Convolutional Neural Network for the Identification of Spiders of Medical Importance

Juan Carlos Ángeles Cerón
Tecnológico de Monterrey, Campus Querétaro
May, 2019

***Abstract*— This document presents the development work and implementation of a Deep Learning-based spider classifier. The classifier is capable of discerning between medically important spiders and harmless ones. The objective of this project is to create a tool that can help people identify dangerous spiders in their daily life, as well as promoting a culture of respect and interest towards arachnids. For this purpose, a Twitter bot (@arachno_bot) was created so people can tweet it photos of spiders and get a classification as a reply. The classifier achieved a test accuracy of 83.1% on medically important spiders and 70.4% on harmless spiders. Global accuracy may seem low for current state-of-the-art deep learning models but this is due to the few amounts of data available for this project, the dangerous to harmless spider ratio and distribution in the real world. Taking into account these problems, this accuracy is as best as it could get yet.**

I. INTRODUCTION

Spiders are usually perceived as dangerous and frightening creatures. This is mainly due to the small quantity of information available, as well as the large amount of misconceptions that exist about spiders, which are commonly exaggerated by pop culture and spread across social media. There are more than 40,000 species of spiders in the world, but contrary to what most people might think, less than 0.4% of them are capable of posing a threat to human health. Taking this into consideration, we can observe that the fearing surrounding spiders is mostly groundless. [1]

The objective of this project is to apply state-of-the-art computational technology along with machine learning, to help people overcome their phobia through exposition and distribution of accurate information, and hopefully transform fear and repulsion into fascination.

To achieve the best image-based classification possible, a Deep Learning algorithm called a Convolutional Neural Network (ConvNet/CNN) was used. A ConvNet can take in a set of input images, assign importance in the form of weights and biases to various features in the image, and be able to differentiate one from the other. [2] One advantage CNNs have over other algorithms is that the amount of image pre-processing required is much lower. While in earlier methods of image classification filters were hand-engineered, with enough training, CNNs have the ability to learn these filters.

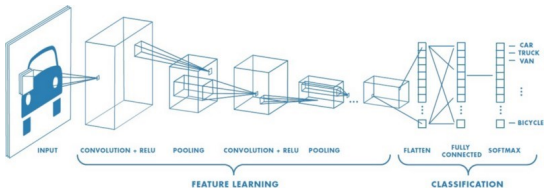

Fig. 1. ConvNet architecture with many convolutional layers. [3]

II. Understanding Convolutional Neural Networks

The architecture of Convolutional Neural Networks is analogous to that of the connectivity pattern of neurons in the human brain and was inspired by the organization of the visual cortex, where individual neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. A collection o such fields overlap to cover the entire visual area.

ConvNets are usually the way-to-go option in image-based tasks, where they greatly outperform regular dense neural networks. This is due to the fact that CNNs are able to successfully capture the spatial and temporal dependencies in an image through the application of filters. This architecture performs great fitting to images due to the reduction in the number of parameters involved and the reusability of weights, preventing vanishing gradients. Otherwise speaking, a Convolutional Neural Network can be trained to understand better the abstraction of an image.

Vanishing gradients is a problem that commonly exists in bigger networks. As more layers using certain activation functions are added to neural networks, the gradients of the loss function approach zero, making the network hard to train. Activation functions, like the sigmoid function, squishes a large input space into a space between 0 and 1. Therefore, a large change in the input of the sigmoid function will cause a small change in the output. When many layers are used, it can cause the gradient to be too small for training to work effectively, and when a big number of hidden layers use an activation like the sigmoid function, a lot of small derivatives are multiplied together. Thus, the gradient decreases exponentially as it propagates down to the initial layers. [19]

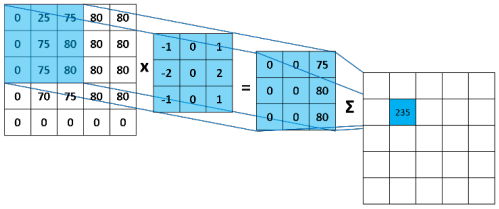

Fig. 2. A step in the convolution process [5]

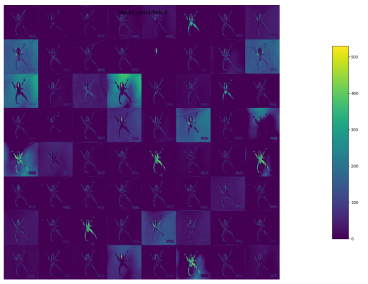

Fig. 3. Activation maps from the first layer of VGG19

Unlike regular neural networks, where the input is a vector, ConvNets take as inputs a multi-channeled image (usually 3 channels, one for every color in an RGB image). [4] Convolutional Layers, as the name implies, are in charge of applying convolutions to an input image or volume, often referred as tensor, and output a new volume called activation or feature map.

A. Convolutional Layer

A convolution is the process of adding each element of an image to its local neighbors, weighted by the kernel. This is performed by sliding a filter over the complete image while computing the dot product between the kernel and the corresponding chunks of the image. The purpose of a convolution is to extract features on the images, starting from simple characteristics such as corners, lines, and edges in the first layers, up to more abstract features such as eyes or faces in the deeper layers.

B. Kernel/Filter

The filter, also known as kernel, is a small matrix (usually 3x3x3 although it could be of any size) used to apply convolutions on images. Several kernels are applied simultaneously in a convolutional layer. Since filters are responsible of feature extraction, these elements are randomly initialized at the beginning of the training, and are updated through backpropagation on every iteration, thus decreasing the loss. On Figure 2 it can be appreciated how the kernel is applied to the input image.

