Shree Ganeshay Namah

Assignment 2

Name - Anurag Gupta

Rollno - MPIITDCV53525

1. download Mnist fashion dataset, and then create a datset of using any two fashion accessories and then classify the same. try to optimise the same using any Mean square error or R2 error.
2. Use digit dataset to classify the digits 8 and 3 or 6 and 9.

```
import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist
from sklearn import linear_model
import numpy as np
import matplotlib.pyplot as plt

# Load the Fashion MNIST dataset
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

# Normalize the images
train_images = train_images / 255.0
test_images = test_images / 255.0


class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
               "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()

```


```
# Select only "Bag" and "Ankle boot"
selected_classes = [8, 9]

# Filter the dataset
train_filter = np.isin(train_labels, selected_classes)
test_filter = np.isin(test_labels, selected_classes)

train_images, train_labels = train_images[train_filter], train_labels[train_filter]
test_images, test_labels = test_images[test_filter], test_labels[test_filter]

# Convert labels to binary (0 and 1)
train_labels = np.where(train_labels == 8, 0, 1)
test_labels = np.where(test_labels == 8, 0, 1)

```

```
Start coding or generate with AI.
```

```
array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout

# Build the model
model = Sequential([

    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

# Compile the model using Mean Squared Error (MSE)
model.compile(optimizer='adam',
              loss='mean_squared_error',
              metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))

```

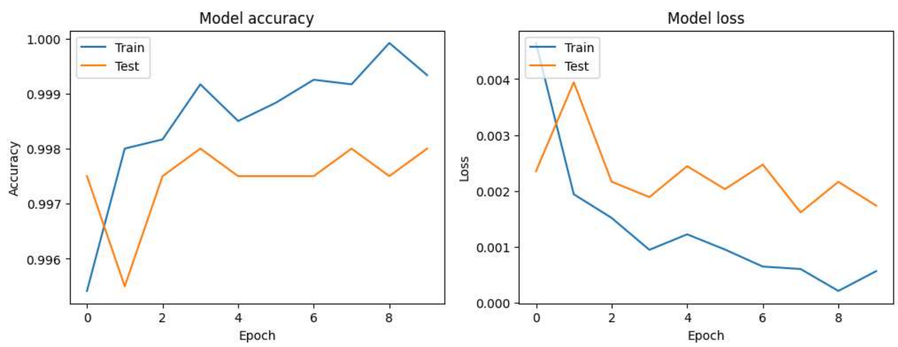
```
Epoch 1/10
375/375 [==============================] - 4s 5ms/step - loss: 0.0046 - accuracy: 0.9954 - val_loss: 0.0023 - val_accuracy: 0.9975
Epoch 2/10
375/375 [==============================] - 2s 4ms/step - loss: 0.0019 - accuracy: 0.9980 - val_loss: 0.0039 - val_accuracy: 0.9955
Epoch 3/10
375/375 [==============================] - 2s 4ms/step - loss: 0.0015 - accuracy: 0.9982 - val_loss: 0.0022 - val_accuracy: 0.9975
Epoch 4/10
375/375 [==============================] - 2s 4ms/step - loss: 9.4151e-04 - accuracy: 0.9992 - val_loss: 0.0019 - val_accuracy: 0.9980
Epoch 5/10
375/375 [==============================] - 2s 4ms/step - loss: 0.0012 - accuracy: 0.9985 - val_loss: 0.0024 - val_accuracy: 0.9975
Epoch 6/10
375/375 [==============================] - 2s 6ms/step - loss: 9.4643e-04 - accuracy: 0.9988 - val_loss: 0.0020 - val_accuracy: 0.9975
Epoch 7/10
375/375 [==============================] - 2s 6ms/step - loss: 6.4166e-04 - accuracy: 0.9992 - val_loss: 0.0025 - val_accuracy: 0.9975
Epoch 8/10
375/375 [==============================] - 1s 4ms/step - loss: 5.9827e-04 - accuracy: 0.9992 - val_loss: 0.0016 - val_accuracy: 0.9980
Epoch 9/10
375/375 [==============================] - 1s 4ms/step - loss: 2.0528e-04 - accuracy: 0.9999 - val_loss: 0.0022 - val_accuracy: 0.9975
Epoch 10/10
375/375 [==============================] - 2s 4ms/step - loss: 5.5933e-04 - accuracy: 0.9993 - val_loss: 0.0017 - val_accuracy: 0.9980
```

```python
# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

