Animal Recognition Using Different Deep Convolutional Neural Networks

Bachelor's Project Thesis

Wijnand Karsens, s2420295, h.j.w.karsens@student.rug.nl,
Supervisors: Emmanuel Okafor & dr. Marco Wiering

Abstract: This thesis describes the use of four Deep Convolutional Neural Network techniques for training and classifying various kinds of animals. To achieve this aim, several convolutional neural network architectures are used. The AlexNet, LeNet, GoogLeNet, and FlickrStyle architectures are used to train and evaluate the classification performance on three animal datasets: The Wild-Anim dataset and two other novel species-specific animal datasets: the RUG-Goats and RUG-Snakes datasets. These datasets are considered challenging due to the limited number of images per class and less discriminatory features between similar kinds of animal species. The experimental activities on the deep neural network architectures on these datasets were carried out on the Caffe deep learning framework. The results show that GoogLeNet outperforms all other Convolutional Neural Network (CNN) methods. Details of the experimental settings and the results are discussed in this paper.

1 Introduction

1.1 Preamble

Mimicking the efficiency and robustness by which the human brain represents information has been a core challenge in artificial intelligence for decades. Recent neuroscience findings have provided insight into the principles governing information representation in the mammalian brain, leading to new ideas for designing systems that represent information. One of the key findings has been that the neo-cortex, which is associated with many cognitive abilities, does not explicitly pre-process sensory signals, but rather allows them to propagate through a complex hierarchy of modules (Lee and Mumford, 2003). These modules have the ability to learn and represent observations based on the regularities they exhibit (Jia, Shelhamer, Donahue, Karayev, Long, Girshick, Guadarrama, and Darrell, 2014). This discovery motivated the emergence of the subfield of machine learning which is called deep learning, which focuses on the computation of models for information representation that exhibits similar characteristics in comparison to the neo-cortex. Deep learning is a field of machine learning and computer vision that uses a neural network with several layers to learn and extract useful features from an image. The field has gained popularity rapidly in recent years. The performance of conventional feature extractors with the use of local feature descriptors such as SIFT (Lowe, 1999), the histogram of oriented gradient (HOG) (Dalal and Triggs, 2005) and Bag of Visual Words (BOW) (Csurka, Dance, Fan, Willamowski, and Bray, 2004) have become outdated with respect to recognizing objects. The emergence of deep neural network architectures such as AlexNet (Krizhevsky, Sutskever, and Hinton, 2012), GoogleNet (Szegedy, Liu, Jia, Sermanet, Reed, Anguelov, Erhan, Vanhoucke, and Rabinovich, 2015) and Residual Network (ResNet) (He, Zhang, Ren, and Sun, 2016) have aided in mitigating this outdated vision by presenting a robust and vivid vision system.

The advances in deep learning have resulted in the exploration of new ideas and algorithms in the following areas: improvements in network architectures, the quality of image and object recognition and feature extraction. The progress of deep learning is not limited to the development of algorithms for recognition or evaluation of performance but has encouraged cloud computing with the use of the parallel core of processors as well as the integration of graphical processing units (GPU). This hardware has accelerated the processing activity in machine learning and computer vision community.

Convolutional Neural Networks (CNN) are biologically-inspired variants of multi-layer perceptrons (MLP). It is a family of multi-layer neural networks, particularly designed for use in two- or three-dimensional data, such as images and videos. CNNs resemble time-delay neural networks (TDNN), which reduce learning computation requirements by sharing weights in a temporal dimension and are used for speech and time-series processing. CNNs are the first truly successful deep learning approach where many layers of a hierarchy are successfully trained in a robust manner. A CNN is a choice of topology or architecture that leverages spatial relationships to reduce the number of parameters which must be learned and thus improves upon general feed-forward back propagation training. CNNs were proposed as a deep learning framework that is motivated by minimal data preprocessing requirements. In CNNs, small portions of the image (dubbed a local receptive field) are treated as inputs to the lowest layer of the hierarchical structure. Information generally propagates through the different layers of the network whereby at each layer digital filtering is applied in order to obtain salient features of the data observed. The method provides a level of invariance to shift, scale and rotation as the local receptive field allows the neuron or processing unit to access elementary features such as oriented edges or corners (Jia et al., 2014).

