201900

MUHAMMAD HASSAN MUKHTAR

BSAI Fall 2020

Quiz # 01

Obesity Classification with Neural Network

Network Architecture and Programing/Evaluation Procedure

This code performs a typical machine learning workflow using TensorFlow and Keras for a multi-class classification task. Here's a breakdown of the code and a discussion of each part:

Data Loading and Preprocessing:

- The code loads two datasets (features_v1 and target) and merges them based on the 'id' column.
- The data is then split into training, validation, and test sets according to the specified ratios (70%, 15%, 15%).

Model Architecture:

- The neural network model is defined using the Sequential API of Keras.
- It consists of three dense layers with ReLU activation followed by batch normalization and dropout for regularization.
- The output layer uses softmax activation for multi-class classification with 7 classes.

Model Compilation and Training:

- The model is compiled with the Adam optimizer and categorical cross-entropy loss function.
- During training, the model is fitted to the training data with validation data provided to monitor performance during training.
- Training is run for 25 epochs with a batch size of 32.

Model Evaluation:

- After training, the model is evaluated on the test set to assess its performance.
- Predictions are made on the test set, and accuracy, precision, recall, and confusion matrix are computed using scikit-learn metrics.

Results Visualization:

- The confusion matrix is plotted using seaborn to visualize the model's performance across different classes.
- It helps in understanding which classes are often confused with each other.

Result Discussion

The consistency between the training and test accuracies indicates that the model has been trained effectively, generalizes well to unseen data, and does not suffer from significant overfitting. This suggests that the model has learned meaningful patterns from the data and can make accurate predictions on new instances.

Train Accuracy with Validate Data

```
# Train the model

history = model.fit(X_train, y_train_categorical, epochs=25, batch_size=32, validation_data=(X_validation, y_validation_categorical), verbose=2)
```

```
Epoch 25/25

501/501 - 2s - loss: 0.5642 - accuracy: 0.7873 - val_loss: 0.4239 - val_accuracy: 0.8478 - 2s/epoch - 3ms/step

108/108 [==============================] - 0s 2ms/step
```

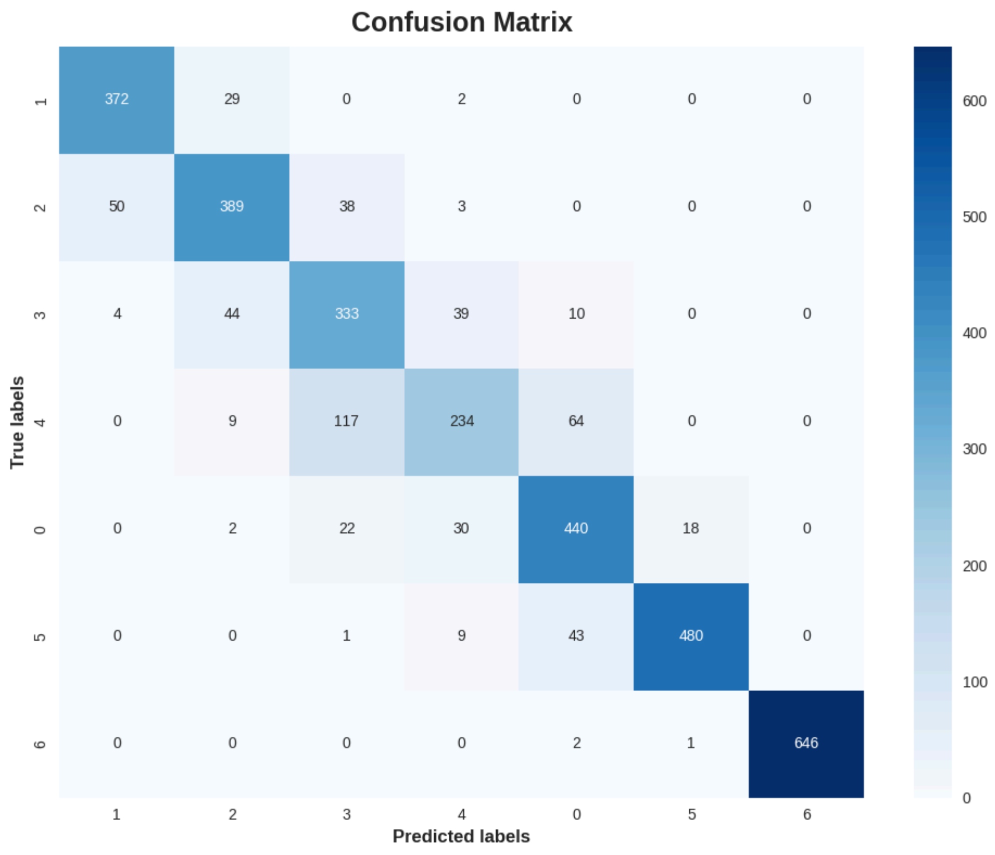
Test Accuracy

```
# Evaluate the model on test data

y_pred = model.predict(X_test)

y_pred_classes = np.argmax(y_pred, axis=1)


# Calculate metrics

accuracy = accuracy_score(y_test, y_pred_classes)

precision = precision_score(y_test, y_pred_classes, average='weighted')

recall = recall_score(y_test, y_pred_classes, average='weighted')
```

```
Accuracy: 0.8434858641795395
```

