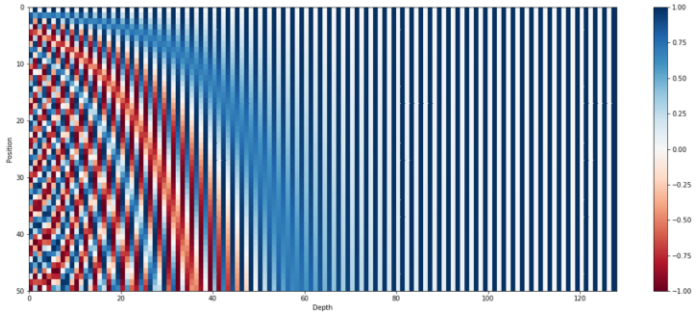
The position encoding vector for feature at position *pos* against others of every dimensions:

$$\mathrm{PE}_{(pos)} = [\sin(w_1.pos), \cos(w_1.pos) \ldots]_{1\times d}$$

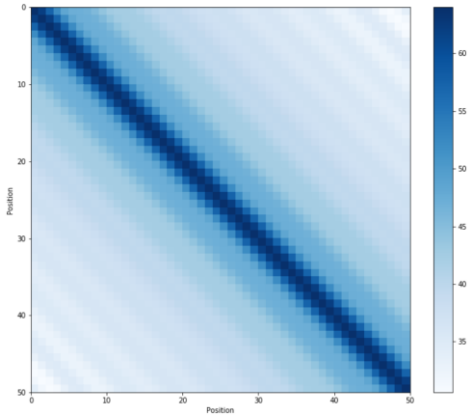
Where *pos* is the position and i is the dimension which *pos* belongs to. And the vector $\mathrm{PE} \in R^d$ contains pairs of sines and cosines for each frequency, where *d* is the number of dimensions. Each dimension corres-

ponds to sinusoid wavelength. And the frequencies w_i are decreased along dimensions. This approaching method is similar to binary encoding, but it has more efficent and flexible. [5]

Another characteristic of sinusoidal positional encoding is that it allows the model to attend relative positions effortlessly. *"Since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}."* [4] (easily proved by trigonometric addition formulas, the matrix PE_{pos+k} is the matrix PE_{pos} clockwise rotated). The illustration of positional encoding output and the correlation among encoded positions can be found in figure 3 and figure 4 respectively.



Figue 3: The 128-dimensional positonal encoding for a sentence with the maximum length of 50. Each row represents the encoding vector PE_{pos}



Figue 4: The correlation of a feature among the rest of embedded positions (calculated by dot product). Closer features will be given more attention (which get darker blue on figure).

The result of this progress is the positional encoded feature which is the *sum* of positional encoding and embedded input.

3.1.2. Attention

Self-attention

This mechanism aims to encode a feature component from a embedded (positional encoded) sequence. It measures the encoding of the feature against the encoding of others and gives a new encoding. The description of mechanism can be described by the equation aboves:

$$\text{Attention} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right)V$$

For comprehensible, it should be separated into basic steps below:

Step 1. Create three parts out of every embedded feature: Given an embedding x, it *learns* three separate smaller embeddings from it query ($q \in Q$), key ($k \in K$) and value ($v \in V$). They have the same number of dimensions. Learning is the progess of updating W^Q, W^K and W^V matrices based on the loss which has back-propagated. x is multiplied with these three matrices to get the query, key and value embeddings.

Considering embedded feature as a living entities, Each feature want to know much each other values it, then create a better version of itself that represents this valuation.

Step 2. Given the feature x_1 wants to know its value with respect to x_2. So it'll query x_2. x_2 will provide the answer in the form of its own *key*. The key can be used to get a value-represented score by measure its value by taking a dot product with the query. Then scaled into $\sqrt{d_{k,q \text{ or } v}}$ deviding for assurance the dot product does not go large in magnitude (which may causes exploding gradient). And x_1 takes all this scores and perform softmax to distributing from 0 to 1. The scoring of softmax computation is performed by every feature against the others.

Step 3. x_1 now multiply its softmax score and the *value* vector to get a new *value* of itself with respect to that feature. If the other features just have a little or not revelant, score will be reduced, bring about the *value.* On the other hand, the significant others get their value consolidated.

Step 4. x_1 creates the new *value* by summing up the value received from step 3 above. This is the new embedding of the feature. This result is also called the self-attention of x_1. [6]

Based on the mentioned positional encoding method, the features will pay more attention to details of

the local features. Self-attention can be known as local attention.

Multi-head attention

After each component has their "self-attention". If we just have the attetion mechanism for a feature in the individual sequence, we can not be possible to generalize the feature that is used in multiple contexts. Multi-head attention is generated in order to attend a various features parallely in a sequence.

Set of q, k, v are seperated by respectively learned weights and embedding x_1, x_2, x_3,... Then self-attention. are performed on each set separately and new embeddings v'_1, v'_2, v'_3,... are created for each set. Then these are concatenanted and multiply with W^O (weighed sum) reducing the mutiple embeddings into single one. Each v' represents a head of the self-attention model. Concatenate and synthesize multi v', we have the multi-head attention.

Multi-head attention process are redacted by the formulation below:

$$\text{MultiHead} = \text{Concat}(head_1, \ldots, head_h)\, W^O$$

$$\text{where} \qquad head_1 = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$$

Layer normalization

For batch normalization, input values of the same neuron from different images in one mini batch are normalized. In layer normalization, input values for different neurons in the same layer are normalized without consideration of mini batch. In Transformer, each input is considered to be sequential, due to the data tells individual story in each different stage. Thus the caculation of mean/var must be independence. Thefore the layer norm is chosen.

