UNDERSTANDING AND EXEMPLIFYING A CNN

Renan Mantuanelli de Aquino
Deivison Augusto da Vitória
Universidade Federal do Espírito Santo (UFES)
Laboratório de Computação de Alto Desempenho (LCAD)

***Abstract* — This article shows how the Convolutional Neural Network (CNN) has an impressive classification performance. In this article it was considered a smaller CNN because the network was only with one GPU GeForce GTX 1050, and for best results, it is recommended to use more than one GPU with greater capacity. Therefore, it is possible to make a deeper CNN and consequently obtain better results with more data and many different classes. The results show that it is possible to achieve good accuracy with smaller data sets, even using a small network but deep CNN as created by Yann LeCun with some other techniques developed by Geoffrey Hilton to avoid overfitting. Were performed four tests with different architectures, with three different classes, and one of the tests showed a precision of 89.55% accuracy in the validation test.**

***Keywords* — Convolutional Neural Network.**

I. INTRODUCTION

Since your introduction by LeCun in the early 1990's, Convolutional Networks (convnets) have demonstrated excellent performance at tasks such as hand-written digit classification and face detection.

Most notably, (Krizhevsky et al., 2012) show record beating performance on the ImageNet 2012 classification benchmark, with their convnet model achieving an error rate of 16.4% (Supervision Network / AlexNet), compared to the 2nd place result of 26.1% (ISI Network).

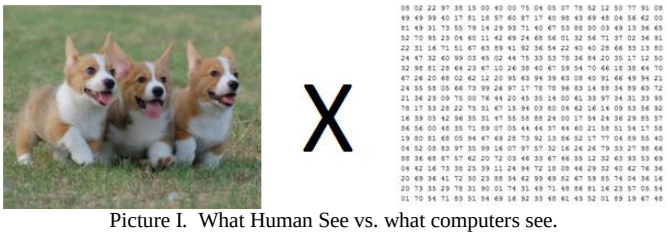
Several factors are responsible for this renewed interest in convnet models. First of all, the availability of much larger training sets, with millions of labeled examples. Second, powerful GPU implementations and lastly, better model regularization strategies, such as Dropout (Hinton et al., 2012).

Ever since then, the companies have been using deep learning at the core of their services. Facebook, for exemple, uses neural nets for their automatic tagging algorithms, Google for their photo search, Amazon for their product recommendations, Pinterest for their home feed personalization, the Instagram for their search infrastructure, among many other applications.

II. LITERATURE REVIEW

A. *Image Processing*

Image classification is the task of taking an input image and outputting a class (cat, dog, etc.) or a probability of classes that best describes the image. For humans, this task of recognition is one of the first skills we learn from the moment we are born and is comes naturally and effortlessly when adults. Without even thinking twice, we are able to quickly and seamlessly identify the environment we are in as well as the objects that surround us. When we see an image or just when we look at the world around us, most of the time we are able to immediately characterize the scene and give each object a label, all done automatically by the brain. That skill of being able to quickly recognize patterns, generalization from prior knowledge, and adapt to different image environments are ours and we not share with machines.



Picture I. What Human See vs. what computers see.

When a computer sees an image (takes an image as input), it will see an array of pixel values. Depending on the resolution and size of the image, it will see a 32 x 32 x 3 array of numbers (the 3 refers to RGB values). For exemple, a JPG form has a color image and its size is 480 x 480. The representative array will be 480 x 480 x 3. Each of these numbers is given a value from 0 to 255, which describes the pixel intensity at that point. These numbers, while meaningless to us when we perform image classification, are the only inputs available to the computer. The idea is that you give the computer this array of numbers and it will output numbers that describe the probability of the image being a certain class (.80 for cat, .15 for dog, .05 for car).

