COL828 Assignment 2
Report

Eshan Jain

1. Introduction

This project focuses on implementing the DETR (DEtection TRansformer) model for detecting and classifying different types of bone fractures in X-ray images.

I conducted experiments to train DETR from scratch and fine-tune a pre-trained DETR model on the provided dataset, evaluating the performance using mean Average Precision (mAP), Precision, and Recall metrics.

---

2. Dataset Details

The dataset provided has the following characteristics:

- **Train/Validation/Test Split:**
  - Train: 3361 images
  - Validation: 348 images
  - Test: 169 images
- **Annotation Format:** COCO
  - **Images:** Contains metadata like `id`, `file_name`, `height`, and `width`.
  - **Annotations:** Includes `id`, `image_id`, `category_id`, and `bbox` ([x, y, width, height]).

---

3. Experiment [E1]: DETR from Scratch

3.1 Design Choices

- **Backbone:** Pre-trained ResNet-50 was used as the CNN encoder to extract image features.
- **Transformer Architecture:**
    - **Number of Layers:** 3 Transformer blocks in both encoder and decoder.
    - **Self-Attention and Cross-Attention:** Implemented standard multi-head attention with 8 heads.
    - **Feedforward Networks:** Each layer contains a two-layer MLP with a hidden dimension of 2048.
- **Positional Encoding:** Implemented sinusoidal positional encoding to retain spatial information in transformer operations.
- **Object Queries:** Used 4 learnable object queries for decoder predictions. This number was chosen based on the dataset's object density (Max 4 objects per image).
- **Loss Function:**
    - Implemented Hungarian matching loss for classification and bounding box regression.
    - **Components:**
        - Classification Loss: Cross-Entropy
        - Bounding Box Loss: L1 Loss
        - Generalized IoU Loss.

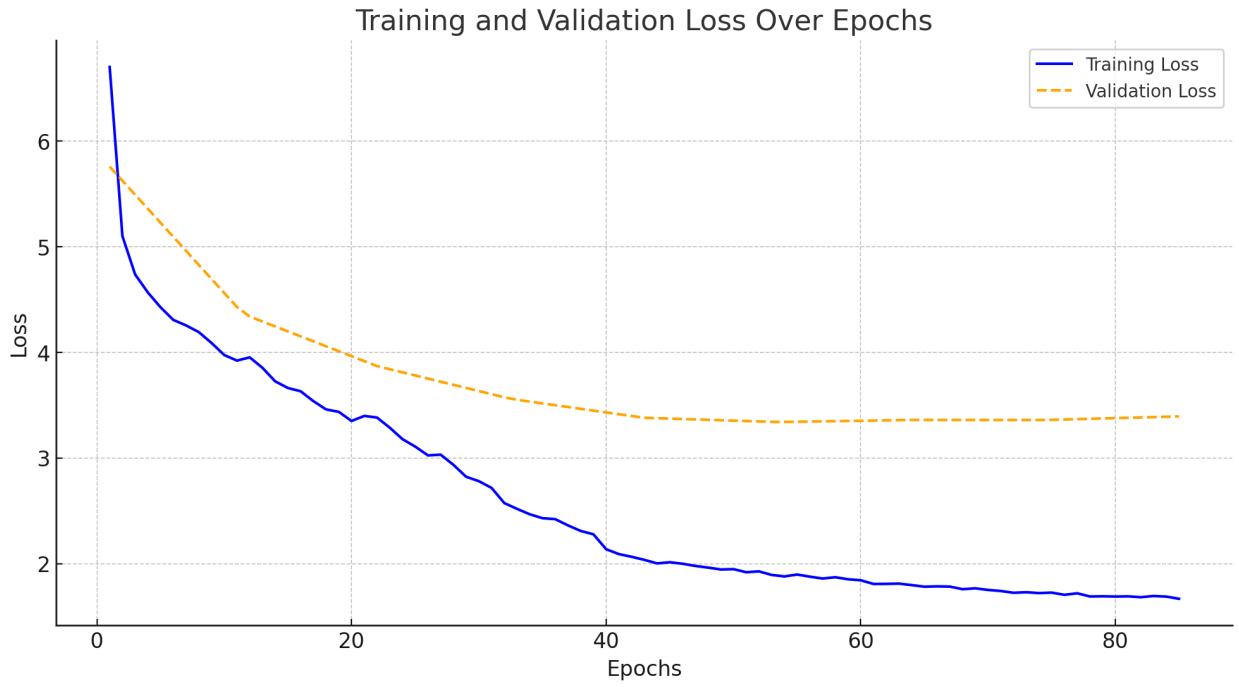
3.2 Training Strategy

- **Augmentations:** Applied all data augmentations like random horizontal flips, random crop, and random resize, as per the official DETR Paper.
- **Optimizer:** AdamW with a learning rate of `1e-4`
- **Batch Size:** 4 (due to gpu constraints)
- **Epochs:** 100

