1. Create Tensors and perform basic operations with tensors

```
import numpy as np

t = np.array([[1, 2, 3], [4, 5, 6]])
t1 = t * 2
print(t1)
```

```
[[ 2  4  6]
 [ 8 10 12]]
```

```
# Element-wise addition
t2 = t + 3
print(" \n" , t2)
```

```

 [[4 5 6]
 [7 8 9]]
```

```
# Element-wise exponentiation
t3 = np.power(t, 2)
print(" \n" ,t3)
```

```

 [[ 1  4  9]
 [16 25 36]]
```

```
# Transpose
t4 = np.transpose(t)
print(" \n" ,t4)
```

```

 [[1 4]
 [2 5]
 [3 6]]
```

2. Create Tensors and apply split & merge operations and statistics operations.

```
import numpy as np

# Get user input for tensor dimensions
rows = int(input("Enter number of rows: "))
cols = int(input("Enter number of columns: "))
```

```
Enter number of rows: 2
Enter number of columns: 2
```

```
# Create an empty tensor with the specified dimensions
tensor = np.empty((rows, cols))

# Get user input for tensor values
for i in range(rows):
 for j in range(cols):
  tensor[i][j] = float(input(f"Enter value for [{i}][{j}]: "))
```

```
Enter value for [0][0]: 1
Enter value for [0][1]: 2
Enter value for [1][0]: 3
Enter value for [1][1]: 4
```

```
# Print the tensor
print("Tensor:\n", tensor)

# Get user input for split axis
split_axis = int(input("Enter the axis to split along (0 for rows, 1 for columns): "))
```

```
Tensor:
 [[1. 2.]
 [3. 4.]]
Enter the axis to split along (0 for rows, 1 for columns): 0
```

```
# Split the tensor along the specified axis
tensor_split = np.split(tensor, 2, axis=split_axis)

# Print the split tensors
print("Split tensors:\n", tensor_split)
```

```
Split tensors:
 [array([[1., 2.]]), array([[3., 4.]])]
```

```
# Merge the split tensors along the same axis
tensor_merged = np.concatenate(tensor_split, axis=split_axis)

# Print the merged tensor
print("Merged tensor:\n", tensor_merged)
```

```
Merged tensor:
 [[1. 2.]
 [3. 4.]]
```

```
# Calculate the mean of the tensor
mean = np.mean(tensor)
print("Mean:", mean)

# Calculate the standard deviation of the tensor
std = np.std(tensor)
print("Standard deviation:", std)

# Calculate the variance of the tensor
```

```
var = np.var(tensor)
print("Variance:", var)

# Find the maximum and minimum values in the tensor
max_val = np.max(tensor)
min_val = np.min(tensor)
print("Maximum value:", max_val)
print("Minimum value:", min_val)
```

```
Mean: 2.5
Standard deviation: 1.118033988749895
Variance: 1.25
Maximum value: 4.0
Minimum value: 1.0
```

```
# Reshape the tensor to a 1D array
reshaped_tensor = np.reshape(tensor, (1, -1))
print("Reshaped tensor:\n", reshaped_tensor)

# Transpose the tensor
transposed_tensor = np.transpose(tensor)
print("Transposed tensor:\n", transposed_tensor)

# Perform a dot product between the tensor and its transpose
dot_product = np.dot(tensor, transposed_tensor)
print("Dot product:\n", dot_product)
```

```
Reshaped tensor:
 [[1. 2. 3. 4.]]
Transposed tensor:
 [[1. 3.]
 [2. 4.]]
Dot product:
 [[ 5. 11.]
 [11. 25.]]
```

3. Design single unit perceptron for classification of iris dataset without using predefined models

```
import numpy as np
from sklearn.datasets import load_iris

# Load the iris dataset
iris = load_iris()
```

```
# Select two classes of flowers for binary classification
X = iris.data[:100, :2]
y = iris.target[:100]

# Define the learning rate and number of iterations
```

