QR Detection

Data Generation :

- Colected 4000 images from the PASCAL VOC dataset and augmented them with random orientations and random cropping.
- Put QR with different sizes on each image at a random position and also stored the bounding box coordinates of QR in the image as (top, left, height, width)
- Data was divided into train, validation and test sets with 75%, 15% and 15% of original data, respectively.

Model:

- I've used a pre-trained ResNet18 model as a backbone and added an additional layer of 4 neurons corresponding to the bounding box coordinates of the image.
- ResNet18 is a relatively simple architecture that has been shown to achieve state-of-the-art performance on many image classification benchmarks.
- Since ReNet18 was trained on millions of images, it could identify important features from an image.
- I've trained all the layers of the model by initially starting with the pre-trained ResNet18 model weights.
- Since our QR dataset was not part of any publicly available datasets, we need a good model that can easily adapt to the downstream problem of QR detection.
- ResNet18 has shown that it can be adapted for a wide range of image classification tasks, even when the data is quite different from ImageNet.

Model parameters :

- **Loss Function:**
    - I have used Mean Squared Error as a Loss function since the outputs of the model are continuous values.

- **Optimizer :**
    - It turned out that SGD gave me better performance against other optimizers. It was also found to be faster and computationally efficient.
    - SGD can give better generalisation performance, meaning that the trained model performs well on new, unseen data. It could possibly be due to the learning rate being constant so that it will not overfit the model.
    - SGD is used with additional momentum, which ensures that weight updates of the model will not be noisy and contributes towards faster convergence.
    - Finetuned learning rate and found 0.001 as better initialisation, additionally Learning rate scheduler is used to scale down the learning rate after every five epochs so that the model takes sufficiently larger steps at the start and as

it approaches to optimum, it takes relatively smaller steps that ensure the model will not overlook the optimum.

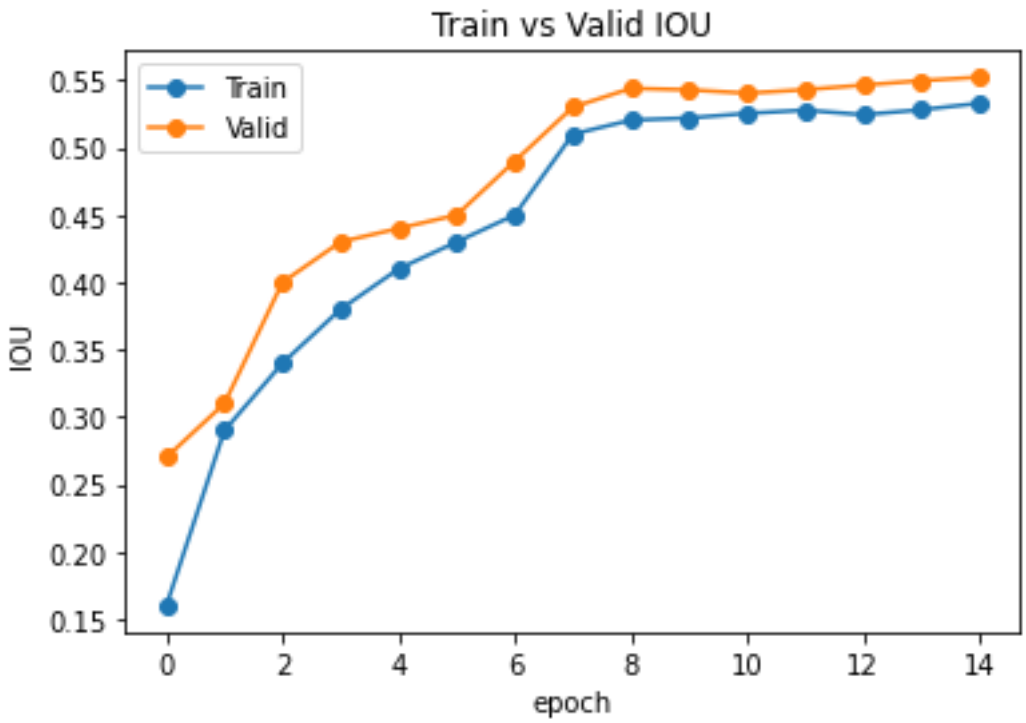
- **Batch size :**
  - I've chosen a relatively smaller batch size = 32 majorly because of memory constraints. Smaller batch sizes can also help model converge faster during training. This is because smaller batches allow model to update its parameters more frequently, which can lead to faster convergence.
  - Smaller batches can also help the model avoid overfitting to any particular subset of the data.