C. Activation Maps

The result of a convolution is an activation map, also known as feature map, which is a representation of higher features from the previous layers. The feature maps are generated applying activation functions (such as ReLU, Sigmoid or Tanh) to the scalar results of a filter and describe how the given kernel matches the previous input and how much such input resembles the filter. Several activation maps are stacked together, and a pooling function is applied to reduce their dimensions before setting it as the next layer's input. [6] Figure 3 represents all the feature maps that were generated on the first convolutional layer of the VGG19 model.

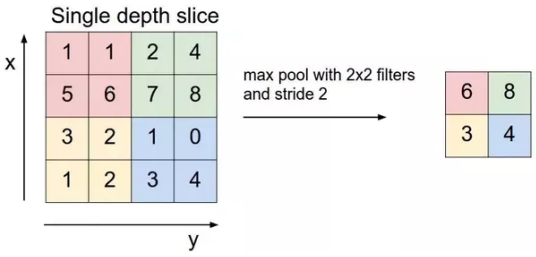

Fig. 4. Max Pooling Operation [3]

D. Pooling Layer

The next building block of ConvNets is the Pooling Layer. As mentioned before, its function is to progressively reduce the spatial size of the representation to reduce the number of parameters and computation requirements in the network. Pooling layers operate on each feature map independently. There are several pooling algorithms, but the most common approach is max pooling, represented in Figure 4, where the highest value in a sliding filter is taken as a representative of that region of the image.

E. Fully Connected Layer

Adding a fully connected layer at the end of the network is usually an easy way of learning non-linear combinations of the high-level features represented by the output of the convolutional layer. The fully connected layer is learning a non-linear function in that space. [2]

Before being able to use the outputs from the convolutional layer, the data must be flattened into a column vector so it can be inputted to the dense network. The flattened output is then fed through a feed-forward network and the error is backpropagated trough all layers on every iteration of training. Over a series of epochs, the network is able to distinguish features and patterns in images and classify them using SoftMax or Sigmoid classification.

F. Batch

When the size of the whole dataset is too large to pass it into a neural network, as in most of the times when working with ConvNets, you must divide the dataset into smaller parts, or batches, which are a subset of data from the original dataset. [7] Working with batches makes it

easier for the computer to train the network since it does not have to load hundreds or thousands of images into memory at the same time.

G. Loss Function

CNNs are trained using Stochastic Gradient Descent as their optimization algorithm. As a part of it, the error for the current state of the model must be estimated repeatedly. This requires the choice of an error function, also called loss function, that can be used to estimate the loss of the model so that the weights can be updated to reduce the error on the next evaluation. [8]

For image classification, there are two popular loss functions that can be used depending on the desired application of the network, either for binary classification or categorical classification. The functions are called binary cross-entropy and categorical cross-entropy, respectively. Cross-entropy will calculate a score that summarizes the average difference between the actual and predicted probability distributions. The main difference between both functions is that binary cross-entropy calculates loss for a set of only two classes (0 or 1), while categorical cross-entropy can calculate loss for as many classes as wanted.

$$J(Y) = \frac{1}{n}\sum_{i=1}^{n} -(Y_i log(\hat{Y}_i) + (1 - Y_i) log(1 - \hat{Y}_i)) \quad (1)$$

Binary cross-entropy equation. As it can be observed, the first of the terms cancels out if the expected value of Y is 0, and the other term cancels if the expected value is 1. This allows the function to converge smoothly.

H. Learning Rate

The learning rate is a hyperparameter of neural networks that controls how much the weights are adjusted with respect to the cost gradient. The lower the value, the slower we travel along the downward slope, lowering the chance of missing local minima, but may result in slow convergence. On the other hand, a higher learning rate may speed up the process, but it may overshoot the minimum, failing to converge or even diverge.

III. Development

For the development of this project, the arachnology expert Diego Barrales Alcalá, Master of Science by the National Autonomous University of Mexico, was asked to provide his help as an adviser regarding medically important spider species, as well as to help validating the project's usefulness and safety.

Spiders of medical importance belong to four genera: Atrax, commonly referred as Australian funnel-web spiders; Latrodectus, commonly known as black widows or brown widows; Loxosceles, also known as recluse spiders or violin spiders; and Phoneutria, sometimes called Brazilian wandering spiders. All spiders belonging to these genera pose a threat to human health, therefore, the network's priority must be to recognize them as accurately as possible.

A. Dataset

After a thorough search on the Internet for a dataset that could be useful for the project, it was found that there is no dataset available for this specific project. This represented a problem, since it meant the dataset had to be created from scratch, which implied downloading and classifying thousands of images of spiders by hand.

The site `https://www.inaturalist.org/`, which is a popular webpage where people upload animal observations for classification, was suggested by Diego as a source of spider images with their respective taxonomy classification already done. Luckily, the site is able to perform queries where the user is able to filter observations by species, genus, family, order, location, etc. and include fields to the search such as image URL and more. The query can be then downloaded as a .csv file with all the fields selected by the user.

Once the query is made, the next step is to download the images. To do this, a python script was created. The script can read the .csv file and iterate through every element in the table, get the URL and download the image into the current observation's corresponding directory.

The first problem that was found was the highly uneven number of images among species. The total number of spider images in the database reaches almost 475,000, however from that total, the number of images composing dangerous spiders is only 9,921. Even between medically important spiders, the difference in number of images is enormous. The number of images from genus Latrodectus reaches 8,680 (making up for 87.5% of all images), while genus Phoneutria has 239 elements (0.024%), and genus Atrax has only 9 elements (less than 0.001%).

