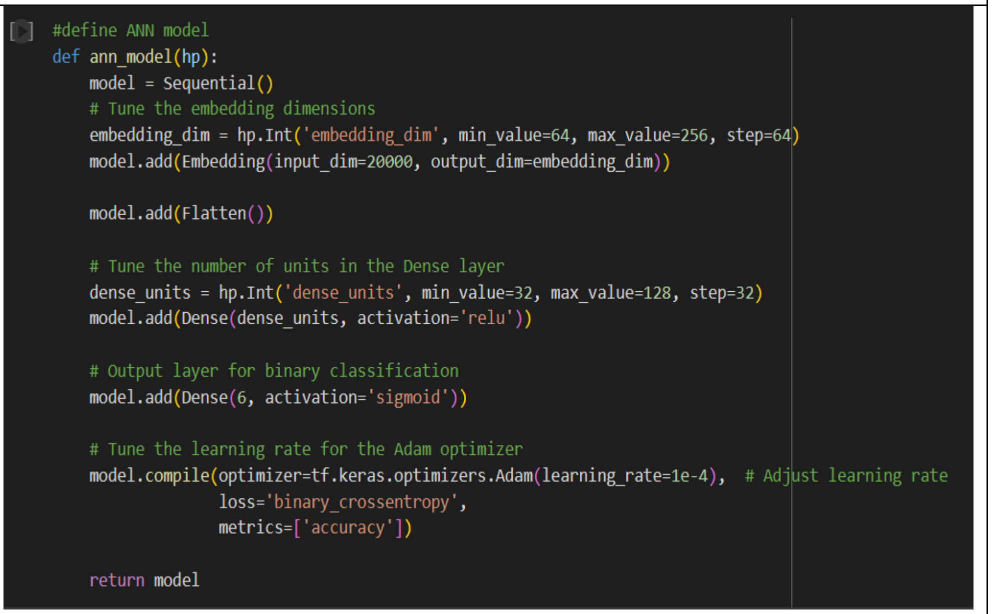
Model Optimization and Tuning Phase Template

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

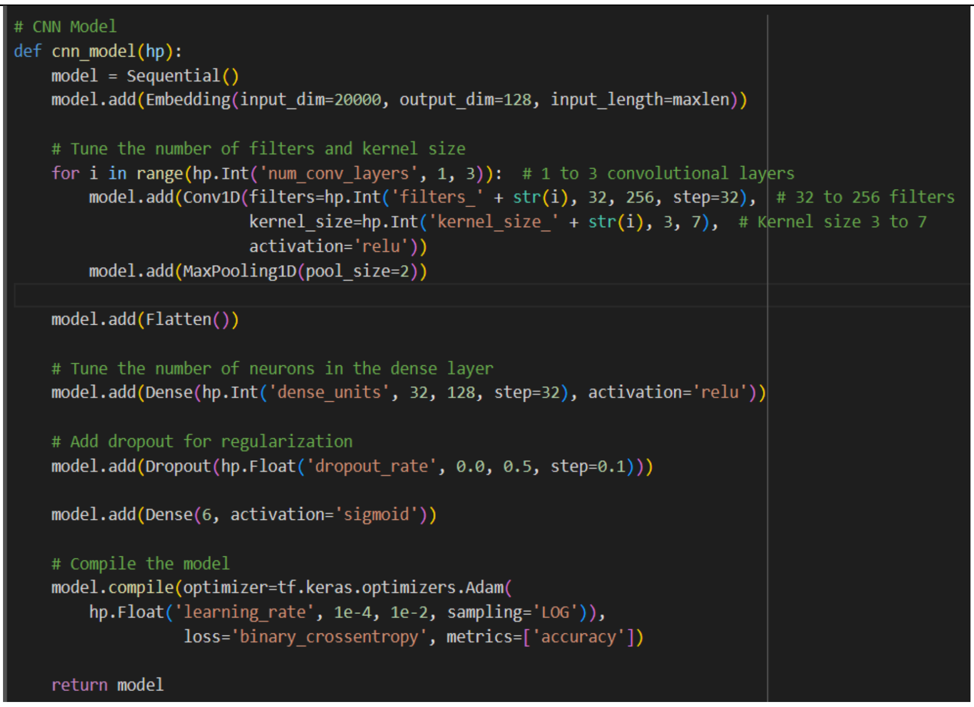
Model Optimization and Tuning Phase

The Model Optimization and Tuning Phase focuses on improving neural network performance through refined code, hyperparameter adjustments, performance metric comparisons, and rationale for selecting the best model, ensuring higher predictive accuracy and efficiency.

