[ ]
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
```

[ ]
```
transform = transforms.Compose([
    transforms.Resize((128,128)),   # resize images
    transforms.ToTensor(),          # convert to tensor
    transforms.Normalize((0.5,), (0.5,))  # normalize
])
```

[ ]
```
data_dir = "/content/drive/MyDrive/PetImages"
```

[ ]
```
Start coding or generate with AI.
```

[ ]
```
!ls /content/drive/MyDrive/PetImages
!ls /content/drive/MyDrive/PetImages/Cat | head
!ls /content/drive/MyDrive/PetImages/Dog | head
```

```
!ls /content/drive/MyDrive/PetImages
!ls /content/drive/MyDrive/PetImages/Cat | head
!ls /content/drive/MyDrive/PetImages/Dog | head
```

```
Cat  Dog
0.jpg
10000.jpg
10001.jpg
10002.jpg
10003.jpg
10004.jpg
10005.jpg
10006.jpg
10007.jpg
10008.jpg
0.jpg
10000.jpg
10001.jpg
10002.jpg
10003.jpg
10004.jpg
10005.jpg
10006.jpg
10007.jpg
10008.jpg
```

```
from torchvision import datasets
from torchvision import transforms
from PIL import Image
import os
transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor()
])

def pil_loader(path):
    try:
        with open(path, 'rb') as f:
            img = Image.open(f)
            return img.convert('RGB')
    except Exception as e:
        print("Skipping corrupted file:", path)
        return None
```

```
data_dir = "/content/drive/MyDrive/PetImages"
```

```
!ls /content/drive/MyDrive/PetImages
```

```
Cat  Dog
```

```
!ls /content/drive/MyDrive/PetImages/Dog | wc -l
```

```
3454
```

```
!ls /content/drive/MyDrive/PetImages/Cat/ | wc -l
```

```
5644
```

```
from torchvision import datasets, transforms

dataset = datasets.ImageFolder(root=data_dir, transform=transform)
```

```
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])
```

```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

```
print("Classes:", dataset.classes)
print("Training samples:", len(train_dataset))
print("Testing samples:", len(test_dataset))
```

```
Classes: ['Cat', 'Dog']
Training samples: 7278
Testing samples: 1820
```

```
class SimpleCNN(nn.Module):
    def __init__(self, num_classes=2):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.fc1 = nn.Linear(64 * 
        self.fc2 = nn.Linear(128, num_classes)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = x.view(x.size(0), -1)  # Flatten
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

(parameter) num_classes: int

led twice → 32x32

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SimpleCNN(num_classes=len(dataset.classes)).to(device)
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SimpleCNN(num_classes=len(dataset.classes)).to(device)
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
epochs = 5
from tqdm import tqdm

for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    for images, labels in tqdm(train_loader, desc=f"Epoch {epoch+1}/{epochs}"):
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f"Epoch {epoch+1}/{epochs}, Loss: {running_loss/len(train_loader):.4f}")
```

```
Epoch 1/5: 100%|██████████| 228/228 [31:57<00:00,  8.41s/it]
Epoch 1/5, Loss: 0.6830
Epoch 2/5: 100%|██████████| 228/228 [00:42<00:00,  5.41it/s]
Epoch 2/5, Loss: 0.6382
Epoch 3/5: 100%|██████████| 228/228 [00:41<00:00,  5.43it/s]
Epoch 3/5, Loss: 0.6020
Epoch 4/5: 100%|██████████| 228/228 [00:42<00:00,  5.39it/s]
Epoch 4/5, Loss: 0.5469
Epoch 5/5: 100%|██████████| 228/228 [00:42<00:00,  5.38it/s]Epoch 5/5, Loss: 0.5029
```

```
model.eval()
correct, total = 0, 0
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
```

```python
model.eval()
correct, total = 0, 0
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"Test Accuracy: {100 * correct / total:.2f}%")
```

```
Test Accuracy: 72.53%
```

16/1/25

7. Build a CNN model to classify cat and dog image

Aim :

Building a CNN model to classify cat and dog image.

Objective :

- To understand CNN implementation using PyTorch layers
- To Preprocess cat/dog images using torchvision transforms
- To train the model using DataLoader, loss function and optimizer
- To evaluate model Performance of CNN model using accuracy and loss metrics.

