

MASTER OF TECHNOLOGY (INTELLIGENT SYSTEMS)

PROJECT REPORT (CA2)

Tutor: Dr Tan Jen Hong

Image Classification Using Deep Learning Techniques

TEAM MEMBERS

YE CHANGHE

LIM LI WEI

PREM s/o PIRAPALA CHANDRAN

Contents

Contents	2
EXECUTIVE SUMMARY	3
1.1) PROBLEM DESCRIPTION	4
1.2) CHALLENGES IN IMAGE RECOGNITION	4
2.1) STEPS TO BUILD DATASET.....	5
2.2) IMAGE CLASSIFICATION CLASSES	10
2.3) MANUAL IMAGE PRE-PROCESSING (AFTER DOWNLOAD)	12
2.4) RESIZING AND LABELLING OF IMAGES.....	13
2.5) DATA PARTITIONING	13
3.1) MODEL BUILDING	14
3.2) DESIGN OF THE MODELS	15
3.3) MODEL TUNING.....	21
3.4) MODEL PERFORMANCE.....	24
4.1) STEPS TO TEST MODEL.....	28
5.1) CHALLENGES	30
5.2) FINDINGS AND CONCLUSION.....	30
5.3) REFERENCE	31

EXECUTIVE SUMMARY

Machine learning (ML) is an application of artificial intelligence (AI) which provides systems the functionality to learn automatically so as to improve from experience without being explicitly programmed. The process of learning begins with observations of data, such as examples of footages, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide to learn without human intervention or assistance and adjust actions accordingly.

Deep Learning as sub-field in Machine Learning has been gaining traction in industrial applications requiring intensive, repetitive, process tasks one of which is computational image recognition tasks such as object detection, blob elimination, image synthesis, reconstruction and colorization. The problem of image recognition for classification is a good launch point to analyze model performance within ML to see how it can be improved by fine tuning the parameters in the algorithm to achieve a better performance. Deep learning can be either supervised where a labelled training data set is used to guide a neural network to learn and identify for itself the desired labels or categories so that it can recognize new data and sort it accordingly. Another variation is unsupervised learning where unlabeled data is clustered or sorted into categories based on features identified by the neural network.

In this project, we have applied deep learning to the domain of image recognition analysis so see how a neural network can be trained to recognize different images of animals using a set of training data such that it is able to recognize a new image to categories it accordingly based on what it has learnt previously from the extracted features of the training set of images.

Our findings show that we can achieve up to 84% accuracy using the Resnet Model. The report will elaborate further on the design of the models and provide further details on how the different strategies adopted influence the outcome.

1.1) PROBLEM DESCRIPTION

For this project, the problem of image classification can be briefly described as follows. In recognizing an image, there are potentially n number of categories in which a given image can be classified. Manual checking and classifying are very tedious and time-consuming process which becomes near impossible 10,000 or even 100,000, hence the need for automation of the process for quick labelling to their corresponding classes.

1.2) CHALLENGES IN IMAGE RECOGNITION

Before embarking on this project, it is important as in any project to be aware of the current challenges in such system designs to ensure due diligence is given to all aspects of the problem such that our approach may add value to the knowledge domain to build upon the current research in the field.

Based on a literature survey, these challenges can be broadly categorized into image granularity and variability of specific classes. Image granularity depends on the resolution of the image. Variability of class refer to the subdivided into inter-class and intra-class. Inter-class variability is the differences at a fine-grained level between specific or sub-classes within a labelled category which can be handled by a model/algorithm. The Intra-class variability is the difference between two images of the same class which we want out deep learning model to interpret to look similar quantitatively

For our project objectives, we are not concerned with inter-class variability as the animals are all so distinctly different and have enough features to be distinguished. What is more of a challenge is intra-class variability which we want to minimize.

