16-9-25

LAB-7 Build a Convolutional Neural Network to classify images of Cat or dog

Aim: Build a CNN to classify eighter the given image is Cat Or dog.

Objectives:

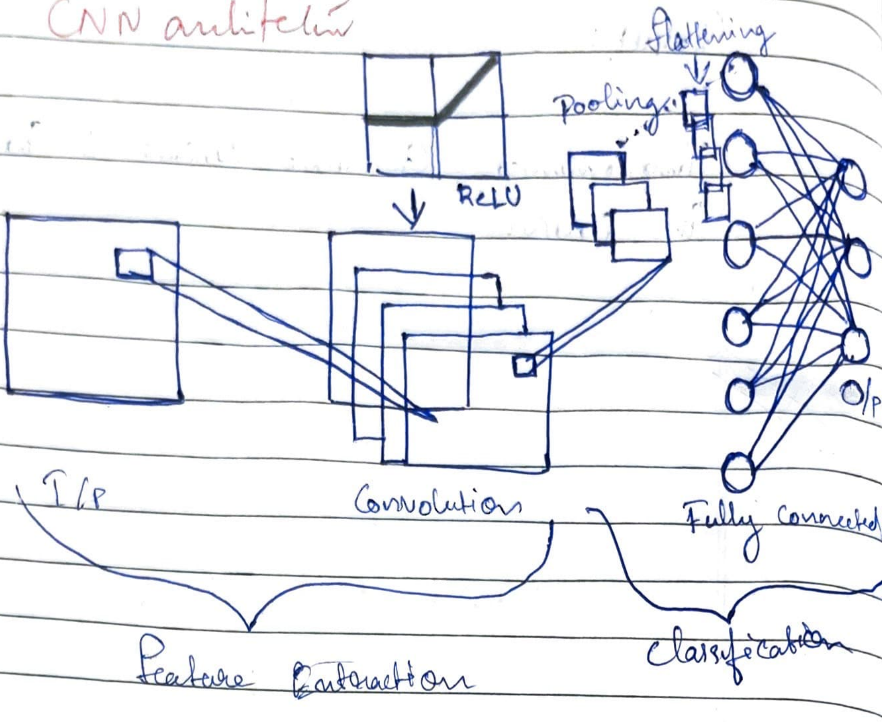
- Preprocess and augment image data.
- Design and train a CNN Model to learn features distinguishing Cats and dogs.
- Evaluate model performance on unseen test data.
- Achieve high accuracy and minimize overfitting

Pseudo Code:

- ~~train~~ load and preprocess images.
- Define CNN model:
  - Conv layer + ReLU
  - Max Pooling
  - Conv layer + ReLU
  - Max pooling
  - Flatten
  - Dense layer + ReLU
  - Output layer with Sigmoid.
- Compile model with Optimizer and loss.
- Train model on training data.
- Evaluate model on test data.
- ~~Bui~~

Page
Date

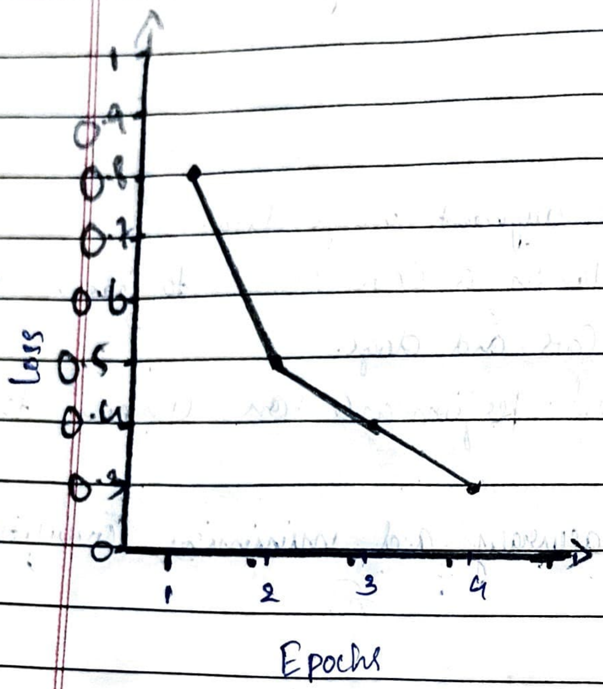
CNN architecture



Page
Date

Loss Curve

CNN Training Loss Curve (Cats Vs Dogs)



• print accuracy.

ReLU: $f(x) = max(0, x)$

Observation:

- CNN learns key features like shapes & edges automatically
- Training and validation accuracy improve over epochs.
- Small gap b/w training and validation accuracy means good generalization.
- Pooling layers help reduce overfitting.
- Overfitting occurs if validation accuracy stops improving while training accuracy rises.
- Final accuracy shows how well the model classify cats & dogs.

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 generalization

Result:

The Model achieved around 85% accuracy in classifying cat and dog images, demonstrating effective feature learning and good generalization.

[ ]

```
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
```

CO Untitled0.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

Share M

Connect

[ ]

```
            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)
```

Variables Terminal