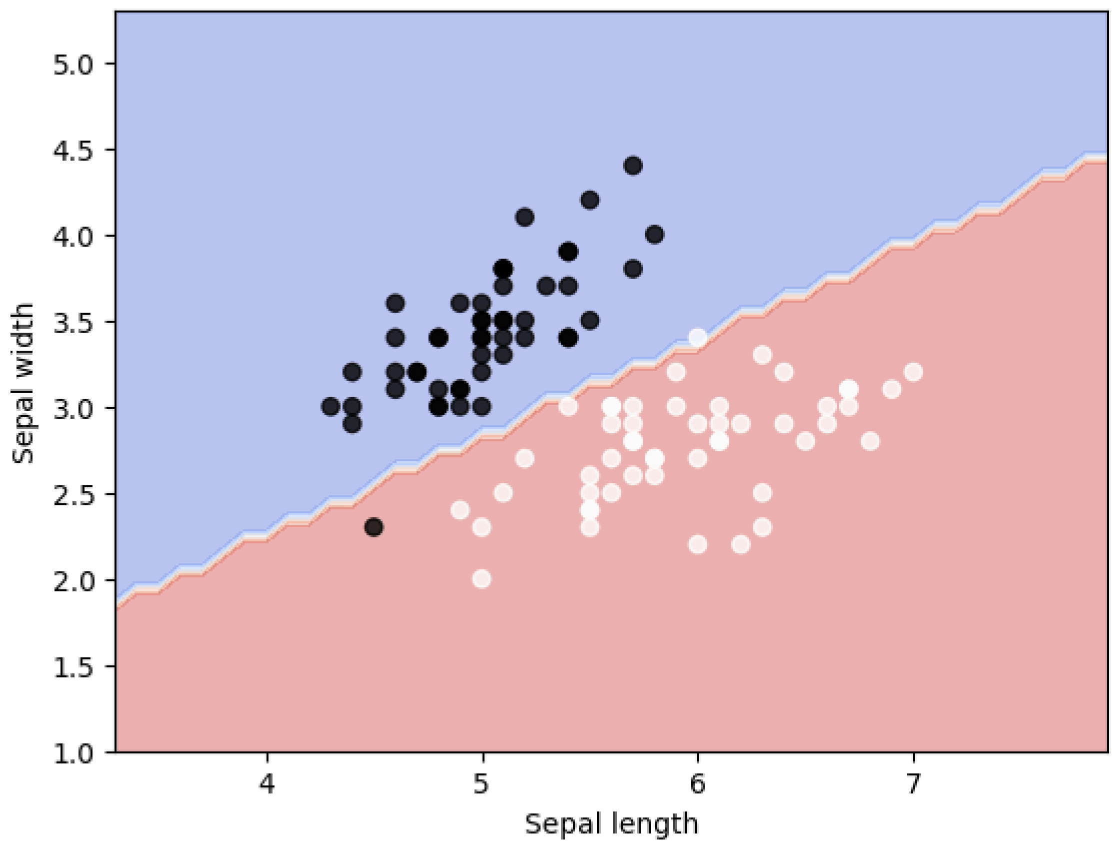
```
learning_rate = 0.1
num_iterations = 100
```

```
# Initialize the weights randomly
weights = np.random.rand(2)

# Define the activation function (here we use a simple threshold function)
def activation(x):
 return np.where(x >= 0, 1, 0)
```

```
# Train the perceptron
for i in range(num_iterations):
  # Initialize the gradient and cost
  gradient = np.zeros(2)
  cost = 0
  # Loop over the training examples

  for j in range(len(X)):
    # Compute the output and error for the current example
    output = activation(np.dot(X[j], weights))
    error = y[j] - output

    # Update the gradient and cost
    gradient += error * X[j]
    cost += error ** 2

  # Update the weights based on the gradient and learning rate
  weights += learning_rate * gradient

  # Print the cost for this iteration
  print("Iteration:", i, "Cost:", cost)
```

```
Iteration: 65 Cost: 2
Iteration: 66 Cost: 2
Iteration: 67 Cost: 1
Iteration: 68 Cost: 2
Iteration: 69 Cost: 2
Iteration: 70 Cost: 2
Iteration: 71 Cost: 1
Iteration: 72 Cost: 2
Iteration: 73 Cost: 2
Iteration: 74 Cost: 2
Iteration: 75 Cost: 1
Iteration: 76 Cost: 2
Iteration: 77 Cost: 2
Iteration: 78 Cost: 1
Iteration: 79 Cost: 2
Iteration: 80 Cost: 2
Iteration: 81 Cost: 2
Iteration: 82 Cost: 1
Iteration: 83 Cost: 2
Iteration: 84 Cost: 2
Iteration: 85 Cost: 2
Iteration: 86 Cost: 1
Iteration: 87 Cost: 2
Iteration: 88 Cost: 2
Iteration: 89 Cost: 2
Iteration: 90 Cost: 1
Iteration: 91 Cost: 2
Iteration: 92 Cost: 2
Iteration: 93 Cost: 1
Iteration: 94 Cost: 2
Iteration: 95 Cost: 2
Iteration: 96 Cost: 2
Iteration: 97 Cost: 1
Iteration: 98 Cost: 2
Iteration: 99 Cost: 2
```

```
# Plot the decision boundary
import matplotlib.pyplot as plt
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),np.arange(y_min, y_max, 0.1))
Z = activation(np.dot(np.c_[xx.ravel(), yy.ravel()], weights))
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.4, cmap="coolwarm")
plt.scatter(X[:, 0], X[:, 1], c=y, alpha=0.8)
plt.xlabel("Sepal length")
plt.ylabel("Sepal width")
plt.show()
```



