



LAB MANUAL

LAB PRACTICE- IV

Course Code: 414447

Class: B.E.I.T.

Course Pattern: 2019

Semester VII A.Y. 2023-24

SUBJECT INCHARGE:- Ms. Kashmira P. Jayakar

LIST OF LAB EXPERIMENTS
ACADEMIC YEAR : 2023-24
DEPARTMENT : INFORMATION TECHNOLOGY
CLASS : B.E.
SUBJECT: 414447: Lab Practice IV

Sr. NO	PROBLEM STATEMENT
1	Study of Deep learning Packages: Tensorflow, Keras, Theano and PyTorch. Document the distinct features and functionality of the packages.
2	Implementing Feedforward neural networks with Keras and TensorFlow a. Import the necessary packages b. Load the training and testing data (MNIST/CIFAR10) c. Define the network architecture using Keras d. Train the model using SGD e. Evaluate the network f. Plot the training loss and accuracy
3	Build the Image classification model by dividing the model into following 4 stages: a. Loading and preprocessing the image data b. Defining the model's architecture c. Training the model d. Estimating the model's performance
4	Use Autoencoder to implement anomaly detection. Build the model by using: a. Import required libraries b. Upload / access the dataset c. Encoder converts it into latent representation d. Decoder networks convert it back to the original input e. Compile the models with Optimizer, Loss, and Evaluation Metrics
5	Implement the Continuous Bag of Words (CBOW) Model. Stages can be: a. Data preparation b. Generate training data c. Train model d. Output
6	Object detection using Transfer Learning of CNN architectures a. Load in a pre-trained CNN model trained on a large dataset b. Freeze parameters (weights) in model's lower convolutional layers c. Add custom classifier with several layers of trainable parameters to model d. Train classifier layers on training data available for task e. Fine-tune hyper parameters and unfreeze more layers as needed

Assignment No.1

Title : Study of Deep learning Packages: Tensorflow, Keras, Theano and PyTorch.
Document the distinct features and functionality of the packages.

Aim: Study and installation of following Deep learning Packages :

- i. Tensor Flow
- ii. Keras
- iii. Theano
- iv. PyTorch

Theory : 1) What is Deep learning ?

2) What are various packages in python for supporting Machine Learning libraries and which are mainly used for Deep Learning ?

3) Compare Tensorflow / Keras/Theano and PyTorch on following points (make a table) :

- i. Available Functionality
- ii. GUI status
- iii. Versions.
- iv. Features
- v. Compatibility with other environments.
- vi. Specific Application domains.

4) Enlist the Models Datasets and pretrained models, Libraries and Extensions , Tools related to Tensorflow also discuss any two case studies like (PayPal, Intel, Etc.) related to Tensor Flow.

[Ref: <https://www.tensorflow.org/about>]

5) Explain the Keras Ecosystem. (kerastuner, kerasNLP, kerasCV, Autokeras and Model optimization.) Also explain following concepts related to keras : 1. Developing sequential Model 2. Training and validation using the inbuilt functions 3. Parameter Optimization. [Ref: <https://keras.io/>]

6) Explain simple Theano program.

7) Explain PyTorch Tensors . And also explain Uber's Pyro, Tesla Autopilot. [<https://pytorch.org/>]

Steps/ Algorithm

Installation of Tensorflow On Ubuntu:

1. 1. Install the Python Development Environment:

You need to download Python, the PIP package, and a virtual environment. If these packages are already installed, you can skip this step.

You can download and install what is needed by visiting the following links:

<https://www.python.org/>

<https://pip.pypa.io/en/stable/installing/>

<https://docs.python.org/3/library/venv.html>

To install these packages, run the following commands in the terminal:

```
sudo apt update
```

```
sudo apt install python3-dev python3-pip python3-venv
```

2. Create a Virtual Environment

[Type here]

Navigate to the directory where you want to store your Python 3.0 virtual environment. It can be in your home directory, or any other directory where your user can read and write permissions.

```
mkdir tensorflow_files
```

```
cd tensorflow_files
```

Now, you are inside the directory. Run the following command to create a virtual environment:

```
python3 -m venv virtualenv
```

The command above creates a directory named virtualenv. It contains a copy of the Python binary, the PIP package manager, the standard Python library, and other supporting files.