2.1) STEPS TO BUILD DATASET

To build a repository of images for the training data, we used a command line python program from <https://google-images-download.readthedocs.io/en/latest/index.html> that allowed easy download of multiple sets of images based on a keyword. This allowed efficient pre-processing of the downloaded images to build a repository of data set to work with. It is necessary to pre-process the dataset to remove erroneous images that do not fit the required search. The pre-processing of the downloaded set requires firstly an understanding of the limitations of computer image recognition systems to ensure that the Deep Learning Machine Learning model will learn the attributes of the images in a meaningful way that will allow it to perform reliably and consistently on future images. How this is to be done is to remove artefacts from the images that may be a source of confusion to the system such as additional actors in the image such as humans or more than one animal, text and low contrast images that do not create enough distinguishing features.

For the choice of the dataset, we decided to apply the classification problem to the domain of image recognition and classification of animal types. This would serve as a good test of the performance of our models against a human recognition of animals as a task that comes second nature to us. In recognizing animals with our human vision, we look for certain attributes in the taxonomy of the morphology of the animal.

For the data set we are dealing with, the features are only at a perceptual level based on computer vision.

There is already existing literature of Machine Learning techniques applied to the domain of animal recognition such as automated photography systems for wildlife photography in the form of camera traps to allow wildlife study.

Our source code used to generate the dataset of training images can be accessed inside the submitted zip file

The steps taken to build the dataset are listed in detail here with screenshots of the corresponding screen output

No	Step
1	<p>Pre-requisition</p> <p>Open cmd and check python version</p> <p>> python --version</p>
	 <pre>命令提示符 Microsoft Windows [版本 10.0.18362.387] (c) 2019 Microsoft Corporation。保留所有权利。 C:\Users\65913>python --version Python 3.7.0 C:\Users\65913>_</pre>
2	<p>Install Google Images Download package via pip</p> <p>> pip install google_images_download</p>
	 <pre>命令提示符 Microsoft Windows [版本 10.0.18362.387] (c) 2019 Microsoft Corporation。保留所有权利。 C:\Users\65913>pip install google_images_download Collecting google_images_download Requirement already satisfied: selenium in c:\users\65913\appdata\local\programs\python\python37\lib\site-packages (from google_images_download) (3.141.0) Requirement already satisfied: urllib3 in c:\users\65913\appdata\local\programs\python\python37\lib\site-packages (from selenium->google_images_download) (1.25.3) Installing collected packages: google-images-download Successfully installed google-images-download-2.8.0 C:\Users\65913>_</pre>

3

Prepare the parameters for image downloading

3.1 Define the directory path of image

```
# import the necessary packages
import ...

warnings.filterwarnings("ignore")

from google_images_download import google_images_download

response = google_images_download.googleimagesdownload()

























download_path = 'D://iss//S2//CA2//Dataset'
```

(please modify it according to your local path)

3.2 Define the animal parameter for API call

```
arguments = [
    {
        "keywords": "fish",
        "suffix_keywords": "white,black",
        "limit": 70,
        "output_directory": download_path,
        "image_directory": "fish",
        "print_urls": "true",
        "prefix": "fish",
        "no_numbering": "true"
    },
]
```

keywords	k	Denotes the keywords/key phrases you want to search for. For more than one keywords, wrap it in single quotes. Tips: If you simply type the keyword, Google will best try to match it If you want to search for exact phrase, you can wrap the keywords in double quotes ("") If you want to search to contain either of the words provided, use OR between the words. If you want to explicitly not want a specific word use a minus sign before the word (-)
suffix_keywords	sk	Denotes additional words added after main keyword while making the search query. The final search query would be: <keyword> <suffix keyword> So, for example, if the keyword is 'car' and suffix_keyword is 'red,yellow,blue', it will search and download images for 'car red', 'car yellow' and 'car blue' individually
limit	l	Denotes number of images that you want to download. You can specify any integer value here. It will try and get all the images that it finds in the google image search page. If this value is not specified, it defaults to 100. Note: In case of occasional errors while downloading images, you could get less than 100 (if the limit is set to 100)
output_directory	o	Allows you specify the main directory name in which the images are downloaded. If not specified, it will default to 'downloads' directory. This directory is located in the path from where you run this code The directory structure would look like: <output_directory><image_directory><images>
print_urls	p	Print the URLs of the images on the console. These image URLs can be used for debugging purposes This argument does not take any value. Just add '-print_urls' or '-p' in your query.
prefix	pr	A word that you would want to prefix in front of actual image name. This feature can be used to rename files for image identification purpose.
no_numbering	nn	When you specify this argument, the script does not add ordered numbering as prefix to the images it downloads If this argument is not specified, the images are numbered in order in which they are downloaded This argument does not take any value. Just add '-no_numbering' or '-nn' in your query.

