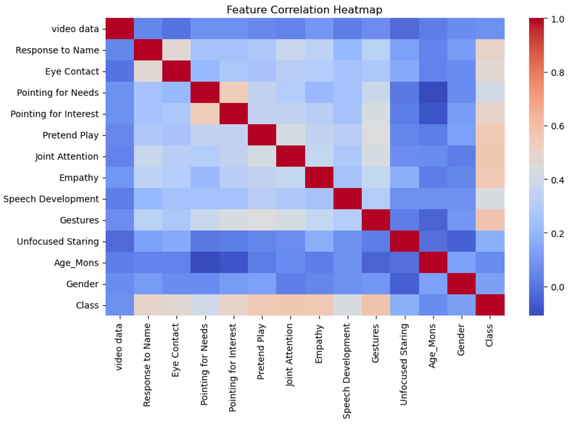
In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import roc_curve, auc, confusion_matrix, classification_report, accuracy_score, precision_score, recall_score, f1_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,Conv1D,MaxPooling2D, Flatten, Dense, LSTM, Reshape, InputLayer, Dropout
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [5]:
```
print("Missing Values:\n", df.isnull().sum())
```

```
Missing Values:
 video data              0
Response to Name         0
Eye Contact              0
Pointing for Needs       0
Pointing for Interest    0
Pretend Play             0
Joint Attention          0
Empathy                  0
Speech Development       0
Gestures                 0
Unfocused Staring        0
Age_Mons                 0
Gender                   0
Class                    0
dtype: int64
```

In [6]:
```
df.columns = df.columns.str.strip()
df.rename(columns={"Class ": "Class"}, inplace=True)
df['Gender'] = df['Gender'].map({'m': 1, 'f': 0})
if 'Class' in df.columns:
    df['Class'] = df['Class'].map({'Yes': 1, 'No': 0})
else:
    print("Warning: 'Class' column not found in dataset")
```

In [7]:
```
# Correlation heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), cmap='coolwarm', cbar=True)
plt.title("Feature Correlation Heatmap")
plt.show()
```



In [8]:
```
# Prepare data for modeling
X = df.drop(columns=['Class'])
y = df['Class']

# Ensure correct feature count
num_features = X.shape[1]

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

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [14]:
```
# Simulated ROC curve data
y_scores = np.random.rand(len(y_test))  # Placeholder scores
fpr, tpr, _ = roc_curve(y_test, y_scores)
roc_auc = auc(fpr, tpr)
```

In [15]:
```
# Evaluation function
def evaluate_model(model, X_test, y_test):
    y_pred = (model.predict(X_test) > 0.5).astype("int32")
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print(f"Accuracy: {acc:.4f}, Precision: {prec:.4f}, Recall: {rec:.4f}, F1 Score: {f1:.4f}")
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(6,6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No ASD', 'ASD'], yticklabels=['No ASD', 'ASD'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()
```

In [16]:
```
# CNN Model
cnn_model = Sequential([
    InputLayer(shape=(num_features,)),
    Reshape((num_features, 1, 1)),
    Conv2D(16, kernel_size=(3,1), activation='relu', padding='same'),  # Adjust kernel size
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])
```

In [18]:
```
# RNN Model
rnn_model = Sequential([
    InputLayer(shape=(num_features,)),
    Reshape((num_features, 1)),  # Reshape for LSTM input
    LSTM(50, activation='relu', return_sequences=False),
    Dense(1, activation='sigmoid')
])
rnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
rnn_model.fit(X_train, y_train, epochs=1, batch_size=32, validation_data=(X_test, y_test))
```

```
27/27 ━━━━━━━━━━━━━━━━━━━━ 8s 64ms/step - accuracy: 0.6344 - loss: 0.6728 - val_accuracy: 0.8531 - val_loss: 0.5609
```

Out[18]:
```
<keras.src.callbacks.history.History at 0x1950f608770>
```

In [19]:
```
# Hybrid CNN + LSTM Model
num_features = X_train.shape[1]  # Number of features in your data

