38

RA2311047010044

INDEX

NAME K.D.D.SaiAbhiram SUBJECT Deep learning lab obcb

STD. DIV. ROLL NO. SCHOOL

SR. NO.	PAGE NO.	TITLE	DATE	TEACHER'S SIGN / REMARKS
1		Analyze and exploring the deep learning platforms	24-07-25	
2		Implement a classifier using open-source data set	7/8/25	1/1/8/25
3.		Study of the classifiers with respect to statistical parameters	7/8/25	14/8/25
4.		Build a simple feed forward neural network to recognize hand written character	14/8/25	1/14/8
5.		Study of activation function and its role	28-8-25	
6.		Implement gradient descent & back propagation in deep neural network.	09-9-25	9/9
7.		Build a CNN model to classify cats & dogs images	16-9-25	23/9

16/9/25

Week-7 – Build a CNN model to classify cats and dogs images

Aim:- to classify the cats & dogs images using CNN model.

Objective:-

* A convolutional neural network (CNN) is a type of neural network specifically designed to process data with grid like images
* the primary goal of a CNN model using tensorflow to classify images of cats and dogs. by end of this lab.
  1. load and preprocess an image dataset.
  2. construct a CNN architecture.
  3. train the model on the dataset.
  4. Evaluate the model's performance.

Pseudo code:

step: 1 – setup

import ~~tensorflow~~ pytorch, matplotlib

define image (image_width = ? 128, image height = ?)

define batch size =

define epoch = 20

step: 2 – data preparation

initialize training data generator with

- rescale = 1/255
- data agumentation (rotate, shift, zoom, flip)

initialize validation generator flab

load training data & validation data

step: 3 – Build CNN model

Create sequential model

flatten output

Add dense layer (512 units, Relu)

Step: 4 - Compile model

optimizer = Adam
loss = Binary cross entropy
metrics = accuracy

Step: 5 - train model

FIT Model using:

- training generator
- Validation generator
- steps per epoch = $\frac{\text{training-samples}}{\text{batch size}}$
- Validation steps = $\frac{\text{validation-s}}{\text{batch-size}}$ - epochs

Step: 6 :- Evaluate and visualize

Plot training vs validation accuracy per plot.
Plot training vs validation loss per epoch.
Print final validation loss and accuracy

