MiniProject 5: Report

STAT 6390

Raheel Ahmed

Rsa170130

Section 1

Question 1

1. **Fit a CNN model with the following architecture (also mentioned on slide #23 of updated Chapter 8 notes). Specify the number of parameters in each layer and the total number of parameters. How did you choose the tuning parameters? Report the training, validation and test accuracy of the model.**  
   Number of parameters is: K × (# input channels × (l × l) + 1) --   
   Conv Layer 1: 320  
   Conv Layer 2: 18496  
   Conv Layer 3: 36928  
   Dense Layer: 36928  
   Output Layer: 650  
   Total: 93322  
   ------------------------------  
   Training Accuracy: 0.9992  
   Validation Accuracy: 0.9921  
   Test Accuracy: 0.9934
2. **Compare model in (a) with the model you proposed in Mini Project 4. Which of the two models would you recommend now?**  
   The model I would recommend is the CNN model; it had a lower test and validation error of ~ 0.006 whereas the other model had errors ~ 0.02.
3. **Come up with a CNN model that has higher test accuracy than the model in (a). What modifications led to the improvement?**  
   The modifications were primarily regularization based where dropout was added after a max pool layer and after the flattening of a layer. I figured, based on other examples, perhaps the addition of dropout regularization could help avoid overfitting. My end results were training accuracy of 0.9935, validation accuracy of 0.9938, and test accuracy of 0.9947.
4. **What further modifications of the model you may try before recommending its adoption in practice?**  
   I may try other forms of regularization (l1 or l2), and I may try different numbers of filters in later layers or perhaps even add or remove layers.

Question 2

1. **Fit a CNN model with the following architecture (also mentioned on slide #25 of updated Chapter 8 notes). Specify the number of parameters in each layer and the total number of parameters. How did you choose the tuning parameters? Report the training, validation and test accuracy of the model.**  
   Number of parameters:  
   Conv Layer 1: 896

Conv Layer 2: 18496

Conv Layer 3: 73856  
Conv Layer 4: 295168

Dense Layer: 524800

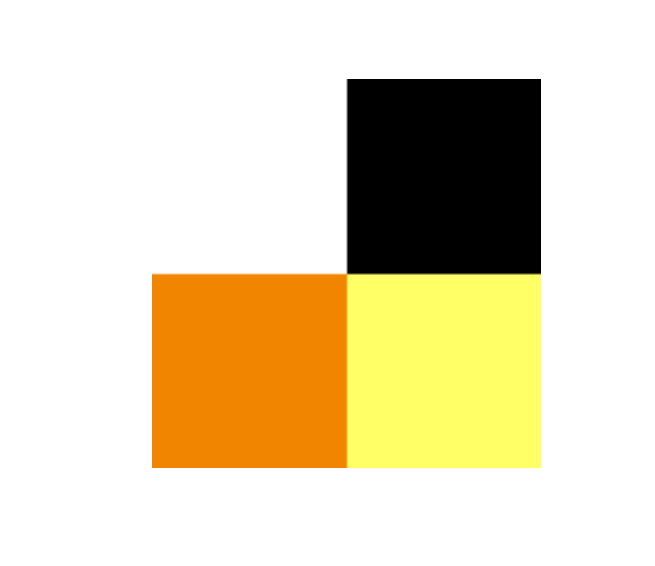
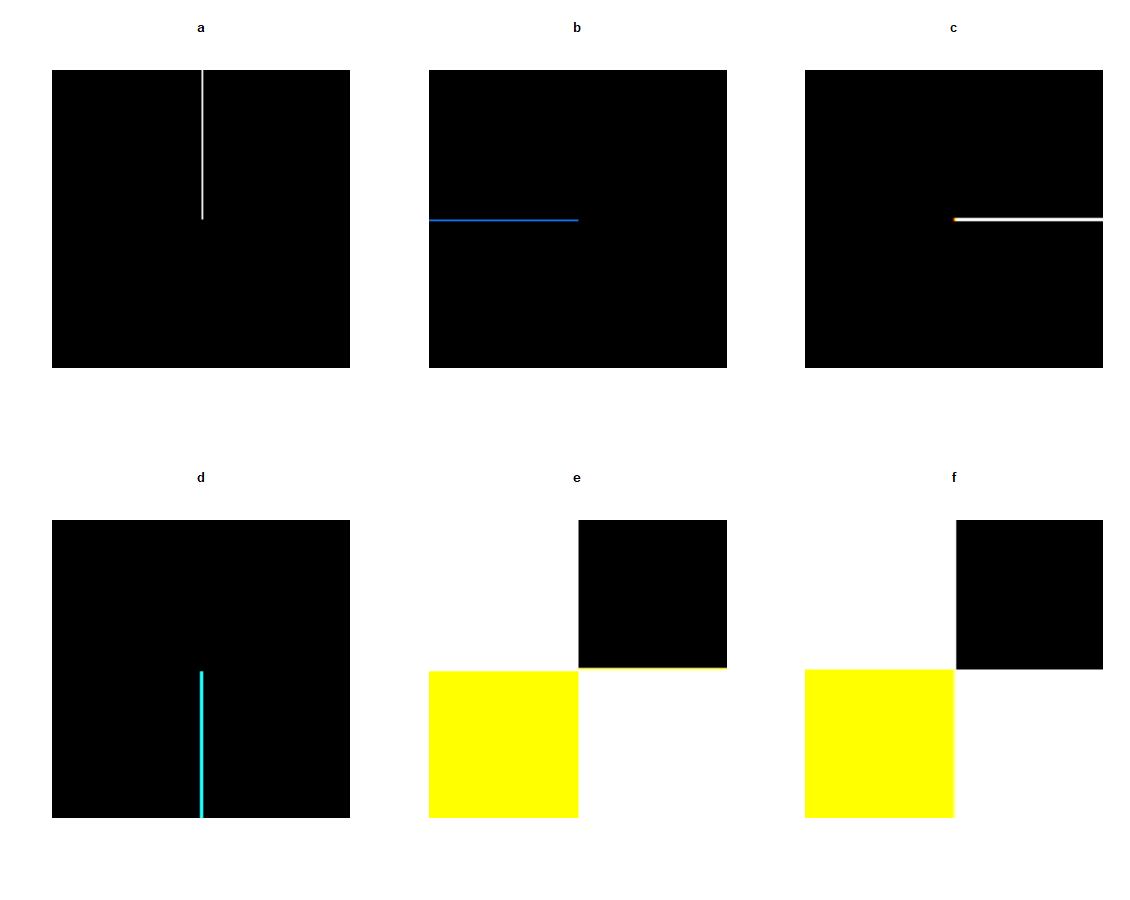
Output Layer: 51300

