|  |  |
| --- | --- |
| EXP.NO: 1  DATE: | ACCURACY OF VARIOUS ACTIVATION FUNCTIONS |

**AIM:**

To test the accuracies of various activation functions.

**ALGORITHM:**

* **Import Libraries**: Load TensorFlow, NumPy, and Matplotlib.
* **Load MNIST Data**: Get training and test data from MNIST.
* **Normalize Data**: Scale image pixel values to [0, 1].
* **Build Model**: Create a neural network with Flatten, Dense, Dropout, and Output layers.
* **Compile Model**: Set optimizer, loss function, and evaluation metric.
* **Train Model**: Train the model for 8 epochs.
* **Evaluate Model**: Test the model on unseen data.
* **Print Results**: Show loss and accuracy values.
* **Plot Chart**: Create a bar chart to compare model accuracies.
* **Show Chart**: Display the bar chart.

**PROGRAM:**

import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.datasets import mnist

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

activations = ['relu', 'sigmoid', 'tanh', 'softmax', 'swish']

accuracy\_results = {}

for activation in activations:

model = Sequential([

Flatten(input\_shape=(28, 28)),

Dense(128, activation=activation),

Dense(10, activation='softmax')

])

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

history = model.fit(x\_train, y\_train, epochs=5, validation\_data=(x\_test, y\_test), verbose=0)

accuracy\_results[activation] = history.history['val\_accuracy']

plt.figure(figsize=(10, 6))

for activation, accuracy in accuracy\_results.items():

plt.plot(range(1, 6), accuracy, label=activation)

plt.xlabel('Epochs')

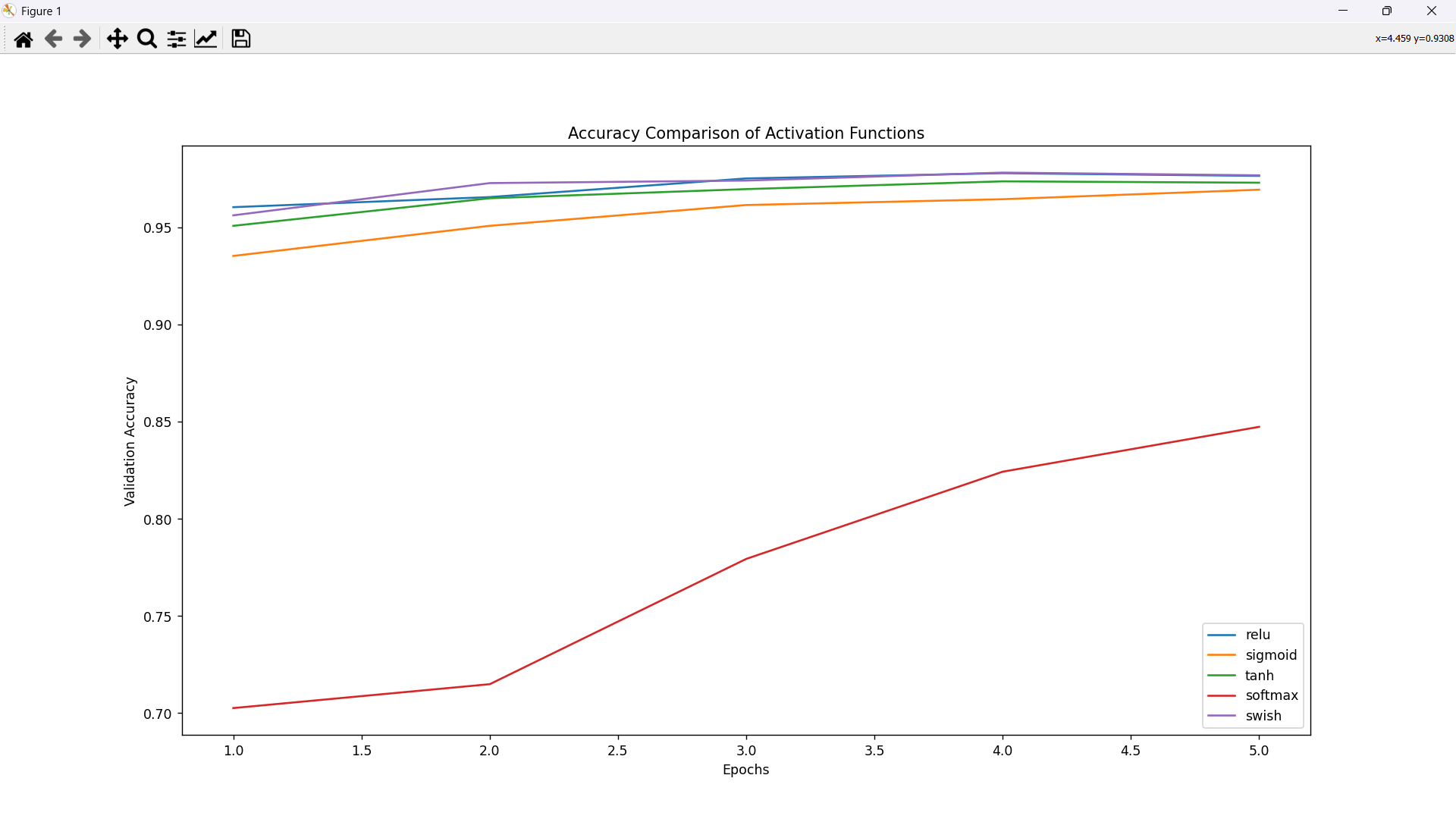
plt.ylabel('Validation Accuracy')

