D. Khaja Nawaz

RA2311047010037

Deep Learning Techniques (LaB)

Date	Title	Sign
24/07/2025	1. Exploring the deep learning platform	
31/07/25	2. Implement a classifier using open-source dataset	7/8/25
31/07/25	3. Study of the classifiers with respect to statistical parameters	14/8/25
14/08/25	4. Build a simple feed forward neural network to recognize handwritten character	14/8/25
22/08/25	5. Study of Activation functions and their role	9/9
09/09/25	6. Implement gradient descent and backpropagation in deep neural network	
16/09/25	7 Build a CNN model to classify Cat and Dog image	23/9/25

16/09/25 LAB : 7 Build a CNN model to classify cat and image

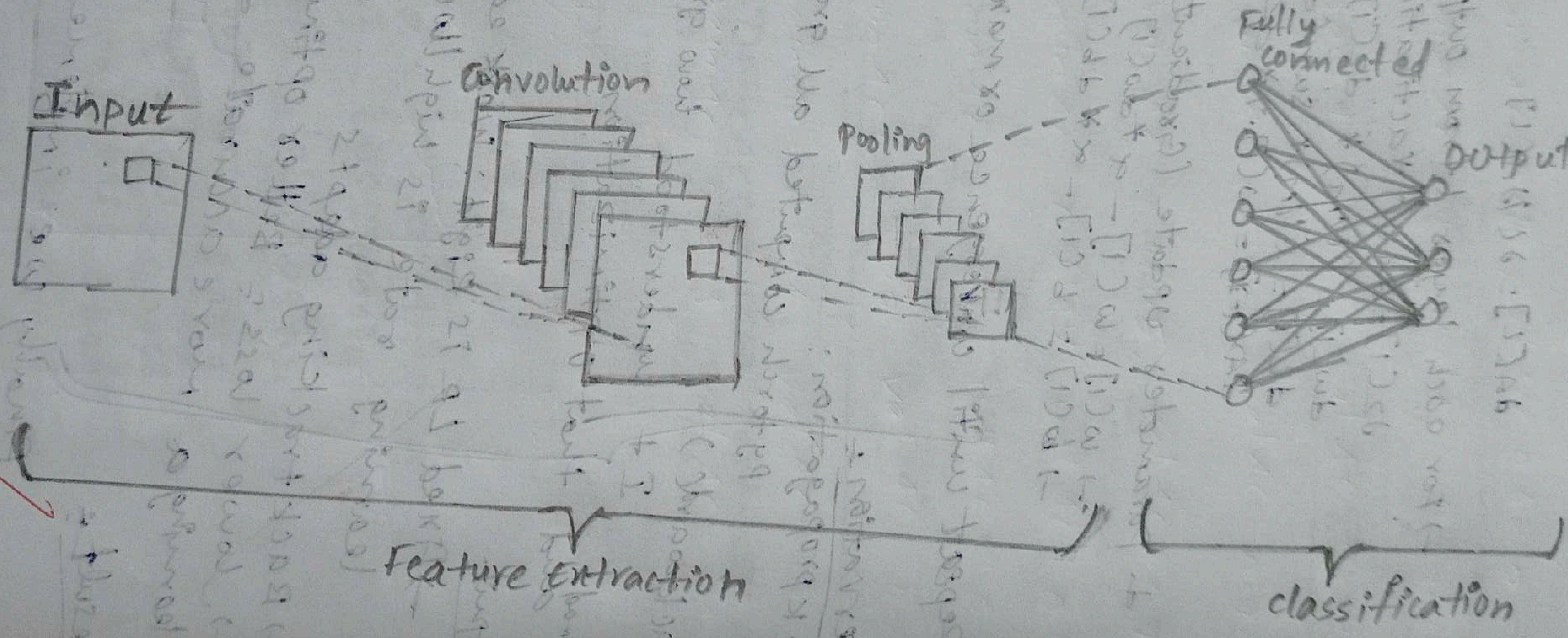
* **Aim** :
To build and train a convolutional Neural network (CNN) model that classifies images into cat or Dog categories.

