Image Processing with Convolutional Neural Networks

(keras with 'tensorflow over local GPU' as a back-end in R)

Stepping into CNN, one toe at a time

Aurélien-Morgan

2019-04-29

Harvard Executive Education – Data Science Professional Certification Program

– Capstone Project 2 / 2 - free subject matter -

HarvardX

Table of Content

Table of Content
List of Figures . . .
List of Tables . . .

0 Preambule **1**

1 Introduction **2**

2 Objects classification **3**
2.1 Discovering the cifar-10 dataset . . . 3
2.2 The model training process . . . 4
2.3 Results . . . 4

3 Style transfer **6**
3.1 Model inner workings . . . 6
3.2 Application . . . 7

4 Discussion **10**

Appendices **11**
Appendix A . . . 12
Appendix B . . . 22
Appendix C . . . 24

References **26**

List of Figures

1 Deep Neural Network . . . 2
2 Architecture of a DenseNet model . . . 3
3 example image for the cifar-10 classes . . . 3
4 Training history . . . 4
5 Architecture of a VGG16 model . . . 6
6 Beach style transfer . . . 8
7 Snowy wood style transfer . . . 9
8 50 cases of “a cat for a dog” mis-classification . . . 23

List of Tables

1 Performance metrics of the Densenet classification . . . 5
2 Distribution of the predictions for cat images by our model . . . 5

0 Preambule

There are quite a few challenges to adopting deep learning. First and foremost, it comprises rapidly evolving algorithms. Indeed, deep learning algorithms continue to improve (and very quickly), so keeping up with all the latest advances can require significant time & effort. In addition, training deep neural networks requires tremendous compute power. So projects shall be planned to take advantage of high performance computing platforms that can process large amounts of data quickly.

> A **Graphics processing unit (GPU)** is a highly specilized chipset designed to rapidly manipulate and alter memory. Its architecture makes it very efficient at matrix computation, paralell processing and mathematically-intensive tasks in general.
> A **Central processing unit (CPU)** is designed for more generic multi-purpose usage and is often better equipped for single computations with, for instance, a higher clock speed.
> **GPU-accelerated computing** is the employment of a GPU along with a CPU in order to facilitate processing-intensive operations[1].

The **Keras** high-level neural networks API is the second most popular deep learning framework after **Tensorflow** itself. We will be using *Keras* with *Tensorflow* as a backend[3] and we will run it all on a local GPU[4]. As **R** is the language of choice for the herein report, we employ the 'keras' R package : an R interface to Keras, hosted on the RStudio GitHub (and on CRAN)[5]. The original keras Deep Learning library is written in **Python**, we thus use **Anaconda** as a *Python* distribution and interface the *rstudio/keras* package to it.

The rstudio/keras package comes with a fairly straighforward set of tutorial pages which allow for rookie data scientists to jump start their swimming the Deep Learning lake : [6].

For the herein report, we implement convolutional neural networks in R thru Python 3.6 (Anaconda) using the Keras wrapper with a Tensorflow backend on a CUDA-enabled NVIDIA GPU (GTX-1060), running in Windows 10 operating system[2].

---

[1] there are quite a few existing packages for GPU Computing with R. If interested, you can for instance refer to the two following very valuable sources :[1][2]

[2] Installation instructions for such an environment alongside multipe references to helpful related guidance web pages are included in the R source file that encloses the algorithms described in the herein report.

1 Introduction

To get an understanding of how a **Convolutional Neural Network (CNN)** and its layers work, one needs a basic understanding of *neural networks* first.

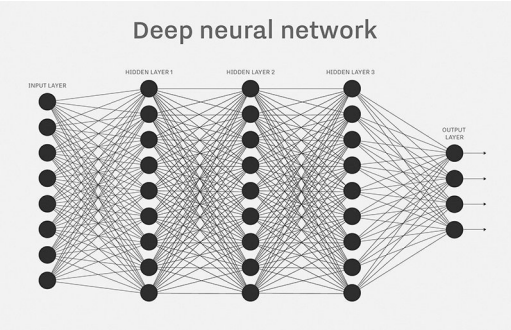

Figure 1: Deep Neural Network

Neural networks are sometimes described in terms of their depth, including how many **layers**[3] they have between input and output (see figure 1 ; source : [8]). This is why the term "neural network" is used almost synonymously with "deep learning".
Layers are each made up of several **nodes**[4]. Data is represented by a tensor (i.e. a higher dimensions matrix[5]). During model training, each node *weights* the importance of input from each of its predecessors. Inputs that contribute to getting right answers are weighted higher. This appropriatedness of answer is measured thanks to a **loss function**.

Adding convolutional layers to a neural network is what turns it into a convolutional neural network. What a convolution layer basically does is detect useful **features**. The layers closer to the input detect lower level features like edges while the ones closer to the output use these features to detect more complex ones like faces[6].

CNNs find applications in **time series forecasting** as it is very good at finding local patterns[12] but Reccurent Neural Networks are often preferred when data has a sequential structure[13][14].
CNNs (when on top of pre-trained word vectors[7]) are often used in **Natural Language Processing (NLP)** for instance for sentence classification[17].
CNNs can also be employed for **speech recognition**, i.e. recognizing and extracting words/sentences/intonations from within audio data[18].
Foremost, Convolutional Neural Networks have become the dominant machine learning approach for **visual object recognition**. In the herein report, we cover two such cases :

- object classification on the cifar-10 images dataset (see the Objects classification section)
- style tranfer on one of my personal pictures (see the Style transfer section)

[3] note that the rstudio/kears package comes with the implementation of many different types of layers (Convolutional, Dropout, Recurrent, etc.) : [7].
[4] Neural networks can also be described by the number of hidden nodes the model has or in terms of how many inputs and outputs each node has.
[5] if you're keen on getting clarifications on tensors : [9], [10].
[6] for more on CNNs, check out this great *edutainment* post : [11].
[7] popular Tensorflow model for *Vector Representations of Words*[15]. Translates words into *neural word embeddings* vectors (of numbers)[16].

2 Objects classification

The code used to create/train/evaluate the algorithm that illustrates this section of the herein report has been inspired by the vignette that comes with the **R/densenet** package (https://github.com/dfalbel/densenet) where is shown how to define a **DenseNet-40-12** model to classify images for the cifar10 dataset. Densenet had originally been introduced by this paper : ***Densely Connected Convolutional Networks***[19].

The architecture of the model we employed here is outlined figure 2 and further detailled Appendix A.

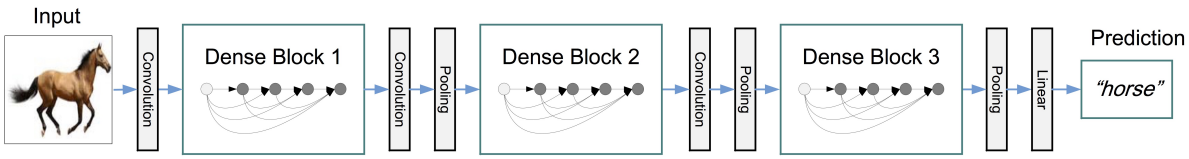

Figure 2: Architecture of a DenseNet model

The model takes ~185s per epoch on a high-end GPU (Nvidia GeForce GTX 1060)[8,9]. It is therefore highly advised to work over a GPU (as opposed to a *-much slower-* CPU alone). We obtained a final accuracy on the test set of 93.7% ±0.5% versus 93% reported on the paper[10].

