Introduction :

ML models have to learn every time from scratch . What if the models can use knowledge learnt from recognising cats, dogs ,fish ,cars , bus and many more to identify a distracted car driver or to identify plant disease . In transfer learning we use a pre trained neural network in extracting features and training a new model for a particular use case.

Why PyTorch :

The name **PyTorch** is inspired from popular library **Torch**, which was written in **Lua**.

The first key feature of PyTorch is **imperative programming**. An imperative program performs computation as you type it.

The second key feature of PyTorch is **Dynamic computational graphs**. PyTorch is defined by run, which means the graph structure is generated at runtime. These graphs arise whenever the amount of work that needs to be done is variable.

There are many frameworks like Keras, Tensoflow, Theano, Torch, Deeplearning.4J, etc. which can be used for deep learning . Since Keras was built in a nice modular fashion it lacks flexibility . PyTorch which is a new entrant ,provides us tools to build various deep learning models in object oriented fashion thus providing a lot of flexibility . A lot of the difficult architectures are being implemented in PyTorch recently.

In this project we will go through how easy it is to build a state of art classifier with a very small dataset and in a few lines of code.

We will build a classifier for detecting Ants and Bees .

Transfer Learning :

We will use a model called ResNet from Microsoft which won the ImageNet competition in 2015. It showed how deep networks can be made possible. We will download the model and most of the modern deep learning frameworks makes loading a model easier. The ResNet model comprises of a bunch of ResNet blocks(Combination of convolution and identity block) and a fully connected layer. The model is trained on Imagenet dataset on 1000 categories , we will remove the last fully connected layer and add a new fully connected layer which outputs 2 categories which tells the probability of the image being Ant or Bee.

Features :

Data augmentation :

Data augmentation is a process where you make changes to existing photos like adjusting the colors , flipping it horizontally or vertically , scaling , cropping and many more. Pytorch provides a very useful library called torchvision.transforms which provides a lot of methods which helps to apply data augmentation. transforms comes with a compose method which takes a list of transformation.

Transfer Learning :

1. We tell the model not to learn or modify the weights / parameters of the model.
2. Then we add a new fully connected layer to the existing model to train our model to classify 2 categories.

Training Model :

For training model we need a couple of more things apart from the model like:

1. PyTorch Variable : A variable wraps pytorch tensor .It contains data and the gradient associated with the data.
2. Loss Function : It helps in calculating how good is our model. We will be using categorical cross entropy here.
3. Optimizer : We will use SGD to optimise our weights with the gradients. In our case we update the weights of only the last layer.
4. Forward propagation : This is the simplest part where we pass our data through the model.
5. Backward propagation : This is the key for modern deep learning networks where all the magic happens. Where the optimizer starts calculating how much the weights need to be updated in order to reduce the loss or improve the accuracy. In most modern frameworks this is automated , so we can focus on building cool applications backed by deep learning.

Decaying Learning Rate :

Most of the times we start with a higher learning rate so that we can reduce the loss faster and then after a few epochs you would like to reduce it so that the learning becoming slower.
We are reducing the learning rate for every nth epoch , in the above example 7 with 0.1 . decay_rate is configurable. Even on a smaller dataset we can achieve state of art results using this approach.

Neural networks use transfer learning for various computer vision tasks
we will look into the following.

1. VGG Architecture
2. Fine tune VGG using pre-convoluted features
3. Accuracy

Software Requirements :

Framework : PyTorch ,

Jupyter Notebook,

Python,

Convolutional Neural Network,

PreTrained ResNet Model .

Advantages :

The main advantage is that PyTorch uses dynamic computation graphs while Tensorflow (or Keras, or Theano) uses static graphs. In Tensorflow everything you do must operate on the same structures, and it must always be the same computations. You must define the computation graph once and for all, you can not use Python control flow, you can not stop the program at every point, etc.

In PyTorch you can use standard python control flow (if-else, loops), and your model can be different for every sample. Without much effort you can create tree-shaped RNNs for example, a thing which is very difficult to do with Tensorflow. You can also use python debuggers with PyTorch, which means you can stop the program at any point and inspect variables, gradients, whatever you like.

IMHO if you don't need extremely high computational efficiency there is no reason to use Tensorflow, Keras, Theano, or the likes instead of PyTorch (or any other dynamic computation graph framework).

Writing complex models (researcher): A lot of research work now-a-days is writing complex models which can interact over the third and fourth axes (batch and temporal series along with length and breadth of individual objects as compared to earlier where last two axes were almost always used) . Keras works on top of static frameworks like Tensorflow and Theano and inherits their rigidity in this regard. PyTorch on the other hand, is just super slick in this sense, no extra effort.

DataFlow:

-Finetuning the convnet

-ConvNet as fixed feature extractor

-Load Data

-Visualize a few images

-Training the model

-Scheduling the learning rate

- Saving the best model
- Visualizing the model predictions

-Finetuning the convnet

-ConvNet as fixed feature extractor

-Train and evaluate

Application :

One particular application of transfer learning that we'll see more of in the future is learning from simulations. For many machine learning applications that rely on hardware for interaction, gathering data and training a model in the real world is either expensive, time-consuming, or simply too dangerous. It is thus advisable to gather data in some other, less risky way.