* **Objective** :
1. To understand the working CNN for image classification
2. To preprocess and Nomalize image data for efficient learning.
3. To design and implement a CNN architecture with convolutional, pooling and fully connected layers
4. To train the CNN model on the cat and Dog dataset
5. To evaluate the model using accuracy and loss Metrics.

* **Pseudocode** :
1. Import required libraries (Tensor Flow/keras Numpy, Matplotlib).
2. Load dataset (catvsdogs).
3. preprocess data:
   → Nomalize pixel values (0-1).
   → split into train and test sets.

4. Define model:
   → Input layer (image size)
   → Convolutional layer + Relu activation.
   → Maxpooling layer
   → Repeat conv + pooling layers to extra Features.
   → Flatten layer
   → Fully connected Dense layer with Relu

CNN Architecture



5. compile model with:

→ loss function = Binary crossentropy

→ optimizer = adam

→ Metrics = Accuracy

6. Train model on training data.
7. Validate using test/validation data.
8. Evaluate model performance (accuracy, Loss)
9. Save the model for future predictions

End

***observation:**

→ The CNN model successfully learns to differentiate cats and dogs after multiple epochs.

→ Data augmentation (Flip, zoom, rotation) improvig generalization

→ Training accuracy gradullay increases and validation accuracy stabilizes around 85-95% depending on dataset size and epochs.

→ overfitting can occur if training too long without droupa or augmentation

***Result:**

successfully implemented cats vs dog classification using CNN.

Output:-

Epoch 1/10 - Train AC: 0.6116, Vall ACC: 0.6220

Epoch 2/10 - Train AC: 0.6790, Val AC: 0.6960

Epoch 3/10 - Train AC: 0.7147, Val AC: 0.7160

Epoch 4/10 - Train AC: 0.7455, val AC: 0.7300

Epoch 5/10 - Train AC: 0.7645, Val AC: 0.7380

Epoch 6/10 - Train AC: 0.7804, val AC: 0.7415

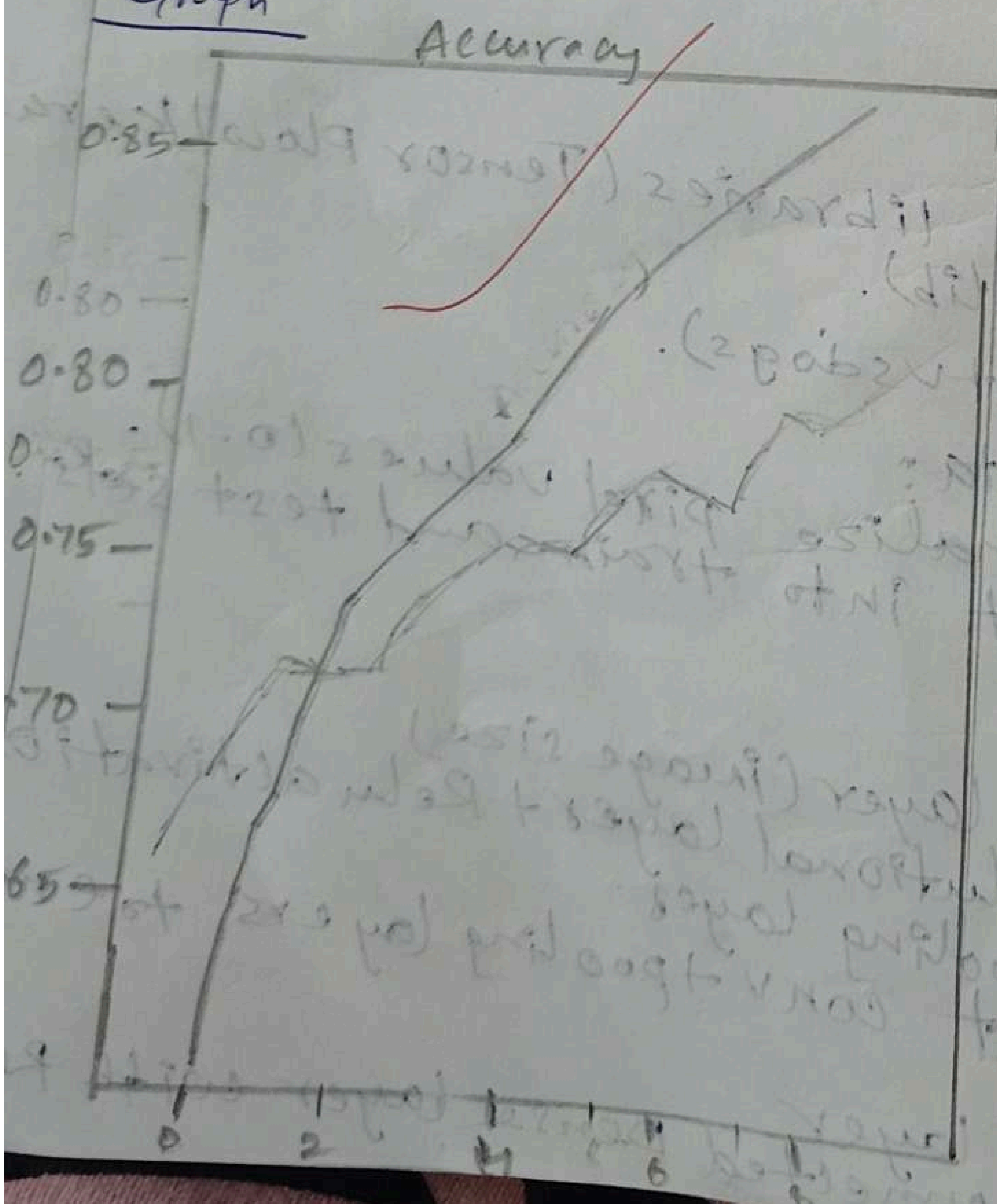
Epoch 7/10 - Train AC: 0.8009, vall AC: 0.7325