Hyperparameter Tuning Documentation (8 Marks):

Model	Tuned Hyperparameters
Simple Neural Network (ANN)	 **Hyperparam1**: embedding_dim

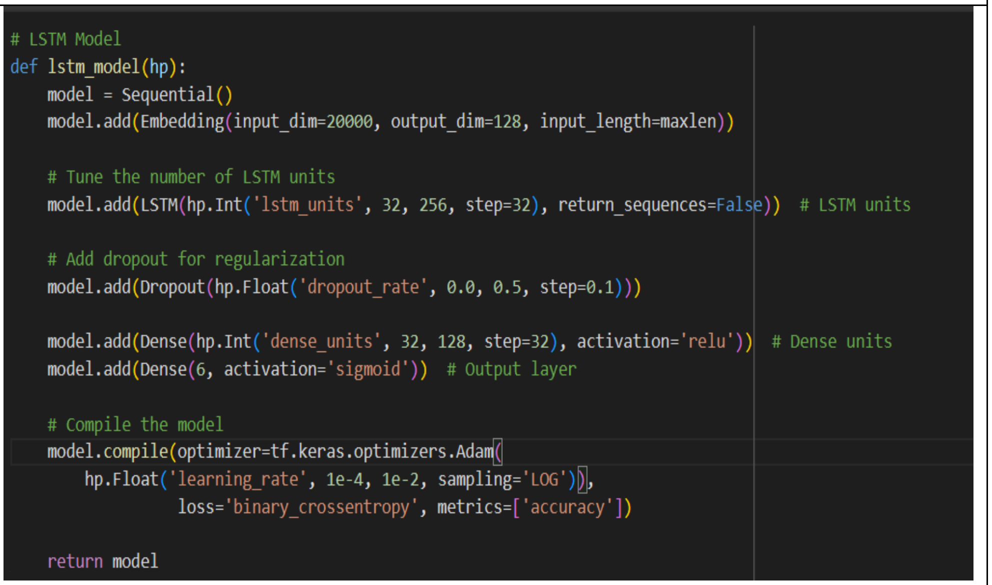
```python
#define ANN model
def ann_model(hp):
    model = Sequential()
    # Tune the embedding dimensions
    embedding_dim = hp.Int('embedding_dim', min_value=64, max_value=256, step=64)
    model.add(Embedding(input_dim=20000, output_dim=embedding_dim))

    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
```

	• Defined as part of the embedding layer tuning. • Specifies the dimensionality of the embedding space, mapping each input token to a continuous vector representation. **Hyperparam2**: dense_units • Defined as part of the Dense layer tuning. • Defines the number of neurons in the dense layer, controlling the model's capacity to learn complex patterns **Hyperparam3**: learning_rate • Set during the model compilation with the Adam optimizer. • Determines the step size at each iteration while updating model weights during training, influencing convergence speed and stability.
Convolutional Neural Network (CNN)	

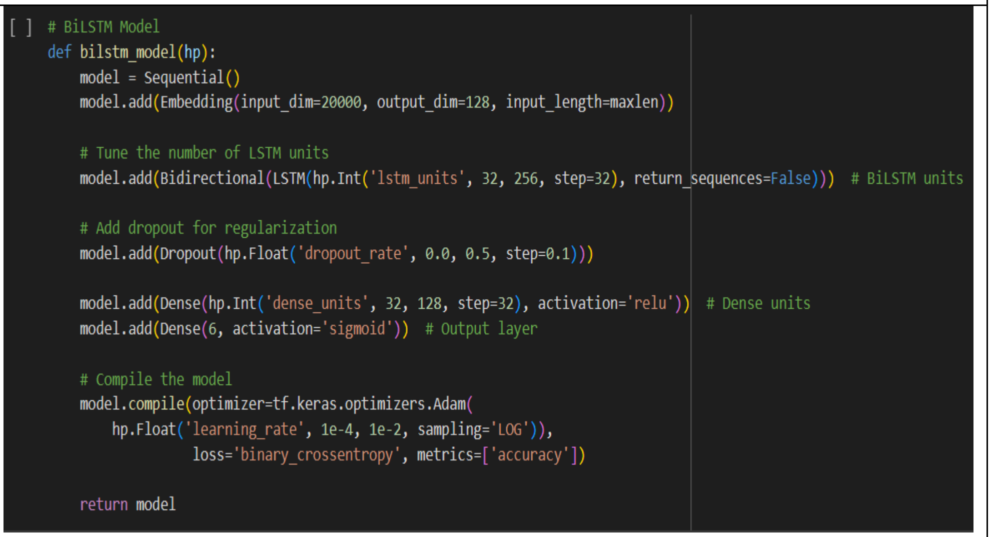
```python
# CNN Model
def cnn_model(hp):
    model = Sequential()
    model.add(Embedding(input_dim=20000, output_dim=128, input_length=maxlen))

    # 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
```

