AML_ASSIGNMENT 1

ROHIT VURADI

2024-02-22

```
library(keras)
library(tensorflow)
library(ggplot2)
```

```
# Load the IMDB dataset
imdb_data <- dataset_imdb(num_words = 10000)
train_data <- imdb_data$train$x
train_labels <- imdb_data$train$y
test_data <- imdb_data$test$x
test_labels <- imdb_data$test$y
```

```
# display the data
str(train_data)
```

```
## List of 25000
##  $ : int [1:218] 1 14 22 16 43 530 973 1622 1385 65 ...
##  $ : int [1:189] 1 194 1153 194 8255 78 228 5 6 1463 ...
##  $ : int [1:141] 1 14 47 8 30 31 7 4 249 108 ...
##  $ : int [1:550] 1 4 2 2 33 2804 4 2040 432 111 ...
##  $ : int [1:147] 1 249 1323 7 61 113 10 10 13 1637 ...
##  $ : int [1:43] 1 778 128 74 12 630 163 15 4 1766 ...
##  $ : int [1:123] 1 6740 365 1234 5 1156 354 11 14 5327 ...
##  $ : int [1:562] 1 4 2 716 4 65 7 4 689 4367 ...
##  $ : int [1:233] 1 43 188 46 5 566 264 51 6 530 ...
##  $ : int [1:130] 1 14 20 47 111 439 3445 19 12 15 ...
##  $ : int [1:450] 1 785 189 438 47 110 142 7 6 7475 ...
##  $ : int [1:99] 1 54 13 1610 14 20 13 69 55 364 ...
##  $ : int [1:117] 1 13 119 954 189 1554 13 92 459 48 ...
##  $ : int [1:238] 1 259 37 100 169 1653 1107 11 14 418 ...
##  $ : int [1:109] 1 503 20 33 118 481 302 26 184 52 ...
##  $ : int [1:129] 1 6 964 437 7 58 43 1402 11 6 ...
##  $ : int [1:163] 1 7092 1662 11 4 1749 9 4 2165 4 ...
##  $ : int [1:752] 1 33 4 5673 7 4 2 194 2 3089 ...
##  $ : int [1:212] 1 13 28 64 69 4 8742 7 319 14 ...
##  $ : int [1:177] 1 3432 26 9 6 1220 731 939 44 6 ...
##  $ : int [1:129] 1 617 11 3875 17 2 14 966 78 20 ...
##  $ : int [1:140] 1 466 49 2036 204 2442 40 4 6724 732 ...
##  $ : int [1:256] 1 13 784 886 857 15 135 142 40 2 ...
##  $ : int [1:888] 1 4 712 19 2 2 2 963 2 26 ...
##  $ : int [1:93] 1 4 204 7610 20 16 93 11 9075 19 ...
##  $ : int [1:142] 1 14 9 6 55 641 2854 212 44 6 ...
##  $ : int [1:220] 1 4 288 310 3202 7 241 672 4 2 ...
##  $ : int [1:193] 1 75 69 8 140 8 35 2 38 75 ...
##  $ : int [1:171] 1 4679 2784 1482 11 450 7 134 364 352 ...
##  $ : int [1:221] 1 13 28 332 4 274 6 378 7 211 ...
##  $ : int [1:174] 1 13 1059 14 33 6 2181 2824 32 13 ...
##  $ : int [1:647] 1 6 565 255 5964 875 103 196 167 2 ...
##  $ : int [1:233] 1 4 86 390 1241 520 6 52 1384 51 ...
##  $ : int [1:162] 1 13 92 124 51 12 9 13 169 38 ...
##  $ : int [1:597] 1 111 28 3431 15 2 475 455 4127 9 ...
##  $ : int [1:234] 1 1255 1223 5547 1265 2390 1747 8 4 268 ...
##  $ : int [1:51] 1 806 13 43 161 169 4 875 551 17 ...
##  $ : int [1:336] 1 23 3333 7449 7505 4 254 157 2 7 ...
##  $ : int [1:139] 1 17 19 111 85 1719 1181 3135 201 14 ...
##  $ : int [1:231] 1 14 38 446 1034 9 394 13 435 8 ...
##  $ : int [1:704] 1 14 16 4 840 5582 20 5 4 236 ...
##  $ : int [1:142] 1 14 22 16 23 522 33 314 54 13 ...
##  $ : int [1:861] 1 1710 14 733 1367 1028 81 24 332 48 ...
##  $ : int [1:132] 1 4 229 18 14 248 1844 1422 9 38 ...
##  $ : int [1:122] 1 3420 9 34 230 61 514 7 32 7 ...
##  $ : int [1:570] 1 11 14 4419 3073 14 9061 50 26 1093 ...
##  $ : int [1:55] 1 568 65 9 4689 31 7 4 118 495 ...
