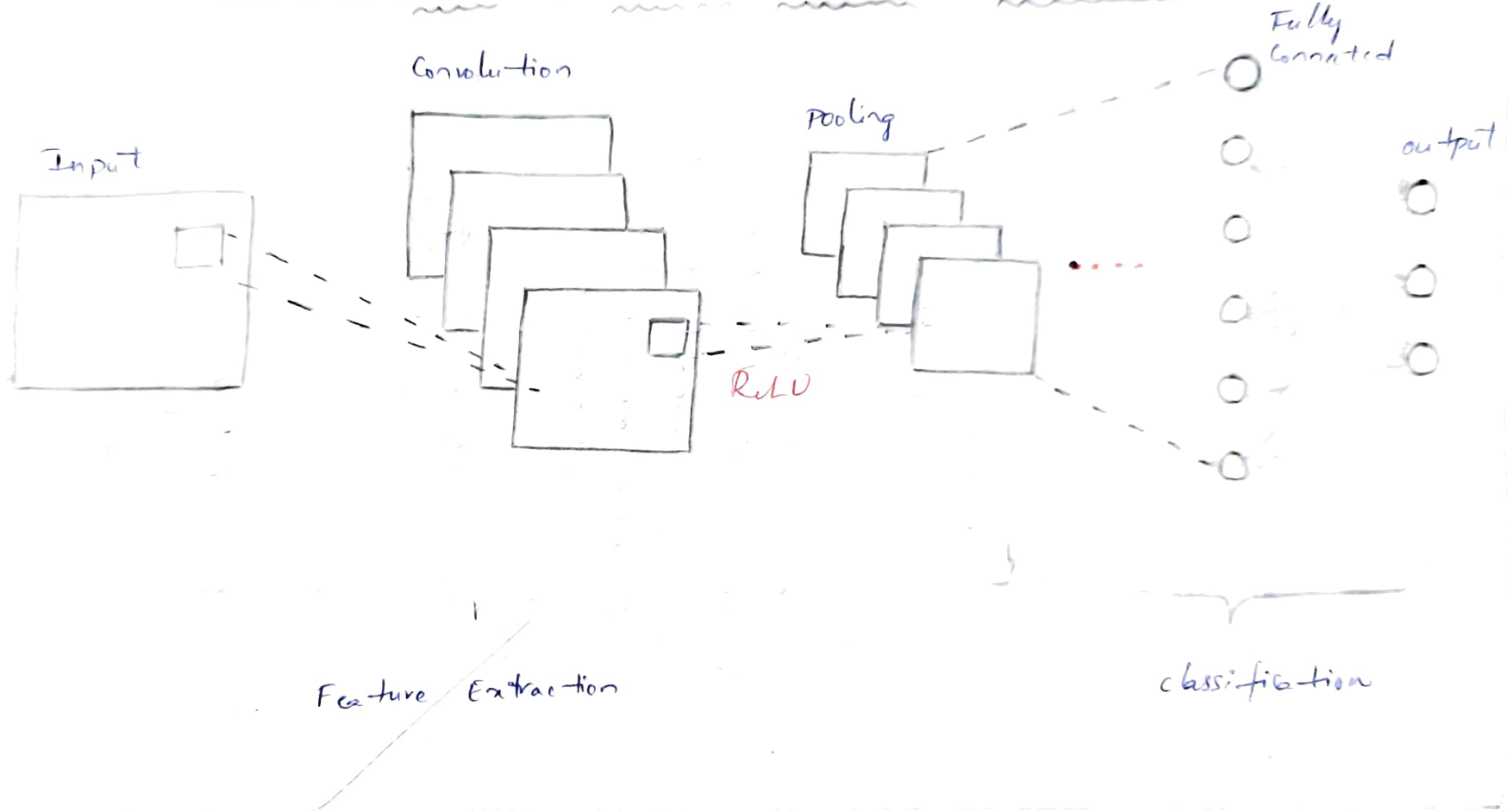
CNN ARCHITECTURE



16/9/25

EXP NO: 7: Implementation of CNN to classify Cat vs Dog images

Aim:-

To design and implement a Convolutional neural network that classifies images of cats and dogs with high accuracy.

Objectives:-

1. preprocess the dataset of Cat & dog images.
2. Design a CNN architecture for feature extraction and classification
3. Train and validate the model using training and validation datasets
4. Evaluate the model on unseen test images.
5. Analyze the performance and derive conclusions.