1.2 Related Work

In recent times, the specific recognition task is becoming an emerging and interesting area of research. Some work has been examined with the use of unsupervised grid alignment (Gavves, Fernando, Snoek, Smeulders, and Tuytelaars, 2015), a method that outperforms the hyper-class augmented regularized CNN (Xie, Yang, Wang, and Lin, 2015) and other CNN methods when evaluated on the Stanford-Dog dataset. The use of the CUB-200 dataset collected by (Welinder, Branson, Mita, Wah, Schroff, Belongie, and Perona, 2010) has been greatly used for species-specific recognition on birds. In our study, we collected two novel species-specific datasets: the RUG-Goats and the RUG-Snakes dataset. In this project, four deep convolutional neural networks: AlexNet, LeNet, FlickrNet and GoogleNet will be used in evaluating and classifying different animal images on three datasets; Wild-Anim, RUG-Goats and RUG-Snakes. Also, the study tries to examine the robustness of these deep CNN architectures on species-specific recognition on different breeds of Goat and Snake images from the novel datasets as well as already existing wild-animal images in the Wild-Anim dataset (Okafor, Pawara, Karaaba, Surinta, Codreanu, Schomaker, and Wiering, 2016). This project tries to answer two questions:

1. Which deep learning model yields the highest accuracy for classification on novel challenging and existing animal datasets?
2. What deep learning architecture is most robust in determining species-specific recognition?

1.3 Thesis Outline

The thesis is organized in the following way. This section briefly introduced deep learning, the motivation of the project and the aim and project outline. Section two describes the various kinds of deep learning architectures and describes the deep learning process more clearly. Section three describes the animal datasets that are used in the experiments. The performance evaluations of the deep learning architectures are discussed in section four. Section five concludes the project and recommends areas for further work.

2 Material and Methods

This chapter entails a brief discussion on the Caffe framework, the deep learning process and different architectures used.

2.1 Caffe Deep Learning Framework

Caffe is a fully open-source framework that affords clear access to deep architectures. The code is written in clean, efficient C++, with CUDA used for GPU computation. Caffe provides a complete toolkit for training, testing, fine-tuning and deploying models, with well-documented examples for all of these tasks (Jia et al., 2014). There exist alternative platforms that can be compared to Caffe. These platforms include: Theano (Team, Al-Rfou, Alain, Almahairi, Angermueller, Bahdanau, Ballas, Bastien, Bayer, Belikov, et al., 2016) and MatConvNet (Vedaldi and Lenc, 2015). Based on preliminary experiments, Caffe is very user-friendly, which informed the choice of this framework. The Caffe inventors stated that the platform is well-suited for research use, due to the careful modularity of the code and the clean separation of network definition from actual implementation. They provide off-the-shelf reference models for visual tasks for academic use.

2.2 Deep Learning Process

In this subsection, the processes involved in understanding how the Deep CNN works will be discussed.

Convolution Process: According to (Bengio and Courville, 2016), convolutional neural networks are neural networks that employ convolution in substitution of matrix multiplication in at least one of their layers. It is expected that each layer of the neural network should have the capability to extract feature maps. These feature maps can be represented using $H^l(x,y,v)$ for neuron v from each layer l of the convolutional layer (Okafor et al., 2016) and can be computed as:

$$H^l(x,y,v) = B^l_v + X^{l-1}(x,y,c) \otimes K^l_v(x,y,c) \quad (2.1)$$

The input to the convolutional neural network can be represented as a tensor X^{l-1} from the previous layer $l-1$ with elements $X(x,y,c)$, this represents the value of the input unit within channel c at row x and column y. The input to the convolution is convolved with the tensor kernel using a bank of filters K^l_v for the current layer l with the same elements as in X. Each convolved feature map in a given layer gets its corresponding bias B^l_v added.