Now that we know the problem, as well as the inputs and outputs, let us think about how to approach this. What we want the computer to do, is to be able to distinguish between all the images that are given and figure out the specifies features that make a dog a dog or that make a cat a cat. This process is done on in our minds subconsciously as well. When we look at a picture of a dog, we can classify it as such if the picture has identifiable features such as paws or four legs. In a similar way, the computer is able perform image classification by looking for low level features such as edges and curves, and then building up to more abstract concepts through a series of convolutional layers. This is a general overview of what a CNN does.

B. Convolutions

The convolutions is a very efficient way of pulling this off, and it makes advantage of the structure of the information encoded within an image. It is assumed that pixels that are spatially closer together will "cooperate" on forming a particular feature of interest much more than ones on opposite corners of the image. Besides, if a particular (smaller) feature is found to be of great importance when defining an image's label, it will be equally important if this feature was found anywhere within the image, regardless of location.

Given a two-dimensional image, I, and a small matrix, K of size h×w, (known as a convolution kernel), which we assume encodes a way interesting of a extracting image feature. Then we compute the convolved image, I∗K, by overlaying the kernel on top of the image in all possible ways, and recording the sum of elementwise products between the image and the kernel:

$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} \cdot I_{x+i-1,y+j-1}$$

The images below show a diagrammatical overview of the above index and the result of applying convolution (with two separate kernels) over an image, to act as an edge detector:

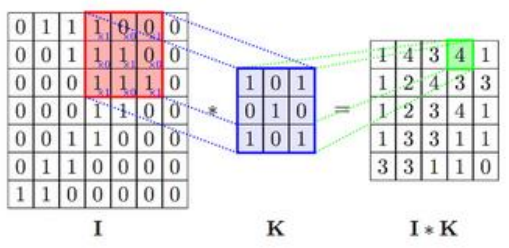


Fig. 2. Convolutions.

C. Convolutions and Pooling Layers

The convolution operator forms the fundamental basis of the convolutional layer of a CNN. The layer is completely specified by a certain number of kernels, $\vec{K}$ (along with additive biases, $\vec{b}$, per each kernel), and it operates by computing the convolution of the output images of a previous layer with each of those kernels, afterwards adding the biases (one per each output image). Finally, an activation function, σ, may be applied to all of the pixels of the output images.

Typically, the input to a convolutional layer will have d *channels* (e.g., red/green/blue in the input layer), in which case the kernels are extended to have this number of channels as well, making the final formula of a single output image channel of a convolutional layer (for a kernel K and bias b) as follows.

$$\mathrm{conv}(I,K)_{xy} = \sigma\left(b + \sum_{i=1}^{h}\sum_{j=1}^{w}\sum_{k=1}^{d} K_{ijk} \cdot I_{x+i-1,y+j-1,k}\right)$$

Finally, note that a convolutional operator is in no way restricted to two-dimensionally structured data. In fact, most machine learning frameworks will provide you with out-of-the-box layers for 1D and 3D convolutions as well.

It is important to note that, while a convolutional layer significantly decreases the number of *parameters* compared to a fully connected (FC) layer, it introduces more hyperparameters.

A very popular approach to downsampling is a *pooling* layer, which consumes small and (usually) disjoint chunks of the image (typically 2×2) and aggregates them into a single value. There are several possible schemes for the aggregation, the most popular being *max pooling*, where the maximum pixel value within each chunk is taken. A diagrammatical illustration of 2×2 max pooling is given below.

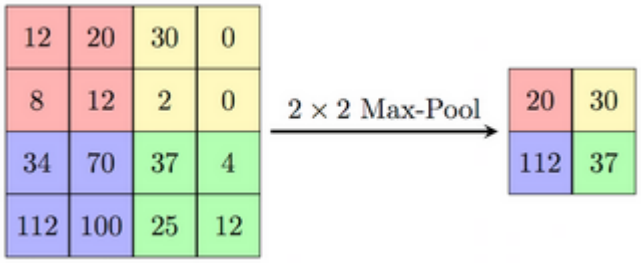


Fig. 3. Max pooling (2x2)

D. Putting All Together: a Common CNN

A typical CNN architecture for a k class image classification can be split into two distinct parts. First a chain of repeating convolutional and pool layers (sometimes with more than one convolutional layer at once), and second followed by a few fully connected layers (taking each pixel of the computed images as an independent input), culminating in a k-way softmax layer, to which a cross-entropy loss is optimized. It is important to remember that after every convolutional or fully connected layer, an activation (e.g. ReLU) will be applied to all of the outputs.

