DABN13 - Assignment 6

Preamble: Data

In this lab we are using a dataset on beer purchases. Our goal is to predict if light beer purchased in the US is BUD light. To achieve this goal, we will use the information provided by the following socioeconomic characteristics: * market - where the beer is bought * buyertype - who is the buyer () * income - ranges of income * childrenUnder6 - does the buyer have children under 6 years old * children6to17 - does the buyer have children between 6 and 17 * age - bracketed age groups * employment - fully employed, partially employed, no employment. * degree - level of occupation * occuptation - which sector you are employed in * ethnic - white, asian, hispanic, black or other * microwave - own a microwave * dishwasher - own a dishwasher * tvcable - what type cable tv subscription you have * singlefamilyhome - are you in a single family home * npeople - number of people you live with 1,2,3,4, +5

First, we load the dataset and create an output variable that indicates purchases of Bud Light.

```
setwd("/Users/viktorsjoberg/Desktop/ML/Assignment 6")

lb    <- read.csv("LightBeer2.csv")
brand <- factor(lb$beer_brand)
y     <- rep(0, dim(lb)[1])
y[brand=="BUD LIGHT"]     <- 1
demog <- lb[,-(1:9)]

# relevel some things
for(name.col in colnames(demog)){
  demog[, name.col] <- as.factor(demog[, name.col] )
}

X <- model.matrix( ~ .,data= demog)[,-1]
```

We also split the data into a training and test sets:

```
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```
n_train        <- floor(3/4*length(y))
X_train        <- X[1:n_train,]
y_train        <- y[1:n_train]

scaling_X      <- preProcess(X_train, method = c("center", "scale"))

X_train        <- predict(scaling_X, newdata = X_train)

X_test         <- predict(scaling_X, newdata = X[(n_train+1):length(y),])
y_test         <- y[(n_train+1):length(y)]
```

Part 1: Specifying and training neural networks

We will now build a neural network to predict the purchase of Bud Light.

Task 1a)

We start with specifying the architecture of our very first and very small neural net `model1`. Add three layers to `model1`, two hidden layers with 30 and 15 hidden units, respectively, and an output layer. For the two hidden layers you should use the ReLU activation function. Additionally, choose a suitable activation for the output layer, given that we have a classification problem. For available activation functions, see the corresponding information in the Keras documentation.

```
library(keras)

model1 <- keras_model_sequential()

model1 %>%
  layer_dense(units = 30, activation = 'relu', input_shape = ncol(X)) %>%
  layer_dense(units = 15, activation = 'relu') %>%
  layer_dense(units = 1, activation = 'sigmoid')
```

Task 1b)

Next, we compile our model specification. From https://keras.io/api/losses/probabilistic_losses/ select a suitable loss function for our classification problem. For optimization function use *Adam* with learning rate 0.00001. Lastly, use 'accuracy' as a metric.

```
model1 %>%
  compile(
    loss = 'binary_crossentropy',
    optimizer = optimizer_adam(lr = 0.00001),
    metrics = c('accuracy')
  )
```

Task 1c)

Now train the model using 250 epochs, a batch_size of 2^8, and use 25% of the data for validation. Furthermore, assign the output of `fit()` to a new object `model1_fit`. Note that this object does **not** contain the fitted model (which is still `model1`) but instead a log of loss and accuracy at each epoch.

