```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import classification_report, confusion_matrix

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Conv1D, MaxPooling1D, Flatten, LSTM,
Embedding, SpatialDropout1D


# Load the CSV file into a DataFra # Update this path accordingly


csv_path = '/kaggle/input/gender-
identification/tensorflow2/genderidentification/1/voice.csv'

voice_data = pd.read_csv(csv_path)


# Encode the labels

le = LabelEncoder()

voice_data['label'] = le.fit_transform(voice_data['label'])


# Prepare features and labels

X = voice_data.drop(columns=['label'])

y = voice_data['label']


# Train-test split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Normalize the features
```

```
X_train = np.expand_dims(X_train.values, axis=2)
X_test = np.expand_dims(X_test.values, axis=2)

# Define CNN Model
cnn_model = Sequential()
cnn_model.add(Conv1D(64, 5, activation='relu', input_shape=(X_train.shape[1], 1)))
cnn_model.add(MaxPooling1D(pool_size=2))
cnn_model.add(Flatten())
cnn_model.add(Dense(10, activation='relu'))
cnn_model.add(Dense(1, activation='sigmoid'))

cnn_model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
cnn_model.summary()

# Train CNN Model
cnn_history = cnn_model.fit(X_train, y_train, epochs=10, batch_size=64,
validation_data=(X_test, y_test), verbose=2)

# Define LSTM Model
lstm_model = Sequential()
lstm_model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2,
input_shape=(X_train.shape[1], 1)))
lstm_model.add(Dense(1, activation='sigmoid'))

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

```
# Train LSTM Model

lstm_history = lstm_model.fit(X_train, y_train, epochs=10, batch_size=64,
validation_data=(X_test, y_test), verbose=2)


# Evaluate CNN Model

cnn_score = cnn_model.evaluate(X_test, y_test, verbose=0)

print(f'CNN Test Accuracy: {cnn_score[1]}')


# Evaluate LSTM Model

lstm_score = lstm_model.evaluate(X_test, y_test, verbose=0)

print(f'LSTM Test Accuracy: {lstm_score[1]}')


# Plotting Accuracy Graph

plt.figure(figsize=(12, 6))


# CNN

plt.plot(cnn_history.history['accuracy'], label='CNN Training Accuracy')

plt.plot(cnn_history.history['val_accuracy'], label='CNN Validation Accuracy')


# LSTM

plt.plot(lstm_history.history['accuracy'], label='LSTM Training Accuracy')

plt.plot(lstm_history.history['val_accuracy'], label='LSTM Validation Accuracy')


plt.title('CNN and LSTM Model Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()
```

```
Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━
━━┓
┃ Layer (type)            ┃ Output Shape        ┃    Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ conv1d (Conv1D)          │ (None, 16, 64)     │      384 │
├──────────────────────────────────┼──────────────────────────┼
─────────────────┤
│ max_pooling1d (MaxPooling1D)  │ (None, 8, 64)      │        0 │
├──────────────────────────────────┼──────────────────────────┼
─────────────────┤
│ flatten (Flatten)        │ (None, 512)       │        0 │
├──────────────────────────────────┼──────────────────────────┼
─────────────────┤
│ dense (Dense)            │ (None, 10)        │    5,130 │
├──────────────────────────────────┼──────────────────────────┼
─────────────────┤
│ dense_1 (Dense)          │ (None, 1)         │       11 │
└──────────────────────────────────┴──────────────────────────┴
─────────────────┘

 Total params: 5,525 (21.58 KB)

 Trainable params: 5,525 (21.58 KB)

 Non-trainable params: 0 (0.00 B)

Epoch 1/10

40/40 - 2s - 41ms/step - accuracy: 0.5055 - loss: 0.7852 - val_accuracy: 0.4921 -
val_loss: 0.7425

Epoch 2/10
```

```
40/40 - 0s - 4ms/step - accuracy: 0.5185 - loss: 0.7093 - val_accuracy: 0.4700 - val_loss:
0.7042

Epoch 3/10

40/40 - 0s - 4ms/step - accuracy: 0.5225 - loss: 0.6998 - val_accuracy: 0.4842 - val_loss:
0.6992

Epoch 4/10

40/40 - 0s - 4ms/step - accuracy: 0.5324 - loss: 0.7640 - val_accuracy: 0.4700 - val_loss:
0.7520

Epoch 5/10

40/40 - 0s - 4ms/step - accuracy: 0.5383 - loss: 0.7073 - val_accuracy: 0.5300 - val_loss:
0.6904

Epoch 6/10

40/40 - 0s - 4ms/step - accuracy: 0.6105 - loss: 0.7112 - val_accuracy: 0.6262 - val_loss:
0.6620

Epoch 7/10

40/40 - 0s - 4ms/step - accuracy: 0.6646 - loss: 0.6496 - val_accuracy: 0.6498 - val_loss:
0.6646

Epoch 8/10

40/40 - 0s - 4ms/step - accuracy: 0.6847 - loss: 0.6396 - val_accuracy: 0.6956 - val_loss:
0.6338

Epoch 9/10

40/40 - 0s - 4ms/step - accuracy: 0.7017 - loss: 0.6465 - val_accuracy: 0.6987 - val_loss:
0.6295

Epoch 10/10

40/40 - 0s - 4ms/step - accuracy: 0.7056 - loss: 0.6054 - val_accuracy: 0.6972 - val_loss:
0.6182