plt.title('Accuracy Comparison of Activation Functions')

plt.legend()

plt.show()

**OUTPUT:**



**RESULT:**

Thus the accuracies of various activation functions have been studied successfully.

|  |  |
| --- | --- |
| EXP.NO: 2  DATE: | MLP NEUTRAL NETWORK TO SOLVE THE XOR PROBLEM |

**AIM:**

To solve the given XOR problem using Multilayer Perceptron Neural Network.

**ALGORITHM:**

* Import TensorFlow and NumPy.
* Define XOR inputs and outputs.
* Create an MLP with one hidden (ReLU) and one output (Sigmoid) layer.
* Compile with Adam optimizer and binary crossentropy loss.
* Train for 500 epochs.
* Test predictions on XOR inputs.
* Print results.

**PROGRAM:**

import numpy as np

import tensorflow as tf

from tensorflow import keras

from tensorflow**.**keras**.**models import Sequential

from tensorflow**.**keras**.**layers import Dense

X = np**.**array**([[**0**,** 0**],** **[**0**,** 1**],** **[**1**,** 0**],** **[**1**,** 1**]])**

y = np**.**array**([[**0**],** **[**1**],** **[**1**],** **[**0**]])**

model = Sequential**([**

    Dense**(**4**,** *activation*=**'**relu**',** *input\_shape*=**(**2**,)),**

    Dense**(**1**,** *activation*=**'**sigmoid**')**

**])**

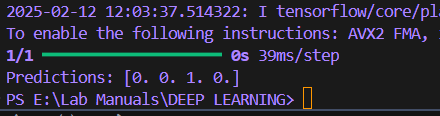
model**.**compile**(***optimizer*=**'**adam**',** *loss*=**'**binary\_crossentropy**',** *metrics*=**['**accuracy**'])**

model**.**fit**(**X**,** y**,** *epochs*=1000**,** *verbose*=0**)**

predictions = model**.**predict**(**X**)**

*print***("**Predictions:**",** np**.**round**(**predictions**).**flatten**())**

**OUTPUT:**



**RESULT:**

Thus, a MLP was built to solve the XOR problem.

|  |  |
| --- | --- |
| EXP.NO: 3  DATE: | ARTIFICIAL NEURAL NETWORK TO RECOGNIZE CHARACTERS AND DIGITS FROM IMAGES |

**AIM:**

To build a Artificial Neural Network to recognize characters and digits from images.

**ALGORITHM:**

* **Import Libraries**: Use Keras and NumPy.
* **Load Data**: Load MNIST dataset (28x28 images).
* **Preprocess Data**: Normalize inputs and one-hot encode outputs.
* **Build Model**:
  + 1. Add Flatten, Dense (120 neurons, ReLU), and Dense (10 neurons, Softmax) layers.
* **Compile**: Use Adam optimizer and categorical crossentropy loss.
* **Train Model**: Train for 10 epochs with validation.
* **Evaluate Model**: Test accuracy on the test set.
* **Predict**: Predict classes for test data and display results.

**PROGRAM:**

import tensorflow as tf

from tensorflow import keras

import numpy as np

import matplotlib**.**pyplot as plt

from PIL import Image**,** ImageOps

import os

mnist = keras**.**datasets**.**mnist

**(**x\_train**,** y\_train**),** **(**x\_test**,** y\_test**)** = mnist**.**load\_data**()**

**#** Normalize and Reshape

x\_train**,** x\_test = x\_train / 255.0**,** x\_test / 255.0

x\_train = x\_train**.**reshape**(**-1**,** 28**,** 28**,** 1**)**

x\_test = x\_test**.**reshape**(**-1**,** 28**,** 28**,** 1**)**

**#** Define CNN Model (More Layers for Accuracy)

model = keras**.**Sequential**([**

    keras**.**Input**(***shape*=**(**28**,** 28**,** 1**)),**

    keras**.**layers**.**Conv2D**(**32**,** **(**3**,** 3**),** *activation*=**'**relu**',** *padding*=**'**same**'),**

    keras**.**layers**.**MaxPooling2D**(**2**,** 2**),**

    keras**.**layers**.**Conv2D**(**64**,** **(**3**,** 3**),** *activation*=**'**relu**',** *padding*=**'**same**'),**

    keras**.**layers**.**MaxPooling2D**(**2**,** 2**),**

    keras**.**layers**.**Conv2D**(**128**,** **(**3**,** 3**),** *activation*=**'**relu**',** *padding*=**'**same**'),**

    keras**.**layers**.**MaxPooling2D**(**2**,** 2**),**

    keras**.**layers**.**Flatten**(),**

    keras**.**layers**.**Dense**(**256**,** *activation*=**'**relu**'),**

    keras**.**layers**.**Dense**(**10**,** *activation*=**'**softmax**')**

**])**

**#** Compile and Train (More epochs for accuracy)

model**.**compile**(***optimizer*=**'**adam**',** *loss*=**'**sparse\_categorical\_crossentropy**',** *metrics*=**['**accuracy**'])**

model**.**fit**(**x\_train**,** y\_train**,** *epochs*=10**,** *validation\_data*=**(**x\_test**,** y\_test**))**

**#** Evaluate Model

test\_loss**,** test\_acc = model**.**evaluate**(**x\_test**,** y\_test**)**

*print***(**f**"**\nTest Accuracy: {test\_acc:.4f}**")**

**#** Function to Load and Preprocess Image

def load\_and\_preprocess\_image**(***image\_path***):**

    if not os**.**path**.**exists**(***image\_path***):**

*print***("**Error: File not found!**")**

        return None

    try**:**

        img = Image**.**open**(***image\_path***).**convert**('**L**')**  **#** Convert to grayscale

        img = ImageOps**.**invert**(**img**)**  **#** Invert colors (MNIST has white text, black bg)

        img = img**.**resize**((**28**,** 28**))**  **#** Resize to match MNIST

        img = np**.**array**(**img**,** *dtype*=np**.**float32**)** */* 255.0  **#** Normalize

        img = img**.**reshape**(**1**,** 28**,** 28**,** 1**)**  **#** Reshape for model input

**#** Debug: Show processed image before prediction

        plt**.**imshow**(**img**.**reshape**(**28**,** 28**),** *cmap*=**'**gray**')**

        plt**.**title**("**Processed Image**")**

        plt**.**axis**('**off**')**

        plt**.**show**()**

        return img

    except *Exception* as e**:**

*print***(**f**"**Error loading image: {e}**")**

        return None

**#** Provide Image Path

image\_path = r**"**E:\Lab Manuals*\D*EEP LEARNING\image*.*png**"**  **#** Ensure correct path

**#** Load and Predict Image

img = load\_and\_preprocess\_image**(**image\_path**)**

if img is not None**:**

    predictions = model**.**predict**(**img**)**

    predicted\_label = np**.**argmax**(**predictions**)**

**#** Show Final Prediction

    plt**.**imshow**(**img**.**reshape**(**28**,** 28**),** *cmap*=**'**gray**')**

    plt**.**title**(**f**"**Predicted: {predicted\_label}**")**

    plt**.**axis**('**off**')**

    plt**.**show**()**

else**:**

*print***("** Failed to load image.**")**

**OUTPUT:**



**RESULT:**

Thus, an Artificial Neural Network to recognize characters and digits from images.

