Model Development Phase Template

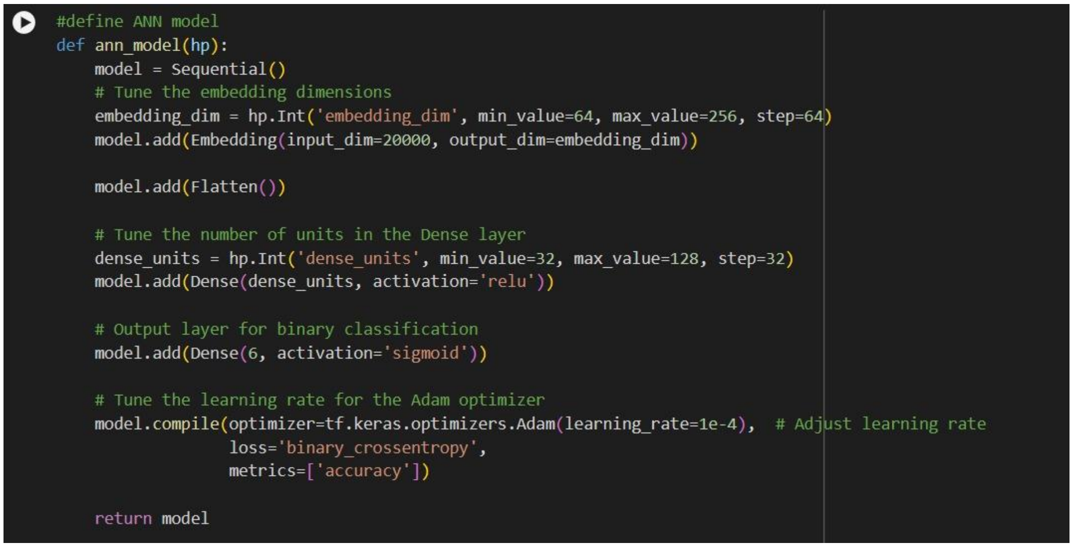
Date	26 November 2024
Team ID	SWTID1726490119
Project Title	Toxic Comment Classification for Social Media.
Maximum Marks	10 Marks

Initial Model Training Code, Model Validation and Evaluation Report

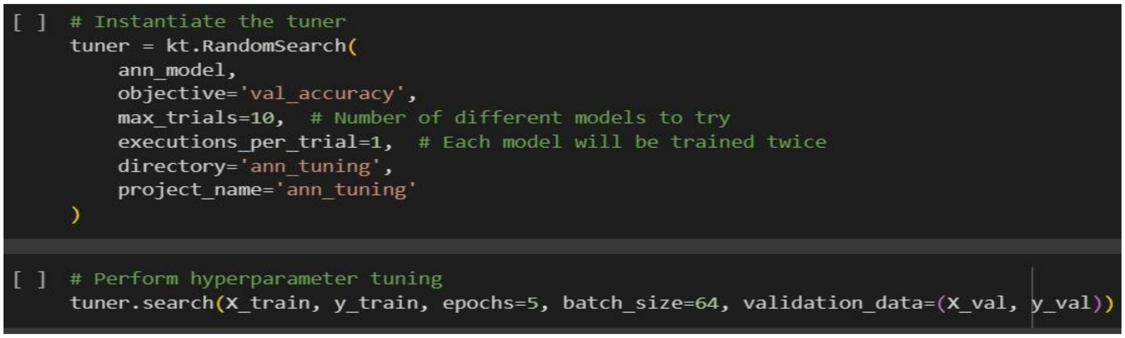
The initial model training code will be displayed via a screenshot, showcasing the architecture and training process for the selected model.The model validation and evaluation report will summarize the training and validation performance metrics (accuracy, loss, etc.) for multiple models, with respective screenshots of evaluation results like classification reports and accuracy plots.

Initial Model Training Code (5 marks):

1. Simple Neural Network (ANN)

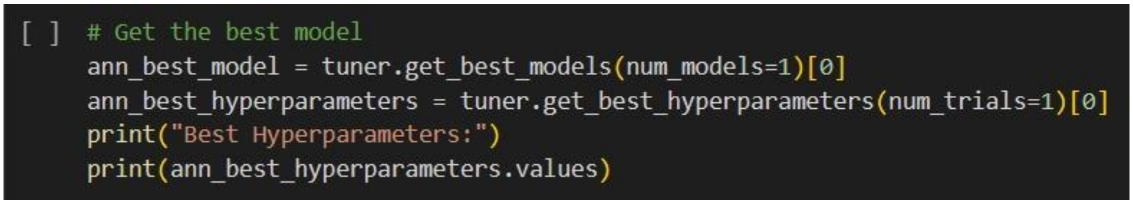


```
#define ANN model
def ann_model(hp):
    model = Sequential()
    # Tune the embedding dimensions
    embedding_dim = hp.Int('embedding_dim', min_value=64, max_value=256, step=64)
    model.add(Embedding(input_dim=20000, output_dim=embedding_dim))

    model.add(Flatten())

    # Tune the number of units in the Dense layer
    dense_units = hp.Int('dense_units', min_value=32, max_value=128, step=32)
    model.add(Dense(dense_units, activation='relu'))

    # Output layer for binary classification
    model.add(Dense(6, activation='sigmoid'))

    # Tune the learning rate for the Adam optimizer
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),  # Adjust learning rate
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    return model
```

