St. Francis Institute of Technology, Mumbai-400 103
Department Of Information Technology

A.Y. 2025-2026
Class: BE-ITA/B, Semester: VII
Subject: Data Science Lab

Experiment – 6

1. **Aim:** To design a Convolution Neural Network (CNN) to classify x-ray images of lungs as COVID infected or not infected.

2. **Objectives:** Students should be familiarized with Learning Architectures and Frameworks using CNN.
3. **Prerequisite:** Python basics

4. **Pre-Experiment Exercise:**
   Theory:

   CNN:
   A convolutional neural network (CNN, or ConvNet) is a class of artificial neural network (ANN), most commonly applied to analyze visual imagery. CNNs are also known as Shift Invariant or Space Invariant Artificial Neural Networks (SIANN), based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation-equivariant responses known as feature maps. Counter-intuitively, most convolutional neural networks are not invariant to translation, due to the down sampling operation they apply to the input. They have applications in image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain–computer interfaces, and financial time series.

6. **Laboratory Exercise**
   Procedure
   - Take input as any image.
   - Process convolution layer on input image to extract features.
   - Apply a pooling layer to reduce the dimension of the image.

Post-Experiments Exercise:

A. Extended Theory:

a. Explain Architecture of CNN.
b. Compare RNN , LSTM and CNN

B. Post Lab Program:

a. Write a program to identify handwritten digits using CNN

C. Conclusion:

1. Write what was performed in the program (s) .
2. What is the significance of the program and what Objective is achieved?

D. References:

[1] https://stackabuse.com/image-recognition-in-python-with-tensorflow-and-keras/
[2] . https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn

x-ray image analysis using cnn

Dataset Link:https://www.kaggle.com/datasets/alifrahman/covid19-chest-xray-image-dataset

import necessary libraries

```
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2 as cv
import random
```

```
!pip install opendatasets
import opendatasets as od
od.download("https://www.kaggle.com/datasets/alifrahman/covid19-chest-xray-image-dataset")
```

```
Collecting opendatasets
  Downloading opendatasets-0.1.22-py3-none-any.whl.metadata (9.2 kB)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from opendatasets) (4.67.1)
Requirement already satisfied: kaggle in /usr/local/lib/python3.12/dist-packages (from opendatasets) (1.7.4.5)
Requirement already satisfied: click in /usr/local/lib/python3.12/dist-packages (from opendatasets) (8.2.1)
Requirement already satisfied: bleach in /usr/local/lib/python3.12/dist-packages (from kaggle->opendatasets) (6.2.0)
Requirement already satisfied: certifi>=14.05.14 in /usr/local/lib/python3.12/dist-packages (from kaggle->opendatasets) (2025.8.3)
Requirement already satisfied: charset-normalizer in /usr/local/lib/python3.12/dist-packages (from kaggle->opendatasets) (3.4.3)
Requirement already satisfied: idna in /usr/local/lib/python3.12/dist-packages (from kaggle->opendatasets) (3.10)
Requirement already satisfied: protobuf in /usr/local/lib/python3.12/dist-packages (from kaggle->opendatasets) (5.29.5)
Requirement already satisfied: python-dateutil>=2.5.3 in /usr/local/lib/python3.12/dist-packages (from kaggle->opendatasets) (2.9.0.post0)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.12/dist-packages (from kaggle->opendatasets) (8.0.4)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from kaggle->opendatasets) (2.32.4)
Requirement already satisfied: setuptools>=21.0.0 in /usr/local/lib/python3.12/dist-packages (from kaggle->opendatasets) (75.2.0)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.12/dist-packages (from kaggle->opendatasets) (1.17.0)
Requirement already satisfied: text-unidecode in /usr/local/lib/python3.12/dist-packages (from kaggle->opendatasets) (1.3)
Requirement already satisfied: urllib3>=1.15.1 in /usr/local/lib/python3.12/dist-packages (from kaggle->opendatasets) (2.5.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.12/dist-packages (from kaggle->opendatasets) (0.5.1)
Downloading opendatasets-0.1.22-py3-none-any.whl (15 kB)
Installing collected packages: opendatasets
Successfully installed opendatasets-0.1.22
Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
Your Kaggle username: shrutikadam221052
Your Kaggle Key: ··········
Dataset URL: https://www.kaggle.com/datasets/alifrahman/covid19-chest-xray-image-dataset
Downloading covid19-chest-xray-image-dataset.zip to ./covid19-chest-xray-image-dataset
100%|██████████| 40.6M/40.6M [00:00<00:00, 1.14GB/s]
```