As other recent network architectures, The Transformer also applies residual connection in order to reduce the complexity of the network, thus help it goes deeper.

Feed-forward network

Each Encoder and Decoder block contains a feed-forward neural net. It consists of two Linear layers with a ReLU activation between them. It is applied to each position separately and identically.

$$\text{FFN}(x) = W_2.\text{ReLU}(W_1 x + b_1) + b_2$$

Arrcoding to the article [4], FFN is applied to each position separately and identically. Hence the input to it is a set of embeddings x_1, x_2, x_3,.... and the output is another set of embeddings x_1, x_2, x_3,... have the same dimension.

3.1.3. Transformer process

The Encoder extracts the embedding feature into sets of *key* and *value* by from multi-head attention. It plays a role of a memory cell, similar to hidden state vector in recurrent neural network.

The Decoder's input is embedded ground truth feature, masked as mention above. The decoding process is corresponding to encoding. However, the masked feature from the first sub-layer does not

Figure 5: A visualization of self-attention from Encoder's output for a set of four reference points (blue dot). We can see the encoder is able to separate individual instance. (Image source: gucci®)

separated into Q, K and V by respective weight. It receives V and K from the Encoder's output as input. Q receives the output from the masked multi-head attention sublayer. As Q receives the output from Decoder's first attention block, the attention values represent the importance given to the Decoder's input based on the Encoder's output. In other words, the Decoder predicts the next feature by looking at the Encoder output and self-attending to its own output.

Why masked? Due to Encoder must know all components in overall. Then self-attention is able to perform the query with all feature against others. But at the time of decoding, when trying to predict the next feature. It should not know what are the features presented after and trying to predict.

3.2. DETR Architecture

DETR Transformer architecture has 3 parts: the backbone CNN as feature extractor, Transformer to learn features dependently in a parallel and comprehensive way by *learned* queries and a traditional FFN to detect label and corresponding bounding box. The figure of summary of DETR architecture can be found in figure 6 below.

3.2.1. Backbone

Backbone is a term used in DeepLab models/papers to refer to the feature extractor network. This feature extractor is used to encode the network's input into a certain feature representation. Here DETR uses the common CNN backbone for feature extraction. Its purpose is learning a 2D representation of an input image. The model flattens it and supplements it with a positional encoding (as it expected input) before passing it into a Transformer Encoder. In this article, the ResNet from TORCHVISION pretrained model are used by transfer learning.

For specificality, given the initial images $x_{image} \in R^{3\times H_0\times W_0}$. Backbone CNN generates to the high level feature maps $f \in R^{C\times H\times W}$, where $C = 2048$; H, $W = \frac{H_0}{32}, \frac{W_0}{32}$. [1]

3.2.2 The Encoder

Image representation

Next, the Encoder employs the 1×1 convolutional layers to reduces the channel dimension of the high-level activation map f from C to a smaller dimension d, the result in this step is $f \in R^{d\times H\times W}$. Then they collapse spatial dimensions H,W into one dimension $H\times W$ (understandable as flattening), resulting $d\times HW$ feature map. Then sum up with corresponding positional embedding result and pass to Encoder. The embedded feature map $\in R^{d\times HW}$ is considered to be a sequence, which has d feature unit $\in R^{HW}$. In figure 5, we introduce the self-attention maps that comes from the last Encoder layer of a pre-trained model, for four given reference points (given self-attention is very large matrix).

Transformer extracts the embedded feature with the help of Q, K and V as mentioned above. Not only considering all the correlation of a single (point of) feature against others by the multi-head attention. We

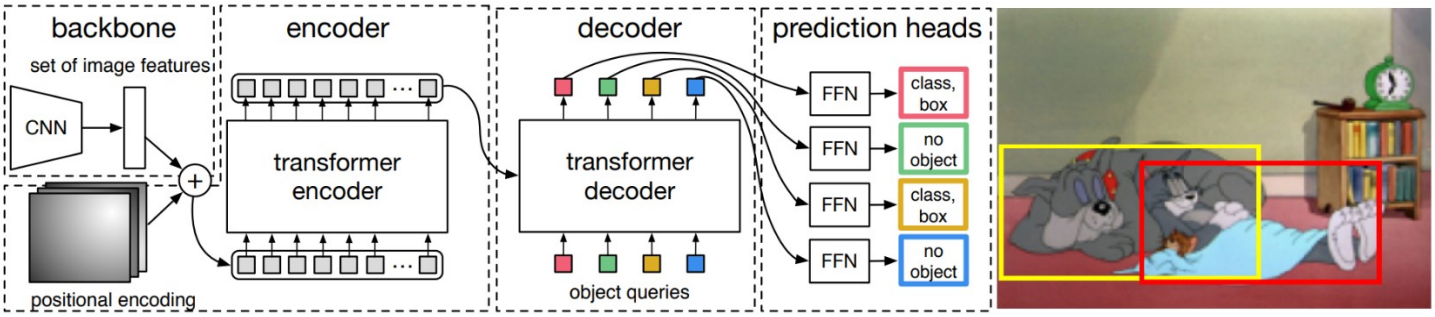


Figure 6: ***DETR architecture summary:*** *First, DETR uses CNN backbone to extract feature of 2D images. Then flatten it into feature vectors and sum up with positional encoding to make it digestible with Transformer's Encoder. The Encoder's output is considered to be mapped with the input data. Next, we transmit the output into the Decoder, with that it takes the object queries, which learned to pay attention properly and independently with the feature's input. Each of the result of these queries is passes into "prediction heads" FFN that predicts either a dectection (class + bounding box) "object" or "not object".* [1]

can also figure out the feature belongs to a single object (or nothing) with other objects located very close or even if the object belongs to the same class. That is why Transformer are so powerful at generalizing the feature.

