
# Snake Identification through Transfer Learning

COMP90055 Computing Project (25 Credits)

Type of Project: Research Project

Supervisor: Prof. Richard Sinnott

Student: Haikuan Liu 1010887

February 2019

# Abstract

Although medicine has now reached a higher level, snake bite is still fatal because snake venom can quickly act and destroy vital organs of our body. To reduce casualties caused by snake bites, avoid being too close to poisonous snake and quickly finding the proper anti-venom are two effective ways. Snake recognition plays an important role in the both solutions. The aim of this paper is to find a solution towards snake identification. With the development of deep learning especially the emergence of **Convolutional Neural Network(CNN)**, some images can be accurately and intelligently recognized by computers. Some studies have already proven deep learning can perform significantly well on cat/dog breeds recognition. These previous works inspire us to develop an deep learning based application for snake species recognition. In this paper, methods of implementing the snake recognition model like **transfer learning, model selection, hyper-parameter tuning** are discussed in detail. The results show that a deep neural network architecture with a **ResNet50 feature extractor + one dense layer with thirty hidden neurons + one softmax layer** could accurately identify **17 snake species**. It achieves **82.48**% classification accuracy. The model also has advantages that it only takes **91MB** space and **50 epochs** for training5. The result shows that we could use methods discussed in this paper to build an efficient deep learning model for snake recognition. And the methods could also be further applied or extended to solve other similar image recognition problems.



**keywords: transfer learning, image recognition, convolutional neural network, hyper-parameter tuning**

# Contents

I certify that

- this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text.
- where necessary I have received clearance for this research from the University's Ethics Committee and have submitted all required data to the School.
- the thesis is 6079 words in length (excluding text in images, table, bibliographies and appendices).

# Acknowledgement

I want to give my special thanks to my supervisor Prof.Richard Sinnott. His invaluable advice helped me a lot not only for this project but also my learning process.

Besides, thanks all the contributors to the toolbox(Tensorflow, Python scikit-learn) which also helped me a lot in this study.

# 1 Introduction

Snake is considered as a dangerous animal because snakebite can be fatal. Although only 15% snakes are considered dangerous to human[1], there are 19,886 deaths per year reported[2]. Together with unreported venomous snakebites, the number of casualties estimated may be as high as 94000[2]. The most deadly part of being bitten by a snake is the snake venom injected into the body. There are two reasons why snake venom is still fatal today even though anti-venom is invented. The first is the toxic chemicals can quickly destroy an important function of the body, the victims must be treated within a valid time. The second is that different snake species could release different kinds of venom which can react differently with human body. For example, the venom of tiger snake and death adder mainly affect the nervous system, viper snake venom mainly does harm to the vascular system and twig snake venom act on blood cell and may thin the blood[3]. So it is difficult to get an effective anti-venom in a limited time unless the snake species is identified. However, even anti-venom itself frequently have side effects[3]. So the best way to prevent people from venomous snakebites is to let them know if the snake in front is poisonous so that people tend to seek help from professionals when they encounter a fatal snake.

In general, one can identify snake species from its visual characteristics like length, color, skin, etc. However, snake identification by human eyes is still known to be a hard problem for two reasons. One is that most people do not have experience in distinguishing snake species. The other one is that some species are similar in appearance(e.g. keelback snake and rough-scaled snake, as shown by Fig.1) while some snakes within a particular species could look different in appearance like Fig.2.To tackle these problems, we could train a computer which can extract and distinguish subtle features to help people identify snake species automatically. Unlike identifying snakes from metadata that can only describe part of snake characteristics like size and length, digital images are better choices that can capture more visual characteristics. And getting images of a snake can be as easy as pressing some buttons on mobile phones or cameras. The next problem we want to solve is how to recognize different snakes from images with the help of computers.

Figure 1: Keelback snake(left) and rough-scaled snake(right) look similar in appearance

Figure 2: Two eastern brown snakes can have different appearances

How to extract useful information or features from images has been studied for a long time in the field of computer vision or digital image processing. Before the concept of deep learning was proposed and without the help of powerful computers, algorithms in 1970s mainly focus on extracting partial, low level characteristics of an image like edge and shape[4]. Later on, the rise of statistical learning algorithms like Principle Component Analysis(PCA) can help computers recognize human faces[5] but still have no ability to "understand" the contents or differences between images. Breakthrough happened at 1990s, Yann LeCun et al.[6] developed a system that can recognize hand-written ZIP code numbers by training a Convolutional Neural Network(CNN) which simulates the behaviour of human eyes. With the improvement of computing powerful, CNN and its variations are used in more and more applications for image recognition. And it is turned out that CNN performs faster and more accurate than other algorithms. For example, researchers used CNN to recognize human actions[7]. Another research shows that CNN can achieve 97.6% accuracy on face recognition[8]. However, training a deep CNN model can be both time and resource consuming. For example, researchers trained a ResNet-50 model on 90-epoch ImageNet-1k dataset with a NVIDIA M40 GPU can take up to 14 days[9]. Another group used 1024 CPUs to train a AlexNet on 100-epoch ImageNet in 11 minutes[9]. Transfer learning makes it possible to train and test our own model on a personal computer within a short period of time. Research shows that we can build and train our own model upon a pre-trained one which solves different but related problem[10]. Some recent researches also adapt transfer learning techniques on CNN model to solve image recognition problem. Scientists in these researches freeze pre-trained layers for feature extraction and train their own feed

forward neural network for classification. For instance, researchers apply supervised fine-tuning on a pre-trained CNN model to recognize emotion, which performs on a small dataset but achieved good performance(55.6%)[11], researchers also find it feasible that pre-trained CNN model can be applied to solve related image recognition problems[12].

In this paper, we illustrate how to build an application for snake species identification using variations of CNN and transfer learning. First, we will talk about several relevant researches that inspire our work in 2. Then the sources of dataset for training and testing is presented in 3. Next, details on what methods we take to build and train the model, how we measure the performance and how we improve its performance are explained in 4. Results and conclusions are illustrated in 5. Finally, in 6, the feasibility, drawbacks of our model and possible future work are discussed.