3. Activate the Virtual Environment

```
source virtualenv/bin/activate
```

Once the environment is activated, the virtual environment's bin directory will be added to the beginning of the \$PATH variable. Your shell's prompt will alter, and it will show the name of the virtual environment you are currently using, i.e. virtualenv.

4. Update PIP

```
pip install --upgrade pip
```

5. 5. Install TensorFlow

The virtual environment is activated, and it's up and running. Now, it's time to install the TensorFlow package.

```
pip install -- upgrade TensorFlow
```

Installation of Keras on Ubuntu :

Prerequisite : Python version 3.5 or above.

STEP 1: Install and Update Python3 and Pip

Skip this step if you already have Python3 and Pip on your machine.

```
sudo apt install python3 python3.pip
```

```
sudo pip3 install —upgrade pip
```

STEP 2: Upgrade Setuptools

```
pip3 install —upgrade setuptools
```

STEP 3: Install TensorFlow

```
pip3 install tensorflow
```

Verify the installation was successful by checking the software package information:

```
pip3 show tensorflow
```

STEP 4: Install Keras

```
pip3 install keras
```

Verify the installation by displaying the package information:

```
pip3 show keras
```

<https://phoenixnap.com/kb/how-to-install-keras-on-linux>

[Type here]

Installation of Theano on Ubuntu:

Step 1: First of all, we will install Python3 on our Linux Machine. Use the following command in the terminal to install Python3.

```
sudo apt-get install python3
```

Step 2: Now, install the pip module

```
sudo apt install python3-pip
```

Step 3: Now, install the Theano

Verifying Theano package Installation on Linux using PIP

```
python3 -m pip show theano
```

Installation of PyTorch

First, check if you are using python's latest version or not. Because PyGame requires python 3.7 or a higher version

```
python3 --version
```

```
pip3 --version
```

```
pip3 install torch==1.8.1+cpu torchvision==0.9.1+cpu torchaudio==0.8.1 -f  
https://download.pytorch.org/whl/torch_stable.html
```

[Ref : <https://www.geeksforgeeks.org/install-pytorch-on-linux/>]

Python Libraries and functions required

1. Tensorflow, keras

numpy : NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy stands for Numerical Python. To import numpy use

```
import numpy as np
```

pandas: pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. To import pandas use

```
import pandas as pd
```

sklearn : Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib. For importing train_test_split use

```
from sklearn.model_selection import train_test_split
```

2. For TheaonRequirements:

- Python3
- Python3-pip
- NumPy

[Type here]

- SciPy
- BLAS

Sample Code with comments

1. Tensorflow Test program:

```
import tensorflow as tf

print(tf.__version__)

2.1.0

print(tf.reduce_sum(tf.random.normal([1000, 1000])))

tf.Tensor(-505.04108, shape=(), dtype=float32)
```

2. Keras Test Program:

```
1 from tensorflow import keras
from keras import datasets

#
# Load MNIST data
#
(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()
#
# Check the dataset loaded
#
train_images.shape, test_images.shape
```

3. Theano test program

```
# Python program showing
# addition of two scalars

# Addition of two scalars
import numpy
import theano.tensor as T
from theano import function

# Declaring two variables
x = T.dscalar('x')
y = T.dscalar('y')

# Summing up the two numbers
z = x + y

# Converting it to a callable object
# so that it takes matrix as parameters
```

[Type here]

```
f = function([x, y], z)
f(5, 7)
4. Test program for PyTorch
```

```
## The usual imports
import torch
import torch.nn as nn

## print out the pytorch version used
print(torch.__version__)
```

Output of Code:

Note: Run the code and attach your output of the code here.

Conclusion :

Tensorflow , PyTorch,Keras and Theano all these packages are installed and ready for Deep learning applications . As per application domain and dataset we can choose the appropriate package and build required type of Neural Network.

Assignment No.2

Title : Implementing Feedforward neural networks

Aim: Implementing Feedforward neural networks with Keras and TensorFlow

- a. Import the necessary packages
- b. Load the training and testing data (MNIST/CIFAR10)
- c. Define the network architecture using Keras
- d. Train the model using SGD
- e. Evaluate the network
- f. Plot the training loss and accuracy

Theory : 1) What is Feedforward Neural Network ?