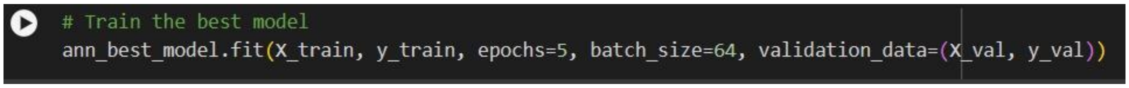


```
[ ]  # Instantiate the tuner
     tuner = kt.RandomSearch(
         ann_model,
         objective='val_accuracy',
         max_trials=10,  # Number of different models to try
         executions_per_trial=1,  # Each model will be trained twice
         directory='ann_tuning',
         project_name='ann_tuning'
     )
```

```
[ ]  # Perform hyperparameter tuning
     tuner.search(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_val, y_val))
```

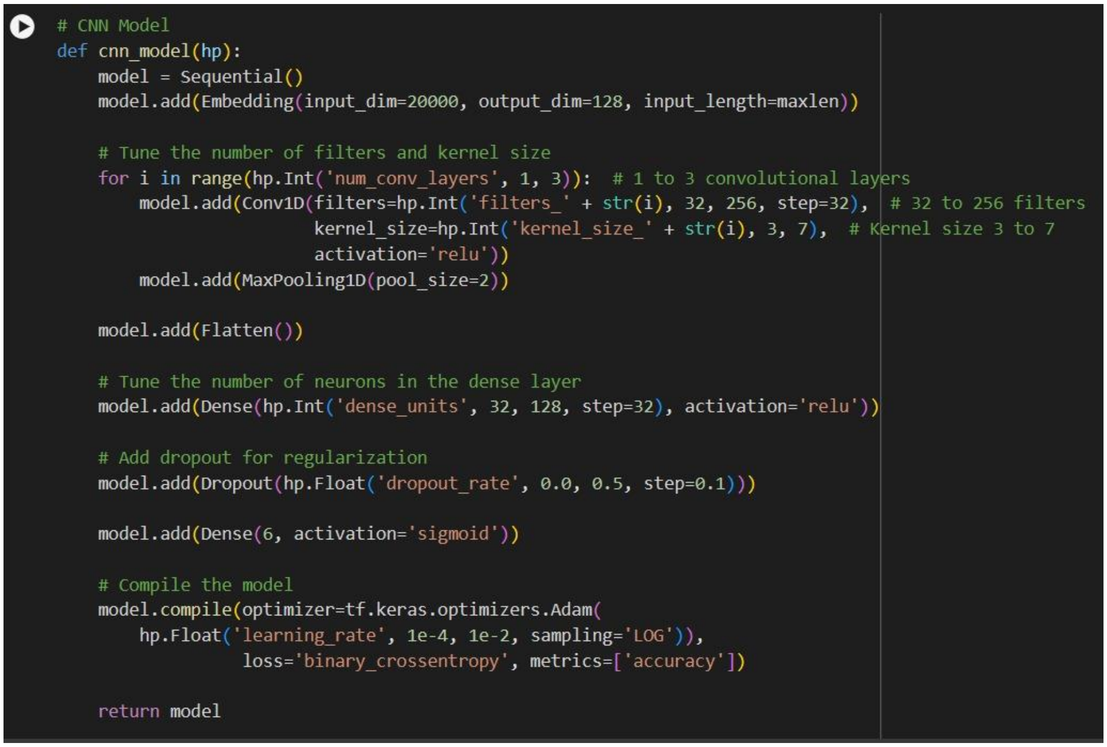


```
[ ]  # Get the best model
     ann_best_model = tuner.get_best_models(num_models=1)[0]
     ann_best_hyperparameters = tuner.get_best_hyperparameters(num_trials=1)[0]
     print("Best Hyperparameters:")
     print(ann_best_hyperparameters.values)
```

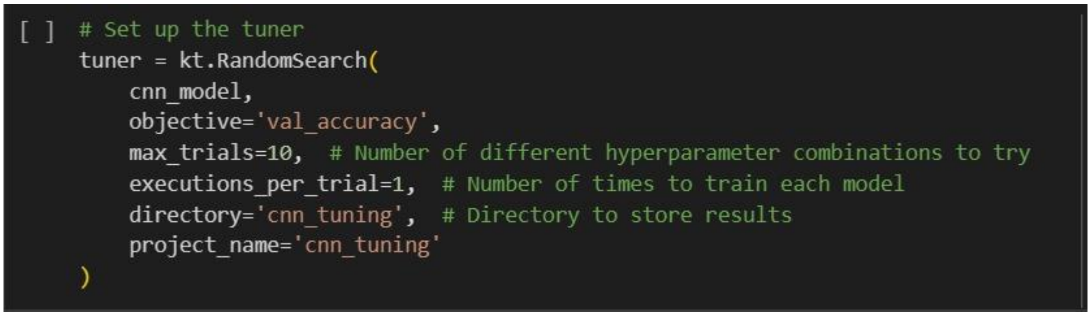


```
# Train the best model
ann_best_model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_val, y_val))
```