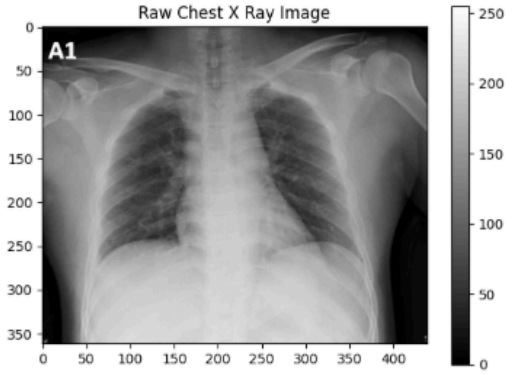
Investigating a single image from the Dataset:

```
# def load_image(path):
#     for img in os.listdir(bacteria_path):
#         print('Image name =',img)
#         image = cv.imread(os.path.join(bacteria_path, img))
#         break

#     return image
```

Investigating single image

```
from keras.preprocessing import image
bacteria_path = '/content/covid19-chest-xray-image-dataset/dataset/covid/1-s2.0-S1684118220300682-main.pdf-002-a1.png'
image = cv.imread(bacteria_path, cv.IMREAD_GRAYSCALE)
plt.imshow(image, cmap='gray')
plt.colorbar()
plt.title('Raw Chest X Ray Image')
print(f"The dimensions are {image.shape[0]} pixels height and {image.shape[1]} pixels width")
print(f"The maximum pixel value is {image.max():.4f}")
print(f"The minimum pixel value is {image.min():.4f}")
print(f"The mean value of the pixels is {image.mean():.4f}")
print(f"The standard deviation is {image.std():.4f}")
```

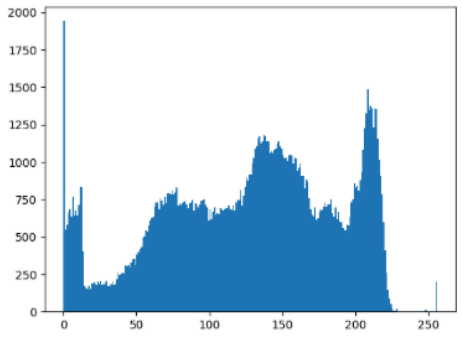
```
The dimensions are 362 pixels height and 439 pixels width
The maximum pixel value is 255.0000
The minimum pixel value is 0.0000
The mean value of the pixels is 126.5941
The standard deviation is 60.3495
```



plot histogram

```
plt.hist(image.ravel(),256,[0,256])
plt.show()
```

```
/tmp/ipython-input-3000102515.py:1: MatplotlibDeprecationWarning: Passing the range parameter of hist() positionally is deprecated since Matplotlib 3.9; the parameter will become keyword-only in 3.11.
  plt.hist(image.ravel(),256,[0,256])
```




```
!unzip data.zip
```

```
unzip:  cannot find or open data.zip, data.zip.zip or data.zip.ZIP.
```

Loading images and labels together and resizing images

```
path = '/content/covid19-chest-xray-image-dataset/dataset'
folders=[]
folders = [f for f in sorted(os.listdir(path))]
print(folders)
```

```
['covid', 'normal']
```

```
labels = folders
print (f'The labels are {labels}')
# setting the size of images that we want
image_size = 256
print(f'All images to be resized into {image_size}*{image_size} pixels')
```