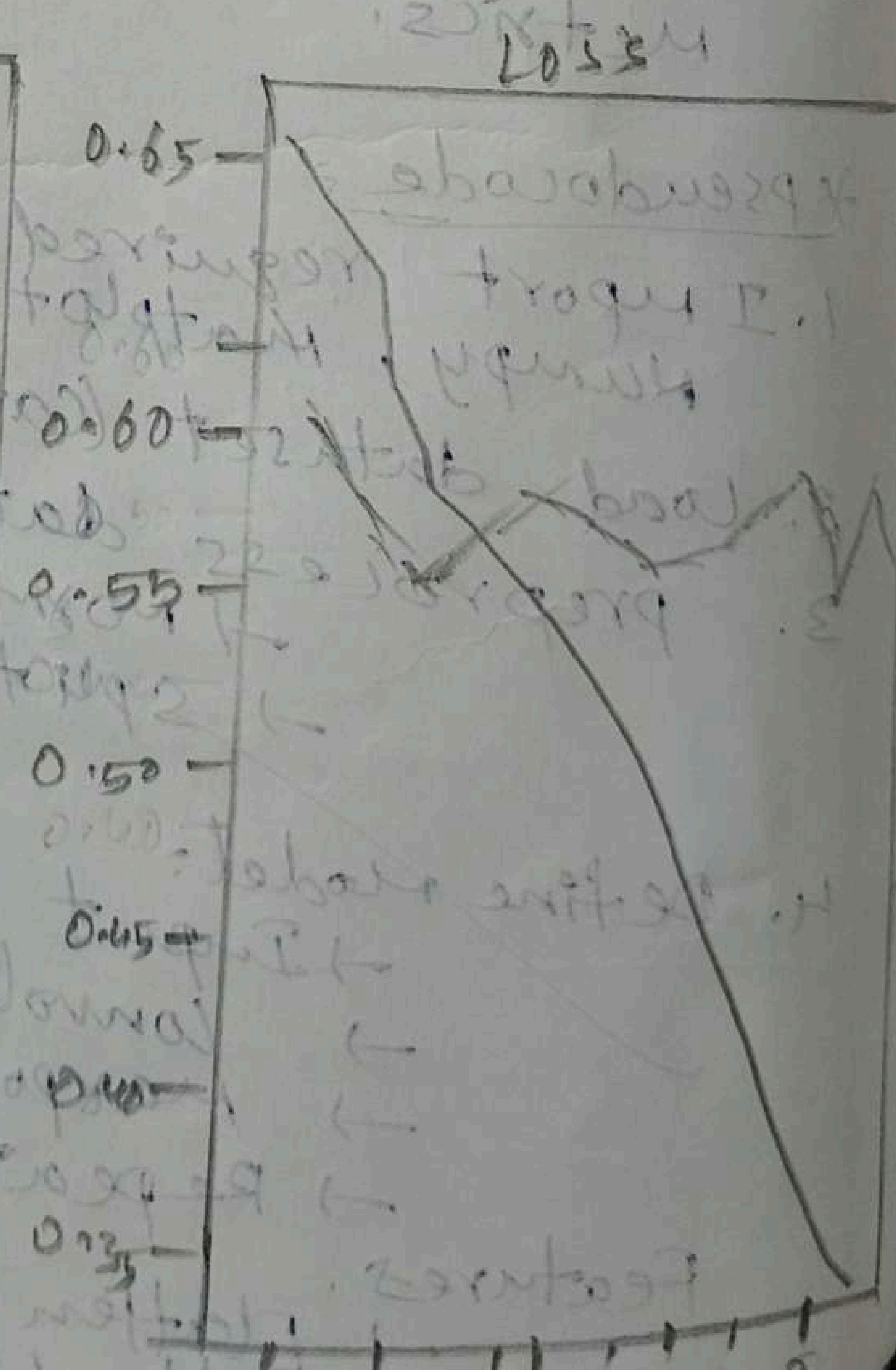
Epoch 8/10 - Train AC: 0.8145, val AC: 0.7370

