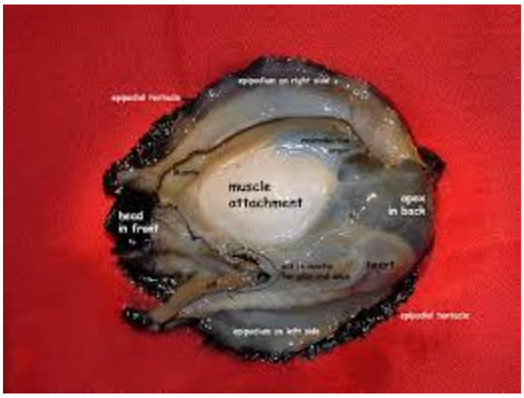
Lab 1 DL

1. Demonstrate the process of creating a simple feed-forward neural network for the Abalone dataset using Tensorflow and Keras libraries.



balone are marine snails that belong to a group of invertebrates (animals lacking a backbone or other forms of internal skeleton) called molluscs. Molluscs also include common bivalves such as scallops, oysters, mussels, pippies, and cockles, as well as octopus, squid, and cuttlefish. Most species of abalone have a moderate to heavily calcified snail-like shell that is flattened

- **Dataset Loading:**
  - The dataset is read from a local CSV file (`abalone.csv`). You can preview it using `head()`, `info()`, and `shape` to understand the data structure, types, and dimensions.
- **Exploratory Data Analysis:**
  - EDA involves checking for null values, visualizing categorical data (`Sex` distribution via a pie chart), and analyzing grouped statistics (e.g., mean values grouped by `Sex`).
- **Preprocessing:**
  - Convert categorical variables into numeric form using one-hot encoding.
  - Normalize the features and target variable for better training stability.
  - Split the data into training and testing sets.
- **Model Building and Training:**
  - A feed-forward neural network is created with two hidden layers and one output layer.
  - The model is trained using the `fit()` function, and validation performance is monitored.
- **Evaluation:**
  - The trained model is evaluated using test data to compute loss and MAE.
  - Performance over epochs (loss and MAE) is visualized for both training and validation.
- **Predictions:**
  - Predictions are made for the test set, and results are compared with actual values.
  - Actual and predicted values are visualized using scatter plots to assess model performance.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
print ("Demo by karthik_22a81a6154\n\n")
```

```
Demo by karthik_22a81a6154
```

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data"
df = pd.read_csv(url, header=None)
# Display the first 5 rows to understand the structure of the dataset.
print("df = pd.read_csv(url, header=None)")
print("Dataset Preview (df.head()):")
print(df.head())
```

```
df = pd.read_csv(url, header=None)
Dataset Preview (df.head()):
   0      1      2      3       4       5       6      7   8
0  M  0.455  0.365  0.095  0.5140  0.2245  0.1010  0.150  15
1  M  0.350  0.265  0.090  0.2255  0.0995  0.0485  0.070   7
2  F  0.530  0.420  0.135  0.6770  0.2565  0.1415  0.210   9
3  M  0.440  0.365  0.125  0.5160  0.2155  0.1140  0.155  10
4  I  0.330  0.255  0.080  0.2050  0.0895  0.0395  0.055   7
```

```
df1 = pd.read_csv(url) # Automatically reads the first row as headers
print("df1 = pd.read_csv(url)")
print("Dataset Preview (df1.head):")
print(df1.head())
```

```
df1 = pd.read_csv(url)
Dataset Preview (df1.head):
   M  0.455  0.365  0.095   0.514  0.2245   0.101   0.15  15
0  M  0.350  0.265  0.090  0.2255  0.0995  0.0485  0.070   7
1  F  0.530  0.420  0.135  0.6770  0.2565  0.1415  0.210   9
2  M  0.440  0.365  0.125  0.5160  0.2155  0.1140  0.155  10
3  I  0.330  0.255  0.080  0.2050  0.0895  0.0395  0.055   7
4  I  0.425  0.300  0.095  0.3515  0.1410  0.0775  0.120   8
```

```
columns = ['Sex', 'Length', 'Diameter', 'Height', 'WholeWeight',
'ShuckedWeight', 'VisceraWeight', 'ShellWeight', 'Rings']

df.columns = columns
print("Dataset Preview:")
print(df.head())
```

```
Dataset Preview:
  Sex  Length  Diameter  Height  WholeWeight  ShuckedWeight  VisceraWeight  \
0   M   0.455     0.365   0.095       0.5140         0.2245         0.1010
1   M   0.350     0.265   0.090       0.2255         0.0995         0.0485
2   F   0.530     0.420   0.135       0.6770         0.2565         0.1415
3   M   0.440     0.365   0.125       0.5160         0.2155         0.1140
4   I   0.330     0.255   0.080       0.2050         0.0895         0.0395

   ShellWeight  Rings
0        0.150     15
1        0.070      7
2        0.210      9
3        0.155     10
4        0.055      7
```

```
print("\nDataset Shape (Rows, Columns):")
print(df.shape)
print("\nDataset Info:")
df.info()
```

```
Dataset Shape (Rows, Columns):
(4177, 9)

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Sex            4177 non-null   object
 1   Length         4177 non-null   float64
 2   Diameter       4177 non-null   float64
 3   Height         4177 non-null   float64
 4   WholeWeight    4177 non-null   float64
 5   ShuckedWeight  4177 non-null   float64
 6   VisceraWeight  4177 non-null   float64
 7   ShellWeight    4177 non-null   float64
 8   Rings          4177 non-null   int64
dtypes: float64(7), int64(1), object(1)
memory usage: 293.8+ KB
```

```
print("\nStatistical Summary of Dataset:")
print(df.describe().T)
```

```
Statistical Summary of Dataset:
                count      mean       std     min     25%     50%     75%  \
Length         4177.0  0.523992  0.120093  0.0750  0.4500  0.5450   0.615
Diameter       4177.0  0.407881  0.099240  0.0550  0.3500  0.4250   0.480
Height         4177.0  0.139516  0.041827  0.0000  0.1150  0.1400   0.165
WholeWeight    4177.0  0.828742  0.490389  0.0020  0.4415  0.7995   1.153
ShuckedWeight  4177.0  0.359367  0.221963  0.0010  0.1860  0.3360   0.502
VisceraWeight  4177.0  0.180594  0.109614  0.0005  0.0935  0.1710   0.253
ShellWeight    4177.0  0.238831  0.139203  0.0015  0.1300  0.2340   0.329
Rings          4177.0  9.933684  3.224169  1.0000  8.0000  9.0000  11.000
```

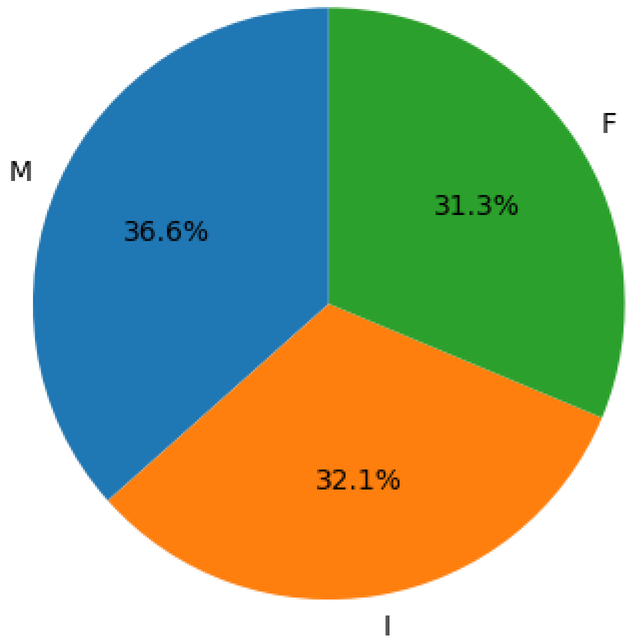
```
                     max
Length            0.8150
Diameter          0.6500
Height            1.1300
WholeWeight       2.8255
ShuckedWeight     1.4880
VisceraWeight     0.7600
ShellWeight       1.0050
Rings            29.0000
```

```
print("\nNumber of Missing Values in Each Column:")
print(df.isnull().sum())
```

```
Number of Missing Values in Each Column:
Sex              0
Length           0
Diameter         0
Height           0
WholeWeight      0
ShuckedWeight    0
VisceraWeight    0
ShellWeight      0
Rings            0
dtype: int64
```

```
x = df['Sex'].value_counts()
labels = x.index # Unique categories in 'Sex'
values = x.values # Counts of each category
plt.pie(values, labels=labels, autopct='%1.1f%%', startangle=90)
plt.title('Distribution of Sex karthik_22a81a6154')
plt.show()
```

Distribution of Sex karthik_22a81a6154




```
df = pd.get_dummies(df, columns=['Sex'], drop_first=True)
```

```
X = df.drop('Rings', axis=1)
y = df['Rings']
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
y = y / y.max()
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,random_sta
```

```
model = Sequential([
Dense(64, input_dim=X_train.shape[1], activation='relu'),
Dense(32, activation='relu'),
Dense(1, activation='linear')
])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarnir
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
```

```
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10,batch_s
```

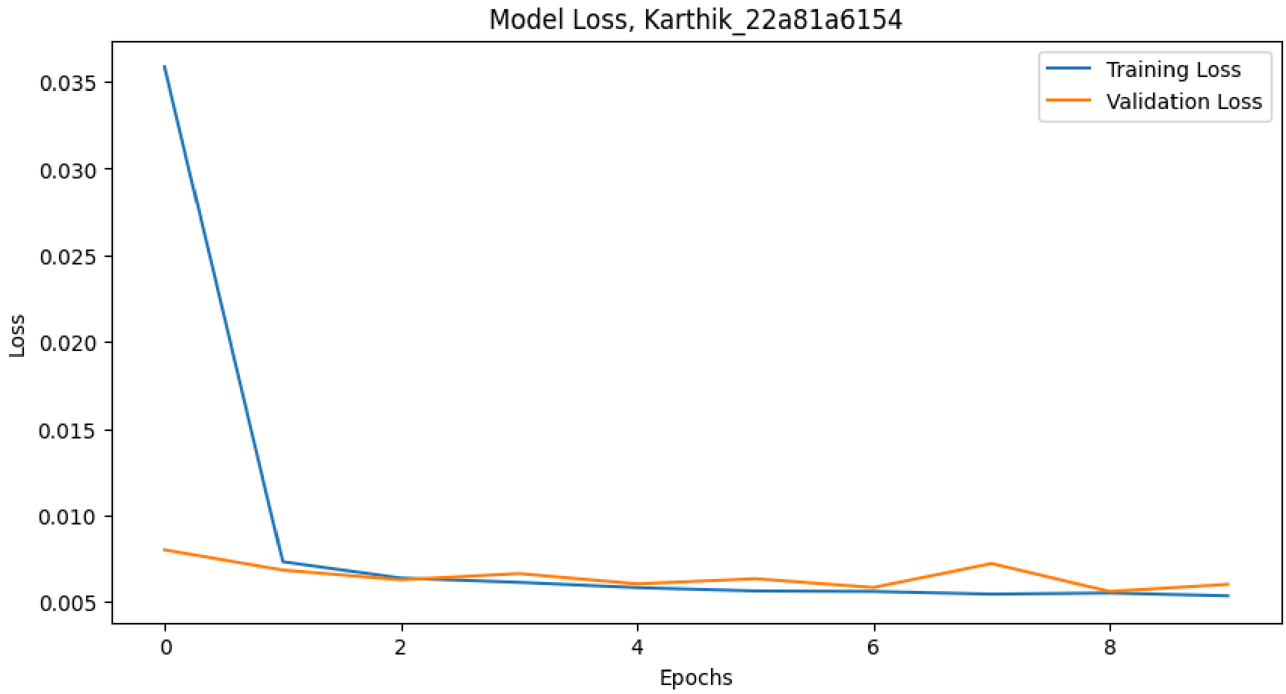
```
Epoch 1/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - loss: 0.0624 - mae: 0.1622 - val_loss: 0.6
```

```
Epoch 2/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0087 - mae: 0.0636 - val_loss: 0.6
Epoch 3/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0061 - mae: 0.0564 - val_loss: 0.6
Epoch 4/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0062 - mae: 0.0576 - val_loss: 0.6
Epoch 5/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.0059 - mae: 0.0562 - val_loss: 0.6
Epoch 6/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0055 - mae: 0.0537 - val_loss: 0.6
Epoch 7/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0053 - mae: 0.0529 - val_loss: 0.6
Epoch 8/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0050 - mae: 0.0525 - val_loss: 0.6
Epoch 9/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0061 - mae: 0.0558 - val_loss: 0.6
Epoch 10/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0058 - mae: 0.0545 - val_loss: 0.6
```

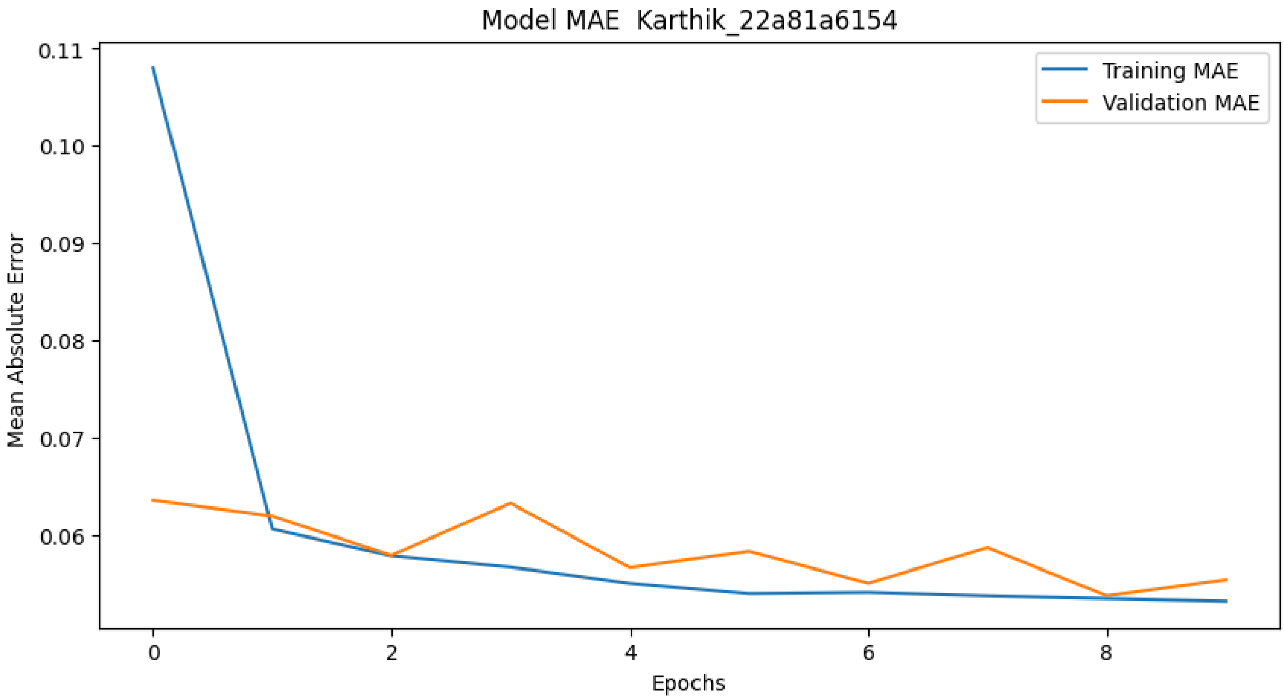
```python
test_loss, test_mae = model.evaluate(X_test, y_test)
print(f"\nTest Loss: {test_loss}")
print(f"Test MAE: {test_mae}")
```

```
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0064 - mae: 0.0562

Test Loss: 0.006020877510309219
Test MAE: 0.055393118411302567
```

```python
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss, Karthik_22a81a6154')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
plt.figure(figsize=(10, 5))
plt.plot(history.history['mae'], label='Training MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.title('Model MAE  Karthik_22a81a6154')
plt.xlabel('Epochs')
plt.ylabel('Mean Absolute Error ')
plt.legend()
plt.show()
```



