Machine Learning Assignment 3

16343261

26/04/2023

To begin this assignment, we will first load in our necessary packages. Furthermore we will generate our training and our validation data.

```
library(keras)
library(jpeg)
setwd("C:/Users/matth/Documents/Machine Learning & AI/Assignment3")
```

```
train_dir <- "C:/Users/matth/Documents/Machine Learning & AI/Assignment3/train"
validation_dir <- "C:/Users/matth/Documents/Machine Learning & AI/Assignment3/validation"
train_datagen <- image_data_generator(rescale = 1/255)
validation_datagen <- image_data_generator(rescale = 1/255)
```

Task 1 and 2.

My approach for this assignment will be to fit 4 completely different models to explore whether a particular methodology yields particularly good results. This is beneficial for three reasons:

1. It quickly identifies a solid direction to build an accurate model upon.
2. It samples a broad range of methodology and allows myself to gain experience with these tools.
3. Knowledge of the length of their fitting is also useful for understanding the computational efficency.

Model 1. To begin these first 2 tasks I will go through a description of each model. To avoid having to run our code in Rmarkdown. I have saved my findings and will load them in from my directory. Rerunning the code in this assignment should result in the same findings.

```
load(file="fit1.Rdata")
```

Our first model is a Convolutional neural network categorical class model with:

- 4 convolutional hidden layers.
- All 4 of these layers are rectified linear units.
- Filters increase from 16, 32, 64 to 256.
- A pool size of (2,2).
- Adam optimiser with learning rate 0.0001.
- Batch size = 20.
- 2 fully connected layers our first relu and our second softmax with 10 class outputs.

These values have been chosen as they represent a "base case" for a simple architecture Convolutional Neural Network. This will allow for comparison to our other models.

We will first generate our training and validation data from our directory with RGB tensors of width/height of 64/64 with batch size 20.

```
train_generator <- flow_images_from_directory(
train_dir,
train_datagen,
target_size = c(64, 64),
batch_size = 20,
class_mode = "categorical"
)

validation_generator <- flow_images_from_directory(
validation_dir,
validation_datagen,
target_size = c(64, 64),
batch_size = 20,
class_mode = "categorical"
)
```

```
model1 <- keras_model_sequential() %>%
layer_conv_2d(filters = 16, 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 = 32, kernel_size = c(3, 3), activation = "relu") %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%
layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu") %>%
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)) %>%
# fully connected layers
layer_flatten() %>%
layer_dense(units = 256, activation = "relu") %>%
layer_dense(units = 10, activation = "softmax") %>%
# compile
compile(
loss = "categorical_crossentropy",
metrics = "accuracy",
optimizer = optimizer_adam(learning_rate = 0.0001)
)
```

We fit our first model using 75 steps per epoch, 75 epochs and 25 validation steps.

```
fit1 <- model1 %>% fit(
train_generator,
steps_per_epoch = 75,
epochs = 75,
validation_data = validation_generator,
validation_steps = 25
)
```

Our model has been fitted successfully.

```
load(file="fit2.Rdata")
```

Model 2. Our second model is a Deep neural network categorical class model with:

- 3 hidden layers.
- The first layer here is a sigmoid layer, the subsequent two are Relu.
- Units increase from 8, 16, 32 to 256.
- A kernal regularizer with lambda 0.01.
- Adam optimiser with learning rate 0.0001.
- Batch size = 20.
- 2 fully connected layers our first relu and our second softmax with 10 class outputs.

I have chosen these DNN settings to be a "bridge" for comparison between the methods of models 1 and 3. It contains a combination of sigmoid and relu layers as will be the same in model 3 and roughly the same architecture as model 1 with the same fully connected layers. I have decided on using less units in each layer as well as one less layer to compensate for any computational inefficiency caused by the inclusion of a sigmoid layer.

```
model2 <- keras_model_sequential()%>%
layer_dense(units = 8, input_shape = c(64, 64,3), activation = "sigmoid",
name =  "layer_1", kernel_regularizer = regularizer_l2(0.01)) %>%
layer_dropout(rate = 0.4) %>%
layer_dense(units = 16, activation = "relu", name = "layer_2",
kernel_regularizer = regularizer_l2(0.01)) %>%
layer_dropout(rate = 0.4) %>%
layer_dense(units = 32, activation = "relu", name = "layer_3",
kernel_regularizer = regularizer_l2(0.01)) %>%
layer_dropout(rate = 0.4) %>%
layer_flatten() %>%
layer_dense(units = 256, activation = "relu") %>%
layer_dense(units = 10, activation = "softmax") %>%
compile(loss = "categorical_crossentropy", metrics = "accuracy",
optimizer = optimizer_adam(learning_rate = 0.0001)
  )
```