## 2 Related Work

Transfer learning on CNN model has been widely adapted by many image recognition researches and applications to avoid training a deep CNN model from the beginning, which can drastically reduce training time without sacrificing accuracy. For example, results of cat and dog breed recognition based on transfer learning can reach 80%[13][14]. A good pre-trained CNN model should capture image features at different level and trained on a general dataset so that it can be adapted to the majority of applications for image recognition.

ImageNet is one of the most largest and complex dataset that contains more than 20,000 categories, 14 million hand-annotated digital images[15] for object recognition research. Its subset which contains one thousand non-overlapping classes is used for competing between software programs by ImageNet Large Scale Visual Recognition Challenge(ILSVRC)[16]. Since 2010, the error rate on recognizing images of ILSVRC has been gradually reduced by different CNN models like AlexNet, GoogleNet, etc. Because of these properties, models trained on ImageNet could be more general and can extract features in large scale and at different levels[12]. So many researchers choose to fine-tuning the pre-trained models on ImageNet rather than other dataset. In this paper, we also decide to use pre-trained model on ImageNet as the feature extractor for transfer learning.

Since the ILSVRC is held, variations of CNN architecture occupied top positions in terms of image classification accuracy. In general, a CNN model contains four different kinds of layers that perform different actions. The convolutional layer applies convolution filters to the input image to emulates the response of an individual neuron to visual stimuli[17]. The pooling layer combines the outputs of local receptive field at one layer into a single neuron in the next layer[18]. The dropout layer randomly drops features extracted by previous layer to avoid over-fitting. The fully connected layer which is also known as classification layer learns from features to generate output labels. An outstanding very deep CNN architecture proposed by Visual Geometry Group(VGG, shown by Fig.3) outperforms other models in 2014 with top-1 accuracy 71.3% and top-5 accuracy 90.1%[19]. However, VGG model contains a large amount of trainable parameters which takes around 540MB(528MB for VGG16, 549MB for VGG19) space. Another widely used architecture with significant performance is ResNet(shown by Fig.4) proposed in [20]. It presented a residual learning framework to ease the training process and gain accuracy from increased network depth. This leads to 74.9% top-1 accuracy and 92.1% top-5 accuracy on the ImageNet dataset. Compared with VGG, ResNet contains less trainable

weights so it only occupies 99MB, which makes it an distinguished architecture. As deep learning playing important role in more and more fields, deploying deep learning model to a variety of devices is a worthwhile research. To meet this demand, researchers in Google invented another small but powerful CNN architecture called MobileNet which only takes about 16MB. The performance of MobileNet on ImageNet can reach 70.4% top-1 accuracy and 89.5% top-5 accuracy. The weights and models of the above architectures pre-trained on ImageNet are publicly available from the Internet. And the three CNN architectures were all tested in our snake species identification project.

To build up a deep learning model, one does not need to implement the data flow, optimizer and analyzer by its own. There are several libraries and APIs supporting us to build or test deep learning model, such as Tensorflow, Pytorch, Kaggle, etc. Among them, Tensorflow turns out to be more suitable for multi-platforms(e.g. TensorflowLite for machine learning on mobile). And functions like loading different pre-trained models are directly provided by Keras[21], an open source neural network library that can run on top of Tensorflow. Consider that our model/application could be deployed to different devices in the future, we choose to implement our model on Tensorflow.

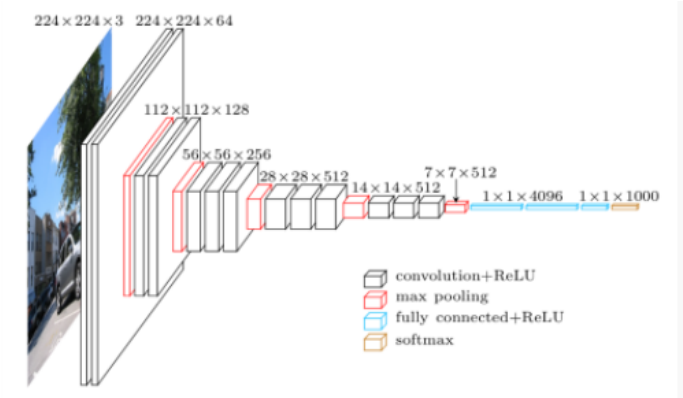


Figure 3: The architecture of VGG model

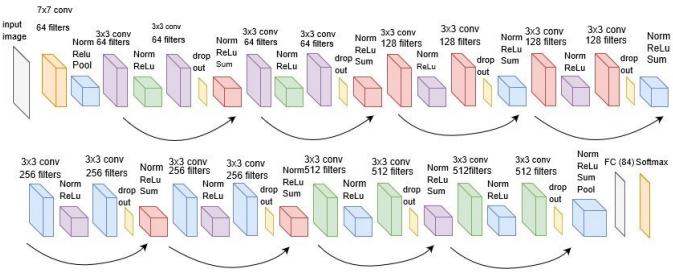


Figure 4: The architecture of ResNet model

# 3 Dataset

Data used in our experiments contains two forms, digital images and text information which describes the characteristics of each snake species. Only images are used to train our deep learning model, the text data is also stored to help us have a deeper understanding of each snake species or it can be used for further development purpose, like it can be displayed beside images or photos identified by our model deployed on mobile applications.

The 683 digital images in '.jpg' format are distributed in 17 directories which are named after 17 different snake categories. And there are about 40 images for each species. For example, if an image shows an garter snake, then the image is placed on the directory named as 'GarterSnake'. Each species contains around 40 images of snake that within the species. All of the images are allocated an unique index for the convenience of manipulating. So there are indexes from 0 to 682 each corresponding with an snake image. All of the common name of snake species and the mapping between snake species and image indexes are shown by Fig. 5. A screenshot of images that record garter snake is shown by Fig. 6. There are three main sources that images are manually retrieved from, which is Google Image, Flickr and ImageNet respectively. From the Google Image and Flickr, images are retrieved by directly searching the common name of snake species, such as 'garter snake'. From ImageNet, images under Wordnet ID n01726692 which contains images of snake, serpent and ophidian are retrieved and selected.