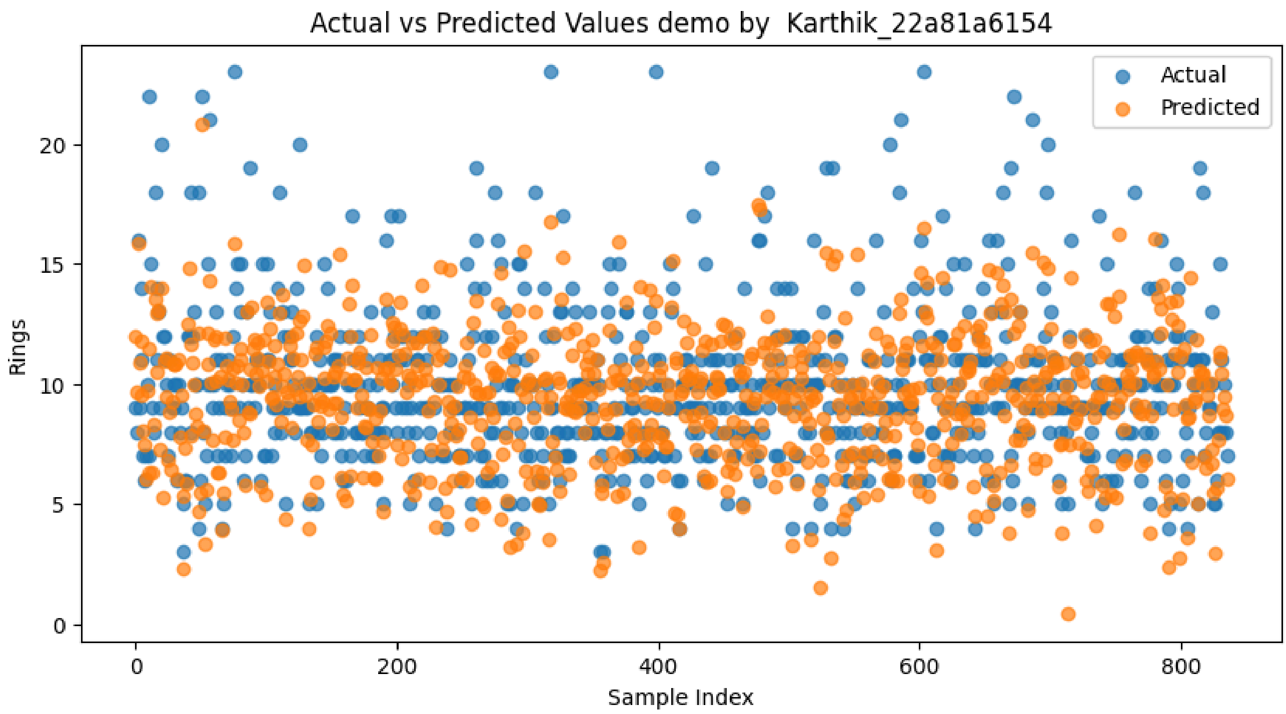

```
y_pred = model.predict(X_test)
```

27/27 ━━━━━━━━━━━━━━━━━━━━ **0s** 3ms/step

```
y_pred_original = y_pred.flatten() * df['Rings'].max()
y_test_original = y_test * df['Rings'].max()


print("\nSample Predictions:")
for i in range(10):
    print(f"Actual: {y_test_original.iloc[i]:.2f}, Predicted:  {y_pred_original[i]:.2f}"
```

```
Sample Predictions:
Actual: 9.00, Predicted:  11.98
Actual: 8.00, Predicted:  9.63
Actual: 16.00, Predicted:  15.86
Actual: 9.00, Predicted:  10.85
Actual: 14.00, Predicted:  11.81
Actual: 11.00, Predicted:  9.53
Actual: 7.00, Predicted:  8.06
Actual: 6.00, Predicted:  7.47
Actual: 7.00, Predicted:  6.04
Actual: 10.00, Predicted:  9.72
```

```
plt.figure(figsize=(10, 5))
plt.scatter(range(len(y_test_original)), y_test_original, label='Actual',  alpha=0.7)
plt.scatter(range(len(y_pred_original)), y_pred_original, label='Predicted',  alpha=0.7)
plt.title('Actual vs Predicted Values demo by  Karthik_22a81a6154')
plt.xlabel('Sample Index')
plt.ylabel('Rings')
plt.legend()
plt.show()
```



Start coding or generate with AI.

EXP-2

Demonstrate the process of saving and loading weights of the neural network constructed in experiment 1 manually and add with checkpoints

```
import os
from tensorflow.keras.callbacks import ModelCheckpoint
print("Exp-02 [Demo by Karthik_22A81A6154]\n")
model.save_weights('model_weights_manual.weights.h5')
print("Model weights saved manually to 'model_weights_manual.weights.h5'")
```

```
Exp-02 [Demo by Karthik_22A81A6154]

Model weights saved manually to 'model_weights_manual.weights.h5'
```

```
del model
model = Sequential([
    Dense(64, input_dim=X_train.shape[1], activation='relu'),
    Dense(32, activation='relu'),
    Dense(1, activation='linear')
])
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
model.load_weights('model_weights_manual.weights.h5')
print("Model weights loaded successfully.")
test_loss, test_mae = model.evaluate(X_test, y_test)
print(f"Test Loss after loading weights: {test_loss}")
print(f"Test MAE after loading weights: {test_mae}")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarnir
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.11/dist-packages/keras/src/saving/saving_lib.py:757: UserWarni
  saveable.load_own_variables(weights_store.get(inner_path))
Model weights loaded successfully.
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0064 - mae: 0.0562
Test Loss after loading weights: 0.006020877510309219
Test MAE after loading weights: 0.055393118411302567
```

```
checkpoint_dir = './checkpoints'
os.makedirs(checkpoint_dir, exist_ok=True)
checkpoint_path = os.path.join(checkpoint_dir, 'model_checkpoint.weights.h5')
checkpoint_callback = ModelCheckpoint(
filepath=checkpoint_path,
save_weights_only=True,
save_best_only=True,
monitor='val_loss',
mode='min',
verbose=1
)


print("\nTraining the model with checkpointing...")
history_with_checkpoint = model.fit(
```

```
X_train, y_train,
validation_data=(X_test, y_test),
epochs=10,
batch_size=32,
callbacks=[checkpoint_callback]
)
```

```
Training the model with checkpointing...
Epoch 1/10
 95/105 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0056 - mae: 0.0573
Epoch 1: val_loss improved from inf to 0.00572, saving model to ./checkpoints/model_c
105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0057 - mae: 0.0572 - val_loss: 0.0
Epoch 2/10
 88/105 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0050 - mae: 0.0514
Epoch 2: val_loss did not improve from 0.00572
105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0051 - mae: 0.0517 - val_loss: 0.0
Epoch 3/10
105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0089 - mae: 0.0608
Epoch 3: val_loss did not improve from 0.00572
105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0089 - mae: 0.0608 - val_loss: 0.0
Epoch 4/10
 87/105 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0071 - mae: 0.0566
Epoch 4: val_loss did not improve from 0.00572
105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.0069 - mae: 0.0564 - val_loss: 0.0
Epoch 5/10
103/105 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0057 - mae: 0.0541
Epoch 5: val_loss did not improve from 0.00572
105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0057 - mae: 0.0540 - val_loss: 0.0
Epoch 6/10
 85/105 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0054 - mae: 0.0541
Epoch 6: val_loss improved from 0.00572 to 0.00567, saving model to ./checkpoints/moc
105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0054 - mae: 0.0538 - val_loss: 0.0
Epoch 7/10
 83/105 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0048 - mae: 0.0502
Epoch 7: val_loss did not improve from 0.00567
105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0049 - mae: 0.0505 - val_loss: 0.0
Epoch 8/10
 89/105 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0051 - mae: 0.0519
Epoch 8: val_loss did not improve from 0.00567
105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0051 - mae: 0.0521 - val_loss: 0.0
Epoch 9/10
 91/105 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0051 - mae: 0.0519
Epoch 9: val_loss did not improve from 0.00567
105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0051 - mae: 0.0521 - val_loss: 0.0
Epoch 10/10
 87/105 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0053 - mae: 0.0538
Epoch 10: val_loss did not improve from 0.00567
105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0053 - mae: 0.0535 - val_loss: 0.0
```

```
model.load_weights(checkpoint_path)
print("Model weights loaded successfully from the checkpoint.")
test_loss_checkpoint, test_mae_checkpoint = model.evaluate(X_test, y_test)
print(f"Test Loss after loading checkpoint weights: {test_loss_checkpoint}")
print(f"Test MAE after loading checkpoint weights: {test_mae_checkpoint}")
```

```
Model weights loaded successfully from the checkpoint.
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0059 - mae: 0.0551
Test Loss after loading checkpoint weights: 0.00566750718280673
Test MAE after loading checkpoint weights: 0.05467785894870758
```

EXP-3

Construct a regression model for predicting the fuel efficiency of cars using the MPG dataset.

The Miles Per Gallon (MPG) dataset is commonly used for building regression models to predict a car's fuel efficiency based on various attributes. The goal is to construct a regression model that can accurately estimate MPG (Miles Per Gallon), which represents fuel efficiency.

Dataset Description

The dataset consists of multiple features describing different aspects of a car, including:

- MPG (Miles Per Gallon) – Target variable (fuel efficiency)
- Cylinders – Number of cylinders in the engine
- Displacement – Engine displacement (in cubic inches)
- Horsepower – Power output of the engine
- Weight – Vehicle weight (in lbs)
- Acceleration – Time taken to reach 60 mph (in seconds)
- Model Year – Year of manufacture
- Origin – Country of manufacture

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

```
url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data'
columns = ['MPG', 'Cylinders', 'Displacement', 'Horsepower', 'Weight','Acceleration', 'Model Year', 'Origin']
data = pd.read_csv(url, names=columns, na_values='?', comment='\t', sep=' ',skipinitialspace=True)
print("Demo by karthik_22a81a6154")
```

```
Demo by karthik_22a81a6154
```

```
print("Dataset preview:")
print(data.head())
```

```
Dataset preview:
    MPG  Cylinders  Displacement  Horsepower  Weight  Acceleration  \
0  18.0          8         307.0       130.0  3504.0          12.0
1  15.0          8         350.0       165.0  3693.0          11.5
2  18.0          8         318.0       150.0  3436.0          11.0
3  16.0          8         304.0       150.0  3433.0          12.0
4  17.0          8         302.0       140.0  3449.0          10.5

   Model Year  Origin
0          70       1
1          70       1
2          70       1
3          70       1
4          70       1
```

```
print("\nMissing values in each column:")
print(data.isnull().sum())
```

```

Missing values in each column:
MPG             0
Cylinders       0
Displacement    0
Horsepower      6
Weight          0
Acceleration    0
Model Year      0
Origin          0
dtype: int64
```

```
data = data.dropna()
data['Origin'] = data['Origin'].map({1: 'USA', 2: 'Europe', 3: 'Japan'})
data = pd.get_dummies(data, columns=['Origin'], drop_first=True)
```

```
<ipython-input-50-e4b60a4ada07>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  data['Origin'] = data['Origin'].map({1: 'USA', 2: 'Europe', 3: 'Japan'})
```

```
X = data.drop('MPG', axis=1)
y = data['MPG']
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,random_state=42)
print("\nTraining set shape:")
print("X_train.shape", X_train.shape)
print("y_train.shape", y_train.shape)
print("\nTest set shape:")
print("X_test.shape", X_test.shape)
print("y_test.shape", y_test.shape)
```

```
Training set shape:
X_train.shape (313, 8)
y_train.shape (313,)

Test set shape:
X_test.shape (79, 8)
y_test.shape (79,)
```

```
model = Sequential([
Dense(64, activation='relu', input_dim=X_train.shape[1]),
Dense(32, activation='relu'),
Dense(1, activation='linear')
])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` arg
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

```
history = model.fit(X_train, y_train, validation_split=0.2, epochs=10,batch_size=32, verbose=1)
```

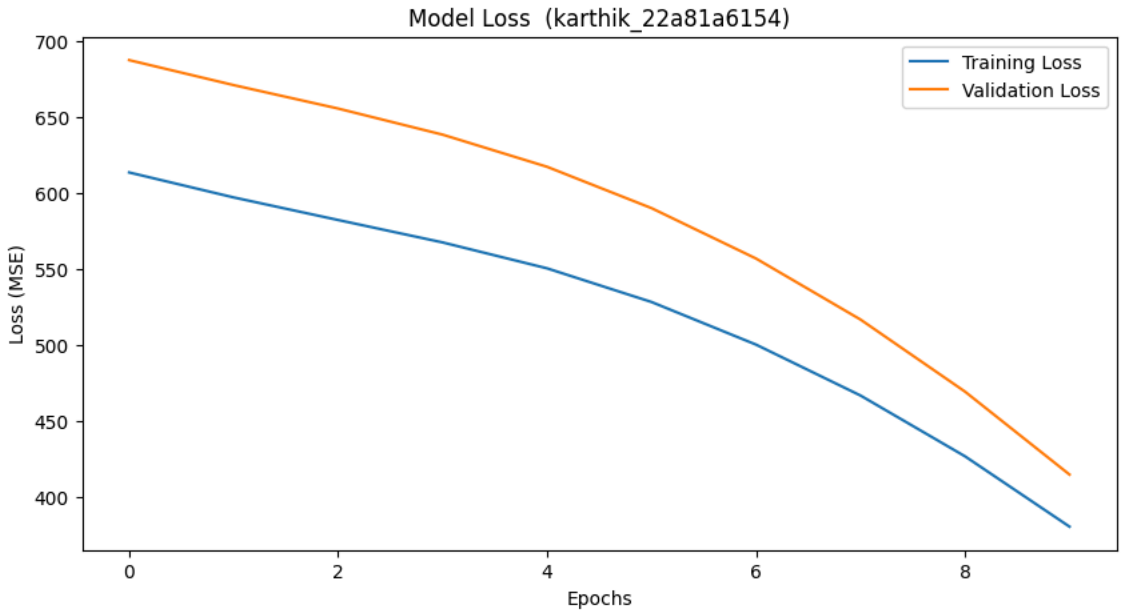
```
Epoch 1/10
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 623.4109 - mae: 23.5587 - val_loss: 687.7822 - val_mae: 25.0080
Epoch 2/10
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 644.3091 - mae: 23.9120 - val_loss: 671.3466 - val_mae: 24.6857
Epoch 3/10
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 578.6078 - mae: 22.7092 - val_loss: 655.9197 - val_mae: 24.3706
Epoch 4/10
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 557.4322 - mae: 22.2489 - val_loss: 638.7531 - val_mae: 24.0136
Epoch 5/10
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 572.0983 - mae: 22.4109 - val_loss: 617.5919 - val_mae: 23.5721
Epoch 6/10
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 539.0898 - mae: 21.7309 - val_loss: 590.3811 - val_mae: 22.9993
Epoch 7/10
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 500.1201 - mae: 20.9478 - val_loss: 557.2279 - val_mae: 22.2805
Epoch 8/10
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 457.6802 - mae: 19.8121 - val_loss: 517.0831 - val_mae: 21.3820
Epoch 9/10
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 435.0889 - mae: 19.3661 - val_loss: 469.6728 - val_mae: 20.2831
Epoch 10/10
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 398.5100 - mae: 18.3814 - val_loss: 415.1200 - val_mae: 18.9542
```

