Import Necessary Library

```
import warnings
```

```
warnings.filterwarnings('ignore')
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout
import matplotlib.pyplot as plt
```

```
2025-01-21 19:40:03.608332: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerica
2025-01-21 19:40:03.617983: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register cuFFT factory: Attempting
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1737466803.629852    7860 cuda_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register factory for plugin cuDN
E0000 00:00:1737466803.633342    7860 cuda_blas.cc:1418] Unable to register cuBLAS factory: Attempting to register factory for plugin cu
2025-01-21 19:40:03.645247: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CP
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compile
```

+ Code + Text

Load dataset

```
data = pd.read_csv("salary.csv")
```

```
data
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	nativ count
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	Unit Sta
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	Unit Sta
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	Unit Sta
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	Unit Sta
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cu
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	Unit Sta
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	Unit Sta

Data Preprocessing

```
categorical_features = ['workclass', 'education', 'marital-status', 'occupation',
                        'relationship', 'race', 'sex', 'native-country']
```

```
one_hot_encoder = OneHotEncoder()
encoded_categorical = one_hot_encoder.fit_transform(data[categorical_features]).toarray()
```

```
encoded_categorical
```

```
array([[0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       ...,
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.]])
```

```
numerical_features = ['age', 'fnlwgt', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']
```

```
scaler = StandardScaler()
scaled_numerical = scaler.fit_transform(data[numerical_features])
```

```
scaled_numerical
```

```
array([[ 0.03067056, -1.06361075,  1.13473876,  0.1484529 , -0.21665953,
        -0.03542945],
       [ 0.83710898, -1.008707  ,  1.13473876, -0.14592048, -0.21665953,
        -2.22215312],
       [-0.04264203,  0.2450785 , -0.42005962, -0.14592048, -0.21665953,
        -0.03542945],
       ...,
       [ 1.42360965, -0.35877741, -0.42005962, -0.14592048, -0.21665953,
        -0.03542945],
       [-1.21564337,  0.11095988, -0.42005962, -0.14592048, -0.21665953,
        -1.65522476],
       [ 0.98373415,  0.92989258, -0.42005962,  1.88842434, -0.21665953,
        -0.03542945]])
```

```
X = np.hstack((scaled_numerical, encoded_categorical))
y = (data['salary'] == '>50K').astype(int)
```

```
X
```

```
array([[ 0.03067056, -1.06361075,  1.13473876, ...,  1.        ,
         0.        ,  0.        ],
       [ 0.83710898, -1.008707  ,  1.13473876, ...,  1.        ,
         0.        ,  0.        ],
       [-0.04264203,  0.2450785 , -0.42005962, ...,  1.        ,
         0.        ,  0.        ],
       ...,
       [ 1.42360965, -0.35877741, -0.42005962, ...,  1.        ,
         0.        ,  0.        ],
       [-1.21564337,  0.11095988, -0.42005962, ...,  1.        ,
         0.        ,  0.        ],
       [ 0.98373415,  0.92989258, -0.42005962, ...,  1.        ,
         0.        ,  0.        ]])
```

```
y.shape
```

```
(32561,)
```

Reshape input for Conv1D

```
X = X.reshape(X.shape[0], X.shape[1], 1)
```

Split dataset

```
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

Build CNN Model with Conv1D

```
model = Sequential([
    Conv1D(32, kernel_size=3, activation='relu', input_shape=(X.shape[1], 1)),
    MaxPooling1D(pool_size=2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

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

```
2025-01-21 19:40:05.124691: E external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:152] failed call to cuInit: INTERNAL: CUDA erro
```

Train the model

```
history = model.fit(X_train, y_train, epochs=5, batch_size=32, validation_data=(X_val, y_val))
```

```
Epoch 1/5
713/713 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9934 - loss: 0.0222 - val_accuracy: 1.0000 - val_loss: 1.8269e-07
Epoch 2/5
713/713 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 1.0000 - loss: 1.1792e-06 - val_accuracy: 1.0000 - val_loss: 1.1829e-08
Epoch 3/5
713/713 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 1.0000 - loss: 2.3776e-07 - val_accuracy: 1.0000 - val_loss: 3.0926e-09
Epoch 4/5
713/713 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 1.0000 - loss: 1.1587e-07 - val_accuracy: 1.0000 - val_loss: 1.1163e-09
Epoch 5/5
713/713 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 1.0000 - loss: 5.3782e-08 - val_accuracy: 1.0000 - val_loss: 4.7718e-10
```

Evaluate the model

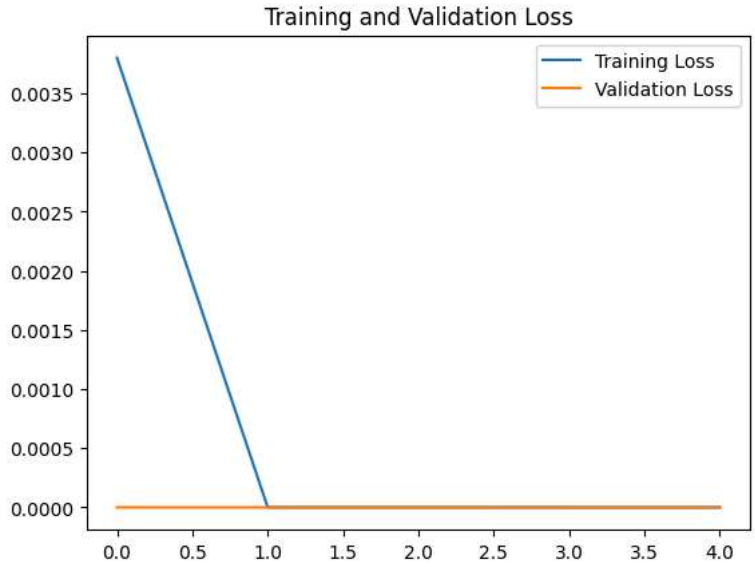
```
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc:.2f}")
```

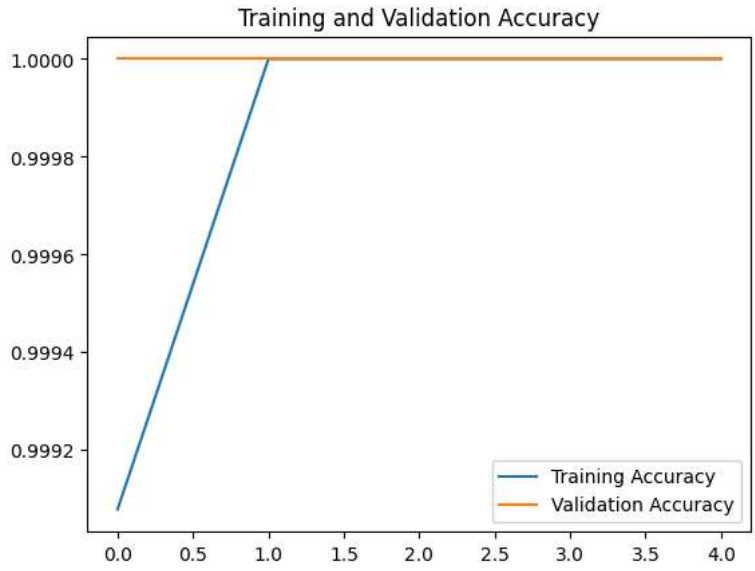
```
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 4.8585e-10
Test Accuracy: 1.00
```

Plot training and validation metrics

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

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





Start coding or generate with AI.