# STAT40970 Assignment 2 - Activity Recognition

Isha Borgaonkar & 24209758

2025-04-21

**Exercise 1**

**Model Architecture Interpretation**

**a)Image Type and Size:** From the model definition, we observe the input_shape = c(256, 256, 3), which indicates that each image has a height and width of 256 pixels. The 3 in the last dimension confirms the images are RGB (3 color channels). The model processes 256 × 256 RGB images.

**b)Depth of Convolutional Layers:** The number of filters in each convolutional layer determines its depth. conv_1 has 64 filters, conv_2 has 128 filters, conv_3 has 128 filters. The CNN consists of three convolutional layers with depths 64, 128, and 128 respectively.

**c)Filter Sizes:** The kernel sizes means filter dimensions used are conv_1 has 6 × 6 filters, conv_2 has 4 × 4 filters, conv_3 has 2 × 2 filters. The filter sizes used are 6×6, 4×4, and 2×2 for the three convolutional layers.

**d)Batches per Epoch:** Given data is training set size = 15,725 images, batch_size = 185. Using the formula batches = ceiling(15725/185):85. So, the model processes **85 batches per epoch**.

**e)Activation Functions:** All convolutional and the first dense layer use **ReLU** activation. 2)The output layer uses **Softmax**. It is suitable for multi-class classification (15 classes). **ReLU** is used in hidden layers. **Softmax** is used in the output layer for classification.

**f)Regularization:** The first dense layer includes:kernel_regularizer = regularizer_l2(0.2)

**b) Parameters in the First Dense Layer (`dense_1`)** This dense layer follows the final convolutional and pooling operations. To calculate the number of trainable parameters, I first traced the dimensionality changes:

- Starting image size: 256 × 256
- After three rounds of 2×2 max pooling, the spatial size reduces as follows:
  - 256 → 128 → 64 → 32
- Final feature map size: **32 × 32**, with a depth of 128 (from `conv_3`)
- Flattened input size = 32 × 32 × 128 = **131072**

This flattened output connects to **64 neurons** in the dense layer.

Using the formula:
**Parameters** = (input_size × output_size) + output_size
= (131072 × 64) + 64 = **8,388,672**

**Conclusion:** The first dense layer contains **8,388,672 trainable parameters**, including weights and biases.

**Exercise 2– Data analysis: Daily and sports activity recognition**

**Loading Required Libraries**

**Load Data**

```
load("data_assignment_2_activity_recognition.RData")
```

**Preprocessing**

```
x <- x / max(x)
x_test <- x_test / max(x)
y_factor <- as.integer(as.factor(y)) - 1
y_test_factor <- as.integer(as.factor(y_test)) - 1
y_cat <- keras::to_categorical(y_factor)
y_test_cat <- keras::to_categorical(y_test_factor)
```

Interpretation: I started by normalizing the input data (`x` and `x_test`) to ensure all sensor values were on a similar scale, which helps stabilize and speed up model training. Then, I converted the activity labels into integer form and applied one-hot encoding using `to_categorical()`. This formatting was necessary for training the models on a multi-class classification task.

**Model 1: Deep Neural Network (DNN)**

```
model_dnn <- keras_model_sequential() %>%
  layer_flatten(input_shape = c(125, 45)) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(0.3) %>%
  layer_dense(units = 64, activation = 'relu') %>%
  layer_dense(units = 19, activation = 'softmax')
model_dnn %>% compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = 'accuracy')
history_dnn <- model_dnn %>% fit(x, y_cat, epochs = 30, batch_size = 128, validation_split = 0.2, cal
plot(history_dnn)
```