Algorithm:-

1. Import required libraries
2. Load the dataset (cat vs dog images)
3. preprocess the data (resize, normalize, augment).
4. split dataset into training and validation sets.
5. Build CNN architecture:
   - Convolution → ReLU → pooling layers
   - Fully connected Dense layers
   - output layer with softmax
6. Compile the model with loss function = binary

crossentropy and optimizer = Adam

7. Train the model on training data, validate on validation data

8. Evaluate model performance on test data

9. Make predictions on new images

Pseudo Code:-

```
BEGIN
Import libraries (TensorFlow, keras, etc.)
Load dataset of cats and dogs
preprocess images
split into train and validation sets.
DEFINE CNN model:
    Conv2D + ReLU + MaxPooling
    Conv2D + ReLU + MaxPooling
    Flatten
    Dense (fully connected) + ReLU
    Dense (1, sigmoid)
Compile model with Adam optimizer and
binary-crossentropy loss
Train model on training set
Validate model on validation set
Evaluate accuracy on test set
Display training/validation accuracy and loss
Make predictions on sample images
END.
```

Observation:-

- The CNN model successfully learns to differentiate cats and dogs after multiple epochs.
- Data augmentation (flip, zoom, rotation) improves generalization
- Training accuracy gradually increases, and validation accuracy stabilizes around 85-95% depending on dataset size and epochs.
- Overfitting can occur if training too long without dropout or augmentation

Conclusion:-

The designed CNN effectively classifies cat and dog images with high accuracy. By using convolution layers for feature extraction and dense layers for classification, the model learns important patterns like fur feature, ear shape, and face structure. This approach demonstrates the power of CNNs in image classification tasks and can be extended to other object recognition problems.

Outputs:

Epoch 1/10 - Train Acc: 0.6116, Val Acc: 0.6220

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

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

Epoch 4/10 - Train Acc: 0.7455, Val Acc: 0.7300

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

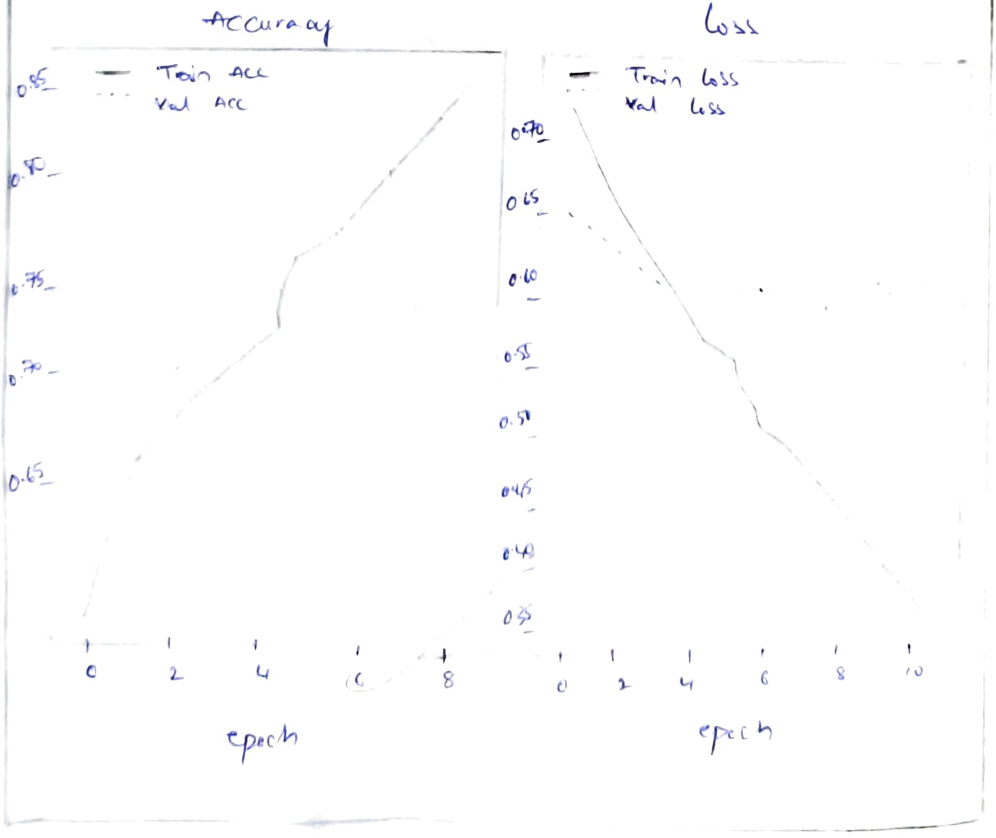
Epoch 6/10 - Train Acc: 0.7804, Val Acc: 0.7415

Epoch 7/10 - Train Acc: 0.8009, Val Acc: 0.7325

Epoch 8/10 - Train Acc: 0.8145, Val Acc: 0.7370

Epoch 9/10 - Train Acc: 0.8366, Val Acc: 0.7455

Epoch 10/10 - Train Acc: 0.8558, Val Acc: 0.7540



[3] ✓ 3m

```
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 = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
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)
def filter_cat_dog(dataset):
    idx = [i for i, (_, label) in enumerate(dataset) if label in [3, 5]]
    return Subset(dataset, idx)
train_dataset = filter_cat_dog(train_dataset_full)
test_dataset = filter_cat_dog(test_dataset_full)
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
n_train = int(len(train_dataset) * 0.8)
n_val = len(train_dataset) - n_train
```