```




```python
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the digits dataset
digits = datasets.load_digits()

# Choose digits 8 and 3 or 6 and 9
selected_digits = [8, 3]  # or [6, 9]

# Filter the dataset
filter_digits = np.isin(digits.target, selected_digits)
images = digits.images[filter_digits]
labels = digits.target[filter_digits]

# Convert labels to binary (0 and 1)
labels = np.where(labels == selected_digits[0], 0, 1)

# Flatten images
n_samples = len(images)
images = images.reshape((n_samples, -1))

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)

# Standardize the dataset
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

```

```python
X_train.shape
```

```
(285, 64)
```

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# Build the model
model = Sequential([
    Dense(64, activation='relu', input_shape=(64,)),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

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

# Train the model
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test))
```

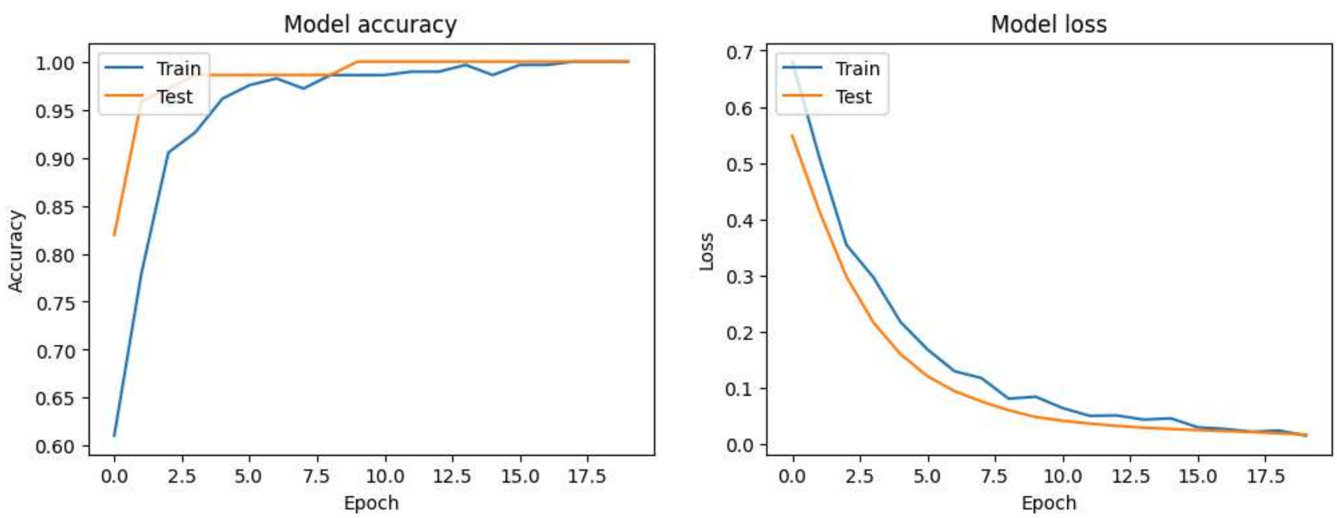
```
Epoch 1/20
9/9 [==============================] - 2s 43ms/step - loss: 0.6804 - accuracy: 0.6105 - val_loss: 0.5481 - val_accuracy: 0.8194
Epoch 2/20
9/9 [==============================] - 0s 11ms/step - loss: 0.5098 - accuracy: 0.7789 - val_loss: 0.4142 - val_accuracy: 0.9583
Epoch 3/20
9/9 [==============================] - 0s 17ms/step - loss: 0.3542 - accuracy: 0.9053 - val_loss: 0.2975 - val_accuracy: 0.9722
Epoch 4/20
9/9 [==============================] - 0s 23ms/step - loss: 0.2960 - accuracy: 0.9263 - val_loss: 0.2157 - val_accuracy: 0.9861
Epoch 5/20
9/9 [==============================] - 0s 11ms/step - loss: 0.2168 - accuracy: 0.9614 - val_loss: 0.1596 - val_accuracy: 0.9861
Epoch 6/20
9/9 [==============================] - 0s 16ms/step - loss: 0.1678 - accuracy: 0.9754 - val_loss: 0.1201 - val_accuracy: 0.9861
Epoch 7/20
9/9 [==============================] - 0s 12ms/step - loss: 0.1290 - accuracy: 0.9825 - val_loss: 0.0936 - val_accuracy: 0.9861
Epoch 8/20
9/9 [==============================] - 0s 13ms/step - loss: 0.1167 - accuracy: 0.9719 - val_loss: 0.0752 - val_accuracy: 0.9861
Epoch 9/20
9/9 [==============================] - 0s 12ms/step - loss: 0.0801 - accuracy: 0.9860 - val_loss: 0.0595 - val_accuracy: 0.9861
Epoch 10/20
9/9 [==============================] - 0s 11ms/step - loss: 0.0836 - accuracy: 0.9860 - val_loss: 0.0475 - val_accuracy: 1.0000
Epoch 11/20
9/9 [==============================] - 0s 8ms/step - loss: 0.0634 - accuracy: 0.9860 - val_loss: 0.0408 - val_accuracy: 1.0000
Epoch 12/20
9/9 [==============================] - 0s 6ms/step - loss: 0.0496 - accuracy: 0.9895 - val_loss: 0.0358 - val_accuracy: 1.0000
Epoch 13/20
9/9 [==============================] - 0s 6ms/step - loss: 0.0503 - accuracy: 0.9895 - val_loss: 0.0317 - val_accuracy: 1.0000
Epoch 14/20
9/9 [==============================] - 0s 8ms/step - loss: 0.0428 - accuracy: 0.9965 - val_loss: 0.0285 - val_accuracy: 1.0000
Epoch 15/20
9/9 [==============================] - 0s 8ms/step - loss: 0.0451 - accuracy: 0.9860 - val_loss: 0.0263 - val_accuracy: 1.0000
Epoch 16/20
9/9 [==============================] - 0s 8ms/step - loss: 0.0291 - accuracy: 0.9965 - val_loss: 0.0241 - val_accuracy: 1.0000
Epoch 17/20
9/9 [==============================] - 0s 8ms/step - loss: 0.0263 - accuracy: 0.9965 - val_loss: 0.0223 - val_accuracy: 1.0000
Epoch 18/20
9/9 [==============================] - 0s 8ms/step - loss: 0.0212 - accuracy: 1.0000 - val_loss: 0.0206 - val_accuracy: 1.0000
Epoch 19/20
9/9 [==============================] - 0s 6ms/step - loss: 0.0234 - accuracy: 1.0000 - val_loss: 0.0184 - val_accuracy: 1.0000
Epoch 20/20
9/9 [==============================] - 0s 8ms/step - loss: 0.0143 - accuracy: 1.0000 - val_loss: 0.0161 - val_accuracy: 1.0000
```