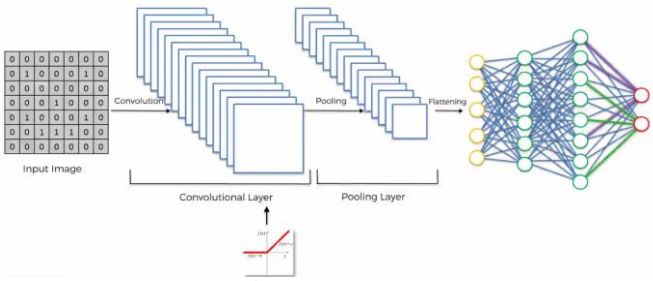


Fig. 4. Example of Full Convolutional Neural Network.

Note the effect of a single convolution and pooling pass through the image, it reduces height and width of the individual channels in favors of their number, i.e., depth.

A softmax layer's purpose is converting any vector of real numbers into a vector of *probabilities* (nonnegative real values that add up to 1). Within this context, the probabilities correspond to the likelihoods that an input image is a member of a particular class. Minimizing the cross-entropy loss has the effect of maximizing the model's confidence in the correct class, without being concerned for the probabilities for other classes. This makes it a more suitable choice for probabilistic tasks compared to, for example, the squared error loss.

E. Overfitting, Regularisation and Dropout

Overfitting corresponds to adapting our model to the training set to such extremes that its generalization potential (performance on samples outside of the training set) is severely limited. In other words, our model might have learned the training set (along with any noise present within it) perfectly, but it has failed to capture the underlying process that generated it. To illustrate, consider a problem of fitting a sine curve, with white additive noise applied to the data points as below.

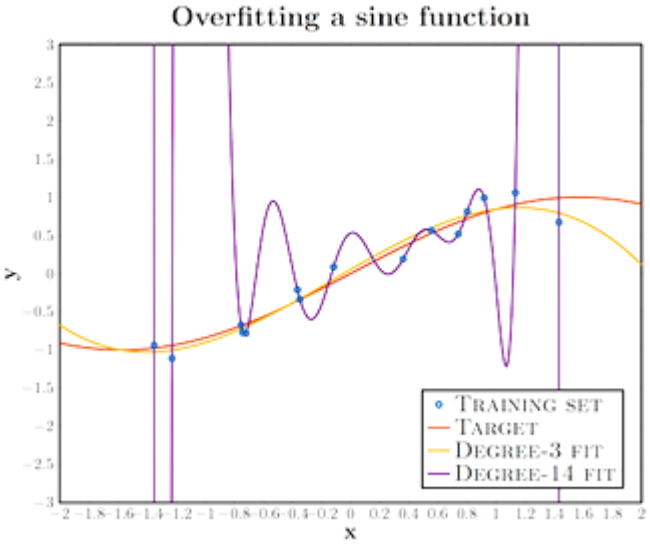


Fig. 5. Example of Overfitting

Here, a training set (denoted by blue circles) derived from the original sine wave, along with some noise. If we fit a degree-3 polynomial to this data, we get a good approximation to the original curve. Someone might argue that a degree-14 polynomial would do better, indeed, given we have 15 points, such a fit would perfectly describe the training data. However, in this case, the additional parameters of the model cause catastrophic results: to cope with the inherent noise of the data, anywhere except in the closest vicinity of the training points, our fit is completely off.