- 2) How the Feedforward Neural Network Works ?
- 3) Enlist at least three Real time scenarios where Feedforward Neural Network is used.
- 4) Explain the components of Feedforward Neural Network.
- 5) What is cost function in Feedforward Neural Network.
- 6) Define mean square error cost function.
- 7) What is Loss function in Feedforward Neural Network.
- 8) What is cross entropy loss.
- 9) What is kernel concept related to Feedforward Neural Network.
- 10) Describe MNIST and CIFAR 10 Dataset.
- 11) Explain use and parameter setting related to feedforward network implementation for following libraries : SKlearn : i) LabelBinarizer (sklearn.preprocessing) ii) classification_report (sklearn.metrics) and tensorflow.keras : models , layers, optimizers, datasets, backend and set to respective values.
- 12) What is meant by flattening the dataset and why it is needed related to standard neural network implementation .
- 13) Explain difference between Sigmoid and Softmax activation function.
- 14) What is significance of optimizer in training model.
- 15) What is Epochs in fit command in training .

Steps/ Algorithm

1. Dataset link and libraries :

Dataset : MNIST or CIFAR 10 : kaggle.com

You can download dataset from above mentioned website.

Libraries required :

Pandas and Numpy for data manipulation

Tensorflow/Keras for Neural Networks

Scikit-learn library for splitting the data into train-test samples, and for some basic model evaluation

<https://pyimagesearch.com/2021/05/06/implementing-feedforward-neural-networks-with-keras-and-tensorflow/>

- a) Import following libraries from SKlearn : i) LabelBinarizer (sklearn.preprocessing) ii) classification_report (sklearn.metrics) .
- b) Import Following libraries from tensorflow.keras : models , layers, optimizers, datasets , backend and set to respective values.

[Type here]

- c) Grab the MNIST dataset or required dataset.
- d) Flatten the dataset.
- e) If required do the normalization of data .
- f) Convert the labels from integers to vectors.(specially for one hot coding)
- g) Decide the Neural Network Architecture : i) Select model (Sequential recommended)
ii) Activation function (sigmoid recommended) iii) Select the input shape iv) see the weights in the output layer
- h) Train the model : i) Select optimizer (SGD recommended) ii) use model that .fit to start training ii) Set Epochs and batch size
- i) Call model.predict for class prediction.
- j) Plot training and loss accuracy
- k) Calculate Precision, Recall, F1-score, Support
- l) Repeat for CIFAR dataset.

Sample Code with comments and Output : Attach Printout with Output .

Conclusion : Should be based on Evaluation model parameters and plots.

Assignment No.3

Title : Build the Image classification model

Aim: Build the Image classification model by dividing the model into following 4 stages:

- a. Loading and pre-processing the image data
- b. Defining the model's architecture
- c. Training the model
- d. Estimating the model's performance

Theory : 1)What is Image classification problem?

2)Why to use Deep learning for Image classification ? State and compare different Type of Neural Networks used for the Image classification?

3) What is CNN?

4) Explain Convolution operation and Convolution kernel related to Deep learning.

5) Explain how kernel operate on the Input image by taking sample matrix.

6) Explain the types of convolution and convolution layers related to CNN.

7) Explain how the feature extraction is done with convolution layers?

8) Explain

Steps/ Algorithm

1. Choose a dataset of your interest or you can also create your own image dataset

(Ref : <https://www.kaggle.com/datasets/>) Import all necessary files.

(Ref : <https://www.analyticsvidhya.com/blog/2021/01/image-classification-using-convolutional-neural-networks-a-step-by-step-guide/>)

Libraries and functions required

1. Tensorflow,keras

numpy : NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy stands for Numerical Python. To import numpy use

```
import numpy as np
```

pandas: pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. To import pandas use

```
import pandas as pd
```

sklearn : Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib. For importing train_test_split use

2. Prepare Dataset for Training : //Preparing our dataset for training will involve assigning paths and creating categories(labels), resizing our images.
3. Create a Training a Data : // Training is an array that will contain image pixel values and the index at which the image in the CATEGORIES list.
4. Shuffle the Dataset
5. Assigning Labels and Features
6. Normalising X and converting labels to categorical data
7. Split X and Y for use in CNN
8. Define, compile and train the CNN Model
9. Accuracy and Score of model.

Sample Code with comments and Output : Attach Printout with Output .

Conclusion :

As per the evaluation of model write down in line with your output about accuracy and other evaluation parameters.