##  $ : int [1:214] 1 806 21 13 80 2372 199 4 114 347 ...
##  $ : int [1:103] 1 54 7850 5397 8633 7455 9 2090 2 23 ...
##  $ : int [1:186] 1 13 244 1713 1681 8 97 134 243 7 ...
##  $ : int [1:113] 1 13 165 219 14 20 33 6 750 17 ...
##  $ : int [1:169] 1 159 13 296 14 22 13 332 6 733 ...
##  $ : int [1:469] 1 14 364 1242 2460 2 47 43 77 7548 ...
##  $ : int [1:138] 1 1400 168 855 19 12 50 26 76 128 ...
##  $ : int [1:302] 1 101 20 11 63 7564 2 46 1421 6 ...
##  $ : int [1:766] 1 5011 892 711 12 2774 38 2428 145 11 ...
##  $ : int [1:351] 1 1318 13 191 264 146 4 86 5 64 ...
##  $ : int [1:146] 1 13 974 13 69 8 702 930 143 14 ...
##  $ : int [1:59] 1 13 296 4 20 11 6 4435 5 13 ...
##  $ : int [1:206] 1 209 888 14 22 47 8 30 31 7 ...
##  $ : int [1:107] 1 13 219 14 33 4 2 22 1413 12 ...
##  $ : int [1:152] 1 591 92 851 42 60 104 44 2644 14 ...
##  $ : int [1:186] 1 13 258 14 20 8 30 6 87 326 ...
##  $ : int [1:431] 1 4 2019 9 149 16 35 221 22 585 ...
##  $ : int [1:147] 1 146 242 31 7 4 1126 2406 2266 451 ...
##  $ : int [1:684] 1 1810 8 516 7732 93 6 227 7 6 ...
##  $ : int [1:383] 1 1028 11 86 8213 14 1327 1210 1124 5205 ...
##  $ : int [1:324] 1 2 8810 322 7398 5 68 1667 476 2 ...
##  $ : int [1:252] 1 13 286 1017 76 5 8 30 1202 13 ...
##  $ : int [1:263] 1 48 335 6 337 7 22 1359 5 104 ...
##  $ : int [1:787] 1 6 1380 6733 3453 54 49 432 7 5682 ...
##  $ : int [1:211] 1 14 20 218 290 4 22 12 16 3551 ...
##  $ : int [1:314] 1 49 24 38 2 1028 1404 10 10 138 ...
##  $ : int [1:118] 1 480 302 18 6 20 7 14 58 6 ...
##  $ : int [1:390] 1 670 5304 1616 97 6 20 40 14 21 ...
##  $ : int [1:132] 1 1756 5663 9 2990 2 133 177 17 6 ...
##  $ : int [1:710] 1 1065 2474 7 3508 2 645 113 17 6 ...
##  $ : int [1:306] 1 17 210 14 9 35 6213 431 7 4 ...
##  $ : int [1:167] 1 2500 1040 4 327 1208 44 14 215 28 ...
##  $ : int [1:115] 1 11 4 402 3469 111 7 178 37 69 ...
##  $ : int [1:95] 1 435 8 67 14 17 72 5 61 761 ...
##  $ : int [1:158] 1 31 7 61 118 369 839 14 20 120 ...
##  $ : int [1:156] 1 66 371 2 2 373 21 284 2 2567 ...
##  $ : int [1:82] 1 4 1126 282 13 69 8 67 14 20 ...
##  $ : int [1:502] 1 14 22 7930 236 314 33 2 5510 750 ...
##  $ : int [1:314] 1 13 144 1260 138 13 520 14 418 7 ...
##  $ : int [1:190] 1 13 92 400 140 46 7 61 96 8 ...
##  $ : int [1:174] 1 61 795 203 30 6 227 7 6 1361 ...
##  $ : int [1:60] 1 18 6 20 19 6 114 40 14 13 ...
##  $ : int [1:145] 1 13 219 14 22 5236 145 137 780 23 ...
##  $ : int [1:214] 1 11 192 5192 2 125 2139 1253 7 2 ...
##  $ : int [1:659] 1 541 5156 517 19 4 4791 7 2408 827 ...
##  $ : int [1:408] 1 474 66 66 473 8 67 14 20 5 ...
##  $ : int [1:515] 1 7931 4 425 410 2568 4 876 7 4 ...
##  $ : int [1:461] 1 121 81 13 895 13 473 8 358 14 ...
##  $ : int [1:202] 1 13 66 215 28 1059 6 275 22 39 ...
##  $ : int [1:238] 1 827 2 2 10 10 1251 8598 300 2 ...
##  $ : int [1:170] 1 24 15 76 183 593 11 14 20 21 ...
##  $ : int [1:107] 1 14 20 9 389 10 10 13 16 2 ...
##   [list output truncated]
```

```r
str(train_labels)
```

```
##  int [1:25000] 1 0 0 1 0 0 1 0 1 0 ...
```

```r
str(test_data)
```