/opt/conda/lib/python3.10/site-packages/keras/src/layers/rnn/rnn.py:204:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first layer in the
model instead.

  super().__init__(**kwargs)
```

**Model: "sequential_1"**

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
━━┓
┃ Layer (type)              ┃ Output Shape          ┃    Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┩
│ lstm (LSTM)               │ (None, 100)           │     40,800 │
├─────────────────────────────────────┼──────────────────────────┼
───────────────┤
│ dense_2 (Dense)           │ (None, 1)             │        101 │
└─────────────────────────────────────┴──────────────────────────┴
───────────────┘
```

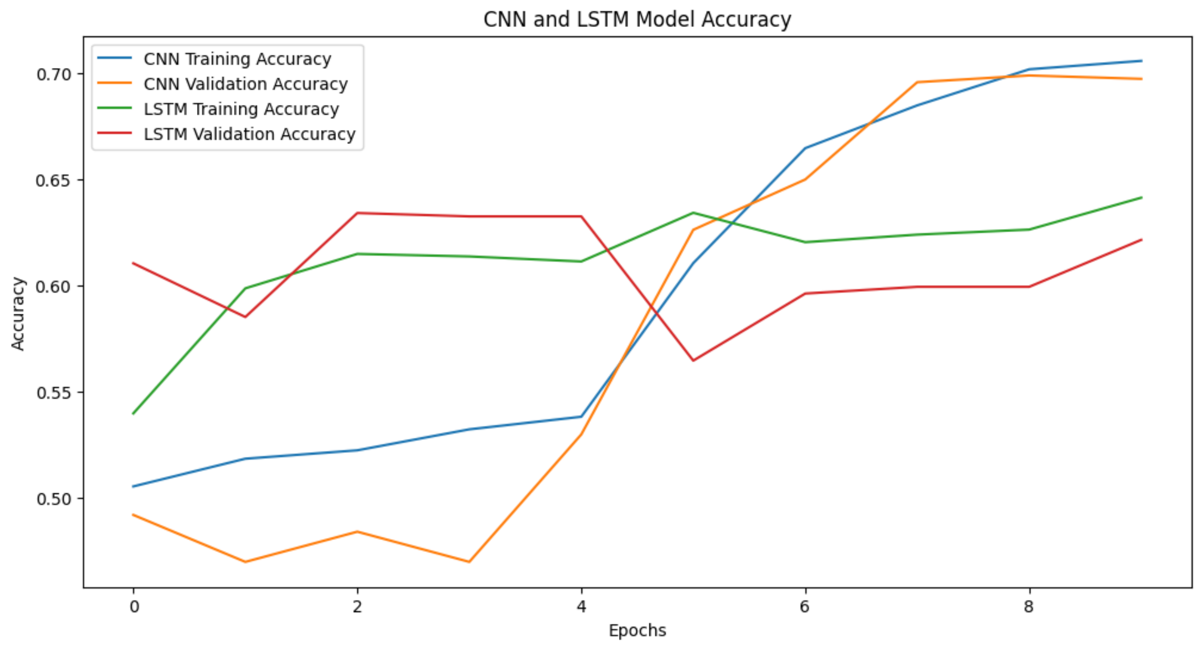
**Total params:** 40,901 (159.77 KB)

**Trainable params:** 40,901 (159.77 KB)

**Non-trainable params:** 0 (0.00 B)

```
Epoch 1/10
40/40 - 3s - 77ms/step - accuracy: 0.5399 - loss: 0.6815 - val_accuracy: 0.6104 -
val_loss: 0.6717
Epoch 2/10
40/40 - 1s - 21ms/step - accuracy: 0.5987 - loss: 0.6627 - val_accuracy: 0.5852 -
val_loss: 0.6762
Epoch 3/10
40/40 - 1s - 20ms/step - accuracy: 0.6148 - loss: 0.6363 - val_accuracy: 0.6341 -
val_loss: 0.6424
Epoch 4/10
40/40 - 1s - 21ms/step - accuracy: 0.6137 - loss: 0.6358 - val_accuracy: 0.6325 -
val_loss: 0.6476
Epoch 5/10
40/40 - 1s - 20ms/step - accuracy: 0.6113 - loss: 0.6291 - val_accuracy: 0.6325 -
val_loss: 0.6410
Epoch 6/10
```

```
40/40 - 1s - 24ms/step - accuracy: 0.6342 - loss: 0.6173 - val_accuracy: 0.5647 -
val_loss: 0.7038

Epoch 7/10

40/40 - 1s - 24ms/step - accuracy: 0.6204 - loss: 0.6254 - val_accuracy: 0.5962 -
val_loss: 0.6501

Epoch 8/10

40/40 - 1s - 29ms/step - accuracy: 0.6239 - loss: 0.6226 - val_accuracy: 0.5994 -
val_loss: 0.6518

Epoch 9/10

40/40 - 1s - 21ms/step - accuracy: 0.6263 - loss: 0.6149 - val_accuracy: 0.5994 -
val_loss: 0.6494

Epoch 10/10

40/40 - 1s - 22ms/step - accuracy: 0.6413 - loss: 0.6168 - val_accuracy: 0.6215 -
val_loss: 0.6474

CNN Test Accuracy: 0.6971608996391296

LSTM Test Accuracy: 0.6214510798454285
```


CNN and LSTM Model Accuracy
CNN Training Accuracy
CNN Validation Accuracy
LSTM Training Accuracy
LSTM Validation Accuracy
Accuracy
Epochs