Given the tiny number of images of medically important spiders from genera Atrax (Australia) and Phoneutria (Brazil), it was decided to circumscribe the scope of the project to only the identification of spiders in Mexico and North America, this was done in order to achieve better learning and greater precision. A reduced scope meant the amount of data for harmless spiders had decreased too, from almost 460,000 to less than 45,000, which represent a better proportion.

The compiled dataset for this project is composed of 43,180 images of spiders, of which 34,237 correspond to images of harmless spiders, and the remaining 8,943 correspond to spiders of medical importance. The dataset was divided in three sections, train, validation, and test, with a proportion of 70%, 15% and 15% respectively for both classes.

B. Techniques

This section describes the techniques used for the design and training of the neural network. These

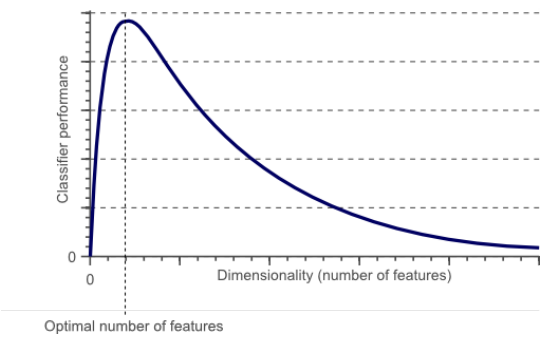

Fig. 5. Curse of Dimensionality [17]

techniques allow a fast training while reducing overfitting, thus producing the best results. Several combinations of these techniques were tested over many models to determine which set provides the optimal results for this application.

1) CUDA Processing: CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs). CUDA allows to dramatically speed up computing applications by harnessing the power of GPUs. The Nvidia CUDA Deep Neural Network library (cuDNN) is a GPU-accelerated library for deep neural networks. It provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization and activation layers. [9]

The training process was performed in a dedicated server with two Nvidia GTX 1080 graphics cards, with 2560 CUDA cores and 8 GB of VRAM each. The server runs CUDA Toolkit version 8.0, along with cuDNN 6.0 for Toolkit version 8.0 To be able to exploit the full power of the GPU while training, the server uses TensorFlow GPU version 1.2.1 and the Keras API version 2.0.5 which runs on Python 3.6.

2) Dimensionality Reduction: Dimensionality reduction is the process of reducing the dimension of the feature set. This is important because of a phenomenon called Curse of Dimensionality. The Curse of Dimensionality refers to all the problems that arise when working with data in the higher dimensions. [17] As it can be observed in Figure 5, as the number of features increase, the number of samples also increase proportionally, and more samples are required to have all combinations of feature values well represented in the sample.

As the number of features increases, the model becomes more complex, and the chances of overfitting increase, as the model gets increasingly dependent on the data it was trained on, resulting in poor performance on real data.

For this project, dimensionality reduction was handled by rescaling the input images to a smaller scale (150x150px or 200x200px). Thus, reducing disk space and improving computation time wile benefitting of the reduced chances of overfitting. However, due to the quality of some samples, the amount of detail that is lost (see Figures 6, 7 and 8) may negatively affect the performance of the network, mainly because in some images, the spider looks already small.

Fig. 6. Input image, no dimensionality reduction applied

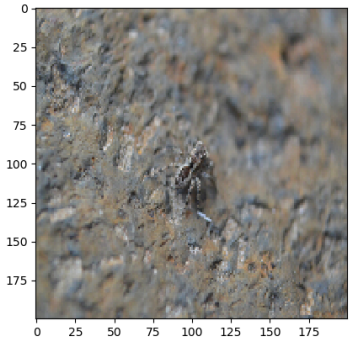

Fig. 7. Input image, reduced by a factor of 4.375 (200x200px) from the original

3) Image Processing: For the training of the model, all images were rescaled to a size of 200x200 pixels to reduce the number of parameters required to train the network. Furthermore, the pixel values in all color channels were normalized to fit in a range from 0 to 1 instead of 0 to 255 to improve the training speed of the model. Neural networks process inputs using small weight values, and inputs with large integer values can disrupt or slow down the learning process. As such it is good practice to normalize the pixel values so that each pixel value has a value between 0 and 1 to improve performance. [10]

4) Data Augmentation: Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset. [11] Training deep learning neural network models on more data can result in smarter

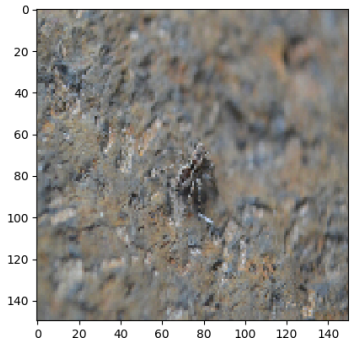

Fig. 8. Input image, reduced by a factor of 7.77 (150x150px) from the original

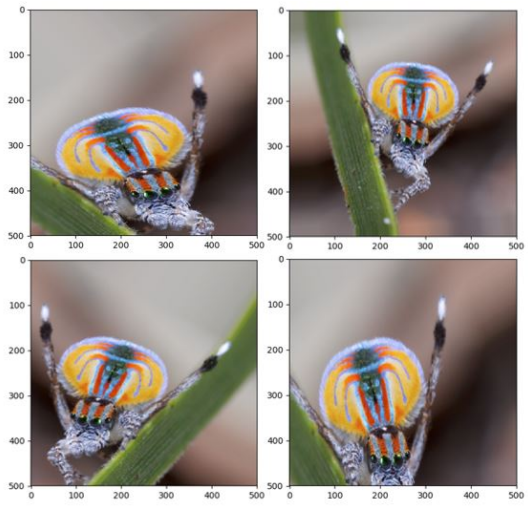

Fig. 9. Generation of spider images via random data augmentation

models, and data augmentation can generate variations of the images that allow to improve the ability of the model to generalize what they have learned to new images. Data augmentation generates more training data from existing training samples, by augmenting the samples via a number of random transformations that yield believable-looking images. [12]