3.2.3. The Decoder

Simply follows the standard architecture of the transformer, transforming N embeddings of size d using multi-headed, self- and Encoder-Decoder attention mechanisms. The Decoder expects the output K, V (embedded features) from Encoder, and its input is introduced as *Object queries.*

Object queries

Basically, these are N *learned* vectors of queries, each vector are randomly initiation that are initialized and transformed and combined with K and V (by attention mechanism) through out the Decoder to query the encoded feature (by paying attention to a certain R.O.I of a feature). It self-improves *independently* (with feature input) by optimization. Its outputs are the queries for output bounding boxes and class labels. The visual version of the object queries that is the output of the Decoder is called the *box predictions*. The box format is written as $[x_{center}, y_{center}, w, h]$ form.

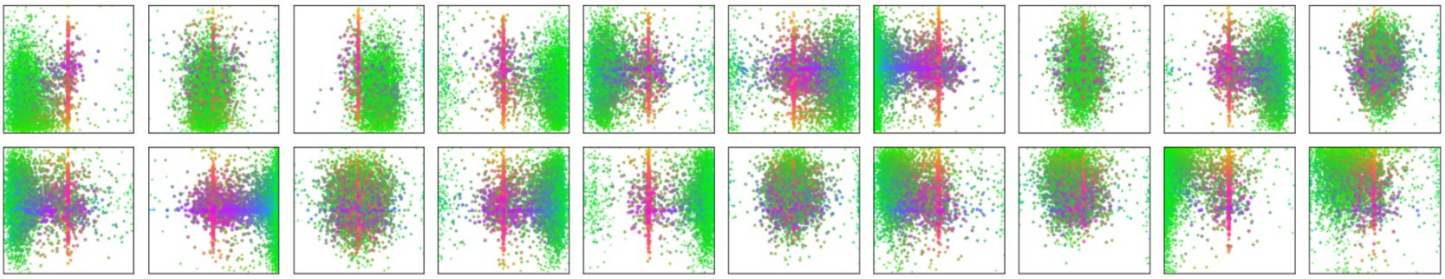


*Figure 7: There are **box predictions** on all images from COCO 2017 val set for 20 out of total N = 100 prediction query in DETR Decoder. Each box prediction is represented as a point with the coordinates of its center in the 1-by-1 square normalized by each image size. The points are color-coded so that green color corresponds to small boxes, red to large horizontal boxes and blue to large vertical boxes. Each box learns to specialize on certain areas and box sizes with several operating modes. DETR learns different speciali-zation for each query box. We observe that each box has several modes of operation focusing on different areas and box sizes.* [1]