| Snake Species | Indexes |
|---|---|
| CarpetPython | 0-39 |
| CoastalTaipan | 40-79 |
| Cobra | 80-119 |
| CommonDeathAdder | 120-159 |
| CommonEuropeanViper | 160-199 |
| Copperhead | 200-239 |
| GarterSnake | 240-279 |
| InlandTaipan | 280-319 |
| Keelback | 320-359 |
| MilkSnake | 360-399 |
| MulgaSnake | 400-441 |
| RedBelliedBlack | 442-481 |
| RoughScaledSnake | 482-521 |
| SmallEyedSnake | 522-561 |
| SpottedPython | 562-601 |
| TigerSnake | 602-642 |
| TreeSnake | 643-682 |

Figure 5: The mapping between snake species and image indexes


Figure 6: A screenshot of all snake speices and images that record garter snake

Besides the digital images, we also prepared text metadata which describes the habitats, the average and maximum length and if poisonous corresponding of each snake species. These metadata are stored in '.txt' file and collected from wikipedia. Each '.txt' file corresponds with and named after a snake species. And within each file, the first line describes the habitats, the second describes the average and maximum length of snakes within the corresponding species, the third line illustrates if that snakes species is poisonous. A screenshot of the metadata of garter snake is shown by Fig.7

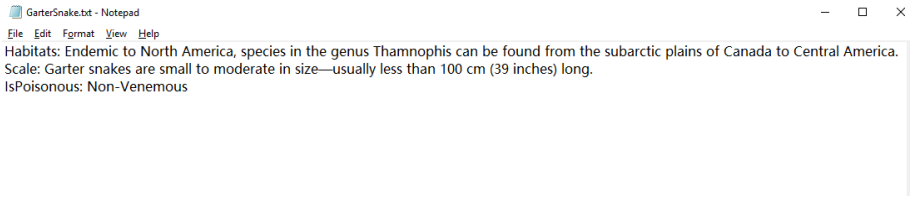


Figure 7: The meta data of garter snake

In order to train a robust deep learning model, the image data is shuffled before fitting into the training process. By doing so, the training data and the test data both contains each snake species, so situations that some snake species may not be trained or tested is avoided. Besides, a robust model requires sufficient training data while convincing results need the support of enough test data. To balance them, the whole dataset is separated into two parts by a ratio of 4 to 1 after being shuffled. That is, 546 images are used for training and 137 images are used for testing in our experiments. Many other researchers also adapt the 4 to 1 separation strategy as rule-of-thumb in their research and get a good result, see [13][14]. And consider that the implemented CNN model takes the input of fixed size(the default size is 224*224*3 for built-in model in Keras), so all of the images are resized to 224*224*3 before training.

# 4 Methods

## 4.1 Data Pre-processing

A deep CNN model cannot directly operate on image objects loaded from disk unless they are converted into an appropriate format. The following paragraphs describe three common methods applied in our experiments to pre-process the training data.

### 4.1.1 One-hot Labels

Supervised learning requires training data with gold-standard labels so that the model could gain right knowledge during training. In our dataset, the species of snake shown by an image can be considered as labels. However, most deep learning toolkit only accept input in the so called 'one-hot' format. To convert the 17 text snake species labels into one-hot vector labels, they are sorted alphabetically first and attached with 0-16 categorical values accordingly. Then the 17 categorical values are converted into one-hot vectors by following the rule: an array with 17 entries initiated to be 0, the entry with index that is equal to the corresponding categorical value is set to 1. The mapping between text labels, categorical labels and one-hot vectors is illustrated by Fig.8. For example, the label 'GarterSnake' is the sixth label after being sorted, so it is attached with a categorical value '6'. Then we set the sixth entry of a zero vector with 17 entries to '1'. This vector is used as the input labels meaning that the corresponding input data is converted from an image showing a garter snake.

```
Snake Species            labels      One Hot Labels
CarpetPython         ==> 0       ==> [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
CoastalTaipan        ==> 1       ==> [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Cobra                ==> 2       ==> [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
CommonDeathAdder     ==> 3       ==> [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
CommonEuropeanViper  ==> 4       ==> [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Copperhead           ==> 5       ==> [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
GarterSnake          ==> 6       ==> [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
InlandTaipan         ==> 7       ==> [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Keelback             ==> 8       ==> [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
MilkSnake            ==> 9       ==> [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
MulgaSnake           ==> 10      ==> [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
RedBelliedBlack      ==> 11      ==> [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
RoughScaledSnake     ==> 12      ==> [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
SmallEyedSnake       ==> 13      ==> [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
SpottedPython        ==> 14      ==> [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
TigerSnake           ==> 15      ==> [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
TreeSnake            ==> 16      ==> [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
```

Figure 8: The mapping between text labels, categorical labels and one-hot vectors

### 4.1.2 Online Learning

The deep learning model operates on the RGB values of each or several pixels rather than the whole image object, so the image instances should be converted to numeric matrix whose entries represent the RGB values of pixels lying on the image. After conversion, we get a four dimensional matrix $\mathbf{A} = (samples, height, width, channels)$ which contains all the RGB values of images. Note that, dimension 1 represents the number of images/samples, dimension 2 and 3 represent the size of the images, dimension 4 represents the number of channels which is 3 in RGB format. We still could not fit the matrix into our model now because it can be infeasible for computers to apply a huge amount of complicated operations on such a huge matrix. To reduce the operations a computer should calculate at the same time, the input matrix can be split into several batches and each batch contains fixed number of samples. We let our model learn one batch at a time. For example, there are 546 images for training in our experiments, we can split them into 110 batches, each batch(except the last batch) contains 5 samples. So each time, the model learns matrix with dimension $(5, 224, 224, 3)$ instead of $(546, 224, 224, 3)$. This strategy is rely on the theory of sequential learning or online learning[22] which explains the feasibility of updating the estimator at each step when data arrives in a sequential order. With this, it is possible to apply machine learning algorithm to a large scale problem like image recognition in our case.

### 4.1.3 Mean Subtraction and Normalization

Besides, two images that show the same content may also have different RGB values for each pixel due to the lighting condition(e.g day and night). Training a deep learning model using images that have high variations in ranges of RGB values may slow the training process because the gradients during training is out of control. To solve this problem, the data should be centered around zero by subtracting the mean for each color channel(RGB channel in our case). And some models require the RGB values to be in the range of 0 to 1 or -1 to 1, so the values for each channel should also be normalized.