Assignment No.4

Title : ECG Anomaly detection using Autoencoders

Aim: Use Autoencoder to implement anomaly detection. Build the model by using:

- Import required libraries
- Upload / access the dataset
- Encoder converts it into latent representation
- Decoder networks convert it back to the original input
- Compile the models with Optimizer, Loss, and Evaluation Metrics

Theory : 1) What is **Anomaly Detection** ?

- What are Autoencoders in Deep learning ?
- Enlist different applications with Autoencoders in DL.
- Enlist different types of anomaly detection Algorithms.
- What is difference between Anomaly detection and Novelty Detection.
- Explain different blocks and working of Autoencoders.
- What is reconstruction and Reconstruction errors .

8) What is Minmaxscaler from sklearn.

8) Explain . train_test_split from sklearn.

9) What is anomaly scores.

10) Explain tensorflow dataset.

11) Describe the ECG Dataset.

12) Explain keras Optimizers

13) Explain keras layers dense and dropouts

14) Explain keras losses and meansquarelogarithmicerror

15) Explain Relu activation function

Steps/ Algorithm

1. Dataset link and libraries :

Dataset : <http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv>

Libraries required :

Pandas and Numpy for data manipulation

Tensorflow/Keras for Neural Networks

Scikit-learn library for splitting the data into train-test samples, and for some basic model evaluation

For Model building and evaluation following libraries:

sklearn.metrics import accuracy_score

tensorflow.keras.optimizers import Adam

sklearn.preprocessing import MinMaxScaler

tensorflow.keras import Model, Sequential

tensorflow.keras.layers import Dense, Dropout

tensorflow.keras.losses import MeanSquaredLogarithmicError

Ref:<https://www.analyticsvidhya.com/blog/2021/05/anomaly-detection-using-autoencoders-a-walk-through-in-python/>

- a) Import following libraries from SKlearn : i) MinMaxScaler (sklearn.preprocessing) ii) Accuracy(sklearn.metrics) . iii) train_test_split (model_selection)
- b) Import Following libraries from tensorflow.keras : models , layers, optimizers, datasets , and set to respective values.
- c) Grab to ECG.csv required dataset
- d) Find shape of dataset
- e) Use train_test_split from sklearn to build model (e.g. train_test_split(features, target, test_size=0.2, stratify=target)
- f) Take usecase Novelty detection hence select training data set as Target class is 1 i.e. Normal class
- g) Scale the data using MinMaxScaler.
- h) Create Autoencoder Subclass by extending model class from keras.
- i) Select parameters as i) Encoder : 4 layers ii) Decoder : 4 layers iii) Activation Function : Relu iv) Model : sequential.
- j) Configure model with following parametrs : epoch = 20 , batch size =512 and compile with Mean Squared Logarithmic loss and Adam optimizer.

```
e.g. model = AutoEncoder(output_units=x_train_scaled.shape[1])
```

```
# configurations of model
```

```
model.compile(loss='msle', metrics=['mse'], optimizer='adam')
```

```
history = model.fit(
```

```
    x_train_scaled,
```

```
    x_train_scaled,
```

```
    epochs=20,
```

```
    batch_size=512,
```

```
    validation_data=(x_test_scaled, x_test_scaled)
```

- k) Plot loss, Val_loss, Epochs and msle loss
- l) Find threshold for anomaly and do predictions :
e.g. : find_threshold(model, x_train_scaled):
reconstructions = model.predict(x_train_scaled)
provides losses of individual instances

```
reconstruction_errors = tf.keras.losses.msle(reconstructions, x_train_scaled)

# threshold for anomaly scores

threshold = np.mean(reconstruction_errors.numpy()) \
            + np.std(reconstruction_errors.numpy())

return threshold
```

m) Get accuracy score

Sample Code with comments : Attach Printout with Output .

Conclusion: Should be based on Evaluation model parameters and plots.

Assignment No.5

Title : Implement the Continuous Bag of Words (CBOW) Model.

Aim: Implement the Continuous Bag of Words (CBOW) Model. Stages can be:

- Data preparation
- Generate training data
- Train model
- Output

Theory : 1)What is NLP ?