2.1 Discovering the cifar-10 dataset

The cifar-10 dataset is a labeled subset of the **80 million tiny images** dataset, a *Visual Dictionary* created to *"teach computers to recognize objects"* which had been put together in 2008[20].
The cifar-10 dataset is made up of 60,000 color images in total[21]. 32x32 pixels in size each, they are distributed equally across 10 classes (see figure 3). There are 50,000 training images and 10,000 test images.

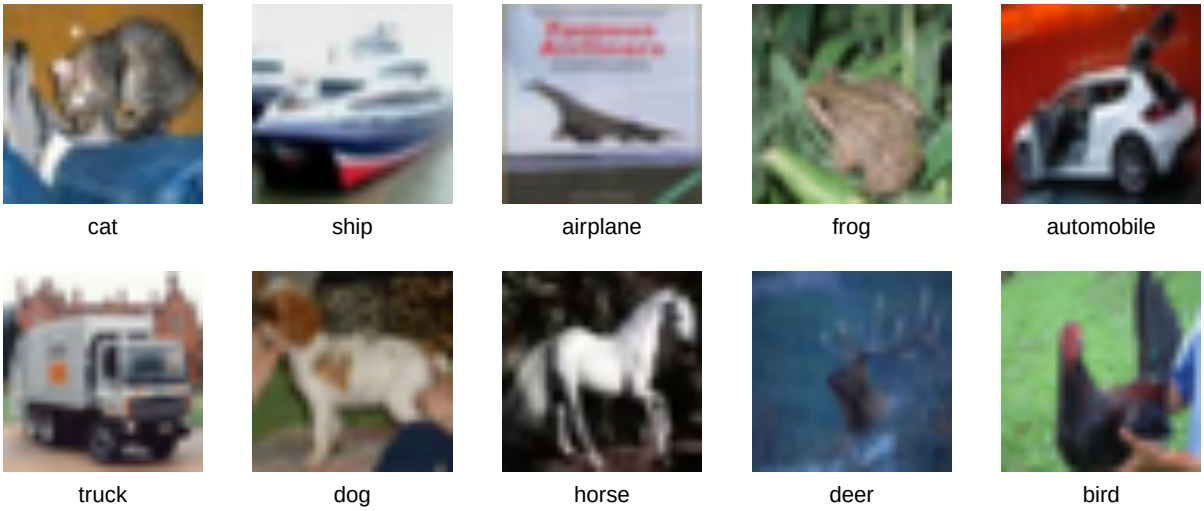

Figure 3: example image for the cifar-10 classes

We realize by looking at the sample images figure 3 that the 32x32 pixels resolution equates to pretty blurry pictures. It does make the performance of our classification model even more impressive, does it not ?

[8] All product and company names are trademarks™ or registered® trademarks of their respective holders.
[9] The model is reported by the author of the "densenet for R" packag to take ~125s per epoch on an Nvidia GeForce 1080 Ti.
[10] an overall accuracy on test set of 0.9351 was reprted by the author of the "densenet for R" package (meaning that 93.51% of the test images have been classified properly).

2.2 The model training process

In order to train a Deep Learning model, data is piped into that model. An **optimizer** is used to minimize the **loss function** by updating the **weights** (and **biases**) following their gradients.

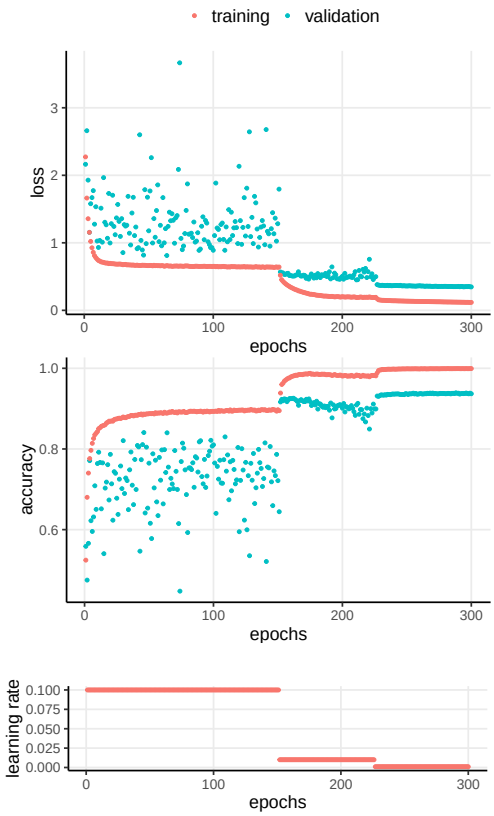

Figure 4: Training history

The *weights* (and *biases*) of the neural network are its internal learnable parameters which are used in computing the output values and are learned and updated in the direction of the optimal solution[22].

An **epoch** is an arbitrary cutoff, generally defined as "one pass over the entire dataset", used to separate *training* into distinct phases. This is useful for logging and periodic evaluation. When training a *keras* model, evaluation is run at the end of every epoch[11].
Within Keras, there is the ability to add **callbacks**[23] specifically designed to be run at the end of an epoch. Examples of these are learning rate changes and model checkpointing (saving).

The **learning rate (lr)** (third quadrant, Figure 4) is an input parameter that establishes how big jumps are to be in the optimization gradient descent between each epoch of the training process.
Along the first 150 epochs, despite managing efficiently to minimize the **loss** (first quadrant) on the training data, it is hardly allowing to converge at all in terms of predicting power ("validation" **accuracy** rather fuzy as can be seen on the second quadrant to say the least). This first phase of training with lr = 0.1 is therefore not sufficient to allow for a useful set of hyperparameters to be found. We must go beyond the first 150 epochs and apply a smaller *learning rate* lr = 0.01. We can see here that we had to perform such a decrease of *learning rate* twice in order to get to a stable *accuracy* on the validation set with a final lr = 0.001.

In our case, we can observe that at the third step of this progressive process, our *overall accuracy* is stable around an average value of 93.6%, slightly increasing until reaching a final plateau.

2.3 Results

As seen earlier, the overall performance of our model can be summarized by its overall accuracy of 93.7% ±0.5%. Like any model though, it does not perform homogeneously accros classes. Table 1 below shows for each of its classes, besides from their respective prevalence[12], a select set of commonly used performance metrics :

- A class **specificity**, also called **True Negative Rate (TNR)**, is the proportion of items of OTHER classes that are appropriately NOT classified as belonging to that class. E.g. 99.2% of items that are NOT airplanes are adequately classfied as NOT being airplanes by our model.
- A class **sensitivity**, also called **recall**, also called **True Positive Rate (TPR)**, is the proportion of items of that class that are appropriately classified as belonging to that class. E.g. 95.0% of airplanes are adequately classfied as being airplanes by our model.

[11] Per default, the data is randomly shuffled at each epoch during training. The same validation set is used for all epochs (within a same call to 'fit'). We provide that 'validation set' to the fitting/training routine. Validation data is never shuffled.
[12] Prevalence is a statistical concept. In our specific case here, it is used to express the proportion of our entire population of images that belong to each individual class. The sum of all prevalences accross all classes always accounts to 1 (100% of the population).

Specificity and *sensitivity*, between the two of them, reflect very well the performance of a model at the 'class' level.

- A class **precision**, also called **Positive Predictive Value (PPV)**, doesn't bring additionnal information compared to the first two performance metrics. It is another way to present it. It in fact corresponds to the proportion of items classified as belonging to that class that are truly elements of that class. E.g. 93.0% of items classfied as being airplanes are indeed airplanes

Alternatively to *Specificity* and *sensitivity*, *Precision* and *recall* are also often refered to together in statistics and machine learning literrature.
Class recall can be thought of as a measure of how often a model is able to find elements of that given class when looking for one.
Class precision can be thought of as a measure of how "believable" a model is when it says an item is of that particular class.