Deep convolutional neural networks have a large number of parameters, especially in the fully connected layers. Overfitting might often manifest in the following form, for example, if we do not have sufficiently many training examples, a small group of neurons might become responsible for doing most of the processing and other neurons becoming redundant. On the other hand, in the other extreme, some neurons might actually become detrimental to performance, with several other neurons of their layer ending up doing nothing else but correcting for their errors.

To help our models generalize better in these circumstances, we introduce techniques of regularization, rather than reducing the number of parameters, we impose constraints on the model parameters during training to keep them from learning the noise in the training data. The particular method is dropout, but actually helps to eliminate exactly the failure modes described above.

Namely, dropout with parameter p will, within a single training iteration, go through all neurons in a particular layer and, with probability p, eliminate them from the network throughout the iteration. This has the effect of forcing the neural network to cope with failures, and not to rely on existence of a particular neuron, or set of neurons, relying more on a consensus of several neurons within a layer. This very simple technique works quite well already for combatting overfitting on its own, without introducing further regularizes.

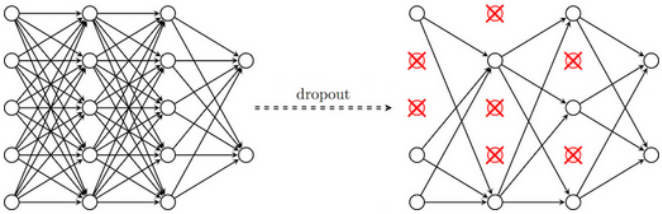


Fig. 6. Example of Dropout

III. Material and Methods

A. Software

Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale.

To install Python, just go to google and text “anaconda phyton” (https://www.continuum.io/downloads), choose the operating system and install.

The next step is go to windows task bar and text anaconda and choose anaconda navigator, choose and install Spyder that is the scientific Python Development Environment.

It is a powerful interactive development environment for the Python language with advanced editing, interactive testing, debugging and introspection features and a numerical computing environment thanks to the support of IPython (enhanced interactive Python interpreter) and popular Python libraries such as NumPy (linear algebra), SciPy (signal and image processing) or matplotlib (interactive 2D/3D plotting).

Spyder may also be used as a library providing powerful console-related widgets for your PyQt (based applications) for example, it may be used to integrate a debugging console directly in the layout of your graphical user interface.

In addition, go to task bar and text Anaconda Prompt, select and install the Theano, Tensorflow and Keras by using the commands below.

- pip install theano
- pip install tensorflow
- pip install keras
- conda update --all (update everything in anaconda).

B. Applying a CNN to classify image

The CNN was develop by using Keras and the model was trained using 15000 images, being 12000 for tests (4000 images for each category) and 3000 to validation tests (1000 images for each category) and finally it was supposed to classify the categories that have most accuracy (category “dog” or “cat” or “car”).

The step by step is described below. In this first case, we classify only the categories "dog" and "cat". The "car" category was inputted in the later example.

To run the CNN, select all the commands and press shift + enter.