- What is Word embedding related to NLP ?
- Explain Word2Vec techniques.
- Enlist applications of Word embedding in NLP.
- Explain CBOW architecture.
- What will be input to CBOW model and Output to CBW model.
- What is Tokenizer .
- 8) Explain window size parameter in detail for CBOW model.**
- 9) Explain Embedding and Lambda layer from keras**
- 10) What is yield()**

Steps/ Algorithm

- Dataset link and libraries :

Create any English 5 to 10 sententece paragraph as input

Import following data from keras :

keras.models import Sequential

keras.layers import Dense, Embedding, Lambda

keras.utils import np_utils

keras.preprocessing import sequence

keras.preprocessing.text import Tokenizer

Import Gensim for NLP operations : requirements :

Gensim runs on Linux, Windows and Mac OS X, and should run on any other platform that supports Python 3.6+ and NumPy. Gensim depends on the following software: Python, tested with versions 3.6, 3.7 and 3.8. NumPy for number crunching.

Ref: <https://analyticsindiamag.com/the-continuous-bag-of-words-cbow-model-in-nlp-hands-on-implementation-with-codes/>

- Import following libraries gensim and numpy set i.e. text file created . It should be preprocessed.
- Tokenize the every word from the paragraph . You can call in built tokenizer present in Gensim
- Fit the data to tokenizer

[Type here]

d) Find total no of words and total no of sentences.

e) Generate the pairs of Context words and target words :

e.g. cbow_model(data, window_size, total_vocab):

```
total_length = window_size*2
```

```
for text in data:
```

```
    text_len = len(text)
```

```
    for idx, word in enumerate(text):
```

```
        context_word = []
```

```
        target = []
```

```
        begin = idx - window_size
```

```
        end = idx + window_size + 1
```

```
        context_word.append([text[i] for i in range(begin, end) if 0 <= i < text_len and i  
!= idx])
```

```
        target.append(word)
```

```
        contextual = sequence.pad_sequences(context_word, total_length=total_length)
```

```
        final_target = np_utils.to_categorical(target, total_vocab)
```

```
        yield(contextual, final_target)
```

f) Create Neural Network model with following parameters . Model type : sequential

Layers : Dense , Lambda , embedding. Compile Options :

(loss='categorical_crossentropy', optimizer='adam')

g) Create vector file of some word for testing

e.g.:dimensions=100

```
vect_file = open('/content/gdrive/My Drive/vectors.txt', 'w')
```

```
vect_file.write('{} {} \n'.format(total_vocab,dimensions))
```

h) Assign weights to your trained model

e.g. weights = model.get_weights()[0]

```
for text, i in vectorize.word_index.items():
```

```
    final_vec = ''.join(map(str, list(weights[i, :])))
```

```
    vect_file.write('{} {} \n'.format(text, final_vec))
```

```
Close()
```

i) Use the vectors created in Gensim :


```
e.g. cbow_output =  
gensim.models.KeyedVectors.load_word2vec_format('/content/gdrive/My  
Drive/vectors.txt', binary=False)
```

```
j) choose the word to get similar type of words:  
cbow_output.most_similar(positive=['Your word'])
```

Sample Code with comments : Attach Printout with Output .

Conclusion: Explain how Neural network is useful for CBOW text analysis.

Assignment No.6

Title : Object detection using Transfer Learning of CNN architectures

Aim: Object detection using Transfer Learning of CNN architectures

- Load in a pre-trained CNN model trained on a large dataset
- Freeze parameters (weights) in model's lower convolutional layers
- Add custom classifier with several layers of trainable parameters to model
- Train classifier layers on training data available for task
- Fine-tune hyper parameters and unfreeze more layers as needed

Theory : 1)What is Transfer learning ?

- What are pretrained Neural Network models ?
- Explain Pytorch library in short.
- What are advantages of Transfer learning.
- What are applications of Transfer learning.
- Explain Caltech 101 images dataset.
- Explain Imagenet dataset .
- list down basic steps for transfer learning.
- What is Data augmentation?
- How and why Data augmentation is done related to transfer learning?
- Why preprocessing is needed on inputdata in Transfer learning.
- What is PyTorch Transforms module.Explain following commands w.r.t it :

```
Compose([
    RandomResizedCrop(size=256, scale=(0.8, 1.0)),
    RandomRotation(degrees=15),
    ColorJitter(),
    RandomHorizontalFlip(),
    CenterCrop(size=224), # Image net standards
    .ToTensor(),
    Normalize
```

- Explain the Validation Transforms steps with Pytorch Transforms .
- Explain VGG-16 model from Pytorch

Steps/ Algorithm

- Dataset link and libraries :