	 fish-0.jpg  fish-1.jpg  fish-2.jpg  fish-3.jpg  fish-4.jpg  fish-5.jpg  fish-6.jpg  fish-7.jpg  fish-8.jpg  fish-9.jpg  fish-10.jpg  fish-11.jpg  fish-12.jpg  fish-13.jpg  fish-14.jpg  fish-15.jpg  fish-16.jpg  fish-17.jpg  fish-18.jpg  fish-19.jpg  fish-20.jpg  fish-21.jpg  fish-22.jpg  fish-23.jpg	2019/9/22 19:54 2019/9/22 20:08 2019/9/21 17:42 2019/9/22 20:08 2019/9/21 17:54 2019/9/21 17:48 2019/9/21 17:43 2019/9/21 17:45 2019/9/21 17:21 2019/9/21 17:28 2019/9/21 17:39 2019/9/21 17:33 2019/9/21 17:49 2019/9/22 19:54 2019/9/21 17:17 2019/9/22 20:07 2019/9/21 17:14 2019/9/21 17:05 2019/9/22 19:55 2019/9/21 17:23 2019/9/21 17:49 2019/9/21 17:12 2019/9/21 17:04 2019/9/21 17:40	JPG 文件 JPG 文件	737 KB 118 KB 658 KB 295 KB 21 KB 24 KB 42 KB 268 KB 153 KB 130 KB 51 KB 59 KB 18 KB 170 KB 33 KB 296 KB 23 KB 191 KB 168 KB 57 KB 377 KB 26 KB 126 KB 116 KB
6	Post-process 6.1 Manually look through the image and delete the unrelated images 6.2 If the number of images less than 1000 modify the parameters of the script and trigger again until enough images are pooled to repeat the post process			

Please see the Dataset_prepare.py inside the submitted zip file

2.2) IMAGE CLASSIFICATION CLASSES

- Bird



- Dog



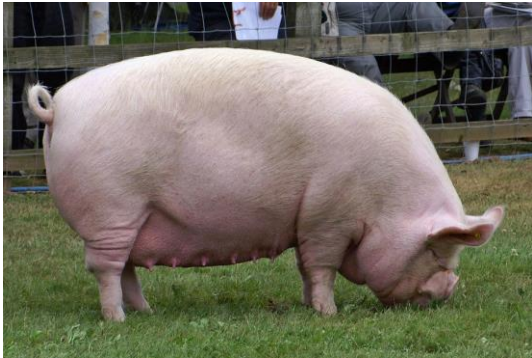
- Horse



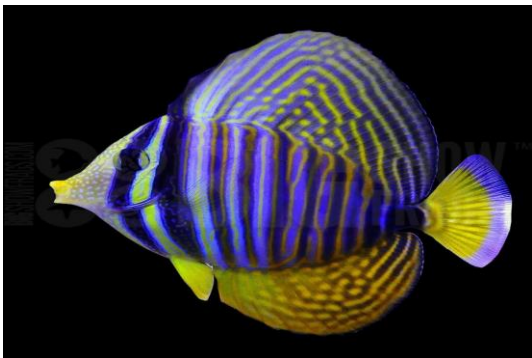
- Cat



- Pig



- Fish



2.3) MANUAL IMAGE PRE-PROCESSING (AFTER DOWNLOAD)

This step is necessary before the training of the data. As with any scrapped images, manual cleaning is needed to remove any wrong/misleading data.