- A class **Balanced accuracy** is somehow redundant with the preceding metrics in the sense that it doesn't bring additionnal information. It is often used to summarize *specificity* and *sensitivity* as it is in fact their *harmonic mean*[13].
- Comparatively, a class **F1 score** is the *harmonic mean* between *precision* and *recall*.

Table 1: Performance metrics of the Densenet classification

class	Prevalence	**Specificity**	**Sensitivity (Recall)**	Precision	**Balanced Accuracy**	F1
airplane	0.1	**0.992**	**0.950**	0.930	**0.971**	0.940
automobile	0.1	**0.995**	**0.973**	0.959	**0.984**	0.966
bird	0.1	**0.992**	**0.916**	0.925	**0.954**	0.921
cat	0.1	**0.989**	**0.854**	0.892	**0.921**	0.873
deer	0.1	**0.993**	**0.945**	0.934	**0.969**	0.939
dog	0.1	**0.988**	**0.901**	0.893	**0.944**	0.897
frog	0.1	**0.996**	**0.948**	0.961	**0.972**	0.955
horse	0.1	**0.995**	**0.971**	0.953	**0.983**	0.962
ship	0.1	**0.996**	**0.957**	0.963	**0.976**	0.960
truck	0.1	**0.995**	**0.954**	0.957	**0.975**	0.955

As we can see table 1, the performance of our object classification model is excellent but, for the sake of argument, let us remark that *sensitivity* is on average lower than *specificity* with our trained model. Since it is the class with lesser *sensitivity* with a value of 85.4%, lets look into images of class "cat" and see how our model has classified the cats images of the validation set :

Table 2: Distribution of the predictions for cat images by our model

airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
9	1	19	854	14	69	12	11	5	6

Table 2 shows how the 1,000 cat images of the validation dataset have been classified by our model. We see there that, when misclassified, they are mostly identified as dog images (69 times).

This might be explainable by the fact that there are many different races of dogs, all having very different characteristics (length of legs, shape of face, etc.) and many different races of cats (with varying ear shapes and various hair thickness). It might be why the model sometimes misclassifies some cats for dogs.

Appendix B shows 50 of such "a cat for a dog" misclassified images, for information.

[13] the average of rates.

3 Style transfer

The will to investigate this part of the Machine Learning applications to Image Processing for the herein report has originally been triggered by coming across this post by Chintan TRIVEDI on **towardsdatascience.com** : "Turning Fortnite into PUBG with Deep Learning (CycleGAN)"[24].
The author there uses hypophora, a figure of speech in which a writer raises a question and then immediately provides an answer to that question. He went on with the following :

> Can we have graphics mods for games that can allow us to choose the visual effects of our liking without having to rely on the game developers providing us that option?

The answer he provided to that was to pick two insanely popular Battle Royale games out, Fortnite and PUBG. He made a Neural Network's attempt at recreating Fortnite in the visual style of PUBG. And he did great !
We will however take a much less ambitious go to *style transfer*. But it's not gonna be less fun of a read !

3.1 Model inner workings

The starting codebase used for this Style Transfer project has been extracted from the "rstudio/keras" GitHub[25]. Contrarily to what we've done in the previous section, we here use a **pre-trained** VGG16 model which, among others, comes with the keras distribution [26]. Examples of *style transfer* performed using that method can be found on the Internet : [27][28].

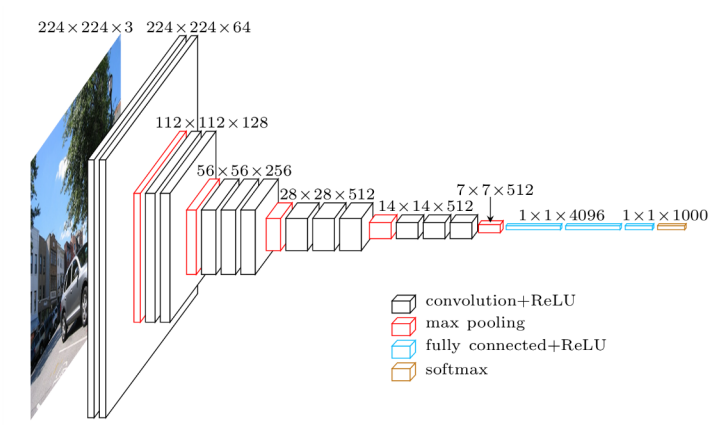

Figure 5: Architecture of a VGG16 model

VGG is a convolutional neural network model for image recognition proposed by the Visual Geometry Group in the University of Oxford[29]. It has originally been introduced with this paper : "Very Deep Convolutional Networks for Large-Scale Image Recognition"[30][31]. VGG16 refers to a VGG model with 16 weight layers[14]. It's a convolutional neural network that is trained on more than a million images from the 'ImageNet' database[32], which contains color images for 1,000 classes.

Figure 5 illustrates the architecture of VGG16: the input layer takes an image in the size of (224 x 224 x 3), and the output layer is a softmax prediction (on 1,000 classes). From the input layer to the last max pooling layer (labeled by 7 x 7 x 512) is regarded as the **feature extraction** part of the model, while the rest of the network is regarded as the **classification** part of the model. We here take advantage of the former and disregard the latter. Further details on the VGG16 architecture can be found Appendix C.

[14] In 2014, 16 and 19 layer networks were considered very deep (although in 2019 we have the ResNet architecture which can be successfully trained at depths of 50-200 for ImageNet and over 1,000 for CIFAR-10).

Within an image, we can distinguish layers that are responsible for the style (basic shapes, colors etc.) and the ones responsible for the content (image-specific features such as outlines and positions), we can separate the CNN layers to independently work on the content and style.

We set our task as an optimization problem where we are going to minimize:

- **content loss** (distance between the input and output images - we strive to preserve the content)
- **style loss** (distance between the style and output images - we strive to apply a new style)
- **total variation loss** (spatial smoothness to denoise the output image)

Finally, we set our gradients and minimize our global loss (which is a combination of content, style and total variation losses) with the L-BFGS-B[33] algorithm[15].

3.2 Application

Back in 2016-2017, in the frame of the **ESSEC&Mannheim EMBA** I pursued[35], we had several exciting opportunities to widen our horizons. We for instance followed the **Leadership Training program at Saint-Cyr Coëtquidan**[36]. We also had several business residencies worldwide. We went to **UCLA** (California, U.S.), to the **Indian Institute of Management Ahmedabad** (Gujarat, India), to the **ESSEC Asia-Pacific** campus (Singapore), as well as to **Mannheim Business School** (Germany). During these residencies, we visited several very exciting companies such as **SpaceX**[37] and **Grab**[38]. Those are all great memories.

To illustrate the capabilities of our *style transfer* model, we use a picture of me taken in the luxurious alleys of the IIM Ahmedabad campus as the "original content" picture.
We'll showcase two distinct examples and apply two very different styles to that picture. We'll generate each time a brand new image, with the content from the 'original content' picture and the style from the 'new style' picture[16]. Those two examples consist in applying a *beach* style on one hand and a *snowy wood* style on the other.
As can be seen on the respective figures 6 & 7, the smallest details from the "original content" are not all grasped by the algorithm but the main components (character in a particular position with landscape characteristics such as the presence of a building at the right in the background) are captured efficiently in the generated new image.

The results are astonishing.

It takes no longer than a couple minutes in processing time for the *style tranfer* algorithm each time to iteratively come up with the results.

[15] the original math behind it all was originally uncovered in the Cornwell University paper by Leon A. Gatys, Alexander S. Ecker, Matthias Bethge[34].