4. Design, train and test the MLP for tabular data and verify various activation functions and optimizers tensor flow.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import SGD, Adam, RMSprop
from tensorflow.keras.activations import relu, sigmoid, tanh

# load the data
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
data = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target,
test_size=0.2, random_state=42)
```

```
# define the model
def create_model(activation_func, optimizer):
 model = Sequential([
 Dense(64, input_dim=X_train.shape[1], activation=activation_func),
 Dropout(0.5),
 Dense(32, activation=activation_func),
 Dropout(0.5),
 Dense(1, activation='sigmoid')])

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

```
 return model

# define the activation functions and optimizers to try
activation_funcs = [relu, sigmoid, tanh]
optimizer_classes = [SGD, Adam, RMSprop]
```

```
for activation_func in activation_funcs:
    for optimizer_class in optimizer_classes:
        # Create a new instance of the optimizer for each model
        optimizer = optimizer_class(learning_rate=0.001)
        model = create_model(activation_func, optimizer)

        print(f'Training model with activation function {activation_func.__name__} and op
        model.fit(X_train, y_train, epochs=50, batch_size=16, verbose=0)
        loss, accuracy = model.evaluate(X_test, y_test)
        print(f'Test loss: {loss:.3f}, Test accuracy: {accuracy:.3f}\n')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarnir
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Training model with activation function relu and optimizer SGD...
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9153 - loss: 0.2728
Test loss: 0.270, Test accuracy: 0.921

Training model with activation function relu and optimizer Adam...
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9128 - loss: 0.3411
Test loss: 0.341, Test accuracy: 0.912

Training model with activation function relu and optimizer RMSprop...
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9428 - loss: 0.1221
Test loss: 0.133, Test accuracy: 0.930

Training model with activation function sigmoid and optimizer SGD...
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6377 - loss: 0.6233
Test loss: 0.631, Test accuracy: 0.623

Training model with activation function sigmoid and optimizer Adam...
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9352 - loss: 0.1615
Test loss: 0.157, Test accuracy: 0.947

Training model with activation function sigmoid and optimizer RMSprop...
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9296 - loss: 0.2246
Test loss: 0.210, Test accuracy: 0.939

Training model with activation function tanh and optimizer SGD...
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6377 - loss: 0.6795
Test loss: 0.694, Test accuracy: 0.623

Training model with activation function tanh and optimizer Adam...
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9240 - loss: 0.1858
Test loss: 0.188, Test accuracy: 0.930

Training model with activation function tanh and optimizer RMSprop...
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9184 - loss: 0.2142
Test loss: 0.208, Test accuracy: 0.921
```

5. Design and implement a simple RNN model with tensorflow / keras and check accuracy.

```
import tensorflow as tf
from  tensorflow.keras.datasets import mnist

(x_train, y_train), (x_test,y_test) = mnist.load_data()

x_train = x_train.astype('float32')/255.0
x_test = x_test.astype('float32')/255.0
y_train = tf.keras.utils.to_categorical(y_train,num_classes=10)
y_test = tf.keras.utils.to_categorical(y_test,num_classes=10)
```

```
input_shape = (28,28)
num_classes =10
hidden_size = 128
batch_size = 128
epochs = 10

model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=input_shape),
    tf.keras.layers.Reshape(target_shape=(input_shape[0],input_shape[1]*1)),
    tf.keras.layers.LSTM(units=hidden_size,activation='tanh'),
    tf.keras.layers.Dense(num_classes,activation='softmax')])
```

```
model.compile(loss='categorical_crossentropy', optimizer=tf.keras.optimizers.Adam(), metr
model.fit(x_train,y_train,batch_size-batch_size,epochs=epochs,validation_data=(x_test,y_t

score = model.evaluate(x_test,y_test,verbose=0)
print('Test loss: ',score[0])
print('Test accuracy: ',score[1])
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 79s 41ms/step - accuracy: 0.7728 - loss: 0.6764 - val_
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 77s 39ms/step - accuracy: 0.9666 - loss: 0.1092 - val_
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 72s 38ms/step - accuracy: 0.9790 - loss: 0.0707 - val_
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 81s 38ms/step - accuracy: 0.9839 - loss: 0.0530 - val_
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 83s 38ms/step - accuracy: 0.9870 - loss: 0.0453 - val_
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 82s 38ms/step - accuracy: 0.9880 - loss: 0.0392 - val_
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 82s 38ms/step - accuracy: 0.9912 - loss: 0.0280 - val_
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 70s 37ms/step - accuracy: 0.9913 - loss: 0.0264 - val_
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 83s 38ms/step - accuracy: 0.9936 - loss: 0.0216 - val_
Epoch 10/10
```

```
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 73s 39ms/step - accuracy: 0.9932 - loss: 0.0222 - val_
Test loss:  0.0445997118949890 14
Test accuracy:  0.9864000082015991
```