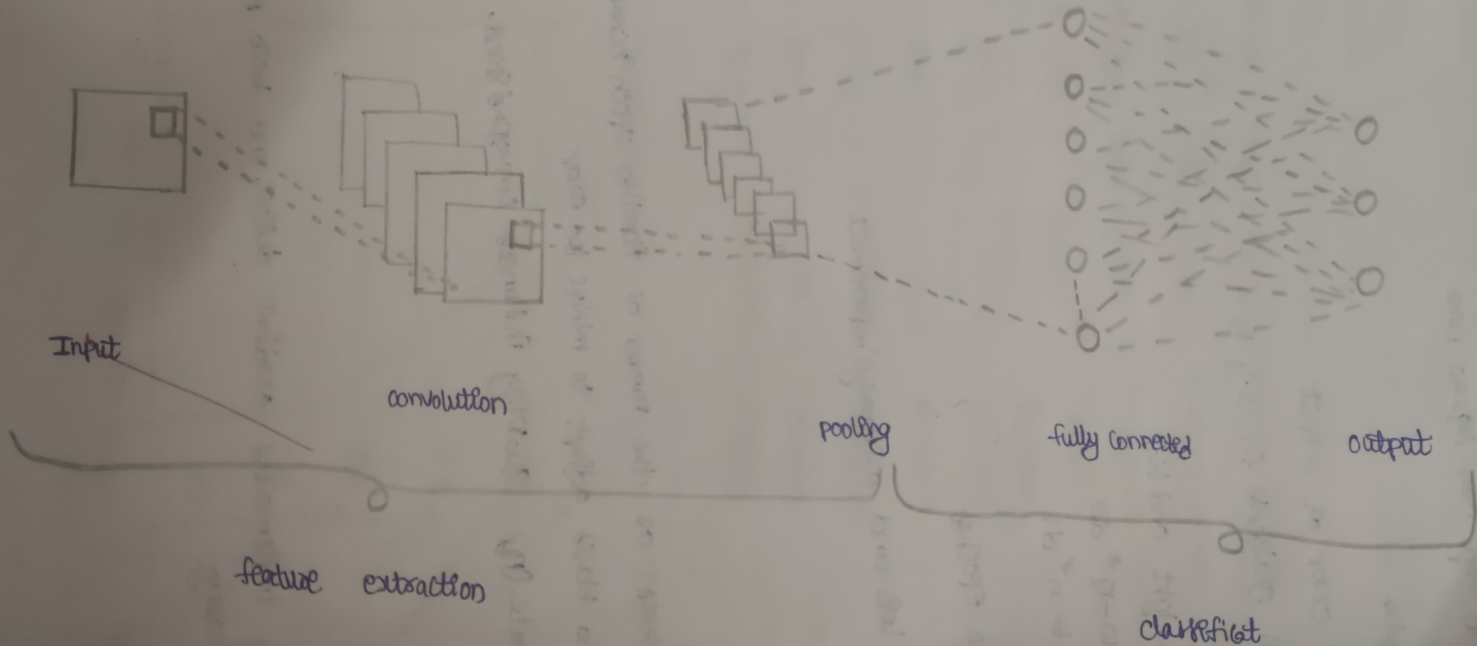
End

Observations :-

1. Loss decreased as the number of epochs increased
2. weights and biases adjusted continuously to minimize the error
3. learning rate influenced the speed of convergence
4. Accuracy gradually improved and reached stable values

Result :- the implementation of CNN model was successfully implemented using ~~tensorflow~~ Python to verify the images of cats & dogs.

CNN Architecture



output

Epoch [1/10] train : loss : 0.6597 , val loss : 0.6277
train : acc : 0.6013 , val acc : 0.6495

Epoch [2/10] t_loss : 0.5968 , val_loss : 0.5953
t-acc : 0.6777 , val-acc : 0.6730

Epoch [3/10] t-loss : 0.5650 , val-loss : 0.5495
t-acc : 0.7017 , val-acc : 0.7185

Epoch [4/10] t-loss : 0.5402 , val-loss : 0.5548
t-acc : 0.7234 , val-acc : 0.7100

Epoch [5/10] t-loss : 0.5133 , val-loss : 0.5238
t-acc : 0.7420 , val-acc : 0.7320

Epoch [6/10] t-loss : 0.4891 , val-loss : 0.5285
t-acc : 0.[illegible] , val-acc : 0.7335

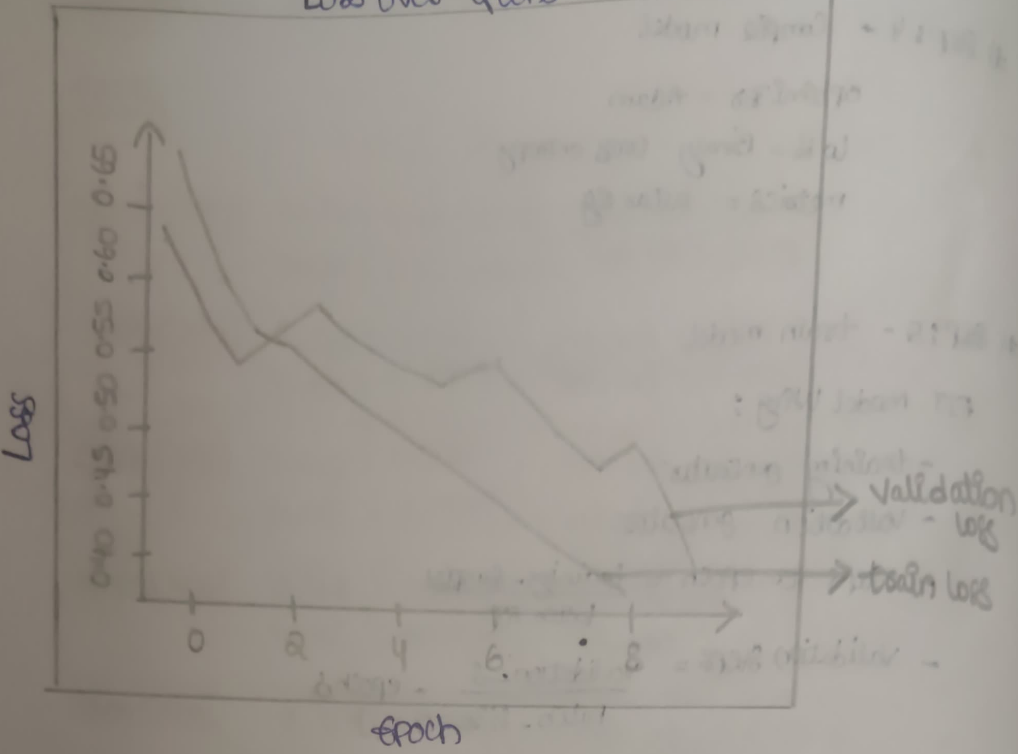
Epoch [7/10] t-loss : 0.4742 , val-loss : 0.5031
t-acc : 0.7696 , val-acc : 0.7450