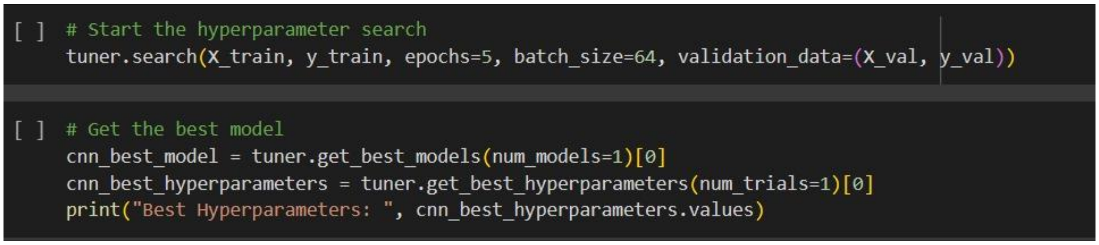
2. Convulational Neural Network (CNN)



```
# CNN Model
def cnn_model(hp):
    model = Sequential()
    model.add(Embedding(input_dim=20000, output_dim=128, input_length=maxlen))

    # Tune the number of filters and kernel size
    for i in range(hp.Int('num_conv_layers', 1, 3)):  # 1 to 3 convolutional layers
        model.add(Conv1D(filters=hp.Int('filters_' + str(i), 32, 256, step=32),  # 32 to 256 filters
                         kernel_size=hp.Int('kernel_size_' + str(i), 3, 7),  # Kernel size 3 to 7
                         activation='relu'))
        model.add(MaxPooling1D(pool_size=2))

    model.add(Flatten())

    # Tune the number of neurons in the dense layer
    model.add(Dense(hp.Int('dense_units', 32, 128, step=32), activation='relu'))

    # Add dropout for regularization
    model.add(Dropout(hp.Float('dropout_rate', 0.0, 0.5, step=0.1)))

    model.add(Dense(6, activation='sigmoid'))

    # Compile the model
    model.compile(optimizer=tf.keras.optimizers.Adam(
        hp.Float('learning_rate', 1e-4, 1e-2, sampling='LOG')),
                  loss='binary_crossentropy', metrics=['accuracy'])

    return model
```

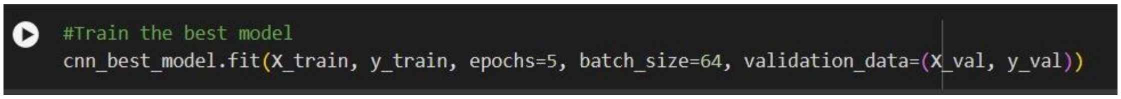


```
[ ]  # Set up the tuner
     tuner = kt.RandomSearch(
         cnn_model,
         objective='val_accuracy',
         max_trials=10,  # Number of different hyperparameter combinations to try
         executions_per_trial=1,  # Number of times to train each model
         directory='cnn_tuning',  # Directory to store results
         project_name='cnn_tuning'
     )
```



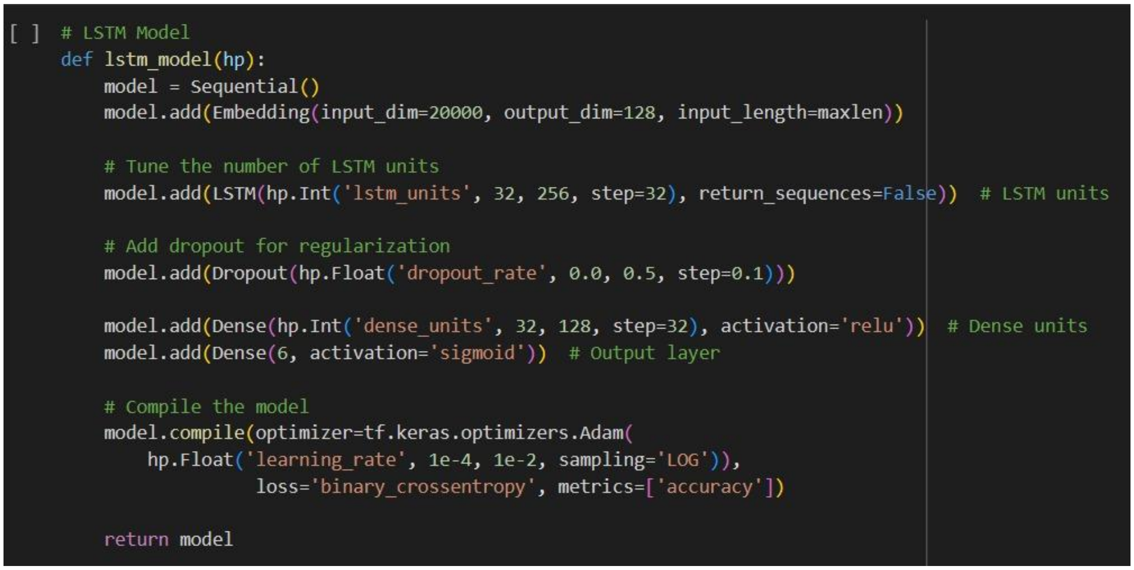
```
[ ]  # Start the hyperparameter search
     tuner.search(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_val, y_val))
```

```
[ ]  # Get the best model
     cnn_best_model = tuner.get_best_models(num_models=1)[0]
     cnn_best_hyperparameters = tuner.get_best_hyperparameters(num_trials=1)[0]
     print("Best Hyperparameters: ", cnn_best_hyperparameters.values)
```