Epoch 9/10 - Train AC: 0.8266, Val AC: 0.7455

Epoch 10/10 - Train AC: 0.8558, val AC: 0.7510

Graph





Untitled25.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▷ Run all

[1]

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

# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

# ----------------------------
# STEP 1: Load and Prepare Dataset
# ----------------------------
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

train_dataset_full = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
test_dataset_full = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

# Filter for cats (3) and dogs (5)
def filter_cat_dog(dataset):
    idx = [i for i, (_, label) in enumerate(dataset) if label in [3, 5]]
```

```python
# Remap labels: cat (3) -> 0, dog (5) -> 1
for i in range(len(train_dataset)):
    img_idx = train_dataset.indices[i]
    if train_dataset_full.targets[img_idx] == 3:
        train_dataset_full.targets[img_idx] = 0
    elif train_dataset_full.targets[img_idx] == 5:
        train_dataset_full.targets[img_idx] = 1

for i in range(len(test_dataset)):
    img_idx = test_dataset.indices[i]
    if test_dataset_full.targets[img_idx] == 3:
        test_dataset_full.targets[img_idx] = 0
    elif test_dataset_full.targets[img_idx] == 5:
        test_dataset_full.targets[img_idx] = 1

# Split train set for validation
n_train = int(len(train_dataset) * 0.8)
n_val = len(train_dataset) - n_train
train_subset, val_subset = torch.utils.data.random_split(train_dataset, [n_train, n_val])

batch_size = 64
train_loader = DataLoader(train_subset, batch_size=batch_size, shuffle=True, num_workers=2)
val_loader = DataLoader(val_subset, batch_size=batch_size, shuffle=False, num_workers=2)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False, num_workers=2)

print(f"Train size: {len(train_subset)}, Val size: {len(val_subset)}, Test size: {len(test_dataset)}")

# -----------------------------
# STEP 2: CNN Model
# -----------------------------
class CatDogCNN(nn.Module):
```

File Edit View Insert Runtime Tools Help

ommands + Code + Text Run all

[1]
3m

```python
class CatDogCNN(nn.Module):
    def __init__(self):
        super(CatDogCNN, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Flatten()
        )
        self.classifier = nn.Sequential(
            nn.Linear(64 * 8 * 8, 128),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(128, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x

model = CatDogCNN().to(device)
print(model)

# ----------------------------
# STEP 3: Train
# ----------------------------
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
epochs = 10
```

What can I help you build?