6. Design and implement LSTM model with tensorflow / keras and check accuracy.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

model = Sequential([LSTM(32,input_shape=(10,1)), Dense(1,activation='sigmoid')])

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

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning:
  super().__init__(**kwargs)
```

```
import numpy as np
x = np.random.rand(100,10,1)
y = np.random.randint(0,2,(100,1))

model.fit(x,y,epochs=10,batch_size=32)

loss, accuracy = model.evaluate(x,y)
print(f'Test loss: {loss}, Test accuracy: {accuracy}')
```

```
Epoch 1/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.4295 - loss: 0.7204
Epoch 2/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.4357 - loss: 0.7168
Epoch 3/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.4492 - loss: 0.7089
Epoch 4/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.3940 - loss: 0.7127
Epoch 5/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.4315 - loss: 0.6994
Epoch 6/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.4050 - loss: 0.6956
Epoch 7/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5539 - loss: 0.6920
Epoch 8/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5549 - loss: 0.6897
Epoch 9/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5320 - loss: 0.6902
Epoch 10/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5747 - loss: 0.6841
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6007 - loss: 0.6786
Test loss: 0.6823850274085999, Test accuracy: 0.5799999833106995
```

7. Design and implement a simple GRU model with tensorflow and check accuracy.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU, Dense
```

```
# define the model
model = Sequential([
    GRU(32, input_shape=(10, 1)),
    Dense(1, activation='sigmoid')])

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

```
# generate some random data
import numpy as np
X = np.random.rand(100, 10, 1)
y = np.random.randint(0, 2, (100, 1))

# train the model
model.fit(X, y, epochs=10, batch_size=32)

# evaluate the model
loss, accuracy = model.evaluate(X, y)
print(f'Test loss: {loss}, Test accuracy: {accuracy}')
```

```
Epoch 1/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.5477 - loss: 0.6921
Epoch 2/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5311 - loss: 0.6950
Epoch 3/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.5318 - loss: 0.6989
Epoch 4/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5400 - loss: 0.6966
Epoch 5/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5474 - loss: 0.6967
Epoch 6/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.5382 - loss: 0.6917
Epoch 7/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.4526 - loss: 0.6982
Epoch 8/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.4477 - loss: 0.6951
Epoch 9/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.4767 - loss: 0.6965
Epoch 10/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.4602 - loss: 0.6946
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.4277 - loss: 0.6985
Test loss: 0.6958896517753601, Test accuracy: 0.46000000834465027
```

8. Write a python code for Design and implement a CNN model to classify multi category JPG images with tensorflow / keras and check accuracy.Predict labels for new images.

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
#Set the parameters
batch_size = 32
image_height = 128
image_width = 128
num_classes = 3
num_epochs =10
```

```
#create the CNN model
model = models.Sequential()
model.add(layers.Conv2D(32, (3,3), activation='relu', input_shape = (image_height, image_
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64, (3,3), activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(128, (3,3), activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss=tf.keras.losses.CategoricalCrossentropy(from_logits=
```

```
# Load and preprocess the image data
train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
train_data_dir = '/content/train'
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(image_height, image_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training')

validation_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(image_height, image_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation')
```

```
Found 11 images belonging to 3 classes.
Found 2 images belonging to 3 classes.
```

```
# Train the model
model.fit(train_generator, validation_data=validation_generator, epochs=num_epochs)

# Evaluate the model
test_data_dir = '/content/test'
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=(image_height, image_width),
    batch_size=batch_size,
    class_mode='categorical')

accuracy = model.evaluate(test_generator)
print("Test accuracy:", accuracy[1])
```

