Deep Learning

Indrajith Wasala Mudiyanselage

Section 1

Question 01

a) To select the tuning parameters, I created a grid of different parameters, fitted the model for each parameter individually, and then chose the model with the lowest validation error. To reduce the run time The code below (In section 2) just represents the best model.

Model	Training accuracy	Validation accuracy	Test accuracy
Fitted CNN model	0.9987	0.9912	0.9921

Table 1: Tabular summary of errors

```
## Model: "sequential"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                     Param #
## =========================================================================
##  conv2d_2 (Conv2D)                  (None, 26, 26, 32)               320
##  max_pooling2d_1 (MaxPooling2D)     (None, 13, 13, 32)               0
##  conv2d_1 (Conv2D)                  (None, 11, 11, 64)               18496
##  max_pooling2d (MaxPooling2D)       (None, 5, 5, 64)                 0
##  conv2d (Conv2D)                    (None, 3, 3, 64)                 36928
##  flatten (Flatten)                  (None, 576)                      0
##  dense_1 (Dense)                    (None, 64)                       36928
##  dense (Dense)                      (None, 10)                       650
## =========================================================================
## Total params: 93,322
## Trainable params: 93,322
## Non-trainable params: 0
## _________________________________________________________________________
```

b) The CNN model (Proposed in Mini Project 05 Q1 part a) outperform the selected best model in Mini Project 04 (Q1). Therefore, now I recommend this CNN model.

Model	Training loss	Validation loss	Test loss
Fitted CNN model	0.003872	0.04854	0.04262
Best model in Pro. 04	0.02544	0.09726	0.14065

Table 2: Tabular summary of errors

c) Check the model in section 2. In the improved model, I have included padding for each convolution layer and dropout with rate= 0.5. The test accuracy for the improved model is 0.9929 (But, this is a small implement).

d) As further modification we can add more depth to the model. Also we can consider adding data augmentation to the model.

Question 02

a) To select the tuning parameters, I created a grid of different parameters (i.e. batch size={32,64 and 128}), fitted the model for each parameter individually, and then chose the model with the lowest validation error (batch size=128). To reduce the run time, the code below (In section 2) just represents the best model.

Model	Training accuracy	Validation accuracy	Test accuracy
Fitted CNN model	0.9551	0.0..3662	0.3735

Table 3: Tabular summary of errors

```
## Model: "sequential_2"
## _________________________________________________________________________
##  Layer (type)                 Output Shape              Param #
## =========================================================================
##  conv2d_9 (Conv2D)            (None, 32, 32, 32)        896
##  max_pooling2d_7 (MaxPooling2D)  (None, 16, 16, 32)     0
##  conv2d_8 (Conv2D)            (None, 16, 16, 64)        18496
##  max_pooling2d_6 (MaxPooling2D)  (None, 8, 8, 64)       0
##  conv2d_7 (Conv2D)            (None, 8, 8, 128)         73856
##  max_pooling2d_5 (MaxPooling2D)  (None, 4, 4, 128)      0
##  conv2d_6 (Conv2D)            (None, 4, 4, 256)         295168
##  max_pooling2d_4 (MaxPooling2D)  (None, 2, 2, 256)      0
##  flatten_2 (Flatten)          (None, 1024)              0
##  dense_5 (Dense)              (None, 512)               524800
##  dense_4 (Dense)              (None, 100)               51300
## =========================================================================
## Total params: 964,516
## Trainable params: 964,516
## Non-trainable params: 0
## _________________________________________________________________________
```

b) Check the model in section 2. In the improved model, I have added dropout with rate= 0.5. The test accuracy for the improved model is 0.4478.

c) As further modification we can add more depth to the model. Also we can consider adding data augmentation to the model.