```python
for epoch in range(epochs):
    # Train
    model.train()
    running_loss, running_corrects = 0.0, 0.0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.float().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)
        preds = (outputs > 0.5).float()
        running_corrects += torch.sum(preds == labels.data)

    train_loss = running_loss / len(train_subset)
    train_acc = running_corrects.double() / len(train_subset)
    train_losses.append(train_loss)
    train_accs.append(train_acc.item())

    # Validate
    model.eval()
    running_loss, running_corrects = 0.0, 0.0
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.float().to(device).unsqueeze(1)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            running_loss += loss.item() * inputs.size(0)
            preds = (outputs > 0.5).float()
            running_corrects += torch.sum(preds == labels.data)
```

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▷ Run all

[1]
✓ 3m

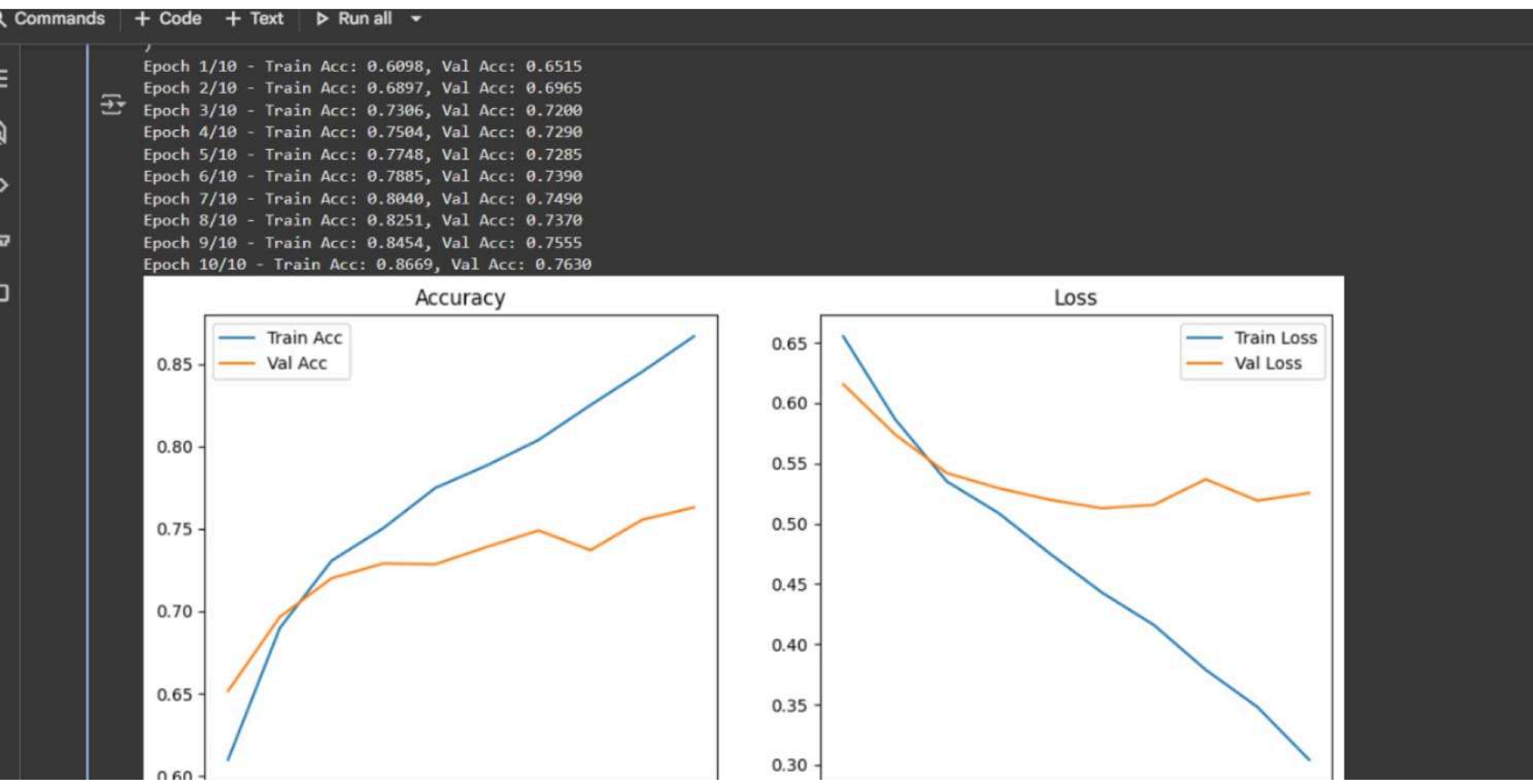
```
    val_loss = running_loss / len(val_subset)
    val_acc = running_corrects.double() / len(val_subset)
    val_losses.append(val_loss)
    val_accs.append(val_acc.item())