Total: 964,516  
------------------------------  
Training Accuracy: 0.9603

Validation Accuracy: 0.3583

Test Accuracy: 0.3705

1. **Come up with a CNN model that has higher test accuracy than the model in (a). What modifications led to the improvement?**  
   I took a similar approach as I did for the previous problem. I used dropout before and after the flattening step done in the neural network, and although the regularization did hurt my training accuracy, it did improve the validation accuracy and test accuracy. I believe that this is as a result of regularization preventing overfitting. My end results were training accuracy of 0.641, validation accuracy of 0.4522, and test accuracy of 0.4575.
2. **What further modifications of the model you may try before recommending its adoption in practice?**  
   Again, I would try to add other forms of regularization other than dropout, consider changing the number of layers or the number of filters. Other filter sizes may also help, as larger filters may capture more information from the original image to help make a more specific prediction.

Question 3  
Original Image:  
  
  
  
  
  
  
  
  
  
  
  
  
  
Filtered Images  


1. This [-1, 1] filter essentially detects the harsh vertical edge between the colors on the left and the colors on the right.
2. This [-1, 1]T filter essentially detects the harsh horizontal edge between the colors on the top and the colors on the bottom. The colors are different than in the previous case because the addition of RGB colors yields a different final value.
3. This filter is very similar to b, it also serves as a horizontal edge detector but considers more pixel values and yields a softer edge in the final image.
4. This filter is very similar to a, it also serves as a vertical edge detector but considers more pixel values again and yields a soft edge in the final image. Again, the coloration is different due to regarding different colors in your computation.
5. This filter should mix colors near the boundary of two different colors in the vertical direction. In my case, it did not appear to be what I expected, but perhaps the individual color channels may have added up to be something different to what I expected.
6. This filter should mix colors near the boundary of two different colors in the horizontal direction. Again, it did not appear to be what I expected, and yielded the same thing as the filter in part e, but this may have been due to the convolve function I used or mathematics behind the color channel additions that I didn’t visualize properly.
7. **Compare the output images and comment on the performance of the various filters. Which ones would you recommend for horizontal and vertical edge detection? Explain.**  
   I believe the edge detection filters worked quite well, any discrepancy in the result may have been a consequence of color channel additions in the convolution process. The middle column / row 1 matrices did not end up how I expected it to be, however. In the end, I would choose the filters from c and d to do horizontal and vertical edge detection, respectively. The filters will cancel out when pixels in the top and bottom row or column respectively are the same and will yield a new color when they are different, which is exactly what we expect when we attempt edge detection. Additionally, the larger size of the kernel will account for more information from the original image and be less strict on the edge detection, even for other images.