|  |  |
| --- | --- |
| EXP.NO: 4  DATE: | PROGRAM USING AUTOENCODERS TO ANALYZE IMAGES FOR IMAGE RECONSTRUCTION TASKS |

**AIM:**

To write a program using autoencoders to analyze images for image reconstruction tasks

**ALGORITHM:**

* **Import Libraries**: Use NumPy, Matplotlib, and Keras modules.
* **Load Dataset**: Load Fashion MNIST data and normalize it to the range [0, 1].
* **Reshape Data**: Reshape the data to include the channel dimension (28, 28, 1).
* **Build Autoencoder**:
  1. **Encoder**: Use Conv2D and MaxPooling2D layers to compress input.
  2. **Decoder**: Use Conv2D and UpSampling2D layers to reconstruct input.
* **Compile Model**: Use Adam optimizer and binary crossentropy loss.
* **Train Model**: Fit the autoencoder on the training data for 5 epochs.
* **Test Reconstruction**: Generate reconstructed images from test data.
* **Visualize Results**: Plot original and reconstructed images side by side.

**PROGRAM:**

from keras**.**datasets import cifar10

import numpy as np

import matplotlib**.**pyplot as plt

from keras**.**models import Model

from keras**.**layers import Input**,** Conv2D**,** MaxPooling2D**,** UpSampling2D

**(**X\_train**,** \_**),** **(**X\_test**,** \_**)** = cifar10**.**load\_data**()**

X\_train = X\_train**.**astype**('**float32**')** / 255.0

X\_test = X\_test**.**astype**('**float32**')** / 255.0

X\_train = X\_train**.**reshape**((***len***(**X\_train**),** 32**,** 32**,** 3**))**

X\_test = X\_test**.**reshape**((***len***(**X\_test**),** 32**,** 32**,** 3**))**

input\_img = Input**(***shape*=**(**32**,** 32**,** 3**))**

x = Conv2D**(**32**,** **(**3**,** 3**),** *activation*=**'**relu**',** *padding*=**'**same**')(**input\_img**)**

x = MaxPooling2D**((**2**,** 2**),** *padding*=**'**same**')(**x**)**

x = Conv2D**(**16**,** **(**3**,** 3**),** *activation*=**'**relu**',** *padding*=**'**same**')(**x**)**

encoded = MaxPooling2D**((**2**,** 2**),** *padding*=**'**same**')(**x**)**

x = Conv2D**(**16**,** **(**3**,** 3**),** *activation*=**'**relu**',** *padding*=**'**same**')(**encoded**)**

x = UpSampling2D**((**2**,** 2**))(**x**)**

x = Conv2D**(**32**,** **(**3**,** 3**),** *activation*=**'**relu**',** *padding*=**'**same**')(**x**)**

x = UpSampling2D**((**2**,** 2**))(**x**)**

decoded = Conv2D**(**3**,** **(**3**,** 3**),** *activation*=**'**sigmoid**',** *padding*=**'**same**')(**x**)**

autoencoder = Model**(**input\_img**,** decoded**)**

autoencoder**.**compile**(***optimizer*=**'**adam**',** *loss*=**'**binary\_crossentropy**')**

autoencoder**.**fit**(**X\_train**,** X\_train**,** *epochs*=10**,** *batch\_size*=256**,** *shuffle*=True**,** *validation\_data*=**(**X\_test**,** X\_test**))**