```
test_loss, test_mae = model.evaluate(X_test, y_test)
print(f"\nTest Loss (MSE): {test_loss}")
print(f"Test MAE: {test_mae}")
```

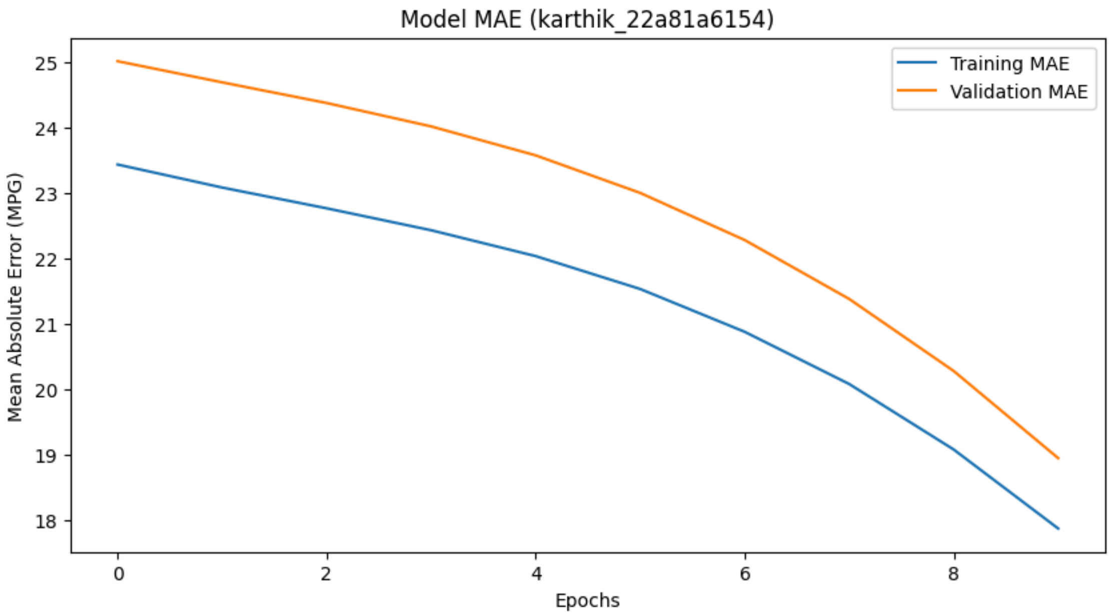
```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 338.0323 - mae: 17.0122

Test Loss (MSE): 324.1715393066406
Test MAE: 16.55548095703125
```

```
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss  (karthik_22a81a6154)' )
plt.xlabel('Epochs')
plt.ylabel('Loss (MSE)')
```

```
plt.legend()
plt.show()
```

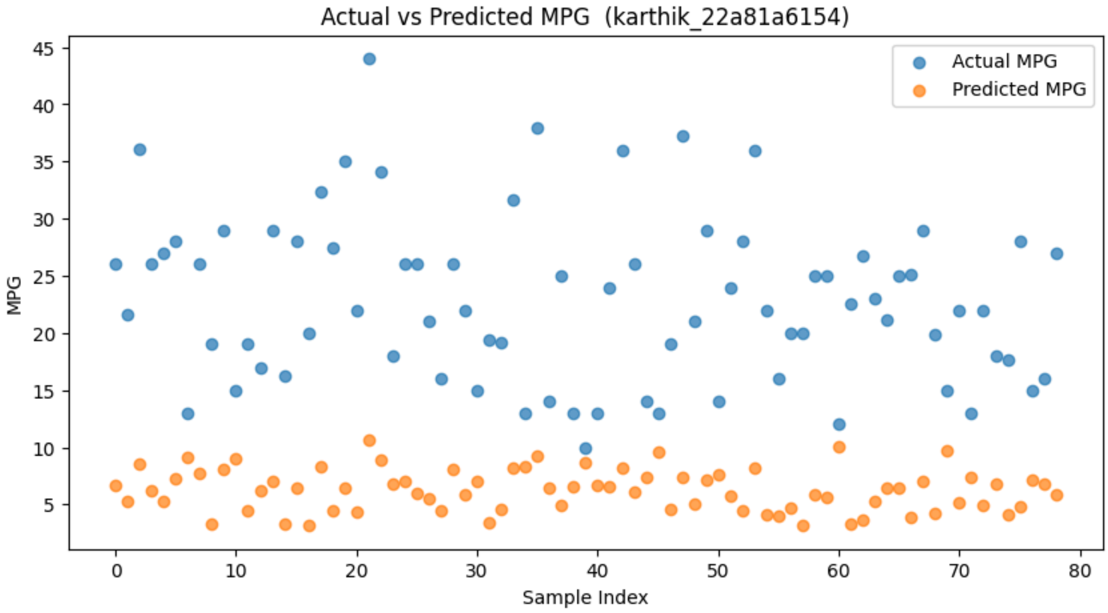



```
plt.figure(figsize=(10, 5))
plt.plot(history.history['mae'], label='Training MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.title('Model MAE (karthik_22a81a6154)')
plt.xlabel('Epochs')
plt.ylabel('Mean Absolute Error (MPG)')
plt.legend()
plt.show()
```




```
y_pred = model.predict(X_test)
# Visualize predictions vs actual MPG values
plt.figure(figsize=(10, 5))
plt.scatter(range(len(y_test)), y_test, label='Actual MPG', alpha=0.7)
plt.scatter(range(len(y_pred)), y_pred, label='Predicted MPG', alpha=0.7)
plt.title('Actual vs Predicted MPG  (karthik_22a81a6154)')
plt.xlabel('Sample Index')
plt.ylabel('MPG')
plt.legend()
plt.show()
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
```



EXP-4

Develop a feed-forward neural network on the MNIST-Handwritten digits dataset

The MNIST dataset is a benchmark dataset in machine learning and deep learning, consisting of 70,000 grayscale images of handwritten digits (0-9). Each image is 28x28 pixels, and the task is to classify the digits using a feed-forward neural network (FNN).

Dataset Description

- Training Set: 60,000 images
- Testing Set: 10,000 images
- Image Size: 28 × 28 pixels
- Number of Classes: 10 (Digits 0-9)

Each image is represented as a flattened 784-dimensional vector (28×28 = 784), where each pixel has a grayscale intensity between 0 and 255.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

print(" Expt 4: Karthik_22a81a6154 ")
```

```
 Expt 4: Karthik_22a81a6154
```

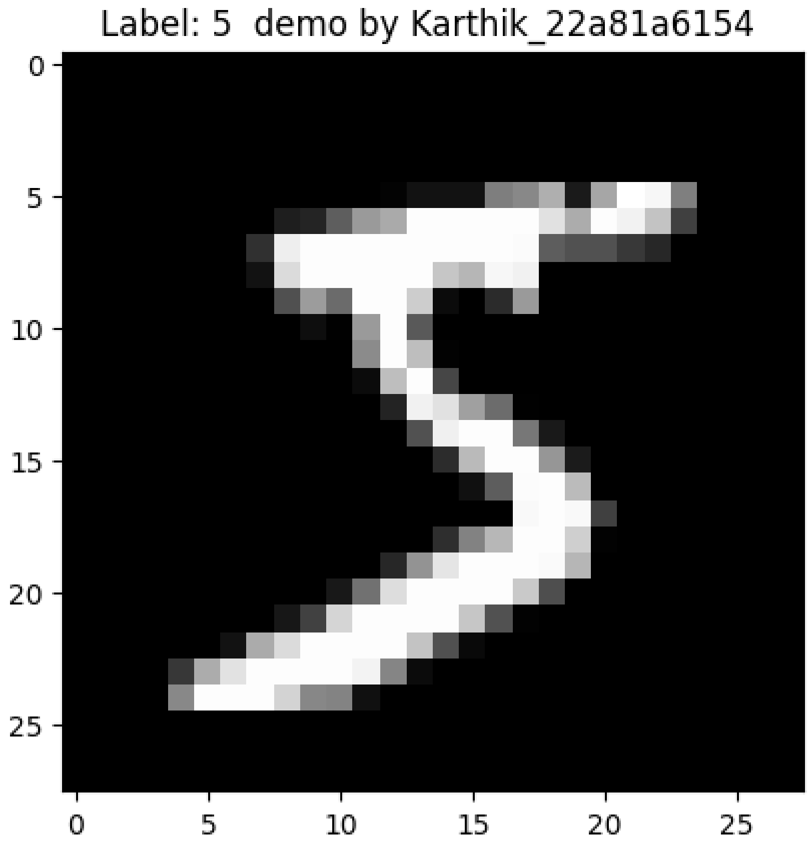
```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

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

```
print("Training data shape:", X_train.shape)
print("Testing data shape:", X_test.shape)
```

```
Training data shape: (60000, 28, 28)
Testing data shape: (10000, 28, 28)
```

```
plt.imshow(X_train[0], cmap='gray')
plt.title(f"Label: {y_train[0]}  demo by Karthik_22a81a6154 ")
plt.show()
```




```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

```
X_train = X_train.reshape(-1, 28 * 28)
X_test = X_test.reshape(-1, 28 * 28)
print("Training data shape after flattening:", X_train.shape)
```

```
Training data shape after flattening: (60000, 784)
```

```
y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=10)
```

```
model = Sequential([
    # First hidden layer with 128 neurons and ReLU activation
    Dense(128, activation='relu', input_dim=28 * 28),
    # Second hidden layer with 64 neurons and ReLU activation
    Dense(64, activation='relu'),
    # Output layer with 10 neurons (one for each class) and softmax activation
    Dense(10, activation='softmax')
])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarnir
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

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

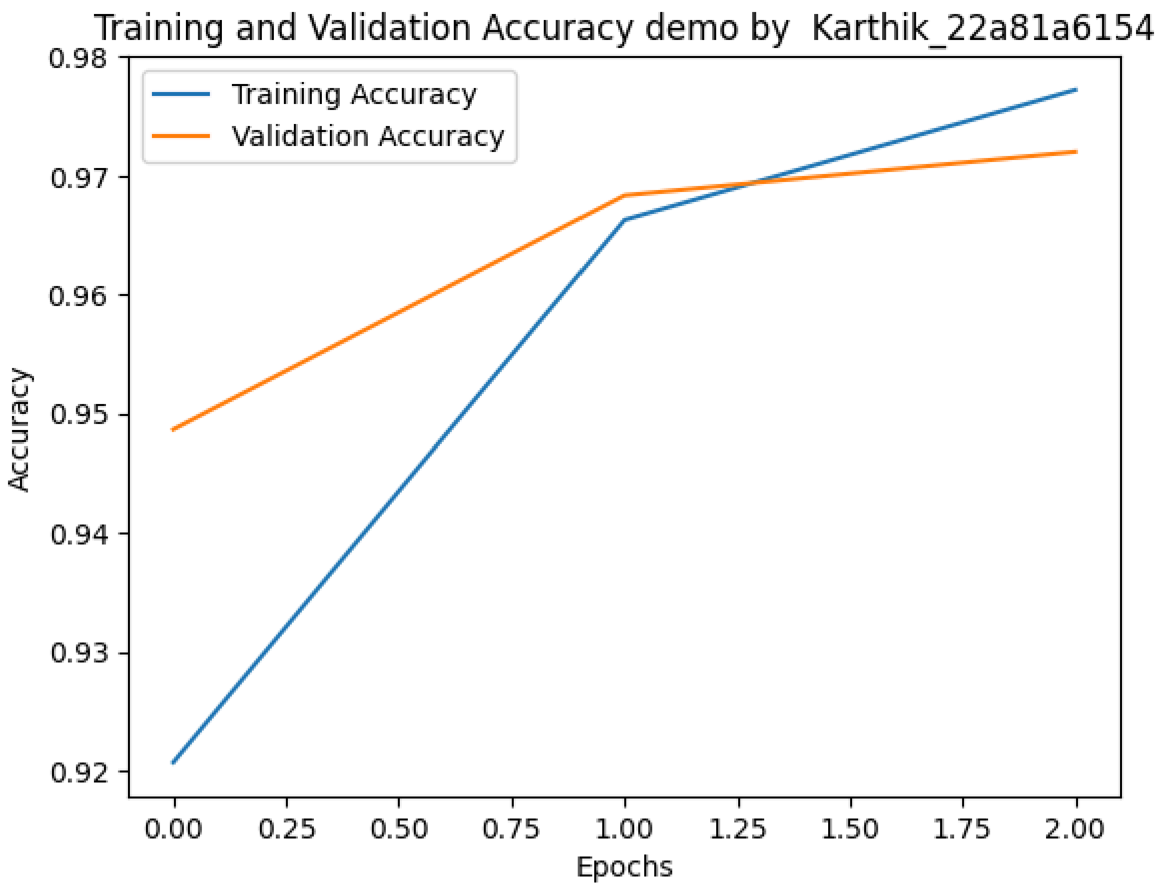
```
history = model.fit(X_train, y_train, validation_split=0.2, epochs=3,
batch_size=32, verbose=1)
```

```
Epoch 1/3
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.8606 - loss: 0.4677 - val_ac
Epoch 2/3
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9643 - loss: 0.1182 - val_a
Epoch 3/3
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9790 - loss: 0.0698 - val_a
```

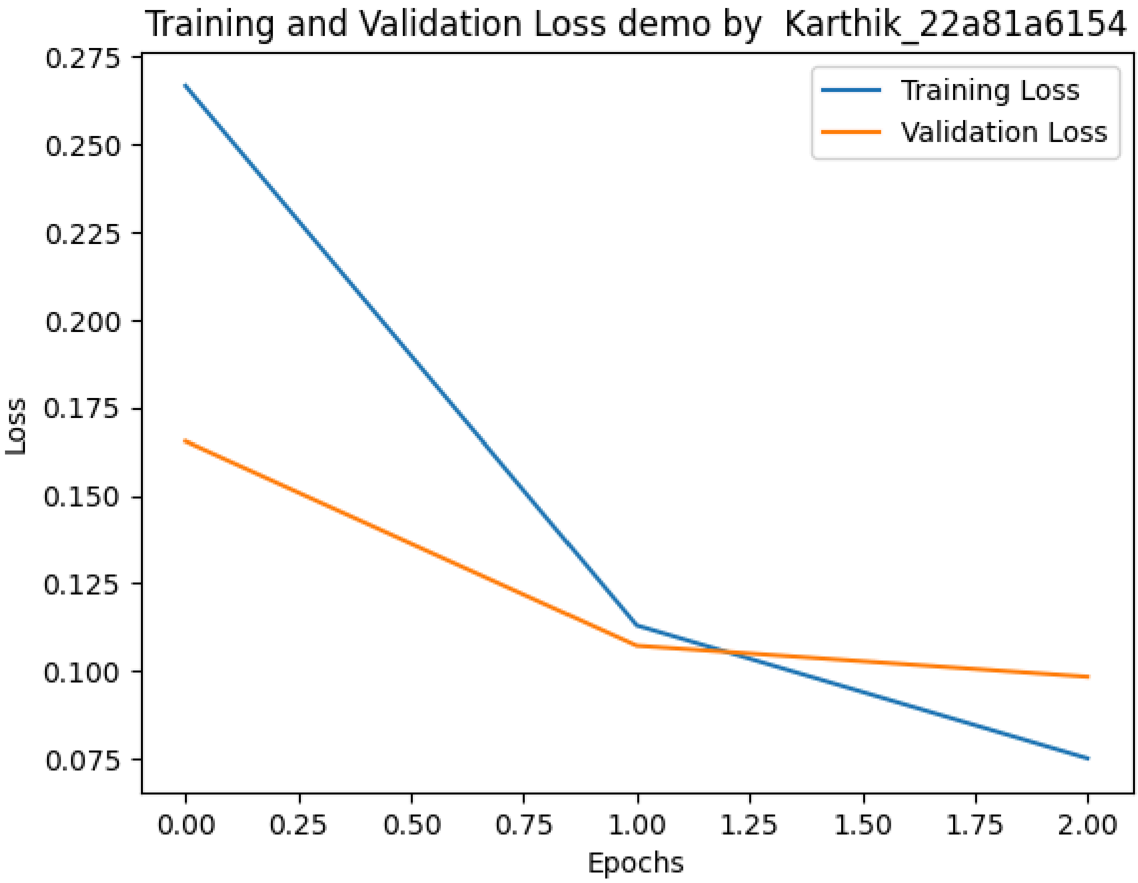
```
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"\nTest Loss: {test_loss}")      # Loss on the test set (categorical cross
print(f"Test Accuracy: {test_accuracy}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9676 - loss: 0.1143