Detector Process: This process requires the use of a non-linear activation function such as the Rectified Linear Unit (ReLU) (Krizhevsky et al., 2012) to calculate the activations of all convolved extracted features. The ReLU is recurrently assigned to the output of each hidden unit in a convolutional layer and the fully connected layers. The output of the ReLU $R^l(x,y,v)$ is computed using the expression:

$$R^l(x,y,v) = max(0, H^l(x,y,v)) \quad (2.2)$$

Normalization Process: In this process, local response normalization is employed for normalizing the output of the ReLU (Krizhevsky et al., 2012; Vedaldi and Lenc, 2015). The role of the local response normalization is to achieve better generalization. The local response normalization (Stutz, 2014) can be computed as:

$$\mathrm{N}^l(x,y,v) = R^l(x,y,v)\left(\lambda + \alpha \textstyle\sum_{j\in m^l} R^l(x,y,j))^2\right)^{-\beta} \quad (2.3)$$

where $N^l(x,y,v)$ computes the response of the normalized activity from the ReLU output $R^l(x,y,v)$. This is done by multiplying the output with an inverse sum of squared ReLU outputs added to an offset λ within a layer l within a specific region of the feature map m^l. We employed the same hyper-parameter setting as in (Krizhevsky et al., 2012) with the following variable constants: $\lambda = 2$, $\alpha = 10^{-4}$ and $\beta = 0.75$.

Spatial Pooling Process: In this process, two spatial pooling approaches are used in the four CNN architectures employed in the experiments.

1. Max-Pooling: The max-pooling operator calculates the maximum response of each feature channel achieved from the normalized output. A modified max-pooling operator from (Marc'Aurelio, 2014) can be computed using the expression:

   $$S^l(\bar{x},\bar{y},v) = max_{x,y\in M(\bar{x},\bar{y},l)} N^l(x,y,v) \quad (2.4)$$

   Where $(\bar{x},\bar{y})$ is the mean image position of (x,y), $M(\bar{x},\bar{y},l)$ denotes the size of the pooling layer, and $S^l(x,y,v)$ is the result of the spatial pooling of the convolutional layers.

2. Average-Pooling: The average-pooling operator calculates the mean response of each feature channel achieved from the normalized output. A modified average-pooling operator from (Marc'Aurelio, 2014) can be computed using the expression:

   $$S^l(\bar{x},\bar{y},v) = \frac{\sum_{x,y\in M(\bar{x},\bar{y},l)} N^l(x,y,v)}{|M(\bar{x},\bar{y},l)|} \quad (2.5)$$

Regularization Process: In order to mitigate overfitting in the network, the use of the dropout (Krizhevsky et al., 2012) regularization scheme is applied to the output of the spatial pooling layer. For instance, the AlexNet and FlickrStyle architectures use a dropout of 0.5, and the GoogleNet architecture use dropouts of 0.7, 0.7, and 0.4 in three sets of classifiers.

Classification Process In this process, the probability of the class labels from the output of the fully connected layer are calculated using the softmax activation function. Recurrently, the classification process uses the top-K classification error for predicting a label. The top-K loss is zero if class i is within the top K ranked scores (Vedaldi and Lenc, 2015):

$$L(y,i) = 0[|\{k : y_k \geq y_i\}| \leq K] \tag{2.6}$$

The top-K loss is one for an example, if

$$L(y,i) = 1[|\{k : y_k \geq y_i\}| > K] \tag{2.7}$$

Where y_i are the final outputs of the CNN and i is the target class. We report results of the top-1 error accuracy in all our experiments. The softmax activation function (Bengio and Courville, 2016) calculates the probabilities of the multi-class labels using the sum of weighted inputs from the previous layer and is used in the learning process:

$$y_i = \frac{\exp(x_i)}{\sum_{i=1}^{I} exp(x_i)} \tag{2.8}$$

where y_i is the output of the softmax activation function for class i, x_i is the summed input of output unit i in the final output layer of the fully connected network and I is the total number of classes.

2.3 Deep Learning Methods

In this subsection, four deep learning architectures will be discussed.

LeNet Architecture

The LeNet Architecture (LeCun, Boser, Denker, Henderson, Howard, Hubbard, and Jackel, 1989) consists of three major layers: input layer, hidden layers and output layer. In the hidden layers, these layers consist of two pooling layers and three convolutional layers which are described with the layers H1, H2 and H3 respectively. The output layer can be refered to as the fully connected layer which contains output neurons that describe the various classes of animals under study. A block diagram illustration of the LeNet architecture is shown in Figure 2.1. The experimental parameters used are shown in Table 2.1 and the test iterations for the RUG-Goat, Snake, and Wild-Anim datasets are set to 78, 41, and 80 respectively.

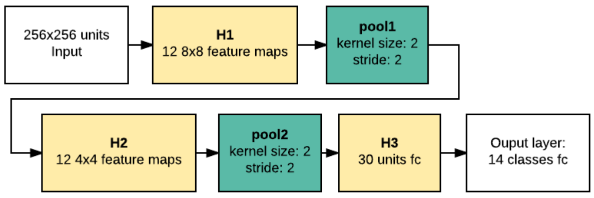


Figure 2.1: Block diagram illustration of the LeNet architecture on the RUG-Goat dataset

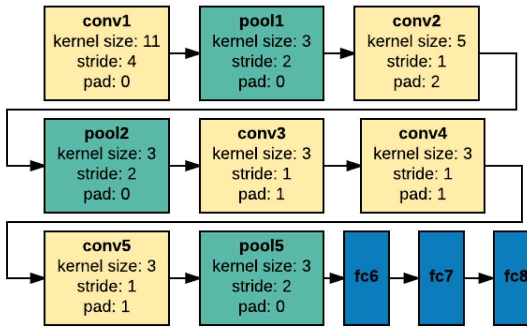
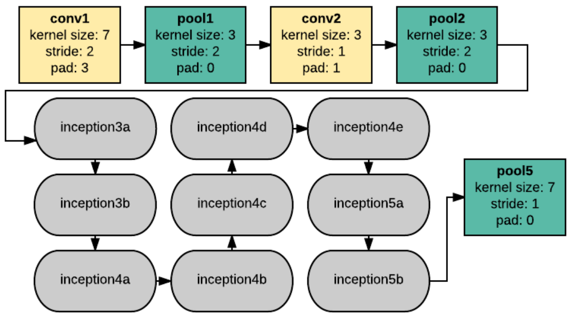


Figure 2.2: Block diagram of the AlexNet architecture adopted from (Krizhevsky et al., 2012) and (Shin et al., 2016)

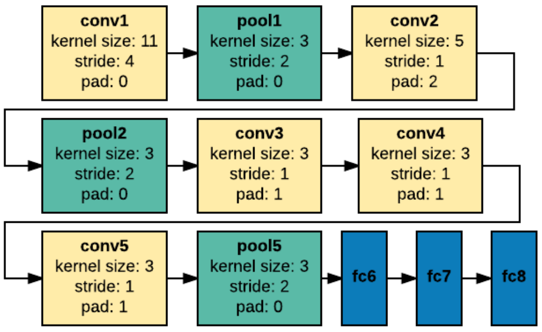
AlexNet Architecture

AlexNet contains eight layers with weights; the layers one to five are convolutional layers with three pooling layers and the remaining three are fully-connected. The output of the last fully-connected layer is fed to an I-way softmax where I is the number of classes of the given dataset (Krizhevsky et al., 2012). This network maximizes the multi-nominal logistic regression objective, which is equivalent to maximizing the average log-probability of the correct label under the prediction distribution. The first convolutional layer filters the $224 \times 224 \times 3$ input image with



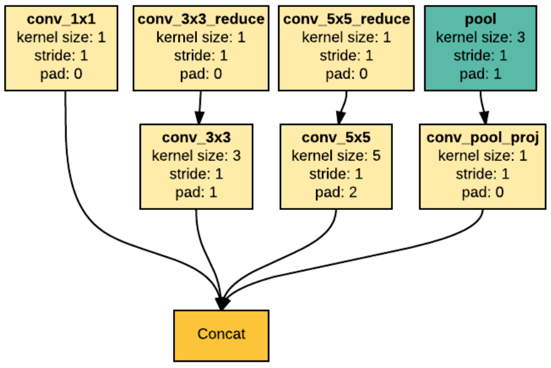
conv1 kernel size: 7 stride: 2 pad: 3
pool1 kernel size: 3 stride: 2 pad: 0
conv2 kernel size: 3 stride: 1 pad: 1
pool2 kernel size: 3 stride: 2 pad: 0
inception3a
inception3b
inception4a
inception4b
inception4c
inception4d
inception4e
inception5a
inception5b
pool5 kernel size: 7 stride: 1 pad: 0

Figure 2.3: Block diagram of the GoogLeNet architecture adopted from (Szegedy et al., 2015) and (Shin et al., 2016)



conv1 kernel size: 11 stride: 4 pad: 0
pool1 kernel size: 3 stride: 2 pad: 0
conv2 kernel size: 5 stride: 1 pad: 2
pool2 kernel size: 3 stride: 2 pad: 0
conv3 kernel size: 3 stride: 1 pad: 1
conv4 kernel size: 3 stride: 1 pad: 1
conv5 kernel size: 3 stride: 1 pad: 1
pool5 kernel size: 3 stride: 2 pad: 0
fc6
fc7
fc8

Figure 2.5: Block diagram of the FlickerStyle architecture adopted from (Shin et al., 2016)



conv_1x1 kernel size: 1 stride: 1 pad: 0
conv_3x3_reduce kernel size: 1 stride: 1 pad: 0
conv_5x5_reduce kernel size: 1 stride: 1 pad: 0
pool kernel size: 3 stride: 1 pad: 1
conv_3x3 kernel size: 3 stride: 1 pad: 1
conv_5x5 kernel size: 5 stride: 1 pad: 2
conv_pool_proj kernel size: 1 stride: 1 pad: 0
Concat

Figure 2.4: Block diagram of inception layer in the GoogleNet architecture adopted from (Szegedy et al., 2015) and (Shin et al., 2016)

96 kernels of size 11 × 11 × 3 with a stride of 4 pixels (this is the distance between the receptive field centres of neighbouring neurons in a kernel map). The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with 256 kernels of size $5 \times 5 \times 48$. The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size 3 × 3 × 256 connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 192$, and the fifth convolutional layer has 256 kernels of size 3 × 3 × 192. The fully-connected layers have 4096 neurons each. This architecture is shown in Figure 2.2.

In this study, the last fully connected layer is represented with the number of classes within each of the datasets under examination. To train this architecture Caffe uses a solver.prototxt and train-val files which contain the network setting and configuration files. Some changes were made in the files with respect to our datasets.

The experimental parameters used are shown in Table 2.1 and the test iterations for the RUG-Goat, Snake, and Wild-Anim datasets are set to 78, 41, and 80 respectively. In our experiment we employed both scratch AlexNet (S-AlexNet) and Pre-trained Alexnet (Pre-AlexNet) architectures in carrying out experiments.

GoogleNet Architecture

The GoogLeNet network is 22 layers deep (Szegedy et al., 2015). It consists of two outer convolutional layers, three pooling layers and nine inception layers. In each of the inception layers there are six inner convolutional layers, and one pooling layer. Also, the architecture has three softmax classifiers. A block diagram illustrating this architecture is shown in Figure 2.3. Figure 2.4 shows one of the inception layers. The experimental settings used are shown in Table 2.1. In our experiment we employed both scratch GoogleNet (S-GoogleNet) and Pre-trained GoogleNet (Pre-GoogleNet) architectures.

FlickrStyle Architecture

The FlickrStyle network, shown in Figure 2.5, has the following layers and dimensions. It consists of 5 convolutional layers and 3 fully-connected layers of which the last has a dimension of $1 \times 1 \times I$ where I is the amount of classes of the dataset. The experimental settings used are shown in Table 2.1. Only the Pre-trained FlickrStyle (Pre-FlickrStyle) was used in carrying out experiments because in preliminary experiments the scratch version didn't perform well.

Table 2.1: Experimental Parameters of The Deep CNN Architectures

	AlexNet	**GoogleNet**	**LeNet**	**FlickrStyle**
Learning Rate	0.001	0.001	0.001	0.001
Gamma	0.1	0.1	0.001	0.5
Step Size	3000	3000	10000	2000
Maximum Iteration	10000	10000	10000	10000
Momentum	0.9	0.9	-	0.9
Weight Decay	0.0005	0.0002	-	0.0005
Train Batch Size	10	15	10	50
Test Batch Size	10	10	10	10
Snapshots	2000	2000	2000	2000

3 Animal dataset and Preprocessing

In this section, 2 things will be discussed: the actual datasets used for the experiments and the preprocessing steps.

3.1 Wild-Anim dataset

The Wild-Anim dataset already existed and was described in (Okafor et al., 2016). This dataset consists of a total of 5000 images with 5 classes: lion, bear, wolf, elephant and leopard. The dataset used for the experiments is partitioned into the ratio 0.64 : 0.2 : 0.16 for the training set, the testing set and the validation set respectively. This implies that there exist 3200, 1000, 800 images within the training set, testing set, and validation set respectively. To do this, a python script was used to create the train.txt, val.txt (validation file) and test.txt. Then, the created '*.txt' files and the raw images were used to create the lmdb dataset for the partitioned sets as well as the mean-image prototxt file from the lmdb training set which were scaled to 256×256 pixels. This introduces slight anamorphic distortions. After the databases were created, an additional mean-image prototxt file was created using a python script. A sample of the dataset is shown in Figure 3.1.

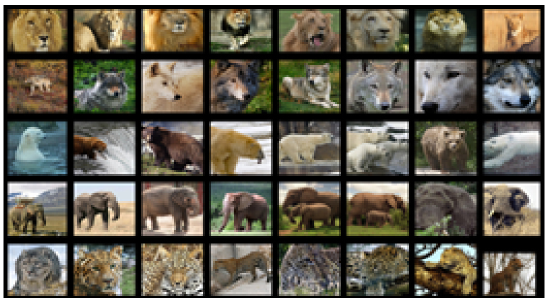


Figure 3.1: A sample of the images in the Wild-Anim dataset (Okafor et al., 2016)

3.2 RUG-Goat dataset

The RUG-Goat dataset was collected by us. This dataset consists of a total of 3711 images with 14 classes: alpine goat, anglo-nubian goat, angora goat, boer goat, golden guernsey goat, kiko goat, lamancha goat, nigerian dwarf goat, oberhasli goat, pygmy goat, pygora goat, saanen goat, spanish goat and toggenburg goat. The dataset used for the experiments is partitioned into the ratio 0.487 : 0.302 : 0.211 for the training set, the testing set and the validation set respectively. This implies that there exist 1807, 1121, 783 images within the training set, testing set, and validation set respectively. To do this, the same steps were taken as for the Wild-Anim dataset. After the databases were created, the mean-image prototxt file was created by a python script. A sample of the dataset is shown in Figure 3.2.

3.3 RUG-Snake dataset

The RUG-Snake dataset was created by us. This dataset consists of a total of 4561 images with 5 classes: adder, boa constrictor, green tree python, rattlesnake and tree viper. The dataset used for the experiments is partitioned into the ratio 0,809 : 0.1 : 0.091 for the training set, the testing set and the validation set respectively. This implies that their exist 3690, 456, 415 images within the training set , testing set, and validation set respectively. To do this, the same steps were taken as for the Wild-Anim dataset. After the databases

Figure 3.2: A sample of the images in the RUG-Goat dataset

were created, the image mean was created using a python script. A sample of the dataset is shown in Figure 3.3.

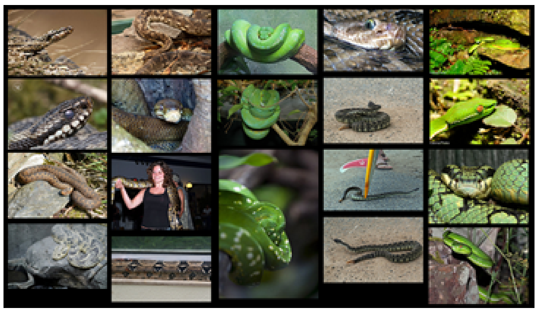


Figure 3.3: A sample of the images in the RUG-Snake dataset

4 Experimental Results and Discussion

The test performances of the CNN techniques on the three datasets were based on two experimental runs and the means of these performances are shown in Table 4.1.

Results for the RUG-Goat dataset

As can be seen from Table 4.1, the Pre-GoogleNet outperforms all other CNN methods with a loss rate of 32.2%. Next to it, is the Pre-FlickrStyle which obtained 39.7%, followed by the Pre-AlexNet which recorded a loss-rate of 47.7%. All the pre-trained architectures

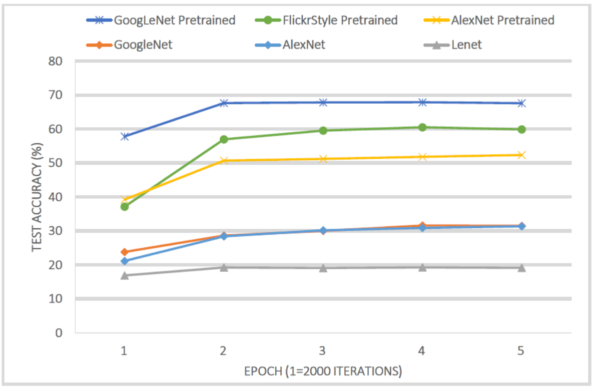


Figure 4.1: Learning curves of the CNN architectures on the RUG-Goat dataset for the first 10000 iterations

outperform the scratch versions. Moreover, the loss-rate recorded by the scratch CNN techniques are as follows: the S-GoogleNet obtained a loss rate of 68.4%, S-AlexNet obtained a loss rate of 68.6% and the LeNet recorded the worst performance with a loss rate of 80.7%. This result can further be improved by increasing the number of iterations as well as increasing the numbers of image examples per class. Based on the results obtained so far, we can deduce that this dataset is relatively challenging and therefore becomes a new benchmark dataset for exploring species-specific recognition task by applying new machine learning techniques and deep architectures. A visual illustration of the learning curves is shown in Figure 4.1.

Table 4.1: Test accuracies of the CNN architectures on the three datasets

	RUG-Goat	RUG-Snake	Wild-Anim
S-Alexnet	31.4	74.8	90.8
S-GoogLeNet	31.6	81.1	93.0
LeNet	19.3	67.1	71.5
Pre-AlexNet	52.3	84.7	98.3
Pre-GoogLeNet	**67.9**	**94.1**	**100**
Pre-FlickrStyle	60.3	81.6	99.2

Results for the RUG-Snake dataset

As can be seen from Table 4.1, Pre-GoogLeNet outperforms other CNN methods, with a loss-rate of 5.9%, followed by Pre-AlexNet with a loss-rate of

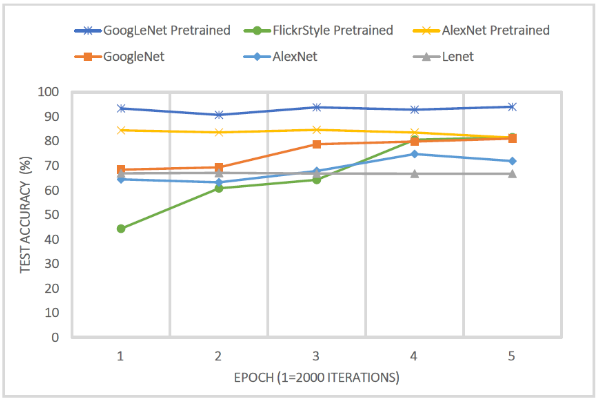


Figure 4.2: Learning curves of the CNN architectures on the RUG-Snake dataset for the first 10000 iterations

15.4% and next is the Pre-FlickrStyle architecture with a loss-rate of 18.4%. All the earlier techniques outperform all scratch versions of the CNNs. The S-GoogLeNet obtains 18.9% followed by S-AlexNet with a loss rate of 25.2%. Again, the LeNet architecture obtains the worst performance on this dataset. The higher accuracies on this dataset can be explained because there exist more images per class and contains less classes relative to that earlier discussed in the RUG-Goat dataset which seems quite challenging. The learning curves of the CNN performance are shown in Figure 4.2.

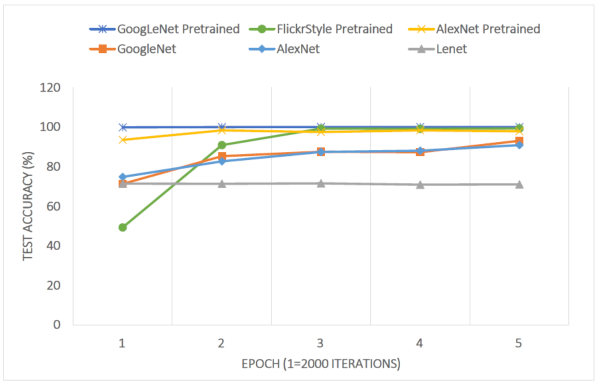


Figure 4.3: Learning curves of the CNN architectures on the Wild-Anim dataset for the first 10000 iterations

Results for the Wild-Anim dataset

The results in Table 4.1 also show that Pre-GoogleNet outperforms all other CNN methods studied and has a loss rate of 0%, followed by Pre-FlickrStyle with a loss-rate of 0.8% and next to this performance is the Pre-AlexNet architecture with a loss rate of 1.7%. These results can be debated on due to the high performances. These performances are no surprise because there exist some classes in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) dataset (Krizhevsky et al., 2012) that are present in the Wild-Anim dataset but with different images. The pre-trained networks of all the used CNN architectures were trained on the ILSVRC dataset and this contributed significantly to the performance. So the fairest results are based on the scratch versions of the CNN architectures. The S-GoogleNet obtained a loss-rate of 7.0% that outperforms than other scratch versions of the CNN methods. The next with respect to its performance, is the S-AlexNet with a loss-rate of 9.2%. Again the worst performance on this dataset is the LeNet architecture with a loss-rate of 28.5%. Based on the graphical illustration of the performance using the CNN methods on this dataset as can seen in Figure 4.3, a deduction can be drawn that increasing the number of maximum iterations significantly improves the classification performance.

Table 4.2: Validation Performance Evaluation of the Deep CNN Architectures on the Three datasets

	RUG-Goat	RUG-Snake	Wild-Anim
S-AlexNet	30.3	69.5	91.1
S-GoogLeNet	35.2	74.4	95.4
LeNet	20.1	59.0	75.4
Pre-AlexNet	50.2	79.0	97.4
Pre-GoogLeNet	62.8	**92.4**	**100**
Pre-FlickrStyle	**64.1**	79.5	99.0

A report on the validation performance of the CNN methods on the three datasets is shown in Table 4.2. It can be seen that the results are fairly consistent with the test results earlier reported in Table 4.1. Only for the RUG-Goat dataset the Pre-FlickrStyle yields a higher performance than the Pre-GoogLeNet.

5 Conclusion

This paper has described the use of four different CNN architectures and some scratch and pre-trained versions of the CNN in accessing the performance on three datasets. The new datasets (RUG-Goat and RUG-Snake) collected for this study are challenging due to less discriminatory features that exist between similar species of animals. Our study was able to answer the questions which Deep CNN method performs best and is most robust in species-specific recognition. The GoogLeNet architecture in both scratch and pre-trained versions showed an outstanding performance relative to other CNN techniques. Our study also shows that the LeNet architecture presents the worst performance relative to all other CNN methods. Further work will involve the use of deeper CNN architectures to examine if it can improve the performance on the new datasets.

References

Ian Goodfellow Yoshua Bengio and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016. URL `http://www.deeplearningbook.org`.

Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In *Computer Vision (ECCV), 8th European Conference on*, pages 1–22, 2004.

Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition (CVPR). IEEE Computer Society Conference on*, volume 1, pages 886–893, 2005.

Efstratios Gavves, Basura Fernando, Cees GM Snoek, Arnold WM Smeulders, and Tinne Tuytelaars. Local alignments for fine-grained categorization. *International Journal of Computer Vision*, 111(2):191–212, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *arXiv preprint arXiv:1603.05027*, 2016.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4): 541–551, 1989.

Tai Sing Lee and David Mumford. Hierarchical bayesian inference in the visual cortex. *JOSA A*, 20(7):1434–1448, 2003.

David G Lowe. Object recognition from local scale-invariant features. In *Computer vision. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157, 1999.

Ranzato Marc'Aurelio. Large-scale visual recognition, part iv: Deep learning. pages 68–69, 2014.

Emmanuel Okafor, Pornntiwa Pawara, Faik Karaaba, Olarik Surinta, Valeriu Codreanu, Lambert Schomaker, and Marco Wiering. Comparative study between deep learning and bag of visual words for wild-animal recognition. pages 1–8. IEEE, Submitted to SSCI (under review), 2016.

Hoo-Chang Shin, Holger R Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura, and Ronald M Summers. Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging*, 35(5): 1285–1298, 2016.

David Stutz. Understanding convolutional neural networks. In *Fakultät für Mathematik,* *Informatik und Naturwissenschaften Lehr- und Forschungsgebiet Informatik VIII Computer Vision*, pages 1–23, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

The Theano Development Team, Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

Andrea Vedaldi and Karel Lenc. MatConvNet: Convolutional neural networks for Matlab. In *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference*, pages 689–692, 2015.

Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. Caltech-UCSD birds 200. 2010.

Saining Xie, Tianbao Yang, Xiaoyu Wang, and Yuanqing Lin. Hyper-class augmented and regularized deep learning for fine-grained image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2645–2654, 2015.