**Image Classification using CNN in PyTorch**

PyTorch is revolutionizing the approach of computer vision or NLP problems. It's a dynamic deep-learning framework, which makes it easier to learn and use. It is an Open-Source Machine Learning library for Python, based on Torch, used for applications such as Natural Language Processing. It is primarily developed by Facebook’s artificial-intelligence research group, and Uber’s “Pyro” software for probabilistic programming is built on it.



In this article, we will be building an image classification model from scratch to complete, start with Exploratory Data Analysis (EDA), which will help us understand the shape of an image and the distribution of its classes. You will learn to prepare data for optimum modelling results and then after we will build a Convolutional Neural Network (CNN) that will classify images according to type of classification task, namely standard multi-class, multi-output and multi-label, there are different sets of possible labels and different predictions.



*This image can be classified with following tags: portrait, woman, smiling, brown hair, wavy hair*

**Library Dependencies:**

* torch
* torchvision
* sklearn
* numpy
* pandas

**Data Overview:**

We will be using wide datasets as a toy problem which contains set of images with pre-defined labels.

Tag: person Tag: sky

Tag: person Tag: sky

Whole lot of dataset can be downloaded from [this site of NUS](https://lms.comp.nus.edu.sg/wp-content/uploads/2019/research/nuswide/NUS-WIDE.html) which contains ~175K images but is highly imbalanced.

**Overview:**

We will be using the standard [ResNeXt50](https://arxiv.org/pdf/1611.05431.pdf) architecture of torchvision. We’ll modify its output layer to apply it in our multi-label classification task.  
Instead of 1000 classes (as in ImageNet), we will be using only 27. We will also replace the softmax function with a sigmoid for a better probabality

|  |  |  |
| --- | --- | --- |
| # Use the torchvision's implementation of ResNeXt, but add FC layer for a different number of classes (27) and a Sigmoid instead of a default Softmax. | | |
|  | class Resnext50(nn.Module): |

|  |  |  |
| --- | --- | --- |
|  | def \_\_init\_\_(self, n\_classes): | |
|  | super().\_\_init\_\_() |

|  |  |  |
| --- | --- | --- |
|  | resnet = models.resnext50\_32x4d(pretrained=True) | |
|  | resnet.fc = nn.Sequential( |

|  |  |
| --- | --- |
|  | nn.Dropout(p=0.2), |
|  | nn.Linear(in\_features=resnet.fc.in\_features, out\_features=n\_classes) | |

|  |  |  |
| --- | --- | --- |
|  | ) | |
|  | | self.base\_model = resnet | |

|  |  |  |
| --- | --- | --- |
|  | self.sigm = nn.Sigmoid() | |
|  |  |

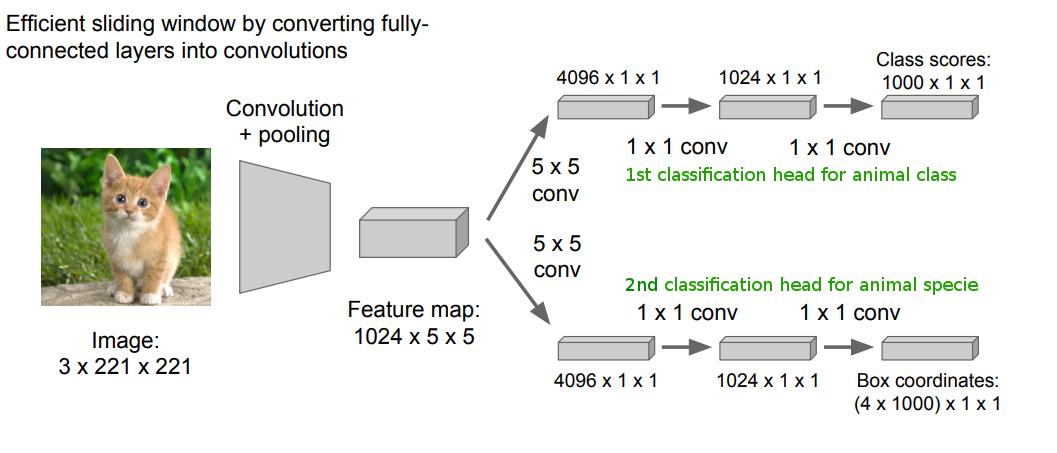
|  |  |
| --- | --- |
|  | def forward(self, x): |
|  | return self.sigm(self.base\_model(x)) | |