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_a
  self._warn_if_super_not_called()
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step - accuracy: 0.4545 - loss: 1.0367 - val_accuracy:
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 998ms/step - accuracy: 0.5455 - loss: 0.9207 - val_accura
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 559ms/step - accuracy: 0.4545 - loss: 4.5569 - val_accura
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 623ms/step - accuracy: 0.4545 - loss: 1.5533 - val_accura
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 594ms/step - accuracy: 0.5455 - loss: 0.7445 - val_accura
Epoch 6/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 557ms/step - accuracy: 0.5455 - loss: 0.9133 - val_accura
Epoch 7/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 676ms/step - accuracy: 0.5455 - loss: 0.8612 - val_accura
Epoch 8/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 941ms/step - accuracy: 0.5455 - loss: 0.8549 - val_accura
Epoch 9/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 820ms/step - accuracy: 0.5455 - loss: 0.8747 - val_accura
Epoch 10/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.5455 - loss: 0.8349 - val_accuracy:
Found 14 images belonging to 3 classes.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 322ms/step - accuracy: 0.5000 - loss: 0.7699
Test accuracy: 0.5
```

```
# Predict labels for new images
new_image_path = '/content/predict/sun.jpg'
new_image=tf.keras.preprocessing.image.load_img(new_image_path,target_size=(image_height,
new_image=tf.keras.preprocessing.image.img_to_array(new_image)
new_image=new_image/255.0
new_image=tf.expand_dims(new_image,axis=0)
predictions=model.predict(new_image)
predicted_class=tf.argmax(predictions[0]).numpy()
class_labels=['class1','class2','class3']
predicted_label=class_labels[predicted_class]
print("Predicted Class:",predicted_label)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
Predicted Class: class2
```

9. Write a python program for Design and implement a CNN model to classify multi category tiff images with tensorflow/keras and check the accuracy. Check whether your model is overfit/underfit/perfect fit and apply the techniques to avoid overfit and underfit like regulizers, dropouts etc.

```
import tensorflow as tf
from tensorflow.keras import layers, models, regularizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
# Set the parameters
batch_size = 32
image_height = 128
image_width = 128
num_classes = 3
num_epochs = 10
```

```
# Create the CNN model
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(image_height, image_w
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu',
kernel_regularizer=regularizers.l2(0.001)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss=tf.keras.losses.CategoricalCrossentropy(from_logits=
metrics=['accuracy'])
```

```
# Load and preprocess the image data
train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
train_data_dir = '/content/train'
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(image_height, image_width),
    batch_size=batch_size,
    class_mode='categorical',
```

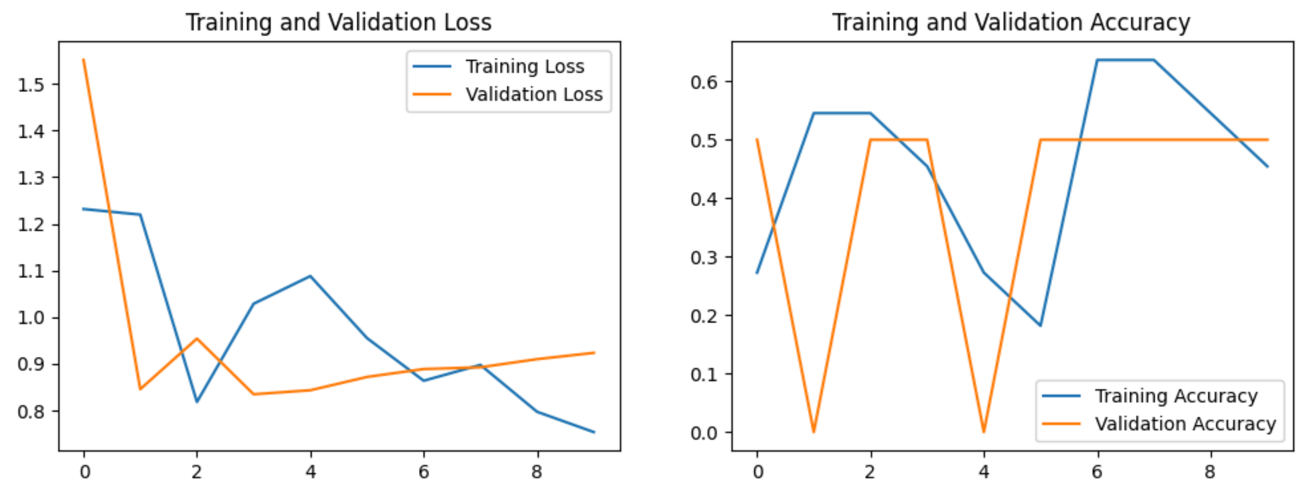
```
    subset='training')

validation_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(image_height, image_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation')
```