Question 03

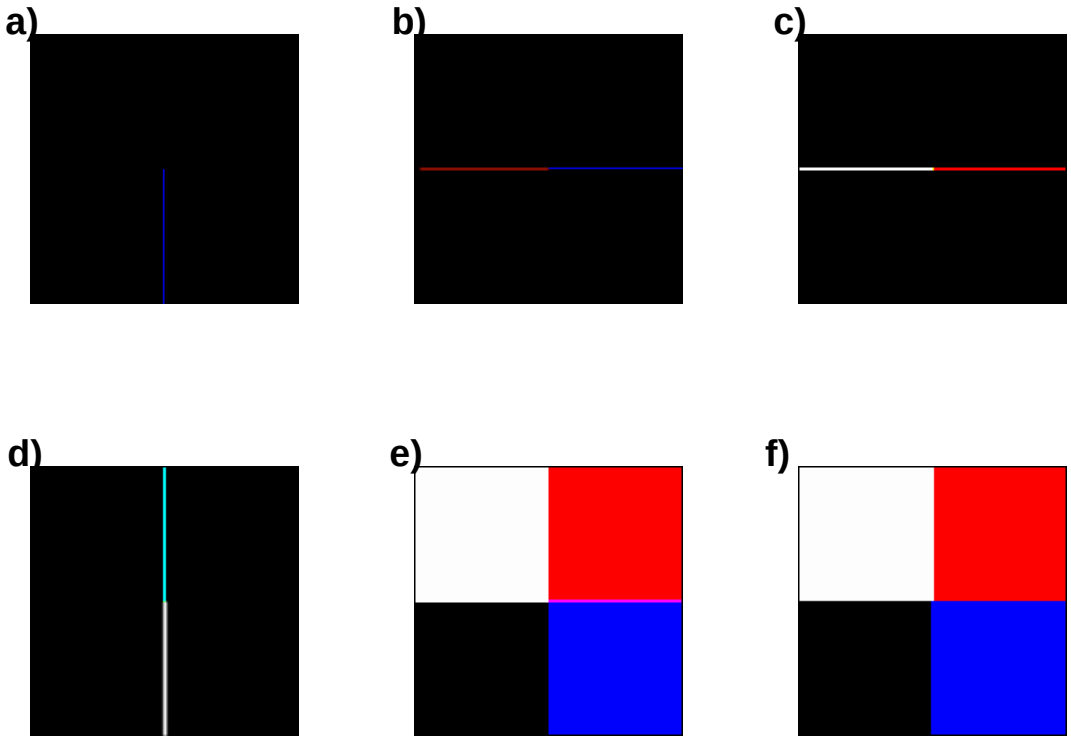

Figure 1: *Output images for part (a) - part (f)*

g) Filters b) and c) can be utilized as horizontal edge detectors since they compute the difference between neighboring horizontal pixels. Filters a) and d), on the other hand, can be employed as vertical edge detectors because they compute the difference between neighboring pixels in the vertical direction.

Question 04

The best prediction is “Airedale” with score of 0.34053415.

Class name	Class description	Score
1) n02096051	Airedale	0.34053415
2) n04399382	Teddy	0.28807443
3) n02113624	Toy Poodle	0.08166376

Table 4: Predicted classes for Bella

Question 05

Figure 2: *Seventh chanel of the activation of the first layer on the Bella picture (Similar to Figure 9.13)*

This channel appears to encode a diagonal edge detector—but note that your own channels may vary, because the specific filters learned by convolution layers aren’t deterministic.

Figure 3: *Every channel of every layer activation on the Bella picture (Similar to Figure 9.14)*

The sparsity of the activations increases with the depth of the layer: in the first layer, almost all filters are activated by the input image, but in the following layers, more and more filters are blank. This means the pattern encoded by the filter isn't found in the input image. Therefore the results consistent with what we discussed in the class.

Section 2 (R codes)

```
knitr::opts_chunk$set(echo = TRUE}

## ----include=FALSE-----------------------------------------------------------
## Question 01

library(keras)

mnist <- dataset_mnist()
train_images <- mnist$train$x
train_labels <- mnist$train$y
test_images <- mnist$test$x
test_labels <- mnist$test$y


### Preprocess so that the training and test data for features are
### in matrix form (as needed for model fitting)

train_images <- train_images/255 # ensures all values are in [0, 1]
test_images <- test_images/255

### Make the response categorical

train_labels <- to_categorical(train_labels)
test_labels <- to_categorical(test_labels)


## ----echo=FALSE--------------------------------------------------------------
## a)

# set up a moderately sized CNN

model_1a <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu",
      input_shape = c(28, 28, 1)) %>%
  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 = 64, kernel_size = c(3, 3), activation = "relu") %>%
  layer_flatten() %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 10, activation = "softmax")
summary(model_1a)


## ----eval=FALSE, include=FALSE-----------------------------------------------
## # compile the model

model_1a %>% compile(loss = "categorical_crossentropy",
    optimizer = optimizer_rmsprop(), metrics = c("accuracy"))

# fit the model (takes about 10 min to run)

history <- model_1a %>% fit(train_images, train_labels, epochs = 15,
    batch_size = 128, validation_split = 0.2)
```

```r
## ----eval=FALSE, include=FALSE-------------------------------------------------------
## ### Examine test performance
## metrics_test <- model_1a %>% evaluate(test_images, test_labels)


## ----include=FALSE-------------------------------------------------------------------
## c)

model_1c <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3),padding = "same", activation = "relu",
      input_shape = c(28, 28, 1)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3),padding = "same", activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3),padding = "same", activation = "relu") %>%
  layer_flatten() %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 10, activation = "softmax")


## ----eval=FALSE, include=FALSE-------------------------------------------------------
## # compile the model

model_1c %>% compile(loss = "categorical_crossentropy",
    optimizer = optimizer_rmsprop(), metrics = c("accuracy"))

# fit the model (takes about 10 min to run)

history <- model_1c %>% fit(train_images, train_labels, epochs = 15,
    batch_size = 128, validation_split = 0.2)



## ----eval=FALSE, include=FALSE-------------------------------------------------------
## ### Examine test performance
metrics_test_c <- model_1c %>% evaluate(test_images, test_labels)

## ----include=FALSE-------------------------------------------------------------------
################################################################################

## Question 02

# load the cifar100 data and get training and test sets

cifar100 <- dataset_cifar100()
names(cifar100)

x_train <- cifar100$train$x
g_train <- cifar100$train$y
x_test <- cifar100$test$x
g_test <- cifar100$test$y

# scale the pixel values to lie between 0 and 1

x_train <- x_train / 255
x_test <- x_test / 255

# make the response a 100-class vector (one-hot encoding)

y_train <- to_categorical(g_train, 100)
y_test <- to_categorical(g_test, 100)
```

```
## ----echo=FALSE-----------------------------------------------------------
## a)

# set up a moderately sized CNN (see the book)

model_2a <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3),
      padding = "same", activation = "relu",
      input_shape = c(32, 32, 3)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3),
      padding = "same", activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3),
      padding = "same", activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 256, kernel_size = c(3, 3),
      padding = "same", activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dense(units = 512, activation = "relu") %>%
  layer_dense(units = 100, activation = "softmax")
summary(model_2a)



## ----eval=FALSE, include=FALSE--------------------------------------------
# compile the model

model_2a %>% compile(loss = "categorical_crossentropy",
    optimizer = optimizer_rmsprop(), metrics = c("accuracy"))

# fit the model

history2 <- model_2a %>% fit(x_train, y_train, epochs = 30,
    batch_size = 128, validation_split = 0.2)



## ----eval=FALSE, include=FALSE--------------------------------------------
## ### Examine test performance
metrics_test_2a <- model_2a %>% evaluate(x_test, y_test)


## ----eval=FALSE, include=FALSE--------------------------------------------
## ## b)

# set up a moderately sized CNN (see the book)

model_2b <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3),
      padding = "same", activation = "relu",
      input_shape = c(32, 32, 3)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3),
      padding = "same", activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3),
      padding = "same", activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 256, kernel_size = c(3, 3),
      padding = "same", activation = "relu") %>%
```

```
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 512, activation = "relu") %>%
  layer_dense(units = 100, activation = "softmax")
summary(model)


## ----eval=FALSE, include=FALSE-------------------------------------------------------
# compile the model

model_2b %>% compile(loss = "categorical_crossentropy",
    optimizer = optimizer_rmsprop(), metrics = c("accuracy"))

# fit the model

history2b <- model_2b %>% fit(x_train, y_train, epochs = 30,
    batch_size = 128, validation_split = 0.2)



## ----eval=FALSE, include=FALSE-------------------------------------------------------
## ### Examine test performance
metrics_test_2b <- model_2b %>% evaluate(x_test, y_test)


## ----echo=FALSE, fig.cap="\\textit{Output images for part (a) - part (f)}",fig.width=6----
##############################################################################

## Question 03

library(cowplot)
p1 <- ggdraw() + draw_image("fltr_image_a.png", scale = 0.7)
p2 <- ggdraw() + draw_image("fltr_image_b.png", scale = 0.7)
p3 <- ggdraw() + draw_image("fltr_image_c.png", scale = 0.7)
p4 <- ggdraw() + draw_image("fltr_image_d.png", scale = 0.7)
p5 <- ggdraw() + draw_image("fltr_image_e.png", scale = 0.7)
p6 <- ggdraw() + draw_image("fltr_image_f.png", scale = 0.7)


plot_grid(p1, p2, p3, p4,p5,p6, ncol =3 , align = "hv", axis = "tb",
labels = c("a)", "b)", "c)", "d)","e)","f)"),hjust = -1, vjust = 3)


## ----include=FALSE-------------------------------------------------------------------
library(png)
col_image <- readPNG("/Users/indrajithwasalamudiyanselage/Documents/UTD/Academic/5-Spring 2021/STAT 6340/Projects


## ----eval=FALSE, include=FALSE-------------------------------------------------------
## a)

# Define the filter
filter <- matrix(c(-1, 1), nrow=1, ncol=2)

# Convert the image to a suitable format
img <- aperm(col_image, c(2,1,3))

# Apply the filter
filtered_img <- array(0, dim=c(nrow(img), ncol(img), 3))
for (k in 1:3) {
  for (i in 1:(nrow(img)-1)) {
    for (j in 1:ncol(img)) {
```

```
      filtered_img[i,j,k] <- sum(img[i:(i+1), j, k] * filter)
    }
  }
}

# Convert the image back to the original format
filtered_img <- aperm(filtered_img, c(2,1,3))

writePNG(filtered_img, "/Users/indrajithwasalamudiyanselage/Desktop/fltr_image_a.png")


## ----eval=FALSE, include=FALSE-----------------------------------------------
## ## b)

# Define the filter
filter <- t(matrix(c(-1, 1), nrow=2, ncol=1))

# Convert the image to a suitable format
img <- aperm(col_image, c(2,1,3))

# Apply the filter
filtered_img <- array(0, dim=c(nrow(img), ncol(img), 3))
for (k in 1:3) {
  for (i in 1:nrow(img)) {
    for (j in 1:(ncol(img)-1)) {
      filtered_img[i,j,k] <- sum(img[i, j:(j+1), k] * filter)
    }
  }
}

# Convert the image back to the original format
filtered_img <- aperm(filtered_img, c(2,1,3))

writePNG(filtered_img, "/Users/indrajithwasalamudiyanselage/Desktop/fltr_image_b.png")



## ----eval=FALSE, include=FALSE-----------------------------------------------
## ## c)

# Define the filter
filter <- matrix(c(1, 1, 1, 0, 0, 0, -1, -1, -1), nrow=3, ncol=3, byrow = T)
filter

# Convert the image to a suitable format
img <- aperm(col_image, c(2,1,3))

# Apply the filter
filtered_img <- array(0, dim=c(nrow(img), ncol(img), 3))
for (k in 1:3) {
  for (i in 2:(nrow(img)-1)) {
    for (j in 2:(ncol(img)-1)) {
      filtered_img[i,j,k] <- sum(img[(j-1):(j+1), (i-1):(i+1), k] * filter)
    }
  }
}

# Convert the image back to the original format
filtered_img <- aperm(filtered_img, c(2,1,3))

writePNG(filtered_img, "/Users/indrajithwasalamudiyanselage/Desktop/fltr_image_c.png")
```

```r
## ----eval=FALSE, include=FALSE-------------------------------------------------------
## ## d)

# Define the filter
filter <- matrix(c(1, 1, 1, 0, 0, 0, -1, -1, -1), nrow=3, ncol=3)

# Convert the image to a suitable format
img <- aperm(col_image, c(2,1,3))

# Apply the filter
filtered_img <- array(0, dim=c(nrow(img), ncol(img), 3))
for (k in 1:3) {
  for (i in 2:(nrow(img)-1)) {
    for (j in 2:(ncol(img)-1)) {
      filtered_img[i,j,k] <- sum(img[(j-1):(j+1), (i-1):(i+1), k] * filter)
    }
  }
}

# Convert the image back to the original format
filtered_img <- aperm(filtered_img, c(2,1,3))

writePNG(filtered_img, "/Users/indrajithwasalamudiyanselage/Desktop/fltr_image_d.png")

## ----eval=FALSE, include=FALSE-------------------------------------------------------
## ## e)

# Define the filter
filter <- matrix(c(0, 0, 0, 1, 1, 1, 0, 0, 0), nrow=3, ncol=3)

# Convert the image to a suitable format
img <- aperm(col_image, c(2,1,3))

# Apply the filter
filtered_img <- array(0, dim=c(nrow(img), ncol(img), 3))
for (k in 1:3) {
  for (i in 2:(nrow(img)-1)) {
    for (j in 2:(ncol(img)-1)) {
      filtered_img[i,j,k] <- sum(img[(j-1):(j+1), (i-1):(i+1), k] * filter)
    }
  }
}

# Convert the image back to the original format
filtered_img <- aperm(filtered_img, c(2,1,3))

writePNG(filtered_img, "/Users/indrajithwasalamudiyanselage/Desktop/fltr_image_e.png")



## ----eval=FALSE, include=FALSE-------------------------------------------------------
## ## f)

# Define the filter
filter <- matrix(c(0, 0, 0, 1, 1, 1, 0, 0, 0), nrow=3, ncol=3,byrow = T)

# Convert the image to a suitable format
img <- aperm(col_image, c(2,1,3))

# Apply the filter
filtered_img <- array(0, dim=c(nrow(img), ncol(img), 3))
for (k in 1:3) {
  for (i in 2:(nrow(img)-1)) {
```

```
    for (j in 2:(ncol(img)-1)) {
      filtered_img[i,j,k] <- sum(img[(j-1):(j+1), (i-1):(i+1), k] * filter)
    }
  }
}

# Convert the image back to the original format
filtered_img <- aperm(filtered_img, c(2,1,3))

writePNG(filtered_img, "/Users/indrajithwasalamudiyanselage/Desktop/fltr_image_f.png")


## ----eval=FALSE, include=FALSE-------------------------------------------------------
## ##########################################################################################
##
## ## Question 04

## Using a Pretrained CNN Model ###

x <- array(dim = c(1, 224, 224, 3))
img <- image_load("/Users/indrajithwasalamudiyanselage/Documents/UTD/Academic/5-Spring 2021/STAT 6340/Projects/ch

x[1,,, ] <- image_to_array(img)


x <- imagenet_preprocess_input(x)

# get the pretrained model

model <- application_resnet50(weights = "imagenet")

# predict the class of the test images

pred6 <- model %>% predict(x) %>%
  imagenet_decode_predictions(top = 3)
names(pred6) <- image_names
print(pred6)


## ----eval=FALSE, include=FALSE-------------------------------------------------------
## ##########################################################################################
##
## ## Question 05

library(fs)
library(tfdatasets)
#zip::unzip('dogs-vs-cats.zip', exdir = "dogs-vs-cats", files = "train.zip")
#zip::unzip("dogs-vs-cats/train.zip", exdir = "dogs-vs-cats")

original_dir <- path("dogs-vs-cats/train")
new_base_dir <- path("cats_vs_dogs_small")

make_subset <- function(subset_name, start_index, end_index) {
  for (category in c("dog", "cat")) {
    file_name <- glue::glue("{category}.{ start_index:end_index }.jpg")
    dir_create(new_base_dir / subset_name / category)
    file_copy(original_dir / file_name,
              new_base_dir / subset_name / category / file_name)
  }
}

# create training, validation and test sets
make_subset("train", start_index = 1, end_index = 1000)
```

```
make_subset("validation", start_index = 1001, end_index = 1500)
make_subset("test", start_index = 1501, end_index = 2500)



## ----eval=FALSE, include=FALSE-------------------------------------------------------
## # further preprocessing of data
train_dataset <-
  image_dataset_from_directory(new_base_dir / "train",
                               image_size = c(180, 180),
                               batch_size = 32)
validation_dataset <-
  image_dataset_from_directory(new_base_dir / "validation",
                               image_size = c(180, 180),
                               batch_size = 32)
test_dataset <-
  image_dataset_from_directory(new_base_dir / "test",
                               image_size = c(180, 180),
                               batch_size = 32)




## ----eval=FALSE, include=FALSE-------------------------------------------------------
## # add data augmentation
data_augmentation <- keras_model_sequential() %>%
  layer_random_flip("horizontal") %>%
  layer_random_rotation(0.1) %>%
  layer_random_zoom(0.2)


# build a model
inputs <- layer_input(shape = c(180, 180, 3))
outputs <- inputs %>%
  data_augmentation() %>%
  layer_rescaling(1 / 255) %>%
  layer_conv_2d(filters = 32, kernel_size = 3, activation = "relu") %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_conv_2d(filters = 64, kernel_size = 3, activation = "relu") %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_conv_2d(filters = 128, kernel_size = 3, activation = "relu") %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_conv_2d(filters = 256, kernel_size = 3, activation = "relu") %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_conv_2d(filters = 256, kernel_size = 3, activation = "relu") %>%
  layer_flatten() %>%
  layer_dropout(0.5) %>%
  layer_dense(1, activation = "sigmoid")
model <- keras_model(inputs, outputs)


## ----eval=FALSE, include=FALSE-------------------------------------------------------
model %>% compile(loss = "binary_crossentropy",
                  optimizer = "rmsprop",
                  metrics = "accuracy")

callbacks <- list(
  callback_model_checkpoint(
    filepath = "convnet_from_scratch.keras",
    save_best_only = TRUE,
    monitor = "val_loss"
  )
)
```

```r
# fit model

history <- model %>%
  fit(
    train_dataset,
    epochs = 100,
    validation_data = validation_dataset,