|  |  |
| --- | --- |
|  |  |
|  | # Initialize the model | |

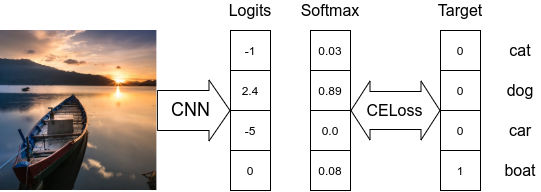
|  |  |  |
| --- | --- | --- |
|  | model = Resnext50(len(train\_dataset.classes)) | |
|  | # Switch model to the training mode |

|  |  |
| --- | --- |
|  | model.train() |

**Loss Definition:**

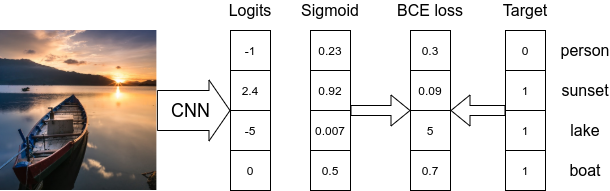
****

We’ve chosen the dataset, the model architecture. The only thing left now is the loss function, and since this is one of the classification type problem, the choice here is pretty obvious – the CrossEntropy loss. Let us understand why we actually cannot use it for the multi-label classification problem.



*model scheme with Softmax classifier and CrossEntropy Loss*

Here we got several correct labels and predicted probability for each label. Now we can compare these probabilities with the probabilities of the correct labels (ones) using BinaryCrossEntropy (BCE) loss.



*model scheme with Sigmoid classifier and BinaryCrossEntropyLoss*

**Metrices:**

We are here using [sklearn.metrics.precision\_score](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html) (as well as recall\_score and f1\_score) with parameter average=’macro’, average=’micro’ or average=’samples’ in order to calculate metrics.

|  |  |  |
| --- | --- | --- |
|  | # Use threshold to define predicted labels and invoke sklearn's metrics with different averaging strategies. | |
|  | def calculate\_metrics(pred, target, threshold=0.5): |

|  |  |
| --- | --- |
|  | pred = np.array(pred > threshold, dtype=float) |
|  | return {'micro/precision': precision\_score(y\_true=target, y\_pred=pred, average='micro'), | |

|  |  |
| --- | --- |
|  | 'micro/recall': recall\_score(y\_true=target, y\_pred=pred, average='micro'), |
|  | 'micro/f1': f1\_score(y\_true=target, y\_pred=pred, average='micro'), |

|  |  |
| --- | --- |
|  | 'macro/precision': precision\_score(y\_true=target, y\_pred=pred, average='macro'), |
|  | 'macro/recall': recall\_score(y\_true=target, y\_pred=pred, average='macro'), |

|  |  |  |
| --- | --- | --- |
|  | 'macro/f1': f1\_score(y\_true=target, y\_pred=pred, average='macro'), | |
|  | | 'samples/precision': precision\_score(y\_true=target, y\_pred=pred, average='samples'), |

|  |  |
| --- | --- |
|  | 'samples/recall': recall\_score(y\_true=target, y\_pred=pred, average='samples'), |
|  | 'samples/f1': f1\_score(y\_true=target, y\_pred=pred, average='samples'), |

|  |  |
| --- | --- |
|  | } |

**Training**

Looks like everything is ready for us to start training now.  
Here are some details about training loop initialization:

|  |  |
| --- | --- |
|  | batch\_size = 32 |
|  | max\_epoch\_number = 35 | |

|  |  |
| --- | --- |
|  | learning\_rate = 1e-3 |
|  | optimizer = torch.optim.Adam(model.parameters(), lr=learning\_rate) | |

Our training loop:

|  |  |
| --- | --- |
|  | epoch = 0 |
|  | iteration = 0 | |

|  |  |
| --- | --- |
|  | while True: |
|  | batch\_losses = [] | |

|  |  |
| --- | --- |
|  | for imgs, targets in train\_dataloader: |
|  | imgs, targets = imgs.to(device), targets.to(device) | |

|  |  |
| --- | --- |
|  |  |
|  | optimizer.zero\_grad() | |

|  |  |  |
| --- | --- | --- |
|  |  | |
|  | | model\_result = model(imgs) | |

|  |  |  |
| --- | --- | --- |
|  | loss = criterion(model\_result, targets.type(torch.float)) | |
|  |  |

|  |  |  |
| --- | --- | --- |
|  | batch\_loss\_value = loss.item() | |
|  | loss.backward() |

|  |  |  |
| --- | --- | --- |
|  | optimizer.step() | |
|  |  |

|  |  |  |
| --- | --- | --- |
|  | batch\_losses.append(batch\_loss\_value) | |
|  |  |