[16] both of the employed 'new style' pictures have been taken from the bank of copyright free images **pixabay.com**[39][40].

original content picture

generated picture

new style picture

Figure 6: Beach style transfer

In this first example here figure 6, tree leaves are shaded, as if to evoke a cloudless sky over a caribbean blue sea. It is also fascinating to notice that the low wall I'm leaning on in the "original content" picture became a yellow sandy beach in the "newly created" picture.

original content picture

generated picture

new style picture

Figure 7: Snowy wood style transfer

On this second *style transfer* example here figure 7, my skin is made much more pale ; as to be closer to the tan of the model from the “new style” picture. An other interesting remark we can make is that it seems that the algorithm has added a couple tree trunks in the image it generated.

4 Discussion

The focus of the herein report has been very narrow. It consisted of only navigating around the topic of image processing with CNNs. For starters, we've seen that object classification could be achieved with a fairly high accuracy[17], and then we've explored in a recreational way the strength of feature extraction with picture style tranfer. The idea was to give a glimpse of how broad the capabilities of Convolutional Neural Networks can be.

Todays' entrepreneurs have a chance to see Deep Learning as a toolset that provides possibilities for endless new building block combinations. With the right association, it can be a valuable contributor to feature engineering and fit plethora of business scenarios. With a little bit of creativity and imagination, there's still so much room to further drive innovation.

Deep Learning is a very dynamic field. Contrarily to other areas where research is concentrated in academic spheres, machine learning gained a lot of private traction in the recent years due to its many applications and promises. Deep learning and Convolutional Neural Networks are not exempt from that phenomenon. It is for instance hard to ignore the potential applications of CNN to the field of computer vision for self-driving cars development[43][44]. And that's just on the algorithm front.

Hardware manufacturers are competing fiercly, trying to elbow their way through providing the best AI chispset[45]. But the future is knocking at our doors already with the advancements made in quantum computing.
The concept of indeterminacy (indefinite state) from Quatum physics offers a whole world of new possibilities when applied to machine learning ; going from Classical to Quantum predictions (as summarized by the famous "quantum coin toss" problem[46]) is a near-future game changer. This was brilliantly articulated in her recent TED Talk by Dr. Shohini GHOSE : "Quantum computing explained in 10 minutes"[47]. Quantum computers are there depicted as unbeatable champions with, in the end, what could be perceived as psychic-like superpowers.

Ideas like that, I certainly find invigorating.

---

[17] Image classification consists in identifying a main object inside a scene. A much harder thing to achieve is object detection (locate and identify objects in an image), as glanced over in that 2 parts "keras with R" tutorial : [41], [42]