We fit our first model using 75 steps per epoch, 75 epochs and 25 validation steps.

```
fit2 <- model2 %>% fit(
train_generator,
steps_per_epoch = 75,
epochs = 75,
validation_data = validation_generator,
validation_steps = 25
)
```

Our model has been fitted successfully.

```
load(file="fit3.Rdata")
```

Model 3. For our 3rd model we will first augment our data. The data considered here has a relatively small number of training samples. Because of this, the model is likely to overfit, since there are potentially too few samples to learn from and hence it won't be able to generalise well to new data. Therefore we will implement a model with data augmentation using the image_data_generator function.

```
data2 <- image_data_generator(
rescale = 1/255,
rotation_range = 40,
width_shift_range = 0.2,
height_shift_range = 0.2,
shear_range = 0.2,
zoom_range = 0.2,
horizontal_flip = TRUE,
fill_mode = "nearest"
)
```

With our data augmented we implement our third model, which is a Convolutional neural network categorical class model with:

- 4 convolutional hidden layers.
- Our first and last layer are Relu and our 2 middle layers are sigmoid layers.
- Use larger filters than our initial model increase from 16, 64, 128 to 512.
- A pool size of (2,2).
- RMSprop optimiser with learning rate 0.0001.
- Batch size = 20.
- 2 fully connected layers our first relu and our second softmax with 10 class outputs.

Model 3 will be similar to model 1 but with notable differences. Firstly, we will be using a much larger number of filters, this could result in higher accuracy for our model. Secondly, we will be using an RMSprop optimiser. There is research online saying this optimiser is more efficient than an Adam optimiser. Lastly, we will use sigmoid layers. This will allow for comparison to model 2.

```
model3 <- keras_model_sequential() %>%
layer_conv_2d(filters = 16, 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 = "sigmoid") %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%
layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "sigmoid") %>%
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 = 512, activation = "relu") %>%
layer_dense(units = 10, activation = "softmax") %>%
compile(
```

```
loss = "categorical_crossentropy",
metrics = "accuracy",
optimizer = optimizer_rmsprop(learning_rate = 0.0001)
)
# train data generator with data augmentation
train_generator <- flow_images_from_directory(
train_dir,
data2,
target_size = c(64, 64),
batch_size = 20,
class_mode = "categorical"
)
```

We fit our first model using 75 steps per epoch, 75 epochs and 25 validation steps.

```
fit3 <- model3 %>% fit(
train_generator,
steps_per_epoch = 75,
epochs = 75,
validation_data = validation_generator,
validation_steps = 25
)
```

Our model has been fitted successfully.

```
load(file="fit4.Rdata")
```

Model 4. ResNet-50 is a 50-layer convolutional neural network (48 convolutional layers, one MaxPool layer, and one average pool layer). Residual neural networks are a type of artificial neural network (ANN) that forms networks by stacking residual blocks. This will be the basis for our 4th model. In this case adding more layers should allow more features to be identified and therefore a potentially higher accuracy.

As this model already has 50 convolutional layers, we will only fit one resnet "layer" using the "application_resnet50" function. Here we are implementing 10 classes, (2,2) pool and a relu classification. This is followed by our 2 fully connected layers our first relu with 128 units and our second softmax with 10 class outputs.

```
conv_base <- application_resnet50(
  weights = NULL,
  include_top = TRUE,
  pooling = c(2, 2),
  classes = 10,
  classifier_activation = "relu")

model4 <- keras_model_sequential() %>%
conv_base%>%
# fully connected layers
layer_flatten() %>%
layer_dense(units = 128, activation = "relu") %>%
```

```
layer_dense(units = 10, activation = "softmax") %>%
compile(
loss = "categorical_crossentropy",
metrics = "accuracy",
optimizer = optimizer_adam(learning_rate = 0.0001)
)
```

We fit our first model using 75 steps per epoch, 10 epochs and 25 validation steps. Unlike our previous models, we have been forced to using only 10 epochs as our 4th model is incredibly slow at running.

```
fit4 <- model4 %>% fit(
train_generator,
steps_per_epoch = 75,
epochs = 10,
validation_data = validation_generator,
validation_steps = 25
)
```

Our model has fitted, however each epoch has taken considerably longer than our other models.