3.3 Experiments Conducted for Hyperparameter Tuning

I first resized all images to 800*800, and without applying any data augmentations, I trained the model from scratch till around 100 epochs. I had set the transformer encoder and decoder layers to 6, dropout = 0.1, and weight_decay=1e-4 (as in the DETR paper). I observed that around 20-30 epochs, the model started getting overfitted on the dataset, since the val loss started increasing while the training loss kept on decreasing.

To prevent overfitting, I implemented all the data augmentations as mentioned in the DETR paper, set the encoder and decoder layers to 3, increased dropout to 0.3, and weight_decay to 1e-3. This time the model trained somewhat better but still ended up getting overfitted around 50 epochs.



Additional Experiments Done:

I experimented with the number of queries by setting them to 4, 50 and 100, and I observed that the performance was the best in the case of 4, because the model had lesser number of queries to optimise, and also it took lesser time and GPU memory (so higher batch sizes could be trained) than for 100 queries.

Also, I experimented with different embedding dimensions like 256, 512, and tried changing the number of heads in the multi-head attention mechanism (4, 8), but these did not lead to any improvements in the performance.

I also experimented with fine-tuning the weighting of classification, bounding box regression (L1), and GIoU loss components. Tried out different Hungarian Matching costs for class, bbox, and GIoU (1:5:2, 1:10:5), but the results were more or less the same.

---

4. Experiment [E2]: Fine-Tuning Pre-trained DETR

4.1 Model Details

Used the pre-trained DETR model (`facebook/detr-resnet-50`) from Hugging Face's Transformers library. Fine-tuned the model on the dataset with minimal modifications to the original DETR implementation.