```
## List of 25000
##  $ : int [1:68] 1 591 202 14 31 6 717 10 10 2 ...
##  $ : int [1:260] 1 14 22 3443 6 176 7 5063 88 12 ...
##  $ : int [1:603] 1 111 748 4368 1133 2 2 4 87 1551 ...
##  $ : int [1:181] 1 13 1228 119 14 552 7 20 190 14 ...
##  $ : int [1:108] 1 40 49 85 84 1040 146 6 783 254 ...
##  $ : int [1:132] 1 146 427 5718 14 20 218 112 2962 32 ...
##  $ : int [1:761] 1 1822 424 8 30 43 6 173 7 6 ...
##  $ : int [1:180] 1 4 2 745 2 912 9 2 8 2 ...
##  $ : int [1:134] 1 363 69 6 196 119 1586 19 6514 2 ...
##  $ : int [1:370] 1 14 22 9 121 4 1354 3135 3882 8 ...
##  $ : int [1:209] 1 1581 34 7908 5082 23 6 1374 1120 7 ...
##  $ : int [1:248] 1 54 13 86 219 14 20 11 4 750 ...
##  $ : int [1:398] 1 449 3214 449 3214 12 66 214 23 61 ...
##  $ : int [1:326] 1 13 645 149 14 88 13 197 12 16 ...
##  $ : int [1:131] 1 6 1301 664 15 1457 6 406 393 23 ...
##  $ : int [1:255] 1 387 72 86 380 46 34 660 300 46 ...
##  $ : int [1:127] 1 39 4142 86 13 296 14 20 13 235 ...
##  $ : int [1:184] 1 1659 2 3717 9 6 2343 37 456 18 ...
##  $ : int [1:188] 1 13 16 66 2 56 8 106 54 107 ...
##  $ : int [1:105] 1 208 38 25 28 6 2 3400 7 1093 ...
##  $ : int [1:230] 1 14 22 16 31 15 13 28 4465 8 ...
##  $ : int [1:137] 1 14 9 31 7 61 1640 910 108 12 ...
##  $ : int [1:88] 1 89 1319 8 798 692 1287 6 736 6 ...
##  $ : int [1:70] 1 2127 1256 8 1686 39 109 8 109 11 ...
##  $ : int [1:170] 1 2 8415 5968 2102 2 39 4819 11 5399 ...
##  $ : int [1:305] 1 146 24 252 138 14 22 9 2 38 ...
##  $ : int [1:273] 1 13 69 332 4692 857 2 14 3720 5675 ...
##  $ : int [1:134] 1 3799 117 183 16 6 66 52 20 13 ...
##  $ : int [1:232] 1 688 8 6 55 5328 4593 5653 13 219 ...
##  $ : int [1:264] 1 4 145 7 4 288 18 14 20 2 ...
##  $ : int [1:99] 1 1756 5663 122 6 4796 292 940 14 22 ...
##  $ : int [1:133] 1 48 25 535 15 14 20 9 368 7 ...
##  $ : int [1:121] 1 48 25 28 115 332 4 356 1067 1219 ...
##  $ : int [1:521] 1 48 14 9 24 2699 2561 23 175 1029 ...
##  $ : int [1:133] 1 1018 3577 4150 3188 2 8 30 35 2 ...
##  $ : int [1:124] 1 14 9 44 31 7 4 249 102 474 ...
##  $ : int [1:228] 1 14 20 9 434 31 7 4 833 108 ...
##  $ : int [1:159] 1 2 2688 17 644 961 46 160 480 65 ...
##  $ : int [1:132] 1 449 558 12 100 30 6 55 221 22 ...
##  $ : int [1:141] 1 14 664 16 357 5 179 379 10 10 ...
##  $ : int [1:310] 1 11 2 6 185 132 584 11 6 686 ...
##  $ : int [1:431] 1 117 2 738 8 2223 9 160 11 4 ...
##  $ : int [1:214] 1 260 77 6 4279 337 18 111 153 6662 ...
##  $ : int [1:185] 1 13 219 14 5126 20 88 13 244 6 ...
##  $ : int [1:549] 1 2285 1447 2435 2 2365 2018 18 692 19 ...
##  $ : int [1:136] 1 4 752 7486 16 398 34 72 54 13 ...
##  $ : int [1:89] 1 23 2195 6 513 9 2 34 6 7472 ...
##  $ : int [1:298] 1 313 7 4 6333 82 573 17 2 9 ...
##  $ : int [1:127] 1 402 2 792 448 23 6 1063 868 297 ...
##  $ : int [1:252] 1 13 191 391 138 899 6320 62 967 14 ...
##  $ : int [1:289] 1 86 7 13 144 213 46 15 13 343 ...
##  $ : int [1:125] 1 13 43 332 35 1727 196 733 23 4 ...
##  $ : int [1:64] 1 13 69 210 473 8 67 14 22 5 ...
##  $ : int [1:111] 1 1469 2102 5 2163 26 6 378 7 4 ...
##  $ : int [1:132] 1 12 299 40 129 644 1667 311 830 6 ...
##  $ : int [1:256] 1 14 9 6 318 22 48 25 124 4 ...
##  $ : int [1:100] 1 13 16 55 685 54 14 123 16 6769 ...
##  $ : int [1:152] 1 48 25 447 4 5040 2475 920 1219 6537 ...
##  $ : int [1:203] 1 11 4618 689 2 2 34 2643 745 2 ...
##  $ : int [1:124] 1 449 89 4916 14 20 9 13 92 124 ...
##  $ : int [1:113] 1 13 66 423 4 20 1065 162 2 21 ...
##  $ : int [1:208] 1 4 969 80 168 55 1081 8 25 38 ...
##  $ : int [1:106] 1 43 191 106 14 227 99 111 211 45 ...
##  $ : int [1:234] 1 66 133 4 785 438 1936 741 1324 5 ...
##  $ : int [1:114] 1 146 770 8 332 32 4 1123 795 23 ...
##  $ : int [1:210] 1 1318 45 77 6 196 58 237 207 236 ...
##  $ : int [1:733] 1 14 20 93 72 104 7 89 13 100 ...
##  $ : int [1:173] 1 61 336 4 632 9 1047 163 5 1036 ...
##  $ : int [1:321] 1 39 4 86 8 4 236 136 7 4 ...
##  $ : int [1:86] 1 14 20 203 2 977 8 30 2 5 ...
##  $ : int [1:112] 1 1318 13 447 14 20 14 22 16 1061 ...
##  $ : int [1:195] 1 14 22 188 329 692 74 2756 7 68 ...
##  $ : int [1:101] 1 2676 9 4 243 7 20 36 93 11 ...
##  $ : int [1:194] 1 14 20 9 24 290 4 58 12 304 ...
##  $ : int [1:122] 1 14 20 16 608 17 230 17 102 140 ...
##  $ : int [1:324] 1 4 2474 7 2 47 8 30 31 7 ...
##  $ : int [1:195] 1 13 1053 4480 234 7 61 113 23 14 ...
##  $ : int [1:257] 1 50 28 77 55 171 87 212 108 11 ...
##  $ : int [1:307] 1 50 186 8 30 6 9557 7 438 23 ...
##  $ : int [1:78] 1 14 20 47 188 8 30 4 433 20 ...
##  $ : int [1:157] 1 19 4 578 1401 7 6799 3236 14 16 ...
##  $ : int [1:213] 1 2 4 248 20 2 6 55 2509 5 ...
##  $ : int [1:163] 1 33 2405 197 4 1120 7 14 123 468 ...
##  $ : int [1:122] 1 146 24 83 635 287 15 76 21 14 ...
##  $ : int [1:197] 1 10 10 132 13 43 2488 264 14 20 ...
##  $ : int [1:169] 1 13 122 6 733 18 14 2030 2615 9730 ...
##  $ : int [1:81] 1 73 474 28 8 135 15 13 81 205 ...
##  $ : int [1:242] 1 13 191 79 14 509 125 61 1224 45 ...
##  $ : int [1:218] 1 13 188 5158 8 14 2446 31 2 2655 ...
##  $ : int [1:193] 1 864 13 124 198 1591 623 23 94 2558 ...
##  $ : int [1:467] 1 1318 133 9 160 87 8894 20 198 33 ...
##  $ : int [1:74] 1 13 104 14 9 6 87 356 969 22 ...
##  $ : int [1:76] 1 114 600 229 6 3592 7 6363 116 24 ...
##  $ : int [1:736] 1 4 3768 356 20 1308 47 1084 4 3241 ...
##  $ : int [1:448] 1 2 6172 526 34 4 530 2 5893 2 ...
##  $ : int [1:200] 1 14 9 6 1332 1253 7 4 592 848 ...
##  $ : int [1:185] 1 13 296 14 20 260 115 332 4 274 ...
##  $ : int [1:124] 1 2787 7056 59 93 14 20 18 72 207 ...
##  $ : int [1:122] 1 216 23 150 89 122 32 7 134 1020 ...
##   [list output truncated]
```

```r
str(test_labels)
```