Epoch [8/10] t-loss : 0.4521 , val-loss : 0.4998
t-acc : 0.7841 , val-acc : 0.7445

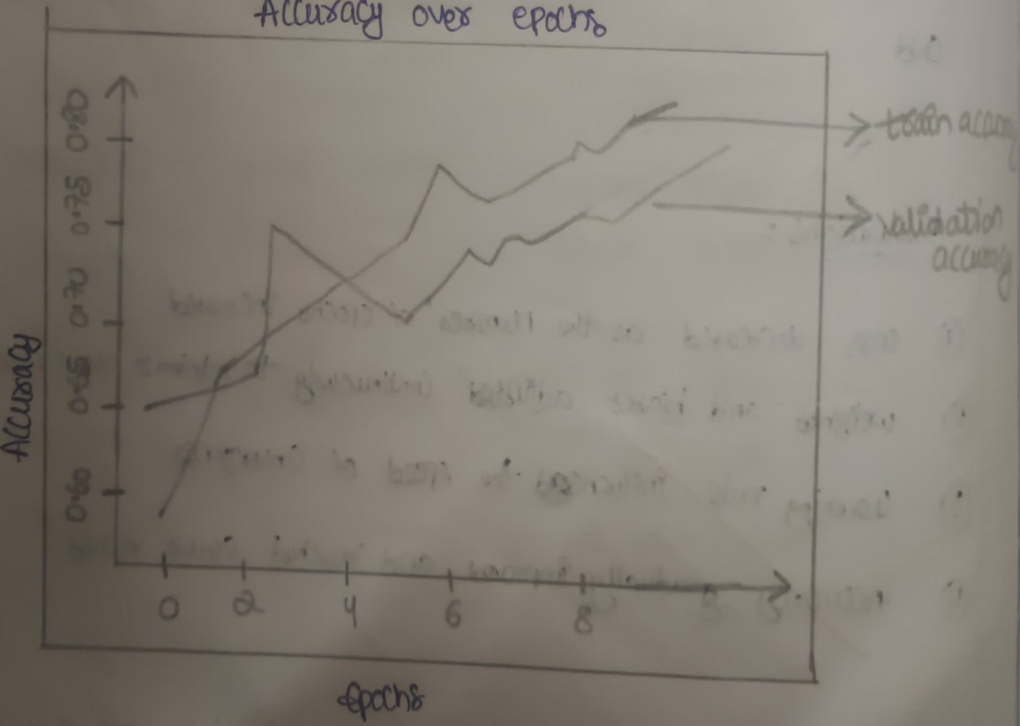
Epoch [9/10] t-loss : 0.4195 , val-loss : 0.5244
t-acc : 0.8031 , val-acc : 0.7420

Epoch [10/10] t-loss : 0.4002 , val-loss : 0.5013
t_acc : 0.8130 , val-acc : 0.7505

Loss over Epochs



Accuracy over epochs



Result :-

the implementation of CNN model was successfully implemented using pytorch to verify the images of cats & dogs

Final validation loss :- 0.5013

final validation accuracy :- 0.7505

S 23/9/23

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, Subset
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import numpy as np

# Configuration
IMG_WIDTH = 32  # CIFAR-10 images are 32x32
IMG_HEIGHT = 32
BATCH_SIZE = 64
EPOCHS = 10
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

#  Data Preparation
transform = transforms.Compose([
    transforms.ToTensor()
])

# Download CIFAR10 dataset
full_train_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
full_valid_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

# Filter dataset: Keep only 'cat' (label 3) and 'dog' (label 5) classes
def filter_cats_dogs(dataset):
    cat_dog_indices = [i for i, (_, label) in enumerate(dataset) if label in [3, 5]]
    return Subset(dataset, cat_dog_indices)

train_dataset = filter_cats_dogs(full_train_dataset)
valid_dataset = filter_cats_dogs(full_valid_dataset)
```

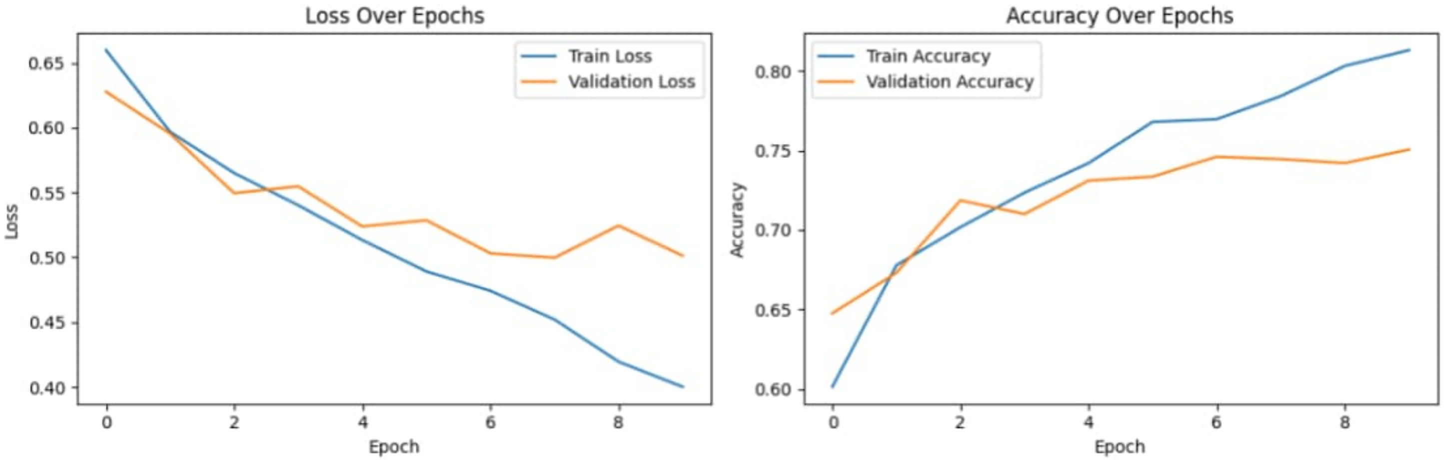
```
    model.eval()
    val_loss, val_correct, val_total = 0.0, 0, 0
    with torch.no_grad():
        for inputs, labels in valid_loader:
            inputs, labels = inputs.to(device), labels.to(device).unsqueeze(1)
            outputs = model(inputs)
            loss = criterion(outputs, labels)

            val_loss += loss.item() * inputs.size(0)
            predicted = (outputs > 0.5).float()
            val_correct += (predicted == labels).sum().item()
            val_total += labels.size(0)

    val_loss /= val_total
    val_acc = val_correct / val_total
    val_losses.append(val_loss)
    val_accuracies.append(val_acc)

    print(f"Epoch [{epoch+1}/{EPOCHS}] "
          f"Train Loss: {train_loss:.4f} | Train Acc: {train_acc:.4f} | "
          f"Val Loss: {val_loss:.4f} | Val Acc: {val_acc:.4f}")