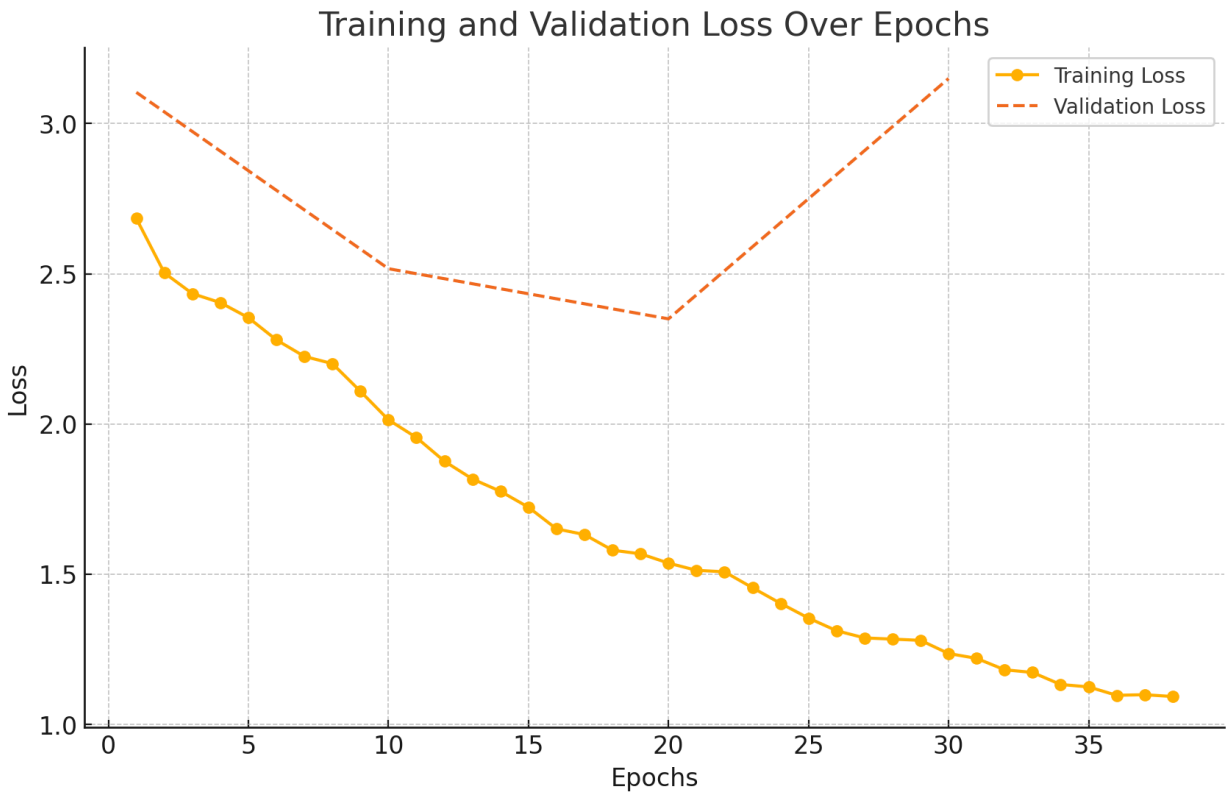
4.2 Training Strategy

- **Augmentations:** Same as Experiment [E1].
- **Optimizer:** AdamW with a learning rate of `1e-5`.
- **Batch Size:** 4
- **Epochs:** 100

4.3 Experiments Conducted for Hyperparameter Tuning

Similar to the scratch case, I had first finetuned the model without any data augmentations, and it led to overfitting in the model very quickly in around 10-20 epochs.

After implementing the data augmentations similar to the scratch case, I saw similar results in the sense that overfitting got delayed, and now it started at around 25 epochs.



Val loss measured every 10 epochs

Additional Experiments Done:

I tried freezing the ResNet backbone for the first 30 epochs and then in the subsequent epochs unfreeze it, but it did not lead to any improve in the performance.

Also, I tried playing around with resizing the input images in different configurations like 800 * 1333 (standard DETR), 1000*1000, and 800*800, but the results were not significantly different.

5. Results

5.1 Quantitative Results:

Experiments after Hyperparameter Tuning

On Test Dataset:

Experiment E1 (Scratch):

```
 IoU metric: bbox
  Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.023
  Average Precision  (AP) @[ IoU=0.50       | area=   all | maxDets=100 ] = 0.074
  Average Precision  (AP) @[ IoU=0.75       | area=   all | maxDets=100 ] = 0.002
  Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
  Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.000
  Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.023
  Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.077
  Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.165
  Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.165
  Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
  Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.000
  Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.166
 Evaluation complete.
```

Precision at IoU=0.5 is 0.07
Average Recall is 0.16
mAP is 0.02

Experiment E2 (Finetuned):

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.055
 Average Precision  (AP) @[ IoU=0.50       | area=   all | maxDets=100 ] = 0.156
 Average Precision  (AP) @[ IoU=0.75       | area=   all | maxDets=100 ] = 0.011
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.055
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.093
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.219
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.219
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.221
Evaluation complete.
```

Precision at IoU=0.5 is 0.15
Average Recall is 0.22
mAP is 0.05

On Val Dataset:

Experiment E1 (Scratch):

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.024
 Average Precision  (AP) @[ IoU=0.50       | area=   all | maxDets=100 ] = 0.087
 Average Precision  (AP) @[ IoU=0.75       | area=   all | maxDets=100 ] = 0.007
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.017
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.024
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.069
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.155
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.155
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.033
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.157
Evaluation complete.
```

Precision at IoU=0.5 is 0.08
Average Recall is 0.15
mAP is 0.02

Experiment E2 (Finetuned):

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.030
 Average Precision  (AP) @[ IoU=0.50       | area=   all | maxDets=100 ] = 0.097
 Average Precision  (AP) @[ IoU=0.75       | area=   all | maxDets=100 ] = 0.008
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.031
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.075
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.181
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.181
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.183
Evaluation complete.
```

Precision at IoU=0.5 is 0.097
Average Recall is 0.18
mAP is 0.03

So, as can be seen, the finetuned model performs slightly better than the one trained from scratch.

Experiments before Hyperparameter Tuning

On Val Dataset:

Experiment E1 (Scratch):

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.001
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.015
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.015
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.015
```

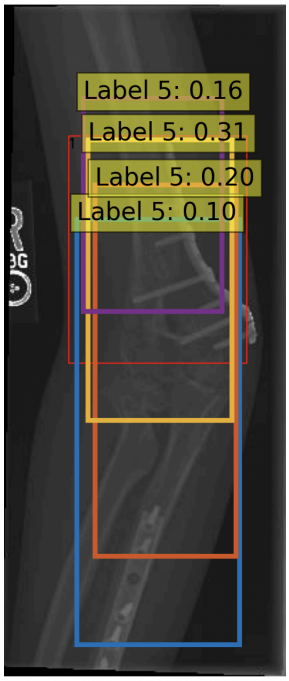
Experiment E2 (Finetuned):

