Deep Learning - PascalVOC Project

Krishna Penukonda (1001781)

> This document only contains snippets of the real code that may not work in an interpreter.
>
> For the full code, please refer to the IPython notebook and accompanying python modules.

Summary

Resnet50-based model with sigmoid outputs.

No learning rate scheduler.

Converges in ~10 epochs on PascalVOC training set.

Hyperparameters

- Learning Rate: `1e-4`
- Optimizer: `Adam`
- Batch size: `8`
- Loss function: *Class-wise Binary Cross-Entropy*

Metrics

- Accuracy: `0.9628`
- Mean AP: `0.7286`
- Mean Tail Accuracy: `0.9116`

Dependencies

Install this project's dependencies via `pip install -r requirements.txt`

I use PyTorch Lightning to facilitate experiment logging. This library does not abstract any significant functionality of Pytorch other than the training loop (which is mostly copy-pasted code anyway).

```
from pathlib import Path
import shutil
from tqdm.notebook import tqdm

import numpy as np
from sklearn.metrics import average_precision_score
import matplotlib.pyplot as plt
from PIL import Image

import torch
import torchvision
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import transforms
import pytorch_lightning as pl  # Helps with logging and other stuff
```

```
from voc_data import VOCDataset, PascalVOC  # Dataloader is defined here
```

Model Training

Multilabel Classifier

The classifier is based on Resnet50 (pre-trained on ImageNet).

The final fully-connected layer is replaced with a new randomly initialized layer that has 20 outputs rather than the default 1000, and the sigmoid function is applied to its outputs.

Loss Function

The loss function I use is based on Pytorch's `binary_cross_entropy`. This function is applied class-wise and averaged to obtain the loss value (see `VOCClassifier._multi_label_loss` for implementation).

Data Pre-processing

The label for each sample is represented as a many-hot vector, with negative classes being `0` and positive ones being `1`.

During both training and inference, images are resized (shortest edge = 400px), FiveCropped and stacked, then normalized.

During training, I apply additional augmentation (random horizontal flipping and rotation).

Model Definition

```
class VOCClassifier(pl.LightningModule):
    def __init__(self, learning_rate=1e-4):
        super().__init__()
        # Hyperparameters
        self.learning_rate = learning_rate

        # Model definition
        self.stem = torchvision.models.resnet50(pretrained=True, progress=True)
        self.stem.fc = torch.nn.Linear(2048, 20)
        self.sigmoid = torch.nn.Sigmoid()

    # FORWARD PASS
    def forward(self, x):
        bs, ncrops, c, h, w = x.size()
        x = self.stem(x.view(-1, c, h, w))
        x = x.view(bs, ncrops, -1).max(1)[0]
        return self.sigmoid(x)

    # LOSS FUNCTION
    @staticmethod
    def _multi_label_loss(pred, labels):
        loss = torch.stack([F.binary_cross_entropy(
            pred[:, i],
            labels[:, i]
        ) for i in range(labels.shape[1])]).mean()
```

```
        return loss

    # ONE TRAINING ITERATION (backward pass handled by Pytorch-lightning)
    def training_step(self, batch, batch_idx):
        image, labels, _ = batch
        pred = self.forward(image)
        loss = self._multi_label_loss(pred, labels)
        output = {"loss": loss}
        if self.log_every_n_steps and batch_idx % self.log_every_n_steps == 0:
            tensorboard_logs = {"train_loss": loss}
            output["log"] = tensorboard_logs
        return output

    # OPTIMIZER
    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=self.learning_rate)
        return optimizer

model = VOCClassifier(learning_rate=1e-4)
```

`pytorch_lightning` comes in here.

It handles the training loop and the moving of tensors to GPU(s), as well as logging of metrics for visualization in Tensorboard.

```
trainer = pl.Trainer(gpus=[0])
trainer.fit(model, train_dataloader=train_dataloader,
val_dataloaders=val_dataloader)
```

Model Evaluation

Accuracy: `0.9628`

Class-wise Average Precision:

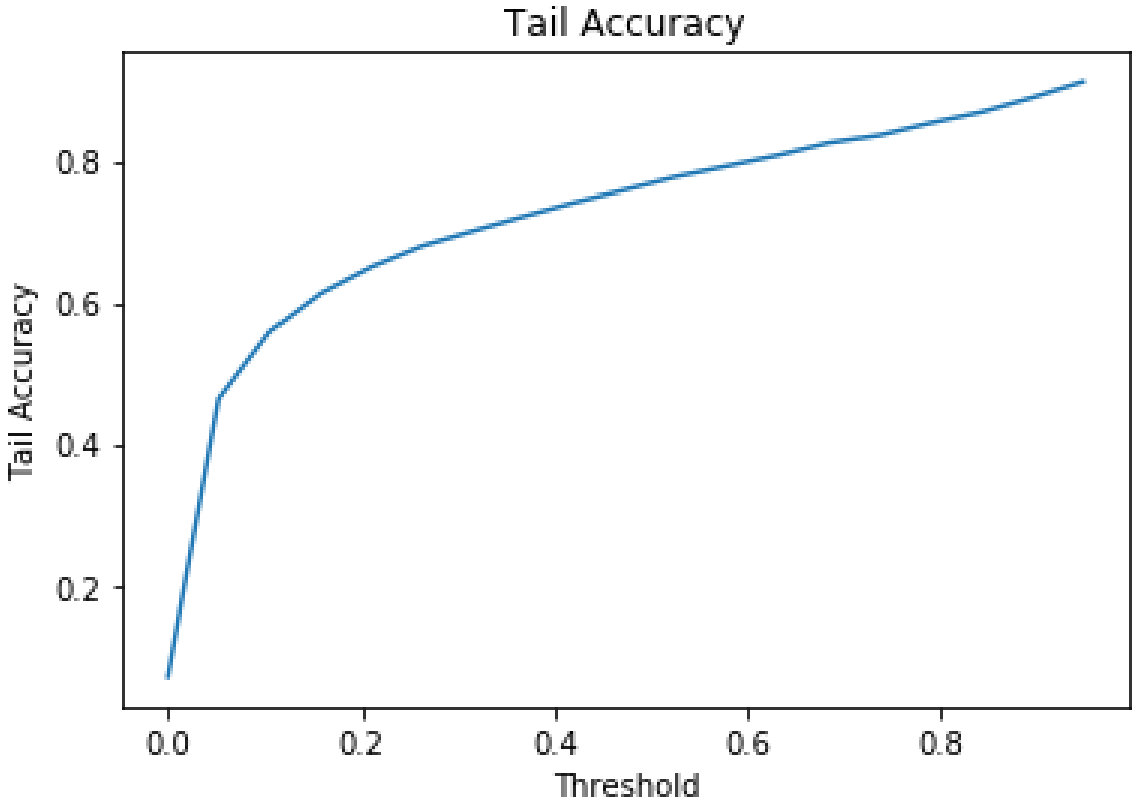
```
[0.90154388 0.75927758 0.81515773 0.72510909 0.5123822  0.85176327 0.80504543
0.89883597 0.57333306 0.61246603 0.53352054 0.83452248 0.70566054 0.78140663
0.94029254 0.56839073 0.75188008 0.45492436 0.80734184 0.74007774]
```

Mean Average Precision: `0.7286`

Class-wise Tail Accuracy:

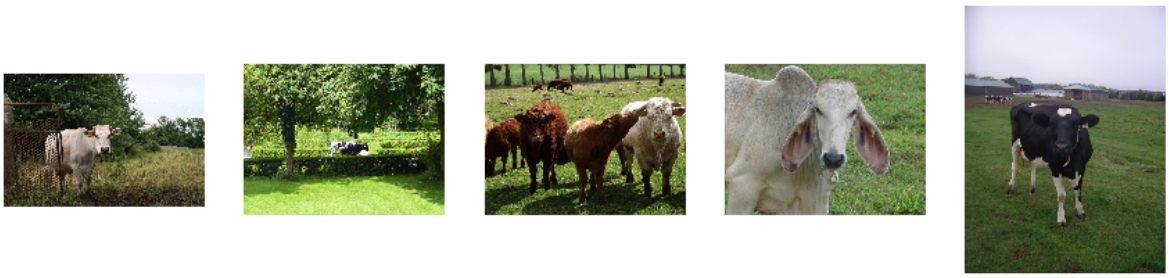
```
[0.95522384 0.93589732 0.98275856 0.83647793 0.9342104  0.9999998 0.97435893
0.9478908  0.75555551 0.73493967 0.72093006 0.96193768 0.92941166 0.97727251
0.98116297 0.8765431  0.93589732 0.81481451 0.990909   0.98684198]
```

Mean Tail Accuracy: `0.9116`



Class-wise Top-5 and Bottom-5 samples

cow: Top 5



cow: Bottom 5

bicycle: Top 5

bicycle: Bottom 5

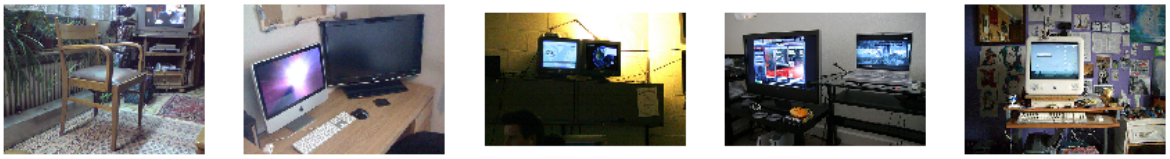
car: Top 5

car: Bottom 5

person: Top 5

person: Bottom 5

tvmonitor: Top 5



tvmonitor: Bottom 5