Question 4

**Use the ResNet50 pretrained model with ImageNet weights (see the R handout) to predict the class of Bella whose image is available as Bella.jpg on eLearning. You may crop the picture as appropriate.**

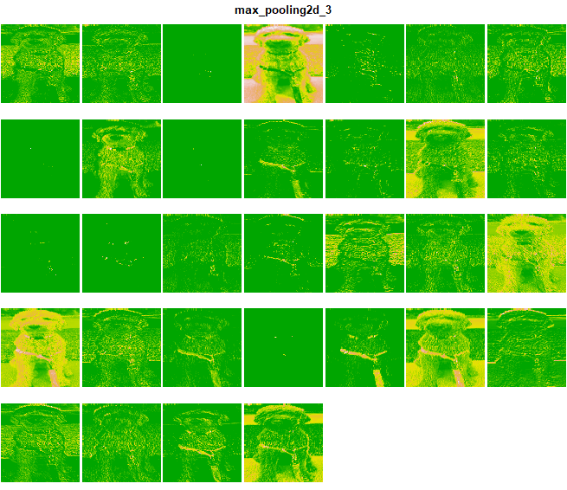
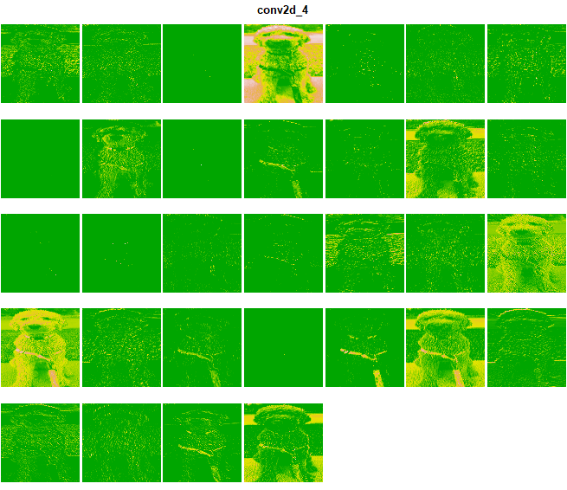
**Text

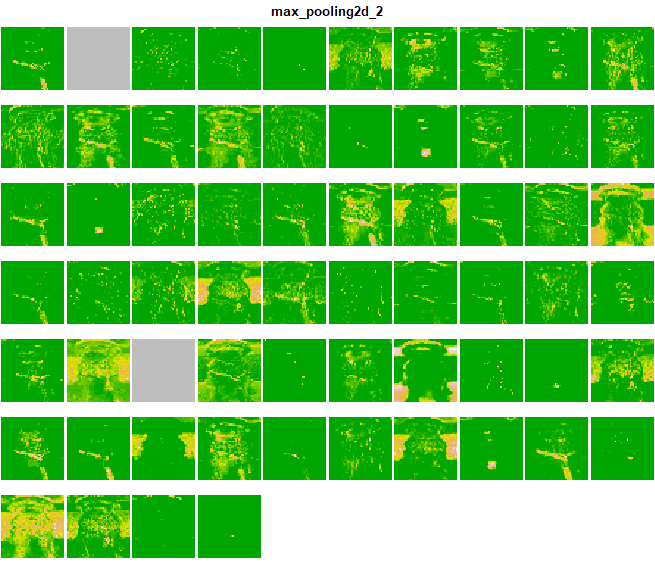
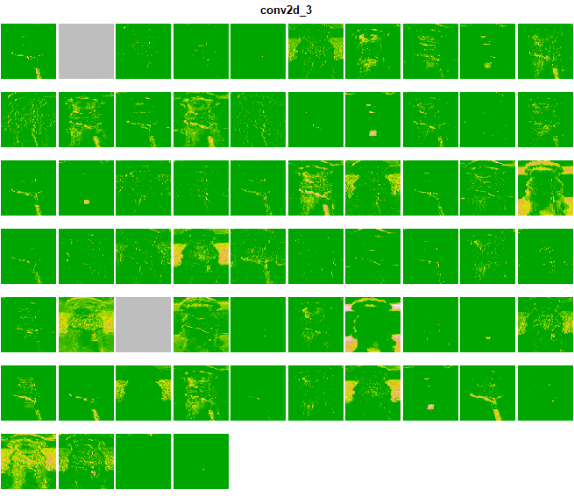
Description automatically generated**