Variables Terminal

```
image_size = 256
print(f'All images to be resized into {image_size}*{image_size} pixels')
```

```
The labels are ['covid', 'normal']
All images to be resized into 256*256 pixels
```

```
# defining a function to load images and labels together
# this function will also resize the images

def load_train(path):
    images = []
    for label in labels:
        direc = os.path.join(path, label)
        class_num = labels.index(label)
        for image_name in os.listdir(direc):
            image_path = os.path.join(direc, image_name)
            image_read = cv.imread(image_path, cv.IMREAD_GRAYSCALE)
            if image_read is not None: # Added check for successful image loading
                image_resized = cv.resize(image_read,(image_size,image_size))
                images.append([image_resized,class_num])
            else:
                print(f"Warning: Could not load image {image_path}") # Add a warning for unreadable images
    return images # Return a list of lists
```

```
train_data = load_train(path)
print(f'Number of loaded images and labels: {len(train_data)}')
```

```
Number of loaded images and labels: 94
```

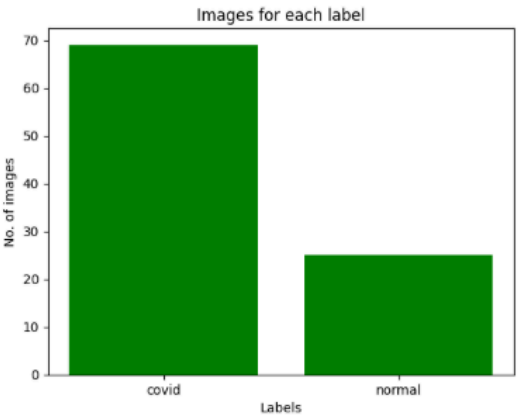
```
#loading the images and labels seperately in X and y, to be used later for training
X = []
y = []
for feature, label in train_data: # Iterate through the list of lists
    X.append(feature)
    y.append(label)

X = np.array(X) # Convert lists to NumPy arrays
y = np.array(y) # Convert lists to NumPy arrays

print (f'Shape of X = {X.shape}')
print (f'Shape of y = {y.shape}')
```

```
Shape of X = (94, 256, 256)
Shape of y = (94,)
```

```
# checking the number of images of each class
a = 0
b = 0
for label in y:
    if label == 0:
```