(Only categories Dog and Cat)

```
# -*- coding: utf-8 -*-
"""
Spyder Editor
"""
# Part 1 - Building the CNN

# Importing the tensorflow, keras libraries and packages

import tensorflow as tf
import keras
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Activation
from keras.layers import Flatten
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import MaxPool2D
from keras.layers import BatchNormalization
from keras.optimizers import SGD
from keras.optimizers import RMSprop
from keras.utils import multi_gpu_model
from keras.preprocessing.image import ImageDataGenerator
import json
sgd = SGD(lr=0.01, decay=1e-2, momentum=0.9,
nesterov=True)
rms = RMSprop(lr=0.01, rho=0.9, epsilon=1e-08, decay=0.0)


# Initialising the CNN

classifier = Sequential()

# Step 1 – Convolution
classifier.add(Convolution2D(96, 7, 7, border_mode='same',
input_shape=(32, 32, 3), activation = 'relu'))

# Step 2 - Max Pooling
classifier.add(MaxPooling2D(pool_size=(3, 3), strides=2,
padding='same', data_format=None))

# Step 3 - Contrast Image
classifier.add(BatchNormalization())
classifier.compile(optimizer = 'RMSprop', loss =
'categorical_crossentropy')

# Step 4 – Convolution
classifier.add(Convolution2D(256, 5, 5, border_mode='same',
activation = 'relu'))

# Step 5 - Max Pooling
classifier.add(MaxPooling2D(pool_size=(3, 3), strides=2,
padding='same', data_format=None))

# Step 6 - Contrast Image
classifier.add(BatchNormalization())
classifier.compile(optimizer = 'RMSprop', loss =
'categorical_crossentropy')

# Step 7 - Convolution
classifier.add(Convolution2D(384, 3, 3, border_mode='same',
activation = 'relu'))

# Step 8 - Max Pooling
classifier.add(MaxPooling2D(pool_size=(3, 3), strides=1,
padding='same', data_format=None))

# Step 9 - Contrast Image
classifier.add(BatchNormalization())
classifier.compile(optimizer = 'RMSprop', loss =
'categorical_crossentropy')

# Step 10 - Convolution
classifier.add(Convolution2D(384, 3, 3, border_mode='same',
activation = 'relu'))

# Step 11 - Max Pooling
classifier.add(MaxPooling2D(pool_size=(3, 3), strides=1,
padding='same', data_format=None))

# Step 12 - Convolution
classifier.add(Convolution2D(256, 3, 3, border_mode='same',
activation = 'relu'))

# Step 13 - Flattening
classifier.add(Flatten())

# Step 14  - Full Connection
classifier.add(Dense(output_dim = 4096, activation = 'relu'))
classifier.add(Dropout(0.5))
classifier.add(Dense(output_dim = 4096, activation = 'relu'))
classifier.add(Dropout(0.5))
classifier.add(Dense(output_dim = 2, activation = 'softmax'))

# Compiling the CNN
classifier.compile(optimizer = 'sgd', loss =
'categorical_crossentropy', metrics = ['accuracy'])
classifier.summary()

# Part 2 - Fitting the CNN to the images

from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
                    rescale=1./255,
                    shear_range=0.2,
                    zoom_range=0.2,
                    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)
```

```
training_set =
train_datagen.flow_from_directory('dataset/training_set',
                                  target_size=(32, 32),
                                  batch_size=8,
                                  class_mode='categorical')

test_set = test_datagen.flow_from_directory('dataset/test_set',
                                  target_size=(32, 32),
                                  batch_size=8,
                                  class_mode='categorical')

classifier.fit_generator(training_set,
                  steps_per_epoch=8000,
                  epochs=15,
                  validation_data=test_set,
                  validation_steps=2000)
```

 # Part 3 - Save the weights

```
classifier.save('Test.h5')
print('Classifier Saved')
```

 # Part 4 - Save the structure model

```
classifier_json = classifier.to_json()
json_file.write(classifier_json)
```

In the step-by-step described below the category "car" has been inputted. As a result, the classifier will be three categories: “Dog”, “Cat” and “Car”.

The main idea of inserting the “car” category is to train the network with something different from an animal and to verify how the accuracy behaves (better or worse).

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

"""
# Part 1 - Building the CNN

# Importing the keras libraries and packages

import tensorflow as tf
import keras
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Activation
from keras.layers import Flatten
from keras.layers import Convolution2D
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import MaxPool2D
from keras.layers import BatchNormalization
from keras.optimizers import SGD
from keras.optimizers import RMSprop
from keras.utils import multi_gpu_model
from keras.preprocessing.image import
ImageDataGenerator
import json

# Optimizers
sgd = SGD(lr=0.01, decay=1e-2, momentum=0.9,
nesterov=True)
rms = RMSprop(lr=0.01, rho=0.9, epsilon=1e-08,
decay=0.0)


