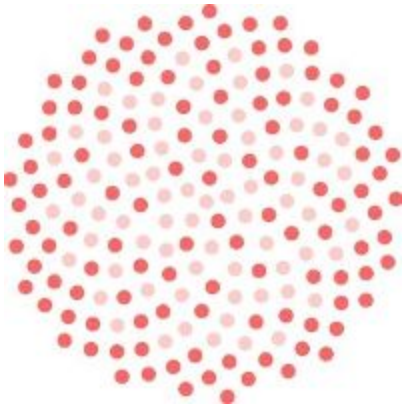
INTERNSHIP REPORT



INTELLIGENCE NODE

SHUBHAM JAIN
(DATA SCIENCE INTERN)

Project Topic: Product Catalog Categorization using Image Classification via Transfer Learning

Duration: 2 months (1st June 2020 - 31st July 2020)
Mentor: Mr. Anmol Jindal

FRAMEWORK SELECTION:

- I started with testing PyTorch and Tensorflow in terms of computation speed. For this, I took a vector of 100 images from Fashion Product Images(FPI) Dataset on Kaggle (https://www.kaggle.com/paramaggarwal/fashion-product-images-small) and passed it through a pre-trained ResNet-50 model to obtain the resulting tensor of shape [100, 2048, 7, 7]. The results were -

  Tensorflow - 0.01248 sec
  PyTorch - 0.00583 sec

- This clearly shows that PyTorch is faster so we decided to carry on further with PyTorch. (Notebooks saved by names - ‘tensorflow’ and ‘pytorch’).

FASHION Dataset - Multi-label, Multi-Class Classifier

DATA PREPROCESSING:

- Since the FPI dataset had more than 44k images, it couldn’t be trained on my system. So I got connected to the server using OpenSSH and transferred all the data.

- After doing some data preprocessing(deleting, normalizing the images) I encountered a problem while saving a NumPy array consisting of all the 44k+ images. This was solved by converting the datatype from float to uint8. (Notebooks saved by names - ‘Data preprocessing’ and ‘Data_loading’).

DATA LOADING:

- Since FPI dataset had all the images in a single folder itself so PyTorch ImageFolder couldn't be used as for PyTorch ImageLoader function to work we must have different categories of images in different folders(and folders named as the category name).
- To tackle this I was provided with a custom dataloader/image-generator function by Janak. After editing this function when I fixed it into the CNN model, I found out it wasn't compatible with the model as I was getting zero values for losses and accuracies after 1st epoch. This first led me to believe that there's some mistake in the train function. After analyzing the Train function, I was confident that it was the data-loading part which was faulty.

Researching about the Train function helped me to understand the following points -

- **loss.item()** - You are calculating the running loss with this line of code so that you could later calculate the mean loss of that epoch.
- These are the two phases in PyTorch-
  - `model.train()` is used to switch the model to training mode,
  - `model.eval()` for evaluation mode. Batchnorm or dropout layers will work in eval mode instead of training mode. Backpropagation doesn't run in eval mode.

- **optimizer.zero_grad()** – zeroes the gradients before starting the operation. We need to set gradients to 0 before starting backpropagation, to avoid the accumulation of gradients.

- **param.requires_grad** = False – freezes the layers that are not to be trained.

- **torch.set_grad_enabled** – switches gradients on/off

- **torch.no_grad()** - impacts the autograd engine and deactivate it. It will reduce memory usage and speed up computations but you won't be able to do backprop (which you don't want in an eval mode).

- In PyTorch if you use CrossEntropyLoss, you should not use softmax/sigmoid layer at the end. In Keras you can use it or not use it but set the from_logits accordingly.

- .data - tensor.data gives a tensor that shares the storage with tensor, but doesn't track history.

- The transforms operations are applied to your original images at every batch generation. So your dataset is left unchanged, only the batch images are copied and transformed every iteration.