In Keras, data augmentation can be done by configuring several random transformations to be performed on the images read by the `ImageDataGenerator`. For this project, the transformations that were applied are:

- Rotation
- Width Shift
- Height Shift
- Shear
- Zoom
- Horizontal Flip
- Vertical Flip

5) Transfer Learning: Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. Transfer learning is a popular approach in deep learning where pre-trained models are used as the starting point in computer vision tasks, given the vast computing and time resources required to develop and train deep learning models. However, it's important to note that transfer learning only works in deep learning if the model features learned from the first task are general. [13]

For this project, since the amount of training data is too small, it was determined that transfer learning would improve considerably the performance of the model. Thankfully, Keras includes a list of pretrained networks that can be easily reused for other application. These pretrained models were trained on large datasets, such as ImageNet dataset, which contains over 1.4 million images belonging to 1,000 different classes. These models perform well on almost any sort of image classification task because they are general enough, this means that the spatial hierarchy of features learned by the pretrained network can act as a generic model of the visual world, and hence its features can prove useful for many computer vision problems.

Three different models were tested in this project: VGG16, VGG19 and Inception V3. Their results will be discussed later in this document.

6) Dropout: In Machine Learning, dropout refers to ignoring units or neurons during the training phase of certain set of neurons which is chosen at random. These ignored neurons are not considered during a particular forward or backward propagation. [14] The purpose of adding dropout to a neural network is to reduce overfitting. A fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which limits the individual power of each neuron leading to overfitting of training data.

In machine learning, regularization is a way to prevent overfitting. Regularization reduces overfitting by adding a penalty to the loss function. Thus, the model is trained such that it does not learn interdependent set of features weights. Dropout is an approach to regularization in neural networks which helps reducing interdependent learning amongst the neurons.

7) Batch Normalization: As previously mentioned, normalizing the input data directly benefits the performance of the model. If the input layer is benefiting from normalized data, it may be a good idea to also normalize values in the hidden layers, since their weights are changing all the time.

Batch normalization reduces the amount by what the hidden unit values shift around, this is called covariance shift. To explain covariance shift, let's imagine this project's deep network. If we train our data on only certain type of spiders, and then try to apply it to different types of spiders, it is obvious that it won't do

TABLE I

CONFUSION MATRIX EXAMPLE LAYOUT

		Prediction: True	Prediction: False
Truth: True		True Positive	False Negative
Truth: False		False Positive	True Negative

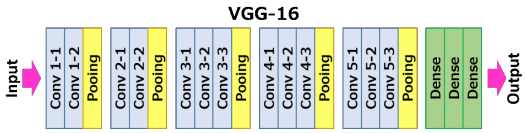

Fig. 10. Architecture of the VGG16 model [16]

well. The training set and the test set are both spiders, but they differ a little bit. In other words, the algorithm learned some X to Y mapping, and if the distribution of X changes, then it might be necessary to retrain the learning algorithm by trying to align the distribution of X with the distribution of Y. Batch normalization allows each layer of a network to learn by itself a little bit more independently of other layers. [15]

8) Confusion Matrix: In the field of machine learning, and specifically in statistical classification, a confusion matrix is a specific table layout that allows the visualization of the performance of an algorithm, typically a supervised learning one. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa). As the name implies, a confusion matrix allows to easily see if the system is confusing two classes. Similarly, it is possible to identify if the model falls into false positives or more false negatives, which may help to define the model's viability for real world applications.

IV. MODEL

Several model configurations and architectures were tested for this application. However, it will only be discussed those models with the highest accuracies. It's important to note that all architectures use preprocessed models as the convolutional base of the network due to the fact that none of the custom-made models presented an accuracy over 40%. The classifier of the pretrained models was removed in such a way that a custom dense classifier could be attached and trained from scratch. The three pretrained models used in this project were VGG16, VGG19 and Inception V3.

A. VGG16

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". [16] The model achieves a 92.7% accuracy in ImageNet dataset of over 1.4 million images belonging to 1000 classes.

The input to the first convolutional layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional layers, where the filters were used with a very small receptive field: 3 x 3 (which is the smallest size to capture the notion of left/right, up/down, center). The convolution stride is fixed to 1 pixel; the spatial padding of convolutional layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3 x 3 convolutional layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the convolutional layers (not all the convolutional layers are followed by max-pooling). Max-pooling is performed over a 2 x 2 pixel window, with stride 2.

Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels each, the third performs 1000-way classification and thus contains 1000 channels. The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks. All hidden layers are equipped with the rectification (ReLU) non-linearity. It is also noted that none of the networks (except for one) contain Local Response Normalization (LRN), such normalization does not improve the performance on the dataset but leads to increased memory consumption and computation time.

In this project, the VGG16-based model with highest accuracy reached a test accuracy of 70.2% on harmless spiders and 81.2% on medically important spiders.

B. VGG19

VGG19 follows the VGG16's architecture but adds one extra convolutional layer for each convolutional block, adding up to 19 layers. This architecture only improves accuracy by 1% compared to VGG16.

In this project, the VGG19-based model with highest accuracy reached a test accuracy of 70.4% on harmless spiders and 83.1% on medically important spiders. Which implies an improvement of 0.2% over harmless spiders and a 2.1% increase on medically important spiders.

C. Inception V3

Inception V3 is deep learning model developed by Google which improves previous architectures by optimizing procedures such as size of the kernels and receptive fields and modifying convolutions in order to optimize both size and accuracy. The model was trained on the ImageNet dataset, and is the proof that a more complex model with depth of 159 can increase accuracy up to 94.4% on the same dataset over similar conditions.