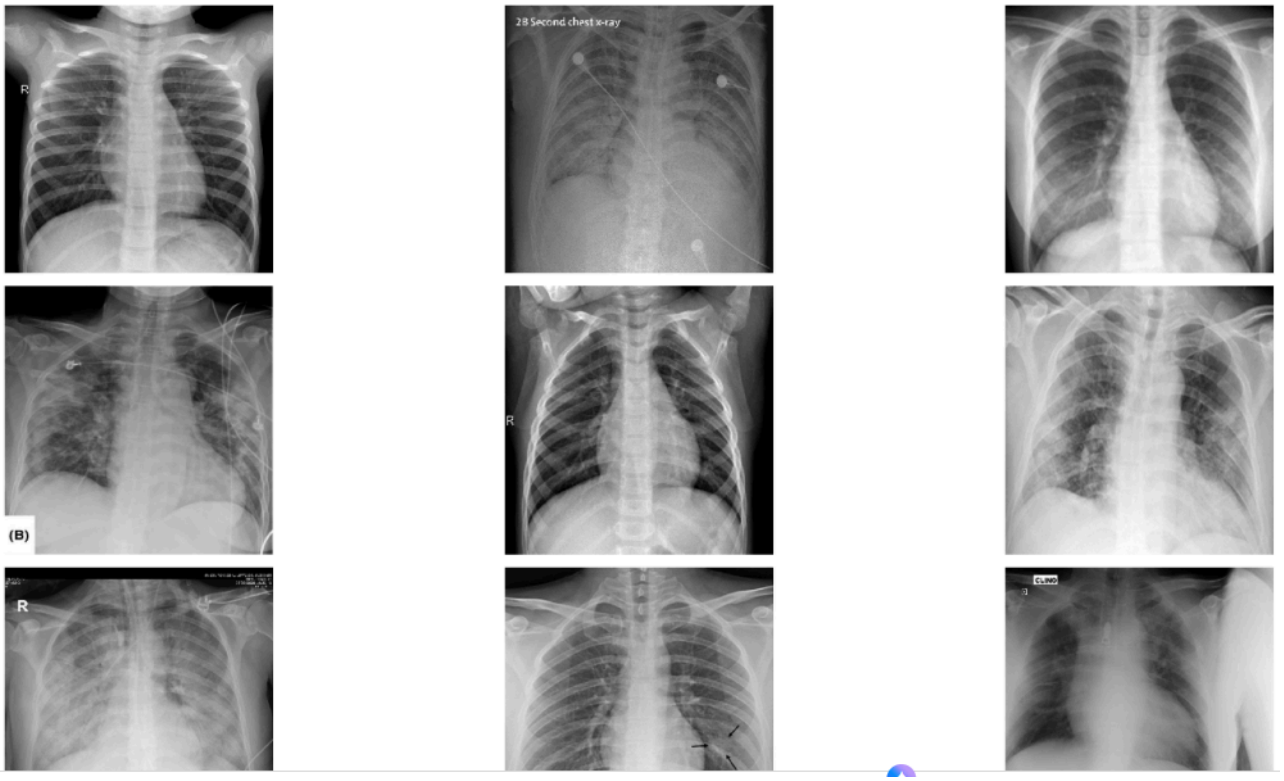
```
print (f'Number of Normal images = {a}')
print (f'Number of Covid images = {b}')

# plotting the data
x_pos = [i for i, _ in enumerate(labels)]
numbers = [a,b]
plt.bar(x_pos,numbers,color = 'green')
plt.xlabel("Labels")
plt.ylabel("No. of images")
plt.title("Images for each label")
plt.xticks(x_pos, labels)
plt.show()
```

```
Number of Normal images = 69
Number of Covid images = 25
```




```
# Displays images
# Extract 9 random images
print('Display Random Images')
# Adjust the size of your images
plt.figure(figsize=(20,10))
for i in range(9):
    num = random.randint(0,len(X)-1)
    plt.subplot(3, 3, i + 1)
    plt.imshow(X[num],cmap='gray')
    plt.axis('off')
# Adjust subplot parameters to give specified padding
plt.tight_layout()
```

```
    plt.subplot(3, 3, i + 1)
    plt.imshow(X[num],cmap='gray')
    plt.axis('off')
# Adjust subplot parameters to give specified padding
plt.tight_layout()
```



Data preprocessing

Normalize the image data by scaling pixel values.

```
# Normalize the image data
X = X / 255.0
print(f'The maximum pixel value after normalization is {X.max():.4f}')
print(f'The minimum pixel value after normalization is {X.min():.4f}')
print(f"Shape of X after normalization = {X.shape}")
```

```
The maximum pixel value after normalization is 0.0039
The minimum pixel value after normalization is 0.0000
Shape of X after normalization = (94, 256, 256)
```

Data splitting

Split the data into training and testing sets.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(f"Shape of X_train: {X_train.shape}")
print(f"Shape of X_test: {X_test.shape}")
print(f"Shape of y_train: {y_train.shape}")
print(f"Shape of y_test: {y_test.shape}")
```

```
Shape of X_train: (75, 256, 256)
Shape of X_test: (19, 256, 256)
Shape of y_train: (75,)
Shape of y_test: (19,)
```

Model building

Define the architecture of the Convolutional Neural Network using Keras.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(image_size, image_size, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: U
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 254, 254, 32)	320
max_pooling2d (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_1 (Conv2D)	(None, 125, 125, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_2 (Conv2D)	(None, 60, 60, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 128)	0
flatten (Flatten)	(None, 115200)	0
dense (Dense)	(None, 128)	14,745,728
dense_1 (Dense)	(None, 1)	129

```
Total params: 14,838,529 (56.60 MB)
Trainable params: 14,838,529 (56.60 MB)
Non-trainable params: 0 (0.00 B)
```

odel Prediction and Comparison

ake predictions on the test data and compare them with the actual labels.