```
Found 11 images belonging to 3 classes.
Found 2 images belonging to 3 classes.
```

```
# Train the model
history = model.fit(train_generator, validation_data=validation_generator, epochs=num_epc

# Evaluate the model
test_data_dir = '/content/test'
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=(image_height, image_width),
    batch_size=batch_size,
    class_mode='categorical')

accuracy = model.evaluate(test_generator)
print("Test accuracy:", accuracy[1])
```

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/backend/tensorflow/nn.py:593: UserW
  output, from_logits = _get_logits(
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_a
  self._warn_if_super_not_called()
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - accuracy: 0.2727 - loss: 1.2317 - val_accuracy:
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 999ms/step - accuracy: 0.5455 - loss: 1.2197 - val_accura
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 580ms/step - accuracy: 0.5455 - loss: 0.8182 - val_accura
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 555ms/step - accuracy: 0.4545 - loss: 1.0287 - val_accura
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 554ms/step - accuracy: 0.2727 - loss: 1.0879 - val_accura
Epoch 6/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 547ms/step - accuracy: 0.1818 - loss: 0.9550 - val_accura
Epoch 7/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 609ms/step - accuracy: 0.6364 - loss: 0.8637 - val_accura
Epoch 8/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 638ms/step - accuracy: 0.6364 - loss: 0.8976 - val_accura
Epoch 9/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 836ms/step - accuracy: 0.5455 - loss: 0.7971 - val_accura
Epoch 10/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.4545 - loss: 0.7536 - val_accuracy:
Found 14 images belonging to 3 classes.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 326ms/step - accuracy: 0.5000 - loss: 0.8738
Test accuracy: 0.5
```

```
# Check for overfitting or underfitting
import matplotlib.pyplot as plt
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs_range = range(num_epochs)

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, train_loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, train_acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.show()
```



10. Implement a CNN architectures(LeNet, Alexnet, VGG, etc) model to classify multi category Satellite images withtensorflow / keras and check the accuracy. Check whether your model is overfit / underfit / perfect fit and apply the techniques to avoid overfit and underfit.

```
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
```

```
# Set the parameters
batch_size = 32
image_height = 128
image_width = 128
num_classes = 3
num_epochs = 50
```

```
# Create the LeNet model
def build_lenet_model():
 model = models.Sequential()
 model.add(layers.Conv2D(32, (5, 5), activation='relu', input_shape=(image_height, image_
 model.add(layers.MaxPooling2D((2, 2)))
 model.add(layers.Conv2D(64, (5, 5), activation='relu'))
 model.add(layers.MaxPooling2D((2, 2)))
 model.add(layers.Flatten())
 model.add(layers.Dense(64, activation='relu'))
 model.add(layers.Dense(num_classes, activation='softmax'))
 return model
```

```
# Create the AlexNet model
def build_alexnet_model():
 model = models.Sequential()
 model.add(layers.Conv2D(96, (11, 11), strides=(4, 4), activation='relu', input_shape=(im
 model.add(layers.MaxPooling2D((3, 3), strides=(2, 2)))
 model.add(layers.Conv2D(256, (5, 5), activation='relu', padding="same"))
 model.add(layers.MaxPooling2D((3, 3), strides=(2, 2)))
 model.add(layers.Conv2D(384, (3, 3), activation='relu', padding="same"))
 model.add(layers.Conv2D(384, (3, 3), activation='relu', padding="same"))
 model.add(layers.Conv2D(256, (3, 3), activation='relu', padding="same"))
 model.add(layers.MaxPooling2D((3, 3), strides=(2, 2)))
 model.add(layers.Flatten())
 model.add(layers.Dense(4096, activation='relu'))
 model.add(layers.Dense(4096, activation='relu'))
 model.add(layers.Dense(num_classes, activation='softmax'))
 return model
```

```
# Create the VGG model
def build_vgg_model():
 model = models.Sequential()
 model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=(imag
 model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
 model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))
 model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
 model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
 model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))
 model.add(layers.Conv2D(256, (3, 3), activation='relu', padding='same'))
```

```
    model.add(layers.Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))
    model.add(layers.Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))
    model.add(layers.Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.Dense(num_classes, activation='softmax'))
    return model
```

```
# Select the model architecture
model = build_vgg_model() # Change the function name to choose a different architecture

# Compile the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:1
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# Load and preprocess the image data
train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
train_data_dir = '/content/train'
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(image_height, image_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training')

validation_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(image_height, image_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation')
```

```
Found 11 images belonging to 3 classes.
Found 2 images belonging to 3 classes.
```

```
# Apply data augmentation to avoid overfitting
train_datagen_augmented = ImageDataGenerator(
    rescale=1./255,
```

```
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

train_generator_augmented = train_datagen_augmented.flow_from_directory(
    train_data_dir,
    target_size=(image_height, image_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training')
```

```
Found 13 images belonging to 3 classes.
```

```
# Early stopping to avoid overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=5)

# Train the model
history = model.fit(
    train_generator_augmented,
    validation_data=validation_generator,
    epochs=num_epochs,
    callbacks=[early_stopping])
```