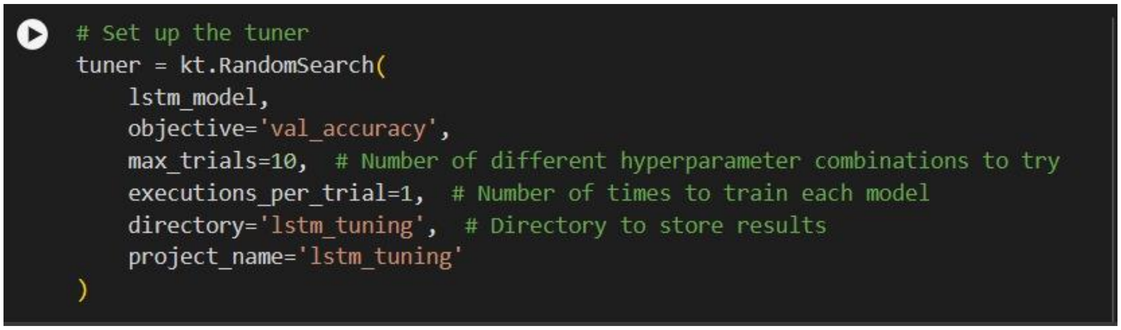


```
#Train the best model
cnn_best_model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_val, y_val))
```

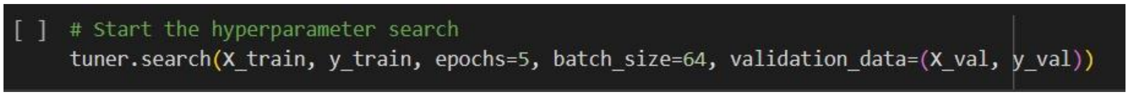
3. Long Short-Term Memory (LSTM)



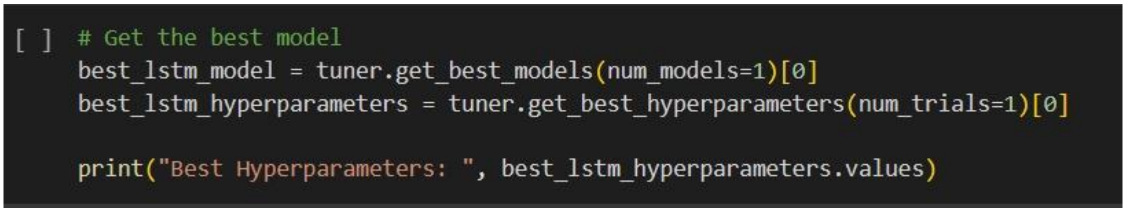
```
[ ]  # LSTM Model
     def lstm_model(hp):
         model = Sequential()
         model.add(Embedding(input_dim=20000, output_dim=128, input_length=maxlen))

         # Tune the number of LSTM units
         model.add(LSTM(hp.Int('lstm_units', 32, 256, step=32), return_sequences=False))  # LSTM units

         # Add dropout for regularization
         model.add(Dropout(hp.Float('dropout_rate', 0.0, 0.5, step=0.1)))

         model.add(Dense(hp.Int('dense_units', 32, 128, step=32), activation='relu'))  # Dense units
         model.add(Dense(6, activation='sigmoid'))  # Output layer

         # Compile the model
         model.compile(optimizer=tf.keras.optimizers.Adam(
             hp.Float('learning_rate', 1e-4, 1e-2, sampling='LOG')),
                       loss='binary_crossentropy', metrics=['accuracy'])

         return model
```



```
# Set up the tuner
tuner = kt.RandomSearch(
    lstm_model,
    objective='val_accuracy',
    max_trials=10,  # Number of different hyperparameter combinations to try
    executions_per_trial=1,  # Number of times to train each model
    directory='lstm_tuning',  # Directory to store results
    project_name='lstm_tuning'
)
```