Appendices

Appendix A

Architecture of the DenseNet-40-12 model employed for cifar-10 objects classification

```
## Model: "model_1"
## __________________________________________________________________________
## Layer (type)            Output Shape     Param #  Connected to
## ==========================================================================
## input_1 (InputLayer)    (None, 32, 32, 3 0
## __________________________________________________________________________
## initial_conv2D (Conv2D) (None, 32, 32, 1 432      input_1[0][0]
## __________________________________________________________________________
## batch_normalization_1 ( (None, 32, 32, 1 64       initial_conv2D[0][0]
## __________________________________________________________________________
## activation_1 (Activatio (None, 32, 32, 1 0        batch_normalization_1[0][
## __________________________________________________________________________
## conv2d_1 (Conv2D)       (None, 32, 32, 1 1728     activation_1[0][0]
## __________________________________________________________________________
## dropout_1 (Dropout)     (None, 32, 32, 1 0        conv2d_1[0][0]
## __________________________________________________________________________
## concatenate_1 (Concaten (None, 32, 32, 2 0        initial_conv2D[0][0]
##                                                   dropout_1[0][0]
## __________________________________________________________________________
## batch_normalization_2 ( (None, 32, 32, 2 112      concatenate_1[0][0]
## __________________________________________________________________________
## activation_2 (Activatio (None, 32, 32, 2 0        batch_normalization_2[0][
## __________________________________________________________________________
## conv2d_2 (Conv2D)       (None, 32, 32, 1 3024     activation_2[0][0]
## __________________________________________________________________________
## dropout_2 (Dropout)     (None, 32, 32, 1 0        conv2d_2[0][0]
## __________________________________________________________________________
## concatenate_2 (Concaten (None, 32, 32, 4 0        concatenate_1[0][0]
##                                                   dropout_2[0][0]
## __________________________________________________________________________
## batch_normalization_3 ( (None, 32, 32, 4 160      concatenate_2[0][0]
## __________________________________________________________________________
## activation_3 (Activatio (None, 32, 32, 4 0        batch_normalization_3[0][
## __________________________________________________________________________
## conv2d_3 (Conv2D)       (None, 32, 32, 1 4320     activation_3[0][0]
## __________________________________________________________________________
## dropout_3 (Dropout)     (None, 32, 32, 1 0        conv2d_3[0][0]
## __________________________________________________________________________
## concatenate_3 (Concaten (None, 32, 32, 5 0        concatenate_2[0][0]
##                                                   dropout_3[0][0]
## __________________________________________________________________________
## batch_normalization_4 ( (None, 32, 32, 5 208      concatenate_3[0][0]
## __________________________________________________________________________
## activation_4 (Activatio (None, 32, 32, 5 0        batch_normalization_4[0][
## __________________________________________________________________________
## conv2d_4 (Conv2D)       (None, 32, 32, 1 5616     activation_4[0][0]
## __________________________________________________________________________
## dropout_4 (Dropout)     (None, 32, 32, 1 0        conv2d_4[0][0]
## __________________________________________________________________________
## concatenate_4 (Concaten (None, 32, 32, 6 0        concatenate_3[0][0]
##                                                   dropout_4[0][0]
## __________________________________________________________________________
## batch_normalization_5 ( (None, 32, 32, 6 256      concatenate_4[0][0]
## __________________________________________________________________________
```

```
## activation_5 (Activatio (None, 32, 32, 6 0        batch_normalization_5[0][[
## ____________________________________________________________________________
## conv2d_5 (Conv2D)       (None, 32, 32, 1 6912     activation_5[0][0]
## ____________________________________________________________________________
## dropout_5 (Dropout)     (None, 32, 32, 1 0        conv2d_5[0][0]
## ____________________________________________________________________________
## concatenate_5 (Concaten (None, 32, 32, 7 0        concatenate_4[0][0]
##                                                   dropout_5[0][0]
## ____________________________________________________________________________
## batch_normalization_6 ( (None, 32, 32, 7 304      concatenate_5[0][0]
## ____________________________________________________________________________
## activation_6 (Activatio (None, 32, 32, 7 0        batch_normalization_6[0][[
## ____________________________________________________________________________
## conv2d_6 (Conv2D)       (None, 32, 32, 1 8208     activation_6[0][0]
## ____________________________________________________________________________
## dropout_6 (Dropout)     (None, 32, 32, 1 0        conv2d_6[0][0]
## ____________________________________________________________________________
## concatenate_6 (Concaten (None, 32, 32, 8 0        concatenate_5[0][0]
##                                                   dropout_6[0][0]
## ____________________________________________________________________________
## batch_normalization_7 ( (None, 32, 32, 8 352      concatenate_6[0][0]
## ____________________________________________________________________________
## activation_7 (Activatio (None, 32, 32, 8 0        batch_normalization_7[0][[
## ____________________________________________________________________________
## conv2d_7 (Conv2D)       (None, 32, 32, 1 9504     activation_7[0][0]
## ____________________________________________________________________________
## dropout_7 (Dropout)     (None, 32, 32, 1 0        conv2d_7[0][0]
## ____________________________________________________________________________
## concatenate_7 (Concaten (None, 32, 32, 1 0        concatenate_6[0][0]
##                                                   dropout_7[0][0]
## ____________________________________________________________________________
## batch_normalization_8 ( (None, 32, 32, 1 400      concatenate_7[0][0]
## ____________________________________________________________________________
## activation_8 (Activatio (None, 32, 32, 1 0        batch_normalization_8[0][[
## ____________________________________________________________________________
## conv2d_8 (Conv2D)       (None, 32, 32, 1 10800    activation_8[0][0]
## ____________________________________________________________________________
## dropout_8 (Dropout)     (None, 32, 32, 1 0        conv2d_8[0][0]
## ____________________________________________________________________________
## concatenate_8 (Concaten (None, 32, 32, 1 0        concatenate_7[0][0]
##                                                   dropout_8[0][0]
## ____________________________________________________________________________
## batch_normalization_9 ( (None, 32, 32, 1 448      concatenate_8[0][0]
## ____________________________________________________________________________
## activation_9 (Activatio (None, 32, 32, 1 0        batch_normalization_9[0][[
## ____________________________________________________________________________
## conv2d_9 (Conv2D)       (None, 32, 32, 1 12096    activation_9[0][0]
## ____________________________________________________________________________
## dropout_9 (Dropout)     (None, 32, 32, 1 0        conv2d_9[0][0]
## ____________________________________________________________________________
## concatenate_9 (Concaten (None, 32, 32, 1 0        concatenate_8[0][0]
##                                                   dropout_9[0][0]
## ____________________________________________________________________________
## batch_normalization_10  (None, 32, 32, 1 496      concatenate_9[0][0]
```

```
## ____________________________________________________________________________
## activation_10 (Activati (None, 32, 32, 1 0        batch_normalization_10[0]
## ____________________________________________________________________________
## conv2d_10 (Conv2D)      (None, 32, 32, 1 13392    activation_10[0][0]
## ____________________________________________________________________________
## dropout_10 (Dropout)    (None, 32, 32, 1 0        conv2d_10[0][0]
## ____________________________________________________________________________
## concatenate_10 (Concate (None, 32, 32, 1 0        concatenate_9[0][0]
##                                                   dropout_10[0][0]
## ____________________________________________________________________________
## batch_normalization_11  (None, 32, 32, 1 544      concatenate_10[0][0]
## ____________________________________________________________________________
## activation_11 (Activati (None, 32, 32, 1 0        batch_normalization_11[0]
## ____________________________________________________________________________
## conv2d_11 (Conv2D)      (None, 32, 32, 1 14688    activation_11[0][0]
## ____________________________________________________________________________
## dropout_11 (Dropout)    (None, 32, 32, 1 0        conv2d_11[0][0]
## ____________________________________________________________________________
## concatenate_11 (Concate (None, 32, 32, 1 0        concatenate_10[0][0]
##                                                   dropout_11[0][0]
## ____________________________________________________________________________
## batch_normalization_12  (None, 32, 32, 1 592      concatenate_11[0][0]
## ____________________________________________________________________________
## activation_12 (Activati (None, 32, 32, 1 0        batch_normalization_12[0]
## ____________________________________________________________________________
## conv2d_12 (Conv2D)      (None, 32, 32, 1 15984    activation_12[0][0]
## ____________________________________________________________________________
## dropout_12 (Dropout)    (None, 32, 32, 1 0        conv2d_12[0][0]
## ____________________________________________________________________________
## concatenate_12 (Concate (None, 32, 32, 1 0        concatenate_11[0][0]
##                                                   dropout_12[0][0]
## ____________________________________________________________________________
## batch_normalization_13  (None, 32, 32, 1 640      concatenate_12[0][0]
## ____________________________________________________________________________
## activation_13 (Activati (None, 32, 32, 1 0        batch_normalization_13[0]
## ____________________________________________________________________________
## conv2d_13 (Conv2D)      (None, 32, 32, 1 25600    activation_13[0][0]
## ____________________________________________________________________________
## dropout_13 (Dropout)    (None, 32, 32, 1 0        conv2d_13[0][0]
## ____________________________________________________________________________
## average_pooling2d_1 (Av (None, 16, 16, 1 0        dropout_13[0][0]
## ____________________________________________________________________________
## batch_normalization_14  (None, 16, 16, 1 640      average_pooling2d_1[0][0]
## ____________________________________________________________________________
## activation_14 (Activati (None, 16, 16, 1 0        batch_normalization_14[0]
## ____________________________________________________________________________
## conv2d_14 (Conv2D)      (None, 16, 16, 1 17280    activation_14[0][0]
## ____________________________________________________________________________
## dropout_14 (Dropout)    (None, 16, 16, 1 0        conv2d_14[0][0]
## ____________________________________________________________________________
## concatenate_13 (Concate (None, 16, 16, 1 0        average_pooling2d_1[0][0]
##                                                   dropout_14[0][0]
## ____________________________________________________________________________
## batch_normalization_15  (None, 16, 16, 1 688      concatenate_13[0][0]
```