Test Loss: 0.09752856940031052
Test Accuracy: 0.9714999794960022
```

```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy demo by  Karthik_22a81a6154')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss demo by  Karthik_22a81a6154')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



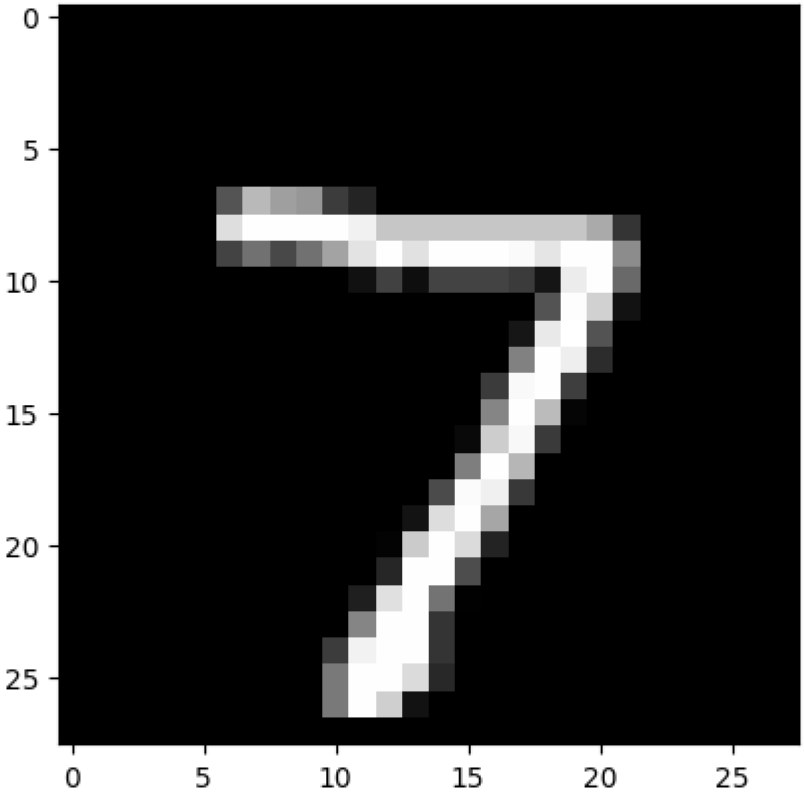

```
predictions = model.predict(X_test[:10])
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step
```

```
for i in range(10):
    actual_label = tf.argmax(y_test[i]).numpy()
    predicted_label = tf.argmax(predictions[i]).numpy()
    print(f"Actual Label: {actual_label}, Predicted Label: {predicted_label}")
    plt.imshow(X_test[i].reshape(28, 28), cmap='gray')
    plt.title(f"Actual: {actual_label}, Predicted: {predicted_label}  demo by  K
    plt.show()
```

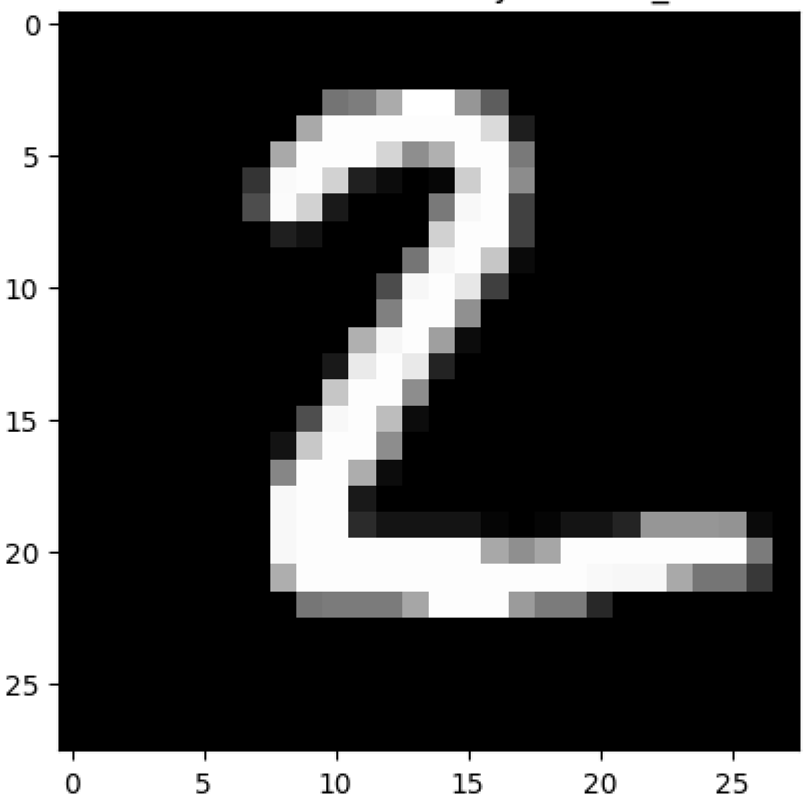
```
Actual Label: 7, Predicted Label: 7
```

Actual: 7, Predicted: 7 demo by Karthik_22a81a6154



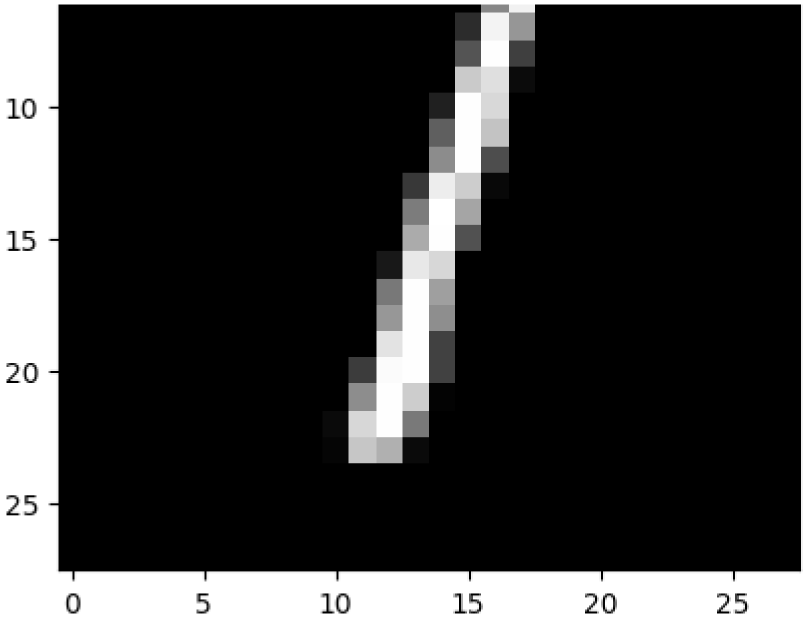
```
Actual Label: 2, Predicted Label: 2
```

Actual: 2, Predicted: 2 demo by Karthik_22a81a6154



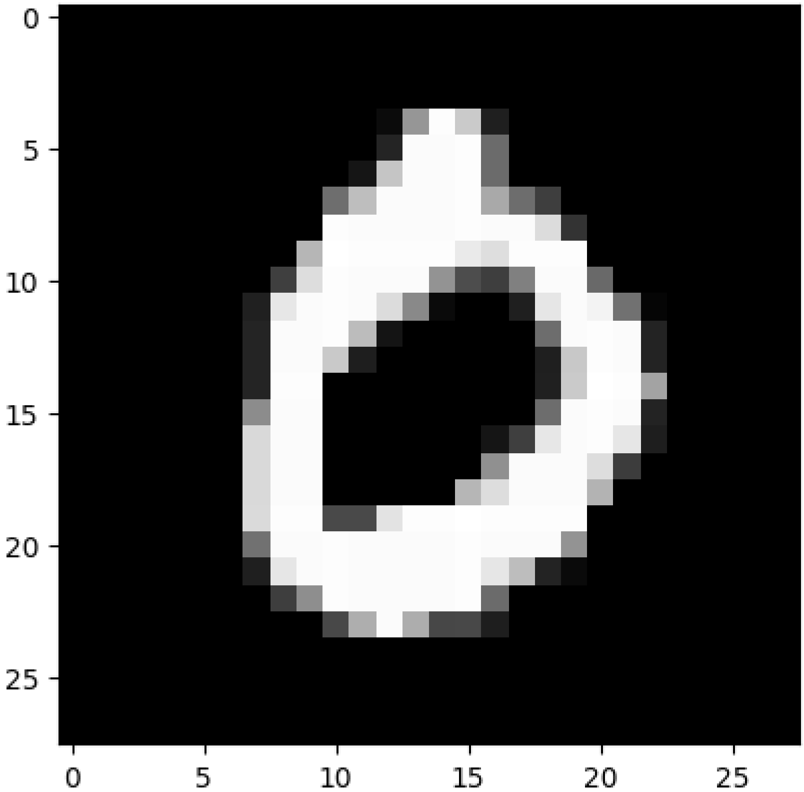
```
Actual Label: 1, Predicted Label: 1
```

Actual: 1, Predicted: 1 demo by Karthik_22a81a6154



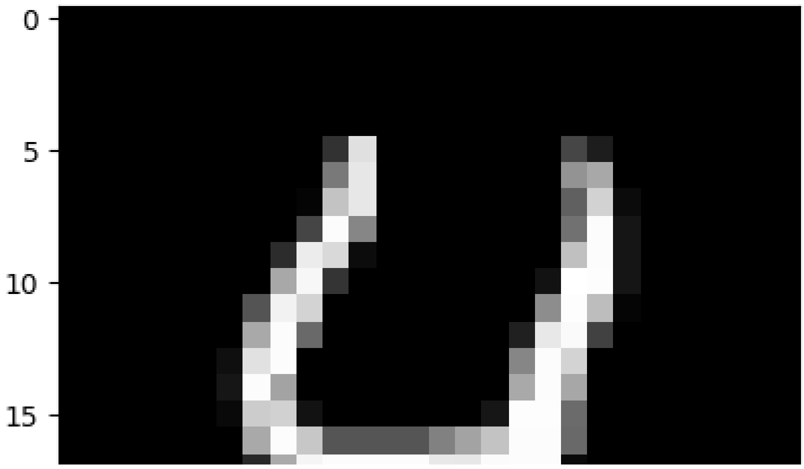

```
Actual Label: 0, Predicted Label: 0
```

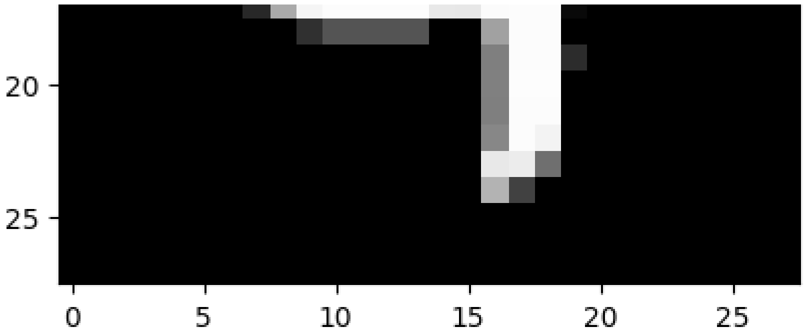
Actual: 0, Predicted: 0 demo by Karthik_22a81a6154



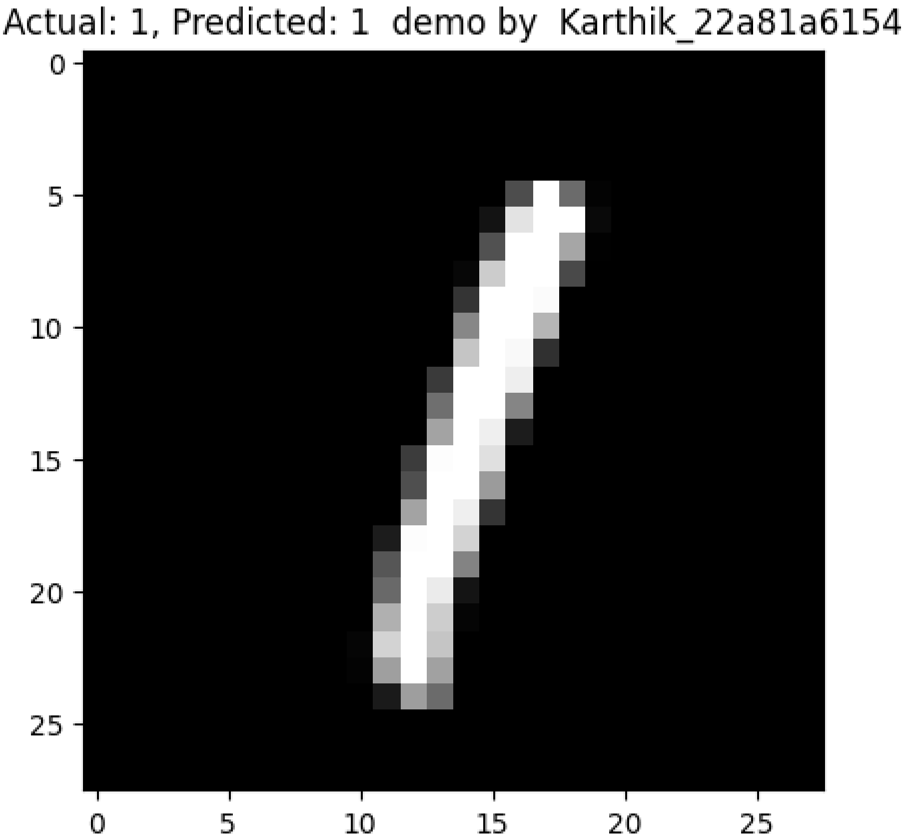

```
Actual Label: 4, Predicted Label: 4
```