Lastly, use the string variable `loss_valloss_difference_1c` to describe and explain the observed difference between validation loss and training loss.

```
 model1_fit <- model1 %>%
  fit(
    X_train, y_train,
    epochs = 250,
    batch_size = 2^8,
    validation_split = 0.25
  )
```

```
## Epoch 1/250
## 161/161 - 2s - loss: 0.7164 - accuracy: 0.5805 - val_loss: 0.5325 - val_accuracy: 0.8128 - 2s/epoch
## Epoch 2/250
## 161/161 - 1s - loss: 0.6394 - accuracy: 0.6318 - val_loss: 0.5305 - val_accuracy: 0.7875 - 1s/epoch
## Epoch 3/250
## 161/161 - 1s - loss: 0.6328 - accuracy: 0.6404 - val_loss: 0.5273 - val_accuracy: 0.7947 - 1s/epoch
## Epoch 4/250
## 161/161 - 1s - loss: 0.6313 - accuracy: 0.6444 - val_loss: 0.5436 - val_accuracy: 0.7731 - 1s/epoch -
## Epoch 5/250
## 161/161 - 1s - loss: 0.6326 - accuracy: 0.6420 - val_loss: 0.5396 - val_accuracy: 0.7754 - 1s/epoch -
## Epoch 6/250
## 161/161 - 1s - loss: 0.6337 - accuracy: 0.6419 - val_loss: 0.5545 - val_accuracy: 0.7472 - 1s/epoch -
## Epoch 7/250
## 161/161 - 1s - loss: 0.6354 - accuracy: 0.6429 - val_loss: 0.5325 - val_accuracy: 0.7874 - 1s/epoch -
## Epoch 8/250
## 161/161 - 1s - loss: 0.6415 - accuracy: 0.6337 - val_loss: 0.5389 - val_accuracy: 0.7903 - 1s/epoch -
## Epoch 9/250
## 161/161 - 1s - loss: 0.6422 - accuracy: 0.6357 - val_loss: 0.5370 - val_accuracy: 0.7657 - 1s/epoch -
## Epoch 10/250
## 161/161 - 1s - loss: 0.6474 - accuracy: 0.6302 - val_loss: 0.5130 - val_accuracy: 0.8288 - 1s/epoch -
## Epoch 11/250
## 161/161 - 1s - loss: 0.6450 - accuracy: 0.6324 - val_loss: 0.5463 - val_accuracy: 0.7796 - 1s/epoch -
## Epoch 12/250
## 161/161 - 1s - loss: 0.6465 - accuracy: 0.6340 - val_loss: 0.5126 - val_accuracy: 0.7842 - 1s/epoch -
## Epoch 13/250
## 161/161 - 1s - loss: 0.6453 - accuracy: 0.6333 - val_loss: 0.5531 - val_accuracy: 0.7501 - 1s/epoch -
## Epoch 14/250
## 161/161 - 1s - loss: 0.6497 - accuracy: 0.6289 - val_loss: 0.5197 - val_accuracy: 0.8124 - 1s/epoch -
## Epoch 15/250
## 161/161 - 1s - loss: 0.6538 - accuracy: 0.6263 - val_loss: 0.5174 - val_accuracy: 0.8156 - 1s/epoch -
## Epoch 16/250
## 161/161 - 1s - loss: 0.6589 - accuracy: 0.6253 - val_loss: 0.5073 - val_accuracy: 0.8062 - 1s/epoch -
## Epoch 17/250
## 161/161 - 1s - loss: 0.6661 - accuracy: 0.6211 - val_loss: 0.5659 - val_accuracy: 0.7162 - 1s/epoch -
## Epoch 18/250
## 161/161 - 1s - loss: 0.6533 - accuracy: 0.6247 - val_loss: 0.5136 - val_accuracy: 0.7583 - 1s/epoch -
## Epoch 19/250
## 161/161 - 1s - loss: 0.6716 - accuracy: 0.6099 - val_loss: 0.5592 - val_accuracy: 0.6914 - 1s/epoch -
## Epoch 20/250
## 161/161 - 1s - loss: 0.6566 - accuracy: 0.6270 - val_loss: 0.5278 - val_accuracy: 0.7544 - 1s/epoch -
## Epoch 21/250
## 161/161 - 1s - loss: 0.6633 - accuracy: 0.6162 - val_loss: 0.5333 - val_accuracy: 0.7886 - 1s/epoch -
## Epoch 22/250
## 161/161 - 1s - loss: 0.6656 - accuracy: 0.6141 - val_loss: 0.5319 - val_accuracy: 0.7656 - 1s/epoch -
## Epoch 23/250
## 161/161 - 1s - loss: 0.6725 - accuracy: 0.6110 - val_loss: 0.5090 - val_accuracy: 0.7731 - 1s/epoch -
## Epoch 24/250
## 161/161 - 1s - loss: 0.6715 - accuracy: 0.6133 - val_loss: 0.5096 - val_accuracy: 0.7839 - 1s/epoch -
## Epoch 25/250
## 161/161 - 1s - loss: 0.6777 - accuracy: 0.6105 - val_loss: 0.6266 - val_accuracy: 0.6671 - 1s/epoch -
## Epoch 26/250
## 161/161 - 1s - loss: 0.6753 - accuracy: 0.6126 - val_loss: 0.5324 - val_accuracy: 0.7886 - 1s/epoch -
## Epoch 27/250
## 161/161 - 1s - loss: 0.6732 - accuracy: 0.6116 - val_loss: 0.5100 - val_accuracy: 0.7648 - 1s/epoch -
## Epoch 28/250
## 161/161 - 1s - loss: 0.6829 - accuracy: 0.6104 - val_loss: 0.5024 - val_accuracy: 0.8352 - 1s/epoch -
## Epoch 29/250
## 161/161 - 1s - loss: 0.6969 - accuracy: 0.6055 - val_loss: 0.5549 - val_accuracy: 0.7197 - 1s/epoch -
## Epoch 30/250
## 161/161 - 1s - loss: 0.6807 - accuracy: 0.6119 - val_loss: 0.5575 - val_accuracy: 0.6936 - 1s/epoch -
## Epoch 31/250
## 161/161 - 1s - loss: 0.6719 - accuracy: 0.6161 - val_loss: 0.5611 - val_accuracy: 0.7100 - 1s/epoch
## Epoch 32/250
## 161/161 - 1s - loss: 0.6968 - accuracy: 0.6053 - val_loss: 0.5954 - val_accuracy: 0.6963 - 1s/epoch
## Epoch 33/250
## 161/161 - 1s - loss: 0.7115 - accuracy: 0.5994 - val_loss: 0.6113 - val_accuracy: 0.6677 - 1s/epoch
## Epoch 34/250
## 161/161 - 1s - loss: 0.6937 - accuracy: 0.6063 - val_loss: 0.5713 - val_accuracy: 0.7076 - 1s/epoch
## Epoch 35/250
## 161/161 - 1s - loss: 0.7026 - accuracy: 0.6071 - val_loss: 0.6056 - val_accuracy: 0.6572 - 1s/epoch
## Epoch 36/250
## 161/161 - 2s - loss: 0.6870 - accuracy: 0.6079 - val_loss: 0.5332 - val_accuracy: 0.7623 - 2s/epoch
## Epoch 37/250
## 161/161 - 2s - loss: 0.7447 - accuracy: 0.5962 - val_loss: 0.5308 - val_accuracy: 0.7766 - 2s/epoch
## Epoch 38/250
## 161/161 - 1s - loss: 0.6859 - accuracy: 0.6104 - val_loss: 0.4966 - val_accuracy: 0.7936 - 1s/epoch
## Epoch 39/250
## 161/161 - 1s - loss: 0.7286 - accuracy: 0.5978 - val_loss: 0.7155 - val_accuracy: 0.5974 - 1s/epoch
## Epoch 40/250
## 161/161 - 2s - loss: 0.6944 - accuracy: 0.6096 - val_loss: 0.5586 - val_accuracy: 0.7390 - 2s/epoch
## Epoch 41/250
## 161/161 - 1s - loss: 0.6788 - accuracy: 0.6105 - val_loss: 0.6192 - val_accuracy: 0.6334 - 1s/epoch
## Epoch 42/250
## 161/161 - 1s - loss: 0.7336 - accuracy: 0.5953 - val_loss: 0.6047 - val_accuracy: 0.6813 - 1s/epoch
## Epoch 43/250
## 161/161 - 1s - loss: 0.7578 - accuracy: 0.5894 - val_loss: 0.6494 - val_accuracy: 0.6470 - 1s/epoch
## Epoch 44/250
## 161/161 - 1s - loss: 0.7283 - accuracy: 0.5978 - val_loss: 0.5556 - val_accuracy: 0.7490 - 1s/epoch
## Epoch 45/250
## 161/161 - 1s - loss: 0.7446 - accuracy: 0.5960 - val_loss: 0.5375 - val_accuracy: 0.7423 - 1s/epoch
## Epoch 46/250
## 161/161 - 1s - loss: 0.7428 - accuracy: 0.5966 - val_loss: 0.6695 - val_accuracy: 0.6146 - 1s/epoch
## Epoch 47/250
## 161/161 - 1s - loss: 0.7569 - accuracy: 0.5996 - val_loss: 0.5969 - val_accuracy: 0.6607 - 1s/epoch
## Epoch 48/250
## 161/161 - 1s - loss: 0.7730 - accuracy: 0.5879 - val_loss: 0.6544 - val_accuracy: 0.6872 - 1s/epoch
## Epoch 49/250
## 161/161 - 1s - loss: 0.7505 - accuracy: 0.5932 - val_loss: 0.5224 - val_accuracy: 0.7764 - 1s/epoch
## Epoch 50/250
## 161/161 - 1s - loss: 0.7032 - accuracy: 0.6060 - val_loss: 0.7805 - val_accuracy: 0.5733 - 1s/epoch
## Epoch 51/250
## 161/161 - 1s - loss: 0.7918 - accuracy: 0.5925 - val_loss: 0.6690 - val_accuracy: 0.6236 - 1s/epoch
## Epoch 52/250
## 161/161 - 1s - loss: 0.7687 - accuracy: 0.5935 - val_loss: 0.7315 - val_accuracy: 0.6143 - 1s/epoch
## Epoch 53/250
## 161/161 - 1s - loss: 0.7920 - accuracy: 0.5901 - val_loss: 0.7968 - val_accuracy: 0.6442 - 1s/epoch
## Epoch 54/250
## 161/161 - 1s - loss: 0.8121 - accuracy: 0.5832 - val_loss: 0.5958 - val_accuracy: 0.6882 - 1s/epoch
## Epoch 55/250
## 161/161 - 1s - loss: 0.8105 - accuracy: 0.5868 - val_loss: 0.5049 - val_accuracy: 0.8066 - 1s/epoch
## Epoch 56/250
## 161/161 - 1s - loss: 0.7345 - accuracy: 0.5980 - val_loss: 0.5487 - val_accuracy: 0.7356 - 1s/epoch
## Epoch 57/250
## 161/161 - 1s - loss: 0.7671 - accuracy: 0.5900 - val_loss: 0.6515 - val_accuracy: 0.6798 - 1s/epoch
## Epoch 58/250
## 161/161 - 1s - loss: 0.7485 - accuracy: 0.5965 - val_loss: 0.4972 - val_accuracy: 0.8162 - 1s/epoch -
## Epoch 59/250
## 161/161 - 1s - loss: 0.8602 - accuracy: 0.5780 - val_loss: 0.5071 - val_accuracy: 0.7850 - 1s/epoch -
## Epoch 60/250
## 161/161 - 1s - loss: 0.8081 - accuracy: 0.5918 - val_loss: 1.2000 - val_accuracy: 0.5485 - 1s/epoch -
## Epoch 61/250
## 161/161 - 1s - loss: 0.9296 - accuracy: 0.5731 - val_loss: 0.5831 - val_accuracy: 0.6976 - 1s/epoch -
## Epoch 62/250
## 161/161 - 1s - loss: 0.8376 - accuracy: 0.5880 - val_loss: 0.7305 - val_accuracy: 0.6842 - 1s/epoch -
## Epoch 63/250
## 161/161 - 1s - loss: 0.8900 - accuracy: 0.5812 - val_loss: 0.5775 - val_accuracy: 0.6863 - 1s/epoch -
## Epoch 64/250
## 161/161 - 1s - loss: 0.7652 - accuracy: 0.5931 - val_loss: 0.5146 - val_accuracy: 0.8244 - 1s/epoch -
## Epoch 65/250
## 161/161 - 1s - loss: 0.7746 - accuracy: 0.5913 - val_loss: 0.8832 - val_accuracy: 0.5742 - 1s/epoch -
## Epoch 66/250
## 161/161 - 1s - loss: 0.8557 - accuracy: 0.5876 - val_loss: 0.8024 - val_accuracy: 0.5804 - 1s/epoch -
## Epoch 67/250
## 161/161 - 2s - loss: 0.9476 - accuracy: 0.5739 - val_loss: 0.8743 - val_accuracy: 0.5719 - 2s/epoch -
## Epoch 68/250
## 161/161 - 2s - loss: 0.8707 - accuracy: 0.5823 - val_loss: 0.5240 - val_accuracy: 0.7653 - 2s/epoch -
## Epoch 69/250
## 161/161 - 1s - loss: 0.8929 - accuracy: 0.5812 - val_loss: 0.6170 - val_accuracy: 0.6881 - 1s/epoch -
## Epoch 70/250
## 161/161 - 2s - loss: 0.9306 - accuracy: 0.5756 - val_loss: 0.8246 - val_accuracy: 0.6179 - 2s/epoch -
## Epoch 71/250
## 161/161 - 1s - loss: 0.8070 - accuracy: 0.5953 - val_loss: 0.9088 - val_accuracy: 0.5644 - 1s/epoch -
## Epoch 72/250
## 161/161 - 1s - loss: 0.8590 - accuracy: 0.5826 - val_loss: 0.5080 - val_accuracy: 0.7710 - 1s/epoch -
## Epoch 73/250
## 161/161 - 1s - loss: 0.8911 - accuracy: 0.5841 - val_loss: 0.4841 - val_accuracy: 0.8411 - 1s/epoch -
## Epoch 74/250
## 161/161 - 1s - loss: 0.9224 - accuracy: 0.5777 - val_loss: 0.7520 - val_accuracy: 0.6346 - 1s/epoch -
## Epoch 75/250
## 161/161 - 1s - loss: 0.9193 - accuracy: 0.5857 - val_loss: 0.4879 - val_accuracy: 0.8189 - 1s/epoch -
## Epoch 76/250
## 161/161 - 1s - loss: 0.8820 - accuracy: 0.5847 - val_loss: 0.5169 - val_accuracy: 0.7609 - 1s/epoch -
## Epoch 77/250
## 161/161 - 1s - loss: 0.9104 - accuracy: 0.5798 - val_loss: 0.7630 - val_accuracy: 0.6474 - 1s/epoch -
## Epoch 78/250
## 161/161 - 1s - loss: 0.9450 - accuracy: 0.5737 - val_loss: 0.7515 - val_accuracy: 0.6514 - 1s/epoch -
## Epoch 79/250
## 161/161 - 1s - loss: 0.9706 - accuracy: 0.5775 - val_loss: 0.8233 - val_accuracy: 0.6160 - 1s/epoch -
## Epoch 80/250
## 161/161 - 1s - loss: 0.8683 - accuracy: 0.5890 - val_loss: 0.7966 - val_accuracy: 0.5792 - 1s/epoch -
## Epoch 81/250
## 161/161 - 1s - loss: 0.9822 - accuracy: 0.5760 - val_loss: 0.5578 - val_accuracy: 0.6925 - 1s/epoch -
## Epoch 82/250
## 161/161 - 1s - loss: 0.9864 - accuracy: 0.5735 - val_loss: 0.7699 - val_accuracy: 0.6692 - 1s/epoch -
## Epoch 83/250
## 161/161 - 1s - loss: 0.9092 - accuracy: 0.5850 - val_loss: 0.5689 - val_accuracy: 0.6988 - 1s/epoch -
## Epoch 84/250
## 161/161 - 1s - loss: 1.1168 - accuracy: 0.5733 - val_loss: 0.5384 - val_accuracy: 0.7385 - 1s/epoch -
## Epoch 85/250
## 161/161 - 1s - loss: 1.0175 - accuracy: 0.5822 - val_loss: 0.6277 - val_accuracy: 0.6835 - 1s/epoch -
## Epoch 86/250
## 161/161 - 1s - loss: 1.2430 - accuracy: 0.5688 - val_loss: 0.5708 - val_accuracy: 0.7203 - 1s/epoch -
## Epoch 87/250
## 161/161 - 1s - loss: 1.0607 - accuracy: 0.5696 - val_loss: 0.4761 - val_accuracy: 0.8294 - 1s/epoch -
## Epoch 88/250
## 161/161 - 1s - loss: 1.2750 - accuracy: 0.5604 - val_loss: 0.9474 - val_accuracy: 0.6313 - 1s/epoch -
## Epoch 89/250
## 161/161 - 1s - loss: 0.9491 - accuracy: 0.5827 - val_loss: 0.6260 - val_accuracy: 0.6906 - 1s/epoch -
## Epoch 90/250
## 161/161 - 1s - loss: 1.0950 - accuracy: 0.5751 - val_loss: 0.8754 - val_accuracy: 0.6023 - 1s/epoch -
## Epoch 91/250
## 161/161 - 1s - loss: 1.0637 - accuracy: 0.5709 - val_loss: 1.5783 - val_accuracy: 0.4961 - 1s/epoch -
## Epoch 92/250
## 161/161 - 1s - loss: 1.0962 - accuracy: 0.5774 - val_loss: 1.1858 - val_accuracy: 0.6044 - 1s/epoch -
## Epoch 93/250
## 161/161 - 2s - loss: 1.0916 - accuracy: 0.5732 - val_loss: 0.4750 - val_accuracy: 0.8385 - 2s/epoch -
## Epoch 94/250
## 161/161 - 1s - loss: 1.0437 - accuracy: 0.5810 - val_loss: 0.5223 - val_accuracy: 0.7321 - 1s/epoch -
## Epoch 95/250
## 161/161 - 1s - loss: 0.9376 - accuracy: 0.5807 - val_loss: 0.4428 - val_accuracy: 0.8459 - 1s/epoch -
## Epoch 96/250
## 161/161 - 1s - loss: 1.3676 - accuracy: 0.5651 - val_loss: 1.2728 - val_accuracy: 0.5447 - 1s/epoch -
## Epoch 97/250
## 161/161 - 1s - loss: 1.1604 - accuracy: 0.5713 - val_loss: 1.0822 - val_accuracy: 0.5493 - 1s/epoch -
## Epoch 98/250
## 161/161 - 1s - loss: 1.4693 - accuracy: 0.5583 - val_loss: 1.8823 - val_accuracy: 0.5193 - 1s/epoch -
## Epoch 99/250
## 161/161 - 1s - loss: 1.1182 - accuracy: 0.5750 - val_loss: 1.6607 - val_accuracy: 0.5089 - 1s/epoch -
## Epoch 100/250
## 161/161 - 1s - loss: 1.2817 - accuracy: 0.5669 - val_loss: 0.6750 - val_accuracy: 0.6653 - 1s/epoch -
## Epoch 101/250
## 161/161 - 1s - loss: 1.2011 - accuracy: 0.5682 - val_loss: 0.5879 - val_accuracy: 0.7052 - 1s/epoch -
## Epoch 102/250
## 161/161 - 1s - loss: 1.0381 - accuracy: 0.5792 - val_loss: 0.9555 - val_accuracy: 0.6261 - 1s/epoch -
## Epoch 103/250
## 161/161 - 1s - loss: 1.1516 - accuracy: 0.5691 - val_loss: 0.7804 - val_accuracy: 0.6329 - 1s/epoch -
## Epoch 104/250
## 161/161 - 1s - loss: 1.2823 - accuracy: 0.5653 - val_loss: 0.9314 - val_accuracy: 0.5759 - 1s/epoch -
## Epoch 105/250
## 161/161 - 1s - loss: 1.2537 - accuracy: 0.5727 - val_loss: 0.7774 - val_accuracy: 0.6057 - 1s/epoch -
## Epoch 106/250
## 161/161 - 1s - loss: 1.1475 - accuracy: 0.5743 - val_loss: 0.7055 - val_accuracy: 0.6165 - 1s/epoch -
## Epoch 107/250
## 161/161 - 1s - loss: 1.3145 - accuracy: 0.5739 - val_loss: 0.4921 - val_accuracy: 0.7588 - 1s/epoch -
## Epoch 108/250
## 161/161 - 1s - loss: 1.1805 - accuracy: 0.5787 - val_loss: 0.7158 - val_accuracy: 0.6384 - 1s/epoch -
## Epoch 109/250
## 161/161 - 1s - loss: 1.2198 - accuracy: 0.5696 - val_loss: 1.2286 - val_accuracy: 0.5380 - 1s/epoch -
## Epoch 110/250
## 161/161 - 1s - loss: 1.5633 - accuracy: 0.5634 - val_loss: 1.4213 - val_accuracy: 0.5900 - 1s/epoch -
## Epoch 111/250
## 161/161 - 1s - loss: 1.3142 - accuracy: 0.5646 - val_loss: 0.7301 - val_accuracy: 0.6731 - 1s/epoch -
## Epoch 112/250
## 161/161 - 1s - loss: 1.3535 - accuracy: 0.5642 - val_loss: 1.2386 - val_accuracy: 0.5911 - 1s/epoch
## Epoch 113/250
## 161/161 - 1s - loss: 1.3247 - accuracy: 0.5672 - val_loss: 1.9555 - val_accuracy: 0.6198 - 1s/epoch
## Epoch 114/250
## 161/161 - 1s - loss: 1.7979 - accuracy: 0.5635 - val_loss: 0.5385 - val_accuracy: 0.7542 - 1s/epoch
## Epoch 115/250
## 161/161 - 1s - loss: 1.3068 - accuracy: 0.5672 - val_loss: 1.6076 - val_accuracy: 0.5337 - 1s/epoch
## Epoch 116/250
## 161/161 - 1s - loss: 1.3145 - accuracy: 0.5740 - val_loss: 0.5064 - val_accuracy: 0.7526 - 1s/epoch
## Epoch 117/250
## 161/161 - 1s - loss: 1.1976 - accuracy: 0.5784 - val_loss: 0.5021 - val_accuracy: 0.7372 - 1s/epoch
## Epoch 118/250
## 161/161 - 1s - loss: 1.1932 - accuracy: 0.5739 - val_loss: 0.9874 - val_accuracy: 0.6494 - 1s/epoch
## Epoch 119/250
## 161/161 - 1s - loss: 1.3914 - accuracy: 0.5601 - val_loss: 1.5607 - val_accuracy: 0.6187 - 1s/epoch
## Epoch 120/250
## 161/161 - 1s - loss: 1.2527 - accuracy: 0.5718 - val_loss: 1.6393 - val_accuracy: 0.5276 - 1s/epoch
## Epoch 121/250
## 161/161 - 1s - loss: 1.3660 - accuracy: 0.5661 - val_loss: 1.2160 - val_accuracy: 0.6664 - 1s/epoch
## Epoch 122/250
## 161/161 - 1s - loss: 1.3270 - accuracy: 0.5706 - val_loss: 1.2986 - val_accuracy: 0.6355 - 1s/epoch
## Epoch 123/250
## 161/161 - 1s - loss: 1.7061 - accuracy: 0.5585 - val_loss: 0.4876 - val_accuracy: 0.7808 - 1s/epoch
## Epoch 124/250
## 161/161 - 1s - loss: 1.4529 - accuracy: 0.5656 - val_loss: 0.5496 - val_accuracy: 0.7086 - 1s/epoch
## Epoch 125/250
## 161/161 - 1s - loss: 1.2808 - accuracy: 0.5745 - val_loss: 0.9532 - val_accuracy: 0.6285 - 1s/epoch
## Epoch 126/250
## 161/161 - 1s - loss: 1.6793 - accuracy: 0.5599 - val_loss: 0.9138 - val_accuracy: 0.6578 - 1s/epoch
## Epoch 127/250
## 161/161 - 1s - loss: 1.6442 - accuracy: 0.5580 - val_loss: 1.1911 - val_accuracy: 0.5452 - 1s/epoch
## Epoch 128/250
## 161/161 - 1s - loss: 1.3003 - accuracy: 0.5779 - val_loss: 1.0377 - val_accuracy: 0.5991 - 1s/epoch
## Epoch 129/250
## 161/161 - 1s - loss: 1.5523 - accuracy: 0.5646 - val_loss: 0.4704 - val_accuracy: 0.7853 - 1s/epoch
## Epoch 130/250
## 161/161 - 1s - loss: 1.3550 - accuracy: 0.5702 - val_loss: 1.4861 - val_accuracy: 0.6491 - 1s/epoch
## Epoch 131/250
## 161/161 - 1s - loss: 1.7011 - accuracy: 0.5601 - val_loss: 1.5468 - val_accuracy: 0.6200 - 1s/epoch
## Epoch 132/250
## 161/161 - 1s - loss: 1.5736 - accuracy: 0.5654 - val_loss: 0.8410 - val_accuracy: 0.6301 - 1s/epoch
## Epoch 133/250
## 161/161 - 1s - loss: 1.7179 - accuracy: 0.5556 - val_loss: 0.6639 - val_accuracy: 0.6860 - 1s/epoch
## Epoch 134/250
## 161/161 - 1s - loss: 1.6607 - accuracy: 0.5732 - val_loss: 0.4498 - val_accuracy: 0.7906 - 1s/epoch
## Epoch 135/250
## 161/161 - 1s - loss: 1.3955 - accuracy: 0.5763 - val_loss: 1.6949 - val_accuracy: 0.5158 - 1s/epoch
## Epoch 136/250
## 161/161 - 1s - loss: 1.6990 - accuracy: 0.5523 - val_loss: 1.7977 - val_accuracy: 0.6054 - 1s/epoch
## Epoch 137/250
## 161/161 - 1s - loss: 1.6502 - accuracy: 0.5600 - val_loss: 1.2967 - val_accuracy: 0.6466 - 1s/epoch
## Epoch 138/250
## 161/161 - 1s - loss: 1.5275 - accuracy: 0.5641 - val_loss: 0.7103 - val_accuracy: 0.7105 - 1s/epoch
## Epoch 139/250
## 161/161 - 1s - loss: 1.6161 - accuracy: 0.5639 - val_loss: 0.7278 - val_accuracy: 0.6782 - 1s/epoch -
## Epoch 140/250
## 161/161 - 1s - loss: 1.5408 - accuracy: 0.5667 - val_loss: 0.6793 - val_accuracy: 0.7087 - 1s/epoch -
## Epoch 141/250
## 161/161 - 1s - loss: 1.7581 - accuracy: 0.5609 - val_loss: 0.8657 - val_accuracy: 0.6387 - 1s/epoch -
## Epoch 142/250
## 161/161 - 1s - loss: 1.7697 - accuracy: 0.5632 - val_loss: 1.5282 - val_accuracy: 0.6348 - 1s/epoch -
## Epoch 143/250
## 161/161 - 1s - loss: 1.9797 - accuracy: 0.5572 - val_loss: 0.7522 - val_accuracy: 0.7130 - 1s/epoch -
## Epoch 144/250
## 161/161 - 1s - loss: 1.6833 - accuracy: 0.5643 - val_loss: 1.7704 - val_accuracy: 0.5876 - 1s/epoch -
## Epoch 145/250
## 161/161 - 1s - loss: 1.7898 - accuracy: 0.5600 - val_loss: 0.8435 - val_accuracy: 0.6835 - 1s/epoch -
## Epoch 146/250
## 161/161 - 1s - loss: 1.8239 - accuracy: 0.5634 - val_loss: 1.5395 - val_accuracy: 0.6599 - 1s/epoch -
## Epoch 147/250
## 161/161 - 1s - loss: 1.7522 - accuracy: 0.5673 - val_loss: 1.2832 - val_accuracy: 0.6236 - 1s/epoch -
## Epoch 148/250
## 161/161 - 1s - loss: 1.9208 - accuracy: 0.5546 - val_loss: 1.5150 - val_accuracy: 0.6415 - 1s/epoch -
## Epoch 149/250
## 161/161 - 1s - loss: 1.8752 - accuracy: 0.5667 - val_loss: 1.1286 - val_accuracy: 0.6239 - 1s/epoch -
## Epoch 150/250
## 161/161 - 1s - loss: 1.9905 - accuracy: 0.5611 - val_loss: 0.7332 - val_accuracy: 0.6271 - 1s/epoch -
## Epoch 151/250
## 161/161 - 1s - loss: 1.7874 - accuracy: 0.5668 - val_loss: 2.1354 - val_accuracy: 0.6159 - 1s/epoch -
## Epoch 152/250
## 161/161 - 1s - loss: 2.0872 - accuracy: 0.5629 - val_loss: 0.8475 - val_accuracy: 0.6904 - 1s/epoch -
## Epoch 153/250
## 161/161 - 1s - loss: 1.7113 - accuracy: 0.5653 - val_loss: 0.7955 - val_accuracy: 0.6711 - 1s/epoch -
## Epoch 154/250
## 161/161 - 1s - loss: 1.6977 - accuracy: 0.5612 - val_loss: 0.8194 - val_accuracy: 0.6519 - 1s/epoch -
## Epoch 155/250
## 161/161 - 1s - loss: 2.1349 - accuracy: 0.5600 - val_loss: 1.8813 - val_accuracy: 0.6202 - 1s/epoch -
## Epoch 156/250
## 161/161 - 1s - loss: 1.9971 - accuracy: 0.5578 - val_loss: 3.7799 - val_accuracy: 0.5127 - 1s/epoch -
## Epoch 157/250
## 161/161 - 1s - loss: 1.6537 - accuracy: 0.5694 - val_loss: 1.3794 - val_accuracy: 0.5709 - 1s/epoch -
## Epoch 158/250
## 161/161 - 1s - loss: 2.4458 - accuracy: 0.5591 - val_loss: 2.9633 - val_accuracy: 0.5041 - 1s/epoch -
## Epoch 159/250
## 161/161 - 1s - loss: 2.2492 - accuracy: 0.5587 - val_loss: 3.7606 - val_accuracy: 0.5954 - 1s/epoch -
## Epoch 160/250
## 161/161 - 1s - loss: 2.5261 - accuracy: 0.5587 - val_loss: 1.6538 - val_accuracy: 0.5489 - 1s/epoch -
## Epoch 161/250
## 161/161 - 1s - loss: 2.4526 - accuracy: 0.5566 - val_loss: 2.4094 - val_accuracy: 0.5125 - 1s/epoch -
## Epoch 162/250
## 161/161 - 1s - loss: 2.4235 - accuracy: 0.5570 - val_loss: 1.0240 - val_accuracy: 0.6168 - 1s/epoch -
## Epoch 163/250
## 161/161 - 1s - loss: 1.8783 - accuracy: 0.5650 - val_loss: 0.5024 - val_accuracy: 0.7750 - 1s/epoch -
## Epoch 164/250
## 161/161 - 1s - loss: 2.6280 - accuracy: 0.5489 - val_loss: 1.2320 - val_accuracy: 0.6218 - 1s/epoch -
## Epoch 165/250
## 161/161 - 1s - loss: 1.9576 - accuracy: 0.5642 - val_loss: 4.7942 - val_accuracy: 0.6053 - 1s/epoch -
## Epoch 166/250
## 161/161 - 1s - loss: 2.7944 - accuracy: 0.5535 - val_loss: 2.2173 - val_accuracy: 0.6207 - 1s/epoch
## Epoch 167/250
## 161/161 - 1s - loss: 1.7622 - accuracy: 0.5738 - val_loss: 3.0020 - val_accuracy: 0.5065 - 1s/epoch
## Epoch 168/250
## 161/161 - 1s - loss: 2.1275 - accuracy: 0.5612 - val_loss: 1.5420 - val_accuracy: 0.5779 - 1s/epoch
## Epoch 169/250
## 161/161 - 1s - loss: 2.0151 - accuracy: 0.5652 - val_loss: 2.5471 - val_accuracy: 0.6079 - 1s/epoch
## Epoch 170/250
## 161/161 - 1s - loss: 3.5493 - accuracy: 0.5452 - val_loss: 2.3278 - val_accuracy: 0.6474 - 1s/epoch
## Epoch 171/250
## 161/161 - 1s - loss: 2.2885 - accuracy: 0.5608 - val_loss: 1.1263 - val_accuracy: 0.5903 - 1s/epoch
## Epoch 172/250
## 161/161 - 1s - loss: 2.4476 - accuracy: 0.5573 - val_loss: 2.5742 - val_accuracy: 0.6178 - 1s/epoch
## Epoch 173/250
## 161/161 - 1s - loss: 2.9508 - accuracy: 0.5572 - val_loss: 0.9481 - val_accuracy: 0.6990 - 1s/epoch
## Epoch 174/250
## 161/161 - 1s - loss: 2.1192 - accuracy: 0.5663 - val_loss: 3.8205 - val_accuracy: 0.5501 - 1s/epoch
## Epoch 175/250
## 161/161 - 1s - loss: 2.2422 - accuracy: 0.5670 - val_loss: 0.6909 - val_accuracy: 0.6984 - 1s/epoch
## Epoch 176/250
## 161/161 - 1s - loss: 2.7261 - accuracy: 0.5624 - val_loss: 0.8671 - val_accuracy: 0.7180 - 1s/epoch
## Epoch 177/250
## 161/161 - 1s - loss: 2.8396 - accuracy: 0.5520 - val_loss: 1.3327 - val_accuracy: 0.5666 - 1s/epoch
## Epoch 178/250
## 161/161 - 1s - loss: 2.3474 - accuracy: 0.5555 - val_loss: 1.0768 - val_accuracy: 0.6200 - 1s/epoch
## Epoch 179/250
## 161/161 - 1s - loss: 1.8718 - accuracy: 0.5708 - val_loss: 3.2705 - val_accuracy: 0.6197 - 1s/epoch
## Epoch 180/250
## 161/161 - 1s - loss: 2.8263 - accuracy: 0.5484 - val_loss: 0.6926 - val_accuracy: 0.7184 - 1s/epoch
## Epoch 181/250
## 161/161 - 1s - loss: 3.2895 - accuracy: 0.5478 - val_loss: 1.9033 - val_accuracy: 0.5956 - 1s/epoch
## Epoch 182/250
## 161/161 - 1s - loss: 2.7245 - accuracy: 0.5573 - val_loss: 1.2378 - val_accuracy: 0.6886 - 1s/epoch
## Epoch 183/250
## 161/161 - 1s - loss: 2.2698 - accuracy: 0.5636 - val_loss: 1.5949 - val_accuracy: 0.5989 - 1s/epoch
## Epoch 184/250
## 161/161 - 1s - loss: 2.3462 - accuracy: 0.5592 - val_loss: 0.9757 - val_accuracy: 0.7052 - 1s/epoch
## Epoch 185/250
## 161/161 - 1s - loss: 2.5752 - accuracy: 0.5557 - val_loss: 0.7064 - val_accuracy: 0.6885 - 1s/epoch
## Epoch 186/250
## 161/161 - 1s - loss: 2.3017 - accuracy: 0.5669 - val_loss: 3.2500 - val_accuracy: 0.5111 - 1s/epoch
## Epoch 187/250
## 161/161 - 1s - loss: 2.1759 - accuracy: 0.5727 - val_loss: 0.9334 - val_accuracy: 0.7009 - 1s/epoch
## Epoch 188/250
## 161/161 - 1s - loss: 2.7784 - accuracy: 0.5549 - val_loss: 1.7359 - val_accuracy: 0.6330 - 1s/epoch
## Epoch 189/250
## 161/161 - 1s - loss: 3.0417 - accuracy: 0.5478 - val_loss: 4.2722 - val_accuracy: 0.4784 - 1s/epoch
## Epoch 190/250
## 161/161 - 1s - loss: 3.0050 - accuracy: 0.5617 - val_loss: 2.7624 - val_accuracy: 0.6200 - 1s/epoch
## Epoch 191/250
## 161/161 - 1s - loss: 2.7295 - accuracy: 0.5635 - val_loss: 2.0281 - val_accuracy: 0.5663 - 1s/epoch
## Epoch 192/250
## 161/161 - 1s - loss: 2.1885 - accuracy: 0.5662 - val_loss: 0.6283 - val_accuracy: 0.7328 - 1s/epoch
## Epoch 193/250
## 161/161 - 1s - loss: 2.3753 - accuracy: 0.5634 - val_loss: 0.7256 - val_accuracy: 0.6734 - 1s/epoch
## Epoch 194/250
## 161/161 - 1s - loss: 3.0416 - accuracy: 0.5557 - val_loss: 3.6215 - val_accuracy: 0.5526 - 1s/epoch
## Epoch 195/250
## 161/161 - 1s - loss: 3.0309 - accuracy: 0.5605 - val_loss: 1.6678 - val_accuracy: 0.6033 - 1s/epoch
## Epoch 196/250
## 161/161 - 1s - loss: 3.2045 - accuracy: 0.5523 - val_loss: 2.5868 - val_accuracy: 0.6411 - 1s/epoch
## Epoch 197/250
## 161/161 - 1s - loss: 2.9934 - accuracy: 0.5577 - val_loss: 1.2568 - val_accuracy: 0.6004 - 1s/epoch
## Epoch 198/250
## 161/161 - 1s - loss: 2.4668 - accuracy: 0.5657 - val_loss: 2.7071 - val_accuracy: 0.6469 - 1s/epoch
## Epoch 199/250
## 161/161 - 1s - loss: 3.2247 - accuracy: 0.5549 - val_loss: 2.1882 - val_accuracy: 0.5663 - 1s/epoch
## Epoch 200/250
## 161/161 - 1s - loss: 3.4658 - accuracy: 0.5499 - val_loss: 0.9333 - val_accuracy: 0.6394 - 1s/epoch
## Epoch 201/250
## 161/161 - 1s - loss: 3.1383 - accuracy: 0.5626 - val_loss: 4.9383 - val_accuracy: 0.6081 - 1s/epoch
## Epoch 202/250
## 161/161 - 1s - loss: 3.3085 - accuracy: 0.5611 - val_loss: 0.5381 - val_accuracy: 0.7987 - 1s/epoch
## Epoch 203/250
## 161/161 - 1s - loss: 2.7159 - accuracy: 0.5572 - val_loss: 2.3859 - val_accuracy: 0.5863 - 1s/epoch
## Epoch 204/250
## 161/161 - 1s - loss: 2.5904 - accuracy: 0.5627 - val_loss: 2.5568 - val_accuracy: 0.5175 - 1s/epoch
## Epoch 205/250
## 161/161 - 1s - loss: 2.9825 - accuracy: 0.5536 - val_loss: 3.4465 - val_accuracy: 0.5580 - 1s/epoch
## Epoch 206/250
## 161/161 - 1s - loss: 2.5015 - accuracy: 0.5688 - val_loss: 3.0385 - val_accuracy: 0.5816 - 1s/epoch
## Epoch 207/250
## 161/161 - 1s - loss: 3.0043 - accuracy: 0.5575 - val_loss: 6.7618 - val_accuracy: 0.4841 - 1s/epoch
## Epoch 208/250
## 161/161 - 1s - loss: 3.3741 - accuracy: 0.5561 - val_loss: 4.0890 - val_accuracy: 0.5004 - 1s/epoch
## Epoch 209/250
## 161/161 - 1s - loss: 3.3939 - accuracy: 0.5536 - val_loss: 0.4563 - val_accuracy: 0.8128 - 1s/epoch
## Epoch 210/250
## 161/161 - 1s - loss: 3.7415 - accuracy: 0.5505 - val_loss: 0.9470 - val_accuracy: 0.6968 - 1s/epoch
## Epoch 211/250
## 161/161 - 1s - loss: 2.9176 - accuracy: 0.5645 - val_loss: 0.9884 - val_accuracy: 0.6753 - 1s/epoch
## Epoch 212/250
## 161/161 - 1s - loss: 3.7990 - accuracy: 0.5501 - val_loss: 3.5326 - val_accuracy: 0.5879 - 1s/epoch
## Epoch 213/250
## 161/161 - 1s - loss: 2.5893 - accuracy: 0.5685 - val_loss: 3.7188 - val_accuracy: 0.5072 - 1s/epoch
## Epoch 214/250
## 161/161 - 1s - loss: 2.7402 - accuracy: 0.5635 - val_loss: 2.0636 - val_accuracy: 0.6192 - 1s/epoch
## Epoch 215/250
## 161/161 - 1s - loss: 3.2286 - accuracy: 0.5535 - val_loss: 3.0481 - val_accuracy: 0.6297 - 1s/epoch
## Epoch 216/250
## 161/161 - 1s - loss: 3.3200 - accuracy: 0.5511 - val_loss: 4.5114 - val_accuracy: 0.6259 - 1s/epoch
## Epoch 217/250
## 161/161 - 1s - loss: 2.3445 - accuracy: 0.5713 - val_loss: 1.4716 - val_accuracy: 0.5791 - 1s/epoch
## Epoch 218/250
## 161/161 - 1s - loss: 3.6024 - accuracy: 0.5484 - val_loss: 3.0335 - val_accuracy: 0.6047 - 1s/epoch
## Epoch 219/250
## 161/161 - 1s - loss: 3.2152 - accuracy: 0.5605 - val_loss: 0.7046 - val_accuracy: 0.7394 - 1s/epoch
## Epoch 220/250
## 161/161 - 1s - loss: 4.2995 - accuracy: 0.5475 - val_loss: 6.7865 - val_accuracy: 0.6049 - 1s/epoch
## Epoch 221/250
## 161/161 - 1s - loss: 3.5883 - accuracy: 0.5584 - val_loss: 1.7666 - val_accuracy: 0.6596 - 1s/epoch
## Epoch 222/250
## 161/161 - 1s - loss: 2.8044 - accuracy: 0.5676 - val_loss: 1.0520 - val_accuracy: 0.6917 - 1s/epoch
## Epoch 223/250
## 161/161 - 1s - loss: 2.7443 - accuracy: 0.5649 - val_loss: 0.6105 - val_accuracy: 0.7627 - 1s/epoch
## Epoch 224/250
## 161/161 - 1s - loss: 3.0362 - accuracy: 0.5578 - val_loss: 7.9842 - val_accuracy: 0.4818 - 1s/epoch
## Epoch 225/250
## 161/161 - 1s - loss: 3.4462 - accuracy: 0.5490 - val_loss: 5.2763 - val_accuracy: 0.6205 - 1s/epoch
## Epoch 226/250
## 161/161 - 1s - loss: 3.9667 - accuracy: 0.5501 - val_loss: 3.4938 - val_accuracy: 0.6321 - 1s/epoch
## Epoch 227/250
## 161/161 - 1s - loss: 3.4992 - accuracy: 0.5583 - val_loss: 3.5137 - val_accuracy: 0.5698 - 1s/epoch
## Epoch 228/250
## 161/161 - 1s - loss: 3.4330 - accuracy: 0.5598 - val_loss: 4.3425 - val_accuracy: 0.5390 - 1s/epoch
## Epoch 229/250
## 161/161 - 1s - loss: 4.6018 - accuracy: 0.5478 - val_loss: 2.9422 - val_accuracy: 0.6506 - 1s/epoch
## Epoch 230/250
## 161/161 - 1s - loss: 3.2162 - accuracy: 0.5617 - val_loss: 3.6082 - val_accuracy: 0.6016 - 1s/epoch
## Epoch 231/250
## 161/161 - 1s - loss: 3.8426 - accuracy: 0.5583 - val_loss: 5.7977 - val_accuracy: 0.4847 - 1s/epoch
## Epoch 232/250
## 161/161 - 1s - loss: 3.7330 - accuracy: 0.5659 - val_loss: 1.4625 - val_accuracy: 0.7080 - 1s/epoch
## Epoch 233/250
## 161/161 - 1s - loss: 3.4048 - accuracy: 0.5591 - val_loss: 2.9542 - val_accuracy: 0.5471 - 1s/epoch
## Epoch 234/250
## 161/161 - 1s - loss: 4.2078 - accuracy: 0.5513 - val_loss: 2.2722 - val_accuracy: 0.5943 - 1s/epoch
## Epoch 235/250
## 161/161 - 1s - loss: 4.5577 - accuracy: 0.5531 - val_loss: 4.8210 - val_accuracy: 0.6109 - 1s/epoch
## Epoch 236/250
## 161/161 - 1s - loss: 3.2832 - accuracy: 0.5591 - val_loss: 2.2442 - val_accuracy: 0.5652 - 1s/epoch
## Epoch 237/250
## 161/161 - 1s - loss: 4.5645 - accuracy: 0.5537 - val_loss: 5.6700 - val_accuracy: 0.5954 - 1s/epoch
## Epoch 238/250
## 161/161 - 1s - loss: 3.3197 - accuracy: 0.5561 - val_loss: 2.0471 - val_accuracy: 0.6620 - 1s/epoch
## Epoch 239/250
## 161/161 - 1s - loss: 3.8058 - accuracy: 0.5540 - val_loss: 3.1533 - val_accuracy: 0.5306 - 1s/epoch
## Epoch 240/250
## 161/161 - 1s - loss: 4.4696 - accuracy: 0.5488 - val_loss: 3.6697 - val_accuracy: 0.5305 - 1s/epoch
## Epoch 241/250
## 161/161 - 1s - loss: 3.0532 - accuracy: 0.5694 - val_loss: 1.0331 - val_accuracy: 0.6698 - 1s/epoch
## Epoch 242/250
## 161/161 - 1s - loss: 3.9214 - accuracy: 0.5567 - val_loss: 2.3960 - val_accuracy: 0.6466 - 1s/epoch
## Epoch 243/250
## 161/161 - 1s - loss: 4.0237 - accuracy: 0.5496 - val_loss: 1.0513 - val_accuracy: 0.6802 - 1s/epoch
## Epoch 244/250
## 161/161 - 1s - loss: 4.4334 - accuracy: 0.5530 - val_loss: 2.2314 - val_accuracy: 0.6409 - 1s/epoch
## Epoch 245/250
## 161/161 - 1s - loss: 4.1108 - accuracy: 0.5585 - val_loss: 4.6551 - val_accuracy: 0.5045 - 1s/epoch
## Epoch 246/250
## 161/161 - 1s - loss: 3.9893 - accuracy: 0.5585 - val_loss: 11.6465 - val_accuracy: 0.4679 - 1s/epoch
## Epoch 247/250
## 161/161 - 1s - loss: 3.6459 - accuracy: 0.5626 - val_loss: 2.7893 - val_accuracy: 0.6254 - 1s/epoch
## Epoch 248/250
## 161/161 - 1s - loss: 5.1838 - accuracy: 0.5521 - val_loss: 0.6600 - val_accuracy: 0.7549 - 1s/epoch
## Epoch 249/250
## 161/161 - 1s - loss: 3.5015 - accuracy: 0.5654 - val_loss: 1.2997 - val_accuracy: 0.6932 - 1s/epoch
## Epoch 250/250
## 161/161 - 1s - loss: 3.4727 - accuracy: 0.5597 - val_loss: 1.5603 - val_accuracy: 0.6391 - 1s/epoch
```

```r
loss_valloss_difference_1c <- "The training loss indicates how well the model is fitting the training da
```