<https://data.caltech.edu/records/mzriq-6wc02>

separate the data into training, validation, and testing sets with a 50%, 25%, 25% split and then structured the directories as follows:

```
/datadir
/train
/class1
/class2
.
.
/valid
/class1
/class2
```

[Type here]

```
.  
.  
/test  
/class1  
/class2  
.  
Libraries required :  
PyTorch  
torchvision import transforms  
torchvision import d  
atsets  
torch.utils.data import DataLoader  
torchvision import models  
torch.nn as nn  
torch import optim
```

Ref: <https://towardsdatascience.com/transfer-learning-with-convolutional-neural-networks-in-pytorch-dd09190245ce>

- m) Prepare the dataset in splitting in three directories Train , alidation and test with 50 25 25
- n) Do pre-processing on data with transform from Pytorch

Training dataset transformation as follows :

```
transforms.Compose([  
    transforms.RandomResizedCrop(size=256, scale=(0.8, 1.0)),  
    transforms.RandomRotation(degrees=15),  
    transforms.ColorJitter(),  
    transforms.RandomHorizontalFlip(),  
    transforms.CenterCrop(size=224), # Image net standards  
    transforms.ToTensor(),  
    transforms.Normalize([0.485, 0.456, 0.406],  
                          [0.229, 0.224, 0.225]) # Imagenet standards
```

Validation Dataset transform as follows :

```
transforms.Compose([  
    transforms.Resize(size=256),  
    transforms.CenterCrop(size=224),  
    transforms.ToTensor(),  
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
```

- o) Create Datasets and Loaders :

```
data = {  
    'train':(Our name given to train data set dir created )  
    datasets.ImageFolder(root=trainindir, transform=image_transforms['train']),  
    'valid':  
    datasets.ImageFolder(root=validdir, transform=image_transforms['valid']),  
}  
dataloaders = {  
    'train': DataLoader(data['train'], batch_size=batch_size, shuffle=True),  
    'val': DataLoader(data['valid'], batch_size=batch_size, shuffle=True)  
}
```

- p) Load Pretrain Model : from torchvision import models

```
model = model.vgg16(pretrained=True)
```

- q) Freez all the Models Weight

```
for param in model.parameters():  
    param.requires_grad = False
```

- r) Add our own custom classifier with following parameters :

```
Fully connected with ReLU activation, shape = (n_inputs, 256)  
Dropout with 40% chance of dropping  
Fully connected with log softmax output, shape = (256, n_classes)  
import torch.nn as nn  
# Add on classifier  
model.classifier[6] = nn.Sequential(  
    nn.Linear(n_inputs, 256),  
    nn.ReLU(),  
    nn.Dropout(0.4),  
    nn.Linear(256, n_classes),  
    nn.LogSoftmax(dim=1))
```

- s) Only train the sixth layer of classifier keep remaining layers off .

```
Sequential(  
    (0): Linear(in_features=25088, out_features=4096, bias=True)  
    (1): ReLU(inplace)  
    (2): Dropout(p=0.5)
```

```
(3): Linear(in_features=4096, out_features=4096, bias=True)
(4): ReLU(inplace)
(5): Dropout(p=0.5)
(6): Sequential(
  (0): Linear(in_features=4096, out_features=256, bias=True)
  (1): ReLU()
  (2): Dropout(p=0.4)
  (3): Linear(in_features=256, out_features=100, bias=True)
  (4): LogSoftmax()
)
)
t) Initialize the loss and optimizer
criterion = nn.NLLLoss()
optimizer = optim.Adam(model.parameters())
u) Train the model using Pytorch
for epoch in range(n_epochs):
    for data, targets in trainloader:
        # Generate predictions
        out = model(data)
        # Calculate loss
        loss = criterion(out, targets)
        # Backpropagation
        loss.backward()
        # Update model parameters
        optimizer.step()
v) Perform Early stopping
w) Draw performance curve
x) Calculate Accuracy
pred = torch.max(ps, dim=1)
equals = pred == targets
# Calculate accuracy
accuracy = torch.mean(equals)
```

Sample Code with comments : Attach Printout with Output .

Conclusion: Explain how Transfer training increases the accuracy of Object detection

<https://www.google.com/url?q=https://towardsdatascience.com/transfer-learning-with-convolutional-neural-networks-in-pytorch-dd0>