- For data loading, passing `pin_memory=True` to a DataLoader will automatically put the fetched data Tensors in pinned memory, and thus enables faster data transfer to CUDA-enabled GPUs.
- with preds == labels.data, - use deepcopy

  with preds == labels.detach() - use clone

This led me to build a PrepareTrain and PrepareVal class which could load all the data easily. Using a class to load the data is the second most common practice in PyTorch for data loading after ImageFolder function.

ABOUT THE DATASET:

- The dataset had 44k+ images of different fashion items and accessories. There were multiple labels on each image. First I decided to build a Multi-Class classifier based on single label “articleType” and if that was successful in time then I would start making a Muti-Label, Multi-Class classifier.

- Multi-Class classifier made by me was trained on the ResNet-50 model had around 20% accuracy in 6th epoch. Early Stopping was also used in the model to stop it if the results are not in favor. Rather than fine-tuning to improve the accuracy score, I first started to build a Muti-Label, Multi-Class classifier.

RESEARCH AND RESULTS:

- PrepareTrain and PrepareVal class worked for training the Multi-Class classifier but it was later that I found out that I was unable to unnormalize the images to see any sample output. Also, I had not applied Transforms on images and I could not apply transforms now as I already had my data saved in the form of NumPy array and Transform can be applied only on PIL images. So I considered making a different dataloader class MyDataset which would directly load the data from the images folder and apply the normalize, resize, and transform operations.
- Muti-Label, Multi-Class classifier is based on labels “masterCategory”, “subCategory” and “articleType”. Only the following 4 out of 7 labels were used for classification according to the requirement:

- Master category - 'Apparel', 'Accessories', 'Footwear', 'Sporting Goods’

I did k-hot encoding which made the data of 141 classes out of which 3 labels must be true. The dataset division was Train: Val: Test = 60:20:20.

- Sigmoid - as a last layer converts [-inf to inf] to probability. Good for multi-label classification.Bad for multi-class classification.
- Softmax - Good for multi-class classification.
- Final layer in deep learning has logit values which are raw values for prediction by softmax.
- PyTorch has built-in `BCEWithLogitsLoss` loss function specially designed for Muti-Label, Multi-Class classification.
- `CrossEntropyLoss()` uses the target to index the logits in your model’s output. Thus it is suitable for multi-class classification use cases (only one valid class in the target).
- `BCEWithLogitsLoss()`, on the other hand, treats each output independently and is suitable for multi-label classification use cases.

Problems faced:

1. Whenever we save a CSV file it adds index column of its own. Index=False helps to avoid this.
2. Whenever any edits are made to the Pandas DataFrame the index is not updated. This creates NaN values. To avoid reset_index() can be used.

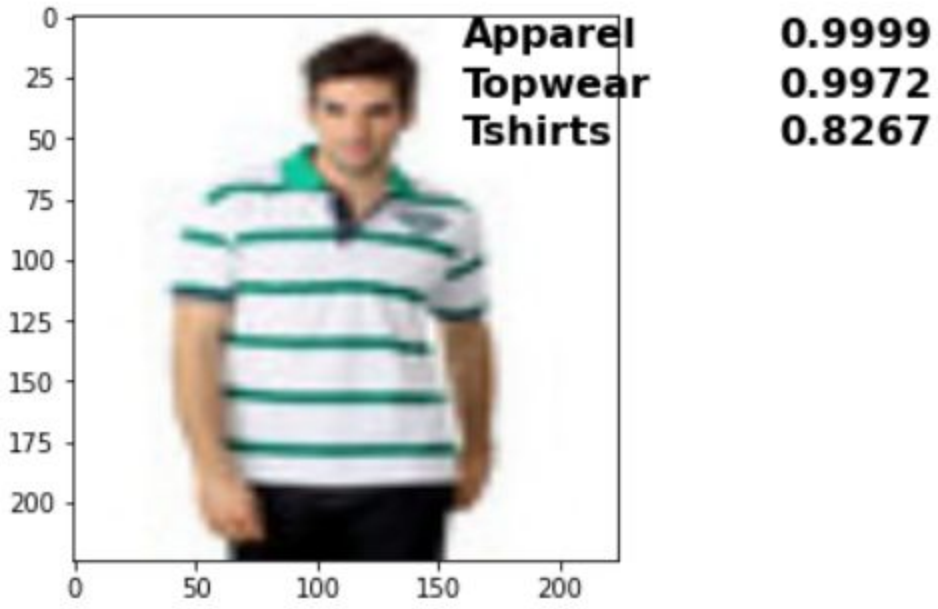
Training the Muti-Label, Multi-Class classifier on different pre-trained CNN architectures gave the following results:- (All notebooks saved at /disk1/notebooks/fashion/)