Task 1d)

In Lecture 9 we used early stopping to avoid overfitting. Apply this here, with `patience` argument set to 20, to a new model `model1b` which otherwise should have a setup identical to `model1`. In which epoch did the model training procedure stop? Figure this out by counting the number of elements in `$model1b.fit$metrics$loss$` and write your answer into the string variable `when_earlystop_1d`.

```r
# # Note: Do NOT assign model1b <- model1 this just creates a link to the existing model1.
model1b <-  clone_model(model1)

model1b %>%
  compile(
    loss = 'binary_crossentropy',
    optimizer = optimizer_adam(lr = 0.00001),
    metrics = c('accuracy')
  )

early_stop <- callback_early_stopping(patience = 20)

model1b_fit <- model1b %>%
  fit(
    X_train, y_train,
    epochs = 250,
    batch_size = 2^8,
    validation_split = 0.25,
    callbacks = list(early_stop)
  )
```

```
## Epoch 1/250
## 161/161 - 2s - loss: 0.7256 - accuracy: 0.5975 - val_loss: 0.5433 - val_accuracy: 0.7868 - 2s/epoch
## Epoch 2/250
## 161/161 - 1s - loss: 0.6396 - accuracy: 0.6453 - val_loss: 0.5321 - val_accuracy: 0.7839 - 1s/epoch
## Epoch 3/250
## 161/161 - 1s - loss: 0.6336 - accuracy: 0.6489 - val_loss: 0.5328 - val_accuracy: 0.7791 - 1s/epoch
## Epoch 4/250
## 161/161 - 1s - loss: 0.6316 - accuracy: 0.6508 - val_loss: 0.5311 - val_accuracy: 0.7820 - 1s/epoch
## Epoch 5/250
## 161/161 - 1s - loss: 0.6304 - accuracy: 0.6521 - val_loss: 0.5449 - val_accuracy: 0.7579 - 1s/epoch
## Epoch 6/250
## 161/161 - 1s - loss: 0.6312 - accuracy: 0.6477 - val_loss: 0.5452 - val_accuracy: 0.7410 - 1s/epoch
## Epoch 7/250
## 161/161 - 1s - loss: 0.6329 - accuracy: 0.6430 - val_loss: 0.5683 - val_accuracy: 0.7381 - 1s/epoch
## Epoch 8/250
## 161/161 - 1s - loss: 0.6345 - accuracy: 0.6425 - val_loss: 0.5485 - val_accuracy: 0.7564 - 1s/epoch
## Epoch 9/250
## 161/161 - 1s - loss: 0.6369 - accuracy: 0.6404 - val_loss: 0.5381 - val_accuracy: 0.7801 - 1s/epoch
## Epoch 10/250
## 161/161 - 1s - loss: 0.6390 - accuracy: 0.6384 - val_loss: 0.5262 - val_accuracy: 0.7901 - 1s/epoch
## Epoch 11/250
## 161/161 - 1s - loss: 0.6426 - accuracy: 0.6320 - val_loss: 0.5326 - val_accuracy: 0.7902 - 1s/epoch
## Epoch 12/250
## 161/161 - 1s - loss: 0.6456 - accuracy: 0.6296 - val_loss: 0.5507 - val_accuracy: 0.7765 - 1s/epoch
## Epoch 13/250
## 161/161 - 1s - loss: 0.6512 - accuracy: 0.6287 - val_loss: 0.5524 - val_accuracy: 0.7787 - 1s/epoch
## Epoch 14/250
## 161/161 - 1s - loss: 0.6533 - accuracy: 0.6228 - val_loss: 0.5160 - val_accuracy: 0.8282 - 1s/epoch
## Epoch 15/250
## 161/161 - 1s - loss: 0.6569 - accuracy: 0.6206 - val_loss: 0.5606 - val_accuracy: 0.7648 - 1s/epoch
## Epoch 16/250
## 161/161 - 1s - loss: 0.6570 - accuracy: 0.6243 - val_loss: 0.5219 - val_accuracy: 0.8158 - 1s/epoch
## Epoch 17/250
## 161/161 - 1s - loss: 0.6611 - accuracy: 0.6152 - val_loss: 0.6020 - val_accuracy: 0.6663 - 1s/epoch
## Epoch 18/250
## 161/161 - 1s - loss: 0.6655 - accuracy: 0.6167 - val_loss: 0.5694 - val_accuracy: 0.7443 - 1s/epoch
## Epoch 19/250
## 161/161 - 1s - loss: 0.6693 - accuracy: 0.6073 - val_loss: 0.5335 - val_accuracy: 0.7623 - 1s/epoch
## Epoch 20/250
## 161/161 - 1s - loss: 0.6738 - accuracy: 0.6056 - val_loss: 0.5416 - val_accuracy: 0.7785 - 1s/epoch
## Epoch 21/250
## 161/161 - 1s - loss: 0.6589 - accuracy: 0.6216 - val_loss: 0.5426 - val_accuracy: 0.7882 - 1s/epoch
## Epoch 22/250
## 161/161 - 1s - loss: 0.6699 - accuracy: 0.6110 - val_loss: 0.5173 - val_accuracy: 0.8176 - 1s/epoch
## Epoch 23/250
## 161/161 - 1s - loss: 0.6781 - accuracy: 0.6057 - val_loss: 0.5896 - val_accuracy: 0.6747 - 1s/epoch
## Epoch 24/250
## 161/161 - 1s - loss: 0.6988 - accuracy: 0.5955 - val_loss: 0.4970 - val_accuracy: 0.8653 - 1s/epoch
## Epoch 25/250
## 161/161 - 1s - loss: 0.6687 - accuracy: 0.6098 - val_loss: 0.5480 - val_accuracy: 0.7339 - 1s/epoch
## Epoch 26/250
## 161/161 - 1s - loss: 0.6879 - accuracy: 0.6000 - val_loss: 0.5194 - val_accuracy: 0.7940 - 1s/epoch
## Epoch 27/250
## 161/161 - 1s - loss: 0.6832 - accuracy: 0.6035 - val_loss: 0.6841 - val_accuracy: 0.6406 - 1s/epoch
## Epoch 28/250
## 161/161 - 1s - loss: 0.6839 - accuracy: 0.6062 - val_loss: 0.5234 - val_accuracy: 0.7764 - 1s/epoch
## Epoch 29/250
## 161/161 - 1s - loss: 0.7214 - accuracy: 0.5914 - val_loss: 0.5342 - val_accuracy: 0.7828 - 1s/epoch
## Epoch 30/250
## 161/161 - 1s - loss: 0.6787 - accuracy: 0.6090 - val_loss: 0.5191 - val_accuracy: 0.8060 - 1s/epoch
## Epoch 31/250
## 161/161 - 1s - loss: 0.6837 - accuracy: 0.6035 - val_loss: 0.5398 - val_accuracy: 0.7577 - 1s/epoch
## Epoch 32/250
## 161/161 - 1s - loss: 0.6971 - accuracy: 0.5968 - val_loss: 0.9651 - val_accuracy: 0.5385 - 1s/epoch
## Epoch 33/250
## 161/161 - 1s - loss: 0.7461 - accuracy: 0.5865 - val_loss: 0.5073 - val_accuracy: 0.8012 - 1s/epoch
## Epoch 34/250
## 161/161 - 1s - loss: 0.7065 - accuracy: 0.5985 - val_loss: 0.6269 - val_accuracy: 0.6693 - 1s/epoch
## Epoch 35/250
## 161/161 - 1s - loss: 0.7625 - accuracy: 0.5817 - val_loss: 0.6281 - val_accuracy: 0.6695 - 1s/epoch
## Epoch 36/250
## 161/161 - 1s - loss: 0.7128 - accuracy: 0.5964 - val_loss: 0.5652 - val_accuracy: 0.7405 - 1s/epoch
## Epoch 37/250
## 161/161 - 1s - loss: 0.6925 - accuracy: 0.6069 - val_loss: 0.5567 - val_accuracy: 0.7083 - 1s/epoch
## Epoch 38/250
## 161/161 - 1s - loss: 0.7610 - accuracy: 0.5866 - val_loss: 0.6598 - val_accuracy: 0.6558 - 1s/epoch
## Epoch 39/250
## 161/161 - 1s - loss: 0.7618 - accuracy: 0.5871 - val_loss: 0.5218 - val_accuracy: 0.8226 - 1s/epoch
## Epoch 40/250
## 161/161 - 1s - loss: 0.7552 - accuracy: 0.5924 - val_loss: 0.7682 - val_accuracy: 0.6290 - 1s/epoch
## Epoch 41/250
## 161/161 - 1s - loss: 0.7749 - accuracy: 0.5846 - val_loss: 0.6888 - val_accuracy: 0.6710 - 1s/epoch
## Epoch 42/250
## 161/161 - 1s - loss: 0.7773 - accuracy: 0.5812 - val_loss: 0.5560 - val_accuracy: 0.7117 - 1s/epoch
## Epoch 43/250
## 161/161 - 1s - loss: 0.7883 - accuracy: 0.5829 - val_loss: 0.5602 - val_accuracy: 0.7331 - 1s/epoch
## Epoch 44/250
## 161/161 - 1s - loss: 0.7174 - accuracy: 0.5969 - val_loss: 0.5312 - val_accuracy: 0.7912 - 1s/epoch
```

```r
when_earlystop_1d <- length(model1b_fit$metrics$loss)

when_earlystop_1d <- "66"
```

Task 1e)

Even though we haven't finished training our neural net, let us use the `evaluate()` function to measure the predictive performance of `model1b` on the test data. Save the result as `res_model1`.

What is the accuracy of the model for validation training data and test data, respectively? What is the difference in accuracy? Save your answer in the string variable `difference_in_accuracy_1e`. *Hint:* the training validation accuracy can be extracted from `model1b_fit`.

```r
res_model1 <- model1b %>%
  evaluate(X_test, y_test)
```

```
## 572/572 - 2s - loss: 0.6268 - accuracy: 0.6797 - 2s/epoch - 4ms/step
```

```r
res_model1
```

```
##      loss  accuracy
## 0.6267844 0.6796871
```

```r
accuracy_validation <- model1b_fit$metrics$val_accuracy[length(model1b_fit$metrics$val_accuracy)]
accuracy_validation
```

```
## [1] 0.7911591
```

```r
difference_in_accuracy_1e <- "The accuracy for the test data is 0.67 which comes from using the evaluat
```

Task 1f)

Now we use the `confusionMatrix()` command from the `caret` library to disaggregate model performance to class-specific performance. First, predict output categories on the test data using `predict()`. Save these as `pred_model1`. Then use class predictions based on a threshold probability of 50% as inputs to `confusionMatrix`. Save your confusion matrix as `CM_model1`. Do your results on sensitivity and specificity suggest that prediction accuracy is approximately equal in both categories? Write your (specific!) answer into the string variable `categorywise_accuracy_1f`

```
pred_model1 <- model1b %>%
  predict(X_test) %>%
  as.numeric() > 0.5
```

```
## 572/572 - 1s - 983ms/epoch - 2ms/step
```

```
pred_model1_adjusted <- factor(pred_model1, levels = c(FALSE, TRUE), labels = c(0, 1))


CM_model1 <- confusionMatrix(as.factor(pred_model1_adjusted), as.factor(y_test))

#print(CM_model1$table)
#print(CM_model1$byClass)
CM_model1
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 10588  3562
##          1  2293  1836
##
##                Accuracy : 0.6797
##                  95% CI : (0.6729, 0.6864)
##     No Information Rate : 0.7047
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.174
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.8220
##             Specificity : 0.3401
##          Pos Pred Value : 0.7483
##          Neg Pred Value : 0.4447
##              Prevalence : 0.7047
##          Detection Rate : 0.5792
##    Detection Prevalence : 0.7741
##       Balanced Accuracy : 0.5811
##
##        'Positive' Class : 0
##
```

```
categorywise_accuracy_1f <- "The sensitivity values suggest that prediction accuracy is not approximate
```

Task 1g)

In the lectures we have utilized explicit regularization to avoid overfitting. Here we will use ℓ_2 regularization to update the weights. Create the architecture of a new neural net `model2` which is identical to that of `model1b` except for ℓ_2 regularization with regularization factor `l2_pen` in the two hidden layers.

Then compile and fit this regularized model with the same parameters as in Task 1d. Save the trained neural net as `model2_fit`.

```
# # 1.
model2 <- keras_model_sequential() %>%
  layer_dense(units = 30, activation = 'relu', kernel_regularizer = regularizer_l2(l = 0.01), input_shap
  layer_dense(units = 15, activation = 'relu', kernel_regularizer = regularizer_l2(l = 0.01)) %>%
  layer_dense(units = 1, activation = 'sigmoid')

model2 %>%
  compile(
    loss = 'binary_crossentropy',
    optimizer = optimizer_adam(lr = 0.00001),
    metrics = c('accuracy')
  )

# # 2.
 model2_fit <- model2 %>%
  fit(
    X_train, y_train,
    epochs = 250,
    batch_size = 2^8,
    validation_split = 0.25,
    callbacks = list(early_stop)
  )
```

```
## Epoch 1/250
## 161/161 - 3s - loss: 1.1116 - accuracy: 0.5866 - val_loss: 0.7361 - val_accuracy: 0.8803 - 3s/epoch
## Epoch 2/250
## 161/161 - 3s - loss: 0.7796 - accuracy: 0.6338 - val_loss: 0.6214 - val_accuracy: 0.8511 - 3s/epoch
## Epoch 3/250
## 161/161 - 2s - loss: 0.7078 - accuracy: 0.6443 - val_loss: 0.6066 - val_accuracy: 0.8094 - 2s/epoch
## Epoch 4/250
## 161/161 - 3s - loss: 0.6951 - accuracy: 0.6487 - val_loss: 0.6239 - val_accuracy: 0.7987 - 3s/epoch
## Epoch 5/250
## 161/161 - 3s - loss: 0.7184 - accuracy: 0.6510 - val_loss: 0.6507 - val_accuracy: 0.8281 - 3s/epoch
## Epoch 6/250
## 161/161 - 3s - loss: 0.7944 - accuracy: 0.6513 - val_loss: 0.7702 - val_accuracy: 0.7952 - 3s/epoch
## Epoch 7/250
## 161/161 - 2s - loss: 0.9366 - accuracy: 0.6514 - val_loss: 0.9657 - val_accuracy: 0.7755 - 2s/epoch
## Epoch 8/250
## 161/161 - 3s - loss: 1.1448 - accuracy: 0.6509 - val_loss: 1.2045 - val_accuracy: 0.7755 - 3s/epoch
## Epoch 9/250
## 161/161 - 3s - loss: 1.4239 - accuracy: 0.6514 - val_loss: 1.5261 - val_accuracy: 0.7469 - 3s/epoch
## Epoch 10/250
## 161/161 - 2s - loss: 1.7545 - accuracy: 0.6518 - val_loss: 1.8913 - val_accuracy: 0.7356 - 2s/epoch
## Epoch 11/250
## 161/161 - 2s - loss: 2.1379 - accuracy: 0.6478 - val_loss: 2.2769 - val_accuracy: 0.7794 - 2s/epoch
## Epoch 12/250
## 161/161 - 2s - loss: 2.5800 - accuracy: 0.6433 - val_loss: 2.7594 - val_accuracy: 0.7308 - 2s/epoch
## Epoch 13/250
## 161/161 - 3s - loss: 3.0408 - accuracy: 0.6447 - val_loss: 3.1792 - val_accuracy: 0.8279 - 3s/epoch
## Epoch 14/250
## 161/161 - 2s - loss: 3.5337 - accuracy: 0.6469 - val_loss: 3.7368 - val_accuracy: 0.7472 - 2s/epoch
## Epoch 15/250
## 161/161 - 2s - loss: 4.0608 - accuracy: 0.6475 - val_loss: 4.2235 - val_accuracy: 0.8119 - 2s/epoch
## Epoch 16/250
## 161/161 - 3s - loss: 4.5861 - accuracy: 0.6477 - val_loss: 4.7567 - val_accuracy: 0.8489 - 3s/epoch
## Epoch 17/250
## 161/161 - 3s - loss: 5.1685 - accuracy: 0.6483 - val_loss: 5.4194 - val_accuracy: 0.7348 - 3s/epoch
## Epoch 18/250
## 161/161 - 2s - loss: 5.7802 - accuracy: 0.6402 - val_loss: 6.0562 - val_accuracy: 0.7397 - 2s/epoch
## Epoch 19/250
## 161/161 - 3s - loss: 6.4060 - accuracy: 0.6466 - val_loss: 6.6162 - val_accuracy: 0.8192 - 3s/epoch
## Epoch 20/250
## 161/161 - 2s - loss: 7.0461 - accuracy: 0.6463 - val_loss: 7.2778 - val_accuracy: 0.7949 - 2s/epoch
## Epoch 21/250
## 161/161 - 3s - loss: 7.7235 - accuracy: 0.6389 - val_loss: 7.9444 - val_accuracy: 0.8018 - 3s/epoch
## Epoch 22/250
## 161/161 - 2s - loss: 8.3703 - accuracy: 0.6489 - val_loss: 8.5992 - val_accuracy: 0.8084 - 2s/epoch
## Epoch 23/250
## 161/161 - 2s - loss: 9.0741 - accuracy: 0.6374 - val_loss: 9.4010 - val_accuracy: 0.6634 - 2s/epoch
```

Task 1h)

In Task 1e) we compared the prediction accuracy on test and training sets. However, this is bad measure when data is not well balanced (similar number of true and false). Instead one can use the cross entropy for the binomial distribution (minus the average log likelihood of the model). In fact, we chose this function as loss function for model training when we compiled `model1` and `model2`.

To compare the test error of `model2` to that of `model1b` we don't want to use `loss` from `evaluate` since this includes the ℓ_2 penalty. In the library `MLmetrics` the function `LogLoss()` computes the cross entropy for the binomial distribution without penalty term.

First, get test data predictions from `model2` and save them as `pred_model2`. Second, use `LogLoss()` to compute the cross-entropy loss for `model2` on the test data and save it as `logloss_model2`. Read the help file for `LogLoss()` if needed. Third, use the string variable `performance_comparison_1h` to describe how the accuracy on test data differs between `model1b` and `model2`.

```r
# install.packages("MLmetrics")
library(MLmetrics)
```

```
##
## Attaching package: 'MLmetrics'

## The following objects are masked from 'package:caret':
##
##     MAE, RMSE

## The following object is masked from 'package:base':
##
##     Recall
```

```r
#
# # 1.
prob_model2 <- model2 %>%
  predict(X_test)
```

```
## 572/572 - 1s - 956ms/epoch - 2ms/step
```

```r
# # 2.
logloss_model2 <- LogLoss(prob_model2, y_test)
print(logloss_model2)
```

```
## [1] 0.6431511
```

```r
# # 3.
performance_comparison_1h <- "The log loss of model2 is lower than model1b, indicating better performan
```

Part 2: Tuning neural nets with caret

Keras does not have any built-in routines for cross-validation. Luckily, we can use `caret` for this purpose. One possible way would be to choose a method for `train()` that fits neural nets with `keras`. However, these built-in methods in `caret` do not give us full control over the model architecture and all aspects of model training. For example, we cannot use early stopping and there is no support for ℓ_1- regularization or a combination of ℓ_1 and ℓ_2 penalties.