```
# Make predictions on the test set
y_pred_prob = model.predict(X_test)

# Convert predicted probabilities to binary labels (0 or 1) using a threshold of 0.5
y_pred = (y_pred_prob > 0.5).astype(int)

# Print the predicted and actual labels for comparison
print("Predicted labels:\n", y_pred.flatten())
print("\nActual labels:\n", y_test)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 938ms/step
```

Terminal

```
[0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0]
```

```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Model training

Train the CNN model on the training data.

```
# Reshape the data to include a channel dimension
X_train = X_train.reshape(-1, image_size, image_size, 1)
X_test = X_test.reshape(-1, image_size, image_size, 1)

print(f"Shape of X_train after reshaping: {X_train.shape}")
print(f"Shape of X_test after reshaping: {X_test.shape}")

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))
```

```
Shape of X_train after reshaping: (75, 256, 256, 1)
Shape of X_test after reshaping: (19, 256, 256, 1)
Epoch 1/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 26s 9s/step - accuracy: 0.5054 - loss: 1.5667 - val_accuracy: 0.1579 - val_loss: 1.5458
Epoch 2/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 30s 3s/step - accuracy: 0.3089 - loss: 1.1480 - val_accuracy: 0.8421 - val_loss: 0.4419
Epoch 3/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 12s 3s/step - accuracy: 0.7127 - loss: 0.6031 - val_accuracy: 0.8421 - val_loss: 0.5640
Epoch 4/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 12s 4s/step - accuracy: 0.7339 - loss: 0.5722 - val_accuracy: 0.8421 - val_loss: 0.3592
Epoch 5/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 19s 3s/step - accuracy: 0.7766 - loss: 0.4595 - val_accuracy: 0.8947 - val_loss: 0.2176
Epoch 6/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 22s 3s/step - accuracy: 0.9237 - loss: 0.2571 - val_accuracy: 1.0000 - val_loss: 0.0863
Epoch 7/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 20s 4s/step - accuracy: 0.9276 - loss: 0.1505 - val_accuracy: 1.0000 - val_loss: 0.0287
Epoch 8/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 12s 4s/step - accuracy: 0.9789 - loss: 0.0623 - val_accuracy: 1.0000 - val_loss: 0.0168
Epoch 9/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 12s 4s/step - accuracy: 0.9800 - loss: 0.1145 - val_accuracy: 0.9474 - val_loss: 0.1834
Epoch 10/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 21s 3s/step - accuracy: 0.9053 - loss: 0.2541 - val_accuracy: 1.0000 - val_loss: 0.0439
```

Model evaluation

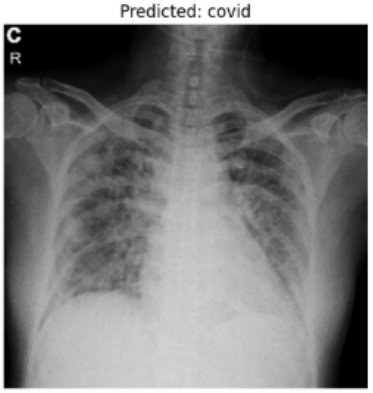
Evaluate the performance of the trained model on the testing data.

```
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)

print(f'\nTest loss: {test_loss:.4f}')
print(f'Test accuracy: {test_acc:.4f}')
```

```
1/1 - 1s - 744ms/step - accuracy: 1.0000 - loss: 0.0439

Test loss: 0.0439
Test accuracy: 1.0000
```

```
# Define a function to load and preprocess a single image
def preprocess_single_image(image_path, image_size):
    img = cv.imread(image_path, cv.IMREAD_GRAYSCALE)
    if img is not None:
        img_resized = cv.resize(img, (image_size, image_size))
        img_normalized = img_resized / 255.0  # Normalize pixel values
        img_reshaped = img_normalized.reshape(1, image_size, image_size, 1) # Reshape for model input
        return img_reshaped
    else:
        return None

# Specify the path to the image you want to test
single_image_path = '/content/covid19-chest-xray-image-dataset/dataset/covid/1-s2.0-S0929664620300449-gr2_lrg-c.jpg'

# Preprocess the image
preprocessed_image = preprocess_single_image(single_image_path, image_size)

if preprocessed_image is not None:
    # Make a prediction
    prediction_prob = model.predict(preprocessed_image)
    prediction_class = (prediction_prob > 0.5).astype(int)

    # Map the prediction to the actual label
    predicted_label = labels[prediction_class[0][0]]

    # Display the image and the prediction
    plt.imshow(preprocessed_image.reshape(image_size, image_size), cmap='gray')
    plt.title(f'Predicted: {predicted_label}')
    plt.axis('off')
    plt.show()

    print(f"The model predicts this image is: {predicted_label}")
else:
    print(f"Could not preprocess image from {single_image_path}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step
```




```
The model predicts this image is: covid
```

ables Terminal

Write a program to identify handwritten digits using CNN

Create a simple handwriting recognition model using a Convolutional Neural Network (CNN) and the MNIST dataset.