### 4.1.4 Deal with Learning Invariant

Consider that the neural network does not have the same recognition ability as the human brain. It still could not learn invariant features from data. For example, the deep learning model cannot recognize rotated images(In this case, the rotation angle is an invariant feature). As shown by Fig.9, the neural network may recognize the image on the left but may not recognize the image on the right although the tow images show the same content. This is because the two images are rotated in different angles and the position of pixels are changed. To solve this problem, a typical methods is to manually enrich the training data. In our experiments, original images are trained first. Then the same set of images are rotated to a different angle and trained. The same strategy of making the neural network have the ability of learning invariant features by enriching the training set is also described in "Pattern Recognition and Machine Learning"[23].

Figure 9: Images rotated by different angles

## 4.2 Transfer Learning and Model Architecture

As discussed in section 2, the best way to create an usable deep CNN model with a short training time is combining a pre-trained feature extractor with self trained fully connected neural network as classifier. Take VGG(Fig.3) as an example of feature extractor, the pre-trained weights or parameters can be downloaded from [24]. Only part without the fully connected layers(the blue layers in Fig.3) are used. The output in shape $(7, 7, 512)$ is the feature vectors extracted from the input image. The feature vectors are then forwarded to the fully connect layer to be classified. What we should do is to train the fully connected layers to classify images from the features extracted(the output of the pre-trained layers). The reason of using a self trained fully connected layers instead of using a pre-trained one is that we are trying to solve a different but similar image recognition problem which may classify the im-

ages into a different number of classes/labels. In our experiments, the model should classify images into 17 labels that represent 17 different snake species, which is different from the model pre-trained on ImageNet that classify images into 1000 labels.

The architecture of the fully connected layers can be different in terms of practice or specific problem we want to solve, although VGG recommends using two dense and one softmax layers, ResNet recommends using one dense and one softmax layers. Note that the softmax layer is only determined by the number of classes which is 17 in our research. And the softmax function generates the results showing the probability that the input image belongs to each class. So the softmax layer should always be the same and the last layer of our model though the architecture of dense layers can be different. The different architectures of the fully connected layers are tested and selected using cross validation(details in section 4.4) based on performance metrics(details in section 4.3) we used. The best architecture which achieves 81.2% accuracy in our experiments is pre-trained ResNet50 layers + one dense layer with 30 hidden neurons + one softmax layer. Details in how to find the most proper architecture is illustrated in section4.4.

Besides architecture, activation function applied by a neuron and algorithm an optimizer used are another two critical component for a deep CNN model. In neural network, the activation function determines how a neuron transfer its inputs to outputs. Selection between different activation functions can be done by using cross validation4.4.1. But based on researches of other scientists, only a neural network with non-linear activation function(e.g. sigmoid, softmax and Rectified linear unit(ReLU), see Fig.10) is proven to be a universal function approximator which can compute non-trivial problem using small number of nodes[25][26]. And to avoid the vanishing gradient problem which happens during training an artificial neural network, ReLU is recommended. Based on the above research results, ReLU is chosen to be the activation function for hidden neurons in this paper.

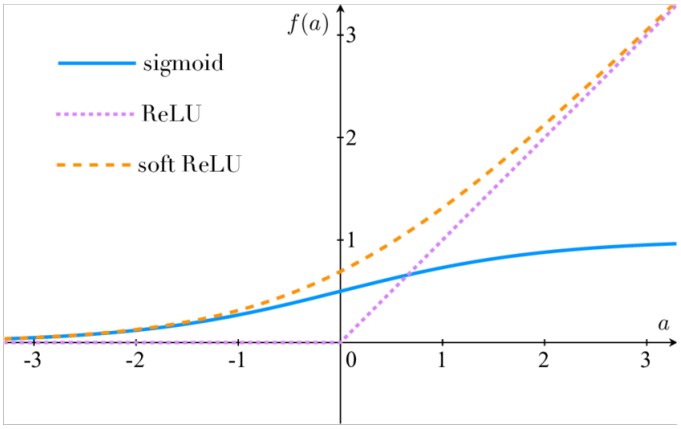


Figure 10: Examples of common non-linear activation function

A powerful optimizer widely used in CNN is the Adaptive Moment Estimation(Adam)[27] which could adapt different learning rate in different applications. It uses both the gradients and the second moments of the gradients in its weights updating rule. The formula for weights updating using Adam is shown by Fig.11.

$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w L^{(t)}$$
$$v_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1-\beta_2)(\nabla_w L^{(t)})^2$$
$$\hat{m}_w = \frac{m_w^{(t+1)}}{1-(\beta_1)^{t+1}}$$
$$\hat{v}_w = \frac{v_w^{(t+1)}}{1-(\beta_2)^{t+1}}$$
$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w}+\epsilon}$$

where $\epsilon$ is a small scalar used to prevent division by 0, and $\beta_1$ and $\beta_2$ are the forgetting factors for gradients and second moments of gradients, respectively. Squaring and square-rooting is done elementwise.

Figure 11: The update rule of Adam optimizer

## 4.3 Performance Metrics

A typical method for measuring classification performance of a model is based on confusion matrix which is also called contingency table[28]. Fig.12 explains how to construct a confusion matrix from classification results. In our research, we only want to test whether our model can classify the snake in an image into the corresponding species, so only accuracy is considered. Accuracy can be calculated by the following formulation:

$$Accuracy = \frac{\sum_{k=1}^{K} TP_k}{N} \tag{1}$$

where $N$ denotes the number of predicted values, $K$ denotes the number of classes.

Another metric that can describe the behavior of a deep learning model is computing the loss during training. In our experiments, the cross entropy loss which is

commonly used in classification problems[28] is chosen to evaluate the output of the model. The cross entropy can be computed by the equation 2

$$L = -\sum_{c=1}^{M} y_{o,c} log(p_{o,c}) \tag{2}$$

Note that $M$ is the number of classes, $y_{o,c}$ is binary indicator shows whether a data $o$ belongs to class $c$, $p_{o,c}$ denotes the probability of data $o$ belongs to class $c$. It computes the difference between the predicted labels(output of model) and the target labels(gold standard labels) and then back propagated to update each weight. So it could let us know how our deep learning model behave towards our desired outputs after each epoch. In the experiments, accuracy and cross entropy loss are used to measure the performance of our model.

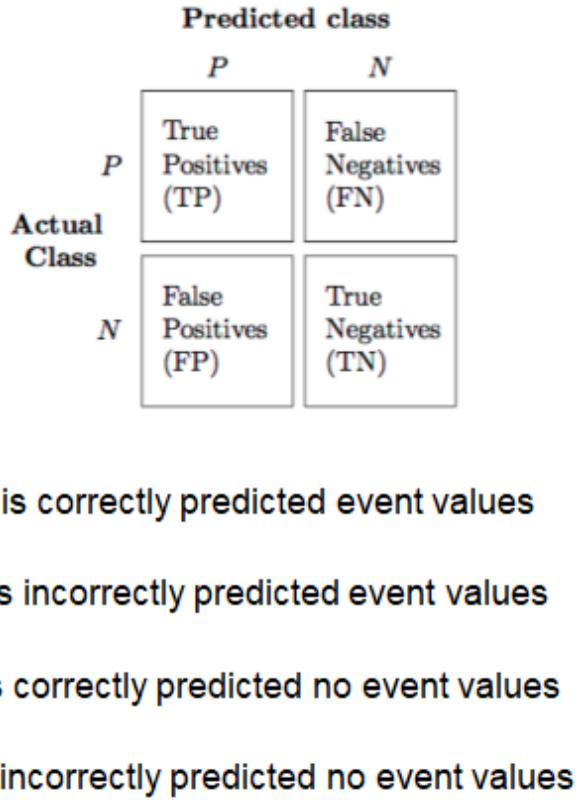


Figure 12: How to construct a confusion matrix

## 4.4 Tuning the Model

Usually, a good deep learning model not only has appropriate weights that can recognize images precisely but also has proper hyper-parameters that support the model to convergent fast during training. Rather than parameters of the model for underlying system like weights, hyper-parameters are settings that can control the behavior of a deep learning algorithm, such as the number of hidden layers and

neurons, learning rate and the algorithm of optimizer, etc.

The number of hidden layers and neurons can directly affect the complexity of a model. More hidden layers and neurons indicate that the model uses more transformations/operations to represent a pattern. A model that is too complicated(too many hidden layers and neurons) would suffer the overfitting problem. However, if a model is too simple, it will not be able to learn the patterns from data. The learning rate controls the learning behavior of the model. A larger learning rate makes the model learn faster but take the risk of never convergent while model using a smaller learning rate is more likely to convergent but takes more time. The algorithm an optimizer using plays a similar role as the learning rate. An appropriate algorithm using by an optimizer should be able to guide the model towards the optimal in a short time.

As the hyper-parameters may greatly affect the performance and the training time of a model, many studies around hyper-parameter tuning that offer a variety of solutions. In our experiments, the following methods are applied.

### 4.4.1 Cross Validation

To test which features lead to better classifying performance, to make most use of our data and gain a convincing results, we decide to apply a so-called 5-fold cross validation in our experiments. Cross validation can reflect the quality of our model directly on the performance metrics we used(accuracy and cross entropy loss). And it can help us find proper hyper-parameters like learning rate or model architecture. The process is illustrated by Figure 13. The tools used for cross validation is python scikit-learn package described in [29].

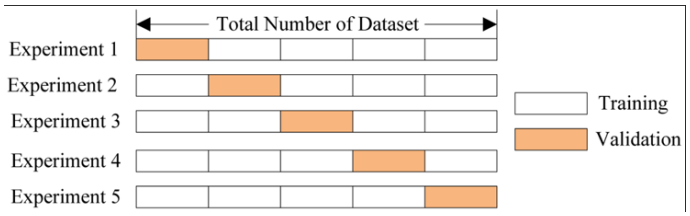


Figure 13: 5-fold cross validation

That is, the training set is separated into five equal sized subsets. Five iterations are run and in each iteration, we select a different subset(validation set) for testing and use the rest four subsets for training. There are five results derived from testing on

the five different validation sets. Those results are averaged to get the final results. Different settings/hyper-parameters are chosen based on the final results.

### 4.4.2 Early Stopping

Early stopping is a common method that helps to determine the number of epochs the model trained on the training set. It is a regularization approach applied on machine learning algorithm to avoid overfitting[30]. Overfitting is occurring when a deep learning model fits the training data well while do not perform well on recognizing the test data. Overfitting can be detected during the training phase by plotting the loss. The following figure(Fig. 14) shows the cross entropy loss of training a ResNet50 + one dense layer(30 hidden neurons) + softmax model. From Fig.14, we can see that the loss stop decreasing or sometimes even start to oscillate around 50 epochs. It indicates that the model tends to learn enough knowledge about the training data and start to overfit at around 50 epochs. Based on this observation, we can stop the training phase once the loss starts to oscillate or barely decreasing. By doing this, overfitting because of too many learning epochs can be avoided and the training time is shortened.

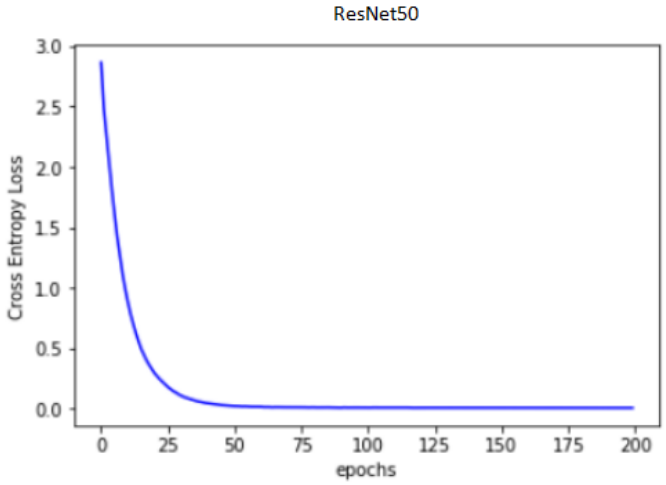


Figure 14: The cross entropy loss when training a model

# 5 Results and Analysis

Firstly, three types of CNN architecture as feature extractor were tested. They were all combined with one dense layer with thirty hidden neurons plus one softmax layer. The accuracy of our model tested on test data when using different CNN architectures is shown by Fig.15. And the cross entropy loss for each architecture is plotted in Fig.16. As illustrated by the table, ResNet50 achieved the highest accuracy which is 80.29% among the architectures tested. Both VGG16 and MobileNet gave us an unsatisfactory result which are 53.47% and 47.52% respectively. The loss also tells us that using ResNet50 as feature extractor suffered less overfitting problem because its loss decreasing smoothly without oscillating. The only drawback of ResNet50 is that it takes 99MB space which is larger than the MobileNet(16MB) does, which may hider its deployment on other devices. However, consider the ResNet50 greatly outperformed other architectures in accuracy, the model based on it is chosen to be tuned in this study.

| CNN Architecture | Accuracy/% |
|---|---|
| VGG16 | 53.47 |
| ResNet50 | 80.29 |
| MobileNet | 47.52 |

Figure 15: Accuracy of model using different CNN architecture

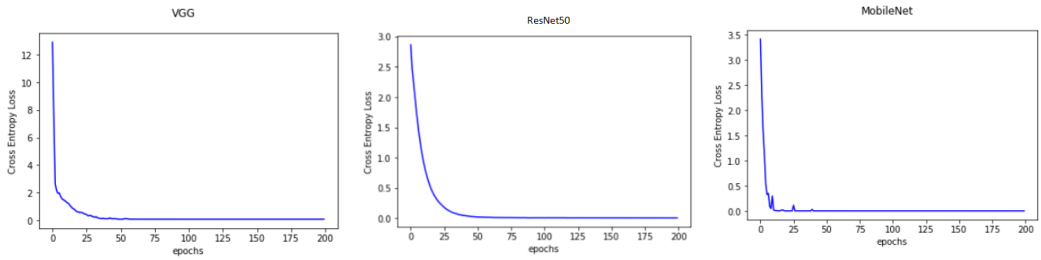


Figure 16: The cross entropy loss of model using different CNN architecture

From Fig.16, we can also find that the loss barely decreased after epoch 50. To avoid overfitting and to save the training time for later experiments, our model stops training at 50th epoch. The early stopping method increased the accuracy on test data from 80.29% to 82.48%.

Then the learning rate is tuned by running a 5-fold cross validation on the training data. From the loss curve shown by table17, we find that the model performed best when the learning rate is set to 0.0002. The proper number of hidden layers and neurons are also selected by cross validation and investigating the behavior of

neurons. Table18 and table19 tells us that when using a single hidden layer with thirty hidden neurons, the model performed the best. Overall, the model using ResNet50 + one hidden layer with thirty neurons + one softmax layer achieved the best performance(82.48%).

| Learning Rate | Accuracy/% |
|---|---|
| 0.00002 | 77.37 |
| 0.0002 | 82.48 |
| 0.002 | 81.02 |
| 0.02 | 77.37 |

Figure 17: Accuracy using different learning rate

| The number of hidden layers | Accuracy/% |
|---|---|
| 1 | 82.48 |
| 2 | 76.64 |
| 3 | 68.61 |

Figure 18: Accuracy using different number of hidden layers

| The number of hidden neurons | Accuracy/% |
|---|---|
| 10 | 69.34 |
| 20 | 81.75 |
| 30 | 82.48 |
| 40 | 79.56 |

Figure 19: Accuracy using different number of hidden neurons

To measure the model more comprehensively, statistics of wrongly labeled images are analyzed, too. In one of the experiments, there are 22 images are given a wrong label(accuracy 83.94%). Note that if an image is processed by the model, the output would be a 17-dimensional vector because there are 17 snake species in our dataset we want to identify. Each dimension of the vector represents the confidence that the model believes the image belongs to the corresponding label. For example, the output of the model when taking image indexed 130 as input is shown by Fig.21. It shows that the model has 53.15% confidence that believe the image belongs to label '12' which represents the Rough-Scaled Snake and the model has 46.49% confidence that believe the image belongs to label '3' which represents the Common Death Adder. So the model would classify the image with index 130 to be a Rough-Scaled Snake by applying argmax to the output vector. Instead of taking argmax to get the 'most likely' label, we printed the top 3 labels the model believes the image belongs to, as shown by Fig.20. By checking the 22 information of wrongly classified images, we found that there were 19 of them contain both the wrongly predicted label and its actually label in the top 3 'most likely' label. This means that, although sometimes the model could misclassify the image into a wrong label, the model could

still attach the image with the right label by giving it 3 chances. So the cognitive ability of our model is proven to be good, further studies on why the model could misclassify those images are discussed in next paragraph.

```
Misclassified image index: 130
Wrong label: RoughScaledSnake
Label should be: CommonDeathAdder
Confidence of each category: {'CarpetPython': 0.09653051383793354, 'CommonDeathAdder': 46.48644030094147, 'RoughScaledSnak
e': 53.15613150596619}

Misclassified image index: 553
Wrong label: CoastalTaipan
Label should be: SmallEyedSnake
Confidence of each category: {'SmallEyedSnake': 10.173668712377548, 'MulgaSnake': 33.953505754470825, 'CoastalTaipan': 45.457
29458332062}

Misclassified image index: 321
Wrong label: CommonDeathAdder
Label should be: Keelback
Confidence of each category: {'Keelback': 15.395092964172363, 'TigerSnake': 22.398488223552704, 'CommonDeathAdder': 43.537834
28668976}
```

Figure 20: Information about wrongly labeled images

```
array([9.65305138e-04, 1.30820055e-07, 1.17273213e-08, 4.64864403e-01,
       7.93930085e-04, 9.45622101e-04, 1.13941478e-05, 4.81248108e-08,
       9.72632552e-05, 3.34270851e-08, 1.54977533e-05, 4.95489871e-07,
       5.31561315e-01, 4.16274124e-05, 1.36765561e-06, 4.27260209e-04,
       2.74246180e-04])
```

Figure 21: The output of the model when taking image indexed 130 as input

In addition to the overall accuracy, accuracy of each snake species is recorded illustrated by Fig.22. It shows that the majority of snake species can be recognized by our model with around 80% accuracy. Among them, Garter Snake, Milk Snake and Small Eyed Snake can be clearly recognized with 100% accuracy. However, the model performed not so well when recognizing the Coastal Taipan and the Tree Snake, whose accuracy only reached 50%. By printing out the 22 misclassified images, we found that there are 4 out of 5 misclassified images of Coastal Taipan were predicted to be Inland Taipan. This is a reasonable problem because the Coastal Taipan and Inland Taipan are all belong to the so called genus Oxyuranus family which shares some common visual features like scale and the shape of head. And according to a 2016 genetic analysis, the central ranges taipan being an offshoot of the common ancestor of the inland and coastal taipans[31]. This indicates that if the model could not distinguish between two species well, the two species may share some common features and thus may have underlying relationship like having a common ancestor. The reason for recognizing tree snake poorly may be that tree snakes have high variations in body color(green, black, blue and yellow tree snakes have been noted) and scale. To better recognize it, feeding more training images to our model may be a solution. Besides, keelback snakes are often misclassified as rough-scaled snakes while rough-scaled snakes are misclassified as keelback snakes, too.