```
## ____________________________________________________________________________
## activation_15 (Activati (None, 16, 16, 1 0        batch_normalization_15[0]
## ____________________________________________________________________________
## conv2d_15 (Conv2D)      (None, 16, 16, 1 18576    activation_15[0][0]
## ____________________________________________________________________________
## dropout_15 (Dropout)    (None, 16, 16, 1 0        conv2d_15[0][0]
## ____________________________________________________________________________
## concatenate_14 (Concate (None, 16, 16, 1 0        concatenate_13[0][0]
##                                                   dropout_15[0][0]
## ____________________________________________________________________________
## batch_normalization_16  (None, 16, 16, 1 736      concatenate_14[0][0]
## ____________________________________________________________________________
## activation_16 (Activati (None, 16, 16, 1 0        batch_normalization_16[0]
## ____________________________________________________________________________
## conv2d_16 (Conv2D)      (None, 16, 16, 1 19872    activation_16[0][0]
## ____________________________________________________________________________
## dropout_16 (Dropout)    (None, 16, 16, 1 0        conv2d_16[0][0]
## ____________________________________________________________________________
## concatenate_15 (Concate (None, 16, 16, 1 0        concatenate_14[0][0]
##                                                   dropout_16[0][0]
## ____________________________________________________________________________
## batch_normalization_17  (None, 16, 16, 1 784      concatenate_15[0][0]
## ____________________________________________________________________________
## activation_17 (Activati (None, 16, 16, 1 0        batch_normalization_17[0]
## ____________________________________________________________________________
## conv2d_17 (Conv2D)      (None, 16, 16, 1 21168    activation_17[0][0]
## ____________________________________________________________________________
## dropout_17 (Dropout)    (None, 16, 16, 1 0        conv2d_17[0][0]
## ____________________________________________________________________________
## concatenate_16 (Concate (None, 16, 16, 2 0        concatenate_15[0][0]
##                                                   dropout_17[0][0]
## ____________________________________________________________________________
## batch_normalization_18  (None, 16, 16, 2 832      concatenate_16[0][0]
## ____________________________________________________________________________
## activation_18 (Activati (None, 16, 16, 2 0        batch_normalization_18[0]
## ____________________________________________________________________________
## conv2d_18 (Conv2D)      (None, 16, 16, 1 22464    activation_18[0][0]
## ____________________________________________________________________________
## dropout_18 (Dropout)    (None, 16, 16, 1 0        conv2d_18[0][0]
## ____________________________________________________________________________
## concatenate_17 (Concate (None, 16, 16, 2 0        concatenate_16[0][0]
##                                                   dropout_18[0][0]
## ____________________________________________________________________________
## batch_normalization_19  (None, 16, 16, 2 880      concatenate_17[0][0]
## ____________________________________________________________________________
## activation_19 (Activati (None, 16, 16, 2 0        batch_normalization_19[0]
## ____________________________________________________________________________
## conv2d_19 (Conv2D)      (None, 16, 16, 1 23760    activation_19[0][0]
## ____________________________________________________________________________
## dropout_19 (Dropout)    (None, 16, 16, 1 0        conv2d_19[0][0]
## ____________________________________________________________________________
## concatenate_18 (Concate (None, 16, 16, 2 0        concatenate_17[0][0]
##                                                   dropout_19[0][0]
## ____________________________________________________________________________
```

```
## batch_normalization_20  (None, 16, 16, 2 928      concatenate_18[0][0]
## ____________________________________________________________________________
## activation_20 (Activati (None, 16, 16, 2 0        batch_normalization_20[0]
## ____________________________________________________________________________
## conv2d_20 (Conv2D)      (None, 16, 16, 1 25056    activation_20[0][0]
## ____________________________________________________________________________
## dropout_20 (Dropout)    (None, 16, 16, 1 0        conv2d_20[0][0]
## ____________________________________________________________________________
## concatenate_19 (Concate (None, 16, 16, 2 0        concatenate_18[0][0]
##                                                   dropout_20[0][0]
## ____________________________________________________________________________
## batch_normalization_21  (None, 16, 16, 2 976      concatenate_19[0][0]
## ____________________________________________________________________________
## activation_21 (Activati (None, 16, 16, 2 0        batch_normalization_21[0]
## ____________________________________________________________________________
## conv2d_21 (Conv2D)      (None, 16, 16, 1 26352    activation_21[0][0]
## ____________________________________________________________________________
## dropout_21 (Dropout)    (None, 16, 16, 1 0        conv2d_21[0][0]
## ____________________________________________________________________________
## concatenate_20 (Concate (None, 16, 16, 2 0        concatenate_19[0][0]
##                                                   dropout_21[0][0]
## ____________________________________________________________________________
## batch_normalization_22  (None, 16, 16, 2 1024     concatenate_20[0][0]
## ____________________________________________________________________________
## activation_22 (Activati (None, 16, 16, 2 0        batch_normalization_22[0]
## ____________________________________________________________________________
## conv2d_22 (Conv2D)      (None, 16, 16, 1 27648    activation_22[0][0]
## ____________________________________________________________________________
## dropout_22 (Dropout)    (None, 16, 16, 1 0        conv2d_22[0][0]
## ____________________________________________________________________________
## concatenate_21 (Concate (None, 16, 16, 2 0        concatenate_20[0][0]
##                                                   dropout_22[0][0]
## ____________________________________________________________________________
## batch_normalization_23  (None, 16, 16, 2 1072     concatenate_21[0][0]
## ____________________________________________________________________________
## activation_23 (Activati (None, 16, 16, 2 0        batch_normalization_23[0]
## ____________________________________________________________________________
## conv2d_23 (Conv2D)      (None, 16, 16, 1 28944    activation_23[0][0]
## ____________________________________________________________________________
## dropout_23 (Dropout)    (None, 16, 16, 1 0        conv2d_23[0][0]
## ____________________________________________________________________________
## concatenate_22 (Concate (None, 16, 16, 2 0        concatenate_21[0][0]
##                                                   dropout_23[0][0]
## ____________________________________________________________________________
## batch_normalization_24  (None, 16, 16, 2 1120     concatenate_22[0][0]
## ____________________________________________________________________________
## activation_24 (Activati (None, 16, 16, 2 0        batch_normalization_24[0]
## ____________________________________________________________________________
## conv2d_24 (Conv2D)      (None, 16, 16, 1 30240    activation_24[0][0]
## ____________________________________________________________________________
## dropout_24 (Dropout)    (None, 16, 16, 1 0        conv2d_24[0][0]
## ____________________________________________________________________________
## concatenate_23 (Concate (None, 16, 16, 2 0        concatenate_22[0][0]
##                                                   dropout_24[0][0]
```

```