Load the dataset

Load a handwriting dataset, such as MNIST, which is commonly used for this task.

```
[ ]
from tensorflow import keras

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

Preprocess the data

Prepare the data for training by normalizing pixel values and reshaping the images.

```
[ ]
import numpy as np

x_train = x_train / 255.0
x_test = x_test / 255.0
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
y_train = keras.utils.to_categorical(y_train, num_classes=10)
y_test = keras.utils.to_categorical(y_test, num_classes=10)
print("x_train shape:", x_train.shape)
print("x_test shape:", x_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
x_train shape: (60000, 28, 28, 1)
x_test shape: (10000, 28, 28, 1)
y_train shape: (60000, 10)
y_test shape: (10000, 10)
```

Build the cnn model

Define the architecture of the CNN model, including convolutional layers, pooling layers, and dense layers.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shape`,
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 128)	204,928
dense_1 (Dense)	(None, 10)	1,290

```
Total params: 225,034 (879.04 KB)
Trainable params: 225,034 (879.04 KB)
Non-trainable params: 0 (0.00 B)
```

Compile the model

Configure the model for training by specifying the optimizer, loss function, and metrics.

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Train the model

Train the CNN model using the preprocessed data.

```
history = model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 55s 28ms/step - accuracy: 0.9107 - loss: 0.2894 - val_accuracy: 0.9855 - val_loss: 0.0431
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 53s 28ms/step - accuracy: 0.9870 - loss: 0.0444 - val_accuracy: 0.9897 - val_loss: 0.0336
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 54s 29ms/step - accuracy: 0.9914 - loss: 0.0281 - val_accuracy: 0.9893 - val_loss: 0.0348
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 52s 28ms/step - accuracy: 0.9934 - loss: 0.0215 - val_accuracy: 0.9899 - val_loss: 0.0306
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 52s 28ms/step - accuracy: 0.9958 - loss: 0.0142 - val_accuracy: 0.9907 - val_loss: 0.0286
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 51s 27ms/step - accuracy: 0.9963 - loss: 0.0112 - val_accuracy: 0.9896 - val_loss: 0.0320
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 51s 27ms/step - accuracy: 0.9972 - loss: 0.0089 - val_accuracy: 0.9907 - val_loss: 0.0303
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 51s 27ms/step - accuracy: 0.9979 - loss: 0.0066 - val_accuracy: 0.9920 - val_loss: 0.0294
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 51s 27ms/step - accuracy: 0.9984 - loss: 0.0054 - val_accuracy: 0.9907 - val_loss: 0.0344
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 52s 27ms/step - accuracy: 0.9979 - loss: 0.0071 - val_accuracy: 0.9898 - val_loss: 0.0452
```

Evaluate the model

Evaluate the performance of the trained CNN model on the test dataset.

```
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", loss)
print("Test accuracy:", accuracy)
```

```
Test loss: 0.18026791512966156
Test accuracy: 0.9451000094413757
```

Visualize test dataset predictions

```
import numpy as np
import matplotlib.pyplot as plt

# Get predictions for the test set
predictions = model.predict(x_test)

# Display a few test images and their predictions
num_images_to_display = 10

plt.figure(figsize=(10, 10))
for i in range(num_images_to_display):
    plt.subplot(5, 2, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    predicted_label = np.argmax(predictions[i])
    true_label = np.argmax(y_test[i])
    plt.title(f"True: {true_label}, Predicted: {predicted_label}")
    plt.axis('off')

plt.tight_layout()
plt.show()
```