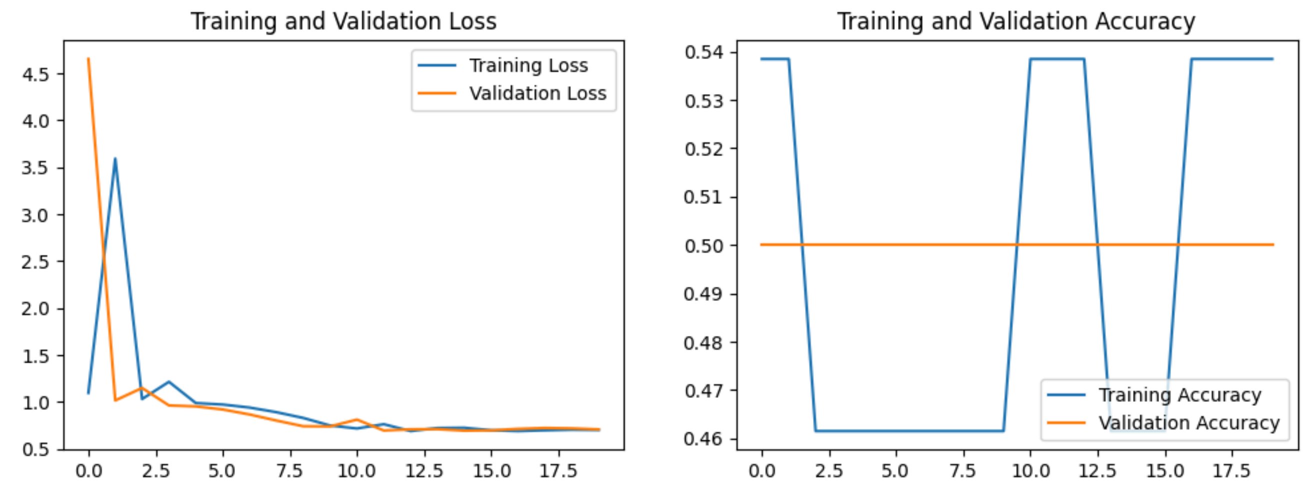
```
Epoch 1/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 16s 16s/step - accuracy: 0.5385 - loss: 1.0981 - val_accurac
Epoch 2/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 20s 20s/step - accuracy: 0.5385 - loss: 3.5924 - val_accurac
Epoch 3/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 10s 10s/step - accuracy: 0.4615 - loss: 1.0324 - val_accurac
Epoch 4/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 12s 12s/step - accuracy: 0.4615 - loss: 1.2159 - val_accurac
Epoch 5/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 20s 20s/step - accuracy: 0.4615 - loss: 0.9895 - val_accurac
Epoch 6/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 20s 20s/step - accuracy: 0.4615 - loss: 0.9740 - val_accurac
Epoch 7/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 21s 21s/step - accuracy: 0.4615 - loss: 0.9417 - val_accurac
Epoch 8/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 11s 11s/step - accuracy: 0.4615 - loss: 0.8924 - val_accurac
Epoch 9/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 10s 10s/step - accuracy: 0.4615 - loss: 0.8310 - val_accurac
Epoch 10/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 11s 11s/step - accuracy: 0.4615 - loss: 0.7502 - val_accurac
Epoch 11/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 20s 20s/step - accuracy: 0.5385 - loss: 0.7189 - val_accurac
Epoch 12/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 10s 10s/step - accuracy: 0.5385 - loss: 0.7644 - val_accurac
Epoch 13/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 11s 11s/step - accuracy: 0.5385 - loss: 0.6912 - val_accurac
Epoch 14/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 11s 11s/step - accuracy: 0.4615 - loss: 0.7239 - val_accurac
Epoch 15/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 11s 11s/step - accuracy: 0.4615 - loss: 0.7265 - val_accurac
```

```
Epoch 16/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 10s 10s/step - accuracy: 0.4615 - loss: 0.6996 - val_accurac
Epoch 17/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 21s 21s/step - accuracy: 0.5385 - loss: 0.6913 - val_accurac
Epoch 18/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 11s 11s/step - accuracy: 0.5385 - loss: 0.6992 - val_accurac
Epoch 19/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 12s 12s/step - accuracy: 0.5385 - loss: 0.7055 - val_accurac
Epoch 20/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 10s 10s/step - accuracy: 0.5385 - loss: 0.7028 - val_accurac
```

```
# Evaluate the model
test_data_dir = '/content/test'
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=(image_height, image_width),
    batch_size=batch_size,
    class_mode='categorical')

accuracy = model.evaluate(test_generator)
print("Test accuracy:", accuracy[1])
```

```
Found 14 images belonging to 3 classes.
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - accuracy: 0.5000 - loss: 0.7092
Test accuracy: 0.5
```

```
# Check for overfitting or underfitting
import matplotlib.pyplot as plt
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

epochs_range = range(len(train_loss))

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, train_loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, train_acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.show()
```