```
##  int [1:25000] 0 1 1 0 1 1 1 0 0 1 ...
```

```r
# Prepare the data
vectorize_sequences <- function(sequences, dimension = 10000) {
  results <- matrix(0, nrow = length(sequences), ncol = dimension)
  for (i in seq_along(sequences)) {
    results[i, sequences[[i]]] <- 1
  }
  results
}

x_train <- vectorize_sequences(train_data)
x_test <- vectorize_sequences(test_data)
y_train <- as.matrix(train_labels)
y_test <- as.matrix(test_labels)
```

```r
# 1. Number of Hidden Layers
# One hidden layer
model_one_hidden <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = 'relu', input_shape = c(10000)) %>%
  layer_dense(units = 1, activation = 'sigmoid')

# Three hidden layers
model_three_hidden <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = 'relu', input_shape = c(10000)) %>%
  layer_dense(units = 16, activation = 'relu') %>%
  layer_dense(units = 16, activation = 'relu') %>%
  layer_dense(units = 1, activation = 'sigmoid')
```

```r
# 2. Number of Hidden Units
# 32 hidden units
model_32_units <- keras_model_sequential() %>%
  layer_dense(units = 32, activation = 'relu', input_shape = c(10000)) %>%
  layer_dense(units = 1, activation = 'sigmoid')

# 64 hidden units
```

```r
model_64_units <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = 'relu', input_shape = c(10000)) %>%
  layer_dense(units = 1, activation = 'sigmoid')
```

```r
# 3. Loss Function
# Mean Squared Error (MSE) loss function
model_mse_loss <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = 'relu', input_shape = c(10000)) %>%
  layer_dense(units = 1, activation = 'sigmoid')

model_mse_loss %>% compile(optimizer = 'rmsprop', loss = 'mse', metrics = c('accuracy'))
```

```r
# 4. Activation Function
# tanh activation function
model_tanh_activation <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = 'tanh', input_shape = c(10000)) %>%
  layer_dense(units = 1, activation = 'sigmoid')
```

```r
# 5. Regularization and Dropout
# L2 Regularization
model_l2_regularization <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = 'relu', input_shape = c(10000), kernel_regularizer = regularizer
  layer_dense(units = 1, activation = 'sigmoid')
```

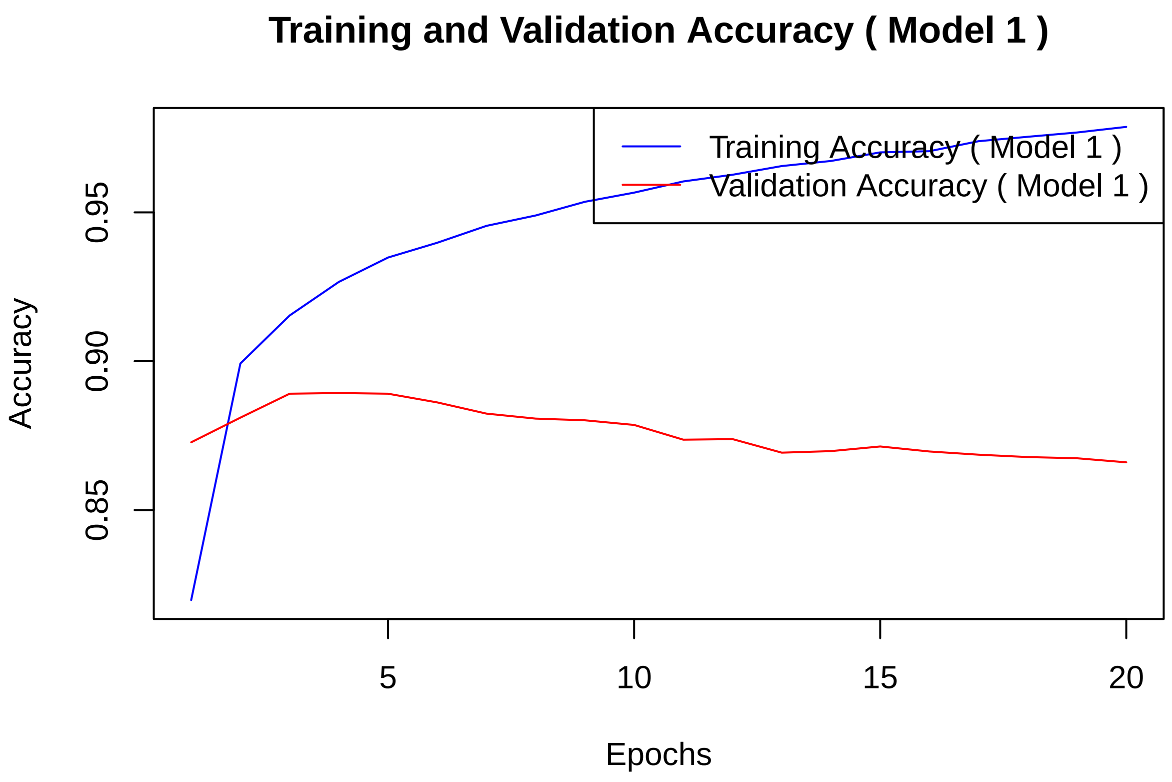
```r