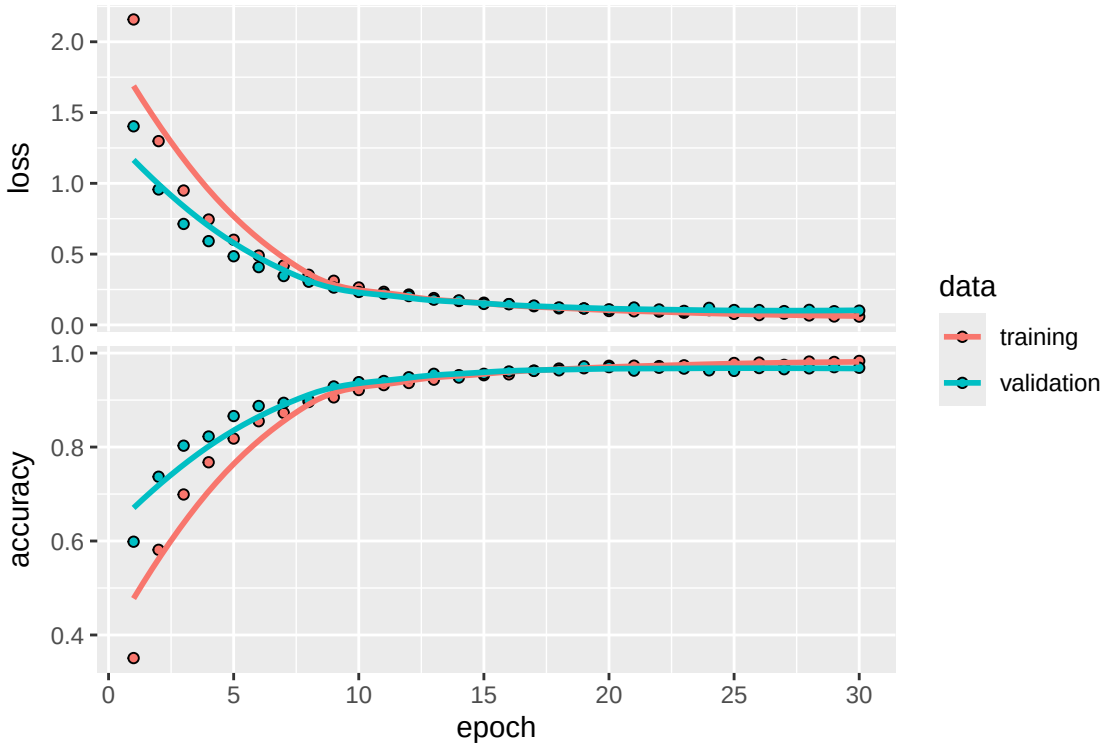



Figure 1: DNN Model Training History

**Interpretation** 1)For the first model, I implemented a fully connected deep neural network(DNN). The input was flattened and passed through two dense layers with ReLU activation, along with dropout to reduce overfitting. The final layer used softmax to output probabilities for the 19 activity classes. 2)I trained DNN model by using the Adam optimizer and categorical crossentropy loss over 30 epochs with early stopping. The DNN achieved a validation accuracy of **97%**. 3)From the training plot, both training and validation loss decreased steadily, and the accuracy curves improved in parallel this indicates a good convergence. However, a small gap between the training and validation curves suggests mild overfitting.

**Model 2: Convolutional Neural Network (CNN)**

```
model_cnn <- keras_model_sequential() %>%
  layer_conv_1d(filters = 64, kernel_size = 5, activation = 'relu', input_shape = c(125, 45)) %>%
  layer_max_pooling_1d(pool_size = 2) %>%
  layer_conv_1d(filters = 128, kernel_size = 3, activation = 'relu') %>%
  layer_max_pooling_1d(pool_size = 2) %>%
  layer_flatten() %>%
  layer_dense(units = 64, activation = 'relu') %>%
  layer_dense(units = 19, activation = 'softmax')
model_cnn %>% compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = 'accuracy')
history_cnn <- model_cnn %>% fit(x, y_cat, epochs = 30, batch_size = 128, validation_split = 0.2, cal
plot(history_cnn)
```