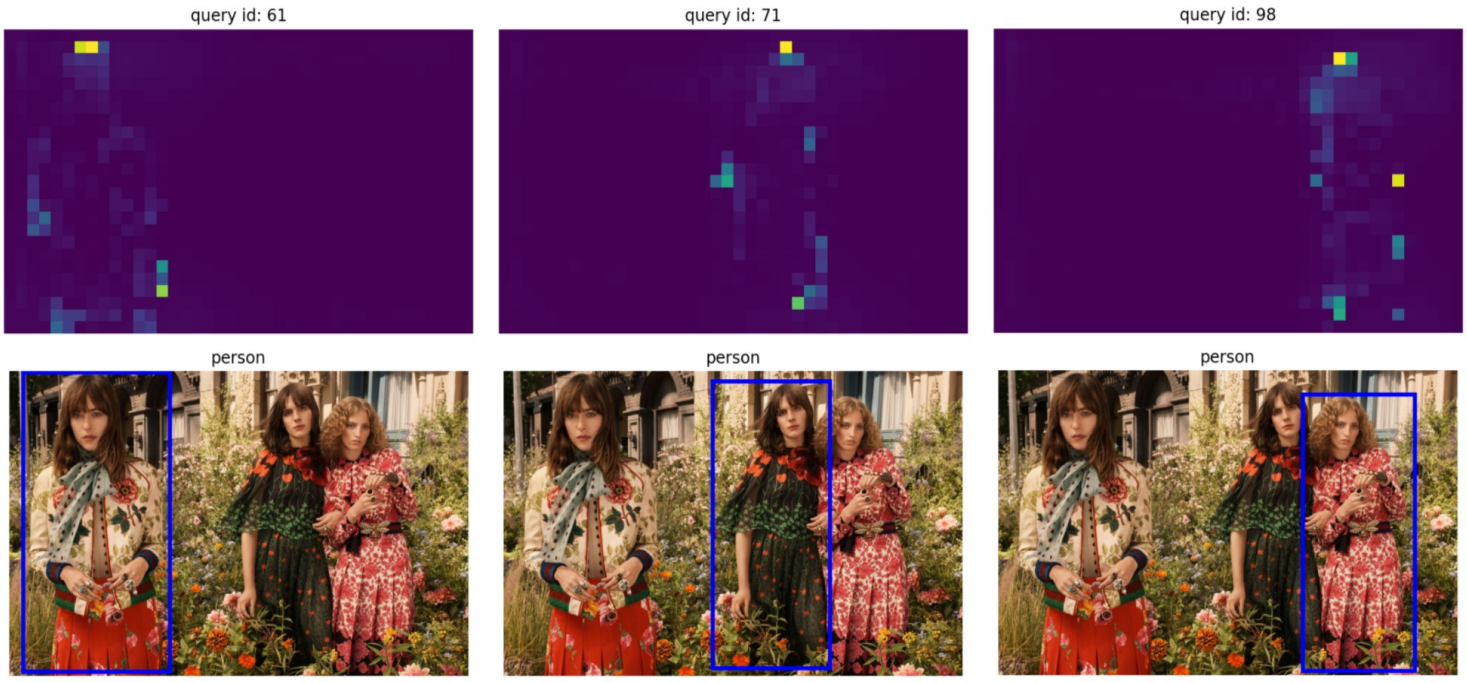


Figure 8: 100 queries stare into an image and predict, then three queries that accordance to the image's multi-head attention release specific bounding box and class.

3.2.4. Prediction heads FFN

The final prediction is computed by a traditional neural network, with contains a 3-layer perceptron with ReLU activation function and hidden dimension *d*, and a linear projection layer. Receive the output queries from Decoder. FFN predicts the normalized center coordinates, height and width of the bounding box with respect to the input image, and the linear layer predicts the class label using a softmax function. Since we predict a fixed-size set of *N* bounding boxes, where *N* is usually much larger than the actual number of objects of interest in an image, thus an additional special class label ∅ (generated by the network) is used to represent that no object is detected within a query.

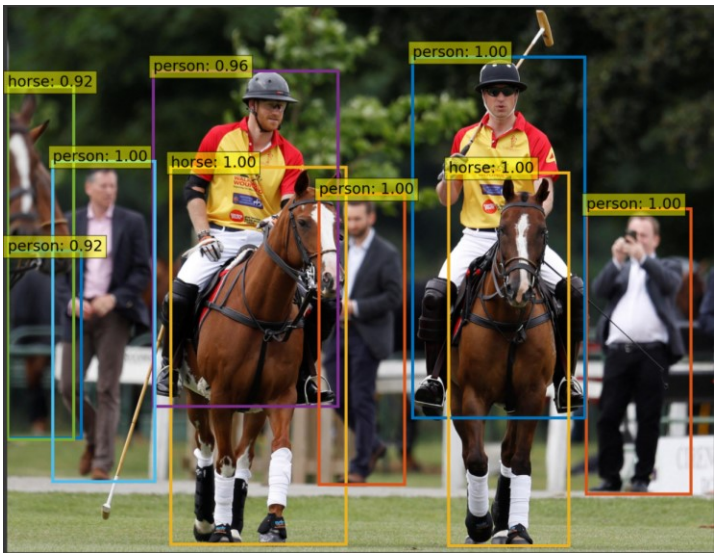


Figure 9: Example of DETR object detection. By self-based lost mechanism, the prediction heads predict class label and bounding box without the need of NMS. The self-based loss will be introduced in detail in the next part. (Image source: goldengatefields.com)

4. Loss computation and training

4.1. Bipartite matching loss

As clarified, DETR network builds and trains the queries through the Decoder in order to predicts a set of *N* class label and the corresponding co-ordinates of the object with respect to the input image (with *N* > objects quantity). On the other hand, the ground truth (also is the set of class label) from dataset's annonation is compared with the prediction. Pair of sets are not compared one by one in order, and avoided detecting prominent features duplicately.

The loss is computed by assign the prediction to one ground truth which lies in the closest proximity of the prediction and the left out predictions are mapped to "no-object" class. Given y as the ground truth of objects and $\hat{y} = \{\hat{y}_i\}_{i=1}^{N}$. To find *bipartite matching* of it and the *unique* suitable candidate, we search for a permutation of *N* elements with the lowest cost. In the article, the cost is descripted as pair-wise matching cost $\mathcal{L}_{match}(y_i, \hat{y}_{\sigma_i})$, the optimal computation of the assignment is performed by Hungarian algorithm.

The matching loss for y and $\hat{y}$ is the linear combination of the standard bipartite matching loss (for evaluating class label) and the specific bounding box loss.

$$\hat{\sigma} = \arg\min_{\sigma \in \mathfrak{S}_N} \sum_{i}^{N} \mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)}),$$

bipartite matching loss

$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \left[-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}}(i)) \right]$$

Hungarian matching loss

Hungarian algorithm

Hungarian algorithm is considered to the optimal algorithm for the assignment permutation problem. Given the set of *N* ground truths y and set of *N* predictions $\hat{y}$ (the loss computation of each element of both set has already performed). Any ground truth can be assigned with any prediction, incurring some loss that may vary depending on the ground truth-prediction assignment. It is required to perform all tasks by assigning exactly one ground truth to each prediction in such a way that the total cost of the assignment is minimized. The detail of process is descripted in [7]

Bounding box loss

The bounding box loss combines two components: IoU loss and l_1 loss for ground truth bounding box b and predicted bounding box $\hat{b}$:

$$\mathcal{L}_{\text{box}}(b_{\sigma(i)}, \hat{b}_i) = \lambda_{\text{iou}} \mathcal{L}_{\text{iou}}(b_{\sigma(i)}, \hat{b}_i) + \lambda_{\text{L1}} ||b_{\sigma(i)} - \hat{b}_i||_1$$

Where two real numbers λ_{iou} and λ_{L1} are hyperparams. The $\mathcal{L}_{iou}$ is the *generalized IoU (Giou)* loss. Which combines IoU and the penalty term.

$$\mathcal{L}_{\text{Giou}}(b_{\sigma(i)}, \hat{b}_i) = 1 - \text{iou}(b_{\sigma(i)}, \hat{b}_i) + \frac{|C - b_{\sigma(i)} \cup \hat{b}_i|}{|C|}$$

Where C is the smallest box covering b and $\hat{b}$, the smaller C in the better case, even if no intersection (ie $IoU = 0$). Thus due to the introduction of penalty term, the predicted box will move towards the target box in non-overlapping cases. The visualization and detail can be found at [8].

