Computer Vision Project 2
Snake Classification Using Convolutional Neural Networks

Nishai Kooverjee (1354477)
Matthew Cockcroft (1415624)

I. INTRODUCTION

This assignment looks at the classification of different snake species from RGB images, using a particular set of convolutional neural network architectures. The three architectures which we make use of are *AlexNet*, *ResNet-18* and *MobileNetV2*. We discuss the basis for choosing these three architectures and the main distinctions between them, after which we detail the approaches taken to set up and train the different models. Finally we compare the predictive accuracy of each of the models using a broad range of different evaluations and discuss some limitations and areas of expansion which we came across over the course of this assignment.

II. BACKGROUND

A. AlexNet

AlexNet is a CNN architecture introduced by Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton in their seminal 2012 paper [1]. The model achieved a Top-5 error[1] of 15.3% in the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC), clearly outperforming the next best model with a Top-5 error of 26.2%.

AlexNet was much larger than other CNNs at the time, using 8 layers consisting of 5 *convolutional* layers and 3 *fully connected* layers. This is shown in Fig. 1 below. The inputs to AlexNet are restricted to 256 x 256 pixels, and require 3 channels (greyscale images must be converted to RGB by replicating channels).

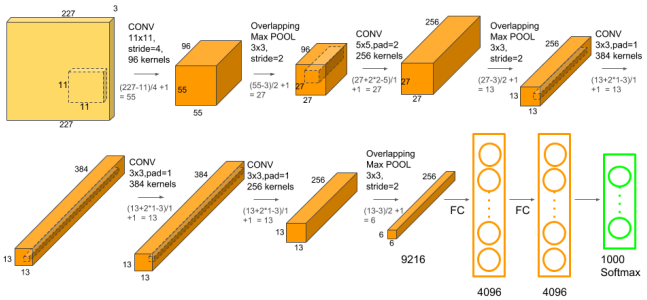


Fig. 1: AlexNet CNN architecture

[1]This is the rate at which the true label does not appear in the model's top 5 predictions.

The first two *convolutional* layers are followed by *Overlapping Max-pooling* layers. This is similar to *Max-Pooling* except the pooling computation considers adjacent windows that overlap. Every *convolutional* layer is followed by a *Rectified Linear (ReLu)* non-linearity. Lastly, the *fully-connected* layers help perform the classification of input images.

We choose to use this network architecture due its simplicity and good performance for image classification.

B. ResNet-18

ResNet, short for *residual networks*, are specific sets of CNN architectures first developed by Microsoft Research in 2015 [2]. An ensemble of 6 ResNet models produced the winning solution in ILSVRC-2015, achieving a Top-5 validation error of 3.57%.

Deep networks often suffer from issues such as vanishing gradients, and degradation in their ability to store complex representations. Residual networks aim to avoid these problems by implementation double- or triple- layer *skip connections* that contain non-linearities (such as ReLu), and use *batch normalisation*. This is shown in Fig. 2. These skip connections have a biological likeness to pyramidal cells in the cerebral cortex.

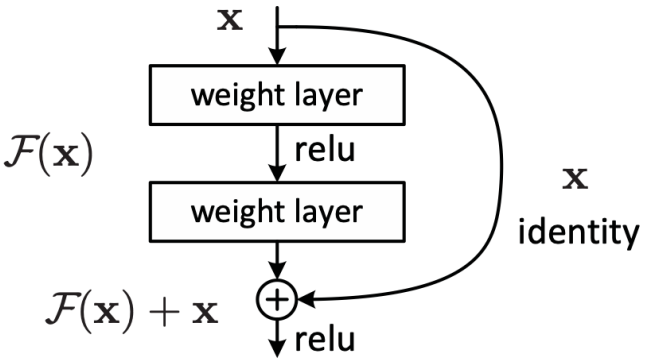


Fig. 2: Skip connection in residual network (also called a *residual block*)

The skip-connections allow for fewer layers to be used during the initial training stages, and expanding the network towards the end of training. The architecture allows deeper networks to converge faster by reducing the impact of vanishing gradients.

We select this network because it avoids the shortcomings simpler networks run into, such as vanishing gradients, by

using residual blocks. Specifically, we use ResNet-18: a residual network with 18 layers.