decoded\_imgs = autoencoder**.**predict**(**X\_test**)**

n = 10

plt**.**figure**(***figsize*=**(**20**,** 4**))**

for i in *range***(**n**):**

    ax = plt**.**subplot**(**2**,** n**,** i + 1**)**

    plt**.**imshow**(**X\_test**[**i**])**

    ax**.**get\_xaxis**().**set\_visible**(**False**)**

    ax**.**get\_yaxis**().**set\_visible**(**False**)**

    ax = plt**.**subplot**(**2**,** n**,** i + 1 + n**)**

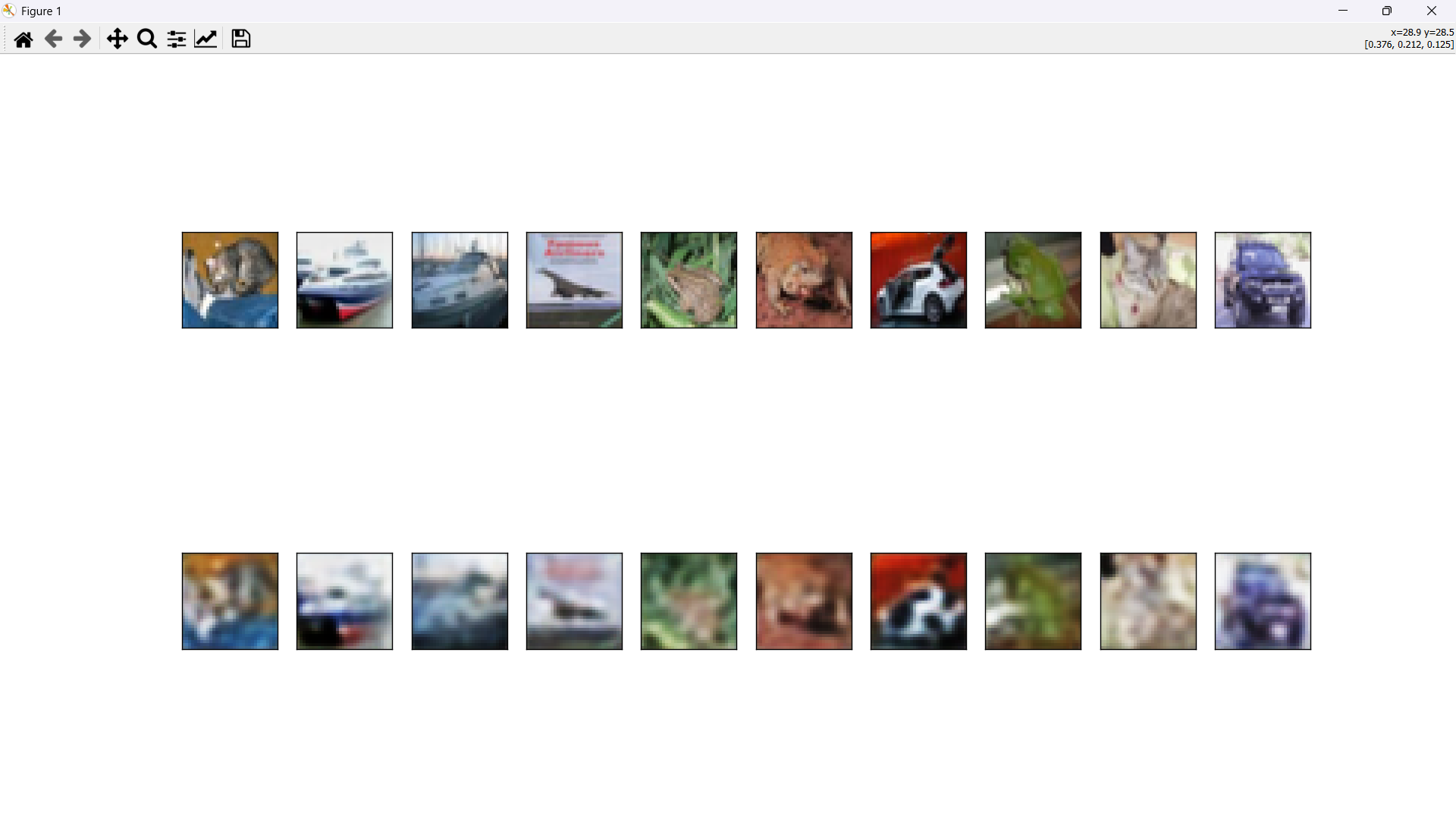
    plt**.**imshow**(**decoded\_imgs**[**i**])**

    ax**.**get\_xaxis**().**set\_visible**(**False**)**

    ax**.**get\_yaxis**().**set\_visible**(**False**)**

plt**.**show**()**

**OUTPUT:**



**RESULT:**

Thus, a program using autoencoders to analyze images for image reconstruction tasks was build successfully.

|  |  |
| --- | --- |
| EXP.NO: 5  DATE: | CONVOLUTIONAL NEURAL NETWORK (CNN) FOR SPEECH RECOGNITION |

**AIM:**

To build Convolutional Neural Network for Speech Recognition task.