Now that we have fitted all 4 of our models, we can now plot each of our model's accuracy and loss against each other for each Epoch. Our first 3 models were fitted for 75 epochs, however it should be noted that our forth model was only run for 10 epochs. Our graph will display it's results for 75 but this should be ignored.

First we will define our straight line function and our learning curves and our 8 necessary colours for both our training and validation findings.

```
# to add a smooth line to points
smooth_line <- function(y) {
x <- 1:length(y)
out <- predict(loess(y ~ x))
return(out)
}

# check learning curves
out <- cbind(fit1$metrics$accuracy,
fit1$metrics$val_accuracy,
fit1$metrics$loss,
fit1$metrics$val_loss)
cols <- c("black", "dodgerblue3")
```

```
out4 <- cbind(out[,1:2],
fit2$metrics$accuracy,
fit2$metrics$val_accuracy,
fit3$metrics$accuracy,
fit3$metrics$val_accuracy,
fit4$metrics$accuracy,
fit4$metrics$val_accuracy,
out[,3:4],
fit2$metrics$loss,
fit2$metrics$val_loss,
```

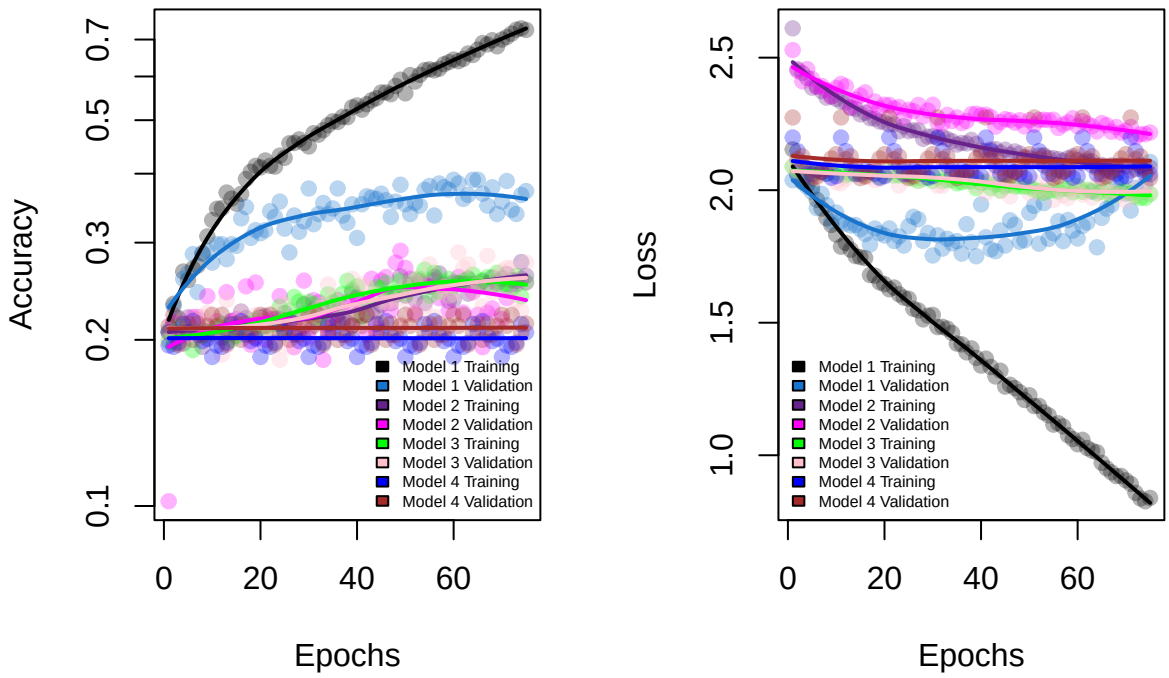
```
fit3$metrics$loss,
fit3$metrics$val_loss,
fit4$metrics$loss,
fit4$metrics$val_loss)
```

```
## Warning in cbind(out[, 1:2], fit2$metrics$accuracy, fit2$metrics$val_accuracy, :
## number of rows of result is not a multiple of vector length (arg 6)
```

```
cols <- c("black", "dodgerblue3", "darkorchid4", "magenta", "green", "pink", "blue", "brown")
```