C. MobileNetV2

Developed by Google AI, MobileNets [3] are a set of CNN architectures which are focused on visual recognition in mobile applications. These models focus on reducing the network size, while maintaining high levels of accuracy. This can be seen in the comparative performance between MobileNet and AlexNet on the ImageNet dataset, where a reduced MobileNet is achieved 4% better accuracy than AlexNet while containing 45 $\times$ less parameters than it.

The model is based on *depthwise separable* filters, which split *convolution* into two separate layers. The *depthwise convolution* layer applies a single *convolutional* filter to each input channel, while the second layer consists of a 1 $\times$ 1 *pointwise convolution*, which uses linear combinations of the input channels to create new features.

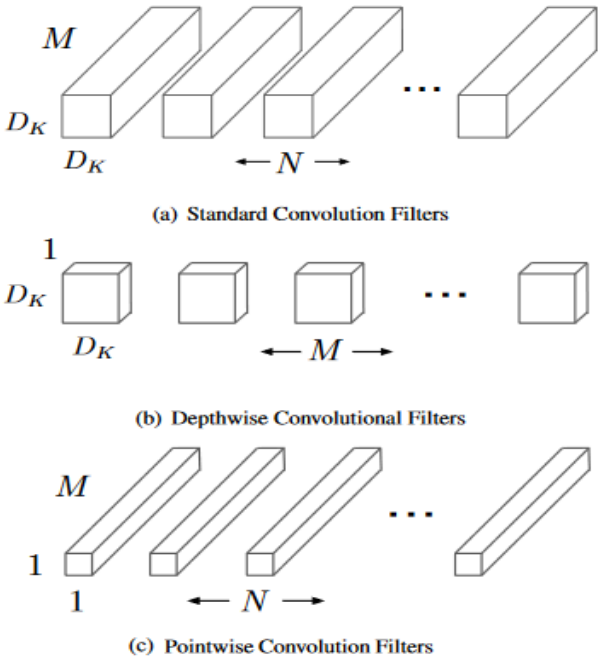


Fig. 3: A standard convolution filter (a) compared to a depthwise separable filter consisting of a depthwise (b) and pointwise (c) convolutional filters

This approach allowed for the large computation reduction, while maintaining accuracy. MobileNetV2 [4] further extended this model by adding linear bottlenecks which encode intermediate inputs and outputs of the model, and shortcuts between bottlenecks to allow for faster training.

We chose this architecture because of its efficiency in training time due its reduction in network size. This was ideal due to the limited time we had available.

III. METHODOLOGY

A. Data and Preprocessing

The dataset comprises of 82 601 RGB images with labels, totalling 21GB. The images are of varying size and there are 45 different species of snakes. We split the data into training, validation and testing sets. Each set comprises of 70%, 15% and 15% of the full dataset respectively. We maintain the proportion of classes during the split.

We resize all images to 224$\times$224, crop the images from the centre and normalise all images within the same values. For the training set, we also perform a random rotation between 0°and 90°. The result of these transforms can be seen in Fig. 4. The dataset also contained many corrupted images, and these were removed.

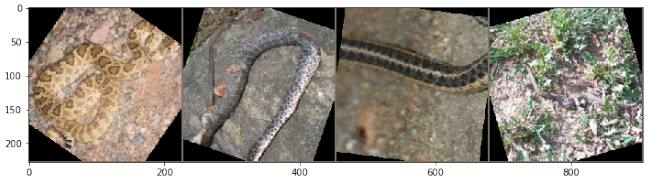


Fig. 4: Sample of transformed images

The training set is augmented by performing a random resized crop , so that more images than are actually contained in the set can be used for training. Rotation and flipping are useful so that our networks do not rely a snake's orientation to classify it.

B. Training networks

We retrain the pre-trained models bundled with the deep learning library *PyTorch* to train our networks. We replace the final fully-connected (FC) layer for each of our chosen networks with a 45 class FC layer corresponding to our snake classification problem.

We use the Adam optimiser with a learning rate of 0.0003 [5] and a learning rate scheduler to decay the learning rate over time. The loss function we use is the Cross-Entropy Loss. The networks are each trained for 10 epochs. We keep track of both training and validation set *loss* and *F1-scores* after every epoch.

At first, we attempted to train ResNet-50 and VGG-19 models, but these were too large and we ran into many issues training them. The three models we selected thereafter were much more manageable as they were smaller, and completed training in a more reasonable amount of time.