No	Animal	Features to look for	Features to exclude
1	Bird	<ul style="list-style-type: none"> • Profile of beak with the head • Eye features • Wing Profile 	In an unnatural setting such as in a cage more than one Object any noise in the form of artefacts such as text, other actors (e.g. humans, other animals) Largely Incomplete Subject Misleading Pictures (e.g. Cat that looks like a Dog)
2	Dog	<ul style="list-style-type: none"> • Dog Facial Profile • More than Half the Body shown 	
3	Horse	<ul style="list-style-type: none"> • Long Legs, Mane, long face. 	
4	Cat	<ul style="list-style-type: none"> • Cat Facial Profile • More than Half the Body shown 	
5	Pig	<ul style="list-style-type: none"> • Big Size, Short Legs, Snout 	
6	Fish	<ul style="list-style-type: none"> • Water, Fish Body Profile (Fins, Tail etc.) 	

Emphasis is on cat and dog as they are the most likely to be misclassified because they are somewhat similar looking. For the rest of the class, their biological features are distinct enough to have lesser search rules imposed on them.

2.4) RESIZING AND LABELLING OF IMAGES

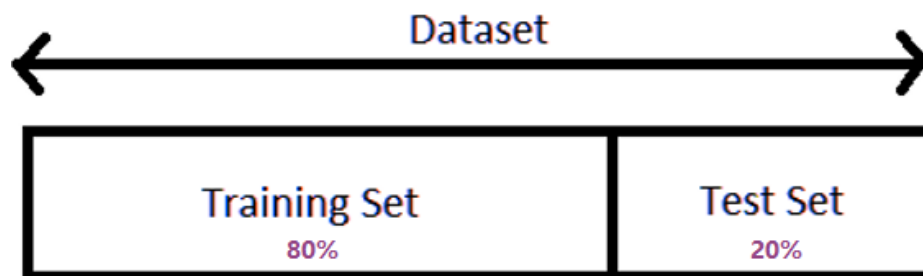
Before the design of the models, the pictures need to undergo a resize process so that all the images are of the same size. In this assignment, we resize all images to 64 x 64 Pixels before we feed to our model.

The images are then labelled by their respective classes. There is a separate array that is exported into .mat file with just the 6 classes name inside and appended to the data during partitioning.

These pictures will be loaded into a NumPy array and saved to a .mat data file. The models will read the images from the .mat file and do the classification.

2.5) DATA PARTITIONING

The image data (Total 6382 images) are partitioned into training and testing data (80% Training 20% Testing) for the making of this model. After building the CNN model training data will be used to train the model, and for the final verification and testing metrics we will use the testing data.



3.1) MODEL BUILDING

In this assignment, models are built progressively, starting from the bare basic CNN model (Convolution and MaxPooling only). A summary is as shown:

- ModelV1 (3 Rounds of Conv2d, MaxPooling2d, Flatten and Dense with 6 classes)

- ModelV1-2 (V1 with Image Augmentation Techniques Applied)
 - ZCA whitening
 - Random rotations
 - Random flips
 - Random Translations

- ModelV2 (V1-2 with Overfitting Prevention Techniques Applied)
 - Regularize the weights
 - Add dropout between layers
 - Apply He initialization
 - Apply batch normalization
 - Apply SGD optimizer
 - Reduce batch size

- ModelV3 (Resnet with Techniques from V2 and V1-3 Applied)

Please see the Final_CA2.ipynb inside the submitted zip file

3.2) DESIGN OF THE MODELS

ModelV1

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 64, 64, 3)]	0
conv2d (Conv2D)	(None, 64, 64, 32)	896
max_pooling2d (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_1 (Conv2D)	(None, 32, 32, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 128)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 1024)	8389632
dense_1 (Dense)	(None, 6)	6150
Total params: 8,489,030		
Trainable params: 8,489,030		
Non-trainable params: 0		

This is the most basic CNN Model with 3 iterations of convolution and max pooling. 3x3 Kernel is used for 2dConv and 2x2 is used for Max Pooling.