[3] ✓ 3m

```
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)}")
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)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
epochs = 10
train_losses, val_losses, train_accs, val_accs = [], [], [], []
for epoch in range(epochs):
```

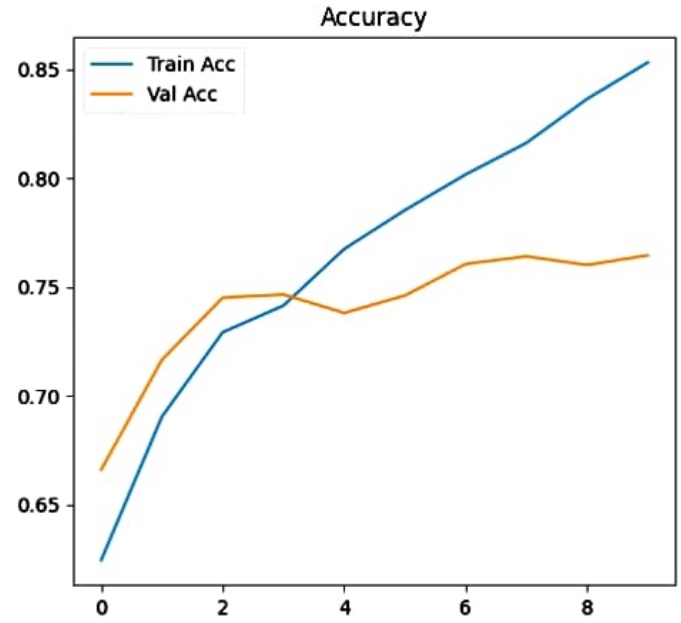
[3] ✓ 3m

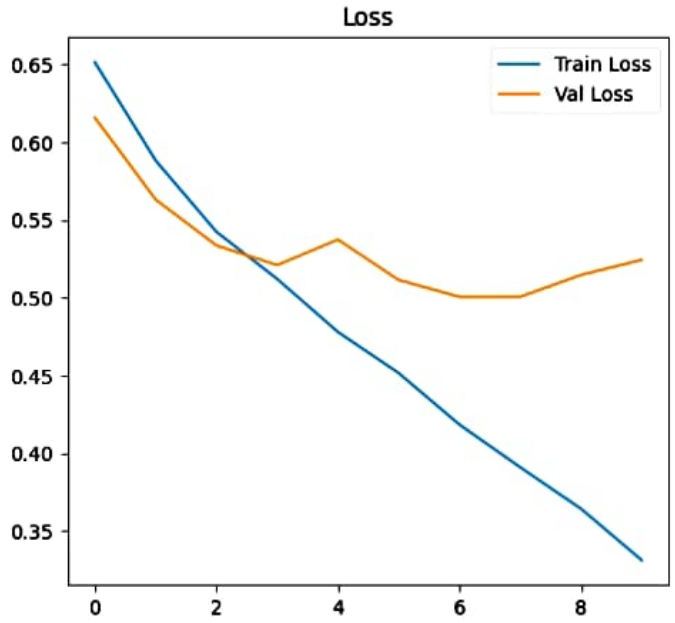
```
for epoch in range(epochs):
    model.train()
    running_loss, running_corrects = 0.0, 0.0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.float().to(device).unsqueeze(1)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, 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())
    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)
    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}")
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")
```

[3] ✓ 3m

```
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()
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)
test_acc = corrects.double() / len(test_dataset)
print(f"Test Accuracy: {test_acc.item()*100:.2f}%")
def imshow(img):
    img = img / 2 + 0.5
    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
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))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Flatten(start_dim=1, end_dim=-1)
  )
  (classifier): Sequential(
    (0): Linear(in_features=4096, out_features=128, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=128, out_features=1, bias=True)
    (4): Sigmoid()
  )
)
Epoch 1/10 - Train Acc: 0.6245, Val Acc: 0.6660
Epoch 2/10 - Train Acc: 0.6904, Val Acc: 0.7165
Epoch 3/10 - Train Acc: 0.7291, Val Acc: 0.7450
Epoch 4/10 - Train Acc: 0.7414, Val Acc: 0.7465
Epoch 5/10 - Train Acc: 0.7674, Val Acc: 0.7380
Epoch 6/10 - Train Acc: 0.7853, Val Acc: 0.7460
Epoch 7/10 - Train Acc: 0.8016, Val Acc: 0.7605
Epoch 8/10 - Train Acc: 0.8161, Val Acc: 0.7640
Epoch 9/10 - Train Acc: 0.8364, Val Acc: 0.7600
Epoch 10/10 - Train Acc: 0.8531, Val Acc: 0.7645
```






```
Test Accuracy: 76.40%
```