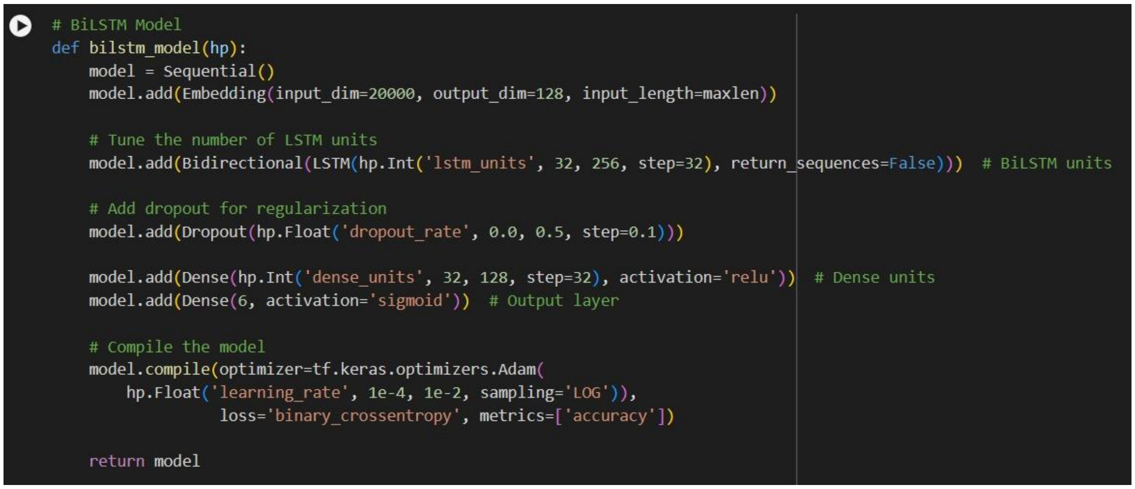


```
[ ] # Start the hyperparameter search
    tuner.search(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_val, y_val))
```

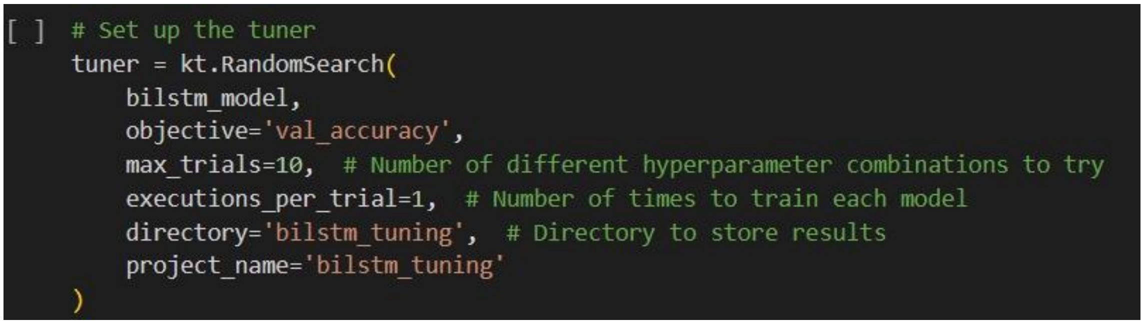


```
[ ] # Get the best model
    best_lstm_model = tuner.get_best_models(num_models=1)[0]
    best_lstm_hyperparameters = tuner.get_best_hyperparameters(num_trials=1)[0]

    print("Best Hyperparameters: ", best_lstm_hyperparameters.values)
```

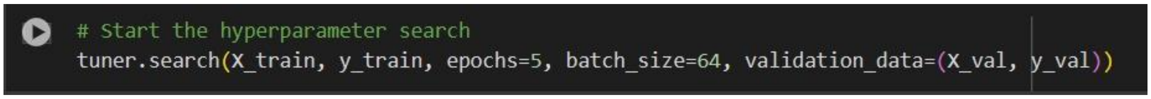
4. Bi-Directional LSTM (BiLSTM)



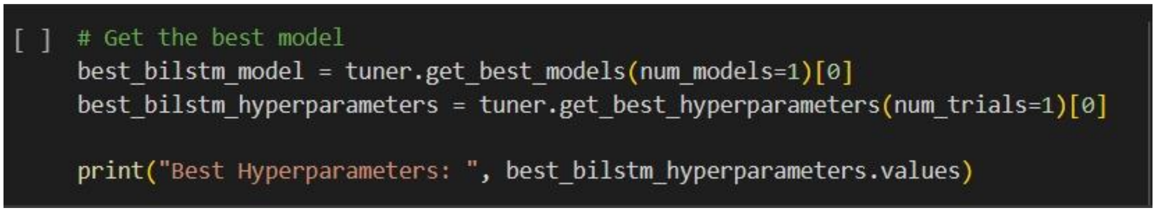
```
# BiLSTM Model
def bilstm_model(hp):
    model = Sequential()
    model.add(Embedding(input_dim=20000, output_dim=128, input_length=maxlen))

    # Tune the number of LSTM units
    model.add(Bidirectional(LSTM(hp.Int('lstm_units', 32, 256, step=32), return_sequences=False)))  # BiLSTM units

    # Add dropout for regularization
    model.add(Dropout(hp.Float('dropout_rate', 0.0, 0.5, step=0.1)))

    model.add(Dense(hp.Int('dense_units', 32, 128, step=32), activation='relu'))  # Dense units
    model.add(Dense(6, activation='sigmoid'))  # Output layer

    # Compile the model
    model.compile(optimizer=tf.keras.optimizers.Adam(
        hp.Float('learning_rate', 1e-4, 1e-2, sampling='LOG')),
                  loss='binary_crossentropy', metrics=['accuracy'])

    return model
```