Actual: 4, Predicted: 4 demo by Karthik_22a81a6154

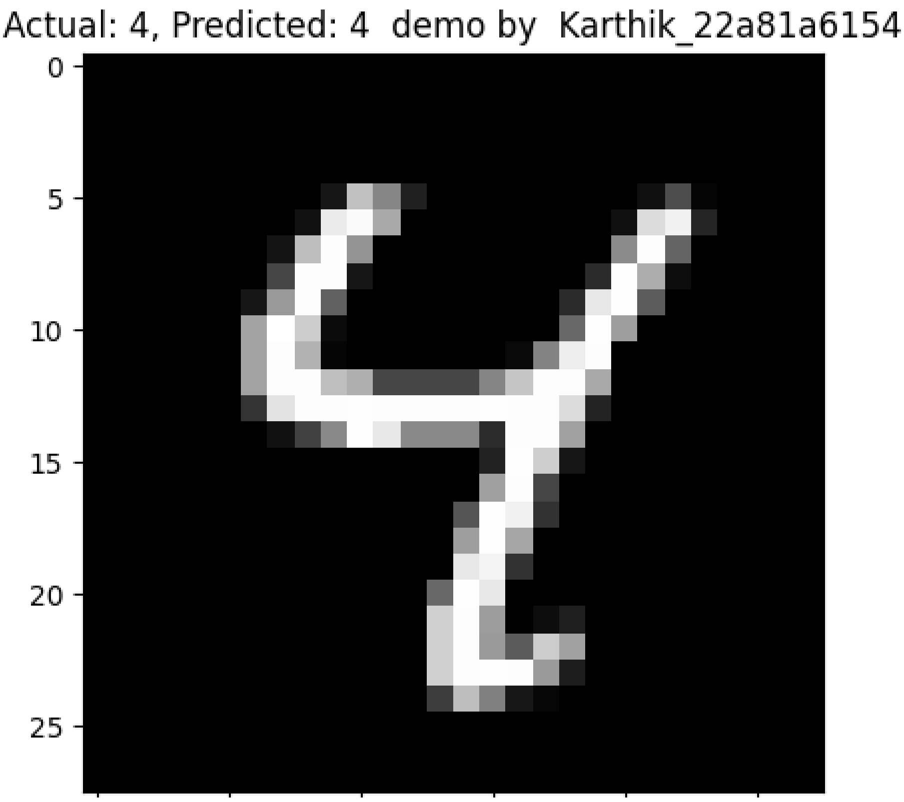





```
Actual Label: 1, Predicted Label: 1
```

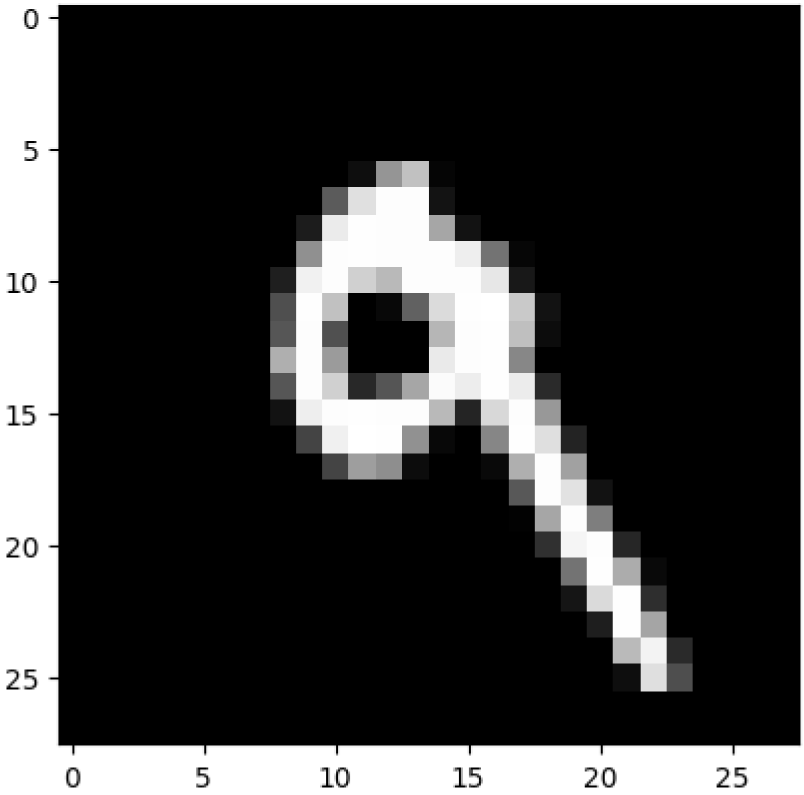



```
Actual Label: 4, Predicted Label: 4
```



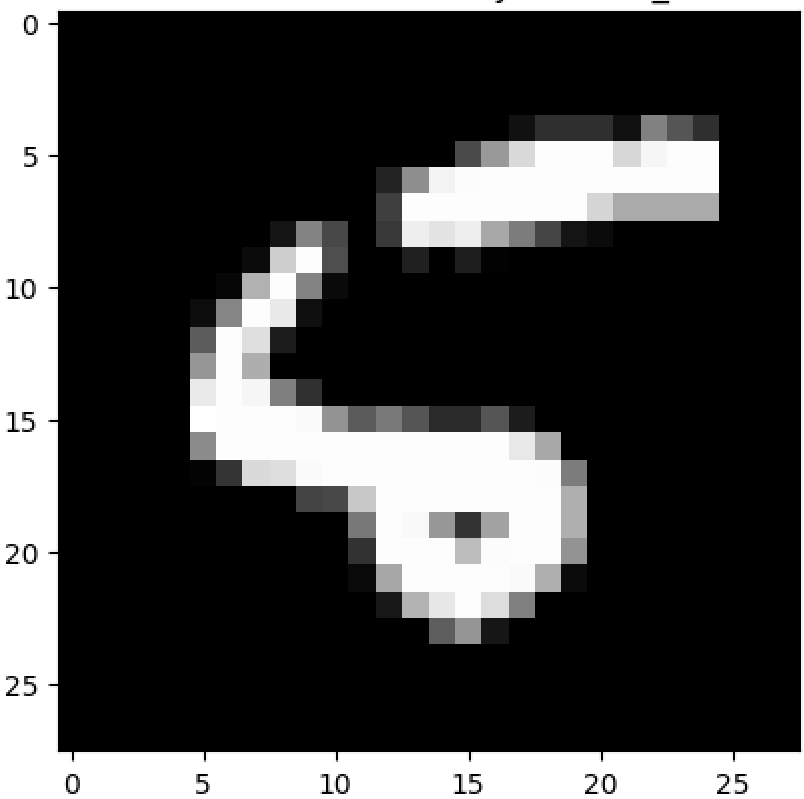
Actual Label: 9, Predicted Label: 9

Actual: 9, Predicted: 9 demo by Karthik_22a81a6154



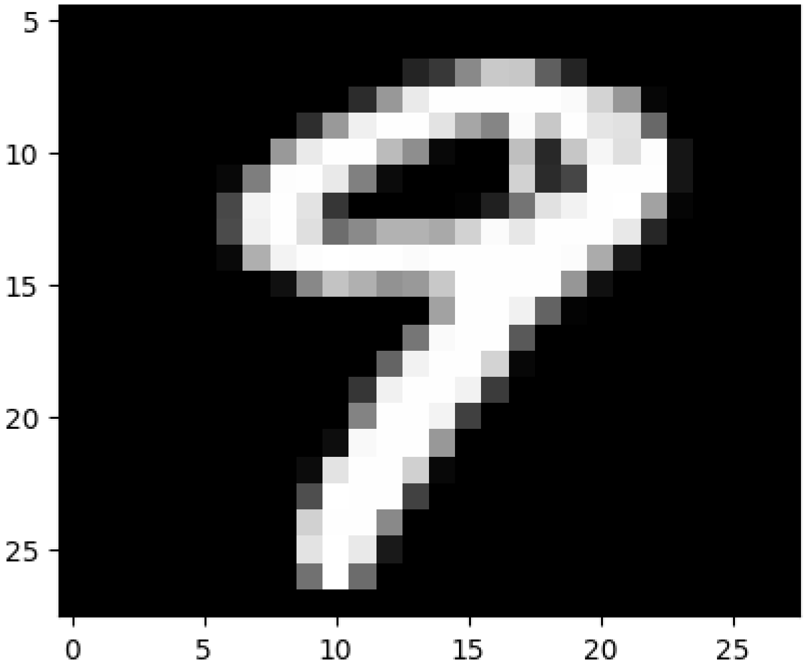
Actual Label: 5, Predicted Label: 2

Actual: 5, Predicted: 2 demo by Karthik_22a81a6154



Actual Label: 9, Predicted Label: 9

Actual: 9, Predicted: 9 demo by Karthik_22a81a6154


5
10
15
20
25
0
5
10
15
20
25

EXP-5

Develop a convolutional neural network on the Fashion-MNIST dataset

The Fashion-MNIST dataset is a benchmark dataset for image classification, similar to the original MNIST dataset but with more complex patterns. It contains grayscale images of 10 different clothing categories, making it an excellent test case for Convolutional Neural Networks (CNNs).

Dataset Overview

- Total Images: 70,000 (60,000 for training, 10,000 for testing)
- Image Size: 28×28 pixels (grayscale)
- Number of Classes: 10 (T-shirts, trousers, pullovers, dresses, coats,sandals, shirts, sneakers, bags, and ankle boots)
- Each Pixel Value: Ranges from 0 to 255 (grayscale intensity)

CNN

A Convolutional Neural Network (CNN) is a type of deep learning model designed for processing and analyzing visual data such as images and videos. It is highly effective in tasks like image classification, object detection, and facial recognition because it can automatically learn ar
extract important features from images.

Why Use CNNs for Fashion-MNIST?

Unlike fully connected networks, CNNs can effectively capture spatial features such as edges, textures, and patterns in clothing images. CNNs use convolutional layers, pooling layers, and fully connected layers to extract and classify features.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.datasets import fashion_mnist
import matplotlib.pyplot as plt
print("Demo by Karthik_22a81a6154")
```

```
Demo by Karthik_22a81a6154
```

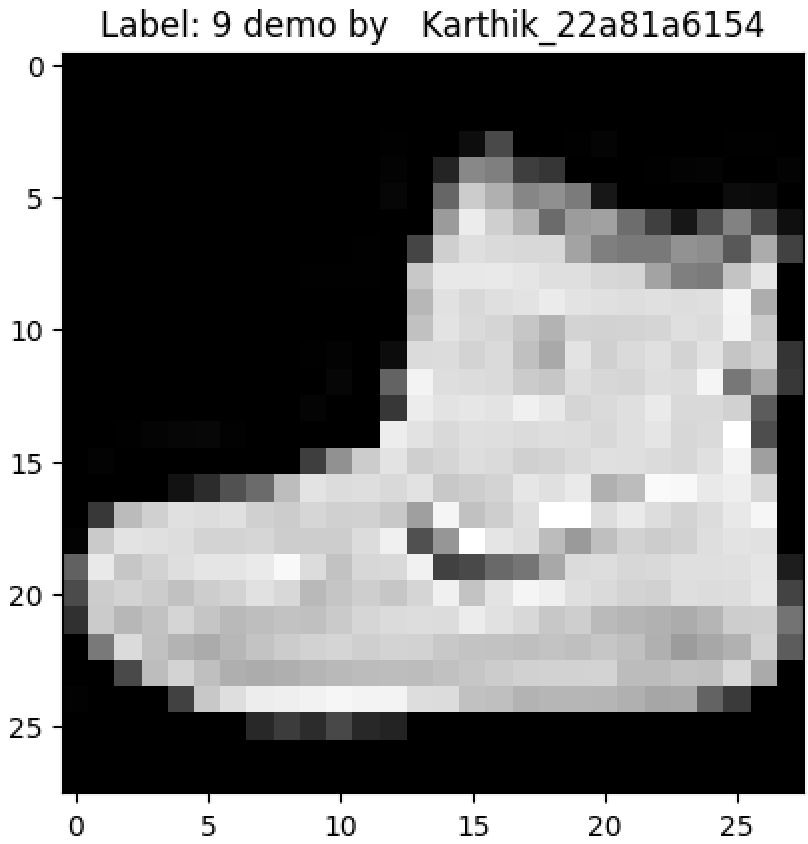
```
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/tra
29515/29515 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/tra
26421880/26421880 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10
5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
print("Training data shape:", X_train.shape)
print("Testing data shape:", X_test.shape)
```

```
Training data shape: (60000, 28, 28)
Testing data shape: (10000, 28, 28)
```

```
plt.imshow(X_train[0], cmap='gray')
plt.title(f"Label: {y_train[0]} demo by   Karthik_22a81a6154")
plt.show()
```



Delete cell Ctrl+M D

```
X_train = X_train
X_test = X_test
```

```
X_train = X_train.reshape(-1, 28, 28, 1)
X_test = X_test.reshape(-1, 28, 28, 1)
```

```python
model = Sequential([
    Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(28, 28, 1)),
MaxPooling2D(pool_size=(2,2)),
Conv2D(64, kernel_size=(3,3), activation='relu'),
MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:1
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

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

```python
history = model.fit(X_train, y_train, validation_split=0.2, epochs=10,
batch_size=32, verbose=1)
```

```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 52s 34ms/step - accuracy: 0.6472 - loss: 1.6770 - val_
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 78s 31ms/step - accuracy: 0.8199 - loss: 0.4919 - val_
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 45s 30ms/step - accuracy: 0.8496 - loss: 0.4162
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 48s 32ms/step - accuracy: 0.8648 - loss: 0.3753
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 82s 32ms/step - accuracy: 0.8723 - loss: 0.3482 - val_
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 82s 32ms/step - accuracy: 0.8781 - loss: 0.3337 - val_
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 49s 33ms/step - accuracy: 0.8837 - loss: 0.3159 - val_
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 80s 32ms/step - accuracy: 0.8863 - loss: 0.3113 - val_
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 77s 29ms/step - accuracy: 0.8907 - loss: 0.2997 - val_
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 84s 30ms/step - accuracy: 0.8965 - loss: 0.2818 - val_
```