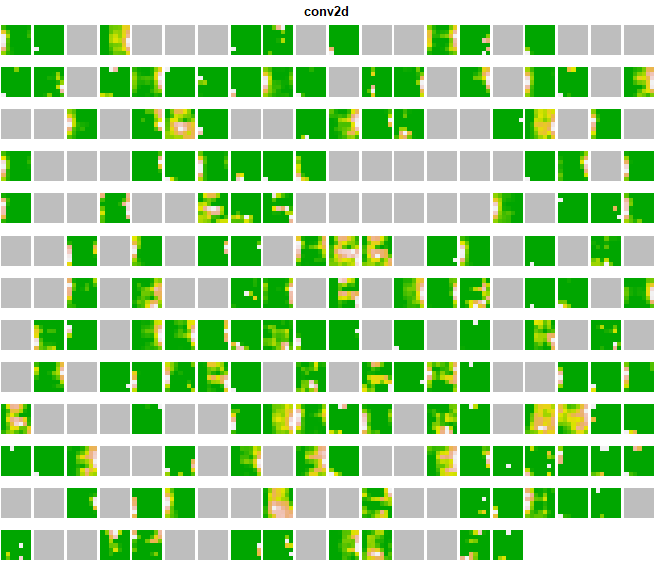
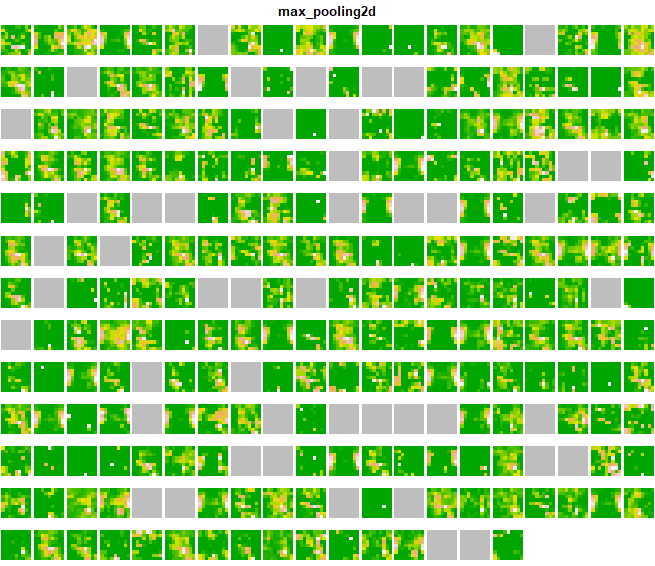
The predictions were as shown above, with the majority indicating that the image was of a poodle of some kind or a terrier, which works great because Bella is a dog! However, the second prediction, Teddy, was one I was unsure about. Teddy could refer to a name for a type of dog breeds of smaller, fuzzy dogs or it could be a mistake because Bella may look like a Teddy Bear toy in her small, compressed image.

Question 5

**Consider the cats vs dogs data and the CNN model fit to the data as described in Section 9.4 of the DLR2 book (see the PDF handout). Follow the process described in the handout to generate visualizations as in Figures 9.13 and 9.14 with Bella’s image from the previous exercise. Comment on what you see. Are the results consistent what you expect from the universal characteristic of learned representations as discussed in the class?**



  
I skipped some from here, but the last two images are as follows:

  
As you can see, the results are consistent with the universal characteristic of learned representations. The initial convolution and max pooling layers have image representations that look a lot like the input image; however, as this continues, you can see that the image representations become far more abstract, relating more to the output than the input. Additionally, in the later layers, more and more filters are blank, not encoding patterns found in the input image.

