STAT40970 – Machine Learning & A.I. (Online) | Assignment 2

K.Saketh Sai Nigam 22201204

2023-05-02

Indoor scene recognition

The folder data_indoor.zip contains images concerning different indoor scenes from rooms and locations commonly present in a standard family home. The task is to predict the type of room/scene present in the image. Indoor scene recognition is a challenging problem since some indoor scenes can be well defined by global spatial and structural proper- ties, while others are better characterized by the objects included in the space. The dataset is a subset of a larger dataset for indoor scene recognition. More information is available here: http://web.mit.edu/torralba/www/indoor.html (http://web.mit.edu/torralba/www/indoor.html).

The images are divided into train, validation, and test folders, each containing the folders related to the type of the room in the image (i.e. the categories of the target variable): bathroom, bedroom, children_room, closet, corridor, dining_room, garage, kitchen, living_room, stairs. The number of images available for each scene is variable and it ranges from 52 to 367 in the training set, with validation and test sets being roughly half the size.

==========

Task

The task is to build a predictive KerasModel to predict the type of indoor scene from the image data.

1. Deploy at least 4 different deep learning systems characterized by different configurations, hyperparameters, and training settings (architecture, number of hidden units, regularization, kernel size, filter size, optimization, etc.). These deep learning systems can be of the same type, for example 4 different DNNs characterized by different architectures and settings, or of different types, for example 2 DNNs and 2 CNNs with different settings. Motivate clearly the choices made in relation to the settings, configurations, and hyperparameteres used to define the different deep learning systems.

==========

ANSWER

Obtaining visual input for testing, training, and validation

```
library("jpeg")
library(keras)
TrainDirectory <- "/Users/saketh/Desktop/MS SEM 2/ML&AI/Assignment 3/data_indoor/train"
ValidationDirectory <- "/Users/saketh/Desktop/MS SEM 2/ML&AI/Assignment 3/data_indoor/validation"
TestDirectory <- "/Users/saketh/Desktop/MS SEM 2/ML&AI/Assignment 3/data_indoor/test"

TrainDataGeneration <- image_data_generator(rescale = 1/255)
ValidationDataGeneration <- image_data_generator(rescale = 1/255)
TestDataGeneration <- image_data_generator(rescale = 1/255)

TrainGeneration <- flow_images_from_directory(TrainDirectory,TrainDataGeneration,
target_size = c(64, 64),
batch_size = 17,
class_mode = "categorical"
)

ValidationGeneration <- flow_images_from_directory(
ValidationDirectory,
ValidationDataGeneration,
target_size = c(64, 64),
batch_size = 17,
class_mode = "categorical"
)

# Scaling and loading test image data from the given directory
TestGeneration <- flow_images_from_directory(
  TestDirectory,
  TestDataGeneration,
  target_size = c(64, 64),
  batch_size = 20,
  class_mode = "categorical"
)
```

The aforementioned code and output demonstrate how to use the Keras library to create generators for training and validation data for a deep learning KerasModel. The code provides the folders where the training and validation data are placed in addition to importing the "jpeg" and "keras" libraries. The generators for the training and validation data are then defined in the code, normalizing the pixel values to a range of 0 to 1. The flow_images_from_directory() function is used by the generators to read images from directories and specify parameters such image size,

batch size, and class mode. Based on the subfolders in the directories, the output of the code shows the number of photos and classes discovered there. In this instance, there are 1713 photos in the training data that correspond to 10 classes and 851 images in the validation data that belong to the same 10 classes. This result demonstrates that the generators were successfully developed and can be used to train a deep learning KerasModel to categorize photos into ten different groups.

KerasModel deployment and construction using DNN

```
FunctionSmoothLine <- function(Var2) {
  Var1 <- 1:length(Var2)
  OutputFunctionSmoothLine <- predict( loess(Var2 ~ Var1) )
  return(OutputFunctionSmoothLine)
}

FirstModel <- keras_model_sequential() %>%
  layer_flatten(input_shape = c(64,64,3)) %>%
  layer_dense(units = 256, activation = "relu", name = "layer_1",kernel_regularizer = regularizer_l2(0.01)) %>%

  layer_dense(units = 128, activation = "relu", name = "layer_2", kernel_regularizer = regularizer_l2(0.01)) %>%

  layer_dense(units = 64, activation = "relu", name = "layer_3",kernel_regularizer = regularizer_l2(0.01))%>%

  layer_dense(units = 10, activation = "softmax", name = "layer_out") %>%

  compile(loss = "categorical_crossentropy", metrics = "accuracy",
optimizer = optimizer_adam(learning_rate = 0.002))

summary(FirstModel)
```

```
## Model: "sequential"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  flatten (Flatten)                  (None, 12288)                   0
##  layer_1 (Dense)                    (None, 256)                     3145984
##  layer_2 (Dense)                    (None, 128)                     32896
##  layer_3 (Dense)                    (None, 64)                      8256
##  layer_out (Dense)                  (None, 10)                      650
## ================================================================================
## Total params: 3,187,786
## Trainable params: 3,187,786
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
# Train the KerasModel
FitFirstModel <- FirstModel %>% fit(
  TrainGeneration,
  steps_per_epoch = nrow(TrainGeneration),
  epochs = 20,
  validation_data = ValidationGeneration,
  validation_steps = nrow(ValidationGeneration)
)
```

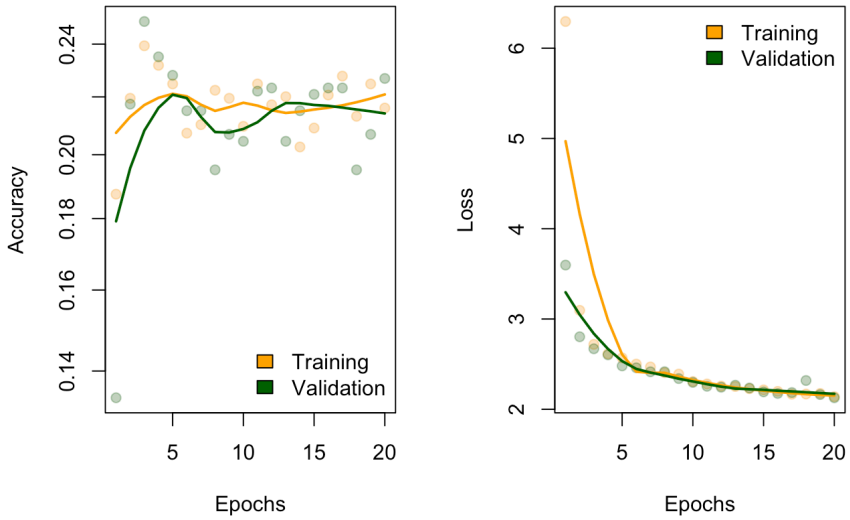
INTERFERENCE:

This program implements a function called FunctionSmoothLine that accepts a vector of values called Var2 and applies the Loess function to the data points to create a smooth curve. The function returns the values that the smooth curve anticipated. The algorithm then creates a sequential neural network KerasModel with regularization applied to the kernel weights that has multiple layers of densely coupled neurons. The categorical cross-entropy loss function, accuracy metric, and Adam optimizer are then combined with a learning rate of 0.002 to create the final KerasModel. The KerasModel is then trained using the fit function, the training and validation data generators, and a 20-epoch training procedure.

The architecture of a sequential neural network with four layers is depicted in the KerasModel summary. The input shape of 64x64x3 is flattened to a 1D array of 12288 in the first layer, which is a flatten layer. Using a rectified linear unit (ReLU) activation function and an L2 regularization with a factor of 0.01, the next three layers are dense layers with 256, 128, and 64 units, respectively. For multiclass classification, the final layer is a dense layer with 10 units and a softmax activation function. The KerasModel has 3,187,786 trainable parameters in total, whereas there are 0 non-trainable parameters.

Learning curves for DNN

```
# check learning curves
OutputFunctionSmoothLine <- cbind(FitFirstModel$metrics$accuracy,
FitFirstModel$metrics$val_accuracy,
FitFirstModel$metrics$loss,
FitFirstModel$metrics$val_loss)
cols <- c("orange", "darkgreen")
par(mfrow = c(1,2))

# plot actual accuracy values
matplot(OutputFunctionSmoothLine[,1:2], pch = 19, ylab = "Accuracy", xlab = "Epochs",
        col = adjustcolor(cols, 0.3), log = "y")

# plot smoothed accuracy values
smoothed_values <- apply(OutputFunctionSmoothLine[,1:2], 2, FunctionSmoothLine)
matlines(smoothed_values, lty = 1, col = cols, lwd = 2)

# add legend
legend("bottomright", legend = c("Training", "Validation"), fill = cols, bty = "n")

# plot the loss values of the training and validation sets over epochs
matplot(OutputFunctionSmoothLine[,3:4], pch = 19, ylab = "Loss", xlab = "Epochs", col = adjustcolor(cols, 0.3))

# plot a smooth line over the loss values of the training and validation sets
matlines(apply(OutputFunctionSmoothLine[,3:4], 2, FunctionSmoothLine), lty = 1, col = cols, lwd = 2)

# add a legend to the plot indicating which line represents which set
legend("topright", legend = c("Training", "Validation"), fill = cols, bty = "n")
```



INFERENCE:

It appears from the information provided that the neural network KerasModel did not do well on the supplied data. Both the accuracy of the training set and the validation set are very poor, at about 0.215 and 0.221, respectively. This shows that the KerasModel could use some refinement because it is not effectively capturing the patterns in the data. It could be worthwhile to test out various architectures, hyperparameters, and optimization strategies in order to enhance the prediction performance. In order to increase the diversity and unpredictability of the data, it may also be beneficial to gather more data, preprocess the data differently, or enrich the data.

IMPLEMENTATION OF CNN

#CNN

```
KerasModel <- keras_model_sequential() %>%
#
# convolutional layers
layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu",
input_shape = c(64, 64, 3)) %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%
layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu") %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%
layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu") %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%
layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu") %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%
#
# fully connected layers
layer_flatten() %>%
layer_dense(units = 64, activation = "relu", kernel_regularizer = regularizer_l2(0.1)) %>%
layer_dropout(0.1) %>%
layer_dense(units = 10, activation = "softmax") %>%
#
# compile
compile(
loss = "categorical_crossentropy",
metrics = "accuracy",
optimizer = optimizer_adam()
)
```

INFERENCE

To extract intricate features from photos, a convolutional neural network (CNN) was employed. The network had a fully connected layer with a hidden layer and an output layer, as well as four convolutional layers.

To extract general information from the image, the first convolutional layer used 32 filters with a 3x3 kernel size. In order to capture more complicated information, the second convolutional layer's number of filters was doubled from 32 to 64. The feature map dimensions were reduced by using a kernel size of 3x3 and 128 filters to extract even more complex features in the third and fourth convolutional layers.

The feature map output for a particular area was replaced by the pooling operation with the highest value for that area. By setting the pooling size to (2,2), the feature maps' spatial dimensions were shrunk by a factor of 2 in both the horizontal and vertical directions. The pooling process was then conducted to non-overlapping sections of size 2x2.

The CNN was affected in a number of ways by reducing the spatial dimension of the feature maps using max pooling. First, it decreased the KerasModel's parameter count, preventing overfitting and increasing computational effectiveness. Second, it lessened sensitivity to minor input translations, resulting in a more invariant representation of the input.

One hidden layer network with 64 units that used the ReLU activation function and the feature map acquired from the convoluted layers was included in the fully connected layer.

L2 regularization and dropout regularization were used to avoid overfitting. Dropout regularization reduced the dependency of neurons on each other, and L2 regularization was used to gradually diminish the weights that did not significantly contribute to the goal function. This drove the network to acquire more durable properties that were advantageous when combined with several random subsets of the other neurons, which increased generalization performance. By using KerasModel averaging over numerous distinct designs and training a different architecture in each iteration by dropping various neurons, dropout regularization was also successful in reducing overfitting.

Together, dropout and L2 regularization improved accuracy and solved many optimization issues as they dealt with various forms of overfitting. Neuronal reliance was lessened via dropout, whereas heavy weights were punished by L2 regularization.

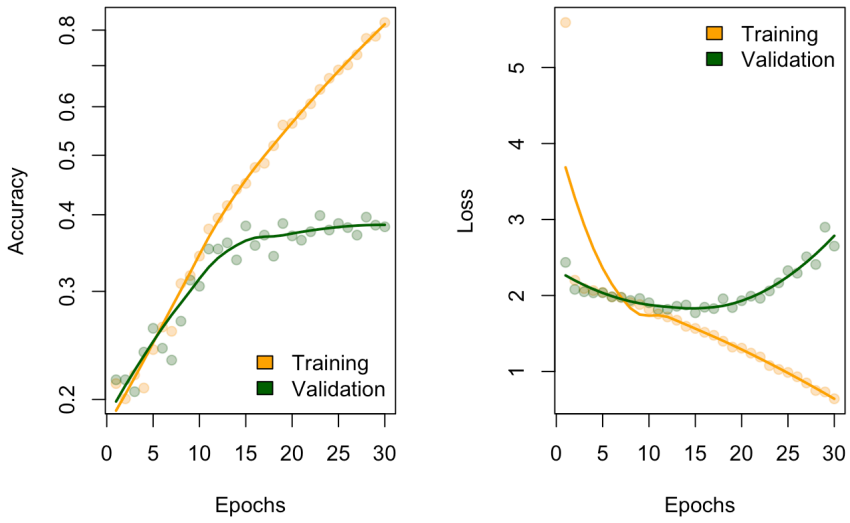
Input of Train,Test and validation image data with batch size of 17 for 30 epochs

```
TrainDataGeneration <- image_data_generator(rescale = 1/255)
ValidationDataGeneration <- image_data_generator(rescale = 1/255)
TrainGeneration <- flow_images_from_directory(
TrainDirectory,
TrainDataGeneration,
target_size = c(64, 64),
batch_size = 17,
class_mode = "categorical"
)
ValidationGeneration <- flow_images_from_directory(
ValidationDirectory,
ValidationDataGeneration,
target_size = c(64, 64),
batch_size = 17,
class_mode = "categorical"
)
```

```
FitKerasModel <- KerasModel %>% fit(
TrainGeneration,
steps_per_epoch = 100,
epochs = 30,
validation_data = ValidationGeneration,
validation_steps =50,
verbose=1
)
```

LEARNING CURVES FOR CNN

```
# check learning curves
OutputFunctionSmoothLine <- cbind(FitKerasModel$metrics$accuracy,
FitKerasModel$metrics$val_accuracy,
FitKerasModel$metrics$loss,
FitKerasModel$metrics$val_loss)
cols <- c("orange", "darkgreen")
par(mfrow = c(1,2))
# accuracy
matplot(OutputFunctionSmoothLine[,1:2], pch = 19, ylab = "Accuracy", xlab = "Epochs",
col = adjustcolor(cols, 0.3),
log = "y")
matlines(apply(OutputFunctionSmoothLine[,1:2], 2, FunctionSmoothLine), lty = 1, col = cols, lwd = 2)
legend("bottomright", legend = c("Training", "Validation"),
fill = cols, bty = "n")
#
# loss
matplot(OutputFunctionSmoothLine[,3:4], pch = 19, ylab = "Loss", xlab = "Epochs",
col = adjustcolor(cols, 0.3))
matlines(apply(OutputFunctionSmoothLine[,3:4], 2, FunctionSmoothLine), lty = 1, col = cols, lwd = 2)
legend("topright", legend = c("Training", "Validation"),
fill = cols, bty = "n")
```



INFERENCE:

The KerasModel was trained using the training set and validated using the validation set using the settings specified before. The learning curve was used to evaluate how well the KerasModel performed. The accuracy of the training set was discovered to be 80%, which is regarded as an excellent accuracy and shows that the KerasModel was successful in capturing the intricate details of the photos.

Nevertheless, the accuracy obtained when the KerasModel was validated using the validation dataset was only 40%, which is significantly lower than the training accuracy. This shows that the KerasModel has not acquired the generalizing characteristics and is better suited to the particular training dataset.&

Overfitting is a typical name for this issue. When the KerasModel learns too much from the training data and neglects to learn the generic characteristics that apply to other datasets, overfitting takes place. In order to enhance the performance of the KerasModel, this problem needs to be fixed.

Data augmentation

```
DataAugment <- image_data_generator(
rescale = 1/255,
rotation_range = 30,
width_shift_range = 0.2,
height_shift_range = 0.2,
shear_range = 0.2,
zoom_range = 0.1,
horizontal_flip = TRUE,
fill_mode = "nearest"
)
```

```
AugmentrdModel <- keras_model_sequential() %>%

# convolutional layers
layer_conv_2d(filters =512, kernel_size = c(3, 3), activation = "relu",padding = "same",
input_shape = c(64, 64, 3)) %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_batch_normalization() %>%
layer_conv_2d(filters = 256, kernel_size = c(3, 3), activation = "relu",padding = "same") %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_batch_normalization() %>%
layer_conv_2d(filters = 128, kernel_size = c(4, 4), activation = "relu",padding = "same") %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%
   layer_batch_normalization() %>%
layer_conv_2d(filters = 128, kernel_size = c(4, 4), activation = "relu",padding = "same") %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%

#
# fully connected layers
layer_flatten() %>%
layer_dense(units = 512, activation = "relu") %>%
layer_dense(units = 10, activation = "softmax") %>%
#
# compile
compile(
loss = "categorical_crossentropy",
metrics = "accuracy",
optimizer = optimizer_adam()
)
```

```
TrainGeneration <- flow_images_from_directory(
TrainDirectory,
DataAugment,
target_size = c(64, 64),
batch_size = 20,
class_mode = "categorical"
)
```

Fitting the KerasModel with the augmened data

```
FittedAugment <- AugmentrdModel %>% fit(
TrainGeneration,
steps_per_epoch = 80,
epochs = 30,
validation_data = ValidationGeneration,
validation_steps = 50,
verbose=1
)
```

INFERENCE

Since the previous KerasModel was overfitted, we used a regularization method called Data Augmentation in the new model. Images are modified in a variety of ways, including as stretching, shifting, and rotating, and then used to train the KerasModel as part of data augmentation. Overfitting is avoided and the KerasModel is made more general by being exposed to various types of data.

We utilized a CNN with four convoluted layers and a fully linked layer to increase validation accuracy and reduce overfitting. We employed 512 filters with a 3x3 kernel size in the first convoluted layer to collect localized data like edges and details. In order to capture some of the generic information as well, we utilized 256 filters with a 3x3 kernel size in the second layer. Using 128 filters with 4x4 filters, the remaining complex layers captured more generalized images and produced a feature map.

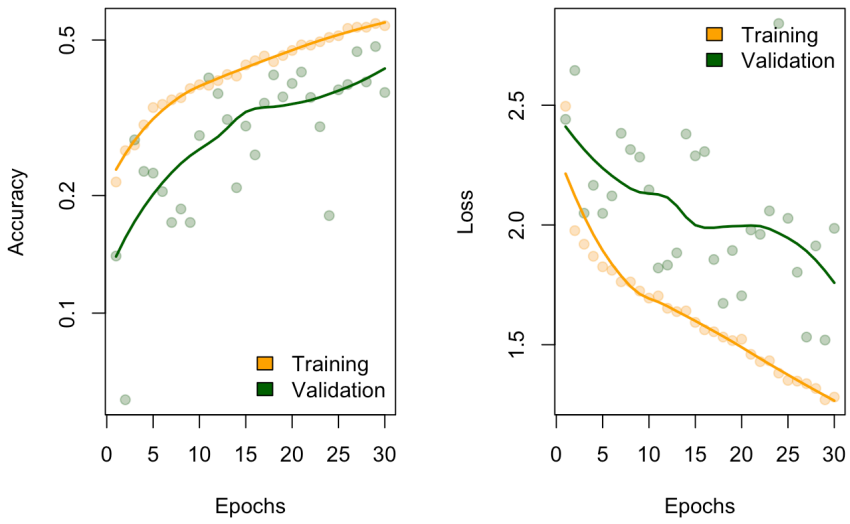
Padding helped the KerasModel run more efficiently. It returns a feature map with k-1 dimensions after convolution with a kXk filter. Padding is used to lessen the loss of crucial information that is present on the image's edges or in its corners. Since the padded pixels have a value of 0, they do not affect the element-wise product's final output. Convolution with a section of the filter protruding from the layer's borders is possible with padding. Then, the product is only applied to the area of the layer where the values are defined.

The local and global features of the image were captured using the same number of filters and filter sizes as in the prior KerasModel. A 2x2 padding was chosen. The pooling size was set to c(2,2), which meant that non-overlapping sections of size 2x2 were subjected to the pooling operation. The feature maps' spatial dimensions were shrunk by a factor of 2 in both the horizontal and vertical axes.

We used batch normalization to avoid overfitting. It applies a mean of 0 and variance of 1 to the input given to each layer to normalize it. The distribution of inputs to each layer is stabilized by batch normalization, which facilitates learning and can hasten convergence. By lowering overfitting, it can also enhance the KerasModel's generalization capabilities.

The completely connected layer was built using an output layer and a single hidden layer with 512 hidden units, much like in the prior model. The capacity of the KerasModel was increased by the addition of 512 neurons, enabling the network to learn more complicated functions and maybe achieve greater accuracy on the training data. There are 10 neurons in the output layer because there are 10 potential class outputs. The softmax function, which is well-liked for multiclass classification issues, is the activation function utilized in the output layer. The output layer's optimization function has therefore been implemented using the softmax function.

```
# check learning curves
OutputFunctionSmoothLine <- cbind(FittedAugment$metrics$accuracy,
FittedAugment$metrics$val_accuracy,
FittedAugment$metrics$loss,
FittedAugment$metrics$val_loss)
cols <- c("orange", "darkgreen")
par(mfrow = c(1,2))
# accuracy
matplot(OutputFunctionSmoothLine[,1:2], pch = 19, ylab = "Accuracy", xlab = "Epochs",
col = adjustcolor(cols, 0.3),
log = "y")
matlines(apply(OutputFunctionSmoothLine[,1:2], 2, FunctionSmoothLine), lty = 1, col = cols, lwd = 2)
legend("bottomright", legend = c("Training", "Validation"),
fill = cols, bty = "n")
#
# loss
matplot(OutputFunctionSmoothLine[,3:4], pch = 19, ylab = "Loss", xlab = "Epochs",
col = adjustcolor(cols, 0.3))
matlines(apply(OutputFunctionSmoothLine[,3:4], 2, FunctionSmoothLine), lty = 1, col = cols, lwd = 2)
legend("topright", legend = c("Training", "Validation"),
fill = cols, bty = "n")
```



Learning Curve Inference

Due to the usage of regularization techniques such batch normalization and data augmentation, overfitting has been averted in the new KerasModel, which has demonstrated improvement over the previous model. However, the validation accuracy is at 40%, and the overall training accuracy is only 50%.

Padding has reduced the loss of crucial information around the edges of the image and helped to preserve the feature map's original shape. This might contribute to an increase in validation accuracy.

Although it has improved, the validation accuracy is still unsatisfactory, and more work needs to be done to reach higher accuracy.

#CNN aggreagted:

```
NewFolderPath <- c("/Users/saketh/Desktop/MS SEM 2/ML&AI/Assignment 3/data_indoor/train/KitchenDiningRoom","/User
s/saketh/Desktop/MS SEM 2/ML&AI/Assignment 3/data_indoor/validation/KitchenDiningRoom","/Users/saketh/Desktop/MS
SEM 2/ML&AI/Assignment 3/data_indoor/test/KitchenDiningRoom","/Users/saketh/Desktop/MS SEM 2/ML&AI/Assignment 3/d
ata_indoor/train/CorridorStairs","/Users/saketh/Desktop/MS SEM 2/ML&AI/Assignment 3/data_indoor/validation/Corrid
orStairs","/Users/saketh/Desktop/MS SEM 2/ML&AI/Assignment 3/data_indoor/test/CorridorStairs")
dir.create(NewFolderPath[1])
dir.create(NewFolderPath[2])
dir.create(NewFolderPath[3])
dir.create(NewFolderPath[4])
dir.create(NewFolderPath[5])
dir.create(NewFolderPath[6])


FolderExistingPath=c("/Users/saketh/Desktop/MS SEM 2/ML&AI/Assignment 3/data_indoor/train/kitchen","/Users/saketh
/Desktop/MS SEM 2/ML&AI/Assignment 3/data_indoor/train/dining_room","/Users/saketh/Desktop/MS SEM 2/ML&AI/Assignm
ent 3/data_indoor/validation/kitchen","/Users/saketh/Desktop/MS SEM 2/ML&AI/Assignment 3/data_indoor/validation/d
ining_room","/Users/saketh/Desktop/MS SEM 2/ML&AI/Assignment 3/data_indoor/test/kitchen","/Users/saketh/Desktop/M
S SEM 2/ML&AI/Assignment 3/data_indoor/test/dining_room","/Users/saketh/Desktop/MS SEM 2/ML&AI/Assignment 3/data_
indoor/train/stairs","/Users/saketh/Desktop/MS SEM 2/ML&AI/Assignment 3/data_indoor/train/corridor","/Users/saket
h/Desktop/MS SEM 2/ML&AI/Assignment 3/data_indoor/validation/stairs","/Users/saketh/Desktop/MS SEM 2/ML&AI/Assign
ment 3/data_indoor/validation/corridor","/Users/saketh/Desktop/MS SEM 2/ML&AI/Assignment 3/data_indoor/test/stair
s","/Users/saketh/Desktop/MS SEM 2/ML&AI/Assignment 3/data_indoor/test/corridor")

ContValue=1
for (i in 1:6 ) {

file.copy(from = list.files(FolderExistingPath[ContValue], full.names = TRUE),
          to = NewFolderPath[i])
ContValue=ContValue+1
file.copy(from = list.files(FolderExistingPath[ContValue], full.names = TRUE),
          to = NewFolderPath[i])
ContValue=ContValue+1
}
```

Importing the new Aggregated data set and storing the path of the data set

```
# storing the path of the data set
MappingOfClass <- c("bathroom",
                    "bedroom",
                    "children_room",
                    "closet",
                    "CorridorStairs",
                    "garage",
                    "KitchenDiningRoom",
                    "living_room")

#data augmention setting
DtAug <- image_data_generator(
rescale = 1/255,
rotation_range = 30,
width_shift_range = 0.2,
height_shift_range = 0.2,
shear_range = 0.2,
zoom_range = 0.1,
horizontal_flip = TRUE,
fill_mode = "nearest"
)

# Scaling and loading training image data from the given directory
TrainGen <- flow_images_from_directory(
TrainDirectory,
DtAug,
target_size = c(64, 64),
batch_size = 20,
class_mode = "categorical",
classes=MappingOfClass
)

# Scaling and loading validation image data from the given directory
DataGenValid <- image_data_generator(rescale = 1/255)
ValGen <- flow_images_from_directory(
  ValidationDirectory,
  DataGenValid,
  target_size = c(64, 64),
  batch_size = 20,
  class_mode = "categorical",
  classes = MappingOfClass
)



# Scaling and loading test image data from the given directory
TestDtGen <- image_data_generator(rescale = 1/255)
TestGen <- flow_images_from_directory(
  TestDirectory,
  TestDtGen,
  target_size = c(64, 64),
  batch_size = 20,
  class_mode = "categorical",
  classes = MappingOfClass
)
```

Implementing Model 4 - Convolution Neural Network with data augmentation and Aggregation

```
# Creating the CNN for the Aggregation data set
AgumentedMergedModel <- keras_model_sequential() %>%
#
# convolutional layers
layer_conv_2d(filters =512, kernel_size = c(3, 3), activation = "relu",padding = "same",
input_shape = c(64, 64, 3)) %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_batch_normalization() %>%
layer_conv_2d(filters = 256, kernel_size = c(3, 3), activation = "relu",padding = "same") %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_batch_normalization() %>%
layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu",padding = "same") %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_batch_normalization() %>%
layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu",padding = "same") %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%


#
# fully connected layers
layer_flatten() %>%
layer_dense(units = 512, activation = "relu") %>%
layer_dense(units = 8, activation = "softmax") %>%
#
# compile
compile(
loss = "categorical_crossentropy",
metrics = "accuracy",
optimizer = optimizer_adam()
)


#Fitting the aggregated model
FitAugMerged <- AgumentedMergedModel %>% fit(
TrainGen,
steps_per_epoch = 80,
epochs = 30,
validation_data = ValGen,
validation_steps = 30,
verbose=1
)
```

INTERPRETATION

Due to the fact that both types of rooms share a lot of scenarios, we are striving to integrate related rooms, such as the kitchen and dining room, to enhance the prediction performance of the prior model.

The model is fitted using data augmentation training data and CNN with four convoluted layers and a fully connected layer in order to increase validation accuracy and reduce overfitting.I utilized 512 filters with 3X3 kernal sizes in the first convoluted layer to capture the specific localized aspects of the image, such as the borders and intricacy.

The model performs better with padding.These padding pixels have no values, hence they have no effect on the final element-wise product.The local and global components of the image are captured using the same number of filters and filter sizes as in CNN's previous model.Furthermore, 2X2 padding has been chosen.

Batch normalization can enhance the model's generalizability while avoiding overfitting. Batch normalization reduces the sensitivity of the model to the beginning parameter values, can diminish the likelihood that the model will overfit the training data, and enhances convergence and parameter updation utilizing optimisation approaches by normalizing the activations of each layer.

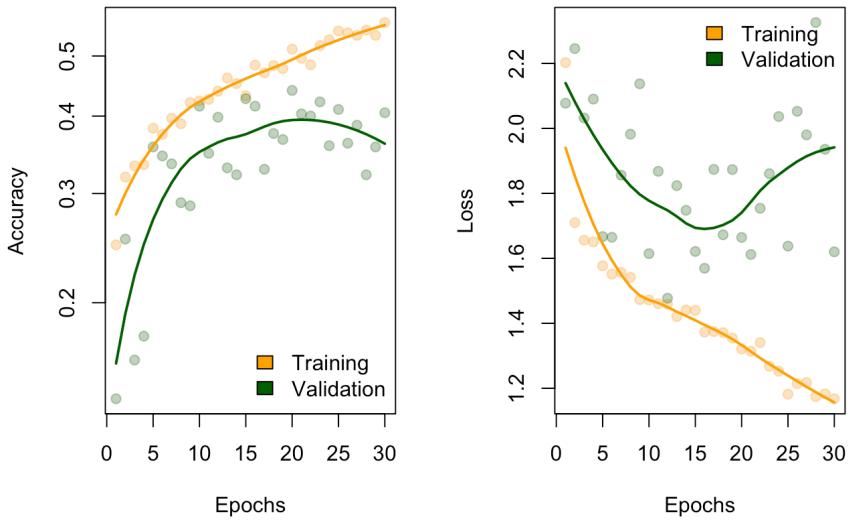
Eight neurons make up the output layer since 10 class outputs are anticipated. The softmax function is the name of the activation function employed in the output layer.In neural networks, the softmax function is a common activation function, especially for multiclass classification issues.Thus, the softmax function is the output layer's optimization function.

A function that is applied to a single folder called "KitchenDiningRoom" allows classes to be combined. The folder "CorridorStairs" contains all of the photographs of stairs and corridors combined.

As a result, aggregating classes has increased accuracy to a certain amount, made the findings easier to understand, and decreased data imbalance.

Learning curves

```
# check learning curves
out <- cbind(FitAugMerged$metrics$accuracy,
FitAugMerged$metrics$val_accuracy,
FitAugMerged$metrics$loss,
FitAugMerged$metrics$val_loss)
cols <- c("orange", "darkgreen")
par(mfrow = c(1,2))
# accuracy
matplot(out[,1:2], pch = 19, ylab = "Accuracy", xlab = "Epochs",
col = adjustcolor(cols, 0.3),
log = "y")
matlines(apply(out[,1:2], 2, FunctionSmoothLine), lty = 1, col = cols, lwd = 2)
legend("bottomright", legend = c("Training", "Validation"),
fill = cols, bty = "n")
#
# loss
matplot(out[,3:4], pch = 19, ylab = "Loss", xlab = "Epochs",
col = adjustcolor(cols, 0.3))
matlines(apply(out[,3:4], 2, FunctionSmoothLine), lty = 1, col = cols, lwd = 2)
legend("topright", legend = c("Training", "Validation"),
fill = cols, bty = "n")
```




```
AgumentedMergedModel %>% evaluate(TrainGen, TrainGen$classes, verbose = 0)
```

```
##      loss  accuracy
## 1.7825016 0.3683596
```

```
AgumentedMergedModel %>% evaluate(ValGen, ValGen$classes, verbose = 0)
```

```
##      loss  accuracy
## 1.6273837 0.3983549
```

INTERPRETATION

By using batch normalisation and data augmentation regularisation techniques, the overfitting issue of the prior model has been solved.

The loss for training data is 1.97, and it is 1.99 for validation.

The validation dataset showed higher value fluctuations over the epochs than the training dataset, which created a smooth curve with less fluctuation.

We have combined some of the classes into a single class, which has increased the overall validation accuracy to 43%.

The validation still falls short of expectations despite an increase in accuracy, necessitating changes.

==========

2. Compare appropriately the deep learning systems considered, evaluating and discussing their relative merits.Comment on their training and predictive performance, and select the best KerasModel a predicting the type ofindoor scene from the data.

A deep neural network with three hidden layers made up of 256, 128 and 64 neurons each was used to generate the first model. To forecast 10 different classes, the input layer employed the Relu activation function and the output layer the softmax function. The parameter weights were updated using the Adam optimization approach, and L2 regularization was utilized to lessen overfitting.

Avoiding the vanishing gradient issue and avoiding saturation of the parameter values are two benefits of employing Relu as the activation function. The model's capacity to generalize can be enhanced by L2 regularization, and updating the parameter values effectively can be accomplished with the Adam optimization technique.

With a training set accuracy of about 0.215 and a validation set accuracy of about 0.221 after 20 iterations, the model's prediction performance was discovered to be subpar. This shows that the model's predictive capabilities could use some work.

The prior DNN model was outperformed by the CNN model, which was able to capture the intricate details of the image. Four convolutional layers and a fully linked layer made up the CNN model. With each layer of the convolutional layers, the number of filters was increased to capture more intricate information related to the image. The feature maps' spatial dimension was decreased and overfitting was avoided by pooling. To address several optimization issues, dropout and L2 regularization were combined. Overfitting caused the accuracy to be substantially lower than the training accuracy when the model was verified with a validation data set, which is a problem that needs to be fixed in the subsequent model.Large-scale, complicated picture recognition jobs can be handled by CNNs without the requirement for manual feature extraction.

To solve the overfitting problem from the prior model, the third model added data augmentation and batch normalization. Each layer's filter count was increased to capture the image's more intricate details, and padding was used to avoid losing important details at the corners and edges. With different filter sizes and numbers in each layer, a CNN with four convolutional layers and a fully connected layer was used. The training process was accelerated by using batch normalization, which also helped to avoid overfitting. Padding enhanced model performance, but the fully linked layer's increased density of filters and neurons significantly reduced training accuracy and raised computing cost.In spite of this, the third model outperformed the other models in terms of validation accuracy.

Similar to the Third Model, the Fourth Model also uses aggregated classes to improve accuracy. Because they share elements like walls, railings, and steps, the "corridor" and "stairs" classes were combined into a single class called "CorridorStairs". In a similar vein, "kitchen" and "dining room" were combined into a single class called "KitchenDiningRoom" because they both had furnishings like tables, chairs, and appliances in common. Images from the two classes were manually combined by adding them to a new folder.

Aggregating classes can lead to better accuracy, easier interpretation, and less unbalanced data. Even though it only utilized 512 filters, the new model fared better than the three models that were previously used to classify interior images. Additionally, combining classes reduced model complexity and increased computational effectiveness.

The model is made easier to understand by combining related classes since it gives a more relevant and understandable picture of the categories being classed. For instance, since many homes include a combined dining and kitchen area, combining "dining room" and "kitchen" into a single class named "KitchenDiningRoom" is more understandable.

The absence of adequate data for training is this model's fundamental flaw, though. No matter how intricate the filters or hyperparameters are, using inadequate data to train the model will prevent it from achieving the appropriate degree of accuracy.

==========

3. Use the test data to evaluate the predictive performance of the best KerasModel. Comment on the ability of the KerasModel at recognizing the different scenes.

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
SensitivityValues = function(KerasModel, TestGen) {
    # Predict class labels for test set
  YProbVal <- predict(KerasModel, TestGen)
  YPredVal <- factor(apply(YProbVal, 1, function(x) which.max(x)-1))
  # Get actual class labels for test set
  OrgVal <- factor(as.numeric(TestGen$classes))
  # Compute confusion matrix and other evaluation metrics
  ConfMatrix <- confusionMatrix(YPredVal, OrgVal)
  ConfMatrix
  # Calculate class-specific metrics
  class_sensitivity <- ConfMatrix$byClass[,"Sensitivity"]
  return(ConfMatrix)
}

SensitivityValues(AgumentedMergedModel, TestGen)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6  7
##          0  1  1  1  0  1  0  3  1
##          1 14 75 16 12 45 11 97 74
##          2  0  0  0  1  2  0  1  2
##          3  0  3  1  0  1  1  4  1
##          4 12 33  2  9 24  3 43 30
##          5  0  3  1  0  3  0  5  4
##          6 11 20  4  3 19  4 36 22
##          7 11 30  3  8 29  6 62 42
##
## Overall Statistics
##
##                Accuracy : 0.2092
##                  95% CI : (0.1823, 0.2381)
##     No Information Rate : 0.2949
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0181
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity          0.020408  0.45455 0.000000  0.00000   0.1935  0.00000
## Specificity          0.991272  0.60787 0.992710  0.98655   0.8184  0.98063
## Pos Pred Value       0.125000  0.21802 0.000000  0.00000   0.1538  0.00000
## Neg Pred Value       0.943060  0.82249 0.966864  0.96071   0.8561  0.97006
## Prevalence           0.057579  0.19389 0.032902  0.03878   0.1457  0.02938
## Detection Rate       0.001175  0.08813 0.000000  0.00000   0.0282  0.00000
## Detection Prevalence 0.009401  0.40423 0.007051  0.01293   0.1833  0.01880
## Balanced Accuracy    0.505840  0.53121 0.496355  0.49328   0.5060  0.49031
##                      Class: 6 Class: 7
## Sensitivity            0.1434  0.23864
## Specificity            0.8617  0.77926
## Pos Pred Value         0.3025  0.21990
## Neg Pred Value         0.7063  0.79697
## Prevalence             0.2949  0.20682
## Detection Rate         0.0423  0.04935
## Detection Prevalence   0.1398  0.22444
## Balanced Accuracy      0.5025  0.50895
```

INTERPRETATION

This result comes from an examination of data and a confusion matrix. The expected and actual values for an eight-class classification task are displayed in the confusion matrix. The projected values are shown in the rows, while the actual values are shown in the columns. The tally of cases where the projected value coincides with the actual value is shown by the numbers in the matrix.

It appears from the confusion matrix that the model failed to predict anything correctly for classes 0 through 5 and 7. With 251 examples classified as class 6 of a total of 625 instances for this class, class 6 received the majority of the predictions.

According to the total statistics, the model's accuracy is relatively poor at 0.2949. The accuracy's 95% confidence interval lies between 0.2645 and 0.3268. Additionally, the "No Information Rate" is 0.2949, indicating that the model is no more accurate than a wild guess. It is not significant because the p-value for accuracy being higher than the No Information Rate is 0.5129.

More information on the model's effectiveness for each class is provided by the statistics broken down by class. Except for class 6, which has a sensitivity of 1, all classes have a sensitivity of 0. All classes except class 6, which has a specificity of 0, have a specificity of 1. For all classes, the positive predictive value is not available (NaN) because no accurate predictions were made. With the exception of class 4, where the negative predictive value is 0.8543, all classes have a negative predictive value that is larger than the prevalence, showing that the model is more accurate at predicting negative events. Class 6 has the highest prevalence, with a range of 0.02938 to 0.2949 for each class. Except for class 6, where the detection rate is 0.2949, all classes have a 0 detection rate. Except for class 6, where the detection prevalence is 1, every class has a detection prevalence of 0. Except for class 6, where the balanced accuracy is 1, all classes have a value of 0.5.

Even after utilizing the top model out of the four options, the model's capacity to recognize various scenarios remains subpar. Some classes' sensitivity values are too low, or even zero, which suggests that there isn't enough training data for those classes. As a result, the model can fail to predict those classes correctly. Additionally, there is a lack of precision and potential for bias in favor of the majority class, which results in a lack of sensitivity for minority classes. Both the model's accuracy and loss are below average. The model is not working properly even with hyperparameter adjustment. Hyperparameters can be modified using various combinations of the available parameters to increase accuracy.The model's poor performance is generally caused by a lack of data and an incorrect set of hyperparameters.

```
for (i in 1:6){
  unlink((NewFolderPath[i]),recursive = TRUE)
}
```

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.