```
# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

```



Pratical Question 2

For Mnist Fashion dataset reduce the dimesionaity to say n, where n is accepted from user and try to classify the same using differnt classifiers like Decision trees, SVM, KNN etc

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score

# Load MNIST Fashion dataset
fashion_mnist = fetch_openml(name='Fashion-MNIST', version=1)
X = fashion_mnist.data
y = fashion_mnist.target

# Normalize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Accept the number of dimensions from the user
n = int(input("Enter the number of dimensions to reduce to: "))

# Reduce dimensionality using PCA
pca = PCA(n_components=n)
X_pca = pca.fit_transform(X_scaled)
```

```python

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42)

# Define classifiers
classifiers = {
    'Decision Tree': DecisionTreeClassifier(),
    'SVM': SVC(),
    'KNN': KNeighborsClassifier()
}

# Train and evaluate classifiers
for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(f"Classifier: {name}")
    print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
    print(classification_report(y_test, y_pred))
    print("\n")

# Optional: Visualize the first two PCA components
plt.figure(figsize=(8, 6))
for i in range(10):
    idx = np.where(y == str(i))
    plt.scatter(X_pca[idx, 0], X_pca[idx, 1], label=f"Class {i}", alpha=0.5)
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.title('PCA of Fashion MNIST')
plt.legend()
plt.show()

```

```
/usr/local/lib/python3.10/dist-packages/sklearn/datasets/_openml.py:968: FutureWarning:
  warn(
Enter the number of dimensions to reduce to: 5
Classifier: Decision Tree
Accuracy: 0.682
              precision    recall  f1-score   support

           0       0.67      0.66      0.66      1394
           1       0.85      0.84      0.85      1402
           2       0.50      0.50      0.50      1407
           3       0.66      0.67      0.66      1449
           4       0.50      0.50      0.50      1357
           5       0.82      0.81      0.82      1449
           6       0.35      0.35      0.35      1407
           7       0.78      0.79      0.79      1359
           8       0.83      0.84      0.84      1342
           9       0.85      0.85      0.85      1434
```