4.2. Training

Backbone CNN

TORCHVISION pretrained ResNet-50 is applied for fearture extraction, removing classtification layer. The backbone are retrained by transfer learning in order to adapt our model. In the warm-up session, the batch norm weights and mini-batch statistics are *frozen*. Then finetune the whole backbone which learning rate 10^{-5}.

Transformer

All Transformer's weights are initialized with Xavier initialization. DETR is trained by Adam optimizer with improved by weight decay. The gradient clipping are also applied in order to avoid exploding gradient (in detal, this technique scales the gradient with its Euclidean norm if it larger than given threshold). The Transformer is trained with the learning rate of 10^{-4} and dropout of 0.1.

AdamW optimizer

Adam is the widely used optimization algorithm that combines features of: *RMSprop* and *SGD* with *momentum* into a fairly robust update rule. One of principal components of the Adam is the couple of quantities v_t and s_t, storing an exponentially decaying average of past both gradients (also known as mean) and squared gradients (also known as uncentered variances).

$$v_t \leftarrow \beta_1 + (1 - \beta_1) g_t \qquad s_t \leftarrow \beta_2 + (1 - \beta_2) g_t^2$$

Where g_t is the gradient of thetas (contains weight: W and bias: b) of the loss function. β_1 and β_2 are nonnegative weighting params. Common choices of them are 0.9 and 0.99 respectively. By these choices above, the uncentered variance s_t moves much more slowly than the means. That makes it is considered as a heavy friction ball and not affected much by small changes of gradient. The update equation is written below:

$$g_t' = \frac{\eta \hat{v}_t}{\sqrt{\hat{s}_t} + \varepsilon} + w x_t$$

Where $\hat{v}_t$ and $\hat{s}_t$ are the bias correction of v_t and s_t adjust for a slow startup when estimating these two quantities. AdamW is performed by the addition of wx_t, in which w is *decoupled weight decay* for x_t. Finally a very small number ε is added for avoiding zero-division. [9]

Finally we have the update of thetas below:

$$x_{t+1} \leftarrow x_t - g_t'$$

5. DETR for panoptic segmentation

DETR + panoptic head

As introduced in the first part, panoptic segmentation (PS) is the task which are attracted a lot of attention from the computer vision community by the comprehensiveness it offers. DETR can be extended by adding a mask head on top of the decoder outputs. This session is the objective of our project. In this session, we perform pairing DETR with a head which can be used to produce PS by treating the *stuff* and *thing* classes in a unified way. Received the input as output from prediction heads (embedded d-dimension vectors contains bounding boxes and corresponding class labels of each object for results of N queries), it generates the PS version of image.

The mask head performes three processes: first, it takes input for each objects and computes multi-head (with M heads) attention *softmax* scores of this embedding over the output of Encoder, generating M attention heat maps per object in a small resolution. The output is $N \times M$ individual small maps.

Second, to make the final prediction and increase the resolution, an *FPN-style CNN* architecture is used. The output is N mask logits corresponding to N queries. Mask logits is also known as the last convolution layer, weights of this layer can render heat maps as binary form. Result can be found in figure 14, from the test image: figure 11.

Finally, the Pixel-wise argmax layer merges all N heat maps by using an argmax over the mask scores at

each pixel, and assign the corresponding categories to the resulting masks. This procedure guarantees that the final masks have no overlaps. The last result of the process is segmented image: figure 15.

FPN - style CNN

Receiving the concatenation of *M* multi-head attention head map. FPN generates higher resolution and extractes mask for image segmentation. Similar to original FPN, it has two pyramic-like pathway: top-down and bottom-up.

The bottom-up pathway is the feedforward computation of the backbone convolution network, each pyramid level has a convolution module and a group norm layer. Through each layer, the spatial resolution decreases, with more high-level structures detected, meaning the semantic value for each layer increases. The output is reduced channel by 1×1 convolution and can be used as the reference set of feature maps and transmit to the top-down pathway.

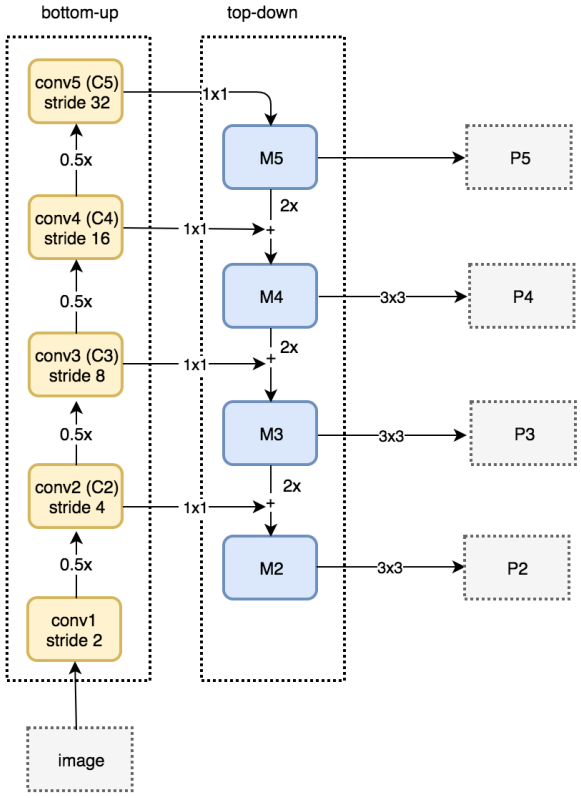


Figure 10: FPN architectre with the simple convolution head as bottom-up. Different from the original, In our architecture, the conv5 layer uses stride 64. [10]

As go down the top-down pathway, we upsample (deconvolute) the previous layer using nearest neighbours upsampling. Then merge with 1×1 convoluted corresponding feature map in the bottom-up pathway.