# Dropout
model_dropout <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = 'relu', input_shape = c(10000)) %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 1, activation = 'sigmoid')
```

```r
# Function to create, compile, and fit the model
compile_and_fit_model <- function(model, x_train, y_train, x_val, y_val, epochs = 20, batch_size = 512)
  model %>% compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', metrics = c('accuracy'))
  history <- model %>% fit(x_train, y_train, epochs = epochs, batch_size = batch_size, validation_data =
  history
}
```

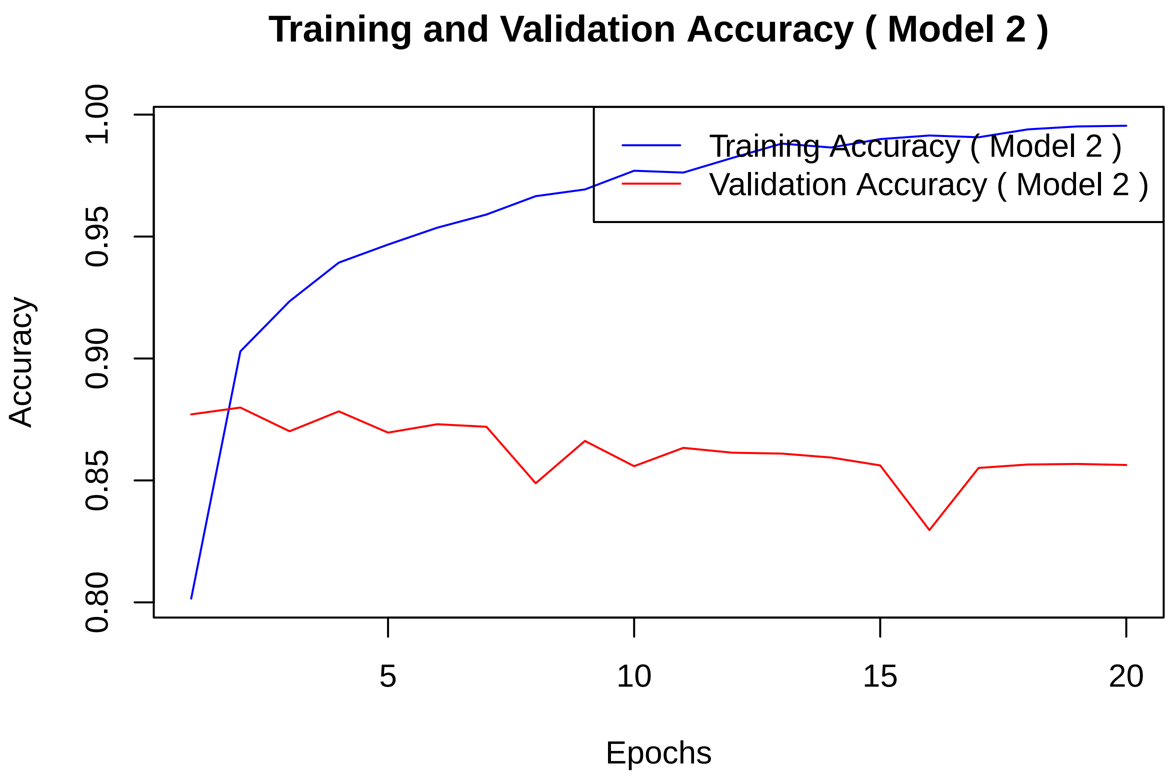
```r
# Function to plot the training and validation accuracy
plot_history <- function(history, label) {
  acc <- history$metrics$accuracy
  val_acc <- history$metrics$val_accuracy
  epochs <- seq_along(acc)
  plot(epochs, acc, type = 'l', col = 'blue', xlab = 'Epochs', ylab = 'Accuracy', main = paste('Training
  lines(epochs, val_acc, type = 'l', col = 'red')
  legend('topright', legend = c(paste('Training Accuracy (', label, ')'), paste('Validation Accuracy ('
}
```

```r
# Create a list of models
models_list <- list(model_one_hidden, model_three_hidden, model_32_units, model_64_units, model_mse_los

# Loop through each model, print summary, compile, fit, and plot
for (i in seq_along(models_list)) {
  label <- paste('Model', i)

  # Print model summary
  cat('\n\n', label, 'Summary:')
  summary(models_list[[i]])

  # Compile and fit the model
  history <- compile_and_fit_model(models_list[[i]], x_train, y_train, x_test, y_test)

  # Plot the training and validation accuracy
  plot_history(history, label)

  # Clear session to release memory
  gc()
}
```

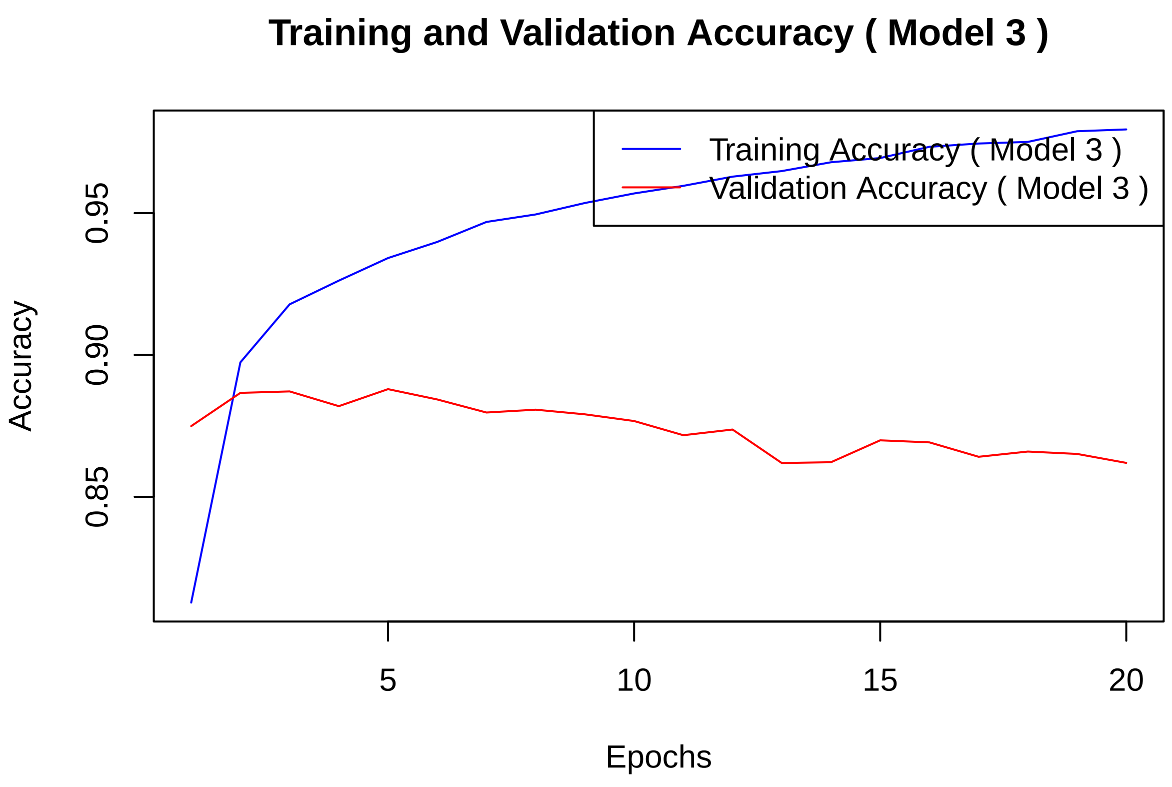
```
##
##
##  Model 1 Summary:Model: "sequential"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  dense_1 (Dense)                    (None, 16)                      160016
##  dense (Dense)                      (None, 1)                       17
## ================================================================================
## Total params: 160033 (625.13 KB)
## Trainable params: 160033 (625.13 KB)
## Non-trainable params: 0 (0.00 Byte)
## ________________________________________________________________________________
```