For this reason, we will supply `train()` a custom method. This can be done by specifying a *list* object which contains functions for model fitting and predictions as well as the definitions of our tuning parameters. The structure of these list objects is described in the caret documentation.

We will not create an entire custom list object from scratch. Instead, we extract and modify the list object for one of the built-in keras-based methods in `caret`. When modifying the list object, we need to change two particular things:

1. Supply a function for model training that uses our desired neural network,
2. Change tuning parameter definitions to the list of parameters that we care about.

Task 2a)

First, create a function `fitfun_2a` that specifies and fits our neural net. Below, I have prepared a fragment of this function. We want the function to train the neural net from Task 1g. Do the following:

1. Copy and paste your code from Task 1g (model architecture, compiling and training) into the code chunk below.
2. Change the pasted code chunk so that the model uses inputs `x` and outputs `y` for training. Replace the previous regularization parameter value `l2_pen` with a reference to an object `l2_pen` *inside* the list object of tuning parameters `param`.
3. Change the code so that your learned model is saved as an object `model`.
4. Add `keras::` before all functions from the `keras` library that you use in the code block below. This ensures that you can train your neural net without having explicitly loaded the `keras` library.
5. Change the loss function used for model training to `categorical_crossentropy`. Change the number of output units in the output layer of your neural net to the number of columns in `y`.

```
fitfun_2a <- function(x, y, wts, param, lev, last, classProbs, ...) {
  # Do not change the six rows below
  require(dplyr)
  K <- keras::backend()
  K$clear_session()
  if (!is.matrix(x))
    x <- as.matrix(x)
  y <- class2ind(y)

  model <- keras::keras_model_sequential() %>%
    layer_dense(units = 30, activation = 'relu', input_shape = ncol(x),
                kernel_regularizer = regularizer_l2(l = 0.01)) %>%
    layer_dense(units = 15, activation = 'relu', kernel_regularizer = regularizer_l2(l = 0.01)) %>%
    layer_dense(units = dim(y)[2], activation = 'sigmoid')

  model %>% compile(
    loss = 'binary_crossentropy',
    optimizer = optimizer_adam(lr = 0.00001),
    metrics = list('accuracy')
  )

  model %>% fit(
```

```r
    x = x,
    y = y,
    epochs = 250,
    batch_size = 2^8,
    validation_split = 0.25
  )

  # Do not change the three rows below
  if (last)
    model <- keras::serialize_model(model)
  list(object = model)
}
```

Task 2b)

Next, we define the tuning parameters of our neural net. This requires us to create a data frame `def_params_2b` with three variables whose names are *parameter*, *class* and *label*.

Our neural net has only one tuning parameter: The degree of ℓ_2 regularization. Therefore, there should be only one data point in `def_params_2b` which tells us that our tuning parameter is `"l2_pen"`, that it is `"numeric"`and that we would label it as a `"Regularization parameter"`.

Now create a data frame `def_params_2b` that contains the information specified above.

```r
def_params_2b <- data.frame(
  parameter = "l2_pen",
  class = "numeric",
  label = "Regularization parameter"
)
```

Task 2c)

Given that our tuning parameters are now defined, we have to specify a default grid of candidate values. I prepared the fragment of a function `defaultgrid_2c` below that you are now asked to complete.

If the value of the `search` argument is `grid`, define a data frame `out` whose only variable `l2_pen` contains zero as well as 10^r for a grid of r-values -4 and -1. In total, `out` should contain `len` data points.

If the values of the `search` argument is *not* `grid`, draw `len` random numbers from a uniform distribution between `-5` and `1` and let 10 to the power of these numbers be your tuning parameter candidates. These candidates are then saved in the data frame `out` as values of your `l2_pen` variable.

```r
defaultgrid_2c <- function (x, y, len = 12, search = "grid") {
  if (search == "grid") {
    out <- data.frame(l2_pen = 10^seq(-4, -1, length.out = len))
  } else {
    out <- data.frame(l2_pen = 10^runif(len, -5, 1))
  }
  out
}

defaultgrid_2c(X_train, y_train)
```

```
##           l2_pen
## 1   0.0001000000
## 2   0.0001873817
## 3   0.0003511192
## 4  0.0006579332
## 5  0.0012328467
## 6  0.0023101297
## 7  0.0043287613
## 8  0.0081113083
## 9  0.0151991108
## 10 0.0284803587
## 11 0.0533669923
## 12 0.1000000000
```

Task 2d)

The last new function we need to write specifies how the cross-validation results need to be ordered in terms of the tuning parameter values. Write a function `result_sort_2d` which takes one argument `res_raw` and which rearranges the rows of `res_raw` according to the values of `res_raw$l2_pen` in *increasing* order.

Note: The `order()` function in R tells you how the indexes of an array have to be rearranged in order to sort its elements in either decreasing or increasing order.

```r
result_sort_2d <- function(res_raw) {
  res_raw[order(res_raw$l2_pen), ]
}
```

Task 2e)

Now we can create our custom model specification object for `caret`. Below, I have written code that extracts the model specification of the `mlpKerasDecay` method in `caret` into an list `modelinfo_custom_2e`. Replace its `fit`, `parameters`, `grid` and `sort` objects with your corresponding functions from Tasks 2a-d.

```r
modelinfo_custom_2e <- getModelInfo("mlpKerasDecay")
modelinfo_custom_2e <- modelinfo_custom_2e$mlpKerasDecay

modelinfo_custom_2e$fit <- fitfun_2a
modelinfo_custom_2e$parameters <- def_params_2b
modelinfo_custom_2e$grid <- defaultgrid_2c
modelinfo_custom_2e$sort <- result_sort_2d
```

Task 2f)

Next, set up the training controls for `caret` and save them as `train_control_2f`. We want to do 5-fold cross-validation. Additionally, we want to use a parallel backend to speed up our grid search.

```r
train_control_2f <- trainControl(
  method = "none",
  verboseIter = TRUE,
  classProbs = TRUE,
  summaryFunction = twoClassSummary,
  savePredictions = TRUE
)
```

Task 2g)

Lastly, we can run `train()` to tune our model. That's computationally quite costly, so we will use merely a fraction of the available training data. The inputs and outputs for this task are given by

```
train_small   <- sample(length(y_train)[1],ceiling(0.3*length(y_train)))
X_train_small <- X_train[train_small,]
y_train_small <- y_train[train_small]
```

Now use cross-validation to find your optimal regularization parameter value and save the resulting object as `tune_NN_2g`. When specifying the arguments of `train()`, ensure the following:

1. Use `X_train_small` and `y_train_small` as input and output data, respectively.
2. Turn `y_train_small` into a factor variable. `train()` cannot handle logical output variables.
3. Feed your custom model specification `train_control_2c` as `method`-argument.
4. Do not choose a custom grid of tuning parameter values. However, state that you want to use the default grid with a length of 12 tuning parameter candidates.