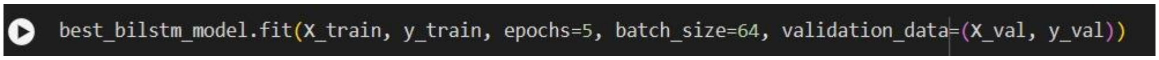
```
[ ] # Set up the tuner
    tuner = kt.RandomSearch(
        bilstm_model,
        objective='val_accuracy',
        max_trials=10,  # Number of different hyperparameter combinations to try
        executions_per_trial=1,  # Number of times to train each model
        directory='bilstm_tuning',  # Directory to store results
        project_name='bilstm_tuning'
    )
```



```
# Start the hyperparameter search
tuner.search(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_val, y_val))
```



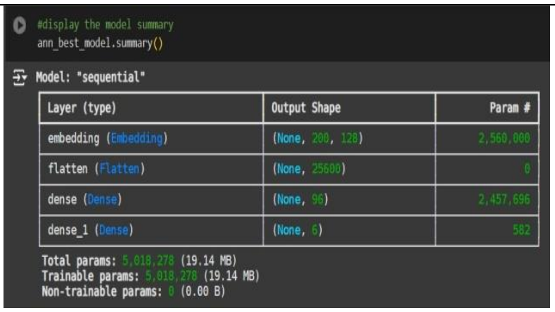
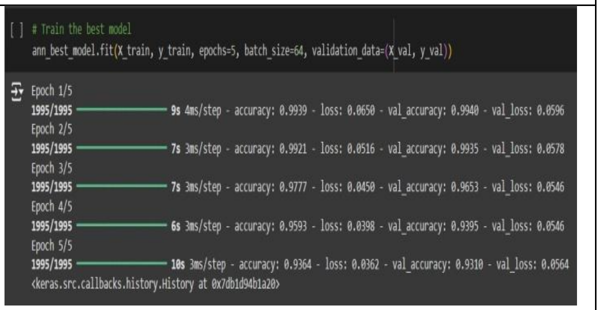
```
[ ] # Get the best model
    best_bilstm_model = tuner.get_best_models(num_models=1)[0]
    best_bilstm_hyperparameters = tuner.get_best_hyperparameters(num_trials=1)[0]

    print("Best Hyperparameters: ", best_bilstm_hyperparameters.values)
```

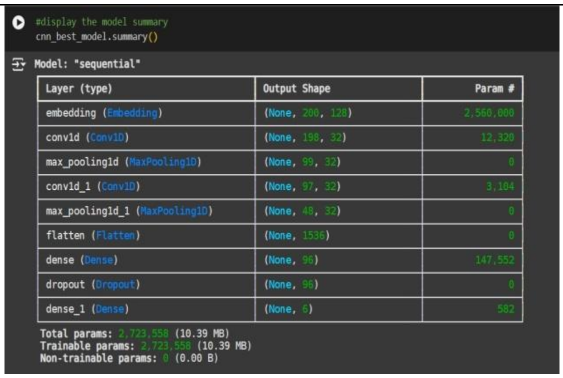
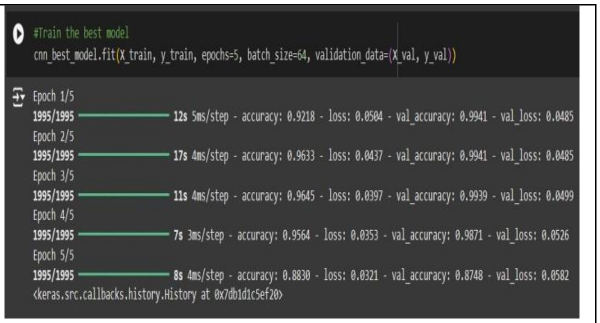
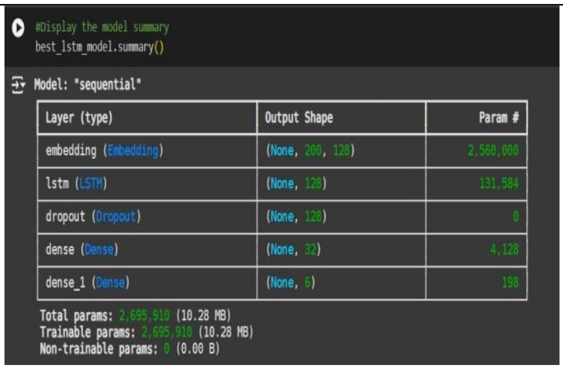
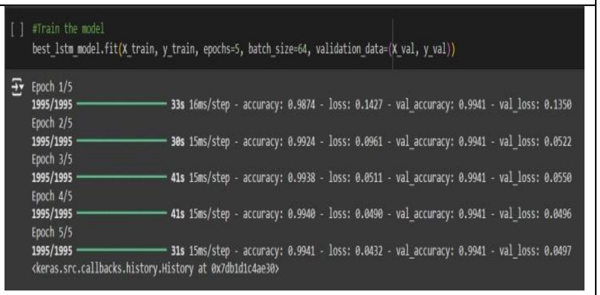
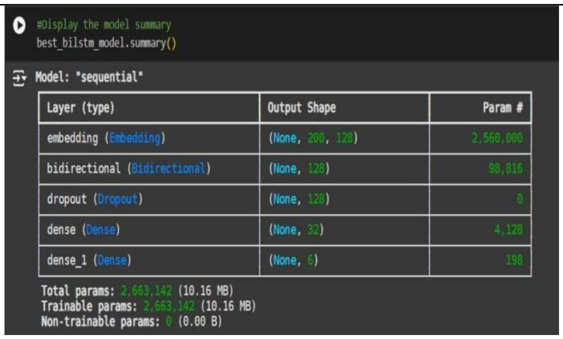
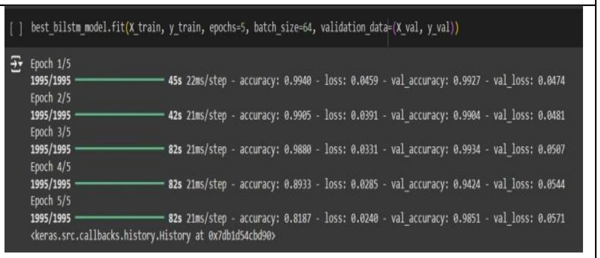


```
best_bilstm_model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_val, y_val))
```