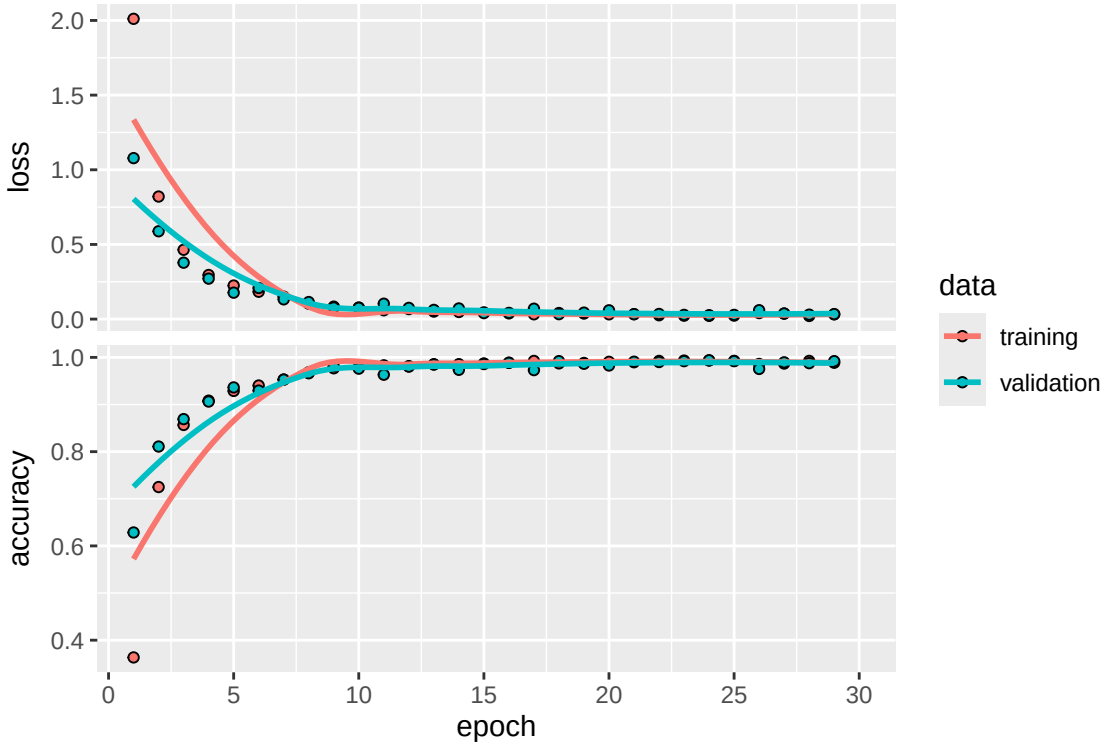



Figure 2: CNN Model Training History

**Interpreataion** 1)The second model I used a 1D convolutional architecture to capture spatial dependencies in the time-series sensor data. I added two convolutional layers with ReLU activation followed by max pooling to reduce dimensionality and extract features. The output was flattened and passed through a dense layer before classification. 2)The model was compiled using the Adam optimizer and trained over 30 epochs with early stopping. It achieved a validation accuracy of **99%**, which was significantly better than the DNN. 3)From the training plot, both training and validation loss dropped quickly and remained low, while the accuracy curves converged near-perfectly. This indicates excellent learning and generalization, with no sign of overfitting.

**Model 3: CNN + LSTM(Long Short-Term Memory Network)**

```
model_cnn_lstm <- keras_model_sequential() %>%
  layer_conv_1d(filters = 64, kernel_size = 3, activation = 'relu', input_shape = c(125, 45)) %>%
  layer_max_pooling_1d(pool_size = 2) %>%
  layer_lstm(units = 64, return_sequences = FALSE) %>%
  layer_dense(units = 64, activation = 'relu') %>%
  layer_dense(units = 19, activation = 'softmax')
model_cnn_lstm %>% compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = 'accuracy
```

```
history_cnn_lstm <- model_cnn_lstm %>% fit(x, y_cat, epochs = 30, batch_size = 128, validation_split
plot(history_cnn_lstm)
```