Finally, a 3×3 convolution is appended on each merged map to generate the final feature map. This final set of feature maps is called {P2, P3, P4, P5}. In the article, the final output is called mask logits, which are defined as the result of each query. Specifically, there are the *N×H/4×H/4* binary maps, each contains segmented object and individual confidence score.

Training

Three processes are trained *independenly*. It has two ways to train the mask head: either jointly, or train the Transformer for boxes and labels first, then *freeze* all the Transformer weights and train the mask. In the pretrained model, they chose the second way.

6. Pretrained model performance

Model	Backbone	PQ	SQ	RQ	PQ^{th}	SQ^{th}	RQ^{th}	PQ^{st}	SQ^{st}	RQ^{st}	AP
PanopticFPN++	R50	42.4	79.3	51.6	49.2	82.4	58.8	32.3	74.8	40.6	37.7
UPSnet	R50	42.5	78.0	52.5	48.6	79.4	59.6	33.4	75.9	41.7	34.3
UPSnet-M	R50	43.0	79.1	52.8	48.9	79.7	59.7	34.1	78.2	42.3	34.3
PanopticFPN++	R101	44.1	79.5	53.3	**51.0**	**83.2**	60.6	33.6	74.0	42.1	**39.7**
DETR	R50	43.4	79.3	53.8	48.2	79.8	59.5	36.3	78.5	45.3	31.1
DETR-DC5	R50	44.6	79.8	55.0	49.4	80.5	60.6	**37.3**	**78.7**	**46.5**	31.9
DETR-R101	R101	**45.1**	**79.9**	**55.5**	50.5	80.9	**61.7**	37.0	78.5	46.0	33.0

Table 1: Comparison DETR + backbone ResNet model with S.O.T.A PanopticFPN++, UPSNet. [1]

In the article [1], they compared their panoptic segmentation method with several established methods that treat things and stuff differently. They reported the Panoptic Quality (PQ) and the break-down on things (PQ^{th}) and stuff (PQ^{st}). They also reported the mask AP (computed on the things classes), before any panoptic post-treatment (before taking the pixel-wise argmax). They showed that DETR outperforms published

7. Conclusion

We introduced Detection Transformer, for the object detection and panoptic segmentation by Transformer and bripate matching loss. Get feature extracted input with the help of ResNet backbone, trained by AdamW optimization algorithm. With the most important role belongs to the Transformer. Which has its strong per-

formance and automatic thanks to attention, paralelly training and bipartite matching mechanism.

Despite the advantage of attention mechanism, there are some limitations need to solve in the future. For determinate the self-attention of 1 token (in this article we use the word *feature* as a token unit), we need to compute the rest of all *N-1* tokens for local attention (with *N* is all the token of input sequence), this requires quadratic computation, as well as quadratic memory size. It can be consider as an Brute-force approach attention.

References

[1] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier,Alexander Kirillov,Sergey Zagoruyko, "End-to-End Object Detection with Transformers," *arXiv:2005.12872,* 2020.

[2] L. Yao, A. Chyau, "A Unified Neural Network for Panoptic Segmentation," *In: CGF:13852,* 2019.

[3] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, Piotr Dollár, "Panoptic Segmentation," *arXiv:1801.00868,* 2018.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, "Attention Is All You Need," *arXiv:1706.03762,* 2017.

[5] A. Kazemnejad, "Amirhossein Kazemnejad's Blog," 20 9 2019. [Online]. Available: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/. [Accessed 12 11 2020].

[6] J. Alammar, "The Illustrated Transformer," 27 6 2018. [Online]. Available: http://jalammar.github.io/illustrated-transformer/. [Accessed 1 12 2020].

[7] R. Goyal, "GreekforGreeks," 4 3 2020. [Online]. Available: https://www.geeksforgeeks.org/hungarian-algorithm-assignment-problem-set-1-introduction/. [Accessed 16 12 2020].

[8] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, Silvio Savarese, "Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression," *arXiv:1902.09630,* 2019.

[9] Anonymous authors, "FIXING WEIGHT DECAY REGULARIZATION IN ADAM," *In: ICLR,* 2018.

[10] Vaishak V.Kumar, "Panoptic Segmentation with UPSNet," towards data science, 21 7 2019. [Online]. Available: https://towardsdatascience.com/panoptic-segmentation-with-upsnet-12ecd871b2a3. [Accessed 12 23 2020].

[11] E. David, S. Madec, P. Sadeghi-Tehran, H. Aasen, B. Zheng, S. Liu, N. Kirchgessner, G. Ishikawa, K. Nagasawa, M.A. Badhon, C. Pozniak, B. de Solan, A. Hund, S.C. Chapman,

F. Baret, I. Stavness, W. Guo, "Global Wheat Head Detection (GWHD) dataset: a large and diverse dataset of high resolution RGB labelled images to develop and benchmark wheat head detection methods," *arXiv:2005.02162,* 2020.