Next we will use the matplot function to plot our Epochs against our Accuracy. Simultaneously we will plot our epochs against our loss. Following this we will compare our models accordingly.

```
par(mfrow = c(1,2))
# accuracy
matplot(out4[,1:8], pch = 19, ylab = "Accuracy", xlab = "Epochs",
col = adjustcolor(cols, 0.3),
log = "y")
matlines(apply(out4[,1:8], 2, smooth_line), lty = 1, col = cols, lwd = 2)
legend("bottomright", legend = c("Model 1 Training", "Model 1 Validation",
"Model 2 Training", "Model 2 Validation","Model 3 Training",
"Model 3 Validation","Model 4 Training", "Model 4 Validation"),cex = 0.5,
fill = cols, bty = "n")
# loss
matplot(out4[,9:16], pch = 19, ylab = "Loss", xlab = "Epochs",
col = adjustcolor(cols, 0.3))
matlines(apply(out4[,9:16], 2, smooth_line), lty = 1, col = cols, lwd = 2)
legend("bottomleft", legend = c("Model 1 Training", "Model 1 Validation",
"Model 2 Training", "Model 2 Validation","Model 3 Training",
"Model 3 Validation","Model 4 Training", "Model 4 Validation"),cex = 0.5,
fill = cols, bty = "n")
```



From our plot we can find our first model is our most accurate at predicting the type of indoor scene from the data reaching close to 0.7 accuracy after 75 epochs. Our second and third models fall considerably short of this but they parallel in their results at around 0.25 accuracy. Our fourth model is our last with 0.2 accuracy.

The training process looks reasonably bad, with some considerable signs of overfitting as displayed in the loss curves plot. Overall our validation performance is quite low. It is clear with the running of model 1 that more Epochs would have resulted in more higher accuracy runs.

One aspect of this analysis that I believe may have only confused matters is the data augmentation in models 3 and 4. It is unclear whether this augmentation has increased the overfitting in our two respective models and furthermore it has not improved the accuracy of our two models despite "increasing" our generated data. Considering we now know model 1 is our most accurate model, it may be worthwhile fitting this model using our augmented dataset to see if it is both increasing overfitting and perhaps increasing the accuracy of our model.

I've recorded the timing of each epoch's fitting for each model below:

- Model 1: 24 - 32 seconds.
- Model 2: 35 - 45 seconds.
- Model 3: 22 - 30 seconds.
- Model 4: 255 - 311 seconds.

These timings provide a crucial understanding to the fitting of our 4 respective models. From our findings, model 3 fit the fastest. This is interesting as it is the closest architecture to our first model. It has a higher number of filters to model 1, so it would be presumed that this would result in a slower process than model 1. It holds 2 crucial differences. The first is that it contains two sigmoid

layers instead of Relu layers. These layers have a tendancy to cause overfitting and ultimately reduce accuracy of our model. They are likely the explanation for our reduced accuracy in model 3. From my research, these layers also should fit slower than Relu layers. ReLu is faster to compute than the sigmoid function, and its derivative is faster to compute. This therefore would leave us to believe that the RMSprop optimiser is running a more computationally efficient calculation than the Adam optimiser. This is an important finding and may be used to improve the performance of model 1.

It should be noted that Model 2, our only DNN model is running noticibly slower than our CNN models 1 and 3. This to me appears to be caused by the sigmoid layer in it's architecture. It is unclear however why there is such a disparity in timing results. Seeing the performance of model 1, it would be worth running an adjusted model 2 with all relu layers and an RMSprop optimiser for more efficiency. It is clear that less filters in model 2 has not increased computational performance in our model. Both model 2 and 3 do have similar architectures with a mix of Relu and Sigmoid layers with one being a DNN and one being a CNN. Despite these differences, both models achieve similar low accuracy and considerable overfitting.