# Initialising the CNN
classifier = Sequential()

# Step 1 – Convolution
classifier.add(Convolution2D(96, 7, 7,
border_mode='same', input_shape=(32, 32, 3),
activation = 'relu'))

# Step 2 - Max Pooling
classifier.add(MaxPooling2D(pool_size=(3, 3),
strides=2, padding='same', data_format=None))

# Step 3 - Contrast Image
classifier.add(BatchNormalization())
classifier.compile(optimizer = 'RMSprop', loss =
'categorical_crossentropy')

# Step 4 - Convolution
classifier.add(Convolution2D(256, 5, 5,
border_mode='same', activation = 'relu'))

# Step 5 - Max Pooling
classifier.add(MaxPooling2D(pool_size=(3, 3),
strides=2, padding='same', data_format=None))

# Step 6 - Contrast Image
classifier.add(BatchNormalization())
classifier.compile(optimizer = 'RMSprop', loss =
'categorical_crossentropy')

# Step 7 - Convolution
classifier.add(Convolution2D(512, 3, 3,
border_mode='same', activation = 'relu'))

# Step 8 - Convolution
classifier.add(Convolution2D(1024, 3, 3,
border_mode='same', activation = 'relu'))

# Step 9 - Convolution
classifier.add(Convolution2D(512, 3, 3,
border_mode='same', activation = 'relu'))
classifier.add(MaxPooling2D(pool_size=(3, 3),
strides=2, padding='same', data_format=None))

# Step 10- Flattening
classifier.add(Flatten())

# Step 11  - Full Connection
classifier.add(Dense(output_dim = 4096, activation =
'relu'))
classifier.add(Dropout(0.5))
classifier.add(Dense(output_dim = 4096, activation =
'relu'))
classifier.add(Dropout(0.5))
classifier.add(Dense(output_dim = 3, activation =
'softmax'))

# Compiling the CNN
```

```
classifier.compile(optimizer = 'sgd', loss =
'categorical_crossentropy', metrics = ['accuracy'])

classifier.summary()

# Part 2 - Fitting the CNN to the images
from keras.preprocessing.image import
ImageDataGenerator

train_datagen = ImageDataGenerator(
                        rescale=1./255,
                        shear_range=0.2,
                        zoom_range=0.2,
                        horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

training_set =
train_datagen.flow_from_directory('dataset/training_set'
,
                                  target_size=(32, 32),
                                  batch_size=12,

class_mode='categorical')

test_set =
test_datagen.flow_from_directory('dataset/test_set',
                                 target_size=(32, 32),
                                 batch_size=12,
                                 class_mode='categorical')

classifier.fit_generator(training_set,
                         steps_per_epoch=12000,
                         epochs=30,
                         validation_data=test_set,
                         validation_steps=3000)
#
# Part 3 - Save the weights
classifier.save('Aquino&VitóriaNet.h5')
print('Classifier Saved')

# Part 4 - Making new pewdictions
import numpy as np
from keras.preprocessing import image

test_image =
image.load_img('dataset/single_prediction/cat_or_dog_
or_car_3.jpg', target_size = (32, 32))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = classifier.predict(test_image)
print(result)
training_set.class_indices

test_image =
image.load_img('dataset/single_prediction/cat_or_dog_
or_car_2.jpg', target_size = (32, 32))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = classifier.predict(test_image)
print(result)
training_set.class_indices

test_image =
image.load_img('dataset/single_prediction/cat_or_dog_
or_car_1.jpg', target_size = (32, 32))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = classifier.predict(test_image)
print(result)
training_set.class_indices