    print(f"Epoch {epoch+1}/{epochs} - Train Acc: {train_acc.item():.4f}, Val Acc: {val_acc.item():.4f}")

# ----------------------------
# STEP 4: Graphs
# ----------------------------
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(train_accs, label="Train Acc")
plt.plot(val_accs, label="Val Acc")
plt.legend(); plt.title("Accuracy")

plt.subplot(1, 2, 2)
plt.plot(train_losses, label="Train Loss")
plt.plot(val_losses, label="Val Loss")
plt.legend(); plt.title("Loss")
plt.show()

# ----------------------------
# STEP 5: Evaluate
# ----------------------------
model.eval()
corrects = 0
with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.float().to(device).unsqueeze(1)
        outputs = model(inputs)
        preds = (outputs > 0.5).float()
        corrects += torch.sum(preds == labels.data)
```

Commands + Code + Text ▷ Run all

[1]
3m

```python
# STEP 6: Prediction Example
# ----------------------------
def imshow(img):
    img = img / 2 + 0.5  # unnormalize
    npimg = img.cpu().numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.axis('off')

dataiter = iter(test_loader)
images, labels = next(dataiter)

idx = random.randint(0, images.size(0) - 1)
sample_img = images[idx]
sample_label = "Dog" if labels[idx] == 1 else "Cat"

with torch.no_grad():
    output = model(sample_img.unsqueeze(0).to(device))
    pred = output.item()
pred_label = "Dog" if pred > 0.5 else "Cat"

imshow(sample_img)
plt.title(f"True: {sample_label}, Pred: {pred_label}")
plt.show()
```

```
Using device: cpu
100%|██████████| 170M/170M [00:04<00:00, 39.4MB/s]
Train size: 8000, Val size: 2000, Test size: 2000
CatDogCNN(
  (features): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```


Commands
+ Code
+ Text
Run all
Epoch 1/10 - Train Acc: 0.6098, Val Acc: 0.6515
Epoch 2/10 - Train Acc: 0.6897, Val Acc: 0.6965
Epoch 3/10 - Train Acc: 0.7306, Val Acc: 0.7200
Epoch 4/10 - Train Acc: 0.7504, Val Acc: 0.7290
Epoch 5/10 - Train Acc: 0.7748, Val Acc: 0.7285
Epoch 6/10 - Train Acc: 0.7885, Val Acc: 0.7390
Epoch 7/10 - Train Acc: 0.8040, Val Acc: 0.7490
Epoch 8/10 - Train Acc: 0.8251, Val Acc: 0.7370
Epoch 9/10 - Train Acc: 0.8454, Val Acc: 0.7555
Epoch 10/10 - Train Acc: 0.8669, Val Acc: 0.7630
Accuracy
Train Acc
Val Acc
0.85
0.80
0.75
0.70
0.65
0.60
Loss
Train Loss
Val Loss
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30