Thyroid Disease Classification using Neural Network

Data Loading and Preprocessing

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

from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
from sklearn.utils import resample

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

```
# Load the dataset
data_path = 'sample_data/thyroidDF.csv'
df = pd.read_csv(data_path)


# Print the initial shape of the dataset
print("Initial dataset shape:", df.shape)
```

```
Initial dataset shape: (9172, 31)
```

```
print(df.iloc[:, :4], df['target'])
```

```
          age sex on_thyroxine query_on_thyroxine
    0      29   F            f                  f
    1      29   F            f                  f
    2      41   F            f                  f
    3      36   F            f                  f
    4      32   F            f                  f
    ...   ...  ..          ...                ...
    9167   56   M            f                  f
    9168   22   M            f                  f
    9169   69   M            f                  f
    9170   47   F            f                  f
    9171   31   M            f                  f

    [9172 rows x 4 columns] 0       -
    1       -
    2       -
    3       -
    4       S
           ..
    9167    -
    9168    -
    9169    I
    9170    -
    9171    -
    Name: target, Length: 9172, dtype: object
```

```
# Handling missing values
for column in df.columns:
    if df[column].dtype in ['float64', 'int64']:
        df[column].fillna(df[column].median(), inplace=True)
    elif df[column].dtype == 'object':
        df[column].fillna(df[column].mode()[0], inplace=True)
```

```
# Encode categorical variables using Label Encoder
for column in df.columns:
    if df[column].dtype == 'object':
        encoder = LabelEncoder()
        df[column] = encoder.fit_transform(df[column])
```

```
# Balancing skewed dataset
# Separate majority and minority classes
df_majority = df[df.target == df.target.mode()[0]]
df_minority = df[df.target != df.target.mode()[0]]

# Upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,     # sample with replacement
                                 n_samples=len(df_majority),    # to match majc
                                 random_state=123) # reproducible results

# Combine majority class with upsampled minority class
df = pd.concat([df_majority, df_minority_upsampled])

# Shuffle the dataset to avoid any order bias
df = df.sample(frac=1, random_state=42).reset_index(drop=True)

print("New dataset shape:", df.shape)  # Print the new shape of the dataset
```

```
New dataset shape: (13542, 31)
```

```
# Separate features and target variable
features = df.drop(['target', 'patient_id'], axis=1)
labels = df['target']
```

```
# Scale the features
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
```

```
# Split the data into training and testing sets
trainX, testX, trainY, testY = train_test_split(features_scaled, labels, test_s
```

Model

```
# Define the neural network
model = Sequential([
    Dense(128, activation='relu', input_shape=(trainX.shape[1],)),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(len(labels.unique()), activation='softmax')
])
```