In these 3 iterations, enough feature should be extracted during the convolutions to classify the images correctly. All convolution layers will use the “ReLU” activation to prevent vanishing gradient from using “Sigmoid”, while the last hidden layer will use the “softmax” activation.

This model is expected to face the most problem and yield the least accuracy since we are not applying any prior techniques to enhance the model.

ModelV2

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 64, 64, 3)	0
conv2d (Conv2D)	(None, 64, 64, 32)	896
batch_normalization_v1 (Batch Normalization)	(None, 64, 64, 32)	128
activation (Activation)	(None, 64, 64, 32)	0
dropout (Dropout)	(None, 64, 64, 32)	0
max_pooling2d (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9248
batch_normalization_v1_1 (Batch Normalization)	(None, 32, 32, 32)	128
activation_1 (Activation)	(None, 32, 32, 32)	0
dropout_1 (Dropout)	(None, 32, 32, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 32)	9248
batch_normalization_v1_2 (Batch Normalization)	(None, 16, 16, 32)	128
activation_2 (Activation)	(None, 16, 16, 32)	0
dropout_2 (Dropout)	(None, 16, 16, 32)	0
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 32)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 1024)	2098176
dense_1 (Dense)	(None, 6)	6150
Total params: 2,124,102		
Trainable params: 2,123,910		
Non-trainable params: 192		

In ModelV2, extra steps are taken to ensure that overfitting does not become an issue.

For every iteration of convolution and MaxPooling, we also do a dropout and batch normalization. Also, He initialization and L2 Regularization is included before convolution or dense is performed. This will help in very deep models in eliminating further vanishing gradient issue, which we will be doing in V3.

ModelV3 (Resnet)

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 64, 64, 3)]	0	
Inpt_conv (Conv2D)	(None, 64, 64, 16)	448	input_1[0][0]
Inpt_bn (BatchNormalization)	(None, 64, 64, 16)	64	Inpt_conv[0][0]
Inpt_relu (Activation)	(None, 64, 64, 16)	0	Inpt_bn[0][0]
Stg1_Blkl1_Res1_conv (Conv2D)	(None, 64, 64, 16)	2320	Inpt_relu[0][0]
Stg1_Blkl1_Res1_bn (BatchNormali	(None, 64, 64, 16)	64	Stg1_Blkl1_Res1_conv[0][0]
Stg1_Blkl1_Res1_relu (Activation	(None, 64, 64, 16)	0	Stg1_Blkl1_Res1_bn[0][0]
Stg1_Blkl1_Res2_conv (Conv2D)	(None, 64, 64, 16)	2320	Stg1_Blkl1_Res1_relu[0][0]
Stg1_Blkl1_Res2_bn (BatchNormali	(None, 64, 64, 16)	64	Stg1_Blkl1_Res2_conv[0][0]
Stg1_Blkl1_add (Add)	(None, 64, 64, 16)	0	Inpt_relu[0][0] Stg1_Blkl1_Res2_bn[0][0]
Stg1_Blkl1_relu (Activation)	(None, 64, 64, 16)	0	Stg1_Blkl1_add[0][0]
Stg1_Blkl2_Res1_conv (Conv2D)	(None, 64, 64, 16)	2320	Stg1_Blkl1_relu[0][0]
Stg1_Blkl2_Res1_bn (BatchNormali	(None, 64, 64, 16)	64	Stg1_Blkl2_Res1_conv[0][0]
Stg1_Blkl2_Res1_relu (Activation	(None, 64, 64, 16)	0	Stg1_Blkl2_Res1_bn[0][0]
Stg1_Blkl2_Res2_conv (Conv2D)	(None, 64, 64, 16)	2320	Stg1_Blkl2_Res1_relu[0][0]
Stg1_Blkl2_Res2_bn (BatchNormali	(None, 64, 64, 16)	64	Stg1_Blkl2_Res2_conv[0][0]
Stg1_Blkl2_add (Add)	(None, 64, 64, 16)	0	Stg1_Blkl1_relu[0][0] Stg1_Blkl2_Res2_bn[0][0]
Stg1_Blkl2_relu (Activation)	(None, 64, 64, 16)	0	Stg1_Blkl2_add[0][0]
Stg1_Blkl3_Res1_conv (Conv2D)	(None, 64, 64, 16)	2320	Stg1_Blkl2_relu[0][0]
Stg1_Blkl3_Res1_bn (BatchNormali	(None, 64, 64, 16)	64	Stg1_Blkl3_Res1_conv[0][0]