Model Validation and Evaluation Report (5 marks):

Model	Summary	Training and Validation Performance Metrics

Simple Neural Network (ANN)	
	`#display the model summary`
	`ann_best_model.summary()`
	`Model: "sequential"`
	`Layer (type) │ Output Shape │ Param #`
	`embedding (Embedding) │ (None, 200, 128) │ 2,560,000`
	`flatten (Flatten) │ (None, 25600) │ 0`
	`dense (Dense) │ (None, 96) │ 2,457,696`
	`dense_1 (Dense) │ (None, 6) │ 582`
	`Total params: 5,018,278 (19.14 MB)`
	`Trainable params: 5,018,278 (19.14 MB)`
	`Non-trainable params: 0 (0.00 B)`
		
		`[ ] # Train the best model`
		`ann_best_model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_val, y_val))`
		`Epoch 1/5`
		`1995/1995 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.9939 - loss: 0.0650 - val_accuracy: 0.9940 - val_loss: 0.0596`
		`Epoch 2/5`
		`1995/1995 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.9921 - loss: 0.0516 - val_accuracy: 0.9935 - val_loss: 0.0578`
		`Epoch 3/5`
		`1995/1995 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.9777 - loss: 0.0450 - val_accuracy: 0.9653 - val_loss: 0.0546`
		`Epoch 4/5`
		`1995/1995 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9593 - loss: 0.0398 - val_accuracy: 0.9395 - val_loss: 0.0546`
		`Epoch 5/5`
		`1995/1995 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9364 - loss: 0.0362 - val_accuracy: 0.9310 - val_loss: 0.0564`
		`<keras.src.callbacks.history.History at 0x7db1d94b1a20>`