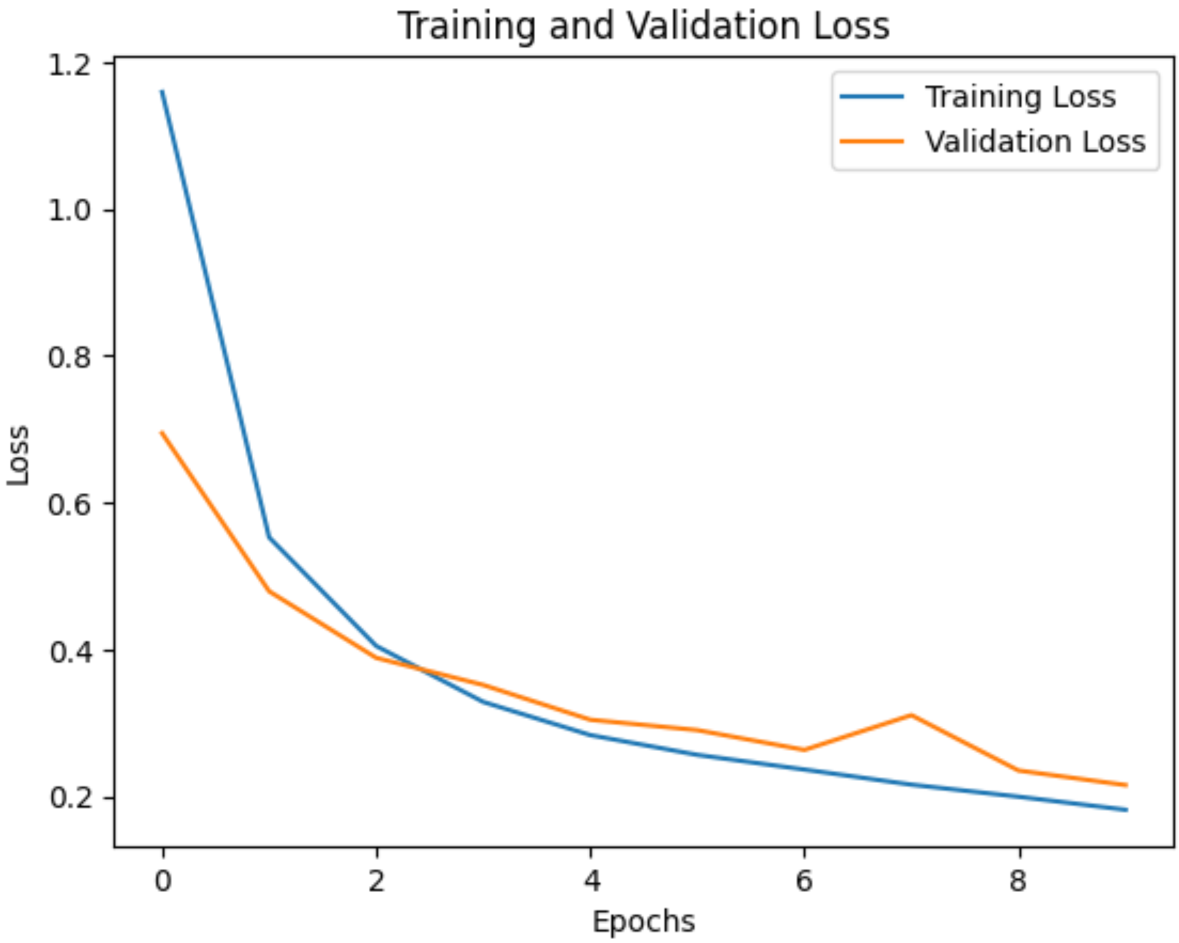
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
# Train the model
history = model.fit(trainX, trainY, epochs=10, validation_data=(testX, testY))
```

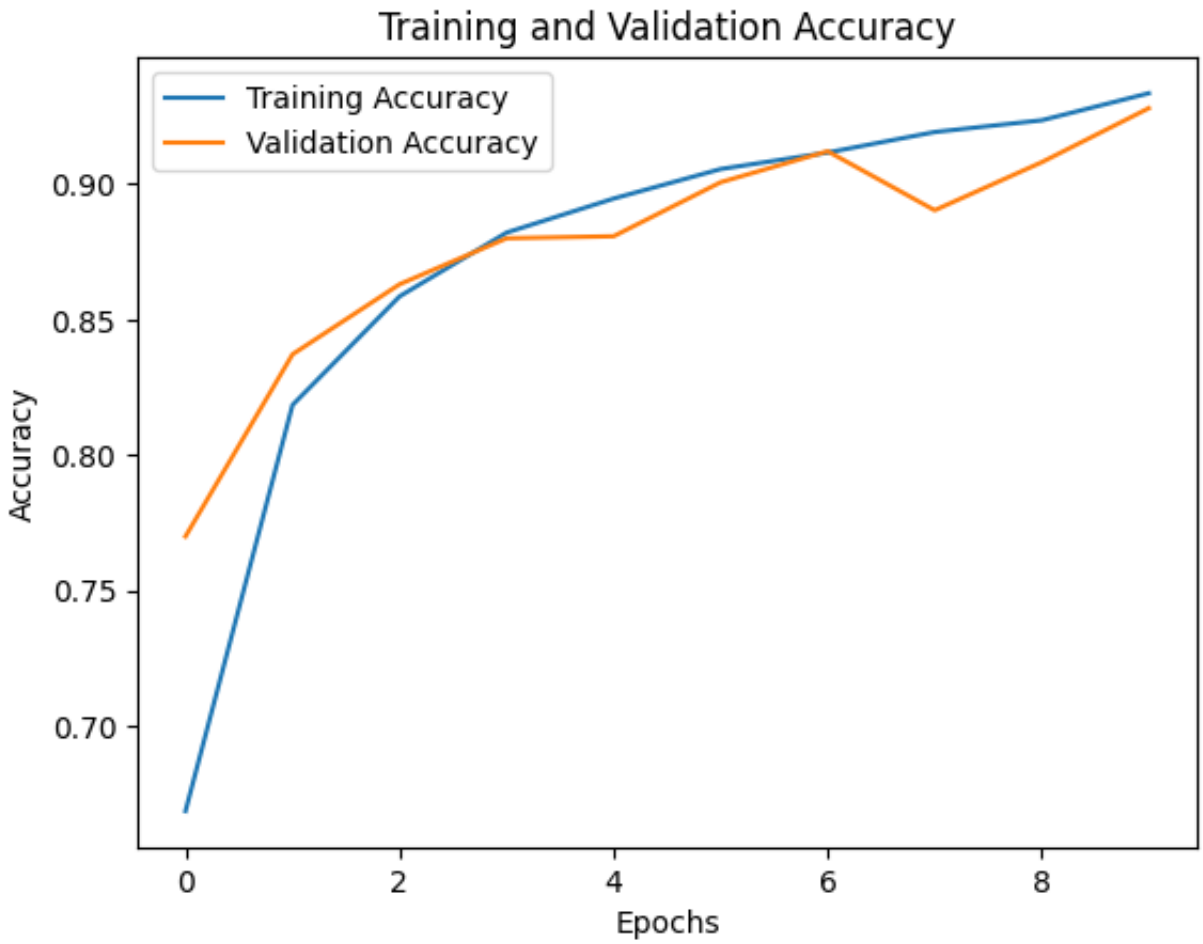
```
Epoch 1/10
339/339 [==============================] - 6s 8ms/step - loss: 1.1593 - acc
Epoch 2/10
339/339 [==============================] - 1s 3ms/step - loss: 0.5526 - acc
Epoch 3/10
339/339 [==============================] - 1s 4ms/step - loss: 0.4047 - acc
Epoch 4/10
339/339 [==============================] - 1s 3ms/step - loss: 0.3284 - acc
Epoch 5/10
339/339 [==============================] - 1s 3ms/step - loss: 0.2832 - acc
Epoch 6/10
339/339 [==============================] - 1s 3ms/step - loss: 0.2563 - acc
Epoch 7/10
339/339 [==============================] - 1s 3ms/step - loss: 0.2363 - acc
Epoch 8/10
339/339 [==============================] - 1s 3ms/step - loss: 0.2157 - acc
Epoch 9/10
339/339 [==============================] - 1s 4ms/step - loss: 0.1993 - acc
Epoch 10/10
339/339 [==============================] - 2s 5ms/step - loss: 0.1815 - acc
```

Plotting Training and Validation Metrics

```
# Plot training and validation loss over epochs
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```



```
# Plot training and validation accuracy over epochs
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()
```



```
predictions = model.predict(testX)
predicted_classes = np.argmax(predictions, axis=1)

accuracy = accuracy_score(testY, predicted_classes)
precision = precision_score(testY, predicted_classes, average='weighted')
recall = recall_score(testY, predicted_classes, average='weighted')
f1 = f1_score(testY, predicted_classes, average='weighted')

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```
85/85 [==============================] - 0s 2ms/step
Accuracy: 0.9280177187153932
Precision: 0.9324928247906574
Recall: 0.9280177187153932
F1-score: 0.9274882283493258
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               3840

 dense_1 (Dense)             (None, 128)               16512

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 32)                2080

=================================================================
Total params: 30688 (119.88 KB)
Trainable params: 30688 (119.88 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```