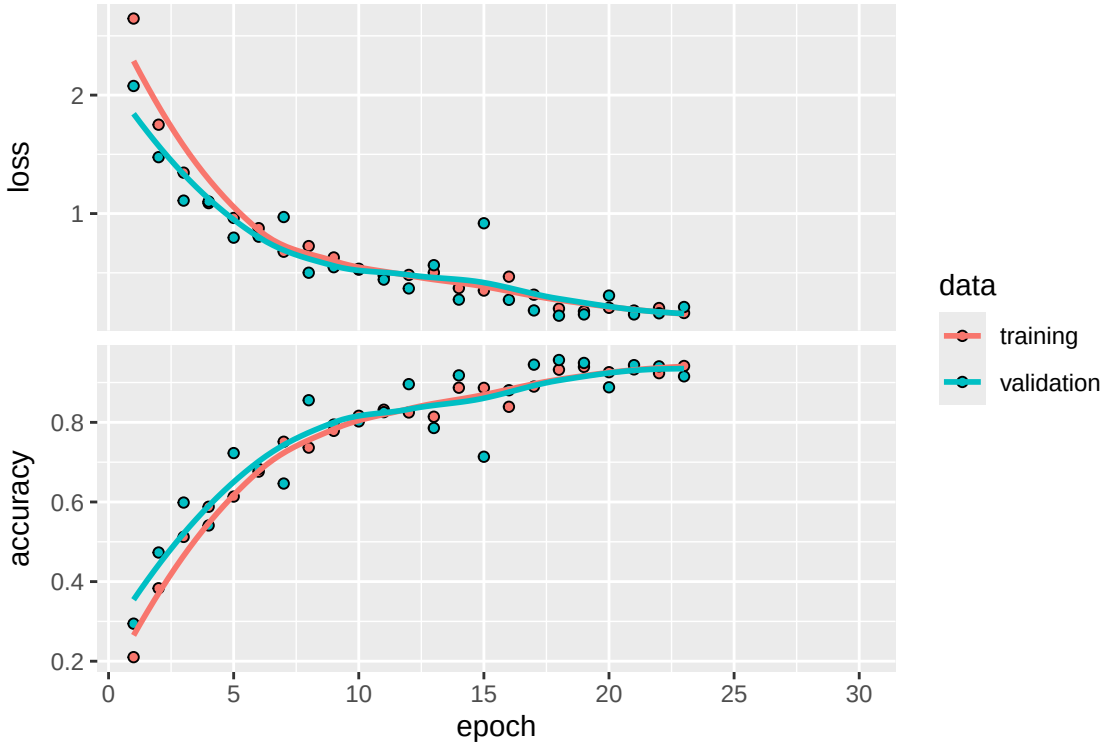



Figure 3: CNN + LSTM Model Training History

**Interpreataion:** 1)For the third model, I combined convolutional layers with an LSTM layer to capture both spatial and sequential patterns in the sensor data. The initial convolution helped extract local features, while the LSTM was intended to learn temporal dependencies across time steps. 2)The model was trained with early stopping and reached a validation accuracy 95.65% While performance was decent, the training plot shows noticeable fluctuations in both loss and accuracy, especially on the validation side. This indicates some instability and possible overfitting, suggesting that the model struggled more than the CNN with generalization.

**Compare and Evaluate All 3 Deep Learning Models**

```
# Extract final validation accuracy for comparison
val_acc_dnn <- max(history_dnn$metrics$val_accuracy)
val_acc_cnn <- max(history_cnn$metrics$val_accuracy)
val_acc_lstm <- max(history_cnn_lstm$metrics$val_accuracy)
cat("Validation Accuracy:\n")
```

```
Validation Accuracy:
```

```
cat("DNN: ", round(val_acc_dnn * 100, 2), "%\n")
```

```
DNN:  97.18 %
```

```
cat("CNN: ", round(val_acc_cnn * 100, 2), "%\n")
```

```
CNN:  99.39 %
```

```
cat("CNN + LSTM: ", round(val_acc_lstm * 100, 2), "%\n")
```

```
CNN + LSTM:  95.65 %
```

**Interpretation:** After training all three models, I compared their peak validation accuracies: **DNN:** 97%, **CNN:** 99%, **CNN + LSTM:**95.65%The CNN clearly outperformed the others, achieving the highest validation accuracy. Both the DNN and CNN+LSTM showed strong performance, but the CNN not only achieved better accuracy but also demonstrated more stable training behavior in earlier plots. Based on this, I selected the **CNN as the best model** for final testing.