Convulational Neural Network (CNN)		
Long Short-Term Memory (LSTM)		
Bi-Directional LSTM (BiLSTM)		

```
#display the model summary
cnn_best_model.summary()

Model: "sequential"
┌──────────────────────────────┬─────────────────────┬─────────────┐
│ Layer (type)                 │ Output Shape        │     Param # │
├──────────────────────────────┼─────────────────────┼─────────────┤
│ embedding (Embedding)        │ (None, 200, 128)    │   2,560,000 │
├──────────────────────────────┼─────────────────────┼─────────────┤
│ conv1d (Conv1D)              │ (None, 198, 32)     │      12,320 │
├──────────────────────────────┼─────────────────────┼─────────────┤
│ max_pooling1d (MaxPooling1D) │ (None, 99, 32)      │           0 │
├──────────────────────────────┼─────────────────────┼─────────────┤
│ conv1d_1 (Conv1D)            │ (None, 97, 32)      │       3,104 │
├──────────────────────────────┼─────────────────────┼─────────────┤
│ max_pooling1d_1 (MaxPooling1D) │ (None, 48, 32)    │           0 │
├──────────────────────────────┼─────────────────────┼─────────────┤
│ flatten (Flatten)            │ (None, 1536)        │           0 │
├──────────────────────────────┼─────────────────────┼─────────────┤
│ dense (Dense)                │ (None, 96)          │     147,552 │
├──────────────────────────────┼─────────────────────┼─────────────┤
│ dropout (Dropout)            │ (None, 96)          │           0 │
├──────────────────────────────┼─────────────────────┼─────────────┤
│ dense_1 (Dense)              │ (None, 6)           │         582 │
└──────────────────────────────┴─────────────────────┴─────────────┘
 Total params: 2,723,558 (10.39 MB)
 Trainable params: 2,723,558 (10.39 MB)
 Non-trainable params: 0 (0.00 B)
```

```
#Train the best model
cnn_best_model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_val, y_val))

Epoch 1/5
1995/1995 ━━━━━━━━━━━━━━━━━━━━ 12s 5ms/step - accuracy: 0.9218 - loss: 0.0504 - val_accuracy: 0.9941 - val_loss: 0.0485
Epoch 2/5
1995/1995 ━━━━━━━━━━━━━━━━━━━━ 17s 4ms/step - accuracy: 0.9633 - loss: 0.0437 - val_accuracy: 0.9941 - val_loss: 0.0485
Epoch 3/5
1995/1995 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.9645 - loss: 0.0397 - val_accuracy: 0.9939 - val_loss: 0.0499
Epoch 4/5
1995/1995 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.9564 - loss: 0.0353 - val_accuracy: 0.9871 - val_loss: 0.0526
Epoch 5/5
1995/1995 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.8830 - loss: 0.0321 - val_accuracy: 0.8748 - val_loss: 0.0582
<keras.src.callbacks.history.History at 0x7db1d1c5ef20>
```

```
#Display the model summary
best_lstm_model.summary()

Model: "sequential"
┌──────────────────────────────┬─────────────────────┬─────────────┐
│ Layer (type)                 │ Output Shape        │     Param # │
├──────────────────────────────┼─────────────────────┼─────────────┤
│ embedding (Embedding)        │ (None, 200, 128)    │   2,560,000 │
├──────────────────────────────┼─────────────────────┼─────────────┤
│ lstm (LSTM)                  │ (None, 128)         │     131,584 │
├──────────────────────────────┼─────────────────────┼─────────────┤
│ dropout (Dropout)            │ (None, 128)         │           0 │
├──────────────────────────────┼─────────────────────┼─────────────┤
│ dense (Dense)                │ (None, 32)          │       4,128 │
├──────────────────────────────┼─────────────────────┼─────────────┤
│ dense_1 (Dense)              │ (None, 6)           │         198 │
└──────────────────────────────┴─────────────────────┴─────────────┘
 Total params: 2,695,910 (10.28 MB)
 Trainable params: 2,695,910 (10.28 MB)
 Non-trainable params: 0 (0.00 B)
```

```
[ ] #Train the model
    best_lstm_model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_val, y_val))

Epoch 1/5
1995/1995 ━━━━━━━━━━━━━━━━━━━━ 33s 16ms/step - accuracy: 0.9874 - loss: 0.1427 - val_accuracy: 0.9941 - val_loss: 0.1350
Epoch 2/5
1995/1995 ━━━━━━━━━━━━━━━━━━━━ 30s 15ms/step - accuracy: 0.9924 - loss: 0.0961 - val_accuracy: 0.9941 - val_loss: 0.0522
Epoch 3/5
1995/1995 ━━━━━━━━━━━━━━━━━━━━ 41s 15ms/step - accuracy: 0.9938 - loss: 0.0511 - val_accuracy: 0.9941 - val_loss: 0.0550
Epoch 4/5
1995/1995 ━━━━━━━━━━━━━━━━━━━━ 41s 15ms/step - accuracy: 0.9940 - loss: 0.0490 - val_accuracy: 0.9941 - val_loss: 0.0496
Epoch 5/5
1995/1995 ━━━━━━━━━━━━━━━━━━━━ 31s 15ms/step - accuracy: 0.9941 - loss: 0.0432 - val_accuracy: 0.9941 - val_loss: 0.0497
<keras.src.callbacks.history.History at 0x7db1d1c4ae30>
```

```
#Display the model summary
best_bilstm_model.summary()

Model: "sequential"
┌──────────────────────────────┬─────────────────────┬─────────────┐
│ Layer (type)                 │ Output Shape        │     Param # │
├──────────────────────────────┼─────────────────────┼─────────────┤
│ embedding (Embedding)        │ (None, 200, 128)    │   2,560,000 │
├──────────────────────────────┼─────────────────────┼─────────────┤
│ bidirectional (Bidirectional) │ (None, 128)        │      98,816 │
├──────────────────────────────┼─────────────────────┼─────────────┤
│ dropout (Dropout)            │ (None, 128)         │           0 │
├──────────────────────────────┼─────────────────────┼─────────────┤
│ dense (Dense)                │ (None, 32)          │       4,128 │
├──────────────────────────────┼─────────────────────┼─────────────┤
│ dense_1 (Dense)              │ (None, 6)           │         198 │
└──────────────────────────────┴─────────────────────┴─────────────┘
 Total params: 2,663,142 (10.16 MB)
 Trainable params: 2,663,142 (10.16 MB)
 Non-trainable params: 0 (0.00 B)
```

```
[ ] best_bilstm_model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_val, y_val))

Epoch 1/5
1995/1995 ━━━━━━━━━━━━━━━━━━━━ 45s 22ms/step - accuracy: 0.9940 - loss: 0.0459 - val_accuracy: 0.9927 - val_loss: 0.0474
Epoch 2/5
1995/1995 ━━━━━━━━━━━━━━━━━━━━ 42s 21ms/step - accuracy: 0.9905 - loss: 0.0391 - val_accuracy: 0.9904 - val_loss: 0.0481
Epoch 3/5
1995/1995 ━━━━━━━━━━━━━━━━━━━━ 82s 21ms/step - accuracy: 0.9880 - loss: 0.0331 - val_accuracy: 0.9934 - val_loss: 0.0507
Epoch 4/5
1995/1995 ━━━━━━━━━━━━━━━━━━━━ 82s 21ms/step - accuracy: 0.8933 - loss: 0.0285 - val_accuracy: 0.9424 - val_loss: 0.0544
Epoch 5/5
1995/1995 ━━━━━━━━━━━━━━━━━━━━ 82s 21ms/step - accuracy: 0.8187 - loss: 0.0240 - val_accuracy: 0.9851 - val_loss: 0.0571
<keras.src.callbacks.history.History at 0x7db1d54cbd90>
```