50.039 Mini Project Report

Yap Wei Lok 1002394
Yufei Wu 1002192

March 19 2019

1 Training Metrics

Model Resnet18 initialized with pretrained ImageNet weights

Data Augmentation We used two different data augmentations, first one being random crop coupled with rotation (RandRotCrop) and the second one being five-crop (FiveCrop).

1. RandomRotation (30 degrees) + RandomResizedCrop (280px × 280px) + Normalization
2. Resize (300px at shortest end) + FiveCrop + Normalization

Normalization Metrics We calculated the 3-channel means and standard deviations of the dataset by ourselves. They are different for the two data augmentations we used. To approximate the means and standard deviations, we apply the data augmentation and load the images into a data loader. Then, we added up the means and standard deviations over batches and average them.

$$\text{means} = \begin{cases} 0.4601, 0.4381, 0.4048 & \text{if FiveCrop} \\ 0.4501, 0.4258, 0.3932 & \text{otherwise} \end{cases}$$

$$\text{standard deviations} = \begin{cases} 0.4501, 0.4258, 0.3932 & \text{if FiveCrop} \\ 0.2315, 0.2255, 0.2261 & \text{otherwise} \end{cases}$$

Optimizer We used Stochastic Gradient Descent (SGD) with a decaying learning rate, η, where $\eta_0 = 0.01$ and

$$\eta_{t+1} = \begin{cases} \eta_t * 0.01 & \text{if } t \pmod{10} = 0 \\ \eta_t & \text{otherwise} \end{cases}$$

Loss Function We used BCEWithLogitsLoss as our loss function. Since a picture may have several labels, we need to account for the prediction accuracy for each label. Therefore, BCEWithLogitsLoss was chosen to mirror 20 one-vs-all binary classifiers. BCELoss in PyTorch works as following.

$$BCELoss(x_i, y_i) = -w_i(y_i \log(x_i) + (1 - y_i) \log(1 - x_i))$$

We weighted the losses with the inverse of sample occurrences for each class multiplied by a constant ie. $w_i = K/n_i$ (where n_i is the number of sample occurrences for class i) to balance the loss since the class distribution is imbalanced. In our case, the constant K was chosen to be the minimum class frequency, therefore the class that appears the least will have its loss weighted by $w_i = 1$ while the rest will have $w_i \leq 1$.

Procedure The model was trained using each data augmentation for 40 epochs. After the 40 epochs, the Python script saves the best model weights, selected using validation loss. After which, the script computes the AP, accuracy and tail accuracy for the selected model weights. We chose 40 epochs because the model losses started decreasing at a negligible rate around that point.

2 Performance

Accuracy: The accuracy is calculated by dividing the number of true positive (TP) and true negative (TN) label predictions (using classification threshold of 0.5) with the total number of labels (sum of false and true positives and negatives) ie.

$$\text{Acc} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- For FiveCrop augmentation, the best model accuracy is: 0.9755
- For RandRotCrop augmentation, the best model accuracy is: 0.9657

In Figure 1, we can see that just by using the ImageNet pretrained weights ie. start of the training process, the model was able to achieve an accuracy > 0.95 and the 40 epochs of training only managed to increase it by approximately 0.015.

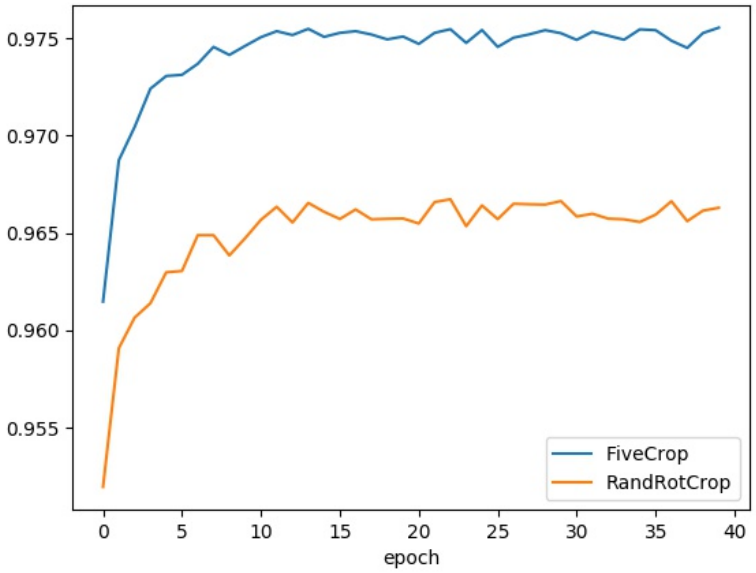

Figure 1: Accuracy over epochs for both data augmentations

The accuracy does not make sense in this case because there are a disproportionately large number of negative labels in the dataset. For reference, a model that only outputs negative predictions would have gotten a 92.83% accuracy on the validation set, since accuracy considers true negatives as well. Therefore some other better metric is needed.

Average Precision (AP) Measure: Instead of accuracy, we can use average precision. We used Scikit-Learn's average precision metrics to calculate the average precision for our model's output.

- For FiveCrop augmentation, the best model AP value is: 0.8821
- For RandRotCrop augmentation, the best model AP value is: 0.8079

Since Five Crop captures information in different parts of the image, it has a broader "view" of the image and is hence able to be trained to predict more labels correctly, which results in its AP value being higher. Thus, we use the FiveCrop model in the GUI prediction. We also demonstrate the predictions done by the FiveCrop model for each class in the GUI.

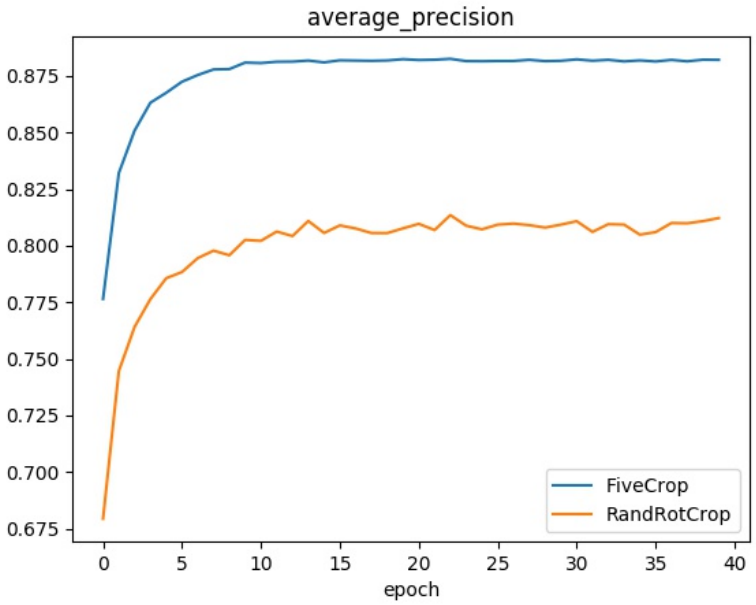

Figure 2: Average precision over epochs for both data augmentations

Tail Accuracy: The demonstrator (GUI) looks good on the top-50 images because images with higher prediction confidence and hence pass the classification threshold, are more likely to be correctly labelled. We can see this in effect in Figure 3 where the tail accuracy increases along with the classification threshold until they are both close to zero.

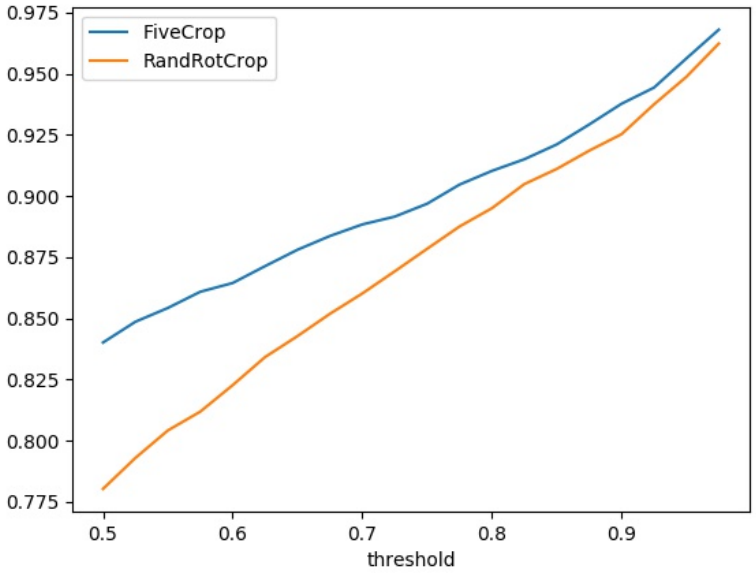

Figure 3: Average tail accuracy against threshold for both data augmentations

3 Additional Plots

Here we have the training and validation losses for both FiveCrop and RandRotCrop. The FiveCrop converges to a much lower loss compared to RandRotCrop.

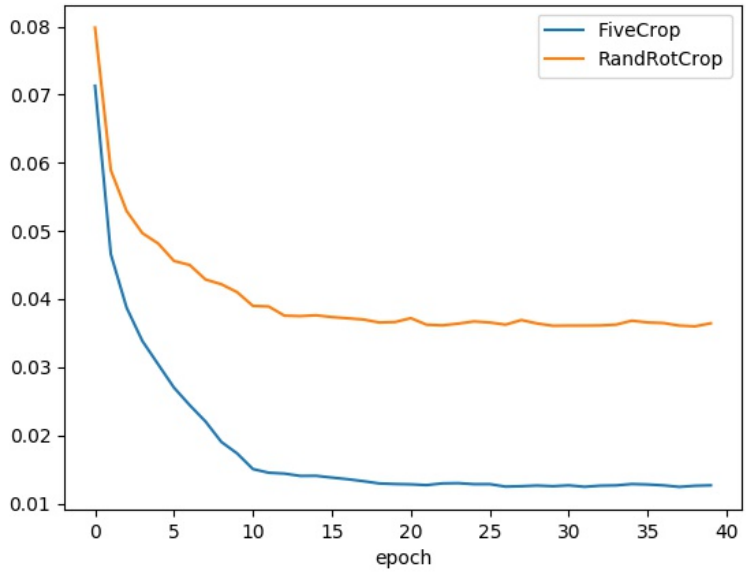

Figure 4: Training loss over epochs for both data augmentations

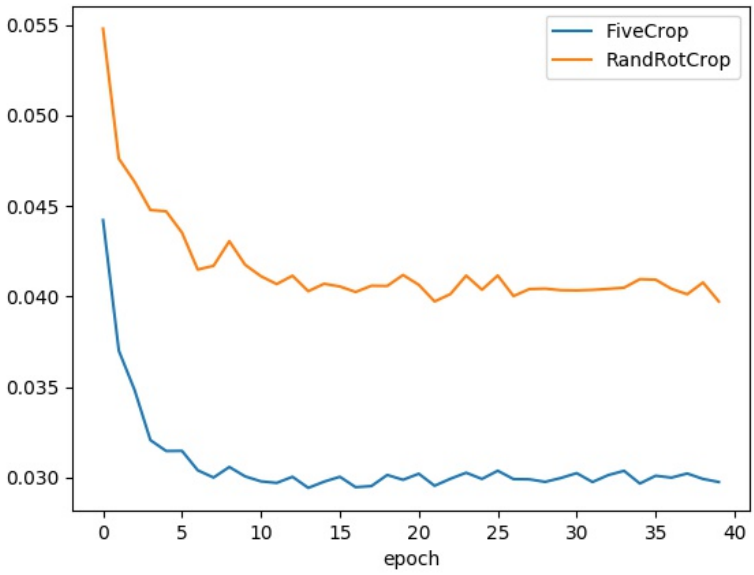

Figure 5: Validation loss over epochs for both data augmentations

4 Instructions

4.1 Training and Validation

There is a need to install Scikit-Learn since we used it to calculate the average precision as well as Pandas since the `vocparse.py` code that was provided uses it to parse the VOC dataset. This is of course in addition to both PyTorch and NumPy that is used for deep learning. To install them,

```
$ python -m pip install sklearn pandas torch numpy
```

The deep learning code is fairly straightforward. There are two data augmentations. To change the data augmentation, scroll down to the bottom of the Python script `pascal.py` and change the `FIVE_CROP` variable to whichever you want: `True` to use FiveCrop and `False` to use RandRotCrop. Assuming the dataset tarfile is in the same directory, we can do the following to run the model training and validation.

```
$ tar -xvf ./VOCtrainval_11-May-2012.tar
$ python pascal.py
```

4.2 Web GUI

For the web GUI, we will need Flask to act as the backend server, as well as a browser to run the frontend. To install Flask,

```
$ python -m pip install flask
```

To get the full experience of the web GUI, you have to unpack the VOC dataset and copy the images to the appropriate folder (for the ranks image preview). The steps for Unix systems are listed below, assuming the dataset tarfile is in the same directory as the Python scripts.

```
$ tar -xvf ./VOCtrainval_11-May-2012.tar
$ cp ./VOCdevkit/VOC2012/JPEGImages/* ./static/images/
$ python app.py
```

After running the Python script, open a browser and head to `localhost:5000`. You will be greeted by two buttons, one for predict and the other to view the pre-computed top-ranked predictions on the validation set. For the prediction, upload a JPG image and press the predict button. On the other hand, the rankings are separated by class and can be sorted by pressing the table headers.

Note: Ignore `misc.py`, it is used for plotting etc. only