Our experiments

1. Implement the COCO pretrained DETR

1.1. Object detection problem

The pretrained model that the authors provides is trained from COCO 2017 dataset, the training and validation images and annotations from *https://cocodataset.org/*. To train the model, they use a single node with 8 NVIDIA Tesla V100 card, it takes 6 days for 300 epochs. The image is augmentated by horizontal flips, scales and crops.

The pretrained DETR can distinguish 80 different classes of objects. For using the model, we have to rescale the image and normalized first. Above is the image we use to test its abilities.

Figure 11: Kanye West and his wife - Kim K (source: https://www.eonline.com/)

And the result of predicted bounding boxes and labels is shown below, we just keep only predictions has 0.9+ confidence (the confidence score is calculated by softmax regression for all queries).

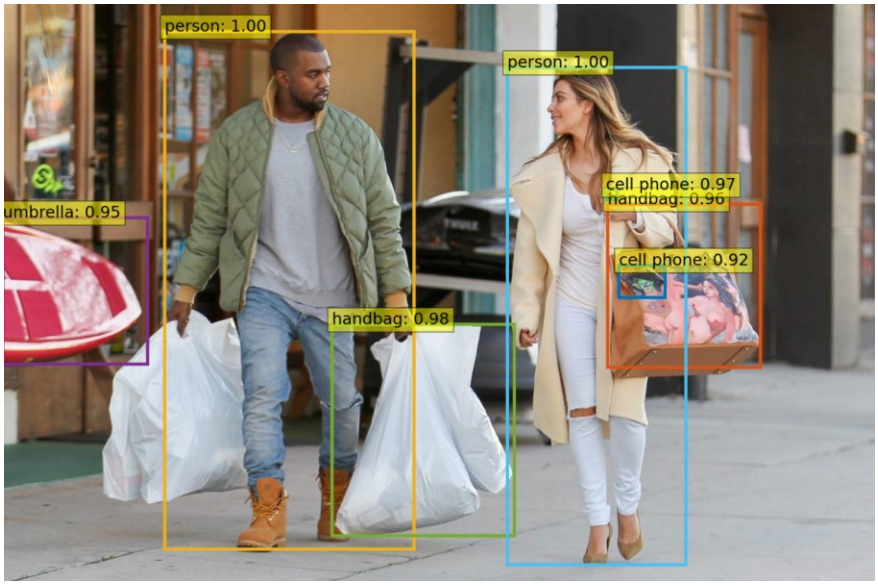


Figure 12: Result of predicted boxes and class labels.

As mentioned above, object queries have opportunity to "look" at the picture and "ask" the multi-head attention. Each query has its own perspectives. this make it notice various properties in the image. Below is the set of queries which have returned predictions. The shape of query maps is similar to the shape of the convolution feature maps.

Figure 13: The first row are query maps, the second one are bounding boxes respect to queries above.

1.2. Panoptic segmenation problem

The training process is recommend in two stages. First, train DETR to detect bounding box and then train panoptic head for segmentation, the first process is the same as the description in 6.1.1. Then they *freeze* pretrained DETR and train panoptic head with 25 epochs.

Figure 14: The output of FPN: mask logits

Figure 15: Panoptic segmenated image, each object has individual color.

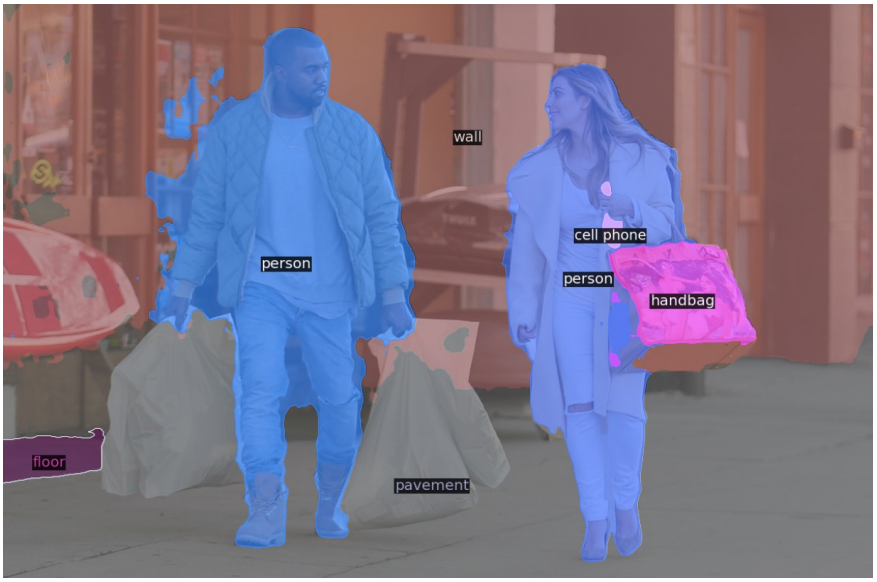


Figure 16: The input image is intergrated with segmenated image, each class of objects has the same color.

Its result are beyond our expectation, DETR segment neatly in right predictions. Beside that there are 2 wrong predictions. It looks at the plastic dust bags that is mistaken with pavement, this is quite effortlessly for explanation, the plastic dust bags are not included in dataset. Most notably is the second case, it mistake the woman's hand for a cell phone, though it does not look like a phone at all. In our opinion, this confusion happens due to the local attention mechanism. The positional encoding by simple math function such as sine and cosine may cause the model rigidly focus and learn on pixel's neighbours. Thus it is in trouble when detecting a small object.