# Part 5 - Save the structure model
classifier_json = classifier.to_json()
with open("Aquino&VitóriaNetStructure.json", "w") as
json_file:
    json_file.write(classifier_json)
```

IV. EXPERIMENTS AND ANALYSIS

The model introduced in the previous section are evaluated in the experiments. The dataset is introduced in this section, then the setups and parameter settings for the experiments are illustrated. In the end, we analyze and discuss the mains problems to get the best configuration of a CNN.

The architecture of the tests is shown in Table I. There were four tests to find out which would be the best architecture, where the highest accuracy was observed and how the training time behaved for each of the tests.

The initial test (zero test) was done with 5 convolutions, 4 max polling, 3 batch norm and 2 dropouts. The other tests were performed with 5 convolutions, 3 max pulling, 2 batch norm and 2 dropouts. The architecture difference between the zero test and the other tests is that in the third, fourth, and fifth convolutions, the number of layer are different (bigger).

The column Param# shows the amount of neurons in each layer.

Layer (type)	Output Shape				Param #			
	Teste 0	Teste 1	Teste 2	Teste 3	Teste 0	Teste 1	Teste 2	Teste 3
Conv 1	32,32,96	32,32,96	32,32,96	32,32,96	14208	14208	14208	14208
Max Pol 1	16,16,96	16,16,96	16,16,96	16,16,96	0	0	0	0
Bath Norm 1	16,16,96	16,16,96	16,16,96	16,16,96	384	384	384	384
Conv 2	16,16,256	16,16,256	16,16,256	16,16,256	614656	614656	614656	614656
Max Pol 2	8,8,256	8,8,256	8,8,256	8,8,256	0	0	0	0
Bath Norm 2	8,8,256	8,8,256	8,8,256	8,8,256	1024	1024	1024	1024
Conv 3	8,8,384	8,8,512	8,8,512	8,8,512	885120	1180160	1180160	1180160
Max Pol 3	8,8,384	-	-	-	0	-	-	-
Bath Norm 3	8,8,384	-	-	-	1536	-	-	-
Conv 4	8,8,384	8,8,1024	8,8,1024	8,8,1024	1327488	4719616	4719616	4719616
Max Pol 4	8,8,384	-	-	-	0	-	-	-
Conv 5	8,8,256	8,8,512	8,8,512	8,8,512	884992	4719104	4719104	4719104
Max Pol 5	-	4,4,512	4,4,512	4,4,512	-	0	0	0
Dropout 1	4096	4096	4096	4096	0	0	0	0
Dropout 2	4096	4096	4096	4096	0	0	0	0

TABLE I: Comparison of four models.

The test results are listed in Table II. The total of the parameters in the zero test was higher than in the other tests. The training time of the zero test was lower than the other tests performed due to the lower epoch number. In addition, when we do the epoch / time relationship, the zero test continues to have the shortest time. On the other hand, Test 2 does not run, so it was not possible to measure its timing and accuracy, but we try again and run the CNN with the same parameters to get results. The training accuracy of tests 2 and 3 was higher than that of the zero test because there were more epochs than test 0. And finally, the accuracy of the validation test in test 3 was better than the others tests.

	Teste 0	Teste 1	Teste 2	Teste 3
Total params	87,631,874	61,597,186	61,597,186	61,601,283
Trainable params	87,630,402	61,596,482	61,596,482	61,600,579
Epoch	1/15	1/50	1/50	1/30
Time (min)	201	Error	847,75	777,16
Time/Epoch	13,4	-	16,95	29,92
Accuracy Training	0,9825	-	0,9994	0,9987
Accuracy Validation Test	0,8221	-	0,8554	0,8955

TABLE II: Results.

In general, test 3 got the best results even with one class (car) over the other tests. The architecture of test 3 proved to be more efficient because it managed to balance the right amount of epochs with the number of classes, making its accuracy result larger. It is important to note that the training time of test 3 was more than double the time of the test 0, but as the result of accuracy was 8.19% higher than the result. Test 1 did not help increase the volume of analysis data as it did not run, but it did help in the adjustments needed to run tests 2 and 3.

As the best test result was from test 3, an implementation test was performed with a dog, cat and car image different from those used in learning, validation and prediction as follows.

Picture II: Image of the dog used in the test.