C. Testing the model

We pass images from the testing set through the fully-trained networks to produce a class prediction. The accuracy score (ratio of correct predictions to total images in the test set) and F1-score (the harmonic mean of *precision* and *recall* scores) are then computed. We also produce confusion matrices to understand how the networks perform for each class.

Due to the heavy class imbalance in the dataset, we report F1-scores instead of accuracy scores as this is a more robust means of measuring performance. We also produce normalised confusion matrices, where each entry $c_{i,j}$ in the matrix is the proportion of class i predicted as class j.

IV. ANALYSIS OF RESULTS

A. *Evaluating training performance*

During the process of training the models we record the loss and F1-scores for each epoch, for both the training and validation sets. We then graph these measurements, with Fig. 5 and Fig. 6 showing our results for the training and validation sets respectively.

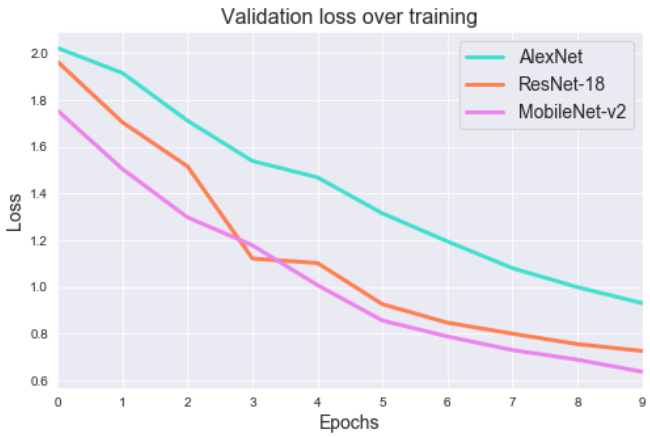
From these graphs we can see that MobileNet-v2 displays the best performance across both the training and validation sets. It obtains an F1-score of 0.612 on the training set, compared to ResNet-18's 0.552 and AlexNet's 0.487 after the completion of 10 epochs. It also reaches a loss value of 0.5989 on the training set, which is lower than both ResNet-18's 0.6572 and AlexNet's 0.8791.

Despite this, the graph in Fig. 5a shows that the training loss of the models had not fully converged after 10 epochs and ideally, without time constraints, we would have liked to train each of the models for a further 5 epochs to ensure convergence.

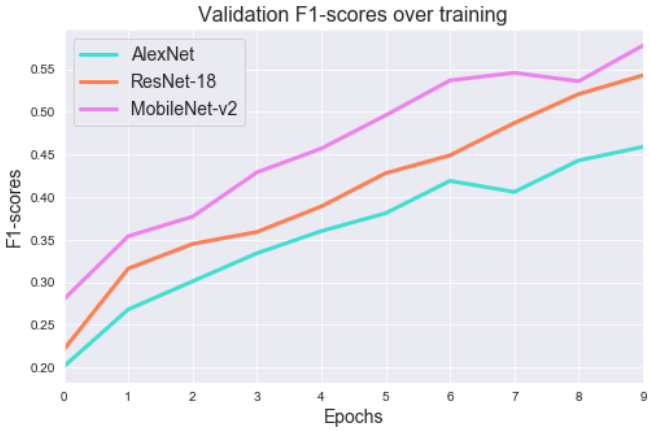
(a) Loss on training set across training

(b) Training set F1-scores across training

Fig. 5: Training set loss and F1-Scores for all networks across 10 epochs of training



(a) Loss on validation set across training



(b) Validation set F1-scores across training

Fig. 6: Validation set loss and F1-Scores for all networks across 10 epochs of training