	Hyperparam1: num_conv_layers • Specifies the number of convolutional layers in the model, allowing flexibility between 1 and 3 layers to capture varying levels of features. **Hyperparam2**: filters_* • Determines the number of filters (feature detectors) in each convolutional layer, ranging from 32 to 256, controlling the model's ability to detect diverse patterns. **Hyperparam3**: kernel_size_* • Defines the size of the convolutional kernel, ranging from 3 to 7, influencing the receptive field and the granularity of features learned from the data. **Hyperparam4**: dense_unit • Sets the number of neurons in the dense layer, controlling the model's complexity and its ability to learn from the extracted features. **Hyperparam5**: dropout_rate • Defines the dropout rate for regularization, which helps prevent overfitting by randomly setting a fraction of input units to zero during training. **Hyperparam6**: learning_rate

	• Specifies the learning rate for the Adam optimizer, influencing the speed at which the model converges during training. It is tuned on a logarithmic scale between 1e-4 and 1e-2.

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`
	
	Hyperparam1: lstm_units
	• Specifies the number of units in the LSTM layer, controlling the model's ability to learn temporal dependencies in the sequence data, with values ranging from 32 to 256.
	Hyperparam2: dropout_rate
	• Defines the dropout rate for regularization, preventing overfitting by randomly setting a fraction of the input units to zero during training, with values between 0.0 and 0.5.
	Hyperparam3: dense_units

	• Sets the number of neurons in the dense layer, controlling the model’s capacity to learn complex features from the LSTM output, with values ranging from 32 to 128. **Hyperparam4**: learning_rate • Specifies the learning rate for the Adam optimizer, influencing the rate at which the model updates its weights during training, with values between 1e-4 and 1e-2 on a logarithmic scale.

Bidirectional
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`
	
	Hyperparam1: lstm_units
	
	• Specifies the number of units in the LSTM layer, controlling the model's ability to capture both past and future sequence dependencies in the BiLSTM layer, with values ranging from 32 to 256.
	
	Hyperparam2: dropout_rate

	• Defines the dropout rate for regularization, preventing overfitting by randomly setting a fraction of the input units to zero during training, with values between 0.0 and 0.5. **Hyperparam3**: dense_units • Sets the number of neurons in the dense layer, controlling the model's complexity and its ability to learn from the BiLSTM output, with values ranging from 32 to 128. **Hyperparam4**: learning_rate • Specifies the learning rate for the Adam optimizer, influencing the rate at which the model updates its weights during training, with values between 1e-4 and 1e-2 on a logarithmic scale.

Final Model Selection Justification (2 Marks):

Final Model	Reasoning
Bidirectional LSTM (BiLSTM)	1. **ANN (0.9429 accuracy):** Decent accuracy but struggles with minority classes. Suitable for basic classification but lacks depth for complex patterns. 2. **CNN (0.8965 accuracy):** High general accuracy but low precision and recall on rare labels like *threat* and *identity_hate*. Good for overall accuracy but lacks nuance. 3. **LSTM (0.9855 accuracy):** High recall on common labels with balanced overall performance. Works well for broad coverage but misses finer details in challenging labels.

	BiLSTM offers the most balanced approach, capturing minority classes effectively while maintaining high accuracy and recall. Its ability to handle challenging categories makes it ideal for a toxic comment classification system requiring both general and specific class performance**.**