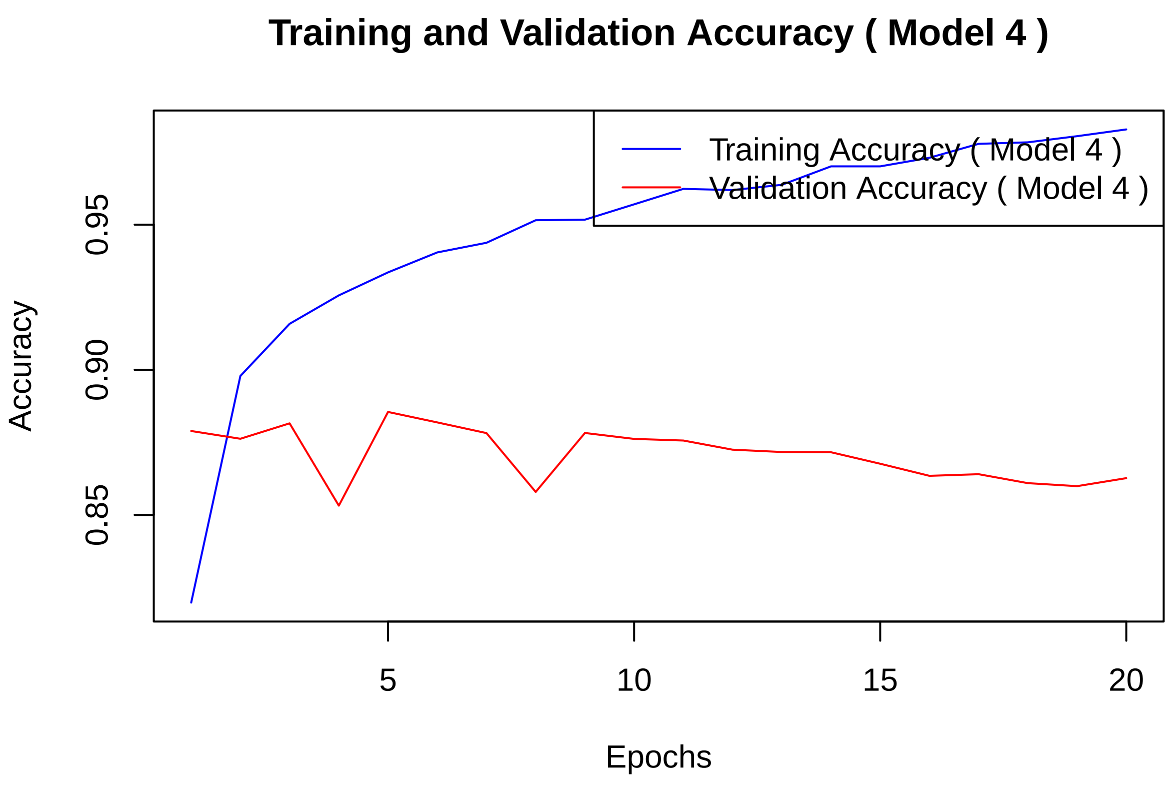
```
##
##
##  Model 2 Summary:Model: "sequential_1"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                  Param #
## ================================================================================
##  dense_5 (Dense)                    (None, 16)                    160016
##  dense_4 (Dense)                    (None, 16)                    272
##  dense_3 (Dense)                    (None, 16)                    272
##  dense_2 (Dense)                    (None, 1)                     17
## ================================================================================
## Total params: 160577 (627.25 KB)
## Trainable params: 160577 (627.25 KB)
## Non-trainable params: 0 (0.00 Byte)
## ________________________________________________________________________________
```



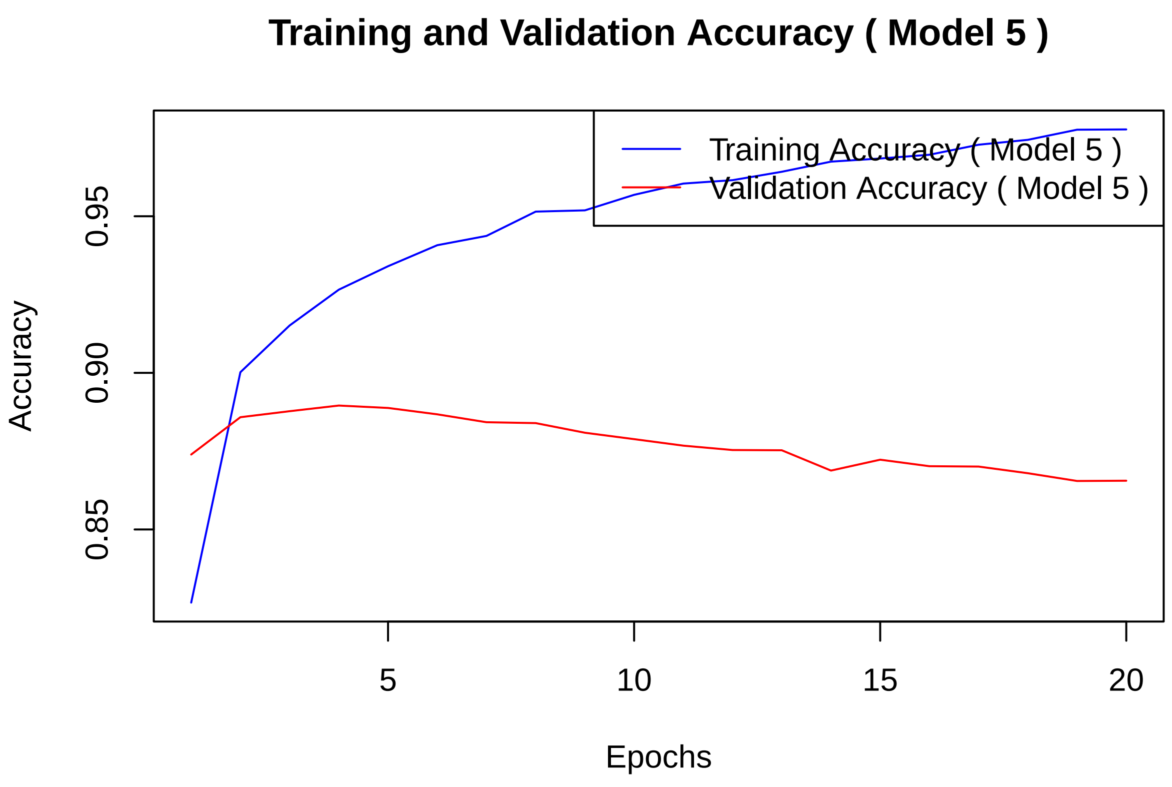
```
##
##
##  Model 3 Summary:Model: "sequential_2"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                  Param #
## ================================================================================
##  dense_7 (Dense)                    (None, 32)                    320032
##  dense_6 (Dense)                    (None, 1)                     33
## ================================================================================
## Total params: 320065 (1.22 MB)
## Trainable params: 320065 (1.22 MB)
## Non-trainable params: 0 (0.00 Byte)
## ________________________________________________________________________________
```



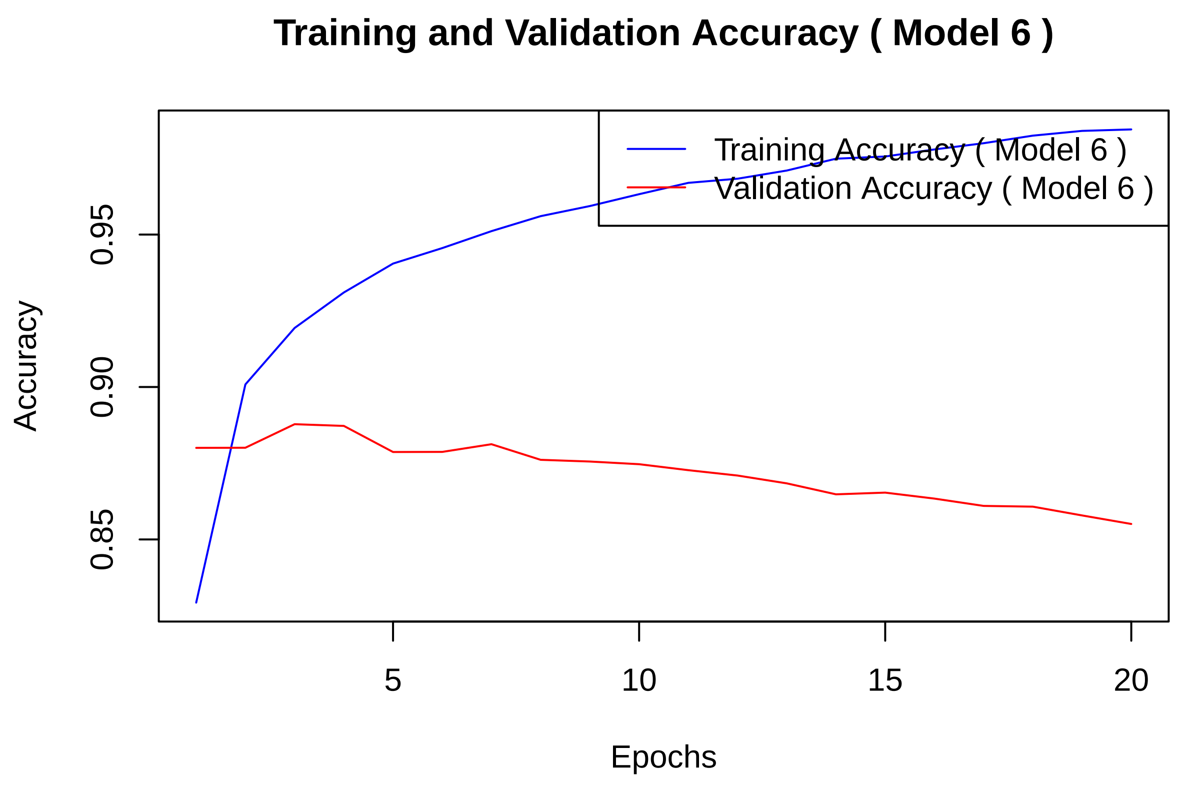
```
##
##
##  Model 4 Summary:Model: "sequential_3"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                       Param #
## ================================================================================
##  dense_9 (Dense)                    (None, 64)                         640064
##  dense_8 (Dense)                    (None, 1)                          65
## ================================================================================
## Total params: 640129 (2.44 MB)
## Trainable params: 640129 (2.44 MB)
## Non-trainable params: 0 (0.00 Byte)
## ________________________________________________________________________________
```



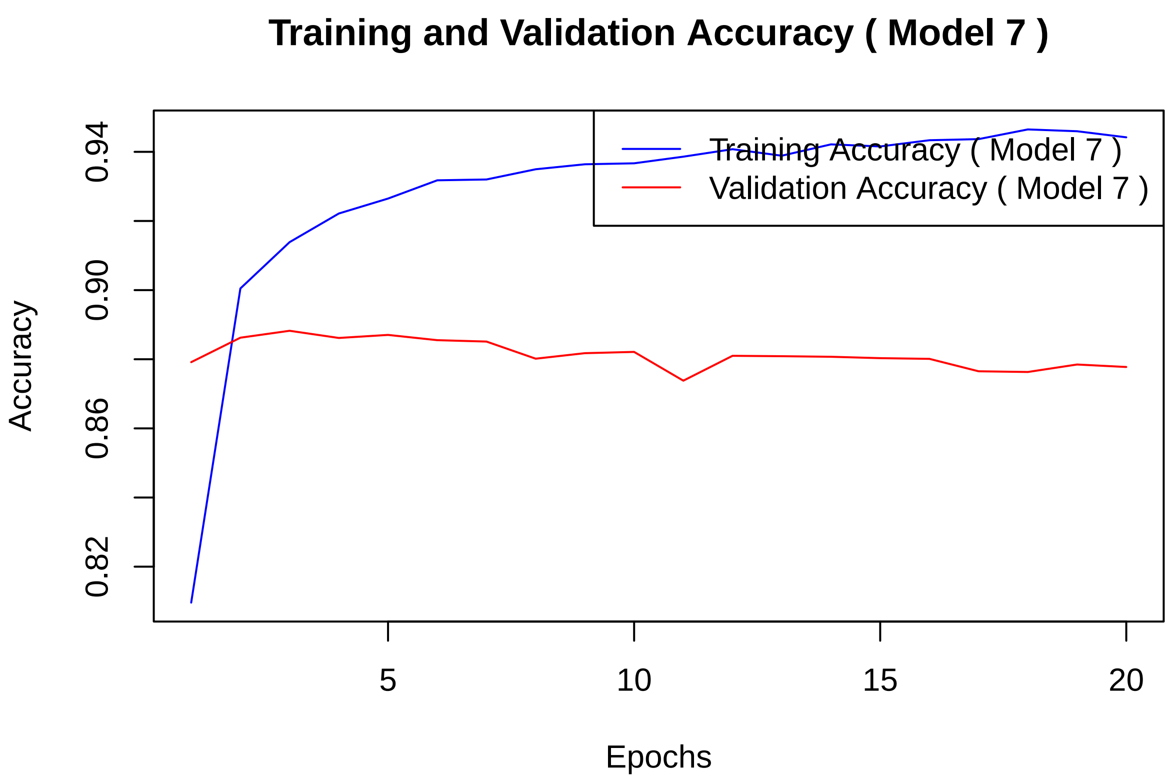