Section 2

Question 1  
library**(**keras**)**

# Load in Mnist data

mnist **<-** dataset\_mnist**()**

# Assign variables to work with training and test data from the mnist dataset

train\_images **<-** mnist**$**train**$**x

train\_labels **<-** mnist**$**train**$**y

test\_images **<-** mnist**$**test**$**x

test\_labels **<-** mnist**$**test**$**y

# Reshape data into a matrix form and divide by 255 to scale

train\_images **<-** array\_reshape**(**train\_images, c**(**60000, 28, 28, 1**))** # matrix

train\_images **<-** train\_images**/**255 # ensures all values are in [0, 1]

test\_images **<-** array\_reshape**(**test\_images, c**(**10000, 28, 28, 1**))**

test\_images **<-** test\_images**/**255

# Set response to categorical for multinomial response

cat\_train\_labels **<-** to\_categorical**(**train\_labels, 10**)**

cat\_test\_labels **<-** to\_categorical**(**test\_labels, 10**)**

# Build convolutional model

model **<-** 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\_max\_pooling\_2d**(**pool\_size **=** c**(**2, 2**))** %>%

layer\_flatten**()** %>%

layer\_dense**(**units **=** 64, activation **=** "relu"**)** %>%

layer\_dense**(**units **=** 10, activation **=** "softmax"**)**

summary**(**model**)**

# Compile the model

model %>% compile**(**loss **=** "categorical\_crossentropy",

optimizer **=** optimizer\_rmsprop**()**, metrics **=** c**(**"accuracy"**))**

# Fit the model to train data with categorical labels

history **<-** model %>% fit**(**train\_images, cat\_train\_labels, epochs **=** 30,

batch\_size **=** 128, validation\_split **=** 0.2**)**

# Print training and validation accuracy

history

# Print test accuracy

**(**model %>% evaluate**(**test\_images, cat\_test\_labels, verbose **=** F**))[**"accuracy"**]**

# Produce a better model (with dropout) and print out metrics of evaluation

better\_model **<-** 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\_max\_pooling\_2d**(**pool\_size **=** c**(**2, 2**))** %>%

layer\_dropout**(**rate **=** 0.25**)** %>%

layer\_flatten**()** %>%

layer\_dropout**(**rate **=** 0.5**)** %>%

layer\_dense**(**units **=** 64, activation **=** "relu"**)** %>%

layer\_dense**(**units **=** 10, activation **=** "softmax"**)**

summary**(**better\_model**)**

better\_model %>% compile**(**loss **=** "categorical\_crossentropy",

optimizer **=** optimizer\_rmsprop**()**, metrics **=** c**(**"accuracy"**))**

history2 **<-** better\_model %>% fit**(**train\_images, cat\_train\_labels, epochs **=** 30,

batch\_size **=** 128, validation\_split **=** 0.2**)**

history2

**(**better\_model %>% evaluate**(**test\_images, cat\_test\_labels, verbose **=** F**))[**"accuracy"**]**

Question 2

library**(**keras**)**

# Load in CIFAR data

cifar100 **<-** dataset\_cifar100**()**

# Portion CIFAR data into train and test sets

x\_train **<-** cifar100**$**train**$**x

g\_train **<-** cifar100**$**train**$**y

x\_test **<-** cifar100**$**test**$**x

g\_test **<-** cifar100**$**test**$**y

# Scale the training and test features

x\_train **<-** x\_train **/** 255

x\_test **<-** x\_test **/** 255

# Convert labels to categorical values (100 classes present this time)

y\_train **<-** to\_categorical**(**g\_train, 100**)**

y\_test **<-** to\_categorical**(**g\_test, 100**)**

# Build a convolutional model

model **<-** 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**)**

# Compile the model

model %>% compile**(**loss **=** "categorical\_crossentropy",

optimizer **=** optimizer\_rmsprop**()**, metrics **=** c**(**"accuracy"**))**

# Fit the model

history **<-** model %>% fit**(**x\_train, y\_train, epochs **=** 30,

batch\_size **=** 128, validation\_split **=** 0.2**)**

#Print out training and test accuracy

history

# Print out the test accuracy

**(**model %>% evaluate**(**x\_test, y\_test, verbose **=** F**))[**"accuracy"**]**

# Produce a better model with dropout and print out the metrics of evaluation

better\_model **<-** 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\_dropout**(**rate **=** 0.25**)** %>%

