Is your deep convolutional network misclassifying images? You can find out why with a heatmap of class activation overlaid on its misclassified pictures.

A heatmap overlay shows parts of an image most activated in a neural network’s last convolutional layer. In this African elephant picture, the top-most convolutional layer of the VGG16 architecture turns the photo into a 14×14 grid highlighting blocks with strongest African\_elephant activation:



Original image source:  
[elephants.com](https://www.elephants.com/) – [African elephant Flora](https://elephants-media.s3.amazonaws.com/images/399/original/Flora%20ears%20out%20carrying%20limb_0002AA.jpg)

What it’s saying with a yellow-green splotch is “Look! There’s an African elephant here!” The learner returns a score of 46%, quite high for a blink-of-an-eye judgment with 1000 objects to choose from and even locates that object in the picture correctly. Impressive.

imagenet\_decode\_predictions(preds, top = 3)[[1]]

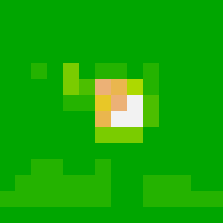
# class\_name class\_description score

#1 n02504458 African\_elephant 0.46432969

#2 n02437312 Arabian\_camel 0.29539737

#3 n01871265 tusker 0.07210348

Shaded parts of this photo have at least some activation to class African\_elephant. These show the elephant’s face and nearby foliage are what distinguish it from an Indian elephant and other classes, like a strawberry or an aircraft carrier. Parts of the photo that have 0 activation on the corresponding heatmap show up as non-shaded, which can be verified from a visualization of the activation heatmap:



or printing it out as a numeric matrix:

round(heatmap, 2)

# [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]

# [1,] 0.00 0.00 0.02 0.02 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

# [2,] 0.00 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

# [3,] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

# [4,] 0.00 0.00 0.05 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

# [5,] 0.00 0.00 0.09 0.07 0.26 0.00 0.11 0.13 0.07 0.16 0.00 0.00 0.00 0.00

# [6,] 0.00 0.00 0.01 0.04 0.30 0.24 0.69 0.63 0.41 0.12 0.04 0.00 0.00 0.00

# [7,] 0.00 0.00 0.01 0.04 0.14 0.14 0.55 0.72 0.92 0.23 0.06 0.00 0.00 0.00

# [8,] 0.00 0.00 0.00 0.01 0.00 0.03 0.61 0.98 1.00 0.22 0.00 0.00 0.00 0.00

# [9,] 0.00 0.00 0.02 0.01 0.00 0.00 0.30 0.27 0.31 0.00 0.00 0.00 0.02 0.00

#[10,] 0.00 0.00 0.04 0.04 0.01 0.00 0.01 0.00 0.02 0.00 0.00 0.02 0.04 0.01

#[11,] 0.00 0.00 0.10 0.09 0.07 0.06 0.10 0.00 0.00 0.00 0.00 0.03 0.05 0.04

#[12,] 0.01 0.14 0.13 0.13 0.11 0.11 0.10 0.00 0.00 0.08 0.09 0.12 0.08 0.08

#[13,] 0.13 0.13 0.15 0.14 0.12 0.10 0.08 0.00 0.03 0.11 0.11 0.15 0.12 0.11

#[14,] 0.04 0.06 0.06 0.04 0.00 0.00 0.02 0.00 0.00 0.00 0.00 0.00 0.00 0.00

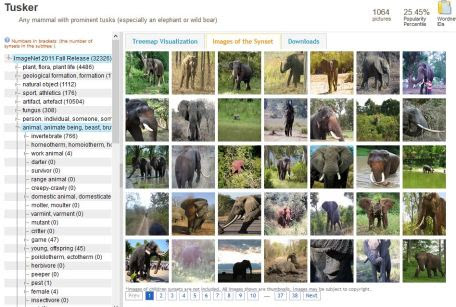
**Detecting sources of errors**

Here is another African elephant with huge ears above its neck, but this time the learner has misclassified it as a tusker with a score of 55% as opposed to 17% for African elephant. Tusker isn’t a terrible judgment. It’s a more generic group that includes wild boars but the classification is not as accurate as African elephant. What threw it off from making a more precise call? Let’s see.



Original image source:  
By [Komar.de](https://www.komar.de/en/elephant.html) – [Non-woven photomural Elephant](https://www.komar.de/en/media/catalog/product/cache/5/image/780x/17f82f742ffe127f42dca9de82fb58b1/x/x/xxl4-529_elephant_ma.jpg)

Looks like the top of the head and the back. Surprising it’s not the tusks. If we take [a sample of ImageNet tusker training images](http://imagenet.stanford.edu/synset?wnid=n01871265), it quickly becomes obvious most tusker images are of elephants. In the first 25 tusker examples shown here, none look like wild boars.



