REGULARIZATION ON IRIS DATASET USING KERAS

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.regularizers import l1, l2, l1_l2
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
```

Step 1: Load and Preprocess the Iris Dataset

The Iris dataset has 4 features and 3 classes. Preprocessing involves scaling features and one-hot encoding the labels for categorical cross-entropy loss

```
# Load the dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split into training and testing data (use a validation set for
early stopping)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train,
y_train, test_size=0.2, random_state=42) # 20% of training data
for validation

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# One-hot encode the labels
encoder = OneHotEncoder(sparse_output=False)
y_train_encoded = encoder.fit_transform(y_train.reshape(-1, 1))
y_val_encoded = encoder.transform(y_val.reshape(-1, 1))
y_test_encoded = encoder.transform(y_test.reshape(-1, 1))

input_dim = X_train_scaled.shape[1]
output_dim = y_train_encoded.shape[1]
```

Step 2: Build and Train a Baseline Model (No Regularization)
Establish a baseline model to compare against the regularized models

```
def build_baseline_model():
    model = Sequential()
    model.add(Dense(10, input_dim=input_dim, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(output_dim, activation='softmax'))
    model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Train the baseline model
baseline_model = build_baseline_model()
history_baseline = baseline_model.fit(X_train_scaled,
y_train_encoded,

validation_data=(X_val_scaled, y_val_encoded),
```

```
                                        epochs=100, batch_size=10,
verbose=0)
```

Step 3: Implement L1 Regularization
Apply L1 regularization to the kernel (weights) of the dense layers

```
def build_l1_model(reg_factor=0.01):
    model = Sequential()
    model.add(Dense(10, input_dim=input_dim, activation='relu',
kernel_regularizer=l1(reg_factor)))
    model.add(Dense(10, activation='relu',
kernel_regularizer=l1(reg_factor)))
    model.add(Dense(output_dim, activation='softmax'))
    model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
    return model


# Train the L1 model
l1_model = build_l1_model()
history_l1 = l1_model.fit(X_train_scaled, y_train_encoded,
                          validation_data=(X_val_scaled,
y_val_encoded),
                          epochs=100, batch_size=10, verbose=0)
```

Step 4: Implement L2 Regularization
Apply L2 regularization (also known as weight decay) to the kernel of the dense layers

```
def build_l2_model(reg_factor=0.01):
    model = Sequential()
    model.add(Dense(10, input_dim=input_dim, activation='relu',
kernel_regularizer=l2(reg_factor)))
```

```
    model.add(Dense(10, activation='relu',
kernel_regularizer=l2(reg_factor)))
    model.add(Dense(output_dim, activation='softmax'))
    model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
    return model


# Train the L2 model
l2_model = build_l2_model()
history_l2 = l2_model.fit(X_train_scaled, y_train_encoded,
                          validation_data=(X_val_scaled,
y_val_encoded),
                          epochs=100, batch_size=10, verbose=0)
```

Step 5: Implement Dropout Regularization

Add Dropout layers between existing layers to randomly drop neurons during training

```
def build_dropout_model(rate=0.3):
    model = Sequential()
    model.add(Dense(10, input_dim=input_dim, activation='relu'))
    model.add(Dropout(rate)) # Dropout layer
    model.add(Dense(10, activation='relu'))
    model.add(Dropout(rate)) # Dropout layer
    model.add(Dense(output_dim, activation='softmax'))
    model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
    return model


# Train the Dropout model
dropout_model = build_dropout_model()
```

```
history_dropout = dropout_model.fit(X_train_scaled,
y_train_encoded,

validation_data=(X_val_scaled, y_val_encoded),
                                        epochs=100, batch_size=10,
verbose=0)
```

Step 6: Implement Early Stopping

Use the EarlyStopping callback to halt training when the validation loss stops improving to prevent overfitting

```
def build_es_model():
    model = Sequential()
    model.add(Dense(10, input_dim=input_dim, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(output_dim, activation='softmax'))
    model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
    return model


# Define the Early Stopping callback
# Monitor 'val_loss' and stop if no improvement for 'patience'
epochs
early_stopping_callback = EarlyStopping(monitor='val_loss',
patience=5, restore_best_weights=True)


# Train the model with Early Stopping
es_model = build_es_model()
history_es = es_model.fit(X_train_scaled, y_train_encoded,
                          validation_data=(X_val_scaled,
y_val_encoded),
```

```
                          epochs=100, batch_size=10,
callbacks=[early_stopping_callback], verbose=0)
```

Step 7: Analyze and Compare Results
Evaluate all models on the test set and visualize the training/validation loss to observe the effects of regularization

```
# Evaluate models

print("Baseline Model Test Accuracy:",
baseline_model.evaluate(X_test_scaled, y_test_encoded,
verbose=0)[1])

print("L1 Model Test Accuracy:",
l1_model.evaluate(X_test_scaled, y_test_encoded, verbose=0)[1])

print("L2 Model Test Accuracy:",
l2_model.evaluate(X_test_scaled, y_test_encoded, verbose=0)[1])

print("Dropout Model Test Accuracy:",
dropout_model.evaluate(X_test_scaled, y_test_encoded,
verbose=0)[1])

print("Early Stopping Model Test Accuracy:",
es_model.evaluate(X_test_scaled, y_test_encoded, verbose=0)[1])


# Plot training and validation loss for comparison (example for
baseline)

plt.figure(figsize=(10, 5))

plt.plot(history_baseline.history['loss'], label='Baseline
Training Loss')

plt.plot(history_baseline.history['val_loss'], label='Baseline
Validation Loss')

plt.title('Model Loss Comparison')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend()

plt.show()
```