Exp 17.

BUILD A CNN MODEL TO CLASSIFY CAT & DOG IMAGE.

16.09.25

Aim: To implement CNN to build a CNN model to classify a cat & dog image.

PSEUDO CODE:

- **Initialize parameters**
  - Define input of shape = (64, 64, 3) → image resolution chosen for uniformity.
  - Define no. of classes = 2 → binary classification (Cat, Dog)
  - Set batch size, learning rate & epochs. (These hyperparameters control how the model learns).
- **Load dataset**
  - Import cat & dog image from dataset
  - Preprocess: resize all images to same dimension.
  - Normalize pixels values [0, 1] to stabilize training.
  - Split into training set (for learning) & Validation set (for performance check)
- Data Augmentation
  - Apply random flips, rotation, zooms & shifts (Increases diversity of training images & prevents overfitting)
- Construct CNN architecture.
  - Convolution layer: Applies filters to capture local spatial features (eg: edges, corners, textures of cats/dogs)
  - Activation function (ReLU): Introduces non-linearity, allowing complex feature learning.
  - Pooling layer: reduces spatial dimension.
  - Repeat convolution + pooling block to learn higher-level features.

- Compile model.
  - optimizer = Adam (adaptive learning, efficient convergence)
  - loss function = Binary cross Entropy.
  - Evaluation metric = Accuracy.

- Train model.
  - feed training set in batches.
  - Validate on validation set after each epoch.
  - Repeat for defined no. of epochs.

- Evaluate model.
  - Calculate accuracy & loss on validation set.

- Prediction.
  - For a new image:
    → Resize to (64, 64)
    → Normalize pixel values.
    → Pass through trained CNN.
    → If output < 0.5 → Cat
      Else → Dog.

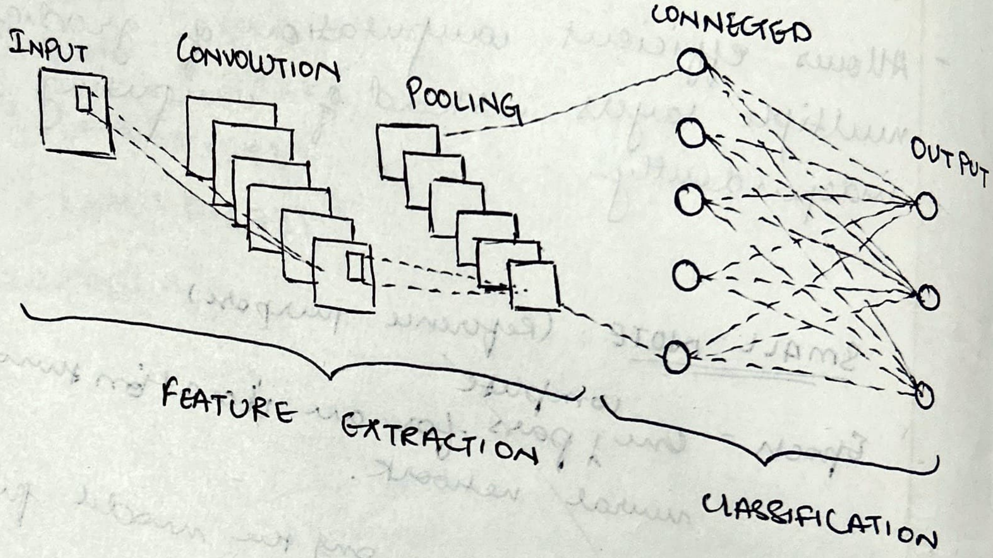
Justification.

→ Why CNN?

- Images have spatial patterns (edges, colors, textures, shapes).
- Specifically designed to capture these local patterns by applying filters (kernels).

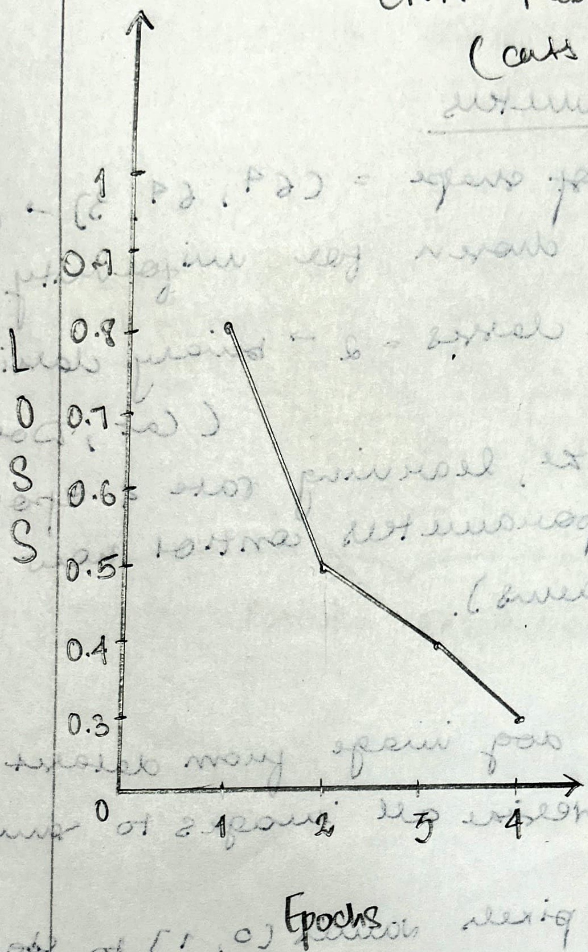
→ Why multiple convolution + pooling layers?

CNN ARCHITECTURE DIAGRAM.



OBSERVATION:

CNN training loss curve (cats v/s dogs).



Table

Epochs	training accuracy	validation accuracy	loss	notes.
1	65%	60%	0.8	Model starts learning
3	80%	75%	0.5	Accuracy improving
5	88%	82%	0.4	Overfitting not yet seen
10	92%	85%	0.3	Good gradi-entation.

```
model = SimpleCNN(img_size=IMG_SIZE).to(device)
print(model)
```

```
SimpleCNN(
  (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))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU()
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=41472, out_features=512, bias=True)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=512, out_features=1, bias=True)
    (5): Sigmoid()
  )
)
```

```
# Loss and optimizer
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
#Training loop
for epoch in range(NUM_EPOCHS):
    model.train()
    running_loss = 0.0
```

```
        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)

    epoch_loss = running_loss / len(train_dataset)
    epoch_acc = running_corrects.double() / len(train_dataset)

    print(f'Epoch {epoch+1}/{NUM_EPOCHS} - Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')
```

```
Epoch 1/10 - Loss: 0.7142 Acc: 0.5440
Epoch 2/10 - Loss: 0.6549 Acc: 0.6070
Epoch 3/10 - Loss: 0.6254 Acc: 0.6640
Epoch 4/10 - Loss: 0.5928 Acc: 0.6765
Epoch 5/10 - Loss: 0.5815 Acc: 0.6960
Epoch 6/10 - Loss: 0.5513 Acc: 0.7260
Epoch 7/10 - Loss: 0.5089 Acc: 0.7570
Epoch 8/10 - Loss: 0.4831 Acc: 0.7670
Epoch 9/10 - Loss: 0.4517 Acc: 0.7870
Epoch 10/10 - Loss: 0.4135 Acc: 0.8075
```

```
#validation
model.eval()
val_corrects = 0

with torch.no_grad():
    for inputs, labels in val_loader:
        inputs = inputs.to(device)
```