There are 8 misclassified images showing a rough-scaled snake or keelback snake, 4 of them shows a keelback snake but recognized to be a rough-scaled snake, 2 of them shows a rough-scaled snake but recognized as keelback snake. This is because the coloration and scale structure of most rough-scaled snakes and keelback snakes are the same like what 1 shows us. Some of the other misclassified images may due to the image quality. Like what Fig.23 shows us, the carpet python only occupies a small part of the image. After being scaled, features which can tell the differences between them like the eyes and spots may be blurred. To solve this problem, techniques(like object detection, image lossless scaling, etc.) that can keep features as its original while the image is being scaled should be applied to our model.

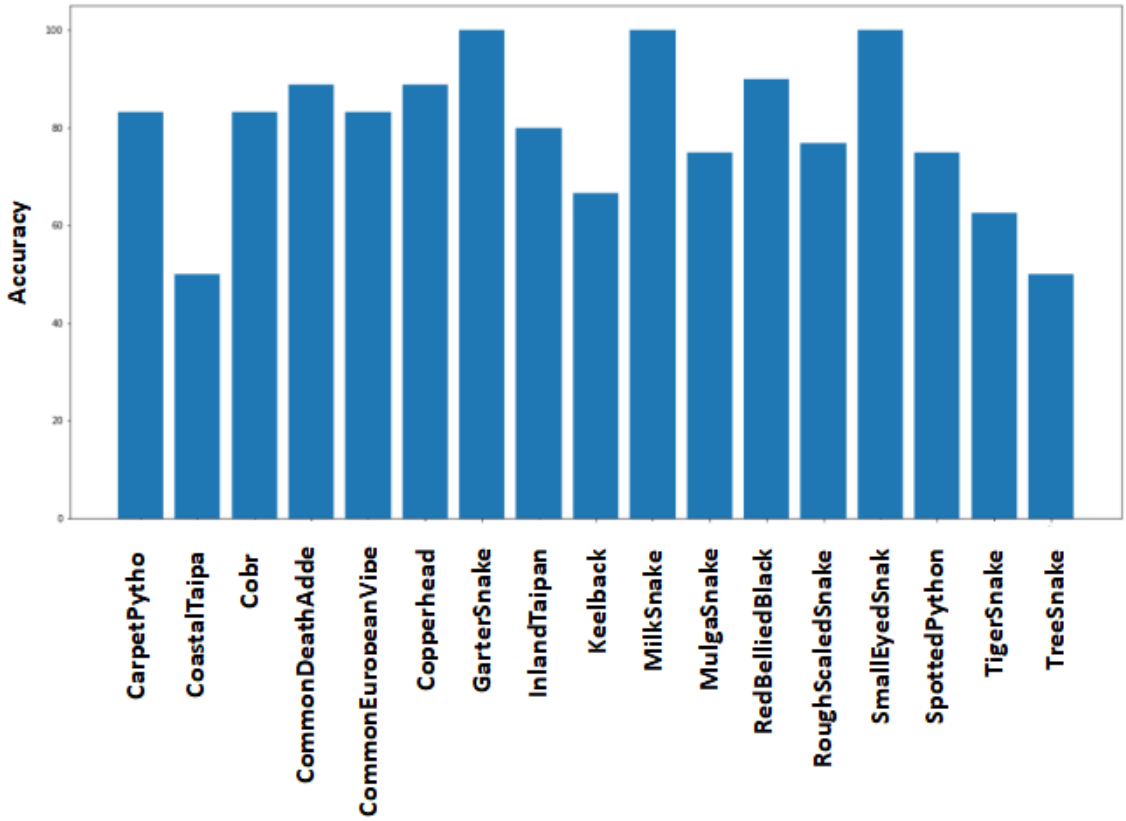


Figure 22: Classification accuracy of each snake species

Figure 23: Misclassification between the carpet python(left) and the spotted python(right)

# 6 Future Work

In this section, how our study can be extended and inspire other researches are discussed. Methods/techniques to be tested in our future study are also illustrated by the following paragraph.

The model we developed only takes around 91MB space and convergent very fast(in 50 epochs), so we could consider build our model into a mobile APP which can help people identify snakes in their daily life. Since Tensorflow Lite toolkit which is compatible with Tensorflow has been designed to help deploy the deep learning model on mobile phones, it is possible to tune or manipulate our model on a mobile phone. By building a snake recognition APP, one can recognize a snake by simply taking a picture using mobile camera, which may help people get away from dangerous snakes. This also helps us to continuously train our deep learning model(a machine learning algorithm can learn from sequential data, see 4.1.2) using real life images rather than those collected from the Internet.

As analysis states in 5, some images are misclassified because unrelated background occupies a large part. Thus, we could improve our model by combining it with object detection techniques to only capture useful objects automatically. Some modern object detection algorithms like Single Shot Detector(SSD) and Faster R-CNN are also based on deep learning and thus can be easily combined with our model. To help improve the classification accuracy, we may also combine the visual features like what are shown by an image/picture with non-visual features like snake weights, where the snake is found, etc.

Methods discussed/implemented in this study are also helpful for other research like taxonomy. For example, if two snake species that are misclassified frequently like the model often confuses keelback snake with rough-scaled snake, it may indicate that there is underlying relationship between the two. The model implemented can also be applied to the other similar image recognition problems. And the methods used in this study like early stopping, cross validation, can be used to build other deep learning models.