**ALGORITHM:**

* **Import Libraries**: Use TensorFlow and TensorFlow Datasets.
* **Load Dataset**: Load the "Speech Commands" dataset and split it into train and test sets.
* **Convert to NumPy**: Convert the dataset to NumPy format for easier manipulation.
* **Build Model**:
  1. Add Conv2D layers with ReLU activation.
  2. Add MaxPooling2D layers for down-sampling.
  3. Flatten the output and add Dense layers for classification.
* **Compile Model**: Use Adam optimizer and categorical crossentropy loss.
* **Train Model**: Train the model for 10 epochs on the training data.
* **Evaluate Model**: Evaluate the model's performance on test data and print accuracy.
* **Predict Classes**: Use the model to predict classes for test data and display them.

**PROGRAM:**

import tensorflow as tf

import numpy as np

from sklearn**.**model\_selection import train\_test\_split

from tensorflow**.**keras**.**utils import to\_categorical

def generate\_dummy\_data**(***num\_samples*=1000**,** *input\_length*=16000**,** *num\_classes*=10**):**

    X = np**.**random**.**rand**(***num\_samples***,** *input\_length***,** 1**)**

    y = np**.**random**.**randint**(**0**,** *num\_classes***,** *num\_samples***)**

    y = to\_categorical**(**y**,** *num\_classes*=*num\_classes***)**

    return X**,** y

def build\_cnn\_model**(***input\_shape***,** *num\_classes***):**

    model = tf**.**keras**.**Sequential**([**

        tf**.**keras**.**layers**.**InputLayer**(***input\_shape*=*input\_shape***),**

        tf**.**keras**.**layers**.**Conv1D**(**32**,** *kernel\_size*=3**,** *activation*=**'**relu**'),**

        tf**.**keras**.**layers**.**MaxPooling1D**(***pool\_size*=2**),**

        tf**.**keras**.**layers**.**BatchNormalization**(),**

        tf**.**keras**.**layers**.**Conv1D**(**64**,** *kernel\_size*=3**,** *activation*=**'**relu**'),**

        tf**.**keras**.**layers**.**MaxPooling1D**(***pool\_size*=2**),**

        tf**.**keras**.**layers**.**BatchNormalization**(),**

        tf**.**keras**.**layers**.**Flatten**(),**

        tf**.**keras**.**layers**.**Dense**(**128**,** *activation*=**'**relu**'),**

        tf**.**keras**.**layers**.**Dropout**(**0.5**),**

        tf**.**keras**.**layers**.**Dense**(***num\_classes***,** *activation*=**'**softmax**')**

**])**

    return model

num\_samples = 1000

input\_length = 16000

num\_classes = 10

X**,** y = generate\_dummy\_data**(**num\_samples**,** input\_length**,** num\_classes**)**

X\_train**,** X\_test**,** y\_train**,** y\_test = train\_test\_split**(**X**,** y**,** *test\_size*=0.2**,** *random\_state*=42**)**

input\_shape = **(**input\_length**,** 1**)**

model = build\_cnn\_model**(***input\_shape*=input\_shape**,** *num\_classes*=num\_classes**)**

model**.**compile**(***optimizer*=**'**adam**',** *loss*=**'**categorical\_crossentropy**',** *metrics*=**['**accuracy**'])**

epochs = 10

batch\_size = 32

*print***("**Training the model...**")**

history = model**.**fit**(**X\_train**,** y\_train**,** *validation\_split*=0.2**,** *epochs*=epochs**,** *batch\_size*=batch\_size**)**

*print***("**Evaluating the model...**")**

test\_loss**,** test\_accuracy = model**.**evaluate**(**X\_test**,** y\_test**,** *verbose*=2**)**

*print***(**f**"**Test Accuracy: {test\_accuracy \* 100:.2f}%**")**

*print***("**Making predictions on test data...**")**

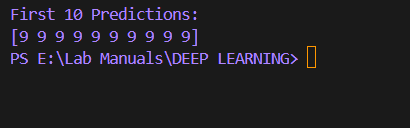
predictions = model**.**predict**(**X\_test**)**

predicted\_classes = np**.**argmax**(**predictions**,** *axis*=1**)**

*print***("**First 10 Predictions:**")**

*print***(**predicted\_classes**[:**10**])**

**OUTPUT:**

****

**RESULT:**

Thus, a Convolutional Neural Network for speech recognition was build successfully.