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.007
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.030
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.008
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.015
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.071
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.071
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.072
Evaluation complete.
```

These results show that without hyperparameter tuning, the results were worse, and probably more so for the scratch model, as it is more affected by the tuning of hyperparameters.

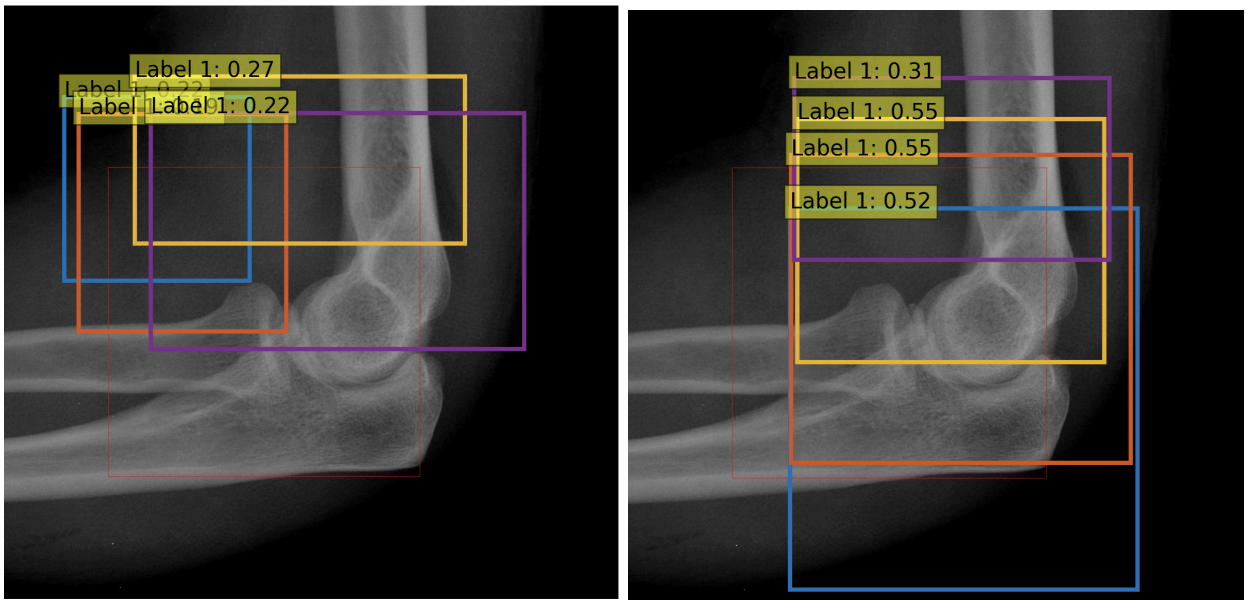
5.2 Qualitative Results

Image_Id:15, val dataset
category_id:1



The left one is the finetuned, and the right one is the scratch
The red box is the ground truth box, and the other ones are the predictions.

It can be observed the finetuned model has learned better localization ability as compared to the scratch one. The finetuned model also predicts the class label correctly, which is misclassified by the scratch model. However the scratch model also has lower confidence thresholds which shows that it does find these predictions to be good.



In this case, we observe that the scratch model is able to get better results and higher confidence score, maybe because the object is bigger so it is able to correctly classify as well as localise the object box.

Visualizations of bounding boxes and class predictions on test images showed the following:

- Experiment [E2] consistently produced tighter bounding boxes and fewer false positives compared to Experiment [E1].
- The model struggled with small or overlapping fractures, like the "Fingers Positive" category.

---

6. Analysis and Challenges

- **Object Queries:** Experimented with 4, 50, and 100 queries but found 4 to work best for this dataset.
- **Fine-Tuning Benefits:** Fine-tuning the pre-trained model provided a boost in mAP
- **Challenges:**

- Small objects were difficult for the model to detect.
- Due to GPU limitations and time constraints on colab and kaggle, could not train the model for large no of epochs.

---

7. Documentation

7.1 Installation Instructions

1. Download/Clone the repository:

```
git clone
https://github.com/eshan-292/{path_to_repository}

Also, download the pretrained models from the
following links:

Resnet_Backbone:
https://drive.google.com/file/d/1Hnxp4-hSJjsbo_pI0f
8Ran7x6Mwhf5DL/view?usp=sharing
Pretrained_scratch:https://drive.google.com/file/d/
1fp_OK4WQ3wpACCkr__YJg7INQ8WA86AH/view?usp=sharing
Pretrained_finetuned:https://drive.google.com/file/
d/13Kurxxl17bYfrucYYndbvDn74ialiT-W/view?usp=sharin
g
```

2. Install dependencies:

```
pip install -r requirements.txt
```

3. Download the dataset and place it in the `data/` directory:

```
data/
    BoneFractureData
        train/
        val/
        test/
        annotations/
```

7.2 Training and Evaluation

1. Train DETR from Scratch:

```
python train.py
```

2. Fine-Tune Pre-Trained DETR:

```
python finetune.py
```

3. Evaluate the Model:

```
python eval.py
```

3. Visualise Model Outputs:

```
python visualise.py
```

7.3 Directory Structure

```
2020CS50424/
├── data/
│   ├── coco.py
│   ├── transforms.py
│   ├── BoneFractureData
```

```
|        ├── train/
|        ├── val/
|        ├── test/
|        ├── annotations/
├── models/
|   ├── detr.py
|   ├── transformer.py
|   ├── loss.py
├── utils/
|   ├── box_ops.py
|   ├── utils.py
├── train.py
├── eval.py
├── finetune.py
├── visualise.py
├── outputs/
|   ├── best_model.pth
|   ├── best_finetuned_model.pth
├── requirements.txt
├── ReadMe
├── report.pdf
```

Data Directory

`data/coco.py`

- Contains the custom dataset class for loading and processing data in **COCO format**.

- Handles loading images, annotations, and applying data augmentations or preprocessing transformations.

`data/transforms.py`

- Implements various data augmentation techniques (e.g., resizing, flipping, normalization).
- Used to dynamically apply augmentations during training.

`data/train/`

- Directory containing training images.

`data/val/`

- Directory containing validation images.

`data/test/`

- Directory containing test images for evaluation.

`data/annotations/`

- COCO-format JSON files containing bounding box annotations for train/val/test splits.

Models Directory

`models/detr.py`

- Implements the **DETR architecture** (DEtection TRansformer), including:
  - CNN backbone (e.g., ResNet).
  - Transformer-based encoder-decoder architecture.
  - Prediction heads for classification and bounding box regression.
- Defines the forward pass and incorporates components like object queries and MLP.

`models/transformer.py`

- Defines the **Transformer architecture**, including:
  - Multi-head self-attention.
  - Cross-attention mechanisms.
  - Feedforward neural networks (FFNs).
- Includes positional encodings to retain spatial information.

`models/loss.py`

- Implements the **Hungarian Matching algorithm** for optimal matching between predictions and ground truth.
- Defines the **SetCriterion** loss function, which computes classification, bounding box regression (L1 loss), and GIoU loss.

Utils Directory

`utils/box_ops.py`

- Contains helper functions for operations on bounding boxes:
  - Conversions between formats (e.g., `cxcywh` ↔ `xyxy`).
  - IoU and GIoU calculations for evaluating overlaps between boxes.

`utils/utils.py`

- Provides utility functions for:
  - Data loading and collation.
  - Logging and tracking metrics.
  - General-purpose operations to simplify the training and evaluation pipelines.

Project Root Files

`train.py`

- Main script for training the DETR model from scratch.

- Handles:
    - Data loading.
    - Model initialization.
    - Training loop with dynamic augmentations.
    - Periodic evaluation and checkpoint saving.

`eval.py`

- Script for evaluating the trained model.
- Computes metrics such as **mAP (Mean Average Precision)** and **Recall**.

`finetune.py`

- Script for **fine-tuning a pretrained DETR model** (e.g., from Hugging Face).
- Adapts a generic pretrained model to the custom dataset with faster convergence.
- Shares similar functionality to `train.py`.

`visualise.py`

- Provides tools to visualize:
    - Model predictions, including bounding boxes and class labels.
    - Ground truth annotations for qualitative analysis.

Outputs Directory

`outputs/best_model.pth`

- Stores the **best checkpoint** from training the DETR model from scratch, based on validation performance.

`outputs/best_finetuned_model.pth`

- Stores the **best checkpoint** from fine-tuning the pretrained DETR model.

Additional Files

`requirements.txt`

- Lists the Python dependencies required for the project (e.g., PyTorch, torchvision, etc.).

`ReadMe`

- Project documentation, including:
    - An overview of the DETR implemetation and the dataset.
    - Instructions for setup, training, fine-tuning, and evaluation.

8. Conclusion

This project demonstrates the implementation of DETR from scratch and the fine-tuning of a pre-trained DETR model for bone fracture detection. While the pre-trained model outperformed the scratch model, both achieved reasonable mAP scores on the given dataset. Fine-tuning leveraged the robustness of pre-trained weights, significantly improving precision and recall.

Future Improvements

- Explore advanced data augmentation techniques to improve generalization.
- Experiment with larger transformer models (e.g., DETR-XL) for potentially better performance.
- Explore other DETR based techniques like Deformable DETR.