layer\_flatten**()** %>%

layer\_dropout**(**rate **=** 0.5**)** %>%

layer\_dense**(**units **=** 512, activation **=** "relu"**)** %>%

layer\_dense**(**units **=** 100, activation **=** "softmax"**)**

summary**(**better\_model**)**

better\_model %>% compile**(**loss **=** "categorical\_crossentropy",

optimizer **=** optimizer\_rmsprop**()**, metrics **=** c**(**"accuracy"**))**

history2 **<-** better\_model %>% fit**(**x\_train, y\_train, epochs **=** 30,

batch\_size **=** 128, validation\_split **=** 0.2**)**

history2

**(**better\_model %>% evaluate**(**x\_test, y\_test, verbose **=** F**))[**"accuracy"**]**

Question 3

library**(**magick**)**

# Produce red, green, and blue matrix data for white, black, most favorite, and least favorite colors

white\_red **<-** matrix**(**data**=**255**/**255, nrow**=**90, ncol**=**90**)**

white\_green **<-** matrix**(**data**=**255**/**255, nrow**=**90, ncol**=**90**)**

white\_blue **<-** matrix**(**data**=**255**/**255, nrow**=**90, ncol**=**90**)**

white\_arr **<-** array**(**data**=**c**(**white\_red, white\_blue, white\_green**)**, dim **=** c**(**90,90,3**))**

black\_red **<-** matrix**(**data**=**0, nrow**=**90, ncol**=**90**)**

black\_green **<-** matrix**(**data**=**0, nrow**=**90, ncol**=**90**)**

black\_blue **<-** matrix**(**data**=**0, nrow**=**90, ncol**=**90**)**

black\_arr **<-** array**(**data**=**c**(**black\_red, black\_blue, black\_green**)**, dim **=** c**(**90,90,3**))**

col1\_red **<-** matrix**(**data**=**242**/**255, nrow**=**90, ncol**=**90**)**

col1\_green **<-** matrix**(**data**=**133**/**255, nrow**=**90, ncol**=**90**)**

col1\_blue **<-** matrix**(**data**=**0, nrow**=**90, ncol**=**90**)**

col1\_arr **<-** array**(**data**=**c**(**col1\_red, col1\_green, col1\_blue**)**, dim**=** c**(**90,90,3**))**

col2\_red **<-** matrix**(**data**=**1, nrow**=**90, ncol**=**90**)**

col2\_green **<-** matrix**(**data**=**1, nrow**=**90, ncol**=**90**)**

col2\_blue **<-** matrix**(**data**=**0.4, nrow**=**90, ncol**=**90**)**

col2\_arr **<-** array**(**data**=**c**(**col2\_red, col2\_green, col2\_blue**)**, dim**=** c**(**90,90,3**))**

# Compile the color data into the final image

image **<-** array**(**dim **=** c**(**180, 180, 3**))**

image**[**1**:**90, 1**:**90, 1**:**3**]** **<-** white\_arr

image**[**1**:**90, 91**:**180, 1**:**3**]** **<-** black\_arr

image**[**91**:**180, 1**:**90, 1**:**3**]** **<-** col1\_arr

image**[**91**:**180, 91**:**180, 1**:**3**]** **<-** col2\_arr

# Plot the image to confirm that the colors are in the right places

plot**(**magick**::**image\_read**(**image**))**

# Show results of filtering the images with each kernel

par**(**mfrow**=**c**(**2, 3**))**

filt1 **<-** t**(**matrix**(**c**(-**1,1,0,0**)**, nrow**=**2, ncol**=**2**))**

filt1\_image **<-** array**(**dim **=** c**(**180, 180, 3**))**

filt1\_image **<-** image\_convolve**(**magick**::**image\_read**(**image**)**, filt1**)**

plot**(**filt1\_image**)**

title**(**main **=** "a"**)**

filt2 **<-** t**(**filt1**)**

filt2\_image **<-** array**(**dim **=** c**(**180, 180, 3**))**

filt2\_image **<-** image\_convolve**(**magick**::**image\_read**(**image**)**, filt2**)**

plot**(**filt2\_image**)**

title**(**main **=** "b"**)**

filt3 **<-** matrix**(**c**(**1,1,1,0,0,0,**-**1,**-**1,**-**1**)**, byrow**=**T, nrow**=**3, ncol**=**3**)**

filt4 **<-** t**(**filt3**)**

filt3\_image **<-** image\_convolve**(**magick**::**image\_read**(**image**)**, filt3**)**

plot**(**filt3\_image**)**

title**(**main **=** "c"**)**

filt4\_image **<-** image\_convolve**(**magick**::**image\_read**(**image**)**, filt4**)**