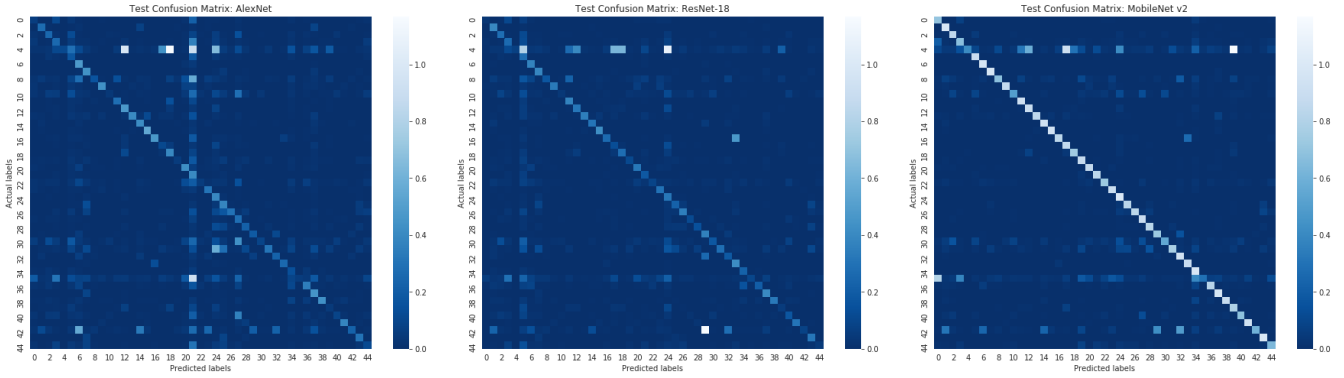
B. *Evaluating trained models*

To get a better idea of how each of the models perform we then test each of them on the unseen test set that we had created earlier. The results obtained can be seen in Table I, with MobileNet-v2 once again performing the best of the three models, achieving a test accuracy of 54.3% and an F1-score of 0.452. Despite typically being considered a *lightweight* architecture, the comparatively good performance of MobileNet is not a big surprise consider that it was developed much more recently, in 2018, compared to 2015 and 2012 for ResNet and AlexNet respectively.

Network Architecture	Accuracy	F1-score
AlexNet	0.325	0.283
ResNet-18	0.497	0.401
MobileNet-v2	0.543	0.452

TABLE I: Accuracy and F1-scores on the testing set for AlexNet, ResNet-18, and MobileNet-v2

The confusion matrices shown in Fig. 7 give a visual representation of how each of the models perform on the test set. From these plots we can observe that all of the



(a) Test confusion matrix for AlexNet (b) Test confusion matrix for ResNet-18 (c) Test confusion matrix for MobileNet-v2

Fig. 7: Confusion matrices for AlexNet, ResNet-18, and MobileNet-v2 on the testing set

models struggle with predicting class 4 correctly. Looking at the composition of the data this makes sense as class 4 is the biggest of all of the classes, with 11092 images (approximately 13.5% of the entire dataset). The images in this class vary greatly and often look very similar to snakes seen in other classes so it is unsurprising that the models often fail to distinguish between them. One potential solution to this issue could be to normalize our confusion matrices by multiplying the accuracies by the class size ratios.

V. LIMITATIONS AND EXTENSIONS

Due to time and resource restrictions, we could not achieve better performance by using larger, deeper networks. As mentioned earlier, we attempted to train *ResNet-50* and *VGG-19* models but these took very log to complete epochs as well as using the full capacity of the GPUs that were available, often resulting in memory errors. We were also impacted by technical issues in the labs due to account restrictions and power issues.

It has been shown by some of the contestants partaking in the Snake Classification Challenge[2] that using some of the larger networks can achieve accuracies of up to 90% on the training set. Had we been able to train such models we would have liked to try improve upon these scores by implementing dropout to prevent overfitting. In our case though, a maximum accuracy of only 72.8% was achieved on the training set and so the introduction of dropout would not have had much of an impact.

With more time available we would have also liked to have tested other network architectures, such as DenseNet [6] and EfficientNet [7] and compared their performance to our selection models.

VI. CONCLUSION

In this project we used convolutional neural networks to build a classification model to classify snake species. We designed various preprocessing techniques to deal limit variance in the input images. Next we chose three CNN architectures and fine-tuned pretrained models for each.

We observe that MobileNet performs the best for this task, achieving better accuracy and lower loss on the test set. We outline some limitations and extensions to the problems as well as analyse confusion matrices for class predictions. We see that CNNs are a useful model for the task, but can be improved further by tweaking hyperparameters and changing architecture.

[2]https://www.aicrowd.com/challenges/snake-species-identification-challenge

REFERENCES

[1] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[2] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[3] Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. and Adam, H., 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

[4] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. and Chen, L.C., 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4510-4520).

[5] Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[6] Huang, G., Liu, Z., Van Der Maaten, L. and Weinberger, K.Q., 2017. Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4700-4708).

[7] Tan, M. and Le, Q.V., 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. arXiv preprint arXiv:1905.11946.