# 7 Conclusion

This study explored the feasibility of building an application for snake species recognition based on deep learning. Firstly, the motivation and significance of this study is talked about which followed by list of related researches. Based on those previous work, we decided to implement our application using deep learning techniques. Then 683 images of different types of snakes are collected, pre-processed and converted to format which can be recognized by deep learning model. We chose the classification accuracy and cross entropy loss to measure the performance of different models. The best implementation is the combination of ResNet50 pre-trained on ImageNet as feature extractor with self trained fully connected neural network as classifier. Then the self trained fully connected neural network is optimized by tuning its hyper-parameters using cross validation, early stopping, etc. Finally, the classifier with one hidden layer, thirty hidden neurons and whose learning rate is set to 0.0002 performed the best. Results show that the model performed remarkable on snake species recognition with 82.48% accuracy on test data and the cross entropy loss tended to decrease smoothly during training. The training process only took 50 epochs to convergent. Besides, the model with trained weights takes 91MB space, which is acceptable for many devices. Based on analysis, the misclassification may due to poor image quality, not sufficient training images and two similar snake species which share many common visual characteristics. At the end of this paper, possible future work/researches like how to further improve the classification accuracy and feasibility of deploying the model on different devices are discussed.

# Bibliography


[1] Barry S.; Richard C. Dart; Robert A. Barish Gold. Bites of venomous snakes. 5:347–356, April 2002.

[2] AR; de Silva N; Gunawardena NK; Pathmeswaran A; Premaratna R; Savioli L; Lalloo DG; de Silva HJ Kasturiratne, A; Wickremasinghe. The global burden of snakebite: a literature analysis and modelling based on regional estimates of envenoming and deaths. 5:218, November 2008.

[3] José María; Bruno Lomonte; Guillermo León; Alexandra Rucavado; Fernando Chaves; Yamileth Angulo Gutiérrez. Trends in snakebite envenomation therapy: Scientific, technological and public health considerations. 13:2935–2950, 2007.

[4] Richard Szeliski. Computer vision: Algorithms and applications. pages 10–16, September 2010.

[5] Javier Navarrete, Pablo; Ruiz-Del-Solar. Analysis and comparison of eigenspace-based face recognition approaches. *International Journal of Pattern Recognition and Artificial Intelligence*, 16:817–830, November 2002.

[6] J. S. Denker D. Henderson R. E. Howard W. Hubbard L. D. Jackel Y. LeCun, B. Boser. Backpropagation applied to handwritten zip code recognition. 1989.

[7] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, Jan 2013.

[8] Masakazu; Katsuhiko Mori; Yusuke Mitari; Yuji Kaneda Matusugu. Subject independent facial expression recognition with robust face detection using a convolutional neural network. 16:1097–1105, 2003.

[9] Cho-Jui Hsieh James Demmel Kurt Keutzer Yang You, Zhao Zhang. Imagenet training in minutes. *Computer Vision and Pattern Recognition*, Sep 2017.

[10] Dan; Warnick-Sean West, Jeremy; Ventura. Spring research presentation: A theoretical foundation for inductive transfer. Aug 2007.

[11] Hong-Wei Ng, Viet Dung Nguyen, Vassilios Vonikakis, and Stefan Winkler. Deep learning for emotion recognition on small datasets using transfer learning. pages 443–449, 2015.

[12] Alexei A. Efros Minyoung Huh, Pulkit Agrawal. What makes imagenet good for transfer learning? *Computer Vision and Pattern Recognition*, Aug 2016.

[13] Luyang Yang Xiaolu Zhang. A mobile application for cat detection and breed recognition based on deep learning. 2018.

[14] Fang Wu WenBin Chen. A mobile application for dog breed recognition and detection based on deep learning.

[15] John Markoff. For web images, creating new technology to seek and find. *The New York Times*, Nov 2012.

[16] Hao Su Jonathan Krause Sanjeev Satheesh Sean Ma Zhiheng Huang Andrej Karpathy Aditya Khosla Michael Bernstein Alexander C. Berg Olga Russakovsky*, Jia Deng* and Li Fei-Fei. Imagenet large scale visual recognition challenge. *IJCV*, 2015.

[17] DeepLearning 0.1. LISA Lab. Convolutional neural networks (lenet) – deeplearning 0.1 documentation. Aug 2013.

[18] Dan; Ueli Meier; Jonathan Masci; Luca M. Gambardella; Jurgen Schmidhuber Ciresan. Flexible, high performance convolutional neural networks for image classification. 2:1237–1242, 2011.

[19] Andrew Zisserman Karen Simonyan. Very deep convolutional networks for large-scale image recognition. *Computer Vision and Pattern Recognition*, Apr 2015.

[20] Shaoqing Ren Jian Sun Kaiming He, Xiangyu Zhang. Deep residual learning for image recognition. *Computer Vision and Pattern Recognition*, Dec 2015.

[21] keras.io. Keras backends. Feb 2018.

[22] T. Poggio L. Rosasco. Machine learning: a regularization approach. *MIT*, Dec 2015.

[23] Christopher M. Bishop. Pattern recognition and machine learning. 2006.

[24] https://gist.github.com/baraldilorenzo/07d7802847aaad0a35d3.

[25] Knut Hinkelmann. Neural networks. *University of Applied Science Northwestern Switzerland*, page 7.

[26] G.V. Cybenko. Approximation by superpositions of a sigmoidal function. pages 303–314, 2006.

[27] Jimmy Diederik, Kingma; Ba. Adam: A method for stochastic optimization. *Computer Vision and Pattern Recognition*, 2014.

[28] Hanna Suominen, Tapio Pahikkala, and Tapio Salakoski. Critical points in assessing learning performance via cross-validation. pages 9–22, 2008.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[30] Federico; Michael Jones; Tomaso Poggio Girosi. Regularization theory and neural networks architectures. *Neural Computation*, 7:219–269, Mar 1995.

[31] A. D.; Grismer L. L.; Bell C. D.; Lailvaux S. P. Figueroa, A.; McKelvy. A species-level phylogeny of extant snakes with description of a new colubrid subfamily and genus. 11, 2016.

# Appendix

Source code link: https://github.com/haikuanl/Snake-Recognition
Dataset link: https://drive.google.com/open?id=1Q9MDFJspVWUA-zK3iURvf0MTd1Z7avXc
A brief demo of how the model can recognize a snake image: https://youtu.be/_ubcuBwIcw0

Running Environment:
Processor: Intel(R) Core(TM) i5-4210U CPU @ 2.40GHz
RAM: 4GB
System: Windows 10 1803