When testing the Inception V3 architecture as the convolutional base for this project, the training accuracy rose drastically leading to the thought that this model might be the best for this application, however, when reviewing individual predictions, it was discovered that the model always predicted one type of spider as the answer, either predicting the class harmless for both harmless and dangerous spiders, or vice versa depending

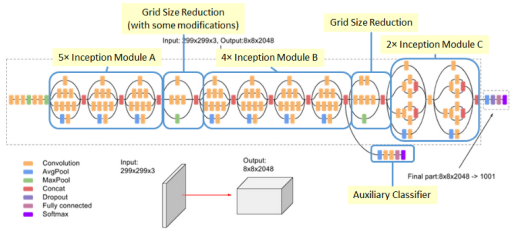

Fig. 11. Inception-v3 Architecture (Batch Norm and ReLU are used after Conv)

```
class_weights = {0: 3.829338,
                 1: 1.}

history = model.fit_generator(train_generator, class_weight=class_weights, steps_per_epoch=50,
epochs=50, validation_data=validation_generator, validation_steps=50)
```

Fig. 12. Weighted class setup in Keras

on how the parameters were tuned. No other result could be obtained.

D. Training

As it was mentioned before, the number of images of harmless spiders greatly outweighs the number of images of medically important spiders (by a factor of 3.82). This tends to create a bias in the model favoring the harmless spiders if trained with the complete dataset. This bias negatively influences the behavior of the model, and the quality of the predictions. Two attempts were made to dampen the bias caused by the imbalance in the classes.

The first attempt was to cut the amount of data of harmless spiders so that there were the same number of elements in both classes when training, however, this affected in a negative way the overall accuracy of the model, since plenty of available spiders are now unknown to the model.

The second attempt to fix the imbalance was modifying the weights of the classes when compiling the model in Keras. This is done by setting a dictionary where the keys are the class indices and the values are the desired class weights (see Figure 12). Assigning class weights means that in the loss function a weighted value is assigned to those instances. Hence, the loss becomes a weighted average, where the weight of each sample is specified in the dictionary. Sadly, this approach didn't work either, since the accuracy of the model stayed the same even when compensating the dangerous class with a weight of 3.82 in order to make up for the difference of elements. Any other set of weights ended up hindering the accuracy.

Due to the previous reasons, it was decided to even up the dataset by reducing the number of images of harmless spiders to the same number as medically important spiders.

An attempt to perform fine tuning by unfreezing the last convolutional block on all three pretrained models resulted in a lower accuracy compared to the case when all convolutional layers were frozen. When attempting the fine tuning the learning rate was decreased from $1x10^{-5}$ to $1x10^{-6}$ to prevent harsh changes in the weights that could lead to suboptimal learning. Also, the number of epochs was increased to compensate for the smaller learning rate, however none of this resulted beneficial overall.

V. Best Models

Now, the three models with the best performance will be listed and described. It is worth mentioning that these three models were selected from a total of 42 different architectures with unique parameters and configurations.

A. Model 1

TABLE II
Layer architecture of Model 1

VGG19	ImageNet Weights, 150x150px input image
Flatten	
Dense	256 nodes, ReLU activation
Dropout	50% Rate
Dense	64 nodes, ReLU activation
Dense	1 node, Sigmoid activation

1) Architecture: Adittionally to the layer layout present in Table II, this model makes use of an `ImageDataGenerator` as a data augmentation technique. Its parameters are as follows:

- Rotation range up to 40%
- Width Shift range up to 20%
- Height Shift range up to 20%
- Shear range up to 20%
- Zoom range up to 20%
- Horizontal Flip
- Fill mode nearest

The model was trained over 100 epochs in batches of 32 elements and 100 steps per epoch. Binary cross entropy was selected as the loss function and a learning rate of $1x10^{-5}$.

2) Result: The model reached a top accuracy of 85% when training and a validation accuracy of 81.1%. As it can be observed in Figure 13, the model presents some overfitting which could be reduced adding regularization techniques. In Figure 14 it can be observed how the training loss keeps decreasing while the validation loss stalls. Test accuracy dropped to 75.7%, this is yet another confirmation of overfitting.

TABLE III
Test results, Model 1

		Prediction	
		Medically Important	Harmless
Truth	Medically Important	0.406	0.094
	Harmless	0.149	0.351

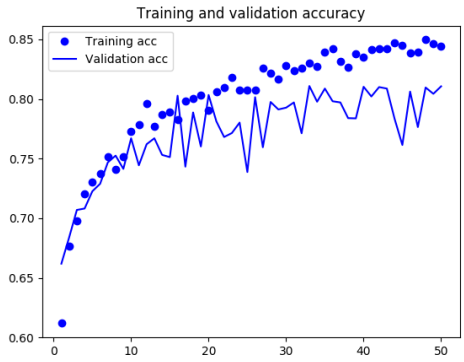

Fig. 13. Training and validation accuracy, Model 1

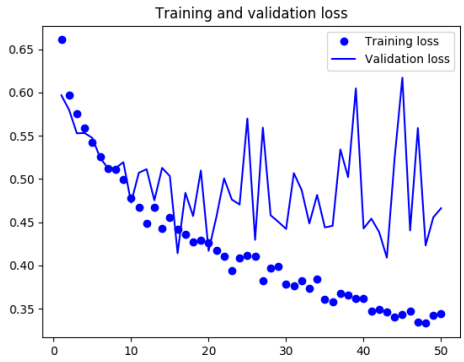

Fig. 14. Training and validation loss, Model 1

Based on the confusion matrix from Table III can be observed that this model might be good for deployment considering that the false-negative proportion is the smallest with a probability of only 0.094.

B. Model 2

TABLE IV
LAYER ARCHITECTURE OF MODEL 2

VGG19	ImageNet Weights, 200x200px input
Flatten	
Dense	2048 nodes, ReLU activation
Batch Normalization	
Dropout	40% Rate
Dense	2048 nodes, ReLU activation
Batch Normalization	
Dropout	40% Rate
Dense	1 node, Sigmoid activation

1) Architecture: Adittionally to the layer layout present in Table IV, this model makes use of an `ImageDataGenerator` as a data augmentation technique. Its parameters are as follows:

- Rotation range up to 40%
- Width Shift range up to 20%
- Height Shift range up to 20%
- Shear range up to 20%

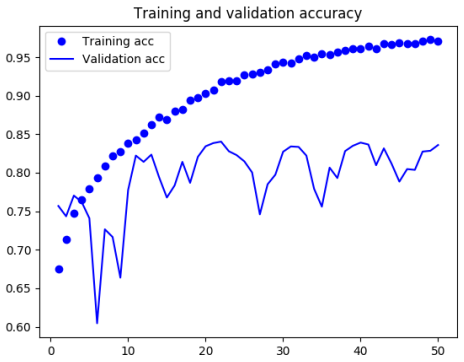

Fig. 15. Training and validation accuracy, Model 2

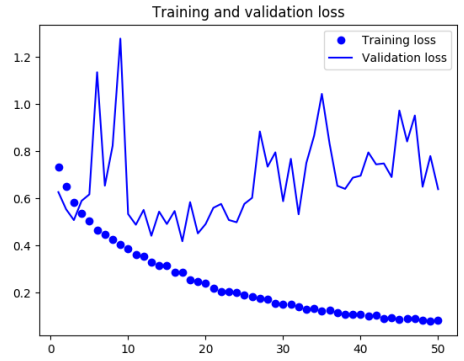

Fig. 16. Training and validation loss, Model 2

- Zoom range up to 20%
- Horizontal Flip enabled
- Fill mode nearest

The model was trained over 50 epochs in batches of 32 elements and 500 steps per epoch. Binary cross entropy was selected as the loss function and a learning rate of $1x10^{-5}$.

2) Result: The model reached a top accuracy of 96.3% when training and a validation accuracy of 84.5%. As it can be observed in Figure 15, the model is highly overfitted which could be fixed by adding regularization techniques such as a more aggressive dropout rate. In Figure 16 it can be observed how the training loss keeps decreasing while the validation loss increases, sign of overfitting. Test accuracy reached 76.75%, which represent an improvement of 1.05% over the previous model.

TABLE V
TEST RESULTS, MODEL 1

		Prediction	
		Medically Important	Harmless
Truth	Medically Important	0.4025	0.0975
	Harmless	0.135	0.365

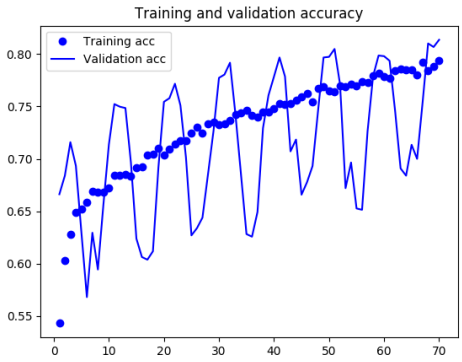

Fig. 17. Training and validation accuracy, Model 3

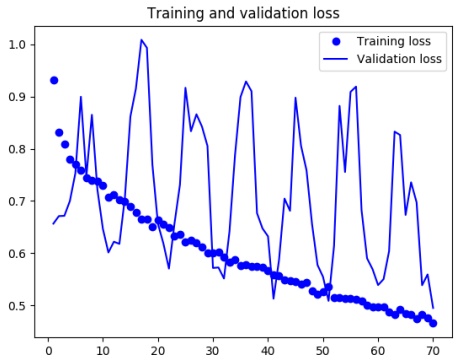

Fig. 18. Training and validation loss, Model 3

Based on the confusion matrix from Table V can be observed that this model might still be good for deployment considering that the false-negative proportion is the smallest with a probability of only 0.0975, nevertheless the previous model is better in that regard.

VI. Model 3

TABLE VI
Layer architecture of Model 3

VGG19	ImageNet Weights, 200x200px input
Batch Normalization	
Flatten	
Dense	2048 nodes, ReLU activation
Batch Normalization	
Dropout	40% Rate
Dense	2048 nodes, ReLU activation
Batch Normalization	
Dropout	40% Rate
Dense	1 node, Sigmoid activation

1) Architecture: Adittionally to the layer layout present in Table IV, this model makes use of an `ImageDataGenerator` as a data augmentation technique. Its parameters are as follows:

- Rotation range up to 40%
- Width Shift range up to 40%
- Height Shift range up to 40%
- Shear range up to 40%
- Zoom range up to 40%
- Horizontal Flip enabled
- Vertical Flip enabled
- Fill mode nearest

The model was trained over 70 epochs in batches of 32 elements and 700 steps per epoch. Binary cross entropy was selected as the loss function and a learning rate of $1x10^{-6}$.

2) Result: The model reached a top accuracy of 80.3% when training and a validation accuracy of 82.1%. As it can be observed in Figure 17, the model is not really overfitted, which means that the model fits well the data. In Figure 18 it can be observed how the training loss keeps decreasing while the validation follows the pace, proof that the model trained optimally. A somewhat strange oscillating behavior can be observed in the validation curves, this pattern is the result of turning off the data shuffling parameter in the validation generator. Doing this is not the optimal case, but since the average value followed the increasing and decreasing slopes for accuracy and loss respectively, it can be inferred that the model behaves correctly. This statement is backed up by the test accuracy. The test accuracy reached 76.75%, which represent an improvement of 0% over the previous model, however, now the percentages have shifted in favor of the medically important spiders.

TABLE VII
Test results, Model 3

		Prediction	
		Medically Important	Harmless
Truth	Medically Important	0.4155	0.0845
	Harmless	0.148	0.352

Based on the confusion matrix from Table VII can be observed that this model might be the best for deployment considering that the false-negative proportion is the smallest within the three models, with a probability of only 0.0845.

A. Statistical Significance

A test of statistical significance was performed in the two models with highest accuracy (Model 2 and Model 3) in order to determine if they indeed come from two different distributions. Statistical significance tests are an important tool to help to interpret the results from machine learning experiments. Additionally, the findings from these tools can help to better and more confidently present the experimental results and choose the right algorithms and configurations for a predictive modeling problem. [18]

The procedure to perform statistical significance for both models is as follows:

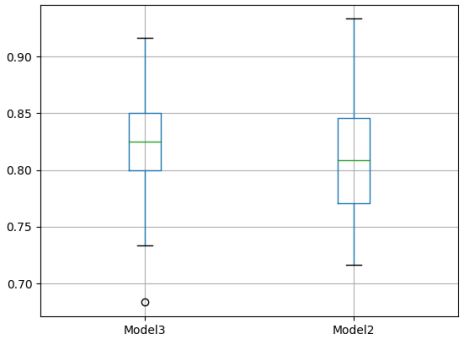

Fig. 19. Box plot of probability distributions of models 2 and 3

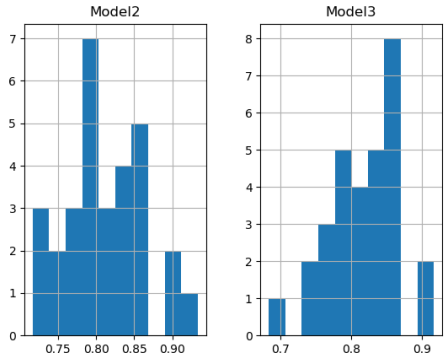

Fig. 20. Histogram of probability distributions of models 2 and 3

1) Run a set of predictions on both models and save the results on independent .csv files
2) Determine if the models present a normal distribution via the function `normaltest()` from the SciPy module.
3) Plot the histogram and box plot of both models to confirm the normality test.
4) If the models present a normal distribution, compare means for Gaussian results. This might be done using the Student t-test to see if the difference between the means of the two distributions is statistically significant or not. In SciPy, we can use the `ttest_ind()` function.
5) If the models do not present a normal distribution, it should be used an alternative statistical significance test called the Kolmogorov-Smirnov test. In SciPy, we can use the `ks_2samp()` function.
6) Observe the result and determine whether the samples come form the same distribution or not.

The normality test determined that both models present a normal distribution, which can be observed in the box plot and histogram from figures 19 and 20, respectively.

Since, the models presented a Gaussian distribution, it is necessary to compare them via the Student t-test to determine if their distributions come from the same population o from different ones.

The test returned a p-value of 0.5827, which is greater than the predefined alpha level of 0.05. Then it can be concluded with a confidence level of 95% that the samples are drawn from the same distributions.

B. Analysis of Results

As it can be observed from the results, none of the models achieved an accuracy greater than 77%, which compared to the current state of the art for deep learning is somewhat low. This may be due to the limited amount of data that was found, as well as the highly uneven number of images between classes. In addition to this, the dataset overall is very noisy, and the quality of the vast majority of the images leaves much to be desired. The statistical significance test concluded that the two models with greater accuracy come from the same population, so there is no real improvement from one to another.

VII. Application

A. Twitter Bot (@arachno_bot)

The development of this project was inspired by the desire to help the Twitter community of people who are interested in biology, entomology and more specifically arachnology. This community is supported by several accounts of Doctors, Academics and Professionals who are constantly sharing their knowledge online. Accounts such as @BoixRichter, @AracnoLoco, @Cataranea, @DrRichJP, @Tone_Killick, @Letirroja, @DRMegafauna, @DrBioblogo and the advisor of this project, @Arachno_cosas have change my perspective towards spiders over the course of a few months.

For this reason, it was decided to create a Twitter bot that is capable of receiving images of spiders attached to Twitter mentions. The image is then downloaded and fed to the convolutional neural network to get a prediction about the spider. Either if it belongs to the medically important class or to the harmless class. Finally, the classification is posted as a reply to the user, where the professional Arachnologist, Diego Barrales is tagged too so he can validate the output of the model and correct it if necessary.

The bot is also constantly looking for more data to train. It is well known that one of the reasons for the relatively low accuracy of the model is the lack of images of medically important spiders compared to the high number of elements of harmless spiders. That's why, the bot is programmed to constantly look for tweets on Diego's twitter account @Arachno_Cosas, that contain the hashtag #IM. The hashtag is included by Diego when he identifies a medically important spider. Sorting by this hashtag eliminates the necessity of manually identify and tag the images, since a professional has already done that.

B. Twitter API

Programming the bot requires access to the Twitter API and to setup a developer account, which can be done by entering to `https://developer.twitter.com/` and clicking the 'Apply' button and follow the on-screen instructions. Once the developer account is setup, the next step is to create an app. This can be done by clicking the 'Create an app' button located in the 'Apps' submenu and following the on-screen instructions just as before. If the app was created successfully, the menu 'Keys and tokens' will be now accessible. Here you can find your consumer key, consumer secret, access token and access token secret. All of them will be necessary to log to the Twitter account via OAuth authentication.

To access the Twitter API with python the Tweepy library was used. The code of the bot is quite simple. It consists of two classes, the class `TwitterStreamer` and the class `TwitterListener`. The `TwitterStreamer` is used to download twitter messages in real time. It is useful for obtaining a high volume of tweets, or for creating a live feed using a site stream or user stream. In this case, the streamer filters the tweets to only care for the ones tagging @arachno_bot. The `TwitterListener` is in charge of the interaction with the users. The `on_data()` method reads and downloads the image which is then fed to the trained model thanks to a couple of helper functions, such as `download_img()` and `predict()`. Once the model outputs a prediction, it replies to the user with the corresponding class using the function `reply()`. Finally, in the main module, the OAuth authentication takes place, the model is loaded and compiled, and the Twitter streamer is set up.

VIII. Conclusions

Even though the accuracy of the final model is not the best, it still serves as a good tool to identify potential medically important spiders when an expert is not around to do the identification. The advantage of this tool is that it has the ability to keep improving, by collecting the data that it is exposed to. Hopefully one day, it will be refined enough that it's predictions will be made with a greater confidence and higher accuracy.

Several problems were faced throughout the development of this project. Starting with the few amounts of research that has been done in the field, followed by the nonexistence of publicly available datasets of medically important/harmless spiders, and the fact the dataset had to be compiled from scratch. Additionally, the quality of the images was not the best, and the amount of noise in the dataset is a hindering factor. However, I strongly believe that dealing with such big amount of problems helped me learn more and settle my knowledge.

As future steps, it is desired to improve the functionality of the bot to interact better with the users. Similarly, more tests will be performed to try to improve the accuracy of the model as much as possible without incrementing the amount of data, so that, in the case of finding more data, the model will be able to improve even more.

IX. Appendix

The code, dataset, model, and all of the important files from this project can be found in `https://github.com/jcarlosangeles/SpiderIdentificationConvNet`

For contact, email to jcarlos.angelesc@gmail.com

X. Acknowledgment

To M.Sc. Diego Barrales Alcalá, for kindly sharing all his knowledge and experience in the field of arachnology in favor of the development of this project. His time, help and disposition are greatly appreciated.

To Dr. Benjamín Valdés Aguirre, for inspiring me and the rest of the students to really learn and get interested in Machine Learning, but most importantly, for encouraging us to face and solve real life problems with help of our knowledge, our skills and a bit of technology.

References

[1] 1395550283894582. (2017, September 23). Epoch vs Batch Size vs Iterations. Retrieved from https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9

[2] A Gentle Introduction to Transfer Learning for Deep Learning. (2018, December 12). Retrieved from https://machinelearningmastery.com/transfer-learning-for-deep-learning/

[3] Atlas7, Atlas7Atlas7 223127, FrobotFrobot 1, & HISIHISI 1311. (n.d.). What is the definition of a "feature map" (aka "activation map") in a convolutional neural network? Retrieved from https://stats.stackexchange.com/questions/291820/what-is-the-definition-of-a-feature-map-aka-activation-map-in-a-convolutio

[4] Budhiraja, A., & Budhiraja, A. (2016, December 15). Learning Less to Learn BetterâĂŁ-âĂŁDropout in (Deep) Machine learning. Retrieved from https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5

[5] Chi-Feng Wang, C. W. (2019, January 08). The Vanishing Gradient Problem. Retrieved from https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484

[6] Chollet, F. (2018). Deep learning with Python. Shelter Island: Manning.

[7] Convolutional Neural Networks - Basics. (n.d.). Retrieved from https://mlnotebook.github.io/post/CNN1/

[8] CUDA Zone. (2019, April 30). Retrieved from https://developer.nvidia.com/cuda-zone

[9] Doukkali, F., & Doukkali, F. (2017, October 20). Batch normalization in Neural Networks. Retrieved from https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c

[10] How to Choose Loss Functions When Training Deep Learning Neural Networks. (2019, April 19). Retrieved from https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/

[11] How to Configure Image Data Augmentation When Training Deep Learning Neural Networks. (2019, April 02). Retrieved from https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

[12] How to Manually Scale Image Pixel Data for Deep Learning. (2019, April 02). Retrieved from https://machinelearningmastery.com/how-to-manually-scale-image-pixel-data-for-deep-learning/

[13] How to Use Statistical Significance Tests to Interpret Machine Learning Results. (2018, May 21). Retrieved from https://machinelearningmastery.com/use-statistical-significance-tests-interpret-machine-learning-results/

[14] Mexican arachnologist is taking Twitter by storm. (2019, February 26). Retrieved from https://www.eluniversal.com.mx/english/mexican-arachnologist-taking-twitter-storm

[15] Pokharna, H. (2016, July 28). The best explanation of Convolutional Neural Networks on the Internet! Retrieved from https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8

[16] Prabhu, & Prabhu. (2018, March 04). Understanding of Convolutional Neural Network (CNN) - Deep Learning. Retrieved from https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

[17] Raj, J. T., & Raj, J. T. (2019, March 11). Dimensionality Reduction for Machine Learning. Retrieved from https://towardsdatascience.com/dimensionality-reduction-for-machine-learning-80a46c2ebb7e

[18] Saha, S., & Saha, S. (2018, December 15). A Comprehensive Guide to Convolutional Neural NetworksâĂŁ-âĂŁthe ELI5 way. Retrieved from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[19] VGG16 - Convolutional Network for Classification and Detection. (2018, November 21). Retrieved from https://neurohive.io/en/popular-networks/vgg16/