|  |  |  |
| --- | --- | --- |
|  | if iteration % test\_freq == 0: | |
|  | model.eval() |

|  |  |  |
| --- | --- | --- |
|  | with torch.no\_grad(): | |
|  | model\_result = [] |

|  |  |
| --- | --- |
|  | targets = [] |
|  | for imgs, batch\_targets in test\_dataloader: | |

|  |  |
| --- | --- |
|  | imgs = imgs.to(device) |
|  | model\_batch\_result = model(imgs) | |

|  |  |  |
| --- | --- | --- |
|  | model\_result.extend(model\_batch\_result.cpu().numpy()) | |
|  | targets.extend(batch\_targets.cpu().numpy()) |

|  |  |
| --- | --- |
|  |  |
|  | result = calculate\_metrics(np.array(model\_result), np.array(targets)) | |

|  |  |  |
| --- | --- | --- |
|  | print("epoch:{:2d} iter:{:3d} test: " | |
|  | "micro f1: {:.3f} " |

|  |  |
| --- | --- |
|  | "macro f1: {:.3f} " |
|  | "samples f1: {:.3f}".format(epoch, iteration, | |

|  |  |
| --- | --- |
|  | result['micro/f1'], |
|  | result['macro/f1'], |

|  |  |  |
| --- | --- | --- |
|  | result['samples/f1'])) | |
|  |  |

|  |  |  |
| --- | --- | --- |
|  | model.train() | |
|  | iteration += 1 |

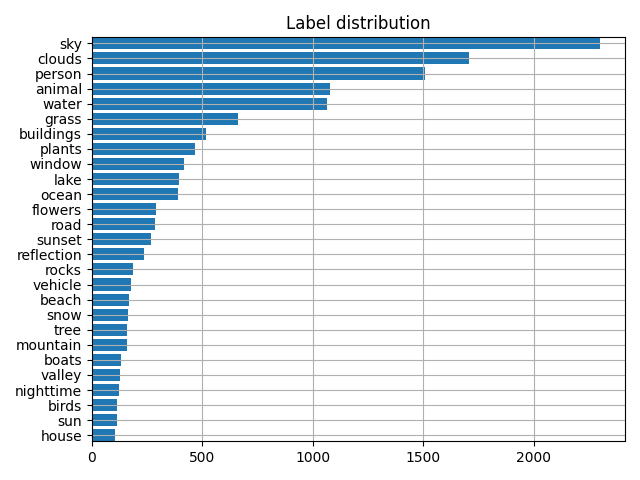
|  |  |
| --- | --- |
|  |  |
|  | loss\_value = np.mean(batch\_losses) | |

|  |  |  |
| --- | --- | --- |
|  | print("epoch:{:2d} iter:{:3d} train: loss:{:.3f}".format(epoch, iteration, loss\_value)) | |
|  | if epoch % save\_freq == 0: |

|  |  |  |
| --- | --- | --- |
|  | checkpoint\_save(model, save\_path, epoch) | |
|  | epoch += 1 |

|  |  |  |
| --- | --- | --- |
|  | if max\_epoch\_number < epoch: | |
|  | break |

We now have trained our neural networks for ~37 epochs until we begin to face overfitting. It took us another ~1.5 hour on 1080Ti. Succesively the best macro F1-score we were able to achieve is 0.624, whereas the corresponding best micro F1-score is 0.668. This difference is explained by data being quite imbalanced as we mentioned earlier and through below graph too.



**References:**

For more insights please refer to the following resources:

* [Multi-Label Text Classification](https://towardsdatascience.com/multi-label-text-classification-5c505fdedca8)
* [Deep dive into multi-label classification](https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff)
* [CNN Fully conculutional image using Tensorflow](https://www.learnopencv.com/cnn-fully-convolutional-image-classification-with-tensorflow/)
* [Image Classification- CNN with PyTorch](https://medium.com/@vivekvscool/image-classification-cnn-with-pytorch-5b2cb9ef9476)
* [CNN-PyTorch fast graph CORNELL](https://arxiv.org/abs/1903.02428)
* [Deep Learning with PyTorch: A practical approach to building neural network](https://books.google.co.in/books?hl=en&lr=&id=DOlODwAAQBAJ&oi=fnd&pg=PP1&dq=image+classification+using+cnn+pytorch&ots=kn5Yk4cAKg&sig=k25m4paBgu5g2buBNQNa2aBt2Bo&redir_esc=y#v=onepage&q=image%20classification%20using%20cnn%20pytorch&f=false)

-Vibhuti Singh