- **Epochs :**
  - Trained the model for 15 epochs and took the model that performed better on the validation dataset.
  - It was trained for relatively more epochs because the QR data was not part of ImageNet, and it takes some epochs to adapt the model to the problem of QR detection.

Results :

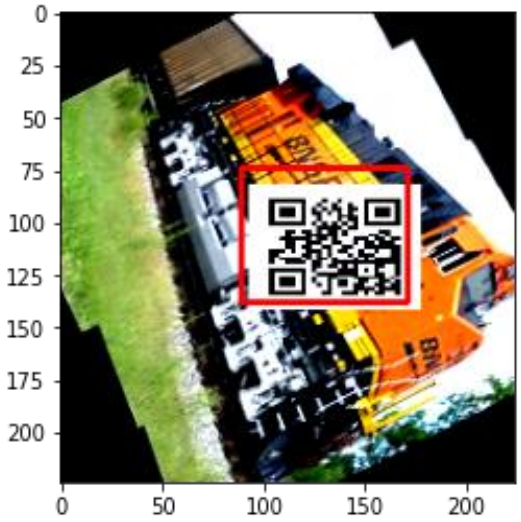
- Achieved IoU of *0.5329* and the training set and *0.546* on the test set.

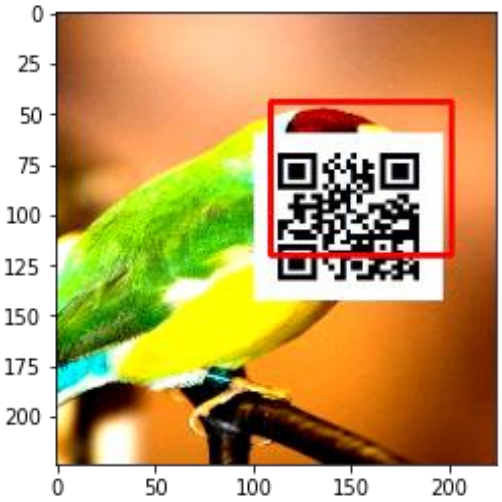
- **Plot of IoU achieved on train and validation sets after each epoch of training :**
  - We can observe that model is performing better as we progress.
  - It found that after 15 epochs, there is no much improvement in loss.

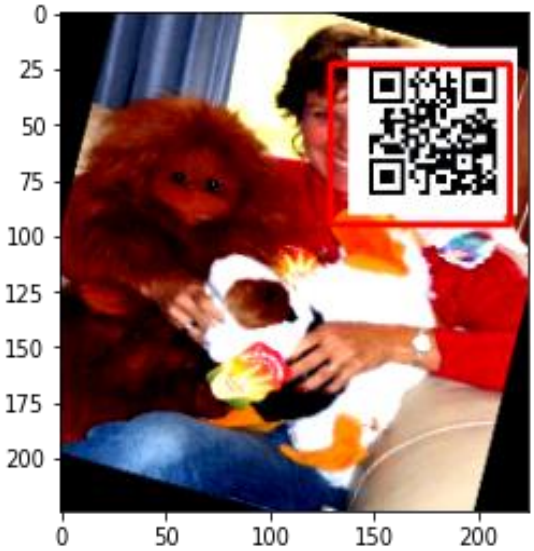


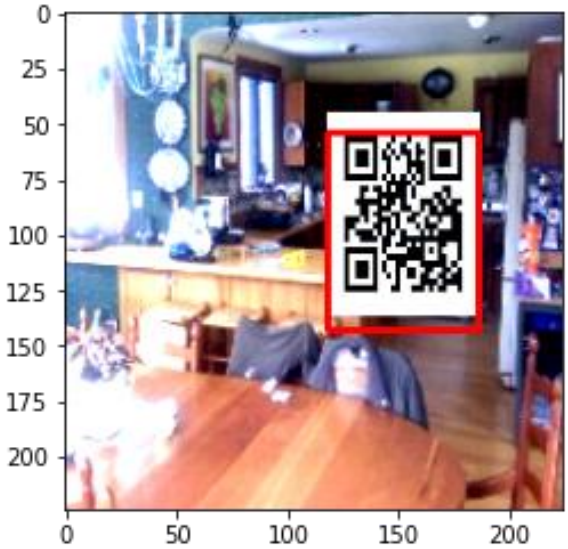
Testing on test dataset images :

- It was able to figure out the region of QR better.









Remarks :

- We can achieve even better performance if we choose the bigger pre-trained architecture and larger training dataset with different variations.
- Due to time and resource constraints, I could not try on using bigger models.

You can access the training data and trained model at the below link:

https://drive.google.com/drive/u/0/folders/1Kd4SVswmqYxT8hLH_SMoDyEgxMpQkQOl