So the cause of our misclassification is understandable, and a training set limitation error. A great first recourse would be to add to ImageNet other kinds of tuskers to better train that class.

imagenet\_decode\_predictions(preds, top = 3)[[1]]

# class\_name class\_description score

#1 n01871265 tusker 0.5496630

#2 n02504013 Indian\_elephant 0.2749955

#3 n02504458 African\_elephant 0.1732897

**R Code**

library(keras)

library(magick)

library(viridis)

model <- application\_vgg16(weights = "imagenet") # keeping top

model # assumes input picture of size 224 x 224

img\_path <- "images/African\_elephant\_1.jpg"

img <- image\_load(img\_path, target\_size = c(224, 224)) %>%

image\_to\_array() %>%

array\_reshape(dim = c(1, 224, 224, 3)) %>% # for batch of this size

imagenet\_preprocess\_input() # channelwise color normalization

preds <- model %>% predict(img)

imagenet\_decode\_predictions(preds, top = 3)[[1]]

# get least likely classes for fun

tail(imagenet\_decode\_predictions(preds, top = 1000)[[1]])

max\_class\_nbr <- which.max(preds[1, ]) # is the class index

# if want to see second most class activations, get at which index

second\_class\_nbr <- which.max((preds[1, ])[-max\_class\_nbr]) # should be second

# add +1 if above the previous index number

second\_class\_nbr <- ifelse(second\_class\_nbr >= max\_class\_nbr,

second\_class\_nbr + 1,

second\_class\_nbr)

# visualize which parts of the image are most class 1 using Grad-CAM

elephant\_output <- model$output[, max\_class\_nbr]

elephant\_output <- model$output[, second\_class\_nbr]

last\_conv\_layer <- model %>% get\_layer("block5\_conv3")

grads <- k\_gradients(elephant\_output, last\_conv\_layer$output)[[1]]

pooled\_grads <- k\_mean(grads, axis = c(1, 2, 3))

iterate <- k\_function(list(model$input),

list(pooled\_grads, last\_conv\_layer$output[1,,,]))

c(pooled\_grads\_value, conv\_layer\_output\_value) %<-% iterate(list(img))

for(i in 1:dim(conv\_layer\_output\_value)[3]){

conv\_layer\_output\_value[,,i] <-

conv\_layer\_output\_value[,,i] \* pooled\_grads\_value[[i]]

}

heatmap <- apply(conv\_layer\_output\_value, c(1, 2), mean)

# normalize heatmap between 0 and 1

heatmap <- pmax(heatmap, 0)

heatmap <- heatmap / max(heatmap)

round(heatmap, 2)

write\_heatmap <- function(heatmap, filename, width = 224, height = 224,

bg = "white", col = terrain.colors(12)){

png(filename, width = width, height = height, bg = bg)

op = par(mar = c(0, 0, 0, 0))

on.exit({par(op); dev.off()}, add = TRUE)

rotate <- function(x) t(apply(x, 2, rev))

image(rotate(heatmap), axes = FALSE, asp = 1, col = col)

}

write\_heatmap(heatmap, paste0(substr(img\_path, 1, nchar(img\_path) - 4), "\_heatmap.png"))

image <- image\_read(img\_path)

info <- image\_info(image)

geometry <- sprintf("%dx%d!", info$width, info$height)

pal <- col2rgb(viridis(20), alpha = TRUE)

alpha <- floor(seq(0, 255, length = ncol(pal)))

pal\_col <- rgb(t(pal), alpha = alpha, maxColorValue = 255)

write\_heatmap(heatmap, "elephant\_overlay.png",

width = dim(heatmap)[1], height = dim(heatmap)[2],

bg = NA, col = pal\_col)

image\_read("elephant\_overlay.png") %>%

image\_resize(geometry, filter = "quadratic") %>%

image\_composite(image, operator = "blend", compose\_args = "20") %>%

plot()

# then save output

image\_read("elephant\_overlay.png") %>%

image\_resize(geometry, filter = "quadratic") %>%

image\_composite(image, operator = "blend", compose\_args = "20") %>%

image\_scale("x480") %>%

image\_convert(format = "jpg") %>%

image\_write(paste0(substr(img\_path, 1, nchar(img\_path) - 4), "\_overlay.jpg"))

# reset the image to second elephant

img\_path <- "images/African\_elephant\_2.jpg"

# then rerun the above from img <-