Stg1_Bl3_Res1_relu (Activation)	(None, 64, 64, 16)	0	Stg1_Bl3_Res1_bn[0][0]
Stg1_Bl3_Res2_conv (Conv2D)	(None, 64, 64, 16)	2320	Stg1_Bl3_Res1_relu[0][0]
Stg1_Bl3_Res2_bn (BatchNormali	(None, 64, 64, 16)	64	Stg1_Bl3_Res2_conv[0][0]
Stg1_Bl3_add (Add)	(None, 64, 64, 16)	0	Stg1_Bl2_relu[0][0] Stg1_Bl3_Res2_bn[0][0]
Stg1_Bl3_relu (Activation)	(None, 64, 64, 16)	0	Stg1_Bl3_add[0][0]
Stg2_Bl1_Res1_conv (Conv2D)	(None, 32, 32, 32)	4640	Stg1_Bl3_relu[0][0]
Stg2_Bl1_Res1_bn (BatchNormali	(None, 32, 32, 32)	128	Stg2_Bl1_Res1_conv[0][0]
Stg2_Bl1_Res1_relu (Activation)	(None, 32, 32, 32)	0	Stg2_Bl1_Res1_bn[0][0]
Stg2_Bl1_Res2_conv (Conv2D)	(None, 32, 32, 32)	9248	Stg2_Bl1_Res1_relu[0][0]
Stg2_Bl1_lin_conv (Conv2D)	(None, 32, 32, 32)	544	Stg1_Bl3_relu[0][0]
Stg2_Bl1_Res2_bn (BatchNormali	(None, 32, 32, 32)	128	Stg2_Bl1_Res2_conv[0][0]
Stg2_Bl1_add (Add)	(None, 32, 32, 32)	0	Stg2_Bl1_lin_conv[0][0] Stg2_Bl1_Res2_bn[0][0]
Stg2_Bl1_relu (Activation)	(None, 32, 32, 32)	0	Stg2_Bl1_add[0][0]
Stg2_Bl2_Res1_conv (Conv2D)	(None, 32, 32, 32)	9248	Stg2_Bl1_relu[0][0]
Stg2_Bl2_Res1_bn (BatchNormali	(None, 32, 32, 32)	128	Stg2_Bl2_Res1_conv[0][0]
Stg2_Bl2_Res1_relu (Activation)	(None, 32, 32, 32)	0	Stg2_Bl2_Res1_bn[0][0]
Stg2_Bl2_Res2_conv (Conv2D)	(None, 32, 32, 32)	9248	Stg2_Bl2_Res1_relu[0][0]
Stg2_Bl2_Res2_bn (BatchNormali	(None, 32, 32, 32)	128	Stg2_Bl2_Res2_conv[0][0]
Stg2_Bl2_add (Add)	(None, 32, 32, 32)	0	Stg2_Bl1_relu[0][0] Stg2_Bl2_Res2_bn[0][0]
Stg2_Bl2_relu (Activation)	(None, 32, 32, 32)	0	Stg2_Bl2_add[0][0]
Stg2_Bl3_Res1_conv (Conv2D)	(None, 32, 32, 32)	9248	Stg2_Bl2_relu[0][0]