## ____________________________________________________________________________
## batch_normalization_25  (None, 16, 16, 2 1168     concatenate_23[0][0]
## ____________________________________________________________________________
## activation_25 (Activati (None, 16, 16, 2 0        batch_normalization_25[0]
## ____________________________________________________________________________
## conv2d_25 (Conv2D)      (None, 16, 16, 1 31536    activation_25[0][0]
## ____________________________________________________________________________
## dropout_25 (Dropout)    (None, 16, 16, 1 0        conv2d_25[0][0]
## ____________________________________________________________________________
## concatenate_24 (Concate (None, 16, 16, 3 0        concatenate_23[0][0]
##                                                   dropout_25[0][0]
## ____________________________________________________________________________
## batch_normalization_26  (None, 16, 16, 3 1216     concatenate_24[0][0]
## ____________________________________________________________________________
## activation_26 (Activati (None, 16, 16, 3 0        batch_normalization_26[0]
## ____________________________________________________________________________
## conv2d_26 (Conv2D)      (None, 16, 16, 3 92416    activation_26[0][0]
## ____________________________________________________________________________
## dropout_26 (Dropout)    (None, 16, 16, 3 0        conv2d_26[0][0]
## ____________________________________________________________________________
## average_pooling2d_2 (Av (None, 8, 8, 304 0        dropout_26[0][0]
## ____________________________________________________________________________
## batch_normalization_27  (None, 8, 8, 304 1216     average_pooling2d_2[0][0]
## ____________________________________________________________________________
## activation_27 (Activati (None, 8, 8, 304 0        batch_normalization_27[0]
## ____________________________________________________________________________
## conv2d_27 (Conv2D)      (None, 8, 8, 12) 32832    activation_27[0][0]
## ____________________________________________________________________________
## dropout_27 (Dropout)    (None, 8, 8, 12) 0        conv2d_27[0][0]
## ____________________________________________________________________________
## concatenate_25 (Concate (None, 8, 8, 316 0        average_pooling2d_2[0][0]
##                                                   dropout_27[0][0]
## ____________________________________________________________________________
## batch_normalization_28  (None, 8, 8, 316 1264     concatenate_25[0][0]
## ____________________________________________________________________________
## activation_28 (Activati (None, 8, 8, 316 0        batch_normalization_28[0]
## ____________________________________________________________________________
## conv2d_28 (Conv2D)      (None, 8, 8, 12) 34128    activation_28[0][0]
## ____________________________________________________________________________
## dropout_28 (Dropout)    (None, 8, 8, 12) 0        conv2d_28[0][0]
## ____________________________________________________________________________
## concatenate_26 (Concate (None, 8, 8, 328 0        concatenate_25[0][0]
##                                                   dropout_28[0][0]
## ____________________________________________________________________________
## batch_normalization_29  (None, 8, 8, 328 1312     concatenate_26[0][0]
## ____________________________________________________________________________
## activation_29 (Activati (None, 8, 8, 328 0        batch_normalization_29[0]
## ____________________________________________________________________________
## conv2d_29 (Conv2D)      (None, 8, 8, 12) 35424    activation_29[0][0]
## ____________________________________________________________________________
## dropout_29 (Dropout)    (None, 8, 8, 12) 0        conv2d_29[0][0]
## ____________________________________________________________________________
## concatenate_27 (Concate (None, 8, 8, 340 0        concatenate_26[0][0]
##                                                   dropout_29[0][0]
```

```
## ____________________________________________________________________________
## batch_normalization_30  (None, 8, 8, 340 1360     concatenate_27[0][0]
## ____________________________________________________________________________
## activation_30 (Activati (None, 8, 8, 340 0        batch_normalization_30[0]
## ____________________________________________________________________________
## conv2d_30 (Conv2D)      (None, 8, 8, 12) 36720    activation_30[0][0]
## ____________________________________________________________________________
## dropout_30 (Dropout)    (None, 8, 8, 12) 0        conv2d_30[0][0]
## ____________________________________________________________________________
## concatenate_28 (Concate (None, 8, 8, 352 0        concatenate_27[0][0]
##                                                   dropout_30[0][0]
## ____________________________________________________________________________
## batch_normalization_31  (None, 8, 8, 352 1408     concatenate_28[0][0]
## ____________________________________________________________________________
## activation_31 (Activati (None, 8, 8, 352 0        batch_normalization_31[0]
## ____________________________________________________________________________
## conv2d_31 (Conv2D)      (None, 8, 8, 12) 38016    activation_31[0][0]
## ____________________________________________________________________________
## dropout_31 (Dropout)    (None, 8, 8, 12) 0        conv2d_31[0][0]
## ____________________________________________________________________________
## concatenate_29 (Concate (None, 8, 8, 364 0        concatenate_28[0][0]
##                                                   dropout_31[0][0]
## ____________________________________________________________________________
## batch_normalization_32  (None, 8, 8, 364 1456     concatenate_29[0][0]
## ____________________________________________________________________________
## activation_32 (Activati (None, 8, 8, 364 0        batch_normalization_32[0]
## ____________________________________________________________________________
## conv2d_32 (Conv2D)      (None, 8, 8, 12) 39312    activation_32[0][0]
## ____________________________________________________________________________
## dropout_32 (Dropout)    (None, 8, 8, 12) 0        conv2d_32[0][0]
## ____________________________________________________________________________
## concatenate_30 (Concate (None, 8, 8, 376 0        concatenate_29[0][0]
##                                                   dropout_32[0][0]
## ____________________________________________________________________________
## batch_normalization_33  (None, 8, 8, 376 1504     concatenate_30[0][0]
## ____________________________________________________________________________
## activation_33 (Activati (None, 8, 8, 376 0        batch_normalization_33[0]
## ____________________________________________________________________________
## conv2d_33 (Conv2D)      (None, 8, 8, 12) 40608    activation_33[0][0]
## ____________________________________________________________________________
## dropout_33 (Dropout)    (None, 8, 8, 12) 0        conv2d_33[0][0]
## ____________________________________________________________________________
## concatenate_31 (Concate (None, 8, 8, 388 0        concatenate_30[0][0]
##                                                   dropout_33[0][0]
## ____________________________________________________________________________
## batch_normalization_34  (None, 8, 8, 388 1552     concatenate_31[0][0]
## ____________________________________________________________________________
## activation_34 (Activati (None, 8, 8, 388 0        batch_normalization_34[0]
## ____________________________________________________________________________
## conv2d_34 (Conv2D)      (None, 8, 8, 12) 41904    activation_34[0][0]
## ____________________________________________________________________________
## dropout_34 (Dropout)    (None, 8, 8, 12) 0        conv2d_34[0][0]
## ____________________________________________________________________________
## concatenate_32 (Concate (None, 8, 8, 400 0        concatenate_31[0][0]
```

```
##                                                  dropout_34[0][0]
## __________________________________________________________________________
## batch_normalization_35  (None, 8, 8, 400 1600     concatenate_32[0][0]
## __________________________________________________________________________
## activation_35 (Activati (None, 8, 8, 400 0        batch_normalization_35[0]
## __________________________________________________________________________
## conv2d_35 (Conv2D)      (None, 8, 8, 12) 43200    activation_35[0][0]
## __________________________________________________________________________
## dropout_35 (Dropout)    (None, 8, 8, 12) 0        conv2d_35[0][0]
## __________________________________________________________________________
## concatenate_33 (Concate (None, 8, 8, 412 0        concatenate_32[0][0]
##                                                  dropout_35[0][0]
## __________________________________________________________________________
## batch_normalization_36  (None, 8, 8, 412 1648     concatenate_33[0][0]
## __________________________________________________________________________
## activation_36 (Activati (None, 8, 8, 412 0        batch_normalization_36[0]
## __________________________________________________________________________
## conv2d_36 (Conv2D)      (None, 8, 8, 12) 44496    activation_36[0][0]
## __________________________________________________________________________
## dropout_36 (Dropout)    (None, 8, 8, 12) 0        conv2d_36[0][0]
## __________________________________________________________________________
## concatenate_34 (Concate (None, 8, 8, 424 0        concatenate_33[0][0]
##                                                  dropout_36[0][0]
## __________________________________________________________________________
## batch_normalization_37  (None, 8, 8, 424 1696     concatenate_34[0][0]
## __________________________________________________________________________
## activation_37 (Activati (None, 8, 8, 424 0        batch_normalization_37[0]
## __________________________________________________________________________
## conv2d_37 (Conv2D)      (None, 8, 8, 12) 45792    activation_37[0][0]
## __________________________________________________________________________
## dropout_37 (Dropout)    (None, 8, 8, 12) 0        conv2d_37[0][0]
## __________________________________________________________________________
## concatenate_35 (Concate (None, 8, 8, 436 0        concatenate_34[0][0]
##                                                  dropout_37[0][0]
## __________________________________________________________________________
## batch_normalization_38  (None, 8, 8, 436 1744     concatenate_35[0][0]
## __________________________________________________________________________
## activation_38 (Activati (None, 8, 8, 436 0        batch_normalization_38[0]
## __________________________________________________________________________
## conv2d_38 (Conv2D)      (None, 8, 8, 12) 47088    activation_38[0][0]
## __________________________________________________________________________
## dropout_38 (Dropout)    (None, 8, 8, 12) 0        conv2d_38[0][0]
## __________________________________________________________________________
## concatenate_36 (Concate (None, 8, 8, 448 0        concatenate_35[0][0]
##                                                  dropout_38[0][0]
## __________________________________________________________________________
## batch_normalization_39  (None, 8, 8, 448 1792     concatenate_36[0][0]
## __________________________________________________________________________
## activation_39 (Activati (None, 8, 8, 448 0        batch_normalization_39[0]
## __________________________________________________________________________
## global_average_pooling2 (None, 448)      0        activation_39[0][0]
## __________________________________________________________________________
## dense_1 (Dense)         (None, 10)       4490     global_average_pooling2d_
## ==========================================================================
```

```
## Total params: 1,037,818
## Trainable params: 1,019,722
## Non-trainable params: 18,096
## _________________________________________________________________________
```

