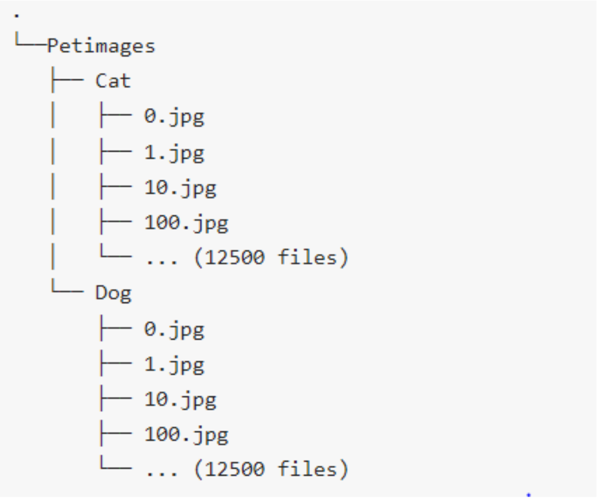
Create your CNN based on the given architecture:

1. The first layer comprises 32 of 3x3 filters
2. The first layer comprises 16 of 3x3 filters
3. For the module of the fully connected layers, it comprises 4 linear layers with size 128, 64, 32 and 10.
4. In each layer, the ReLU function is performed as the activation function.
5. The max pooling is used in every convolution layers.

Step	Implementation
Import Necessary Libraries	from PIL import Image import torch import torch.nn as nn import torch.nn.functional as F # adds some efficiency from torch.utils.data import DataLoader # lets us load data in batches from torch.utils.data import Subset # it is used to split our data from torchvision import datasets, transforms from torchsummary import summary from sklearn.model_selection import train_test_split # it is used to split our data import numpy as np import pandas as pd import matplotlib.pyplot as plt %matplotlib inline

Get Dataset

.
└─Petimages
  ├─ Cat
  | ├─ 0.jpg
  | ├─ 1.jpg
  | ├─ 10.jpg
  | ├─ 100.jpg
  | └─ ... (12500 files)
  └─ Dog
      ├─ 0.jpg
      ├─ 1.jpg
      ├─ 10.jpg
      ├─ 100.jpg
      └─ ... (12500 files)
	# !gdown 1EZ9KGTjz0Xkj7TWru-e-8JSO9OsoyUXl
	# !unzip -q /content/PetImages.zip

Examine the data

```
import os
from PIL import Image
from IPython.display import display

# Filter harmless warnings
import warnings
warnings.filterwarnings("ignore")

#check some images
with
Image.open('PetImages/Cat/0.jpg')
as im:
    display(im)
with
Image.open('PetImages/Dog/0.jpg')
as im:
    display(im)
```

Prepare train and test sets, loaders

```
def check_Image(path):
    try:
        im = Image.open(path)
        return True
    except:
        return False

transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor()
])
data =
datasets.ImageFolder('PetImages',
transform=transform,
is_valid_file=check_Image)

class_names = data.classes
Class_names



train_indices, test_indices, _, _ =
train_test_split(
    range(len(data)),
    data.targets,
    stratify=data.targets,
    test_size=0.3,
)

trainset = Subset(data, train_indices)
testset = Subset(data, test_indices)

trainloader = DataLoader(trainset, batch_size=64, shuffle=True)
testloader = DataLoader(testset, batch_size=64)

for X,y in trainloader:
  break
X.shape
```

Create model architecture

```
#Implement
#Implement

class CNN_cat_dog(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, 3) #convolusi layers dan linear layers
        self.conv2 = nn.Conv2d(32, 16, 3)
        #self.fc1 = nn.Linear(6*6*32, 128)
        self.fc1 = None
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64,32)
        self.fc4 = nn.Linear(32,10)

    def forward(self, X):
        X = F.relu(self.conv1(X))
        X = F.max_pool2d(X, 2, 2)
        X = F.relu(self.conv2(X))
        X = F.max_pool2d(X, 2, 2)

        if self.fc1 is None:
            print(X.shape)
            self.fc1 =
nn.Linear(X.shape[1]*X.shape[2]*X.s
hape[3], 128)

        X = X.view(-1,
X.shape[1]*X.shape[2]*X.shape[3])
        X = F.relu(self.fc1(X))
        X = F.relu(self.fc2(X))
        X = F.relu(self.fc3(X))

        X = self.fc4(X)
        return X

model = CNN_cat_dog()
summary(model, input_size = (3,
224,224))

criterion = nn.CrossEntropyLoss()
optimizer =
torch.optim.Adam(model.parameters()
, lr=0.001)
import time
from PIL import
UnidentifiedImageError  # Import
the UnidentifiedImageError

start_time = time.time()

epochs = 5
train_losses = []
test_losses = []
train_correct = []
```

```python
test_correct = []

for i in range(epochs):
    train_accuracy = 0
    test_accuracy = 0
    N_train = 0
    for X_train, y_train in trainloader:
        N_train += X_train.shape[0]
        try:
            y_pred = model(X_train)
            loss = criterion(y_pred, y_train)

            # Tally the number of correct predictions
            batch_corr = (y_pred.argmax(dim=-1) == y_train).sum()
            train_accuracy += batch_corr

            # Update parameters
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # Update training loss
            train_losses.append(loss.detach().numpy())
        except UnidentifiedImageError:
            print("Error processing image. Skipping...")
            continue  # Skip processing the current image

    # Print results
    print(f'epoch: {i:2} loss:
{loss.item():10.8f} accuracy:
{100*train_accuracy/N_train:7.3f}%'
)

    # Update training accuracy for
the epoch

train_correct.append(100*train_accu
racy.item()/N_train)

    # Run the testing batches
    with torch.no_grad():
        N_test = 0
        for X_test, y_test in
testloader:
            N_test +=
X_test.shape[0]
            try:
                # Apply the model
                y_pred =
model(X_test)

                # Tally the number
of correct predictions
                test_accuracy +=
(y_pred.argmax(dim=-1) ==
y_test).sum()

                # calculate testing
loss
                loss =
criterion(y_pred, y_test)

test_losses.append(loss.detach().nu
mpy())
            except
UnidentifiedImageError:
                print("Error
```

	processing image. Skipping...") continue # Skip processing the current image # Update testing accuracy for the epoch test_correct.append(100*test_accuracy/N_test) print(f'\nDuration: {time.time() - start_time:.0f} seconds') # print the time elapsed

Plot train losses	plt.plot(train_losses, label='training loss')
	plt.title('Loss at the end of each epoch')
	plt.legend();

Plot test losses	plt.plot(test_losses, label='testing loss')
	plt.title('Loss at the end of each epoch')
	plt.legend();

Correctness of train dan test sets	plt.plot(train_correct, label='training accuracy')
	plt.plot(test_correct, label='testing accuracy')
	plt.title('Accuracy at the end of each epoch')
	plt.legend();

Save the model	torch.save(model, 'mnist_model_full.pth')