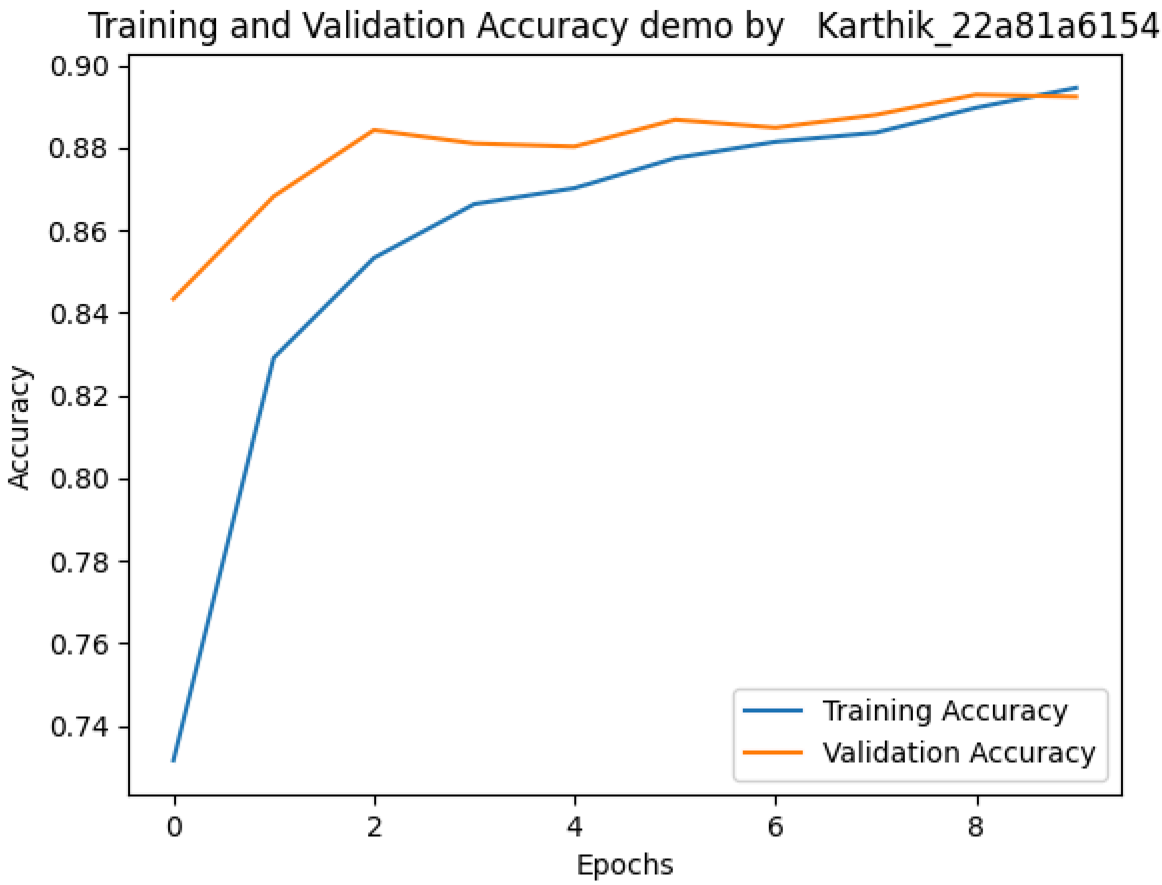
Delete cell Ctrl+M D

```python
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"\nTest Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.8831 - loss: 0.3261

Test Loss: 0.31722429394721985
Test Accuracy: 0.8859000205993652
```

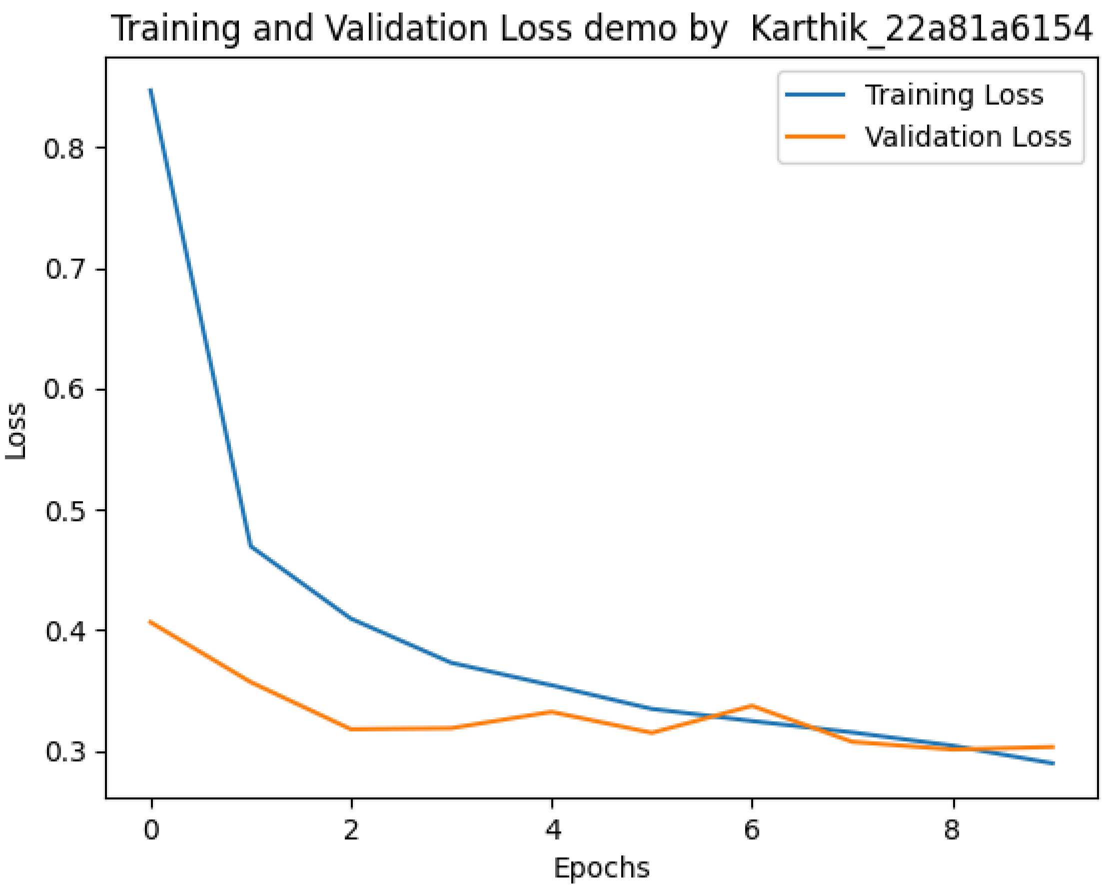
```python
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
```

```
plt.title('Training and Validation Accuracy demo by   Karthik_22a81a6154')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Delete cell Ctrl+M D

```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss demo by  Karthik_22a81a6154')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



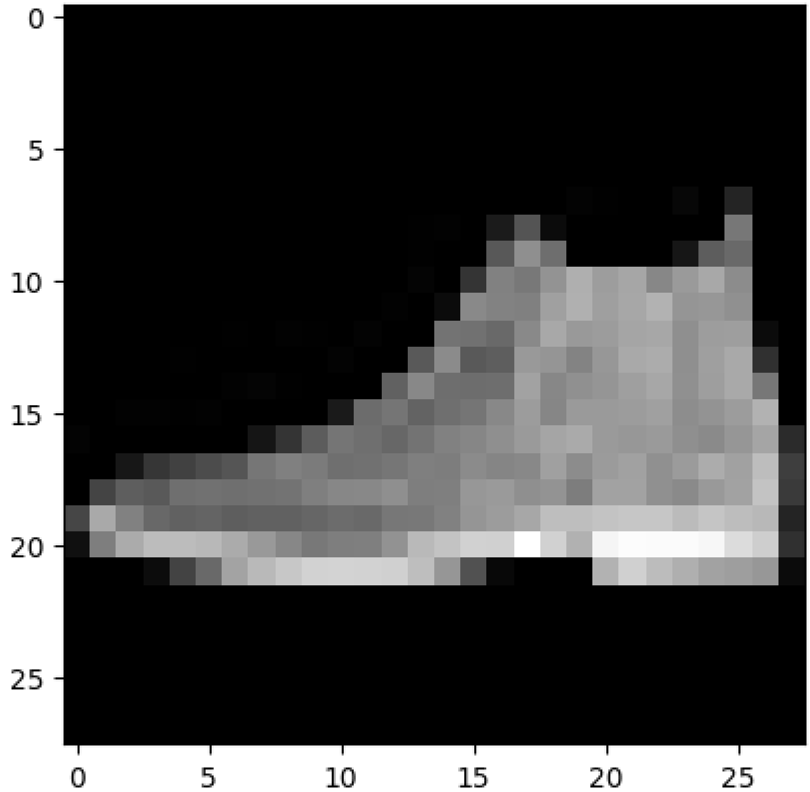

```
y_pred = model.predict(X_test[:10])
```

Delete cell Ctrl+M D

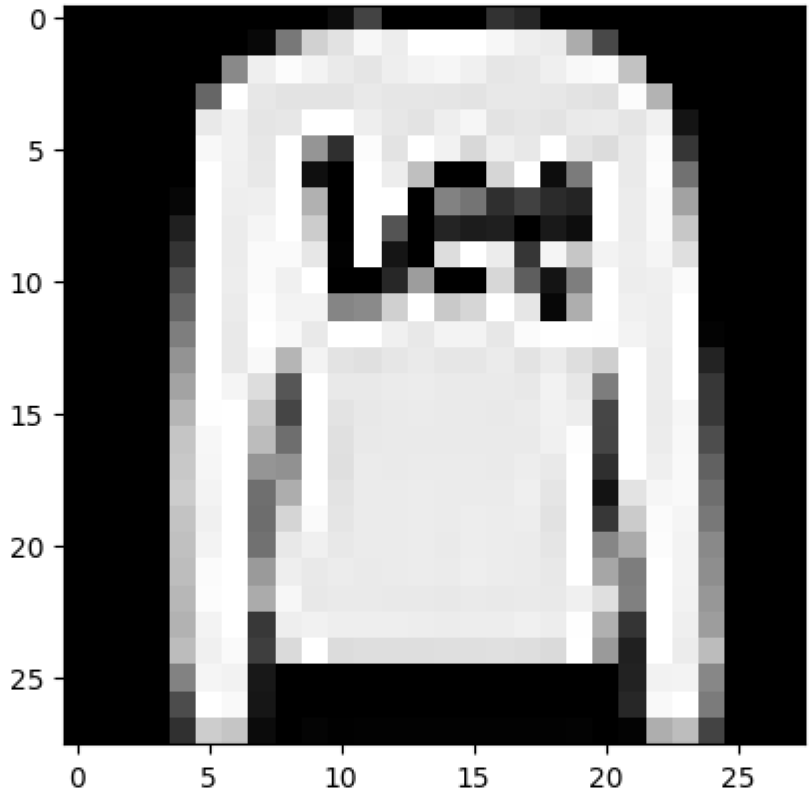
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 99ms/step
```

```
for i in range(10):
  actual_label = y_test[i]
  predicted_label = tf.argmax(y_pred[i]).numpy()
  plt.imshow(X_test[i].reshape(28, 28), cmap='gray')
  plt.title(f"Actual: {actual_label}, Predicted: {predicted_label} demo by   Karthik_22a8
  plt.show()
```

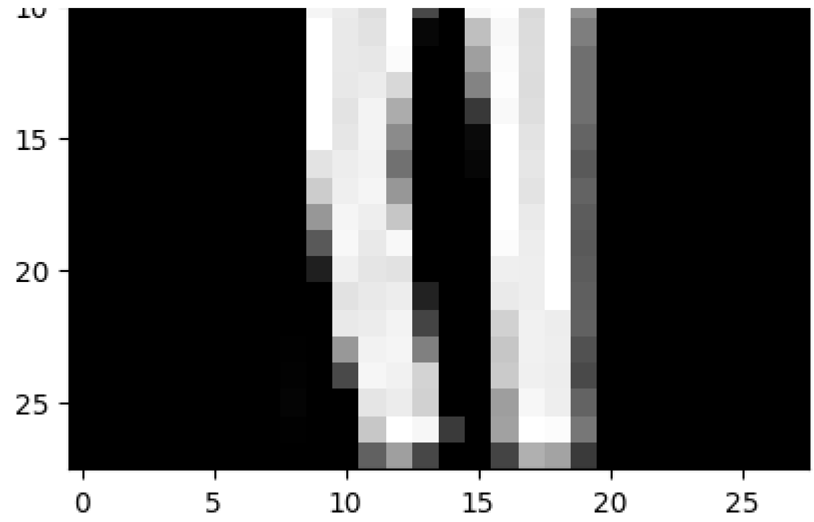
Actual: 9, Predicted: 9 demo by   Karthik_22a81a6154



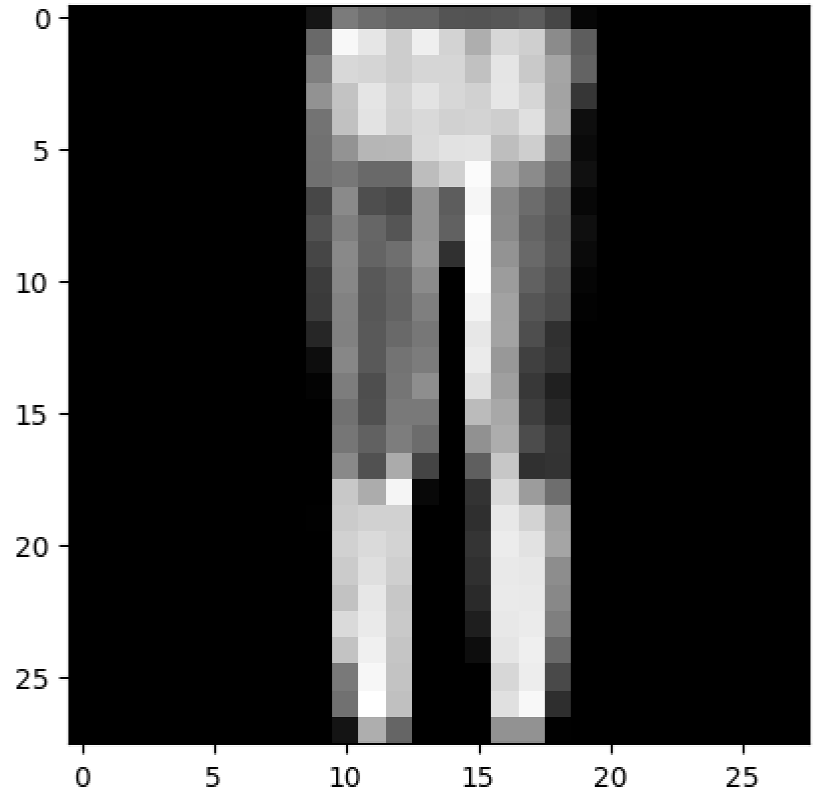
Actual: 2, Predicted: 2 demo by   Karthik_22a81a6154



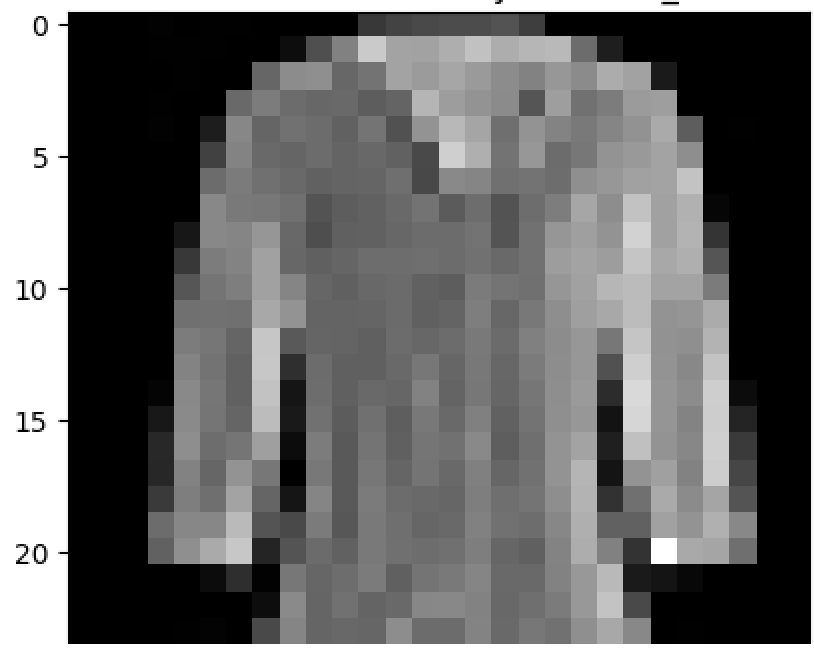
Actual: 1, Predicted: 1 demo by   Karthik_22a81a6154



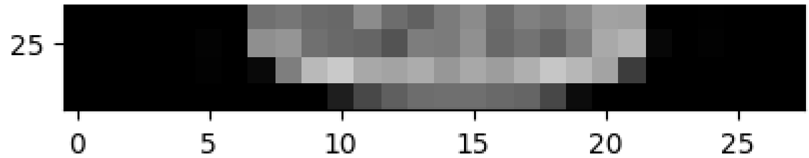
Actual: 1, Predicted: 1 demo by Karthik_22a81a6154



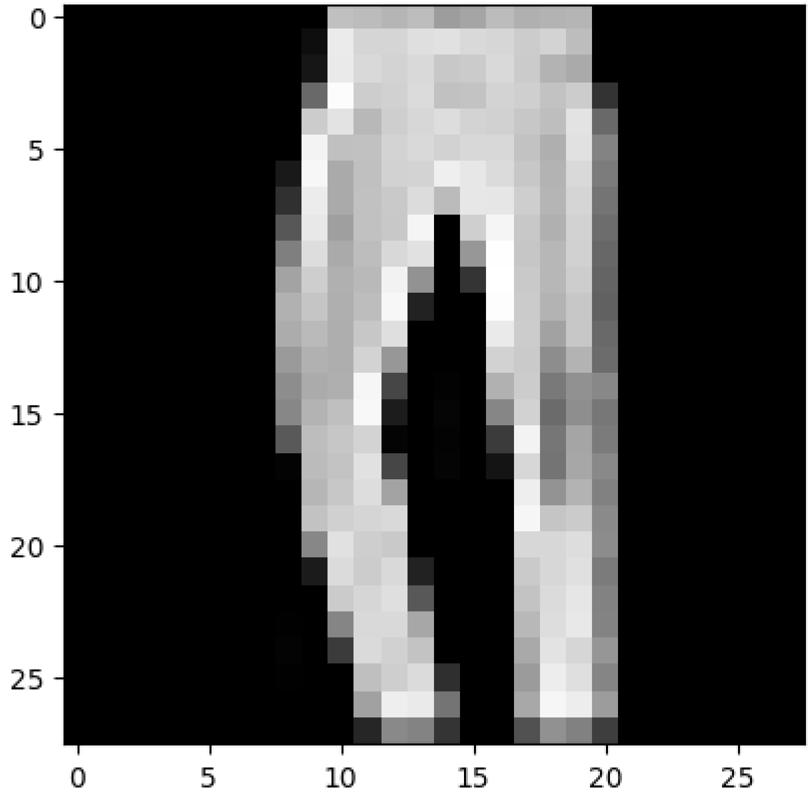
Actual: 6, Predicted: 6 demo by Karthik_22a81a6154



Delete cell Ctrl+M D

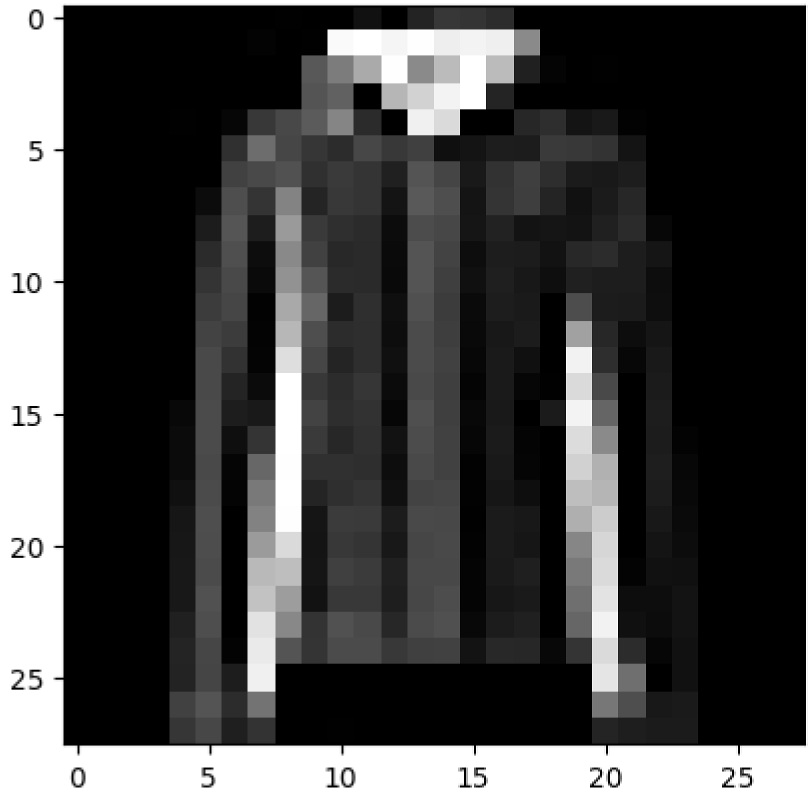


Actual: 1, Predicted: 1 demo by   Karthik_22a81a6154

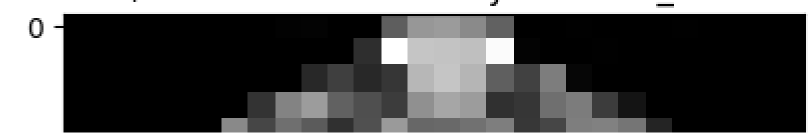


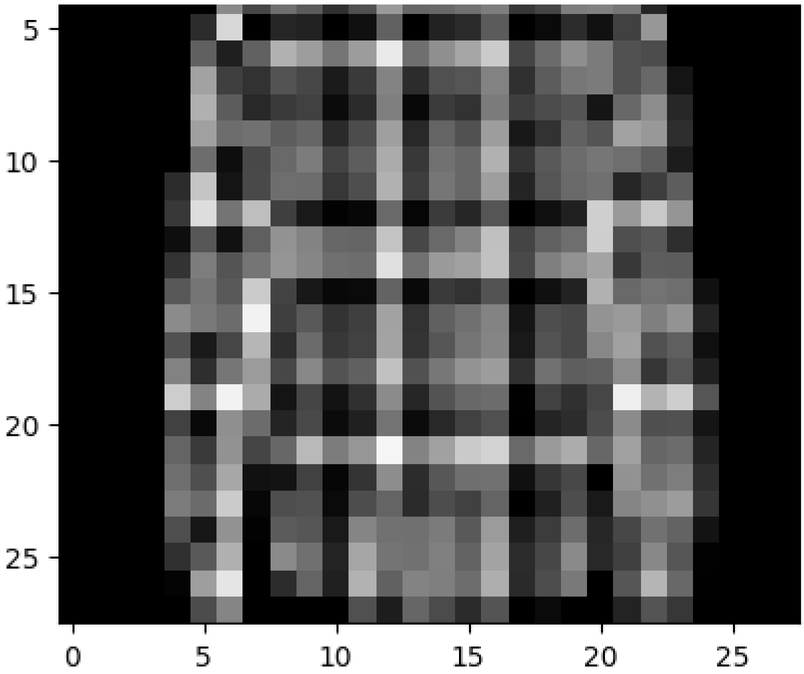
Delete cell Ctrl+M D

Actual: 4, Predicted: 4 demo by   Karthik_22a81a6154

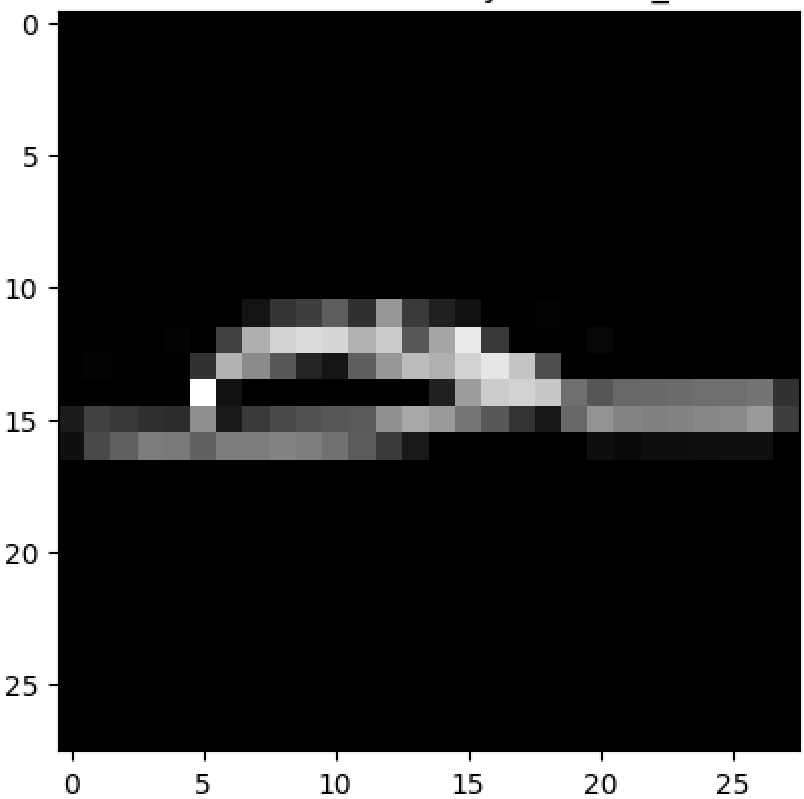


Actual: 6, Predicted: 6 demo by   Karthik_22a81a6154

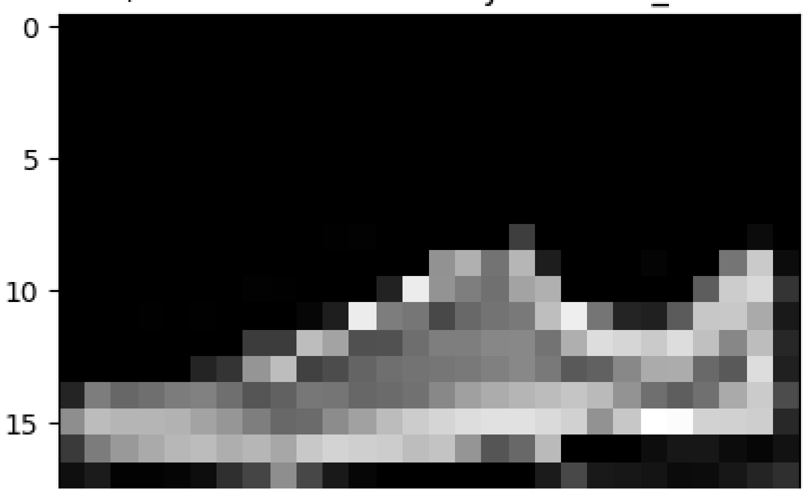


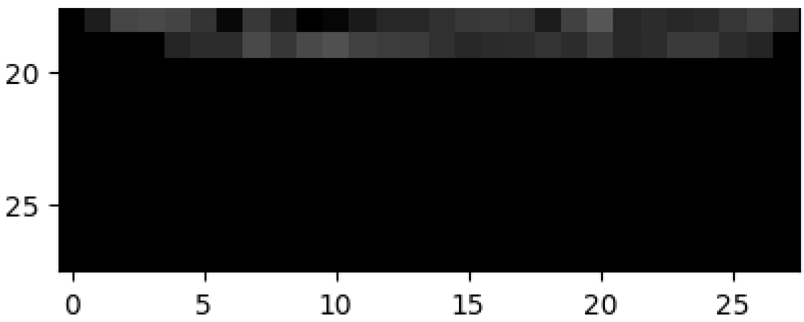


Actual: 5, Predicted: 5 demo by Karthik_22a81a6154



Actual: 7, Predicted: 7 demo by Karthik_22a81a6154





Delete
cell
Ctrl+M
D

Exp 6: Develop and train the VGG-16 network to classify images of Cats & Dogs.

VGG-16 is a deep convolutional neural network (CNN) that was introduced by the Visual Geometry Group (VGG) at the University of Oxford. It is widely used for image classification and won top ranks in the ImageNet competition. VGG-16 consists of 16 layers, including convolutional and fully connected layers, making it a powerful model for image recognition tasks.

Dataset Overview

A typical Cats vs. Dogs dataset contains:

- Training Images: Thousands of images labeled as either cat or dog.
- Testing Images: Unseen images used to evaluate the model's accuracy.
- Image Size: Typically 224×224 pixels, as required by VGG-16.
- Classes: Binary classification (0 = Cat, 1 = Dog).

```
import tensorflow as tf
import cv2
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.utils import get_file
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import os
import zipfile
google_drive_path = "/content/drive/MyDrive/dl/cats_and_dogs_filtered.zip"
extract_path = "/content/cats_and_dogs_filtered"
if not os.path.exists(google_drive_path):
    print("❌ Dataset file not found! Check the path in Google Drive.")
else:
    print("✅ Dataset found in Google Drive!")
```

```
✅ Dataset found in Google Drive!
```

```
if not os.path.exists(extract_path):
    print("🔄 Extracting dataset... Please wait.")
    with zipfile.ZipFile(google_drive_path, 'r') as zip_ref:
        zip_ref.extractall("/content")
    print("✅ Dataset extracted successfully!")
else:
    print("📁 Dataset already extracted.")
```

```
🔄 Extracting dataset... Please wait.
✅ Dataset extracted successfully!
```

```
train_dir = os.path.join(extract_path, 'train')
validation_dir = os.path.join(extract_path, 'validation')

if not os.path.exists(train_dir) or not os.path.exists(validation_dir):
    print("❌ Training or validation directories are missing!")
else:
    print("✅ Training and validation directories exist.")
    print("📁 Training folder contents:", os.listdir(train_dir))
    print("📁 Validation folder contents:", os.listdir(validation_dir))
```

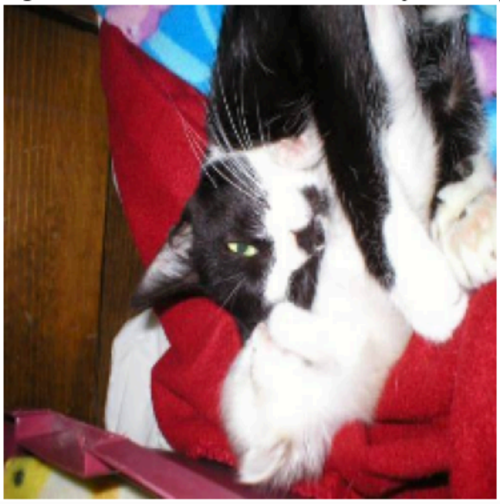
```
✅ Training and validation directories exist.
📁 Training folder contents: ['cats', 'dogs']
📁 Validation folder contents: ['cats', 'dogs']
```

```
import cv2
import matplotlib.pyplot as plt
sample_image_path = os.path.join(train_dir, 'cats', os.listdir(os.path.join(train_dir, 'cats'))[0])
img = cv2.imread(sample_image_path)
if img is None:
    print("❌ Image not loaded! Check the file path.")
else:
    print("✅ Image loaded successfully!")
    print("📏 Image Shape:", img.shape)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_resized = cv2.resize(img_rgb, (224, 224))
    plt.imshow(img_resized)
```

```
    plt.axis("off")
    plt.title("Sample Cat Image (Resized to 224x224) - Demo by Karthik_22A81A6154")
    plt.show()
```

```
✅ Image loaded successfully!
📏 Image Shape: (374, 500, 3)
```

Sample Cat Image (Resized to 224x224) - Demo by Karthik_22A81A6154



```
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1.0/255.0,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
validation_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1.0/255.0)
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary'
)
validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary'
)
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

```
base_model = tf.keras.applications.VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False
base_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_no
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0

```
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14,714,688
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 512)	12,845,568
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 1)	513

Total params: 27,560,769 (105.14 MB)

```
history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=10,
    verbose=1
)
```