11. Implement an Auto encoder to de-noise image

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

```
# Define the autoencoder architecture
input_img = keras.Input(shape=(28, 28, 1))

# Encoder
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2, 2), padding='same')(x)

# Decoder
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = keras.Model(input_img, decoded)
```

```
#Compile the model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Load the dataset (example: MNIST)
(x_train, _), (x_test, _) = keras.datasets.mnist.load_data()
# Normalize and reshape the input images
x_train = x_train.astype('float32') / 255.0
```

```
x_train = tf.expand_dims(x_train, axis=-1)
x_test = x_test.astype('float32') / 255.0
x_test = tf.expand_dims(x_test, axis=-1)
```

```
# Add random noise to the training and test images
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * tf.random.normal(shape=x_train.shape)
x_test_noisy = x_test + noise_factor * tf.random.normal(shape=x_test.shape)

# Clip the values to [0, 1]
x_train_noisy = tf.clip_by_value(x_train_noisy, clip_value_min=0.0, clip_value_max=1.0)
x_test_noisy = tf.clip_by_value(x_test_noisy, clip_value_min=0.0, clip_value_max=1.0)
```

```
#Train the autoencoder
autoencoder.fit(x_train_noisy,
                x_train, epochs=10,
                batch_size=128,
                shuffle=True,
                validation_data=(x_test_noisy, x_test))

#Evaluate the autoencode
decoded_imgs = autoencoder.predict(x_test_noisy)
```

```
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 73s 150ms/step - loss: 0.1062 - val_loss: 0.1052
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 85s 158ms/step - loss: 0.1054 - val_loss: 0.1045
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 75s 160ms/step - loss: 0.1053 - val_loss: 0.1044
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 78s 152ms/step - loss: 0.1052 - val_loss: 0.1041
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 81s 151ms/step - loss: 0.1048 - val_loss: 0.1043
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 88s 164ms/step - loss: 0.1047 - val_loss: 0.1043
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 80s 161ms/step - loss: 0.1045 - val_loss: 0.1040
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 84s 166ms/step - loss: 0.1046 - val_loss: 0.1037
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 88s 179ms/step - loss: 0.1043 - val_loss: 0.1036
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 130s 154ms/step - loss: 0.1043 - val_loss: 0.1035
313/313 ━━━━━━━━━━━━━━━━━━━━ 6s 19ms/step
```

```
# Calculate the reconstruction loss (MSE)
mse = tf.keras.losses.MeanSquaredError()
reconstruction_loss = mse(x_test, decoded_imgs)
print(f"Reconstruction Loss: {reconstruction_loss:.4f}")
```

```
Reconstruction Loss: 0.0134
```

```
#Visualize the results
import matplotlib.pyplot as plt

n = 10 # Number of images to display
plt.figure(figsize=(20, 4))

for i in range(n):
 # Original images
 ax = plt.subplot(2, n, i + 1)
 plt.imshow(tf.squeeze(x_test_noisy[i]))
 plt.title("Original + Noise")
 plt.gray()
 ax.get_xaxis().set_visible(False)
 ax.get_yaxis().set_visible(False)

 # Decoded images
 ax = plt.subplot(2, n, i + n + 1)
 plt.imshow(tf.squeeze(decoded_imgs[i]))
 plt.title("Denoised")
 plt.gray()
 ax.get_xaxis().set_visible(False)
 ax.get_yaxis().set_visible(False)

plt.show()
```