Model	Dataset	Dataset Size	Train/Val Split	Classes	Training Accuracy	Validation Accuracy	No. of Epochs
ResNet50	Fashion Product Ima	41907	36879/5028	141(3 labels)	81.55%	86.43%	2
ResNet50	Fashion Product Ima	41907	25144/8381	141(3 labels)	82.04%	87.36%	2
MobileNetV2	Fashion Product Ima	41907	36879/5028	141(3 labels)	84.73%	87.45%	2
Inception Net	Fashion Product Ima	41907	25144/8381	141(3 labels)	79.86%	86.79%	5

These results can be found at - https://docs.google.com/spreadsheets/d/1w5kQLXMLCpFDtQdanE3wFEBNfhIDRIxkDOYl1tS6Au4/edit?ts=5f2392cc#gid=1859824467

- Results show that ResNet50 gives better results than InceptionNet_v3.
- Stratified sampling is important to have a balanced dataset. Also, the accuracy of ResNet50 increased when stratified sampling is used even when fewer training examples are there.

Sample output: Labels and their corresponding probabilities are shown.



Home and Decor Dataset - Multi-Class Classifier

ABOUT THE DATASET:

- Further, I started building Multi-Class classifier for HnD Dataset which had roughly 1 lakh images of 276 different classes and dataset divided into Train: Val: Test = 60:20:20. ImageFolder function was used for data loading as the data was already in different folders. A lot of images in the dataset were invalid and had to be deleted. This was done by using the Linux command -

  find . -name “*.jpg” -type ‘f’ -size -120b -delete

- To apply stratified sampling I had to delete some classes who had less than 3 images present. On training this model I didn’t receive expected results so I made a copy of the dataset with names - 500+,1000+, and 2000+. These folders had classes with images more than those specified. I did this because the accuracy of the dataset was really low so I wanted to train it on more training images per class to increase the accuracy.

DIFFERENCES FROM FASHION DATASET:

- CrossEntropyLoss() is used instead of BCEWithLogitsLoss().
- ImageFolder function is used for data-loading instead of custom dataloader.
- SGD (Scholastic Gradient Descent) is used instead of Adam Optimizer.

RESULTS:

- Training it on different pre-trained CNN architectures gave the following results:-

Model	Dataset	Dataset Size	Train/Val Split	Classes	Training Accuracy	Validation Accuracy
MobileNetV2	HnD 500+	79886	60 : 20	33	2.01%	1.50%
MobileNetV2	HnD 1000+	68393	60 : 20	15	11.33%	15.21%
MobileNetV2	HnD 2000+	59502	60 : 20	8	11.08%	9.97%
ResNet50	HnD 2000+	59502	60 : 20	8	11.16%	12.02%

- This shows that ResNet50 performs better than MobileNet_v2 on HnD dataset.
- The accuracy of the dataset is low as data augmentation is not used and

REFERENCES:

https://deepsense.ai/keras-vs-pytorch-avp-transfer-learning/

https://www.learnopencv.com/multi-label-image-classification-with-pytorch/

https://github.com/aman5319/Multi-Label/blob/master/Classify_scenes.ipynb

https://github.com/mtagda/transfer-learning-fashion-dataset

https://www.pluralsight.com/guides/image-classification-with-pytorch

https://jhui.github.io/2018/02/09/PyTorch-Data-loading-preprocess_torchvision/