# Adjusting to Conv1D for simplicity
hybrid_model = Sequential([
    InputLayer(input_shape=(num_features, 1)),
    Conv1D(16, kernel_size=3, activation='relu', padding='same'),
    Flatten(),
    Dense(64, activation='relu'),
    Reshape((64, 1)),  # Reshaping to match LSTM input requirements
    LSTM(50, activation='relu'),
    Dense(1, activation='sigmoid')
])
hybrid_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
hybrid_model.fit(X_train[..., np.newaxis], y_train, epochs=6, batch_size=42, validation_data=(X_test[..., np.newaxis], y_test))
```

```
C:\Users\Nirupama G N\anaconda3\Lib\site-packages\keras\src\layers\core\input_layer.py:27: UserWarning: Argument `input_shape` is deprecated. Use `shape` instead.
  warnings.warn(
Epoch 1/6
21/21 ━━━━━━━━━━━━━━━━━━━━ 12s 137ms/step - accuracy: 0.6883 - loss: 0.6804 - val_accuracy: 0.6730 - val_loss: 0.6191
Epoch 2/6
21/21 ━━━━━━━━━━━━━━━━━━━━ 2s 80ms/step - accuracy: 0.6887 - loss: 0.5318 - val_accuracy: 0.6730 - val_loss: 0.3586
Epoch 3/6
21/21 ━━━━━━━━━━━━━━━━━━━━ 3s 85ms/step - accuracy: 0.7013 - loss: 0.3268 - val_accuracy: 0.6730 - val_loss: 0.3008
Epoch 4/6
21/21 ━━━━━━━━━━━━━━━━━━━━ 3s 81ms/step - accuracy: 0.6933 - loss: 0.3440 - val_accuracy: 0.6730 - val_loss: 0.3568
Epoch 5/6
21/21 ━━━━━━━━━━━━━━━━━━━━ 3s 85ms/step - accuracy: 0.7044 - loss: 0.3160 - val_accuracy: 0.6730 - val_loss: 0.2887
Epoch 6/6
21/21 ━━━━━━━━━━━━━━━━━━━━ 2s 87ms/step - accuracy: 0.6858 - loss: 0.2866 - val_accuracy: 0.7488 - val_loss: 0.2648
```

Out[19]:
```
<keras.src.callbacks.history.History at 0x19510abc560>
```

In [20]:
```
# Dictionary to store metrics
results = {
    'Model': [],
    'Accuracy': [],
    'Precision': [],
    'Recall': [],
    'F1 Score': []
}

# Function to evaluate models and return metrics
def evaluate_model(model, X_test, y_test, model_name):
    # Predict probabilities for ROC curve
    y_pred_prob = model.predict(X_test)
    y_pred = (y_pred_prob > 0.5).astype("int32")

    # Calculate metrics
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    # Add results to dictionary
    results['Model'].append(model_name)
    results['Accuracy'].append(acc)
    results['Precision'].append(prec)
    results['Recall'].append(rec)
    results['F1 Score'].append(f1)

# Evaluate each model
evaluate_model(cnn_model, X_test, y_test, 'CNN')
evaluate_model(rnn_model, X_test, y_test, 'RNN')
evaluate_model(hybrid_model, X_test, y_test, 'Hybrid CNN + LSTM')

# Convert the results dictionary into a pandas DataFrame
results_df = pd.DataFrame(results)

# Set 'Model' as the index
results_df.set_index('Model', inplace=True)

# Display the summary table without numerical indexes
print(results_df)
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 101ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 142ms/step
                    Accuracy  Precision  Recall  F1 Score
Model
CNN                 0.549763   0.747368     0.5  0.599156
```

```
RNN               0.853081   0.820809     1.0  0.901587
Hybrid CNN + LSTM  0.748815   0.728205     1.0  0.842730
```

In [ ]: