**Question 1 - MNIST dataset**

a) R code for the implementation of CNN model can be found in Section 2 part 1a). Below represent the number of parameters in each layer and the total number of parameters. Tuning parameters were chosen using grid search and 128 batch size gave the best performance.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_26 (Conv2D)           (None, 26, 26, 32)        320
 max_pooling2d_17 (MaxPooling2D)  (None, 13, 13, 32)    0
 conv2d_25 (Conv2D)           (None, 11, 11, 64)        18496
 max_pooling2d_16 (MaxPooling2D)  (None, 5, 5, 64)      0
 conv2d_24 (Conv2D)           (None, 3, 3, 64)          36928
 flatten_8 (Flatten)          (None, 576)               0
 dense_18 (Dense)             (None, 64)                36928
 dense_17 (Dense)             (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
```

- Training accuracy: 0.9971
- Validation accuracy: 0.9906
- Test accuracy: 0.9917

b) In Mini Project 2, I recommended neural network model with 2 hidden layers (64 and 128 number of units per hidden layer) without any regualaization or dropouts. Test error of the proposed model in miniproject 2 and CNN model is 0.0243 and 0.0083 respectively. Model obtained using CNN is better than the model proposed in Mini Project 2 as it has lower test error. Therefore, I would recommend CNN model for this dataset.

c) R code for the implementation of CNN model after modification can be found in Section 2 part 1c). The addition of a dropout layer with rate 0.5 to the model in part a) led to a higher test accuracy of 0.9946, which surpassed the accuracy of part a).

d) Before recommending the adoption of the model in practice, we could explore incorporating data augmentation, applying regularization techniques such as adding a penalty term to the loss function, and increasing the depth of the network to potentially improve its performance.

**Question 2 - CIFAR100 dataset**

a) R code for the implementation of CNN model can be found in Section 2 part 2a). Below represent the number of parameters in each layer and the total number of parameters. Tuning parameters were chosen using grid search and 128 batch size gave the best performance.

- Training accuracy: 0.6122
- Validation accuracy: 0.3872
- Test accuracy: 0.397

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_179 (Conv2D) | (None, 32, 32, 32) | 896 |
| max_pooling2d_123 (MaxPooling2D) | (None, 16, 16, 32) | 0 |
| conv2d_178 (Conv2D) | (None, 16, 16, 64) | 18496 |
| max_pooling2d_122 (MaxPooling2D) | (None, 8, 8, 64) | 0 |
| conv2d_177 (Conv2D) | (None, 8, 8, 128) | 73856 |
| max_pooling2d_121 (MaxPooling2D) | (None, 4, 4, 128) | 0 |
| conv2d_176 (Conv2D) | (None, 4, 4, 256) | 295168 |
| max_pooling2d_120 (MaxPooling2D) | (None, 2, 2, 256) | 0 |
| flatten_58 (Flatten) | (None, 1024) | 0 |
| dropout_7 (Dropout) | (None, 1024) | 0 |
| dense_120 (Dense) | (None, 512) | 524800 |
| dense_119 (Dense) | (None, 100) | 51300 |

Total params: 964,516
Trainable params: 964,516
Non-trainable params: 0

c) R code for the implementation of CNN model after modification can be found in Section 2 part 2b). The addition of a dropout layer with rate 0.5 to the model in part a) led to a higher test accuracy of 0.463, which surpassed the accuracy of part a).

d) Before recommending the adoption of the model in practice, we could explore incorporating data augmentation, applying regularization techniques such as adding a penalty term to the loss function, and increasing the depth of the network to potentially improve its performance.

**Question 3**

Filters b) and c) can be used as horizontal edge detectors as they would compute the difference between adjacent pixels in the horizontal direction. On the other hand filters a) and d) can be used as vertical edge detectors as they would compute the difference between adjacent pixels in the vertical direction.

- Please note that this question was completed using python. Python notebook can be found at https://colab.research.google.com/drive/1K-_eCp1PAvTfPmod5VgSrlqRNlF0mQlJ?usp=sharing

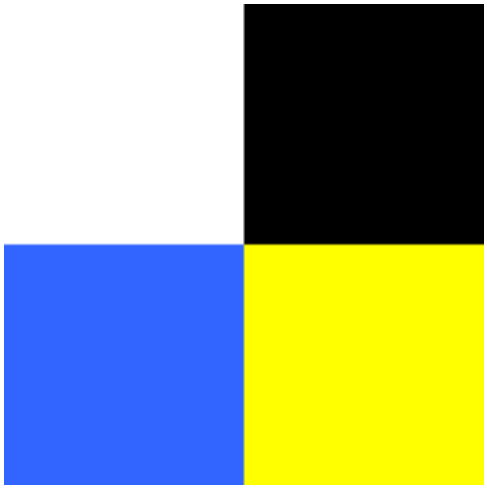

Figure 1: *original image*

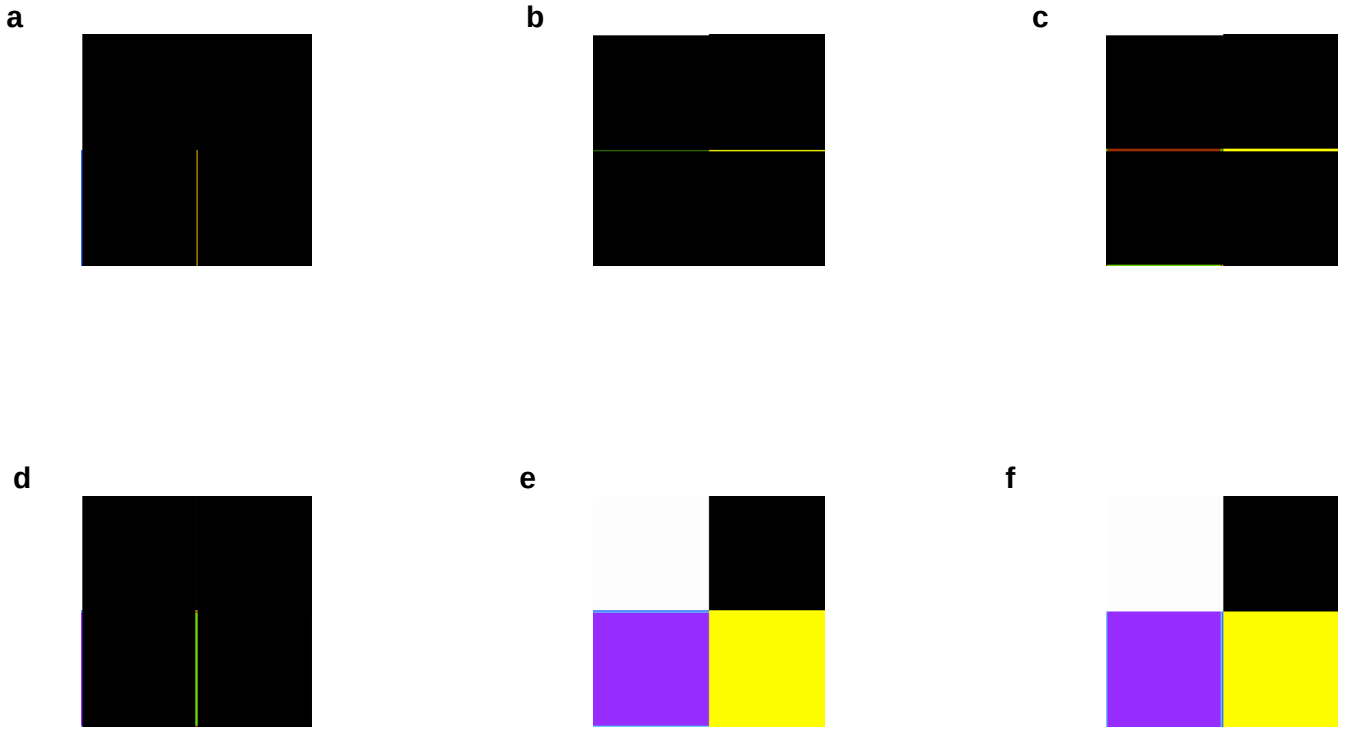


Figure 2: *Filtered outputs*

**Question 4**

Bellow table represent predicted classes of Bella and there associated scores. Airedale is the most suitable class label for Bella.

| class name | class description | score |
|---|---|---|
| n02096051 | Airedale | 0.33139139 |
| n04399382 | teddy | 0.29556641 |
| n02113624 | toy poodle | 0.08234011 |

Table 1: Class prediction for Bella

**Question 5**

Figure 3: *Fifth channel of the activation of the first layer on the Bella*

- Layer 1: Collection of various edge detectors — activations retain much of the information from the input image.
- Later layers: Activations become ↑ more abstract. Some encode higher-level concepts such as “cat ear” and “cat eye;” others are not interpretable.
- Activations become more sparse as depth ↑ — more and more filters are blank. This means the pattern encoded by the filter isn’t found in the input image

This is consistent with *Universal characteristic of learned representations* learned in class

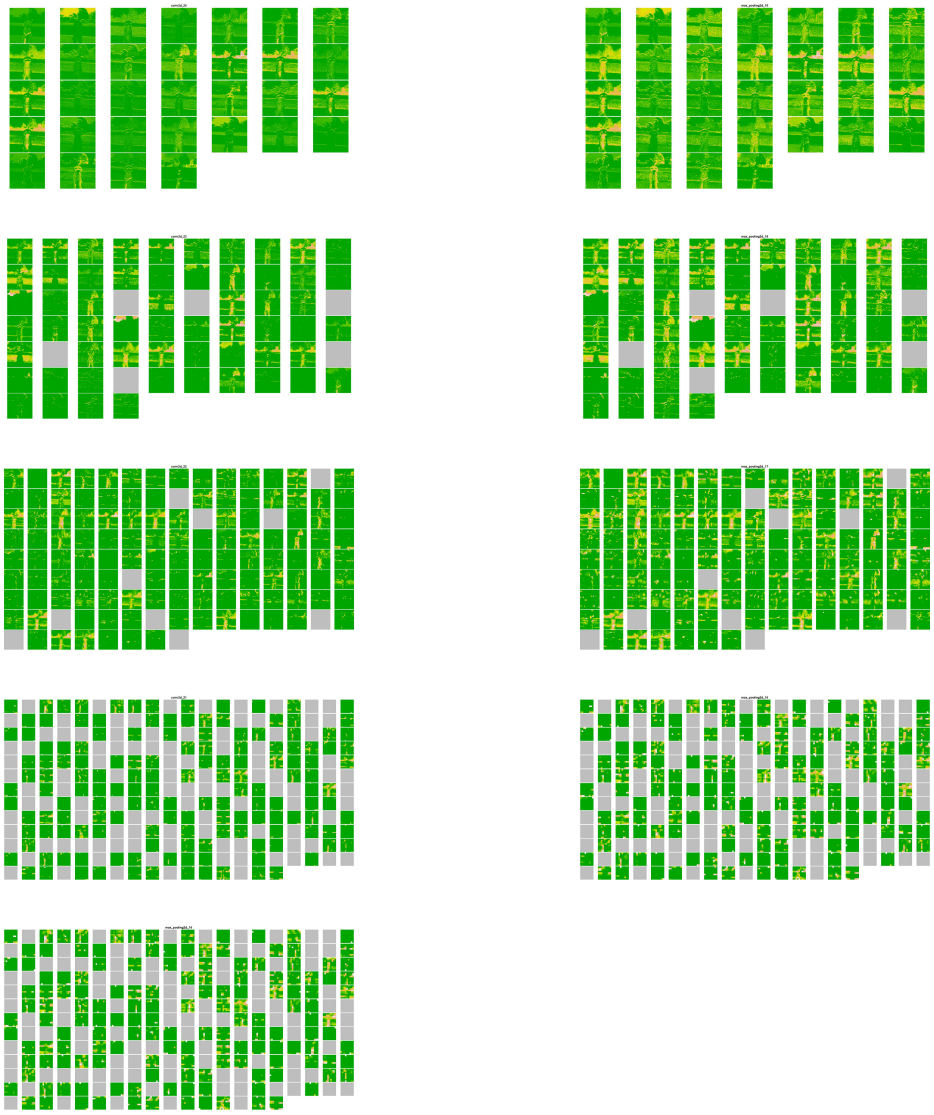

Figure 4: *Every channel of every layer activation on the Bella picture*

## Section 2: R and Python code

```
## ----eval=FALSE, include=FALSE-------------
## ##########################################################################
## #                              Question 1
## ##########################################################################
##
## ######################## Data Pre processing ############################
##
## # Load the MNIST dataset
## mnist <- dataset_mnist()
## x_train <- mnist$train$x
## y_train <- mnist$train$y
## x_test <- mnist$test$x
## y_test <- mnist$test$y
##
## # Preprocess the data
## x_train <- array_reshape(x_train, c(nrow(x_train), 28, 28, 1))
## x_test <- array_reshape(x_test, c(nrow(x_test), 28, 28, 1))
##
## x_train <- x_train / 255
## x_test <- x_test / 255
##
## y_train <- to_categorical(y_train, 10)
## y_test <- to_categorical(y_test, 10)


## ----echo=FALSE, out.width="400px"---------
knitr::include_graphics("/Users/nissi_wicky/Desktop/q1a.png")


## ----eval=FALSE, include=FALSE-------------
##
## ######################## Part 1a) ###########################
##
## build_model_1a <- function(){
##   # Define the model architecture
##   model <- keras_model_sequential()
##   model %>%
##     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")
##
##   # Compile the model
##   model %>% compile(
##     loss = "categorical_crossentropy",
##     optimizer = "rmsprop",
##     metrics = "accuracy"
##   )
## }
##
## #Parameter tuning
## b_size <- c(32, 64, 128)
## train_acc_1a <- val_acc_1a <- c()
##
```

```
## for(i in 1:length(b_size))
## {
##   model_1a <- build_model_1a()
##
##   # Set seed value
##   set.seed(6390)
##
##   # Fit the model with validation set for tuning parameters
##   history_1a <- model_1a %>% fit(
##     x_train, y_train,
##     batch_size = b_size[i],
##     epochs = 10,
##     validation_split = 0.2
##   )
##
##   train_acc_1a[i] <- history_1a$metrics$accuracy[10]
##   val_acc_1a[i] <-  history_1a$metrics$val_accuracy[10]
## }


## ----eval=FALSE, include=FALSE-------------
## # Fit the model with best tuning parameter and full training data
## history_1a <- model_1a %>% fit(
##   x_train, y_train,
##   batch_size = 128,
##   epochs = 10,
## )
##
## # Evaluate the model on the test set
## score_1a <- model_1a %>% evaluate(x_test, y_test, verbose = 0)
## cat("Test loss:", score_1a[1], "\n")
## cat("Test accuracy:", score_1a[2], "\n")


## ----eval=FALSE, include=FALSE-------------
##
## ###################### Part 1c) ##########################
##
## # Define the model architecture
## model_1c <- keras_model_sequential()
## model_1c %>%
##   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_dropout(0.5) %>%
##   layer_dense(units = 64, activation = "relu", kernel_regularizer = regularizer_l2(0.001)) %>%
##   layer_dense(units = 10, activation = "softmax")
##
##   # Compile the model
## model_1c %>% compile(
##   loss = "categorical_crossentropy",
##   optimizer = "rmsprop",
##   metrics = "accuracy"
## )
##
## # Fit the model with best tuning parameter and full training data
```

```
## history_1c <- model_1c %>% fit(
##   x_train, y_train,
##   batch_size = 128,
##   epochs = 10,
## )
##
## train_acc_1c <- history_2a$metrics$accuracy
## cat("Training accuracy:", train_acc_1c, "\n")
##
## # Evaluate the model on the test set
## score_1c <- model_1c %>% evaluate(x_test, y_test, verbose = 0)
## cat("Test loss:", score_1c[1], "\n")
## cat("Test accuracy:", score_1c[2], "\n")


## ----eval=FALSE, include=FALSE-------------
## ##########################################################################
## #                              Question 2
## ##########################################################################
##
## ######################## Data Pre processing ############################
##
## 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
##
## x_train <- x_train / 255
## x_test <- x_test / 255
##
## y_train <- to_categorical(g_train, 100)
## y_test <- to_categorical(g_test, 100)


## ----echo=FALSE, out.width="400px"---------
knitr::include_graphics("/Users/nissi_wicky/Desktop/q2a.png")


## ----eval=FALSE, include=FALSE-------------
##
## ######################## Part 2a) ############################
##
## build_model_2a <- function()
## {
##   # 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")
##
##   # compile the model
##   model_2a %>% compile(
##     loss = "categorical_crossentropy",
##     optimizer = optimizer_rmsprop(),
##     metrics = c("accuracy")
##     )
## }
##
## #Parameter tuning
## b_size <- c(32, 64, 128)
## train_acc_2a <- val_acc_2a <- c()
##
## for(i in 1:length(b_size))
## {
##   model_2a <- build_model_2a()
##
##   set.seed(6390)
##
##   # fit the model (takes about 10 min to run)
##   history_2a <- model_2a %>% fit(
##     x_train, y_train,
##     epochs = 30,
##     batch_size = b_size[i],
##     validation_split = 0.2
##   )
##
##   train_acc_2a[i] <- history_2a$metrics$accuracy[10]
##   val_acc_2a[i] <-  history_2a$metrics$val_accuracy[10]
## }


## ----eval=FALSE, include=FALSE-------------
## # Fit the model with best tuning parameter and full training data
## history_2a <- model_2a %>% fit(
##   x_train, y_train,
##   batch_size = 128,
##   epochs = 30,
## )
##
## # Evaluate the model on the test set
## score_2a <- model_2a %>% evaluate(x_test, y_test, verbose = 0)
## cat("Test loss:", score_2a[1], "\n")
## cat("Test accuracy:", score_2a[2], "\n")


## ----eval=FALSE, include=FALSE-------------
##
## ######################## Part 2b) ############################
##
## model_2c <- 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")
##
##   # compile the model
## model_2c %>% compile(
##   loss = "categorical_crossentropy",
##   optimizer = optimizer_rmsprop(),
##   metrics = c("accuracy")
##   )
##
## # Fit the model with best tuning parameter and full training data
## history_2c <- model_2c %>% fit(
##   x_train, y_train,
##   batch_size = 128,
##   epochs = 30,
## )
##
## train_acc_2c <- history_2c$metrics$accuracy
## cat("Training accuracy:", train_acc_2c, "\n")
##
## # Evaluate the model on the test set
## score_2c <- model_2c %>% evaluate(x_test, y_test, verbose = 0)
## cat("Test loss:", score_2c[1], "\n")
## cat("Test accuracy:", score_2c[2], "\n")


################################################################################
#                              Question 3 : Python code
################################################################################

####################### Generate 180 by 180 image ##########################
from PIL import Image
import requests
from io import BytesIO

# Set the size of the image
size = (180, 180)

# Create a new image with a white background
img = Image.new('RGB', size, color='white')

# Set the colors of the four sections
colors = [(255, 255, 255), (0, 0, 0), (50, 100, 255), (255, 255, 0)]

# Loop through the four sections and set their colors
for i in range(2):
```

```
    for j in range(2):
        color = colors[i*2 + j]
        x0 = j * size[0] // 2
        y0 = i * size[1] // 2
        x1 = (j + 1) * size[0] // 2
        y1 = (i + 1) * size[1] // 2
        box = (x0, y0, x1, y1)
        img.paste(color, box)

# Display the image in Colab
img.show()

# Save the image to a file
img.save('my_image.png')

###################### part 3a) ########################

image = np.array(Image.open('my_image.png'))

# Define filter
filter = np.array([[1, -1]])

# Split image into channels
r = image[:,:,0]
g = image[:,:,1]
b = image[:,:,2]

# Perform convolution on each channel
r_conv = convolve2d(r, filter, mode='same')
g_conv = convolve2d(g, filter, mode='same')
b_conv = convolve2d(b, filter, mode='same')

# Combine channels into an RGB image
convolved_image = np.dstack((r_conv, g_conv, b_conv))

# Save convolved image
Image.fromarray(convolved_image.astype(np.uint8)).save('convolved_image_a.png')


###################### part 3b) ########################

image = np.array(Image.open('my_image.png'))

# Define filter
filter = np.array([[1],[-1]])

# Split image into channels
r = image[:,:,0]
g = image[:,:,1]
b = image[:,:,2]

# Perform convolution on each channel
r_conv = convolve2d(r, filter, mode='same')
g_conv = convolve2d(g, filter, mode='same')
b_conv = convolve2d(b, filter, mode='same')

# Combine channels into an RGB image
convolved_image = np.dstack((r_conv, g_conv, b_conv))
```

```
# Save convolved image
Image.fromarray(convolved_image.astype(np.uint8)).save('convolved_image_b.png')


###################### part 3c) ########################
image = np.array(Image.open('my_image.png'))

# Define filter
filter = np.array([[1, 1, 1],
                   [0, 0, 0],
                   [-1, -1, -1]])

# Split image into channels
r = image[:,:,0]
g = image[:,:,1]
b = image[:,:,2]

# Perform convolution on each channel
r_conv = convolve2d(r, filter, mode='same')
g_conv = convolve2d(g, filter, mode='same')
b_conv = convolve2d(b, filter, mode='same')

# Combine channels into an RGB image
convolved_image = np.dstack((r_conv, g_conv, b_conv))

# Save convolved image
Image.fromarray(convolved_image.astype(np.uint8)).save('convolved_image_c.png')


###################### part 3d) ########################

image = np.array(Image.open('my_image.png'))

# Define filter
filter = np.array([[1, 0, -1],
                   [1, 0, -1],
                   [1, 0, -1]])

# Split image into channels
r = image[:,:,0]
g = image[:,:,1]
b = image[:,:,2]

# Perform convolution on each channel
r_conv = convolve2d(r, filter, mode='same')
g_conv = convolve2d(g, filter, mode='same')
b_conv = convolve2d(b, filter, mode='same')

# Combine channels into an RGB image
convolved_image = np.dstack((r_conv, g_conv, b_conv))

# Save convolved image
Image.fromarray(convolved_image.astype(np.uint8)).save('convolved_image_d.png')

###################### part 3e) ########################

image = np.array(Image.open('my_image.png'))

# Define filter
```

```
filter = np.array([[0, 1, 0],
                   [0, 1, 0],
                   [0, 1, 0]])

# Split image into channels
r = image[:,:,0]
g = image[:,:,1]
b = image[:,:,2]

# Perform convolution on each channel
r_conv = convolve2d(r, filter, mode='same')
g_conv = convolve2d(g, filter, mode='same')
b_conv = convolve2d(b, filter, mode='same')

# Combine channels into an RGB image
convolved_image = np.dstack((r_conv, g_conv, b_conv))

# Save convolved image
Image.fromarray(convolved_image.astype(np.uint8)).save('convolved_image_e.png')


###################### part 3f) ##########################

image = np.array(Image.open('my_image.png'))

# Define filter
filter = np.array([[0, 0, 0],
                   [1, 1, 1],
                   [0, 0, 0]])

# Split image into channels
r = image[:,:,0]
g = image[:,:,1]
b = image[:,:,2]

# Perform convolution on each channel
r_conv = convolve2d(r, filter, mode='same')
g_conv = convolve2d(g, filter, mode='same')
b_conv = convolve2d(b, filter, mode='same')

# Combine channels into an RGB image
convolved_image = np.dstack((r_conv, g_conv, b_conv))

# Save convolved image
Image.fromarray(convolved_image.astype(np.uint8)).save('convolved_image_f.png')

## ----echo=FALSE, fig.cap="\\textit{original image}", out.width="400px"----
knitr::include_graphics("/Users/nissi_wicky/Desktop/my_image.png")


## ----echo=FALSE, fig.cap="\\textit{Filtered outputs}",out.width="500px"----
library(cowplot)
library(ggplot2)
library(magick)

p1 <- ggdraw() + draw_image("/Users/nissi_wicky/Desktop/convolved_image_a.png", scale = 0.5)
p2 <- ggdraw() + draw_image("/Users/nissi_wicky/Desktop/convolved_image_b.png", scale = 0.5)
p3 <- ggdraw() + draw_image("/Users/nissi_wicky/Desktop/convolved_image_c.png", scale = 0.5)
p4 <- ggdraw() + draw_image("/Users/nissi_wicky/Desktop/convolved_image_d.png", scale = 0.5)
```

```r
p5 <- ggdraw() + draw_image("/Users/nissi_wicky/Desktop/convolved_image_e.png", scale = 0.5)
p6 <- ggdraw() + draw_image("/Users/nissi_wicky/Desktop/convolved_image_f.png", scale = 0.5)

plot_grid(p1, p2, p3, p4,p5,p6, ncol =3 , align = "hv", axis = "tb",
          labels = c("a", "b", "c", "d","e","f"),
          hjust = -4, vjust = 5)


## ----eval=FALSE, include=FALSE-------------
## ##########################################################################
## #                              Question 4
## ##########################################################################
##
## img_path <- "/Users/nissi_wicky/Desktop/Bella.jpeg"
## img <- image_load(img_path, target_size = c(224, 224))
##
## # Preprocess the image
## x <- image_to_array(img)
## x <- array_reshape(x, c(1, dim(x)))
## x <- imagenet_preprocess_input(x)
##
## model <- application_resnet50(weights = "imagenet")
## summary(model)
##
## # predict the class of the test images
##
## pred6 <- model %>% predict(x) %>%
##   imagenet_decode_predictions(top = 3)
##
## pred6


##########################################################################
#                             Question 5
##########################################################################

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/nissi_wicky/convnet_from_scratch.keras")


img_path <- get_file(
  fname="Bella_1.jpg",
  origin="/Users/nissi_wicky/Bella_1.jpg")

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 (
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{Fifth channel of the activation of the first layer on the Bella}", out.widt
knitr::include_graphics("/Users/nissi_wicky/Desktop/Bella_913.jpeg")


## ----echo=FALSE, fig.cap="\\textit{Every channel of every layer activation on the Bella picture}",fig.width=6-
```

```r
p1 <- ggdraw() + draw_image("/Users/nissi_wicky/Desktop/c1.jpeg", scale = 0.8)
p2 <- ggdraw() + draw_image("/Users/nissi_wicky/Desktop/p1.jpeg", scale = 0.8)
p3 <- ggdraw() + draw_image("/Users/nissi_wicky/Desktop/c2.jpeg", scale = 0.8)
p4 <- ggdraw() + draw_image("/Users/nissi_wicky/Desktop/p2.jpeg", scale = 0.8)
p5 <- ggdraw() + draw_image("/Users/nissi_wicky/Desktop/c3.jpeg", scale = 0.8)
p6 <- ggdraw() + draw_image("/Users/nissi_wicky/Desktop/p3.jpeg", scale = 0.8)
p7 <- ggdraw() + draw_image("/Users/nissi_wicky/Desktop/c4.jpeg", scale = 0.8)
p8 <- ggdraw() + draw_image("/Users/nissi_wicky/Desktop/p4.jpeg", scale = 0.8)

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