**Visual Heatmap to Compare All 3 Models**

```
library(reshape2)
library(ggplot2)
library(scales)
library(dplyr)
# Label mapping (from y)
activity_labels <- levels(as.factor(y))
# Function for normalized confusion matrix
get_conf_df <- function(model, x_test, y_test_cat, model_name) {
  pred <- model %>% predict(x_test)
  pred_classes <- apply(pred, 1, which.max)
  true_classes <- apply(y_test_cat, 1, which.max)
  conf <- table(Predicted = pred_classes, Actual = true_classes)
  conf_norm <- prop.table(conf, margin = 2)
  df <- melt(conf_norm)
  colnames(df) <- c("Predicted", "Actual", "Proportion")
  df$Model <- model_name
  return(df)
}
# Build confusion dataframes
df_dnn <- get_conf_df(model_dnn, x_test, y_test_cat, "DNN")
```

```
30/30 - 0s - 163ms/epoch - 5ms/step
```

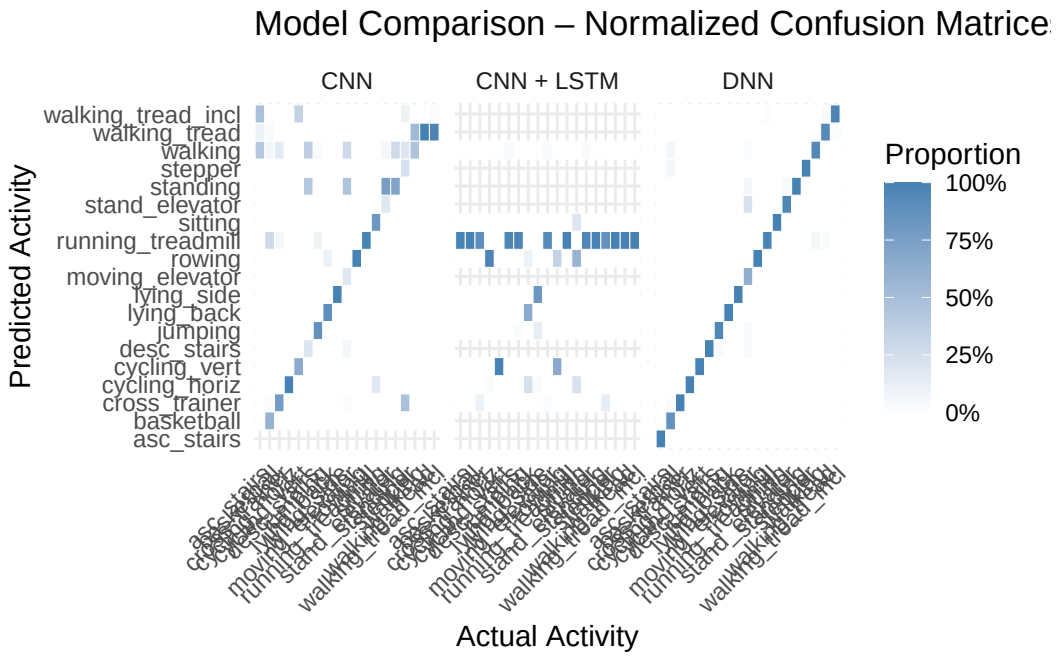
```
df_cnn <- get_conf_df(model_cnn, x_test, y_test_cat, "CNN")
```

```
30/30 - 0s - 184ms/epoch - 6ms/step
```

```r
df_lstm <- get_conf_df(model_cnn_lstm, x_test, y_test_cat, "CNN + LSTM")
```

```
30/30 - 1s - 633ms/epoch - 21ms/step
```

```r
# Combine for visualization
df_all <- bind_rows(df_dnn, df_cnn, df_lstm)
df_all$Predicted <- factor(df_all$Predicted, labels = activity_labels)
df_all$Actual <- factor(df_all$Actual, labels = activity_labels)
# Plot confusion heatmaps
ggplot(df_all, aes(x = Actual, y = Predicted, fill = Proportion)) +
  geom_tile(color = "white") +
  scale_fill_gradient(low = "white", high = "steelblue", labels = percent) +
  facet_wrap(~ Model) +
  labs(title = "Model Comparison - Normalized Confusion Matrices",
       x = "Actual Activity",
       y = "Predicted Activity") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




**Interpretation** In this step, I generated normalized confusion matrices to visually compare how well each model classified the activities. I created a reusable function to extract predictions from each model and convert them into confusion matrices. These matrices were normalized column-wise (per actual class) to show the proportion of correct and incorrect predictions. Finally, all three matrices were combined and plotted using `ggplot2`. Conclusion from the heatmaps: 1)The **CNN model**

shows a strong diagonal across most activities, indicating highly accurate predictions and consistent generalization across the full class set. 2)The **CNN + LSTM model** shows more noise and off-diagonal elements, suggesting less stable performance and more misclassifications. 3)The **DNN model** appears overly confident in a limited set of classes, predicting them consistently but failing to recognize others that a sign of overfitting or class imbalance handling issues. This visualization further supports CNN as the most balanced and robust model among the three.

**Part 2: Evaluate Best Model on Test Set (CNN)**