    callbacks = callbacks
  )


## ----eval=FALSE, include=FALSE-------------------------------------------------------
## # Visualizing a CNN

model <- load_model_tf("/Users/indrajithwasalamudiyanselage/Documents/UTD/Academic/5-Spring 2021/STAT 6340/Projec


img_path <- get_file(
  fname="Bella_1.jpg",
  origin="/Users/indrajithwasalamudiyanselage/Documents/UTD/Academic/5-Spring 2021/STAT 6340/Projects/check/Bella

img_tensor <- img_path %>%
  tf_read_image(resize = c(180, 180))


## ----eval=FALSE, include=FALSE-------------------------------------------------------
## # functions needed
display_image_tensor <- function(x, ..., max = 255,
                                 plot_margins = c(0, 0, 0, 0)) {
  if(!is.null(plot_margins))
    par(mar = plot_margins)

  x %>%
    as.array() %>%
    drop() %>%
    as.raster(max = max) %>%
    plot(..., interpolate = FALSE)
}



plot_activations <- function(x, ...) {
  x <- as.array(x)

  if(sum(x) == 0)
    return(plot(as.raster("gray")))

  rotate <- function(x) t(apply(x, 2, rev))
  image(rotate(x), asp = 1, axes = FALSE, useRaster = TRUE,
        col = terrain.colors(256), ...)
}


## ----eval=FALSE, include=FALSE-------------------------------------------------------
display_image_tensor(img_tensor)

conv_layer_s3_classname <- class(layer_conv_2d(NULL, 1, 1))[1]
pooling_layer_s3_classname <- class(layer_max_pooling_2d(NULL))[1]

is_conv_layer <- function(x) inherits(x, conv_layer_s3_classname)
is_pooling_layer <- function(x) inherits(x, pooling_layer_s3_classname)

layer_outputs <- list()
```

```
for (layer in model$layers)
  if (is_conv_layer(layer) || is_pooling_layer(layer))
    layer_outputs[[layer$name]] <- layer$output

activation_model <- keras_model(inputs = model$input,
                                outputs = layer_outputs)


activations <- activation_model %>%
  predict(img_tensor[tf$newaxis, , , ])

first_layer_activation <- activations[[ names(layer_outputs)[1] ]]
dim(first_layer_activation)


## ----eval=FALSE, include=FALSE-------------------------------------------------------
plot_activations <- function(x, ...) {

  x <- as.array(x)

  if(sum(x) == 0)
    return(plot(as.raster("gray")))

  rotate <- function(x) t(apply(x, 2, rev))
  image(rotate(x), asp = 1, axes = FALSE, useRaster = TRUE,
        col = terrain.colors(256), ...)
}

plot_activations(first_layer_activation[, , , 7])


## ----echo=FALSE, fig.cap="\\textit{ Seventh chanel of the activation of the first layer on the Bella picture (S
library(cowplot)

fig <- ggdraw() + draw_image("fig.png", scale = 0.5)

plot_grid(fig, ncol =1, align = "h", rel_widths = c(1, 1))

## ----eval=FALSE, include=FALSE-------------------------------------------------------
for (layer_name in names(layer_outputs)) {
  layer_output <- activations[[layer_name]]

  n_features <- dim(layer_output) %>% tail(1)
  par(mfrow = n2mfrow(n_features, asp = 1.75),
      mar = rep(.1, 4), oma = c(0, 0, 1.5, 0))
  for (j in 1:n_features)
    plot_activations(layer_output[, , ,j])
  title(main = layer_name, outer = TRUE)
}

## ----echo=FALSE, fig.cap="\\textit{Every channel of every layer activation on the Bella picture (Similar to Fig
library(cowplot)
p1 <- ggdraw() + draw_image("1.png", scale = 1)
p2 <- ggdraw() + draw_image("2.png", scale = 1)
p3 <- ggdraw() + draw_image("3.png", scale = 1)
p4 <- ggdraw() + draw_image("4.png", scale = 1)
p5 <- ggdraw() + draw_image("5.png", scale = 1)
p6 <- ggdraw() + draw_image("6.png", scale = 1)
p7 <- ggdraw() + draw_image("7.png", scale = 1)
p8 <- ggdraw() + draw_image("8.png", scale = 1)
p9 <- ggdraw() + draw_image("9.png", scale = 1.5)

plot_grid(p1,p2,p3,p4,p5,p6,p7,p8,p9, ncol =2, align = "h", rel_widths = c(1, 1))
```