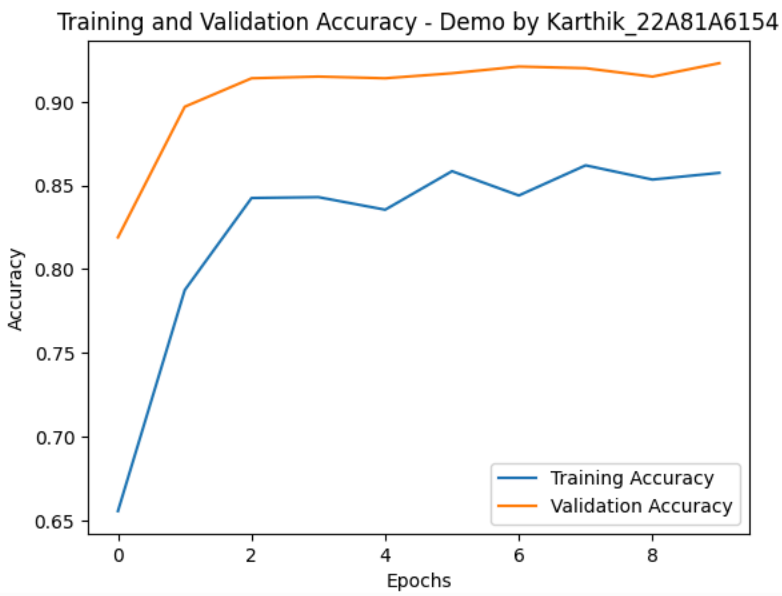
```
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` cl
  self._warn_if_super_not_called()
Epoch 1/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 61s 766ms/step - accuracy: 0.5711 - loss: 2.4510 - val_accuracy: 0.8190 - val_loss: 0.3803
Epoch 2/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 56s 532ms/step - accuracy: 0.7578 - loss: 0.4846 - val_accuracy: 0.8970 - val_loss: 0.2674
Epoch 3/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 33s 531ms/step - accuracy: 0.8375 - loss: 0.3688 - val_accuracy: 0.9140 - val_loss: 0.2344
```

```
Epoch 4/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 33s 530ms/step - accuracy: 0.8442 - loss: 0.3706 - val_accuracy: 0.9150 - val_loss: 0.2287
Epoch 5/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 33s 525ms/step - accuracy: 0.8572 - loss: 0.3479 - val_accuracy: 0.9140 - val_loss: 0.2248
Epoch 6/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 33s 525ms/step - accuracy: 0.8456 - loss: 0.3504 - val_accuracy: 0.9170 - val_loss: 0.2088
Epoch 7/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 42s 533ms/step - accuracy: 0.8456 - loss: 0.3352 - val_accuracy: 0.9210 - val_loss: 0.2040
Epoch 8/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 33s 531ms/step - accuracy: 0.8615 - loss: 0.3309 - val_accuracy: 0.9200 - val_loss: 0.2184
Epoch 9/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 605ms/step - accuracy: 0.8600 - loss: 0.3165 - val_accuracy: 0.9150 - val_loss: 0.2160
Epoch 10/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 33s 526ms/step - accuracy: 0.8677 - loss: 0.3237 - val_accuracy: 0.9230 - val_loss: 0.2020
```

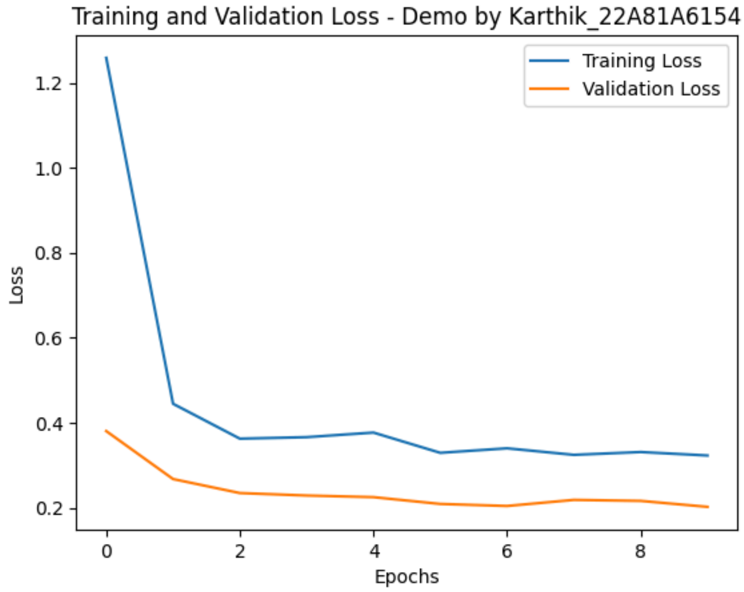
```
test_loss, test_acc = model.evaluate(validation_generator)
print(f"\n✅ Model Test Accuracy: {test_acc:.2f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 161ms/step - accuracy: 0.9232 - loss: 0.2092

✅ Model Test Accuracy: 0.92
```

```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy - Demo by Karthik_22A81A6154')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss - Demo by Karthik_22A81A6154')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```


Training and Validation Loss - Demo by Karthik_22A81A6154
Training Loss
Validation Loss
Loss
Epochs
1.2
1.0
0.8
0.6
0.4
0.2
0
2
4
6
8

Experiment 7: Develop a Neural Network with an Embedding Layer for Text Classication

It involves:

1. Converting words into numerical embeddings using the Embedding layer.
2. Training a neural network to classify text into categories.
3. Applying the model to classify sentiment (positive/negative) or topic-based text classication.

```
print("Demo by Karthik_22a81a6154")
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Embedding, Dense, GlobalAveragePooling1D
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
import matplotlib.pyplot as plt
```

```
Demo by Karthik_22a81a6154
```

```
imdb = keras.datasets.imdb
```

```
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

```
print("Sample Review (Tokenized):", train_data[0])
print("Label (0=Negative, 1=Positive):", train_labels[0])
```

```
Sample Review (Tokenized): [1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 6
Label (0=Negative, 1=Positive): 1
```

```
word_index = imdb.get_word_index()
reverse_word_index = {value: key for (key, value) in word_index.items()}
decoded_review = " ".join([reverse_word_index.get(i - 3, "?") for i in train_data[0]])
print("\nSample Review (Decoded):")
print(decoded_review)
```

```

Sample Review (Decoded):
? this film was just brilliant casting location scenery story direction everyone's re
```

```
max_length = 256
train_data = pad_sequences(train_data, maxlen=max_length, padding='post', truncating='pos
test_data = pad_sequences(test_data, maxlen=max_length, padding='post', truncating='post'
print("Shape of Training Data:", train_data.shape)
print("Shape of Testing Data:", test_data.shape)
```

```
Shape of Training Data: (25000, 256)
Shape of Testing Data: (25000, 256)
```

```
model = keras.Sequential([
Embedding(10000, 16),
GlobalAveragePooling1D(),
Dense(16, activation='relu'),
Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam',
loss='binary_crossentropy',
metrics=['accuracy'])
history=model.fit(train_data,train_labels,epochs=20,batch_size=512,validation_da
model.summary()
```

```
Epoch 1/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 4s 54ms/step - accuracy: 0.5463 - loss: 0.6911 - val_ac
Epoch 2/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 4s 25ms/step - accuracy: 0.6715 - loss: 0.6679 - val_ac
Epoch 3/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.7500 - loss: 0.6131 - val_ac
Epoch 4/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.8125 - loss: 0.5333 - val_ac
Epoch 5/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.8337 - loss: 0.4603 - val_ac
Epoch 6/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 33ms/step - accuracy: 0.8569 - loss: 0.4021 - val_ac
Epoch 7/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - accuracy: 0.8628 - loss: 0.3638 - val_ac
Epoch 8/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.8778 - loss: 0.3323 - val_ac
Epoch 9/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.8856 - loss: 0.3069 - val_ac
Epoch 10/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - accuracy: 0.8901 - loss: 0.2914 - val_ac
Epoch 11/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.8959 - loss: 0.2760 - val_ac
Epoch 12/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.8976 - loss: 0.2719 - val_ac
Epoch 13/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9035 - loss: 0.2515 - val_ac
Epoch 14/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 39ms/step - accuracy: 0.9103 - loss: 0.2430 - val_ac
Epoch 15/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.9162 - loss: 0.2314 - val_ac
Epoch 16/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - accuracy: 0.9175 - loss: 0.2284 - val_ac
Epoch 17/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.9193 - loss: 0.2186 - val_ac
Epoch 18/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - accuracy: 0.9175 - loss: 0.2166 - val_ac
Epoch 19/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - accuracy: 0.9226 - loss: 0.2102 - val_ac
Epoch 20/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9242 - loss: 0.2050 - val_ac
```

Model: "sequential_2"

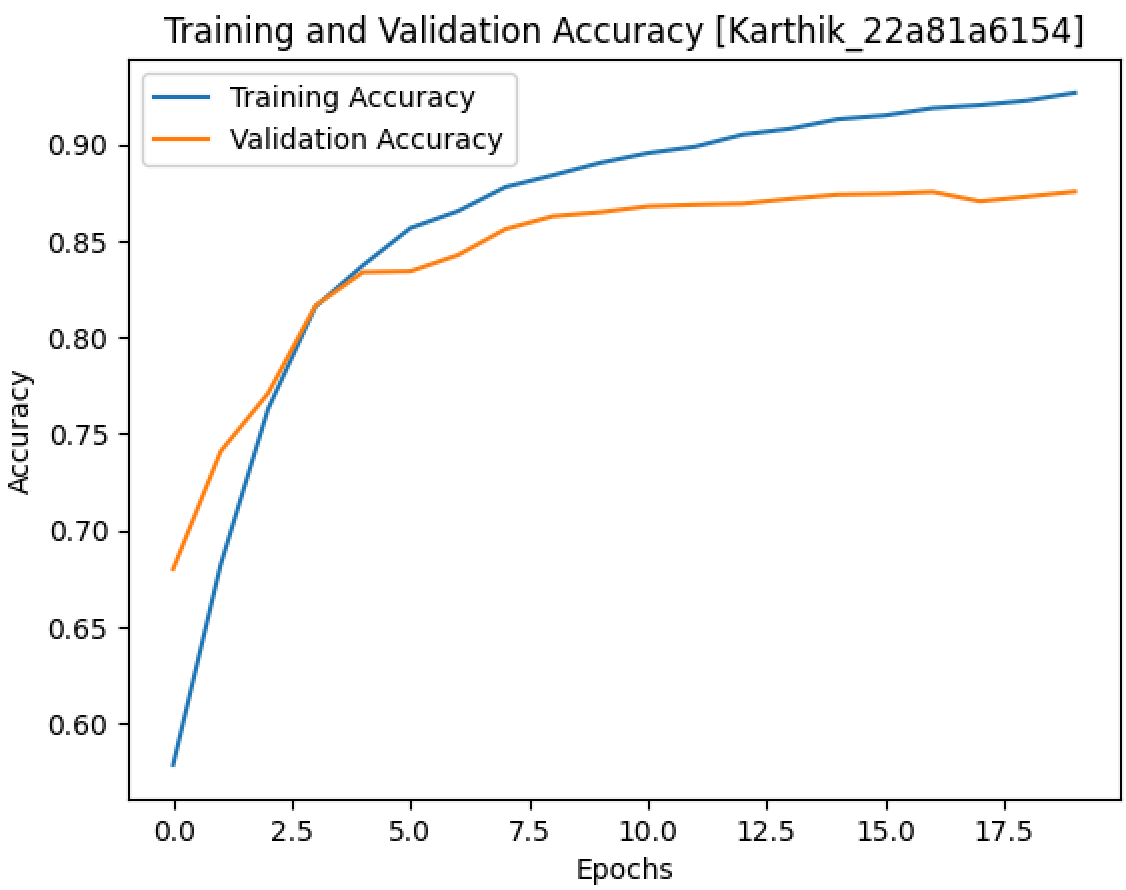
Layer (type)	Output Shape	Par
embedding_2 (Embedding)	(None, 256, 16)	160
global_average_pooling1d_2 (GlobalAveragePooling1D)	(None, 16)	
dense_4 (Dense)	(None, 16)	
dense_5 (Dense)	(None, 1)	

Total params: 480,869 (1.83 MB)
Trainable params: 160,289 (626.13 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 320,580 (1.22 MB)

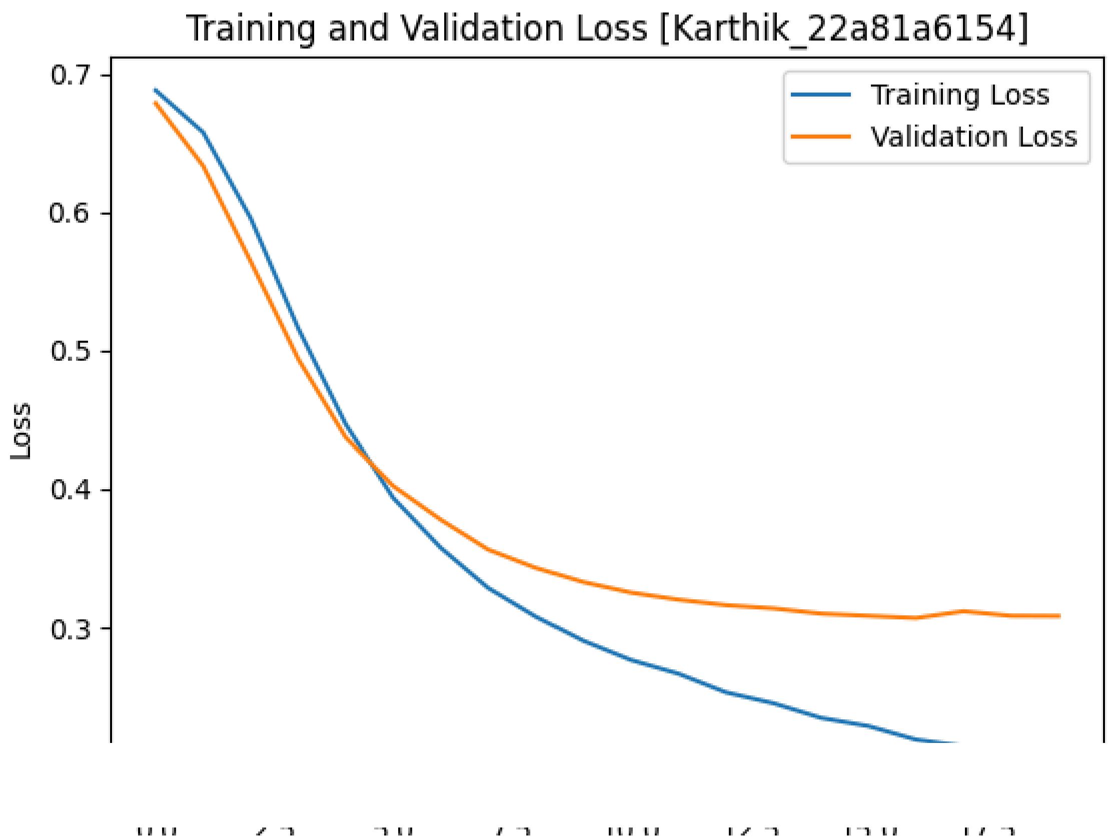
```
test_loss, test_acc = model.evaluate(test_data, test_labels)
print(f"\nTest Accuracy: {test_acc:.4f}")
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.8756 - loss: 0.3084

Test Accuracy: 0.8754
```

```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy [Karthik_22a81a6154]')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss [Karthik_22a81a6154]')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Training and Validation Loss [Karthik_22a81a6154]
0.7
0.6
0.5
0.4
0.3
Loss
Training Loss
Validation Loss