# Visualization
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(train_losses, label='Train Loss')
plt.plot(val_losses, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Loss Over Epochs')

plt.subplot(1, 2, 2)
plt.plot(train_accuracies, label='Train Accuracy')
plt.plot(val_accuracies, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Accuracy Over Epochs')

plt.tight_layout()
plt.show()
```

```python
# Update labels to binary (cat=0, dog=1)
class BinaryCIFAR10(torch.utils.data.Dataset):
    def __init__(self, subset):
        self.subset = subset

    def __len__(self):
        return len(self.subset)

    def __getitem__(self, idx):
        img, label = self.subset[idx]
        binary_label = 0 if label == 3 else 1  # 3 → cat → 0, 5 → dog → 1
        return img, torch.tensor(binary_label, dtype=torch.float32)

train_dataset = BinaryCIFAR10(train_dataset)
valid_dataset = BinaryCIFAR10(valid_dataset)

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
valid_loader = DataLoader(valid_dataset, batch_size=BATCH_SIZE, shuffle=False)

#  Model Definition
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, 3)
        self.fc1 = nn.Linear(64 * 6 * 6, 512)
        self.fc2 = nn.Linear(512, 1)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 64 * 6 * 6)
        x = F.relu(self.fc1(x))
```

```python
model = SimpleCNN().to(device)
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Training Loop
train_losses, val_losses = [], []
train_accuracies, val_accuracies = [], []

for epoch in range(EPOCHS):
    model.train()
    running_loss, correct, total = 0.0, 0, 0

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device).unsqueeze(1)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * inputs.size(0)
        predicted = (outputs > 0.5).float()
        correct += (predicted == labels).sum().item()
        total += labels.size(0)

    train_loss = running_loss / total
    train_acc = correct / total
    train_losses.append(train_loss)
    train_accuracies.append(train_acc)
```

```python
#  Final Evaluation
model.eval()
final_loss, final_correct, final_total = 0.0, 0, 0
with torch.no_grad():
    for inputs, labels in valid_loader:
        inputs, labels = inputs.to(device), labels.to(device).unsqueeze(1)
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        final_loss += loss.item() * inputs.size(0)
        predicted = (outputs > 0.5).float()
        final_correct += (predicted == labels).sum().item()
        final_total += labels.size(0)

final_loss /= final_total
final_accuracy = final_correct / final_total
print(f"\nFinal Validation Loss: {final_loss:.4f}")
print(f"Final Validation Accuracy: {final_accuracy:.4f}")
```

```
100%|██████████| 170M/170M [00:06<00:00, 26.6MB/s]
Epoch [1/10] Train Loss: 0.6597 | Train Acc: 0.6013 | Val Loss: 0.6277 | Val Acc: 0.6475
Epoch [2/10] Train Loss: 0.5968 | Train Acc: 0.6777 | Val Loss: 0.5953 | Val Acc: 0.6730
Epoch [3/10] Train Loss: 0.5650 | Train Acc: 0.7017 | Val Loss: 0.5495 | Val Acc: 0.7185
Epoch [4/10] Train Loss: 0.5402 | Train Acc: 0.7234 | Val Loss: 0.5548 | Val Acc: 0.7100
Epoch [5/10] Train Loss: 0.5133 | Train Acc: 0.7420 | Val Loss: 0.5238 | Val Acc: 0.7310
Epoch [6/10] Train Loss: 0.4891 | Train Acc: 0.7679 | Val Loss: 0.5285 | Val Acc: 0.7335
Epoch [7/10] Train Loss: 0.4742 | Train Acc: 0.7696 | Val Loss: 0.5031 | Val Acc: 0.7460
Epoch [8/10] Train Loss: 0.4521 | Train Acc: 0.7841 | Val Loss: 0.4998 | Val Acc: 0.7445
Epoch [9/10] Train Loss: 0.4195 | Train Acc: 0.8031 | Val Loss: 0.5244 | Val Acc: 0.7420
Epoch [10/10] Train Loss: 0.4002 | Train Acc: 0.8130 | Val Loss: 0.5013 | Val Acc: 0.7505
```




```
Final Validation Loss: 0.5013
Final Validation Accuracy: 0.7505
```