Lastly Model 4 is simply not a practical model for this assignment or for this student's laptop. My research has shown Resnet-50 to be a popular CNN in a number of industries. This is due to it's ability to efficiently identify a large number of features and potentially resulting in terrific accuracy over a long number of runs on big sample sizes. However it simply takes too long to fit a model of our generated data. Business level computers may have no problem running so many convulutional layers but for students with limited computational resources and time, this is simply not a practical model.

From this analysis it is clear that model 1 would be the best architecture to build upon from our 4 models, perhaps with a method such as batch normalisation and using an RMSprop optimiser. We will now use the test data to evaluate the predictive performance of the best model.

Task 3.

```
load(file="fit1_test.Rdata")
load(file="acc.Rdata")
load(file="tab.Rdata")
```

To create our fit using the test data, we will first have to generate our test data, similar to the process shown at the beginning of this assignment. Furthermore, I have compiled our training and validation data into one folder to act as training data "train_val". I am aware this process technically is not reproducible directly for the corrector, but a simple copy paste of both training and validation folders into one will generate the folder referenced in this assignment. This should not take longer than a minute and therefore I see this process as reasonably reproducible.

```
train_val_dir <- "C:/Users/matth/Documents/Machine Learning & AI/Assignment3/train_val"
train_val_datagen <- image_data_generator(rescale = 1/255)

test_dir <- "C:/Users/matth/Documents/Machine Learning & AI/Assignment3/test"
test_datagen <- image_data_generator(rescale = 1/255)

train_val_generator <- flow_images_from_directory(
train_val_dir,
train_val_datagen,
target_size = c(64, 64),
batch_size = 20,
class_mode = "categorical",
```

```
shuffle = FALSE
)

test_generator <- flow_images_from_directory(
test_dir,
test_datagen,
target_size = c(64, 64),
batch_size = 20,
class_mode = "categorical",
shuffle = FALSE
)
```

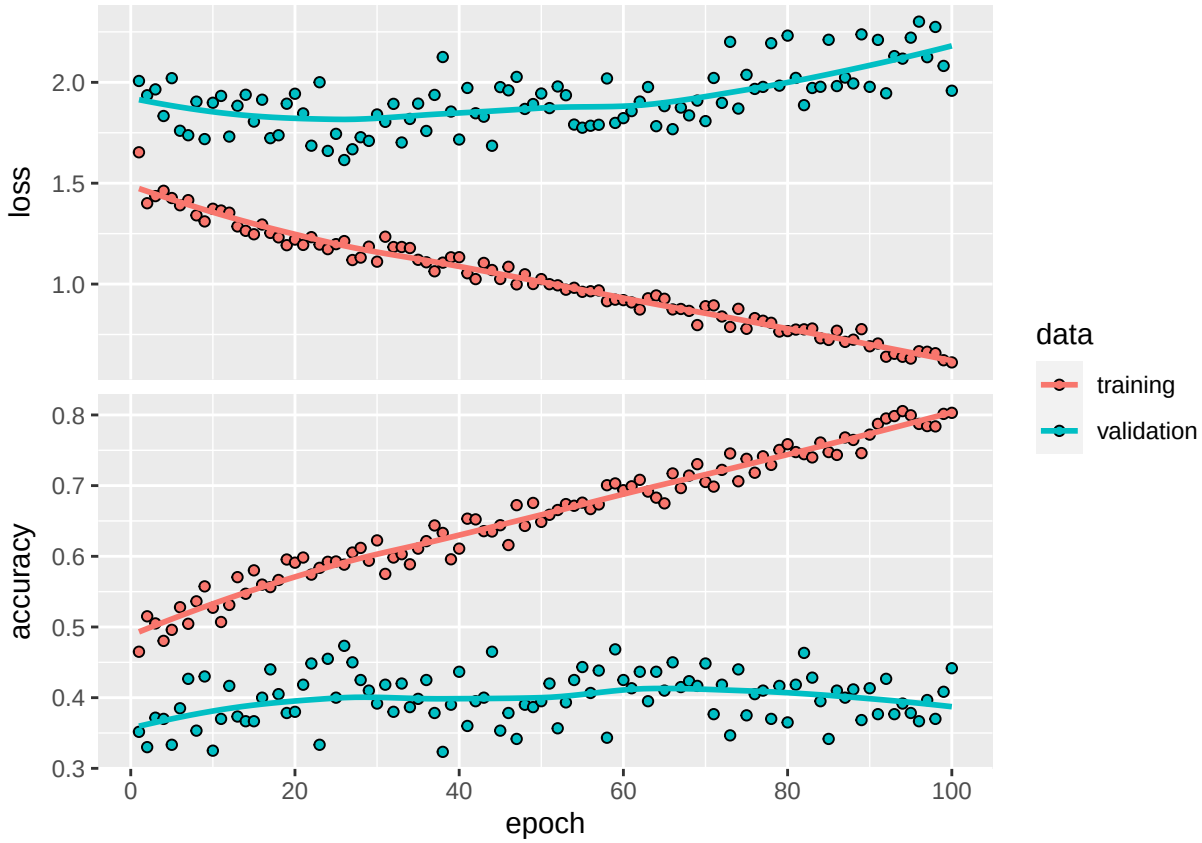
Next we will fit our model to our test data and evaluate predictive performance. To improve our fit, I am running 100 steps per epoch and 100 epochs, as well as 30 validation steps.

```
fit1_test <- model1 %>% fit(
train_val_generator,
steps_per_epoch = 100,
epochs = 100,
validation_data = test_generator,
validation_steps = 30
)
```