Pseudo code:

Start

1. Import necessary libraries (tensorflow / keras, numpy, matplotlib)
2. Load dataset
3. - Apply Preprocessing : resize, normalize pixel values (0-1)
   - split into training and testing set
4. Define CNN architecture :
   - Conv2D layer + ReLU activation
   - Maxpooling2D layer
   - Conv2D layer + ReLU activation
   - Maxpooling2D layer
   - Flatten layer
   - Dense (fully connected) hidden layer + ReLU
   - Dense output layer + Sigmoid (binary classification)

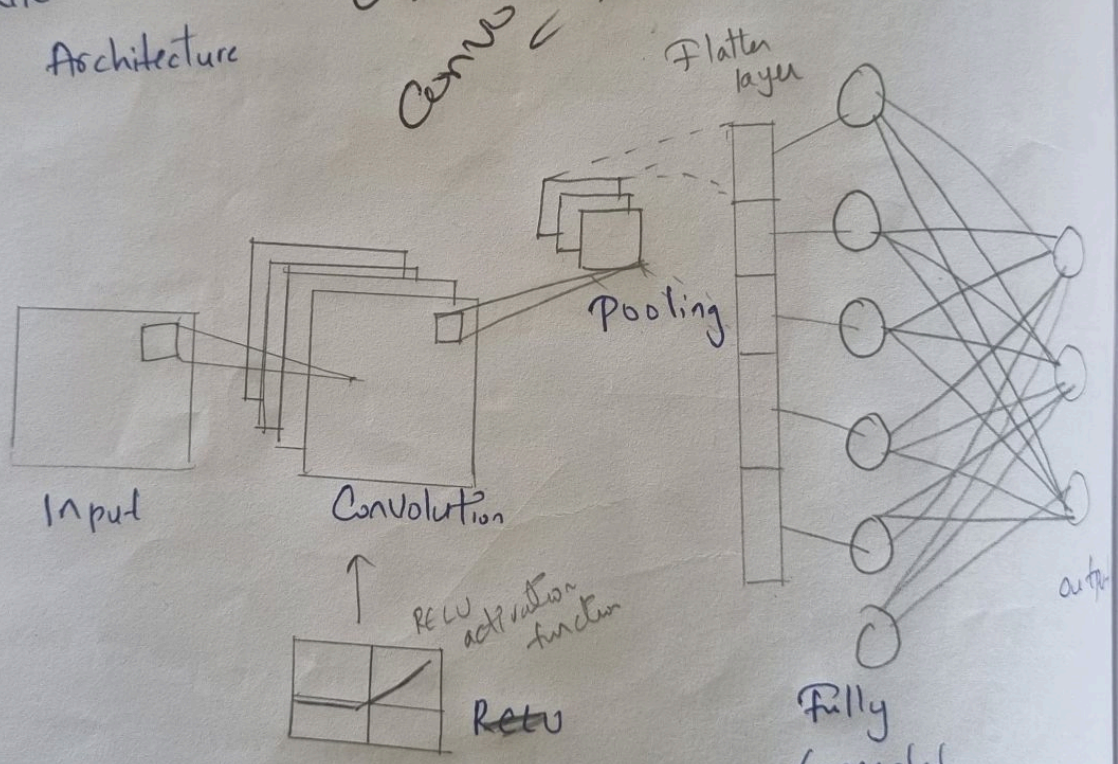
output

CNN ?

Convolution ?

CNN

Architecture



4. Compile the model:
   - Optimizer : Adam
   - Loss : Binary cross entropy
   - Metrics : Accuracy
5. Train the model on training data with validation set split
6. Evaluate model Performance on test data
7. Display training and Validation accuracy / loss curves
8. Predict on new images and classify as "Cat" or "Dog"

End

Observation:

- Training accuracy increases with epochs while Validation accuracy stabilizes.
- Loss decreases as training Progresses
- The CNN learns to extract features likes edges, textures, and shapes to differentiate cats from dogs.

Result:

Successfully built and trained a CNN model that classifies images of cats and dogs.

Output

Epoch 1/5, loss : 0.6530
Epoch 2/5, loss : 0.6312
Epoch 3/5, loss : 0.6020
Epoch 4/5, loss : 0.5469
Epoch 5/5, loss : 0.5029

Test accuracy : 72.53%

Epoch vs loss graph