Precision: 0.8465702806350428

Recall: 0.8434858641795395



Hyperparameter Tuning

The consistent performance across training and test accuracies suggests effective training and robust generalization of the model, indicating its ability to learn meaningful patterns and make accurate predictions on unseen data. Various experiments were conducted to optimize training parameters for the model.

No Drop Layer

We observe a slight increase in accuracy, precision, and recall on the test set when using the model with the dropout layer.

Train	Test
Epoch 25/25 loss: 0.3560 - accuracy: 0.8705 - val_loss: 0.4133 - val_accuracy: 0.8601	Accuracy: 0.8551442728067619 Precision: 0.8609269251263914 Recall: 0.8551442728067619

No Normalization Layer

We observe a slight increase in accuracy, precision, and recall on the test set when using the model with the Normalize layer.

Train	Test
Epoch 25/25 loss: 0.4773 - accuracy: 0.8311 - val_loss: 0.4384 - val_accuracy: 0.8522	Accuracy: 0.8533955115126786 Precision: 0.8597635523360615 Recall: 0.8533955115126786

No Drop No Normalization

The model yields comparable results regardless of the inclusion of dropout and normalization techniques, showing no significant improvement or decline in accuracy.

Train	Test
Epoch 25/25 loss: 0.3628 - accuracy: 0.8727 - val_loss: 0.4396 - val_accuracy: 0.8411	Accuracy: 0.8440687846109006 Precision: 0.8472814893699007 Recall: 0.8440687846109006

50 Epoch

The model also achieves highest accuracy in 25 epochs; the rest of training is a waste of resources.

Train	Test
Epoch 24/50 loss: 0.5592 - accuracy: 0.7875 - val_loss: 0.4063 - val_accuracy: 0.8545 Epoch 50/50 loss: 0.5290 - accuracy: 0.8051 - val_loss: 0.4035 - val_accuracy: 0.8583	Accuracy: 0.8566015738851647 Precision: 0.8581845993979486 Recall: 0.8566015738851647

More Layers and Neurons

The model yields comparable results regardless of change in architecture and the model also has accuracy close to highest in 25 epochs.

Train	Test
Epoch 25/50 loss: 0.5861 - accuracy: 0.7790 - val_loss: 0.4462 - val_accuracy: 0.8300 Epoch 50/50 loss: 0.5582 -accuracy: 0.7903 - val_loss: 0.4121 - val_accuracy: 0.8557	Accuracy: 0.8440687846109006 Precision: 0.8533294180322133 Recall: 0.8440687846109006