Note: Running the code may take up to 30 minutes.

```
library(doParallel)
```

```
## Loading required package: foreach

## Loading required package: iterators

## Loading required package: parallel
```

```
y_train_small <- ifelse(y_train_small == 0, "not bud light", "bud light")
y_train_small <- factor(y_train_small, levels = c("not bud light", "bud light"))
levels(y_train_small) <- make.names(levels(y_train_small))
tuneGrid_single <- data.frame(l2_pen = 0.001)

tune_NN_2g <- train(
   x = X_train_small,
   y = y_train_small,
   method = modelinfo_custom_2e,
   tuneGrid = tuneGrid_single,
   metric = "Accuarcy",
   trControl = train_control_2f,
   verbose = FALSE
  )
```

```
## Fitting l2_pen = 0.001 on full training set

## Loading required package: dplyr

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Epoch 1/250
## 49/49 - 1s - loss: 1.3968 - accuracy: 0.5608 - val_loss: 1.1521 - val_accuracy: 0.6414 - 1s/epoch - 
## Epoch 2/250
## 49/49 - 1s - loss: 1.0306 - accuracy: 0.6747 - val_loss: 0.9283 - val_accuracy: 0.6963 - 827ms/epoch
## Epoch 3/250
## 49/49 - 1s - loss: 0.8603 - accuracy: 0.7056 - val_loss: 0.8046 - val_accuracy: 0.7129 - 778ms/epoch
## Epoch 4/250
## 49/49 - 1s - loss: 0.7635 - accuracy: 0.7133 - val_loss: 0.7321 - val_accuracy: 0.7136 - 798ms/epoch
## Epoch 5/250
## 49/49 - 1s - loss: 0.7053 - accuracy: 0.7143 - val_loss: 0.6871 - val_accuracy: 0.7102 - 766ms/epoch
## Epoch 6/250
## 49/49 - 1s - loss: 0.6684 - accuracy: 0.7131 - val_loss: 0.6588 - val_accuracy: 0.7107 - 770ms/epoch
## Epoch 7/250
## 49/49 - 1s - loss: 0.6435 - accuracy: 0.7142 - val_loss: 0.6395 - val_accuracy: 0.7109 - 769ms/epoch
## Epoch 8/250
## 49/49 - 1s - loss: 0.6268 - accuracy: 0.7136 - val_loss: 0.6263 - val_accuracy: 0.7121 - 775ms/epoch
## Epoch 9/250
## 49/49 - 1s - loss: 0.6166 - accuracy: 0.7124 - val_loss: 0.6184 - val_accuracy: 0.7114 - 769ms/epoch
## Epoch 10/250
## 49/49 - 1s - loss: 0.6096 - accuracy: 0.7098 - val_loss: 0.6136 - val_accuracy: 0.7112 - 775ms/epoch
## Epoch 11/250
## 49/49 - 1s - loss: 0.6057 - accuracy: 0.7102 - val_loss: 0.6116 - val_accuracy: 0.7061 - 777ms/epoch
## Epoch 12/250
## 49/49 - 1s - loss: 0.6030 - accuracy: 0.7071 - val_loss: 0.6098 - val_accuracy: 0.7063 - 785ms/epoch
## Epoch 13/250
## 49/49 - 1s - loss: 0.6020 - accuracy: 0.7089 - val_loss: 0.6091 - val_accuracy: 0.7075 - 770ms/epoch
## Epoch 14/250
## 49/49 - 1s - loss: 0.6020 - accuracy: 0.7090 - val_loss: 0.6087 - val_accuracy: 0.7075 - 775ms/epoch
## Epoch 15/250
## 49/49 - 1s - loss: 0.6025 - accuracy: 0.7111 - val_loss: 0.6107 - val_accuracy: 0.7046 - 801ms/epoch
## Epoch 16/250
## 49/49 - 1s - loss: 0.6048 - accuracy: 0.7086 - val_loss: 0.6133 - val_accuracy: 0.7090 - 772ms/epoch
## Epoch 17/250
## 49/49 - 1s - loss: 0.6072 - accuracy: 0.7064 - val_loss: 0.6177 - val_accuracy: 0.7082 - 786ms/epoch
## Epoch 18/250
## 49/49 - 1s - loss: 0.6112 - accuracy: 0.7077 - val_loss: 0.6219 - val_accuracy: 0.7056 - 780ms/epoch
## Epoch 19/250
## 49/49 - 1s - loss: 0.6180 - accuracy: 0.7090 - val_loss: 0.6295 - val_accuracy: 0.7070 - 775ms/epoch
## Epoch 20/250
## 49/49 - 1s - loss: 0.6249 - accuracy: 0.7051 - val_loss: 0.6365 - val_accuracy: 0.7034 - 789ms/epoch
## Epoch 21/250
## 49/49 - 1s - loss: 0.6321 - accuracy: 0.7062 - val_loss: 0.6441 - val_accuracy: 0.7075 - 780ms/epoch
## Epoch 22/250
## 49/49 - 1s - loss: 0.6418 - accuracy: 0.7073 - val_loss: 0.6557 - val_accuracy: 0.7046 - 781ms/epoch
## Epoch 23/250
## 49/49 - 1s - loss: 0.6558 - accuracy: 0.7051 - val_loss: 0.6693 - val_accuracy: 0.7048 - 782ms/epoch
## Epoch 24/250
## 49/49 - 1s - loss: 0.6673 - accuracy: 0.7092 - val_loss: 0.6863 - val_accuracy: 0.7107 - 782ms/epoch
## Epoch 25/250
## 49/49 - 1s - loss: 0.6845 - accuracy: 0.7068 - val_loss: 0.6992 - val_accuracy: 0.7053 - 786ms/epoch
## Epoch 26/250
## 49/49 - 1s - loss: 0.7021 - accuracy: 0.7071 - val_loss: 0.7187 - val_accuracy: 0.7070 - 785ms/epoch
## Epoch 27/250
## 49/49 - 1s - loss: 0.7222 - accuracy: 0.7067 - val_loss: 0.7407 - val_accuracy: 0.7073 - 793ms/epoch
## Epoch 28/250
## 49/49 - 1s - loss: 0.7440 - accuracy: 0.7078 - val_loss: 0.7630 - val_accuracy: 0.7075 - 798ms/epoch
## Epoch 29/250
## 49/49 - 1s - loss: 0.7698 - accuracy: 0.7072 - val_loss: 0.7908 - val_accuracy: 0.7061 - 778ms/epoch
## Epoch 30/250
## 49/49 - 1s - loss: 0.7924 - accuracy: 0.7077 - val_loss: 0.8140 - val_accuracy: 0.7078 - 799ms/epoch
## Epoch 31/250
## 49/49 - 1s - loss: 0.8190 - accuracy: 0.7082 - val_loss: 0.8417 - val_accuracy: 0.7114 - 786ms/epoch
## Epoch 32/250
## 49/49 - 1s - loss: 0.8459 - accuracy: 0.7081 - val_loss: 0.8706 - val_accuracy: 0.7048 - 771ms/epoch
## Epoch 33/250
## 49/49 - 1s - loss: 0.8789 - accuracy: 0.7059 - val_loss: 0.9034 - val_accuracy: 0.7041 - 773ms/epoch
## Epoch 34/250
## 49/49 - 1s - loss: 0.9149 - accuracy: 0.7097 - val_loss: 0.9442 - val_accuracy: 0.7102 - 777ms/epoch
## Epoch 35/250
## 49/49 - 1s - loss: 0.9513 - accuracy: 0.7080 - val_loss: 0.9825 - val_accuracy: 0.7095 - 786ms/epoch
## Epoch 36/250
## 49/49 - 1s - loss: 0.9918 - accuracy: 0.7071 - val_loss: 1.0297 - val_accuracy: 0.7095 - 771ms/epoch
## Epoch 37/250
## 49/49 - 1s - loss: 1.0345 - accuracy: 0.7083 - val_loss: 1.0667 - val_accuracy: 0.7041 - 778ms/epoch
## Epoch 38/250
## 49/49 - 1s - loss: 1.0828 - accuracy: 0.7098 - val_loss: 1.1165 - val_accuracy: 0.7051 - 776ms/epoch
## Epoch 39/250
## 49/49 - 1s - loss: 1.1268 - accuracy: 0.7072 - val_loss: 1.1698 - val_accuracy: 0.7129 - 782ms/epoch
## Epoch 40/250
## 49/49 - 1s - loss: 1.1830 - accuracy: 0.7111 - val_loss: 1.2161 - val_accuracy: 0.7073 - 775ms/epoch
## Epoch 41/250
## 49/49 - 1s - loss: 1.2359 - accuracy: 0.7099 - val_loss: 1.2727 - val_accuracy: 0.7056 - 807ms/epoch
## Epoch 42/250
## 49/49 - 1s - loss: 1.2915 - accuracy: 0.7080 - val_loss: 1.3413 - val_accuracy: 0.7102 - 783ms/epoch
## Epoch 43/250
## 49/49 - 1s - loss: 1.3547 - accuracy: 0.7089 - val_loss: 1.3955 - val_accuracy: 0.7095 - 776ms/epoch
## Epoch 44/250
## 49/49 - 1s - loss: 1.4184 - accuracy: 0.7101 - val_loss: 1.4630 - val_accuracy: 0.7085 - 794ms/epoch
## Epoch 45/250
## 49/49 - 1s - loss: 1.4914 - accuracy: 0.7077 - val_loss: 1.5334 - val_accuracy: 0.7002 - 768ms/epoch
## Epoch 46/250
## 49/49 - 1s - loss: 1.5538 - accuracy: 0.7074 - val_loss: 1.6007 - val_accuracy: 0.7063 - 776ms/epoch
## Epoch 47/250
## 49/49 - 1s - loss: 1.6354 - accuracy: 0.7092 - val_loss: 1.6872 - val_accuracy: 0.7051 - 784ms/epoch
## Epoch 48/250
## 49/49 - 1s - loss: 1.7041 - accuracy: 0.7098 - val_loss: 1.7559 - val_accuracy: 0.7068 - 767ms/epoch
## Epoch 49/250
## 49/49 - 1s - loss: 1.7728 - accuracy: 0.7105 - val_loss: 1.8585 - val_accuracy: 0.6978 - 781ms/epoch
## Epoch 50/250
## 49/49 - 1s - loss: 1.8456 - accuracy: 0.7100 - val_loss: 1.8951 - val_accuracy: 0.7061 - 778ms/epoch
## Epoch 51/250
## 49/49 - 1s - loss: 1.9270 - accuracy: 0.7093 - val_loss: 1.9813 - val_accuracy: 0.7065 - 783ms/epoch
## Epoch 52/250
## 49/49 - 1s - loss: 2.0243 - accuracy: 0.7111 - val_loss: 2.0699 - val_accuracy: 0.7039 - 786ms/epoch
## Epoch 53/250
## 49/49 - 1s - loss: 2.0954 - accuracy: 0.7104 - val_loss: 2.1774 - val_accuracy: 0.7075 - 768ms/epoch
## Epoch 54/250
## 49/49 - 1s - loss: 2.1862 - accuracy: 0.7083 - val_loss: 2.2480 - val_accuracy: 0.7126 - 820ms/epoch
## Epoch 55/250
## 49/49 - 1s - loss: 2.2677 - accuracy: 0.7102 - val_loss: 2.3294 - val_accuracy: 0.7022 - 788ms/epoch
## Epoch 56/250
## 49/49 - 1s - loss: 2.3646 - accuracy: 0.7088 - val_loss: 2.4272 - val_accuracy: 0.7058 - 783ms/epoch
## Epoch 57/250
## 49/49 - 1s - loss: 2.4653 - accuracy: 0.7127 - val_loss: 2.5276 - val_accuracy: 0.7022 - 815ms/epoch
## Epoch 58/250
## 49/49 - 1s - loss: 2.5743 - accuracy: 0.7123 - val_loss: 2.6243 - val_accuracy: 0.7075 - 782ms/epoch
## Epoch 59/250
## 49/49 - 1s - loss: 2.6610 - accuracy: 0.7095 - val_loss: 2.7167 - val_accuracy: 0.7022 - 783ms/epoch
## Epoch 60/250
## 49/49 - 1s - loss: 2.7560 - accuracy: 0.7139 - val_loss: 2.8091 - val_accuracy: 0.7036 - 782ms/epoch
## Epoch 61/250
## 49/49 - 1s - loss: 2.8302 - accuracy: 0.7093 - val_loss: 2.8929 - val_accuracy: 0.7036 - 773ms/epoch
## Epoch 62/250
## 49/49 - 1s - loss: 2.9428 - accuracy: 0.7089 - val_loss: 3.0111 - val_accuracy: 0.7085 - 784ms/epoch
## Epoch 63/250
## 49/49 - 1s - loss: 3.0511 - accuracy: 0.7120 - val_loss: 3.1104 - val_accuracy: 0.7097 - 781ms/epoch
## Epoch 64/250
## 49/49 - 1s - loss: 3.1498 - accuracy: 0.7107 - val_loss: 3.2156 - val_accuracy: 0.7114 - 777ms/epoch
## Epoch 65/250
## 49/49 - 1s - loss: 3.2589 - accuracy: 0.7096 - val_loss: 3.3191 - val_accuracy: 0.7124 - 785ms/epoch
## Epoch 66/250
## 49/49 - 1s - loss: 3.3470 - accuracy: 0.7141 - val_loss: 3.4266 - val_accuracy: 0.6954 - 772ms/epoch
## Epoch 67/250
## 49/49 - 1s - loss: 3.4435 - accuracy: 0.7098 - val_loss: 3.5055 - val_accuracy: 0.7082 - 804ms/epoch
## Epoch 68/250
## 49/49 - 1s - loss: 3.5593 - accuracy: 0.7152 - val_loss: 3.6085 - val_accuracy: 0.7082 - 773ms/epoch
## Epoch 69/250
## 49/49 - 1s - loss: 3.6607 - accuracy: 0.7106 - val_loss: 3.7205 - val_accuracy: 0.7160 - 777ms/epoch
## Epoch 70/250
## 49/49 - 1s - loss: 3.7588 - accuracy: 0.7107 - val_loss: 3.8503 - val_accuracy: 0.7036 - 795ms/epoch
## Epoch 71/250
## 49/49 - 1s - loss: 3.8658 - accuracy: 0.7088 - val_loss: 3.9224 - val_accuracy: 0.7109 - 776ms/epoch
## Epoch 72/250
## 49/49 - 1s - loss: 3.9847 - accuracy: 0.7096 - val_loss: 4.0460 - val_accuracy: 0.7056 - 781ms/epoch
## Epoch 73/250
## 49/49 - 1s - loss: 4.0935 - accuracy: 0.7114 - val_loss: 4.1530 - val_accuracy: 0.7007 - 778ms/epoch
## Epoch 74/250
## 49/49 - 1s - loss: 4.2071 - accuracy: 0.7102 - val_loss: 4.2841 - val_accuracy: 0.7112 - 779ms/epoch
## Epoch 75/250
## 49/49 - 1s - loss: 4.3294 - accuracy: 0.7114 - val_loss: 4.4068 - val_accuracy: 0.7044 - 780ms/epoch
## Epoch 76/250
## 49/49 - 1s - loss: 4.4635 - accuracy: 0.7098 - val_loss: 4.5360 - val_accuracy: 0.7041 - 781ms/epoch
## Epoch 77/250
## 49/49 - 1s - loss: 4.5936 - accuracy: 0.7117 - val_loss: 4.6743 - val_accuracy: 0.6944 - 778ms/epoch
## Epoch 78/250
## 49/49 - 1s - loss: 4.7418 - accuracy: 0.7126 - val_loss: 4.8257 - val_accuracy: 0.7012 - 780ms/epoch
## Epoch 79/250
## 49/49 - 1s - loss: 4.8780 - accuracy: 0.7128 - val_loss: 4.9403 - val_accuracy: 0.7019 - 777ms/epoch
## Epoch 80/250
## 49/49 - 1s - loss: 5.0025 - accuracy: 0.7139 - val_loss: 5.0726 - val_accuracy: 0.7095 - 798ms/epoch
## Epoch 81/250
## 49/49 - 1s - loss: 5.1275 - accuracy: 0.7079 - val_loss: 5.2047 - val_accuracy: 0.7007 - 777ms/epoch
## Epoch 82/250
## 49/49 - 1s - loss: 5.2643 - accuracy: 0.7103 - val_loss: 5.3582 - val_accuracy: 0.7039 - 781ms/epoch
## Epoch 83/250
## 49/49 - 1s - loss: 5.3956 - accuracy: 0.7107 - val_loss: 5.4857 - val_accuracy: 0.7053 - 791ms/epoch
## Epoch 84/250
## 49/49 - 1s - loss: 5.5240 - accuracy: 0.7125 - val_loss: 5.6073 - val_accuracy: 0.7031 - 777ms/epoch
## Epoch 85/250
## 49/49 - 1s - loss: 5.6709 - accuracy: 0.7095 - val_loss: 5.7634 - val_accuracy: 0.7039 - 773ms/epoch
## Epoch 86/250
## 49/49 - 1s - loss: 5.8380 - accuracy: 0.7120 - val_loss: 5.9094 - val_accuracy: 0.7121 - 781ms/epoch
## Epoch 87/250
## 49/49 - 1s - loss: 5.9822 - accuracy: 0.7097 - val_loss: 6.0695 - val_accuracy: 0.7109 - 785ms/epoch
## Epoch 88/250
## 49/49 - 1s - loss: 6.1216 - accuracy: 0.7125 - val_loss: 6.2049 - val_accuracy: 0.7053 - 794ms/epoch
## Epoch 89/250
## 49/49 - 1s - loss: 6.2769 - accuracy: 0.7124 - val_loss: 6.3534 - val_accuracy: 0.7075 - 785ms/epoch
## Epoch 90/250
## 49/49 - 1s - loss: 6.4260 - accuracy: 0.7096 - val_loss: 6.5057 - val_accuracy: 0.6973 - 776ms/epoch
## Epoch 91/250
## 49/49 - 1s - loss: 6.5650 - accuracy: 0.7103 - val_loss: 6.6505 - val_accuracy: 0.7175 - 782ms/epoch
## Epoch 92/250
## 49/49 - 1s - loss: 6.7216 - accuracy: 0.7092 - val_loss: 6.8050 - val_accuracy: 0.7087 - 781ms/epoch
## Epoch 93/250
## 49/49 - 1s - loss: 6.8724 - accuracy: 0.7124 - val_loss: 6.9813 - val_accuracy: 0.6985 - 794ms/epoch
## Epoch 94/250
## 49/49 - 1s - loss: 7.0096 - accuracy: 0.7115 - val_loss: 7.1438 - val_accuracy: 0.6761 - 779ms/epoch
## Epoch 95/250
## 49/49 - 1s - loss: 7.1469 - accuracy: 0.7040 - val_loss: 7.1991 - val_accuracy: 0.7017 - 779ms/epoch
## Epoch 96/250
## 49/49 - 1s - loss: 7.2659 - accuracy: 0.7077 - val_loss: 7.3675 - val_accuracy: 0.6973 - 967ms/epoch
## Epoch 97/250
## 49/49 - 1s - loss: 7.4254 - accuracy: 0.7115 - val_loss: 7.5074 - val_accuracy: 0.7095 - 787ms/epoch
## Epoch 98/250
## 49/49 - 1s - loss: 7.5952 - accuracy: 0.7094 - val_loss: 7.6862 - val_accuracy: 0.6854 - 776ms/epoch
## Epoch 99/250
## 49/49 - 1s - loss: 7.7322 - accuracy: 0.7111 - val_loss: 7.8162 - val_accuracy: 0.7046 - 781ms/epoch
## Epoch 100/250
## 49/49 - 1s - loss: 7.8992 - accuracy: 0.7118 - val_loss: 7.9830 - val_accuracy: 0.6975 - 785ms/epoch
## Epoch 101/250
## 49/49 - 1s - loss: 8.0572 - accuracy: 0.7127 - val_loss: 8.1421 - val_accuracy: 0.7095 - 782ms/epoch
## Epoch 102/250
## 49/49 - 1s - loss: 8.1987 - accuracy: 0.7131 - val_loss: 8.3042 - val_accuracy: 0.6842 - 782ms/epoch
## Epoch 103/250
## 49/49 - 1s - loss: 8.3449 - accuracy: 0.7085 - val_loss: 8.4603 - val_accuracy: 0.7029 - 817ms/epoch
## Epoch 104/250
## 49/49 - 1s - loss: 8.5089 - accuracy: 0.7076 - val_loss: 8.5919 - val_accuracy: 0.7027 - 773ms/epoch
## Epoch 105/250
## 49/49 - 1s - loss: 8.6482 - accuracy: 0.7090 - val_loss: 8.7522 - val_accuracy: 0.7051 - 812ms/epoch
## Epoch 106/250
## 49/49 - 1s - loss: 8.8211 - accuracy: 0.7089 - val_loss: 8.8896 - val_accuracy: 0.7034 - 776ms/epoch
## Epoch 107/250
## 49/49 - 1s - loss: 8.9642 - accuracy: 0.7089 - val_loss: 9.0611 - val_accuracy: 0.6995 - 781ms/epoch
## Epoch 108/250
## 49/49 - 1s - loss: 9.1242 - accuracy: 0.7109 - val_loss: 9.2425 - val_accuracy: 0.7029 - 780ms/epoch
## Epoch 109/250
## 49/49 - 1s - loss: 9.2676 - accuracy: 0.7098 - val_loss: 9.3501 - val_accuracy: 0.7097 - 791ms/epoch
## Epoch 110/250
## 49/49 - 1s - loss: 9.4146 - accuracy: 0.7141 - val_loss: 9.5068 - val_accuracy: 0.7090 - 779ms/epoch
## Epoch 111/250
## 49/49 - 1s - loss: 9.5767 - accuracy: 0.7119 - val_loss: 9.6714 - val_accuracy: 0.7034 - 776ms/epoch
## Epoch 112/250
## 49/49 - 1s - loss: 9.7589 - accuracy: 0.7089 - val_loss: 9.8528 - val_accuracy: 0.7099 - 786ms/epoch
## Epoch 113/250
## 49/49 - 1s - loss: 9.9258 - accuracy: 0.7098 - val_loss: 9.9987 - val_accuracy: 0.6956 - 772ms/epoch
## Epoch 114/250
## 49/49 - 1s - loss: 10.0641 - accuracy: 0.7080 - val_loss: 10.1685 - val_accuracy: 0.7022 - 783ms/epoc
## Epoch 115/250
## 49/49 - 1s - loss: 10.2213 - accuracy: 0.7115 - val_loss: 10.3290 - val_accuracy: 0.7036 - 781ms/epoc
## Epoch 116/250
## 49/49 - 1s - loss: 10.3837 - accuracy: 0.7094 - val_loss: 10.4686 - val_accuracy: 0.7027 - 780ms/epoc
## Epoch 117/250
## 49/49 - 1s - loss: 10.5512 - accuracy: 0.7055 - val_loss: 10.6250 - val_accuracy: 0.6961 - 782ms/epoc
## Epoch 118/250
## 49/49 - 1s - loss: 10.7053 - accuracy: 0.7078 - val_loss: 10.8560 - val_accuracy: 0.6975 - 793ms/epoc
## Epoch 119/250
## 49/49 - 1s - loss: 10.8541 - accuracy: 0.7084 - val_loss: 10.9607 - val_accuracy: 0.6971 - 775ms/epoc
## Epoch 120/250
## 49/49 - 1s - loss: 11.0258 - accuracy: 0.7092 - val_loss: 11.1098 - val_accuracy: 0.7002 - 776ms/epoc
## Epoch 121/250
## 49/49 - 1s - loss: 11.1940 - accuracy: 0.7085 - val_loss: 11.3034 - val_accuracy: 0.6997 - 778ms/epoc
## Epoch 122/250
## 49/49 - 1s - loss: 11.3638 - accuracy: 0.7093 - val_loss: 11.4532 - val_accuracy: 0.6997 - 801ms/epoc
## Epoch 123/250
## 49/49 - 1s - loss: 11.5180 - accuracy: 0.7091 - val_loss: 11.6372 - val_accuracy: 0.6978 - 791ms/epoc
## Epoch 124/250
## 49/49 - 1s - loss: 11.6878 - accuracy: 0.7133 - val_loss: 11.7977 - val_accuracy: 0.7129 - 781ms/epoc
## Epoch 125/250
## 49/49 - 1s - loss: 11.8650 - accuracy: 0.7104 - val_loss: 11.9688 - val_accuracy: 0.7017 - 790ms/epoc
## Epoch 126/250
## 49/49 - 1s - loss: 12.0334 - accuracy: 0.7072 - val_loss: 12.1501 - val_accuracy: 0.6871 - 783ms/epoc
## Epoch 127/250
## 49/49 - 1s - loss: 12.1760 - accuracy: 0.7101 - val_loss: 12.2753 - val_accuracy: 0.7027 - 780ms/epoc
## Epoch 128/250
## 49/49 - 1s - loss: 12.3766 - accuracy: 0.7082 - val_loss: 12.4953 - val_accuracy: 0.7000 - 770ms/epoc
## Epoch 129/250
## 49/49 - 1s - loss: 12.5318 - accuracy: 0.7061 - val_loss: 12.5985 - val_accuracy: 0.6963 - 778ms/epoc
## Epoch 130/250
## 49/49 - 1s - loss: 12.6848 - accuracy: 0.7068 - val_loss: 12.8134 - val_accuracy: 0.6893 - 777ms/epoc
## Epoch 131/250
## 49/49 - 1s - loss: 12.8656 - accuracy: 0.7100 - val_loss: 12.9576 - val_accuracy: 0.6949 - 809ms/epoc
## Epoch 132/250
## 49/49 - 1s - loss: 13.0241 - accuracy: 0.7108 - val_loss: 13.1196 - val_accuracy: 0.6968 - 784ms/epoc
## Epoch 133/250
## 49/49 - 1s - loss: 13.1998 - accuracy: 0.7076 - val_loss: 13.3051 - val_accuracy: 0.6937 - 786ms/epoc
## Epoch 134/250
## 49/49 - 1s - loss: 13.3505 - accuracy: 0.7098 - val_loss: 13.4464 - val_accuracy: 0.7027 - 780ms/epoc
## Epoch 135/250
## 49/49 - 1s - loss: 13.5237 - accuracy: 0.7072 - val_loss: 13.6243 - val_accuracy: 0.7041 - 792ms/epoc
## Epoch 136/250
## 49/49 - 1s - loss: 13.7035 - accuracy: 0.7088 - val_loss: 13.8398 - val_accuracy: 0.6985 - 786ms/epoc
## Epoch 137/250
## 49/49 - 1s - loss: 13.8812 - accuracy: 0.7084 - val_loss: 13.9882 - val_accuracy: 0.7063 - 778ms/epoc
## Epoch 138/250
## 49/49 - 1s - loss: 14.0489 - accuracy: 0.7053 - val_loss: 14.1249 - val_accuracy: 0.6971 - 785ms/epoc
## Epoch 139/250
## 49/49 - 1s - loss: 14.2206 - accuracy: 0.7081 - val_loss: 14.3237 - val_accuracy: 0.6941 - 779ms/epoc
## Epoch 140/250
## 49/49 - 1s - loss: 14.4123 - accuracy: 0.7062 - val_loss: 14.5480 - val_accuracy: 0.7005 - 788ms/epoc
## Epoch 141/250
## 49/49 - 1s - loss: 14.6047 - accuracy: 0.7090 - val_loss: 14.6806 - val_accuracy: 0.7121 - 780ms/epoc
## Epoch 142/250
## 49/49 - 1s - loss: 14.7851 - accuracy: 0.7071 - val_loss: 14.8789 - val_accuracy: 0.7031 - 775ms/epoc
## Epoch 143/250
## 49/49 - 1s - loss: 14.9591 - accuracy: 0.7082 - val_loss: 15.0579 - val_accuracy: 0.7073 - 775ms/epoc
## Epoch 144/250
## 49/49 - 1s - loss: 15.1452 - accuracy: 0.7068 - val_loss: 15.2494 - val_accuracy: 0.7080 - 803ms/epoc
## Epoch 145/250
## 49/49 - 1s - loss: 15.3594 - accuracy: 0.7122 - val_loss: 15.4597 - val_accuracy: 0.7065 - 783ms/epoc
## Epoch 146/250
## 49/49 - 1s - loss: 15.5507 - accuracy: 0.7072 - val_loss: 15.6525 - val_accuracy: 0.7051 - 779ms/epoc
## Epoch 147/250
## 49/49 - 1s - loss: 15.7177 - accuracy: 0.7090 - val_loss: 15.8547 - val_accuracy: 0.6847 - 780ms/epoc
## Epoch 148/250
## 49/49 - 1s - loss: 15.8999 - accuracy: 0.7094 - val_loss: 16.0270 - val_accuracy: 0.6834 - 786ms/epoc
## Epoch 149/250
## 49/49 - 1s - loss: 16.0889 - accuracy: 0.7038 - val_loss: 16.1771 - val_accuracy: 0.7027 - 793ms/epoc
## Epoch 150/250
## 49/49 - 1s - loss: 16.2770 - accuracy: 0.7086 - val_loss: 16.3725 - val_accuracy: 0.6912 - 773ms/epoc
## Epoch 151/250
## 49/49 - 1s - loss: 16.4779 - accuracy: 0.7078 - val_loss: 16.6118 - val_accuracy: 0.6883 - 777ms/epoc
## Epoch 152/250
## 49/49 - 1s - loss: 16.6737 - accuracy: 0.7045 - val_loss: 16.7725 - val_accuracy: 0.7022 - 782ms/epoc
## Epoch 153/250
## 49/49 - 1s - loss: 16.8187 - accuracy: 0.7088 - val_loss: 16.9166 - val_accuracy: 0.7065 - 776ms/epoc
## Epoch 154/250
## 49/49 - 1s - loss: 17.0105 - accuracy: 0.7094 - val_loss: 17.1177 - val_accuracy: 0.6941 - 782ms/epoc
## Epoch 155/250
## 49/49 - 1s - loss: 17.2001 - accuracy: 0.7074 - val_loss: 17.3071 - val_accuracy: 0.7056 - 781ms/epoc
## Epoch 156/250
## 49/49 - 1s - loss: 17.4032 - accuracy: 0.7076 - val_loss: 17.5275 - val_accuracy: 0.7109 - 772ms/epoc
## Epoch 157/250
## 49/49 - 1s - loss: 17.5896 - accuracy: 0.7066 - val_loss: 17.6746 - val_accuracy: 0.7078 - 803ms/epoc
## Epoch 158/250
## 49/49 - 1s - loss: 17.7779 - accuracy: 0.7087 - val_loss: 17.8812 - val_accuracy: 0.7141 - 784ms/epoc
## Epoch 159/250
## 49/49 - 1s - loss: 17.9717 - accuracy: 0.7101 - val_loss: 18.0917 - val_accuracy: 0.7019 - 782ms/epoc
## Epoch 160/250
## 49/49 - 1s - loss: 18.1890 - accuracy: 0.7074 - val_loss: 18.2920 - val_accuracy: 0.7012 - 786ms/epoc
## Epoch 161/250
## 49/49 - 1s - loss: 18.3903 - accuracy: 0.7102 - val_loss: 18.5098 - val_accuracy: 0.6898 - 777ms/epoc
## Epoch 162/250
## 49/49 - 1s - loss: 18.6126 - accuracy: 0.7077 - val_loss: 18.7608 - val_accuracy: 0.6810 - 801ms/epoc
## Epoch 163/250
## 49/49 - 1s - loss: 18.7865 - accuracy: 0.6996 - val_loss: 18.9632 - val_accuracy: 0.6895 - 784ms/epoc
## Epoch 164/250
## 49/49 - 1s - loss: 18.9627 - accuracy: 0.7056 - val_loss: 19.0344 - val_accuracy: 0.7053 - 798ms/epoc
## Epoch 165/250
## 49/49 - 1s - loss: 19.1271 - accuracy: 0.7098 - val_loss: 19.3179 - val_accuracy: 0.6618 - 769ms/epoc
## Epoch 166/250
## 49/49 - 1s - loss: 19.3299 - accuracy: 0.7053 - val_loss: 19.4409 - val_accuracy: 0.7012 - 785ms/epoc
## Epoch 167/250
## 49/49 - 1s - loss: 19.5227 - accuracy: 0.7046 - val_loss: 19.6486 - val_accuracy: 0.7056 - 791ms/epoc
## Epoch 168/250
## 49/49 - 1s - loss: 19.7361 - accuracy: 0.7032 - val_loss: 19.8270 - val_accuracy: 0.6951 - 784ms/epoc
## Epoch 169/250
## 49/49 - 1s - loss: 19.8988 - accuracy: 0.7119 - val_loss: 20.0018 - val_accuracy: 0.7099 - 774ms/epoc
## Epoch 170/250
## 49/49 - 1s - loss: 20.1044 - accuracy: 0.7098 - val_loss: 20.2249 - val_accuracy: 0.6890 - 792ms/epoc
## Epoch 171/250
## 49/49 - 1s - loss: 20.2944 - accuracy: 0.7059 - val_loss: 20.4041 - val_accuracy: 0.7063 - 783ms/epoc
## Epoch 172/250
## 49/49 - 1s - loss: 20.5094 - accuracy: 0.7124 - val_loss: 20.6155 - val_accuracy: 0.7022 - 781ms/epoc
## Epoch 173/250
## 49/49 - 1s - loss: 20.6976 - accuracy: 0.7081 - val_loss: 20.8037 - val_accuracy: 0.7126 - 779ms/epoc
## Epoch 174/250
## 49/49 - 1s - loss: 20.9103 - accuracy: 0.7081 - val_loss: 21.0430 - val_accuracy: 0.6958 - 775ms/epoc
## Epoch 175/250
## 49/49 - 1s - loss: 21.0953 - accuracy: 0.7062 - val_loss: 21.1821 - val_accuracy: 0.7019 - 795ms/epoc
## Epoch 176/250
## 49/49 - 1s - loss: 21.2658 - accuracy: 0.7075 - val_loss: 21.3735 - val_accuracy: 0.6958 - 788ms/epoc
## Epoch 177/250
## 49/49 - 1s - loss: 21.4788 - accuracy: 0.7074 - val_loss: 21.6417 - val_accuracy: 0.6830 - 774ms/epoc
## Epoch 178/250
## 49/49 - 1s - loss: 21.6689 - accuracy: 0.7063 - val_loss: 21.7973 - val_accuracy: 0.6990 - 774ms/epoc
## Epoch 179/250
## 49/49 - 1s - loss: 21.8709 - accuracy: 0.7062 - val_loss: 21.9848 - val_accuracy: 0.7124 - 774ms/epoc
## Epoch 180/250
## 49/49 - 1s - loss: 22.1098 - accuracy: 0.7089 - val_loss: 22.2719 - val_accuracy: 0.6696 - 783ms/epoc
## Epoch 181/250
## 49/49 - 1s - loss: 22.3082 - accuracy: 0.7066 - val_loss: 22.4229 - val_accuracy: 0.7097 - 786ms/epoc
## Epoch 182/250
## 49/49 - 1s - loss: 22.5169 - accuracy: 0.7098 - val_loss: 22.6337 - val_accuracy: 0.6992 - 778ms/epoc
## Epoch 183/250
## 49/49 - 1s - loss: 22.7305 - accuracy: 0.7098 - val_loss: 22.8782 - val_accuracy: 0.7090 - 806ms/epoc
## Epoch 184/250
## 49/49 - 1s - loss: 22.9598 - accuracy: 0.7091 - val_loss: 23.0974 - val_accuracy: 0.6910 - 775ms/epoc
## Epoch 185/250
## 49/49 - 1s - loss: 23.1975 - accuracy: 0.6987 - val_loss: 23.2627 - val_accuracy: 0.6949 - 777ms/epoc
## Epoch 186/250
## 49/49 - 1s - loss: 23.3668 - accuracy: 0.6930 - val_loss: 23.5587 - val_accuracy: 0.6754 - 786ms/epoc
## Epoch 187/250
## 49/49 - 1s - loss: 23.4758 - accuracy: 0.7011 - val_loss: 23.5482 - val_accuracy: 0.7029 - 781ms/epoc
## Epoch 188/250
## 49/49 - 1s - loss: 23.6258 - accuracy: 0.7102 - val_loss: 23.7595 - val_accuracy: 0.7012 - 805ms/epoc
## Epoch 189/250
## 49/49 - 1s - loss: 23.8440 - accuracy: 0.7025 - val_loss: 23.9440 - val_accuracy: 0.6985 - 776ms/epoc
## Epoch 190/250
## 49/49 - 1s - loss: 24.0710 - accuracy: 0.7041 - val_loss: 24.1555 - val_accuracy: 0.7034 - 777ms/epoc
## Epoch 191/250
## 49/49 - 1s - loss: 24.2469 - accuracy: 0.7116 - val_loss: 24.4226 - val_accuracy: 0.6820 - 776ms/epoc
## Epoch 192/250
## 49/49 - 1s - loss: 24.4829 - accuracy: 0.7046 - val_loss: 24.5943 - val_accuracy: 0.7068 - 785ms/epoc
## Epoch 193/250
## 49/49 - 1s - loss: 24.6762 - accuracy: 0.7077 - val_loss: 24.9444 - val_accuracy: 0.6533 - 796ms/epoc
## Epoch 194/250
## 49/49 - 1s - loss: 24.8797 - accuracy: 0.7048 - val_loss: 25.0016 - val_accuracy: 0.6961 - 785ms/epoc
## Epoch 195/250
## 49/49 - 1s - loss: 25.0682 - accuracy: 0.7086 - val_loss: 25.1808 - val_accuracy: 0.7034 - 793ms/epoc
## Epoch 196/250
## 49/49 - 1s - loss: 25.3004 - accuracy: 0.7027 - val_loss: 25.4142 - val_accuracy: 0.7029 - 800ms/epoc
## Epoch 197/250
## 49/49 - 1s - loss: 25.5326 - accuracy: 0.7079 - val_loss: 25.6959 - val_accuracy: 0.6764 - 786ms/epoc
## Epoch 198/250
## 49/49 - 1s - loss: 25.7332 - accuracy: 0.7021 - val_loss: 25.8357 - val_accuracy: 0.6934 - 786ms/epoc
## Epoch 199/250
## 49/49 - 1s - loss: 25.9088 - accuracy: 0.7075 - val_loss: 26.0377 - val_accuracy: 0.6944 - 784ms/epoc
## Epoch 200/250
## 49/49 - 1s - loss: 26.1341 - accuracy: 0.7119 - val_loss: 26.2539 - val_accuracy: 0.6992 - 768ms/epoc
## Epoch 201/250
## 49/49 - 1s - loss: 26.3543 - accuracy: 0.7036 - val_loss: 26.4804 - val_accuracy: 0.6956 - 786ms/epoc
## Epoch 202/250
## 49/49 - 1s - loss: 26.5982 - accuracy: 0.7057 - val_loss: 26.7458 - val_accuracy: 0.7022 - 774ms/epoc
## Epoch 203/250
## 49/49 - 1s - loss: 26.8165 - accuracy: 0.7086 - val_loss: 26.8979 - val_accuracy: 0.6992 - 779ms/epoc
## Epoch 204/250
## 49/49 - 1s - loss: 26.9938 - accuracy: 0.7086 - val_loss: 27.1152 - val_accuracy: 0.7097 - 787ms/epoc
## Epoch 205/250
## 49/49 - 1s - loss: 27.2045 - accuracy: 0.7085 - val_loss: 27.3592 - val_accuracy: 0.6737 - 784ms/epoc
## Epoch 206/250
## 49/49 - 1s - loss: 27.4092 - accuracy: 0.7038 - val_loss: 27.6238 - val_accuracy: 0.6720 - 781ms/epoc
## Epoch 207/250
## 49/49 - 1s - loss: 27.6270 - accuracy: 0.7042 - val_loss: 27.7302 - val_accuracy: 0.7044 - 787ms/epoc
## Epoch 208/250
## 49/49 - 1s - loss: 27.8510 - accuracy: 0.7101 - val_loss: 28.0853 - val_accuracy: 0.6555 - 781ms/epoc
## Epoch 209/250
## 49/49 - 1s - loss: 28.0886 - accuracy: 0.7025 - val_loss: 28.1924 - val_accuracy: 0.7041 - 801ms/epoc
## Epoch 210/250
## 49/49 - 1s - loss: 28.2835 - accuracy: 0.7109 - val_loss: 28.4066 - val_accuracy: 0.7080 - 777ms/epoc
## Epoch 211/250
## 49/49 - 1s - loss: 28.5010 - accuracy: 0.7095 - val_loss: 28.6808 - val_accuracy: 0.6851 - 778ms/epoc
## Epoch 212/250
## 49/49 - 1s - loss: 28.7541 - accuracy: 0.7041 - val_loss: 28.8673 - val_accuracy: 0.6941 - 781ms/epoc
## Epoch 213/250
## 49/49 - 1s - loss: 28.9624 - accuracy: 0.7066 - val_loss: 29.1025 - val_accuracy: 0.6971 - 790ms/epoc
## Epoch 214/250
## 49/49 - 1s - loss: 29.1705 - accuracy: 0.7077 - val_loss: 29.2985 - val_accuracy: 0.7116 - 798ms/epoc
## Epoch 215/250
## 49/49 - 1s - loss: 29.4220 - accuracy: 0.7049 - val_loss: 29.5889 - val_accuracy: 0.6749 - 785ms/epoc
## Epoch 216/250
## 49/49 - 1s - loss: 29.6454 - accuracy: 0.7021 - val_loss: 29.7421 - val_accuracy: 0.6988 - 785ms/epoc
## Epoch 217/250
## 49/49 - 1s - loss: 29.8625 - accuracy: 0.7029 - val_loss: 29.9919 - val_accuracy: 0.7029 - 774ms/epoc
## Epoch 218/250
## 49/49 - 1s - loss: 30.1375 - accuracy: 0.6992 - val_loss: 30.3509 - val_accuracy: 0.6601 - 788ms/epoc
## Epoch 219/250
## 49/49 - 1s - loss: 30.3215 - accuracy: 0.7025 - val_loss: 30.4174 - val_accuracy: 0.6958 - 786ms/epoc
## Epoch 220/250
## 49/49 - 1s - loss: 30.5126 - accuracy: 0.6990 - val_loss: 30.6361 - val_accuracy: 0.7046 - 781ms/epoc
## Epoch 221/250
## 49/49 - 1s - loss: 30.7667 - accuracy: 0.7096 - val_loss: 30.9062 - val_accuracy: 0.6917 - 777ms/epoc
## Epoch 222/250
## 49/49 - 1s - loss: 31.0470 - accuracy: 0.7051 - val_loss: 31.1530 - val_accuracy: 0.6893 - 799ms/epoc
## Epoch 223/250
## 49/49 - 1s - loss: 31.2466 - accuracy: 0.7047 - val_loss: 31.3625 - val_accuracy: 0.7102 - 790ms/epoc
## Epoch 224/250
## 49/49 - 1s - loss: 31.4791 - accuracy: 0.7046 - val_loss: 31.6398 - val_accuracy: 0.6992 - 783ms/epoc
## Epoch 225/250
## 49/49 - 1s - loss: 31.6890 - accuracy: 0.7049 - val_loss: 31.7947 - val_accuracy: 0.7075 - 776ms/epoc
## Epoch 226/250
## 49/49 - 1s - loss: 31.9219 - accuracy: 0.7071 - val_loss: 32.0772 - val_accuracy: 0.7007 - 790ms/epoc
## Epoch 227/250
## 49/49 - 1s - loss: 32.1465 - accuracy: 0.7086 - val_loss: 32.2936 - val_accuracy: 0.6849 - 791ms/epoc
## Epoch 228/250
## 49/49 - 1s - loss: 32.3669 - accuracy: 0.7030 - val_loss: 32.5838 - val_accuracy: 0.6808 - 785ms/epoc
## Epoch 229/250
## 49/49 - 1s - loss: 32.5961 - accuracy: 0.7050 - val_loss: 32.6920 - val_accuracy: 0.7070 - 782ms/epoc
## Epoch 230/250
## 49/49 - 1s - loss: 32.7799 - accuracy: 0.7032 - val_loss: 32.8709 - val_accuracy: 0.6949 - 775ms/epoc
## Epoch 231/250
## 49/49 - 1s - loss: 32.9770 - accuracy: 0.7008 - val_loss: 33.1192 - val_accuracy: 0.7024 - 787ms/epoc
## Epoch 232/250
## 49/49 - 1s - loss: 33.2032 - accuracy: 0.7052 - val_loss: 33.3221 - val_accuracy: 0.7095 - 786ms/epoc
## Epoch 233/250
## 49/49 - 1s - loss: 33.4340 - accuracy: 0.7042 - val_loss: 33.5473 - val_accuracy: 0.7129 - 783ms/epoc
## Epoch 234/250
## 49/49 - 1s - loss: 33.6658 - accuracy: 0.7085 - val_loss: 33.7775 - val_accuracy: 0.7136 - 787ms/epoc
## Epoch 235/250
## 49/49 - 1s - loss: 33.9014 - accuracy: 0.7007 - val_loss: 33.9544 - val_accuracy: 0.6915 - 800ms/epoc
## Epoch 236/250
## 49/49 - 1s - loss: 34.0722 - accuracy: 0.7024 - val_loss: 34.1940 - val_accuracy: 0.6954 - 781ms/epoc
## Epoch 237/250
## 49/49 - 1s - loss: 34.3174 - accuracy: 0.7071 - val_loss: 34.4708 - val_accuracy: 0.6954 - 772ms/epoc
## Epoch 238/250
## 49/49 - 1s - loss: 34.5722 - accuracy: 0.7025 - val_loss: 34.7603 - val_accuracy: 0.6582 - 782ms/epoc
## Epoch 239/250
## 49/49 - 1s - loss: 34.7750 - accuracy: 0.7032 - val_loss: 34.8681 - val_accuracy: 0.7019 - 783ms/epoc
## Epoch 240/250
## 49/49 - 1s - loss: 34.9983 - accuracy: 0.7057 - val_loss: 35.1214 - val_accuracy: 0.7104 - 782ms/epoc
## Epoch 241/250
## 49/49 - 1s - loss: 35.2797 - accuracy: 0.7066 - val_loss: 35.4103 - val_accuracy: 0.6903 - 840ms/epoc
## Epoch 242/250
## 49/49 - 1s - loss: 35.4832 - accuracy: 0.7032 - val_loss: 35.6285 - val_accuracy: 0.6990 - 785ms/epoc
## Epoch 243/250
## 49/49 - 1s - loss: 35.7405 - accuracy: 0.7086 - val_loss: 35.9262 - val_accuracy: 0.6839 - 787ms/epoc
## Epoch 244/250
## 49/49 - 1s - loss: 36.0016 - accuracy: 0.7012 - val_loss: 36.0971 - val_accuracy: 0.6825 - 773ms/epoc
## Epoch 245/250
## 49/49 - 1s - loss: 36.2368 - accuracy: 0.7043 - val_loss: 36.3669 - val_accuracy: 0.7068 - 784ms/epoc
## Epoch 246/250
## 49/49 - 1s - loss: 36.4954 - accuracy: 0.7004 - val_loss: 36.6414 - val_accuracy: 0.6791 - 788ms/epoc
## Epoch 247/250
## 49/49 - 1s - loss: 36.7625 - accuracy: 0.6987 - val_loss: 36.8781 - val_accuracy: 0.7090 - 788ms/epo
## Epoch 248/250
## 49/49 - 1s - loss: 36.9908 - accuracy: 0.7081 - val_loss: 37.1534 - val_accuracy: 0.7000 - 804ms/epo
## Epoch 249/250
## 49/49 - 1s - loss: 37.2654 - accuracy: 0.7094 - val_loss: 37.4426 - val_accuracy: 0.6703 - 773ms/epo
## Epoch 250/250
## 49/49 - 1s - loss: 37.5185 - accuracy: 0.7021 - val_loss: 37.6159 - val_accuracy: 0.7175 - 790ms/epo
```