Next we will fit our classification and predict our test data.

```
class_hat <- model1 %>% predict(test_generator) %>% k_argmax()+1
y_labels_test <- test_generator$labels +1
# check performance and class-specific performance
tab <- table(y_labels_test, as.vector(class_hat))
acc <- diag(tab)/rowSums(tab)
```

```
plot(fit1_test)
```




```
cbind(tab, acc)
```

```
##      1  2 3  4  5  6 7   8  9 10       acc
## 1   13  8 0  0  2  3 0  15  6  2 0.26530612
## 2    6 68 1  2  5  7 3  45 26  2 0.41212121
## 3    0  5 2  2  2  5 0   7  2  3 0.07142857
## 4    0  1 0 16  5  1 0   5  4  1 0.48484848
## 5    2  8 0  5 53  3 1  10  3  1 0.61627907
## 6    0  9 3  1  2 19 0  16 16  2 0.27941176
## 7    0  1 0  3  4  2 7   3  4  1 0.28000000
## 8    3 19 1  5 12 21 2 101 17  2 0.55191257
## 9    4 29 3  2  2 27 3  52 52  2 0.29545455
## 10   0  6 2  2  4  3 0  12  6  3 0.07894737
```

It should be noted that our model when fitted using both the training and validation data for validation is noticeably more accurate, reaching 0.8 accuracy after 100 runs. Our model ran roughly the same as before at 24 - 32 seconds per epoch.

Here we can find that our best model has alright but not particularly great accuracy at predicting the type of room/scene present in the image. It is best at identifying corridors with 0.616 accuracy, second is kitchens with 0.552 accuracy and third is closets with 0.485 accuracy. A look at the corridors data shows me that many corridors are photographed incredibly similarly, usually illustrating the length of the corridor. This may make them particularly easy to identify. Kitchens may have been easily identifiable due to the reoccurence of appliances. Closets, similar to corridors are shot considerably similarly.

Our model is particularly bad at identifying children's rooms with 0.0714 accuracy and stairs with 0.079. accuracy. Children's rooms may be particularly easy to misidentify due to their similarity to bedrooms. This can be seen in our results with more kitchens and bedrooms misidentifications than children's rooms being identified. It should also be noted that there were less children's room photos in our training data than anything other than garages. I had expected that stairs would be easily identifiable by our model. A potential explanation for the number of mis-identifications is the large preponderance of spiral staircases in the training data.

It still remains that overfitting is considerable. The changes outlined to model 1 at the end of task 2 involving the RMSprop optimiser and the inclusion of batch normalisation could be a solution. This may reduce overfitting due to the regularization of batch normalisation whilst simultaneously reduce computation speed of our model fitting.