Stg2_Bl3k3_Res1_bn (BatchNormali	(None, 32, 32, 32)	128	Stg2_Bl3k3_Res1_conv[0][0]
Stg2_Bl3k3_Res1_relu (Activation	(None, 32, 32, 32)	0	Stg2_Bl3k3_Res1_bn[0][0]
Stg2_Bl3k3_Res2_conv (Conv2D)	(None, 32, 32, 32)	9248	Stg2_Bl3k3_Res1_relu[0][0]
Stg2_Bl3k3_Res2_bn (BatchNormali	(None, 32, 32, 32)	128	Stg2_Bl3k3_Res2_conv[0][0]
Stg2_Bl3k3_add (Add)	(None, 32, 32, 32)	0	Stg2_Bl3k2_relu[0][0] Stg2_Bl3k3_Res2_bn[0][0]
Stg2_Bl3k3_relu (Activation)	(None, 32, 32, 32)	0	Stg2_Bl3k3_add[0][0]
Stg3_Bl1k1_Res1_conv (Conv2D)	(None, 16, 16, 64)	18496	Stg2_Bl3k3_relu[0][0]
Stg3_Bl1k1_Res1_bn (BatchNormali	(None, 16, 16, 64)	256	Stg3_Bl1k1_Res1_conv[0][0]
Stg3_Bl1k1_Res1_relu (Activation	(None, 16, 16, 64)	0	Stg3_Bl1k1_Res1_bn[0][0]
Stg3_Bl1k1_Res2_conv (Conv2D)	(None, 16, 16, 64)	36928	Stg3_Bl1k1_Res1_relu[0][0]
Stg3_Bl1k1_lin_conv (Conv2D)	(None, 16, 16, 64)	2112	Stg2_Bl3k3_relu[0][0]
Stg3_Bl1k1_Res2_bn (BatchNormali	(None, 16, 16, 64)	256	Stg3_Bl1k1_Res2_conv[0][0]
Stg3_Bl1k1_add (Add)	(None, 16, 16, 64)	0	Stg3_Bl1k1_lin_conv[0][0] Stg3_Bl1k1_Res2_bn[0][0]
Stg3_Bl1k1_relu (Activation)	(None, 16, 16, 64)	0	Stg3_Bl1k1_add[0][0]
Stg3_Bl1k2_Res1_conv (Conv2D)	(None, 16, 16, 64)	36928	Stg3_Bl1k1_relu[0][0]
Stg3_Bl1k2_Res1_bn (BatchNormali	(None, 16, 16, 64)	256	Stg3_Bl1k2_Res1_conv[0][0]
Stg3_Bl1k2_Res1_relu (Activation	(None, 16, 16, 64)	0	Stg3_Bl1k2_Res1_bn[0][0]
Stg3_Bl1k2_Res2_conv (Conv2D)	(None, 16, 16, 64)	36928	Stg3_Bl1k2_Res1_relu[0][0]
Stg3_Bl1k2_Res2_bn (BatchNormali	(None, 16, 16, 64)	256	Stg3_Bl1k2_Res2_conv[0][0]
Stg3_Bl1k2_add (Add)	(None, 16, 16, 64)	0	Stg3_Bl1k1_relu[0][0] Stg3_Bl1k2_Res2_bn[0][0]

Stg3_Bl3k2_relu (Activation)	(None, 16, 16, 64)	0	Stg3_Bl3k2_add[0][0]
Stg3_Bl3k3_Res1_conv (Conv2D)	(None, 16, 16, 64)	36928	Stg3_Bl3k2_relu[0][0]
Stg3_Bl3k3_Res1_bn (BatchNormali	(None, 16, 16, 64)	256	Stg3_Bl3k3_Res1_conv[0][0]
Stg3_Bl3k3_Res1_relu (Activation	(None, 16, 16, 64)	0	Stg3_Bl3k3_Res1_bn[0][0]
Stg3_Bl3k3_Res2_conv (Conv2D)	(None, 16, 16, 64)	36928	Stg3_Bl3k3_Res1_relu[0][0]
Stg3_Bl3k3_Res2_bn (BatchNormali	(None, 16, 16, 64)	256	Stg3_Bl3k3_Res2_conv[0][0]
Stg3_Bl3k3_add (Add)	(None, 16, 16, 64)	0	Stg3_Bl3k2_relu[0][0] Stg3_Bl3k3_Res2_bn[0][0]
Stg3_Bl3k3_relu (Activation)	(None, 16, 16, 64)	0	Stg3_Bl3k3_add[0][0]
AvgPool (AveragePooling2D)	(None, 2, 2, 64)	0	Stg3_Bl3k3_relu[0][0]
flatten (Flatten)	(None, 256)	0	AvgPool[0][0]
dense (Dense)	(None, 6)	1542	flatten[0][0]
=====			
Total params: 275,334			
Trainable params: 273,958			
Non-trainable params: 1,376			