Task 2h)

Given that we found the optimal degree of regularization in the hidden units of our neural net, we want to extract the neural net with best tuning parameter value from `tune_NN_2g` and save it as `model3`. You need to do this using the `unserialize_model()` function since the best model is saved as object `tune_NN_2g$finalModel$object` in an R-compatible format.

```
model3 <- unserialize_model(tune_NN_2g$finalModel$object)

model3
```

```
## Model: "sequential"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                     Param #
## ================================================================================
##  dense_2 (Dense)                    (None, 30)                       3720
##  dense_1 (Dense)                    (None, 15)                       465
##  dense (Dense)                      (None, 2)                        32
## ================================================================================
## Total params: 4217 (16.47 KB)
## Trainable params: 4217 (16.47 KB)
## Non-trainable params: 0 (0.00 Byte)
## ________________________________________________________________________________
```

Part 3: Saving, loading and retraining neural nets

Task 3a)

Training and tuning neural nets can take a lot of time. Therefore, it is possible to save entire fitted models to disk and to import them at a later point in time. Apply the `save_model_tf()` function to save your most recent `model3` as *DABN13_asst6_saved_model3* in your working directory.

Ensure that the model is saved in TensorFlow SavedModel format.

Additionally, use the file explorer in your operating system to look how exactly the model was saved on your hard drive. Describe this shortly in the string_variable `saved_model_3a`.

```
# # 1.

save_model_tf(model3,
             "DABN13_asst6_saved_model3",
             overwrite = TRUE,
             include_optimizer = TRUE,
             signatures = NULL,
             options = NULL
)

# # 2.
```

```
saved_model_3a <- "The model was saved in TensorFlow SavedModel format in the current working directory
/Users/viktorsjoberg/Desktop/ML/Assignment 6."
```

Task 3b)

Now, use the `load_model_tf` function in the models module of Keras to load your saved model into your Python session again. Save this model as `model4`.

Note: The possibility to load a previously saved model from your hard disk is useful for more than just your own models. It even allows you to load pretrained models for specific purposes from the TensorFlow Hub or from Hugging Face. These models could then directly be used for prediction or fine-tuned on your data.

```
model4 <- load_model_tf("DABN13_asst6_saved_model3")
summary(model4)
```

```
## Model: "sequential"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  dense_2 (Dense)                    (None, 30)                      3720
##  dense_1 (Dense)                    (None, 15)                      465
##  dense (Dense)                      (None, 2)                       32
## ================================================================================
## Total params: 4217 (16.47 KB)
## Trainable params: 4217 (16.47 KB)
## Non-trainable params: 0 (0.00 Byte)
## ________________________________________________________________________________
```

Task 3c)

We will now evaluate the tuned model `model4`. Do the following

1. When we tuned our most recent neural net, we did this on a relatively small fraction of the training data to reduce the computational cost. This was also the data used to train the best model that we extracted from `NN_tune_2d`. Now that we have chosen an optimal tuning parameter, it makes sense to retrain the `model4` on the entire training data `X_train` and `y_train`. Do this by using the `fit()` function again with `model4` as first argument. As previously, training should be done for 250 epochs, unless early stopping with a patience of 20 epochs kicks in. Minibatches of 2^8 data points should be used. Given the large amount of training data, hold only 10% of the data aside for monitoring validation loss.
2. Once you retrained your model, obtain predicted class probabilities and save them as `prob_model4`.
3. `prob_model4` contains two columns because the neural network that we tuned with caret returned probabilities for both the purchase of Bud Light or any other beer brand. Hence, keep only the column in `prob_model4` that shows probabilities for the purchase of Bud Light.
4. Get the log loss `logloss_model4` on the test data.
5. To what extend did model tuning and retraining change test set accuracy relative to that of `model2`? Comment on this in the string variable `performance_comparison_3c`

```
# # 1.

y_train_numeric <- as.numeric(y_train) - 1
y_train_one_hot <- to_categorical(y_train_numeric, num_classes = 2)

model4_fit <-  fit(
                model4,
                x = X_train,
                y = y_train_one_hot,
                epochs = 250,
                batch_size = 2^8,
                validation_split = 0.1,
                callbacks = list(early_stop)
               )
```

```
## Epoch 1/250
## 193/193 - 3s - loss: 36.2842 - accuracy: 0.5609 - val_loss: 34.3021 - val_accuracy: 0.8147 - 3s/epoc
## Epoch 2/250
## 193/193 - 3s - loss: 32.8714 - accuracy: 0.6297 - val_loss: 31.0641 - val_accuracy: 0.8563 - 3s/epoc
## Epoch 3/250
## 193/193 - 3s - loss: 29.9625 - accuracy: 0.6465 - val_loss: 28.4606 - val_accuracy: 0.9586 - 3s/epoc
## Epoch 4/250
## 193/193 - 3s - loss: 27.5810 - accuracy: 0.6561 - val_loss: 26.3233 - val_accuracy: 0.9411 - 3s/epoc
## Epoch 5/250
## 193/193 - 3s - loss: 25.6066 - accuracy: 0.6584 - val_loss: 24.6098 - val_accuracy: 0.9770 - 3s/epoc
## Epoch 6/250
## 193/193 - 3s - loss: 24.0752 - accuracy: 0.6587 - val_loss: 23.1250 - val_accuracy: 0.9900 - 3s/epoc
## Epoch 7/250
## 193/193 - 3s - loss: 22.7888 - accuracy: 0.6651 - val_loss: 22.1132 - val_accuracy: 0.8466 - 3s/epoc
## Epoch 8/250
## 193/193 - 3s - loss: 21.7708 - accuracy: 0.6679 - val_loss: 21.2056 - val_accuracy: 0.8671 - 3s/epoc
## Epoch 9/250
## 193/193 - 3s - loss: 20.9639 - accuracy: 0.6650 - val_loss: 20.4309 - val_accuracy: 0.9592 - 3s/epoc
## Epoch 10/250
## 193/193 - 3s - loss: 20.4008 - accuracy: 0.6708 - val_loss: 19.9250 - val_accuracy: 0.9770 - 3s/epoc
## Epoch 11/250
## 193/193 - 3s - loss: 19.9965 - accuracy: 0.6699 - val_loss: 19.6509 - val_accuracy: 0.9871 - 3s/epoc
## Epoch 12/250
## 193/193 - 3s - loss: 19.7444 - accuracy: 0.6718 - val_loss: 19.4701 - val_accuracy: 0.9690 - 3s/epoc
## Epoch 13/250
## 193/193 - 3s - loss: 19.6526 - accuracy: 0.6774 - val_loss: 19.4378 - val_accuracy: 0.9639 - 3s/epoc
## Epoch 14/250
## 193/193 - 3s - loss: 19.6224 - accuracy: 0.6761 - val_loss: 19.5016 - val_accuracy: 0.9210 - 3s/epoc
## Epoch 15/250
## 193/193 - 3s - loss: 19.7523 - accuracy: 0.6748 - val_loss: 19.6140 - val_accuracy: 0.9778 - 3s/epoc
## Epoch 16/250
## 193/193 - 3s - loss: 19.9996 - accuracy: 0.6736 - val_loss: 19.9856 - val_accuracy: 0.8904 - 3s/epoc
## Epoch 17/250
## 193/193 - 3s - loss: 20.3377 - accuracy: 0.6713 - val_loss: 20.2647 - val_accuracy: 0.9592 - 3s/epoc
## Epoch 18/250
## 193/193 - 3s - loss: 20.6622 - accuracy: 0.6778 - val_loss: 20.5775 - val_accuracy: 0.9694 - 3s/epoc
## Epoch 19/250
## 193/193 - 3s - loss: 21.0486 - accuracy: 0.6703 - val_loss: 21.0970 - val_accuracy: 0.8793 - 3s/epoc
## Epoch 20/250
## 193/193 - 3s - loss: 21.5432 - accuracy: 0.6705 - val_loss: 21.4876 - val_accuracy: 0.9708 - 3s/epoc
## Epoch 21/250
## 193/193 - 3s - loss: 22.0290 - accuracy: 0.6752 - val_loss: 22.1267 - val_accuracy: 0.9181 - 3s/epoc
## Epoch 22/250
## 193/193 - 3s - loss: 22.6241 - accuracy: 0.6768 - val_loss: 22.7105 - val_accuracy: 0.8979 - 3s/epoc
## Epoch 23/250
## 193/193 - 3s - loss: 23.1683 - accuracy: 0.6760 - val_loss: 23.1932 - val_accuracy: 0.9648 - 3s/epoc
## Epoch 24/250
## 193/193 - 3s - loss: 23.8101 - accuracy: 0.6745 - val_loss: 23.9330 - val_accuracy: 0.9389 - 3s/epoch
## Epoch 25/250
## 193/193 - 3s - loss: 24.4897 - accuracy: 0.6642 - val_loss: 24.7428 - val_accuracy: 0.7939 - 3s/epoch
## Epoch 26/250
## 193/193 - 3s - loss: 25.2020 - accuracy: 0.6742 - val_loss: 25.3305 - val_accuracy: 0.9499 - 3s/epoch
## Epoch 27/250
## 193/193 - 3s - loss: 25.9208 - accuracy: 0.6703 - val_loss: 26.1053 - val_accuracy: 0.8973 - 3s/epoch
## Epoch 28/250
## 193/193 - 3s - loss: 26.7166 - accuracy: 0.6673 - val_loss: 26.9379 - val_accuracy: 0.9389 - 3s/epoch
## Epoch 29/250
## 193/193 - 3s - loss: 27.5509 - accuracy: 0.6716 - val_loss: 27.7647 - val_accuracy: 0.9152 - 3s/epoch
## Epoch 30/250
## 193/193 - 3s - loss: 28.3962 - accuracy: 0.6609 - val_loss: 28.6265 - val_accuracy: 0.9444 - 3s/epoch
## Epoch 31/250
## 193/193 - 3s - loss: 29.3421 - accuracy: 0.6668 - val_loss: 29.5337 - val_accuracy: 0.8893 - 3s/epoch
## Epoch 32/250
## 193/193 - 3s - loss: 30.2400 - accuracy: 0.6733 - val_loss: 30.4217 - val_accuracy: 0.9486 - 3s/epoch
## Epoch 33/250
## 193/193 - 3s - loss: 31.1663 - accuracy: 0.6656 - val_loss: 31.4532 - val_accuracy: 0.8764 - 3s/epoch
```

```
#
# # 2.
prob_model4 <- predict(model4, X_test)
```

```
## 572/572 - 1s - 976ms/epoch - 2ms/step
```

```
head(prob_model4)
```

```
##           [,1]      [,2]
## [1,] 0.3837405 0.6377624
## [2,] 0.2968995 0.7029212
## [3,] 0.2013703 0.8486812
## [4,] 0.2684305 0.7116122
## [5,] 0.5162200 0.5047175
## [6,] 0.5162200 0.5047175
```

```
# # 3.
prob_model4_bud_light <- prob_model4[, 1]
head(prob_model4_bud_light)
```

```
## [1] 0.3837405 0.2968995 0.2013703 0.2684305 0.5162200 0.5162200
```

```
# # 4.
logloss_model4 <- LogLoss(prob_model4_bud_light, as.numeric(y_test))
print(logloss_model4)
```

```
## [1] 0.6020202
```

```
#
# # 5.
performance_comparison_3c <-  "The value of logloss in model4 is lower then the value obtained in model2
```

Part 4: Manual predictions from a trained neural net

In this part we will build predictions *manually* by extracting weights from the trained `model2` and by constructing the transformations in the layers of the neural net ourselves.

Task 4a)

Start this task by creating your own ReLU activation function. Save it as `ReLU`. Then, write your own sigmoid function for the output layer transformation. Save it as `sigmoid`.

Hint: For `ReLU`, `pmax` is a useful function.

```
# # 1.
ReLU <- function(x) {
  pmax(0, x)
}

# # 2.
sigmoid <- function(x) {
  1 / (1 + exp(-x))
}
```

Task 4b)

In the slides for lecture 8 we discussed how units in the different layers of a neural net look like. Now we are going to use the equations both hidden units and output unit to construct output predictions for the n_{test} data points in `X_test`. Please do the following:

1. Use the `get_weights()` function on `model2` to obtain a list object which stores the weights and biases of the learned model. Save it as `weight_and_bias_4b`.
2. Extract the objects inside `weight_and_bias_4b` into the objects for $\mathbf{b}_1, \mathbf{W}_1, \mathbf{b}_2, \mathbf{W}_2, \mathbf{b}_3, \mathbf{W}_3$ defined in the code chunk below.
3. Construct the linear term of the first layer hidden units and save these linear terms as a $30 \times n_{train}$ vector `Z_1`.
4. Construct the $15 \times n_{train}$ vector `Z_2` of linear terms for the second layer hidden units. Then, obtain the linear term of the output unit and save it as `Z_3`.
5. Put `Z_3` into the output layer activation function in order to get predictions for the probability of a Bud Light purchase. Save this as $n_{train} \times 1$ vector `pred_model2_own_4b`.

Hint: You can use `dim()` to check the dimension of matrices and `length()` to check that of vectors. Additionally, to ensure that you got the correct result for `pred_own_4b` you can compare it with the output of `predict()`.

```
# # 1.
weight_and_bias_4b <- get_weights(model2)


# # 2.
bb_1 <- weight_and_bias_4b[[2]]
WW_1 <- weight_and_bias_4b[[1]]
bb_2 <- weight_and_bias_4b[[4]]
WW_2 <- weight_and_bias_4b[[3]]
bb_3 <- weight_and_bias_4b[[6]]
WW_3 <- weight_and_bias_4b[[5]]
#


# # 3.
Z_1 <- X_train %*% (WW_1) + matrix(rep(bb_1, times = n_train), nrow = n_train)

# # 4.
Z_2 <- Z_1 %*% (WW_2) + matrix(rep(bb_2, times = n_train), nrow = n_train)
```

```r
Z_3 <- Z_2 %*% (WW_3) + matrix(rep(bb_3, times = n_train), nrow = n_train)
#
# # 5.
pred_model2_own_4b <- sigmoid(Z_3)
```