Appendix B

Examples of faulty object classification by our Densenet deep learning model

Figure 8: 50 cases of "a cat for a dog" mis-classification

Appendix C

Architecture of the VGG-16 model employed for picture style transfer

```
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## input_3 (InputLayer)         (None, None, None, 3)     0
## _________________________________________________________________
## block1_conv1 (Conv2D)        (None, None, None, 64)    1792
## _________________________________________________________________
## block1_conv2 (Conv2D)        (None, None, None, 64)    36928
## _________________________________________________________________
## block1_pool (MaxPooling2D)   (None, None, None, 64)    0
## _________________________________________________________________
## block2_conv1 (Conv2D)        (None, None, None, 128)   73856
## _________________________________________________________________
## block2_conv2 (Conv2D)        (None, None, None, 128)   147584
## _________________________________________________________________
## block2_pool (MaxPooling2D)   (None, None, None, 128)   0
## _________________________________________________________________
## block3_conv1 (Conv2D)        (None, None, None, 256)   295168
## _________________________________________________________________
## block3_conv2 (Conv2D)        (None, None, None, 256)   590080
## _________________________________________________________________
## block3_conv3 (Conv2D)        (None, None, None, 256)   590080
## _________________________________________________________________
## block3_pool (MaxPooling2D)   (None, None, None, 256)   0
## _________________________________________________________________
## block4_conv1 (Conv2D)        (None, None, None, 512)   1180160
## _________________________________________________________________
## block4_conv2 (Conv2D)        (None, None, None, 512)   2359808
## _________________________________________________________________
## block4_conv3 (Conv2D)        (None, None, None, 512)   2359808
## _________________________________________________________________
## block4_pool (MaxPooling2D)   (None, None, None, 512)   0
## _________________________________________________________________
## block5_conv1 (Conv2D)        (None, None, None, 512)   2359808
## _________________________________________________________________
## block5_conv2 (Conv2D)        (None, None, None, 512)   2359808
## _________________________________________________________________
## block5_conv3 (Conv2D)        (None, None, None, 512)   2359808
## _________________________________________________________________
## block5_pool (MaxPooling2D)   (None, None, None, 512)   0
## =================================================================
## Total params: 14,714,688
## Trainable params: 14,714,688
## Non-trainable params: 0
## _________________________________________________________________
```

References

[1] R – GPU Programming for All with 'gpuR'. https://www.r-bloggers.com/r-gpu-programming-for-all-with-gpur/], March 2016.

[2] R tutorial - GPU Computing with R. http://www.r-tutor.com/gpu-computing.

[3] Keras backends. https://keras.io/backend/.

[4] Local GPU - TensorFlow for R. https://tensorflow.rstudio.com/tools/local_gpu.html.

[5] Install Keras and the TensorFlow backend - RStudio. https://keras.rstudio.com/reference/install_keras.html.

[6] Getting Started with Keras - RStudio. https://keras.rstudio.com/articles/getting_started.html#tutorials.

[7] Keras available layers - RStudio. https://keras.rstudio.com/reference/#section-core-layers.

[8] Deep Neural Network - image source. https://searchenterpriseai.techtarget.com/definition/neural-network.

[9] What's the difference between a matrix and a tensor? https://medium.com/@quantumsteinke/whats-the-difference-between-a-matrix-and-a-tensor-4505fbdc576c], August 2017.

[10] In what do machine learning "tensors" differ from tensors in mathematics? https://stats.stackexchange.com/questions/198061/why-the-sudden-fascination-with-tensors#198395], Sptember 2016.

[11] The Ultimate Guide to Convolutional Neural Networks (CNN). https://www.superdatascience.com/blogs/the-ultimate-guide-to-convolutional-neural-networks-cnn], August 2018.

[12] CNN - Web-traffic-forecasting - GitHub. https://github.com/sjvasquez/web-traffic-forecasting.

[13] Predicting Sunspot Frequency with Keras - RStudio. https://blogs.rstudio.com/tensorflow/posts/2018-06-25-sunspots-lstm/.

[14] Time Series Forecasting with Recurrent Neural Networks - RStudio. https://blogs.rstudio.com/tensorflow/posts/2017-12-20-time-series-forecasting-with-recurrent-neural-networks/.

[15] Word2Vec - Vector Representations of Words - TensorFlow. https://www.tensorflow.org/tutorials/representation/word2vec.

[16] A Beginner's Guide to Word2Vec and Neural Word Embeddings. https://skymind.ai/wiki/word2vec.

[17] Convolutional Neural Networks for Sentence Classification - New York University. https://arxiv.org/pdf/1408.5882.pdf], September 2014.

[18] Convolutional Neural Networks for Speech Recognition. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/CNN_ASLPTrans2-14.pdf], February 2016.

[19] Densely Connected Convolutional Networks. https://arxiv.org/pdf/1608.06993.pdf, August 2016.

[20] 80 million tiny images - a large dataset for non-parametric object and scene recognition. http://people.csail.mit.edu/torralba/publications/80millionImages.pdf, 2008.

[21] The CIFAR-10 dataset. https://www.cs.toronto.edu/~kriz/cifar.html.

[22] Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent. https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f, June 2017.

[23] Callback function. https://developer.mozilla.org/en-US/docs/Glossary/Callback_function.

[24] Turning Fortnite into PUBG with Deep Learning (CycleGAN). https://towardsdatascience.com/turning-fortnite-into-pubg-with-deep-learning-cyclegan-2f9d339dcdb0, June 2018.

[25] rstudio/keras - neural_style_transfer - GitHub. https://github.com/rstudio/keras/blob/master/vignettes/examples/neural_style_transfer.R.

[26] rstudio/keras - Applications. https://keras.rstudio.com/reference/#section-applications.

[27] Experiments with style transfer. http://genekogan.com/works/style-transfer/, 2015.

[28] Style transfer - flickr. https://www.flickr.com/photos/genekogan/sets/72157658785675071.

[29] Visual Geometry Group - Department of Engineering Science, University of Oxford. https://www.robots.ox.ac.uk/~vgg/.

[30] Very Deep Convolutional Networks for Large-Scale Visual Recognition - Overview. http://www.robots.ox.ac.uk/~vgg/research/very_deep/, 2014.

[31] Very Deep Convolutional Networks for Large-Scale Visual Recognition - Paper. https://arxiv.org/pdf/1409.1556.pdf, 2014.

[32] The ImageNet database. http://www.image-net.org.

[33] Limited-memory BFGS - Wikipedia. https://en.wikipedia.org/wiki/Limited-memory_BFGS#L-BFGS-B.

[34] A Neural Algorithm of Artistic Style. https://arxiv.org/pdf/1508.06576.pdf, August 2018.

[35] ESSEC&Mannheim EMBA. https://www.mannheim-business-school.com/en/mba-master/essec-mannheim-executive-mba/program-structure/.

[36] Saint-Cyr Executive Education. http://www.scyfco.fr/?lang=en.

[37] SpaceX - designs, manufactures and launches advanced rockets and spacecraft. https://www.spacex.com/.

[38] Grab - Transport, Food Delivery & Payment Solutions. https://www.grab.com/.

[39] Beach style picture - pixabay.com. https://cdn.pixabay.com/photo/2014/04/26/04/25/fitness-332278_960_720.jpg.

[40] Snowy wood style picture - pixabay.com. https://cdn.pixabay.com/photo/2017/08/06/15/20/people-2593421_960_720.jpg.

[41] rstudio/keras - Naming and locating objects in images. https://blogs.rstudio.com/tensorflow/posts/2018-11-05-naming-locating-objects/.

[42] rstudio/keras - Concepts in object detection. https://blogs.rstudio.com/tensorflow/posts/2018-12-18-object-detection-concepts/.

[43] A Self-Driving Robot Using Deep Convolutional Neural Networks on Neuromorphic Hardware. https://arxiv.org/pdf/1611.01235.pdf, November 2016.

[44] End to End Learning for Self-Driving Cars - NVIDIA Corporation. https://becominghuman.ai/self-driving-cars-deep-neural-networks-and-convolutional-neural-networks-applied-to-clone-driving-ee2c623ac9b5, April 2016.

[45] 8 Powerful AI Chips Challenging NVIDIA's Dominance In Computing Industry. https://www.analyticsindiamag.com/8-powerful-ai-chips-challenging-nvidias-dominance-in-computing-industry/, December 2018.

[46] An introduction to quantum coin-tossing. https://arxiv.org/pdf/quant-ph/0206088.pdf, December 2018.

[47] Shohini GHOSE. Quantum computing explained in 10 minutes - TEDWomen. https://www.ted.com/talks/shohini_ghose_quantum_computing_explained_in_10_minutes/transcript?language=en, November 2018.