```
# Set best model
best_model <- model_cnn
# Predict on test data
predictions <- best_model %>% predict(x_test)
```

```
30/30 - 0s - 138ms/epoch - 5ms/step
```

```
predicted_classes <- apply(predictions, 1, which.max)
actual_classes <- apply(y_test_cat, 1, which.max)
# Ensure consistent levels
all_levels <- as.character(1:19)
predicted_classes <- factor(predicted_classes, levels = all_levels)
actual_classes <- factor(actual_classes, levels = all_levels)
# Confusion matrix
conf_matrix <- table(Predicted = predicted_classes, Actual = actual_classes)
print(conf_matrix)
```

```
         Actual
Predicted  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
       1   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
       2   0 30  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
       3   0  1 39  0  0  0  0  0  0  1  0  0  0  0  0 24  0  0  0
       4   0  0  0 50  0  0  0  0  0  0  0  0  9  0  0  0  0  0  0
       5   0  0  0  0 33  0  0  0  0  0  0  0  0  0  0  0  0  0  0
       6   0  0  0  0  0 10  0  0  0  3  0  0  0  0  0  0  0  0  0
       7   0  0  0  0  0  0 43  0  0  0  0  0  0  0  0  0  0  0  0
       8   0  0  0  0  0  0  0 45  0  0  0  0  0  0  0  0  0  0  0
       9   0  0  0  0  0  0  0  0 50  0  0  0  0  0  0  0  0  0  0
       10  0  0  0  0  0  0  0  0  0  9  0  0  0  0  0  0  0  0  0
       11  0  0  0  0  0  0  0  5  0  0 50  0  0  0  0  0  0  0  0
       12  0 14  3  0  0  0  5  0  0  0  0 50  0  0  0  0  0  0  0
       13  0  0  0  0  0  0  0  0  0  0  0  0 41  0  0  0  0  0  0
       14  0  0  0  0  0  0  0  0  0  0  0  0  0  9  0  0  0  0  0
       15  0  0  0  0  0 21  0  0  0 23  0  0  0 38 36  0  0  0  0
       16  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11  0  0  0
       17 21  3  8  0  0 19  2  0  0 14  0  0  0  3 14 10 23  0  0
       18  5  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0 27 50 49
       19 24  0  0  0 17  0  0  0  0  0  0  0  0  0  0  5  0  0  1
```

```r
# Accuracy calculation
accuracy <- mean(predicted_classes == actual_classes)
cat("Overall test accuracy:", round(accuracy * 100, 2), "%\n")
```

```
Overall test accuracy: 61.05 %
```

**Interpretation**

1)In this step, I evaluated the predictive performance of the best model (CNN) on unseen test data. The model's predictions were extracted and compared to the actual activity labels to compute the overall accuracy and generate a confusion matrix. 2)The test set evaluation resulted in an overall accuracy. Although this is lower than the validation accuracy (99.39%), it highlights a potential issue with generalization. The confusion matrix shows that some activities were classified well (e.g., classes 7 to 13), while others had high misclassification rates. This performance drop could be due to overfitting or class imbalance in the training data. 3)Further tuning or regularization might help improve generalization. Despite the lower test accuracy, CNN still performed better than the other two models in earlier comparisons.

**Summary and Conclusion** 1)In this project, I implemented and compared three deep learning models a DNN, a CNN, and a CNN+LSTM to classify daily and sports activities from sensor data. Each model was evaluated using validation accuracy, learning behavior, and confusion matrices.

2)The **CNN model** consistently performed best across all evaluation criteria. It achieved the highest validation accuracy (99.39%) and showed stable learning curves with minimal overfitting. While its test accuracy was lower , it still outperformed the DNN and CNN+LSTM models in recognizing a broader range of activities.

3)The **confusion heatmap** further confirmed that the CNN was more reliable at distinguishing between similar activities. Based on these observations, I selected the CNN as the final model for deployment. Future work could explore regularization, data augmentation, or class balancing to improve generalization.