2. DETR model for Wheat detection problem

Initially, we planned to train DETR model with UIT logo dataset and use them for AI challenge competition, as well as a result for our project. But we have failed! Due to the dataset is too small to make model works properly. So we decide to implement DETR model with Global wheat dataset. This is the suitable dataset for training in limited time and modest computer configuration, which both small (less than 1GB) and have many annotations (about 150K).

2.1. Global wheat detection dataset

Global WHEAT Dataset is the first large-scale dataset for wheat head detection from field optical images. It included a very large range of cultivars from differents continents. These images are used to estimate the density and size of wheat heads in different varieties. The Global wheat dataset is led by nine research institutes from seven countries around the world: the University of Tokyo, Institut national de recherche pour l'agriculture, l'alimentation et l'environnement, Arvalis, ETHZ, University of Saskatchewan, University of Queensland, Nanjing Agricultural University, and Rothamsted Research. Specifically, it includes 3422 images, 150K bounding boxes, each image has about 40 bounding boxes, each box respects to each wheat head. In additon we have 10 images in the test set, with no bounding box.

Our work is detecting wheat heads exactly from outdoor images of wheat plants, including wheat datasets from around the world. By our result, Farmers can use their data to assess health and maturity when making management decisions in their fields.

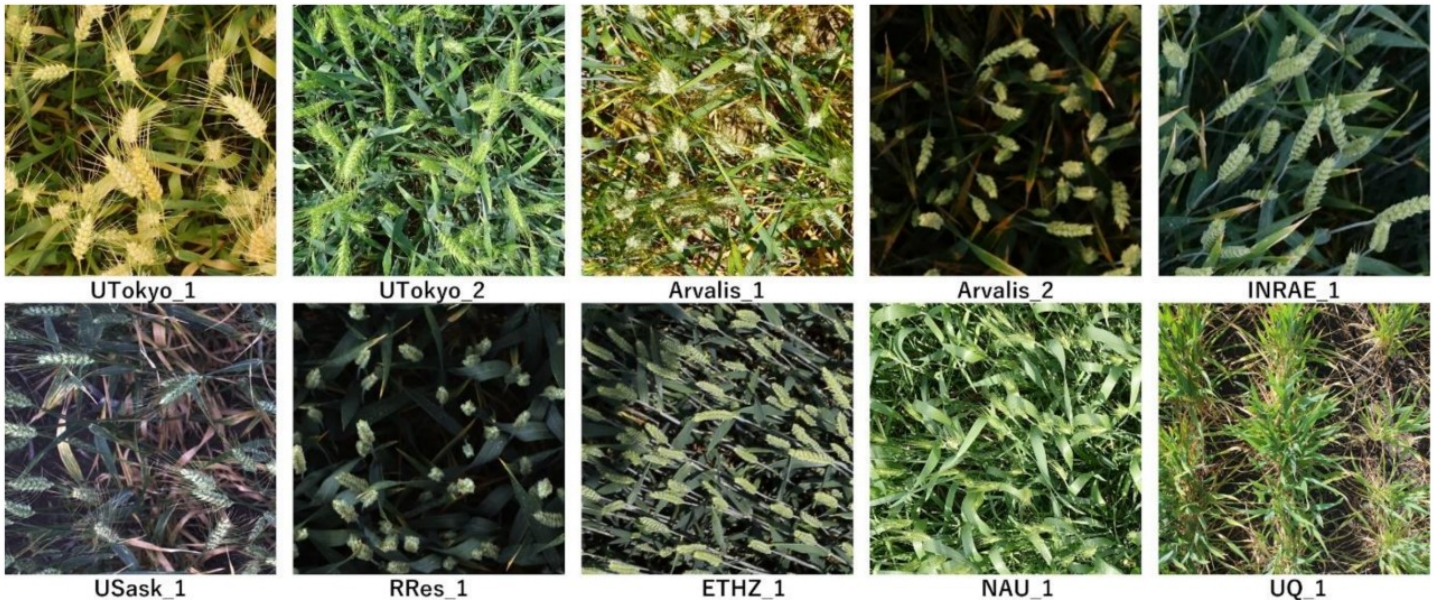


Figure 17: Image of dataset and its source (research institutes)

This is really hard problem, due to the wheats has a similar shape and color to the leaves and stems. Furthermore, each variety has different types of wheat, leaf and stem. The environment factors must also be considered, images are taken in various conditions: day, night, flash light, weak light, some images are stained to yellow or green. The bounding boxes in the most images are messy, overlapping and variety in size.

2.2. Training

Machine

We train the model on a basic kernel of Kaggle. Which has one GPU Tesla P-100-PCIE 16GB, CPU 2-core Intel Xeon and 13GB RAM.

Experiment

We use stratified K-fold cross validation for splitting dataset and training model. This method is useful for small dataset. We devide the data to 4 folds. We also apply image agumentation. Specifically, we resize the image to 512×512 .Also apply HSV shift, adjust brightness and contrast, horizontal and vertical flip and cutout randomly to each image of training set.

We train DETR by reuse the pretrained DETR-backbone ResNet-101.With AdamW algorithm for optimization, setting learning rate at 2.10^{-5}. The number of batch size is 12 and 10 epochs for each fold. Totally we have 40 epochs, training shedule takes around 2 hours.

2.3. Result

The best model we found has 0.852 loss on training set and 0.73 loss on validation set. These two indicators are quite modest.

We try using the model to predict some images in the test set. It can predict well in basic cases:

In the harder case, like with this image, the model is confused, it is difficult for people to see, let alone machine.