plot**(**filt4\_image**)**

title**(**main **=** "d"**)**

filt5 **<-** matrix**(**c**(**0,0,0,1,1,1,0,0,0**)**, nrow**=**3, ncol**=**3**)**

filt6 **<-** t**(**filt5**)**

filt5\_image **<-** image\_convolve**(**magick**::**image\_read**(**image**)**, filt5**)**

plot**(**filt5\_image**)**

title**(**main **=** "e"**)**

filt6\_image **<-** image\_convolve**(**magick**::**image\_read**(**image**)**, filt6**)**

plot**(**filt6\_image**)**

title**(**main **=** "f"**)**

Question 4

library**(**keras**)**

# Load in cropped image of Bella

img **<-** image\_load**(**"crop\_bella.jpg", target\_size **=** c**(**224, 224**))**

# Ensure the dimensions of the input are correct

# (1 for providing only 1 image and 224, 224, 3 as the dimensions required for each image)

x **<-** array**(**dim **=** c**(**1, 224, 224, 3**))**

# Convert the image to an array

x**[**1,,,**]** **<-** **(**image\_to\_array**(**img**))**

# Plot for visualization prior to evaluation

plot**(**as.raster**(**x**[**1,,,**]/**255**))**

# Preprocess the picture with imagenet

x **<-** imagenet\_preprocess\_input**(**x**)**

# Develop resnet50 model

model **<-** application\_resnet50**(**weights **=** "imagenet"**)**

summary**(**model**)**

# Predict the model and display the top 5 predictions for our picture of Bella

pred **<-** model %>% predict**(**x**)** %>%

imagenet\_decode\_predictions**(**top **=** 5**)**

#names(pred) <- image\_names

print**(**pred**)**

Question 5

library**(**zip**)**

library**(**fs**)**

library**(**keras**)**

library**(**tfdatasets**)**

# Preliminary steps to produce folders

# unlink("dogs-vs-cats", recursive = TRUE)

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

# zip::unzip("dogs-vs-cats/train.zip", exdir = "dogs-vs-cats")

# unlink("cats\_vs\_dogs\_small", recursive = TRUE)

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**)**

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**)**

# Read image function to process images into tensor data

tf\_read\_image **<-**

**function(**path, format**=**"image", resize **=** **NULL**, ...**){**

img **<-** path %>%

tf**$**io**$**read\_file**()** %>%

tf**$**io**[[**paste0**(**"decode\_", format**)]](**...**)**

**if** **(!**is.null**(**resize**))**

img **<-** img %>%

tf**$**image**$**resize**(**as.integer**(**resize**))**

img

**}**

# Function to display an image tensor

display\_image\_tensor **<-** **function(**x, ..., max **=** 255,

plot\_margins **=** c**(**0, 0, 0, 0**))** **{**

**if** **(!**is.null**(**plot\_margins**))**

withr**::**local\_par**(**mar **=** plot\_margins**)**

x %>%

as.array**()** %>%

drop**()** %>%

as.raster**(**max **=** max**)** %>%

plot**(**..., interpolate **=** **FALSE)**

**}**

# Find the image path for the cropped picture of Bella

img\_path **<-** normalizePath**((**"crop\_bella.jpg"**))**

# Get an image tensor of the picture

img\_tensor **<-** img\_path %>%

tf\_read\_image**(**resize **=** c**(**180, 180**))**

display\_image\_tensor**(**img\_tensor**)**

# Allows us to check if a layer is for convolution or for pooling

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**)**

# Store all layer outputs that we need (convolutions and pooling layers)

layer\_outputs **<-** list**()**

**for** **(**layer **in** model**$**layers**)**

**if** **(**is\_conv\_layer**(**layer**)** **||** is\_pooling\_layer**(**layer**))**

layer\_outputs**[[**layer**$**name**]]** **<-** layer**$**output

# Produce the activation model and predict the img tensor data across all axes

activation\_model **<-** keras\_model**(**inputs **=** model**$**input,

outputs **=** layer\_outputs**)**

activations **<-** activation\_model %>%

predict**(**img\_tensor**[**tf**$**newaxis, , , **])**

# Get first layer

first\_layer\_activation **<-** activations**[[** names**(**layer\_outputs**)[**1**]** **]]**

# Produce function to plot all activations produced

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 each desired layer's plot activation and observe differences per layer.

par**(**mfrow**=**c**(**5,2**))**

**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)**

**}**