```
test_image =
image.load_img('dataset/single_prediction/cat_or_dog_or_ca
r_1.jpg', target_size = (32, 32))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = classifier.predict(test_image)
print(result)
training_set.class_indices
[[ 0.  0.  1.]]
Out[51]: {'cars': 0, 'cats': 1, 'dogs': 2}
```

Picture III: Image of the cat used in the test.

```
test_image =
image.load_img('dataset/single_prediction/cat_or_dog_or_ca
r_2.jpg', target_size = (32, 32))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = classifier.predict(test_image)
print(result)
training_set.class_indices
[[ 0.  1.  0.]]
Out[52]: {'cars': 0, 'cats': 1, 'dogs': 2}
```

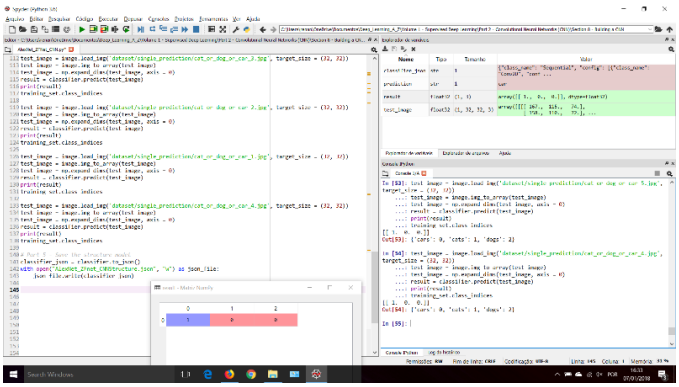
Picture IV: Image of the car used in the test.

```
test_image =
image.load_img('dataset/single_prediction/cat_or_dog_or_ca
r_4.jpg', target_size = (32, 32))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = classifier.predict(test_image)
print(result)
training_set.class_indices
[[ 1.  0.  0.]]
Out[53]: {'cars': 0, 'cats': 1, 'dogs': 2}
```

Finally, the implementation tests, was used an image of IARA (Intelligent Autonomous Robotic Automobile) to verify that the network would correctly identify it as a car.

Picture V: Image of the IARA used in the test.

```
test_image =
image.load_img('dataset/single_prediction/cat_or_dog_or_ca
r_4.jpg', target_size = (32, 32))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = classifier.predict(test_image)
print(result)
training_set.class_indices
[[ 1.  0.  0.]]
Out[54]: {'cars': 0, 'cats': 1, 'dogs': 2}
```



All tests with the test 3 architecture have been successful and the network is operating within expected normalcy as shown above.

V. Conclusion and Future Work

The article has covered the essentials of convolutional neural networks, introduced the problem of overfitting, and made a very brief dent into how it could be rectified via regularization (by applying dropout) and successfully implemented a CNN in Keras for image classification.

An ablation study on the model revealed that having a minimum depth to the network, rather than any individual section, is vital to the model's performance.

We can conclude that it is necessary to balance the amount of epochs with the amount of layers to obtain the optimal point (accuracy x learning time). In addition, it is important to be well aware of the processor's ability to ensure that it is capable of supporting the architecture developed for the intended purpose.

For future work, other network structures can be created to improve accuracy and / or reduce training time. It is also possible to increase the number of classes, since only 3 classes were considered in this work.

VI. References

[1] HINTON, G. E., Osindero, S., and The, Y. A fast learning algorithm for deep belief nets. Neural Computation, 18:1527{1554, 2006.

[2] HINTON, G.E., Srivastave, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580, 2012.

[3] KRIZHEVSKY, A., Sutskever, I., and Hinton, G.E. Imagenet classication with deep convolutional neural networks. In NIPS, 2012.

[4] LECUN, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. Neural Comput., 1(4):541{551, 1989.

[5] SOHN, K., Jung, D., Lee, H., and Hero III, A. Efficient learning of sparse, distributed, convolutional feature representations for object recognition. In ICCV, 2011.