```
##
##
##  Model 5 Summary:Model: "sequential_4"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                       Param #
## ================================================================================
##  dense_11 (Dense)                   (None, 16)                         160016
##  dense_10 (Dense)                   (None, 1)                          17
## ================================================================================
## Total params: 160033 (625.13 KB)
## Trainable params: 160033 (625.13 KB)
## Non-trainable params: 0 (0.00 Byte)
## ________________________________________________________________________________
```



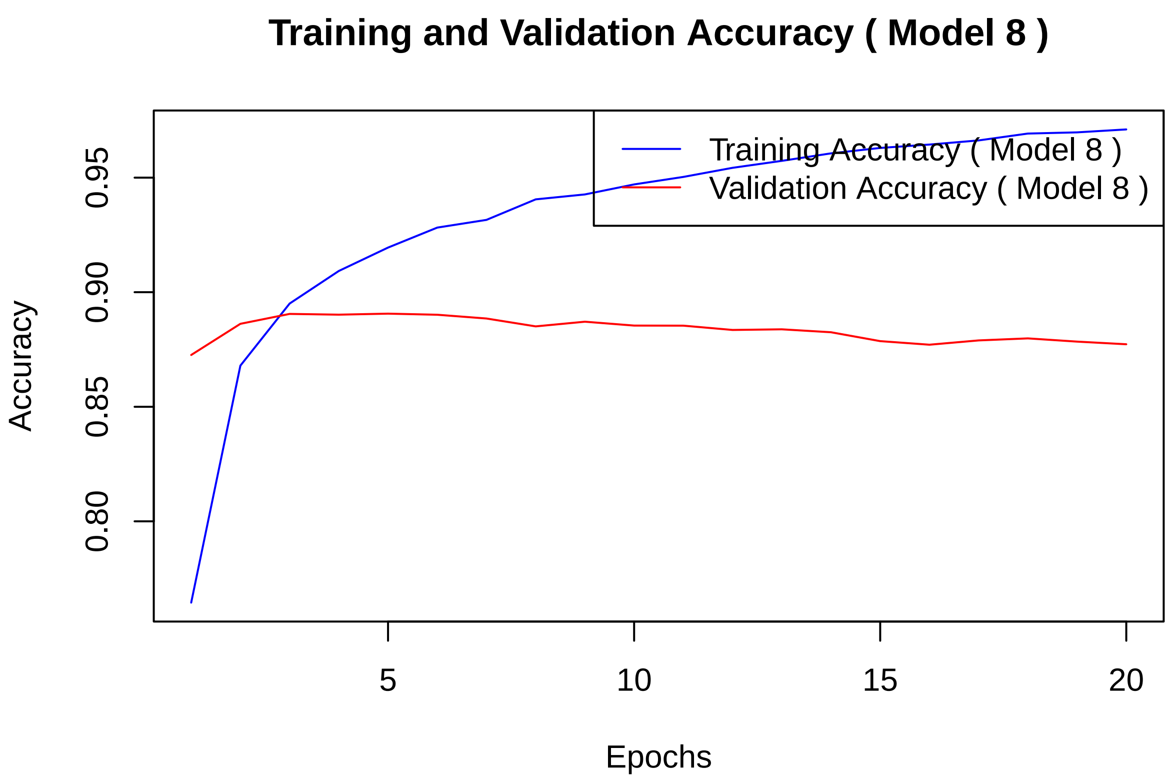
```
##
##
##  Model 6 Summary:Model: "sequential_5"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                     Param #
## ================================================================================
##  dense_13 (Dense)                   (None, 16)                       160016
##  dense_12 (Dense)                   (None, 1)                        17
## ================================================================================
## Total params: 160033 (625.13 KB)
## Trainable params: 160033 (625.13 KB)
## Non-trainable params: 0 (0.00 Byte)
## ________________________________________________________________________________
```



```
##
##
##  Model 7 Summary:Model: "sequential_6"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                      Param #
## ================================================================================
##  dense_15 (Dense)                   (None, 16)                        160016
##  dense_14 (Dense)                   (None, 1)                         17
## ================================================================================
## Total params: 160033 (625.13 KB)
## Trainable params: 160033 (625.13 KB)
## Non-trainable params: 0 (0.00 Byte)
## ________________________________________________________________________________
```



```
##
##
##  Model 8 Summary:Model: "sequential_7"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                      Param #
## ================================================================================
##  dense_17 (Dense)                   (None, 16)                        160016
##  dropout (Dropout)                  (None, 16)                        0
##  dense_16 (Dense)                   (None, 1)                         17
## ================================================================================
## Total params: 160033 (625.13 KB)
## Trainable params: 160033 (625.13 KB)
## Non-trainable params: 0 (0.00 Byte)
## ________________________________________________________________________________
```



1. **Data Loading and Preparation:**
   - The code starts by loading the IMDB dataset using Keras, consisting of movie reviews and their associated sentiment labels (positive or negative).
   - The reviews are vectorized to binary sequences, where each word is represented by a binary indicator.
2. **Model Architectures:**
   - Four different model architectures are defined to explore various configurations:
     - `model_one_hidden`: One hidden layer with 16 units.
     - `model_three_hidden`: Three hidden layers with 16 units each.
     - `model_32_units`: One hidden layer with 32 units.
     - `model_64_units`: One hidden layer with 64 units.
     - `model_mse_loss`: One hidden layer with Mean Squared Error (MSE) loss.
     - `model_tanh_activation`: One hidden layer with tanh activation.
     - `model_l2_regularization`: One hidden layer with L2 regularization.
     - `model_dropout`: One hidden layer with dropout.
3. **Model Compilation, Training, and Plotting:**
   - A function `compile_and_fit_model` is defined to compile and fit a model with binary cross-entropy loss and RMSprop optimizer.
   - Another function `plot_history` is defined to plot the training and validation accuracy over epochs.
   - The code then iterates through each model, prints its summary, compiles, fits on the training data, and plots the training and validation accuracy.
4. **Results Visualization:**
   - Finally, a plot is generated to compare the training and validation accuracy of different model configurations over epochs.
5. **Observations:**
   - The code aims to explore how different architectural choices impact the model's training and validation performance on the IMDB sentiment analysis task.