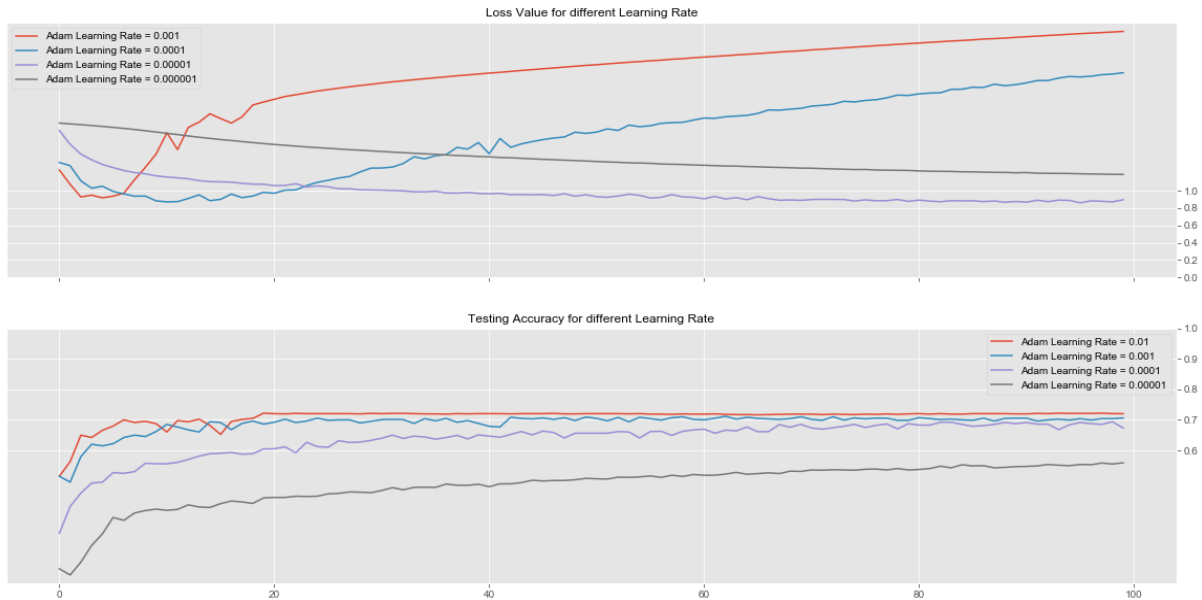
Lastly, we applied the basic deep Residual Network which was arguably the most groundbreaking work in the computer vision/deep learning community in the last few years. The ResNet model is also done with Scheduled Learning Rate and Small Batches to improve the accuracy further.

This final model will be expected to produce the best accuracy as it incorporates all the techniques learnt so far.

3.3) MODEL TUNING

Optimizer

For ModelV1 we use the adam optimizer and train the model with different learning rate:



Learning Rate	Best Accuracy
0.001	72.20%
0.0001	71.18%
0.00001	69.38%
0.000001	55.91%

As we see, although using 0.001 as the learning rate can get the highest accuracy, from the loss value it is obviously overfitting the data, so decide to choose 0.0001 as the learning rate for optimizer afterward.

Image Generator

After we apply data augmentation technique to generate more image can be feed to our model, they still some parameters we can test:

- ZCA whitening
- Random rotations
- Random flip
- Image shift

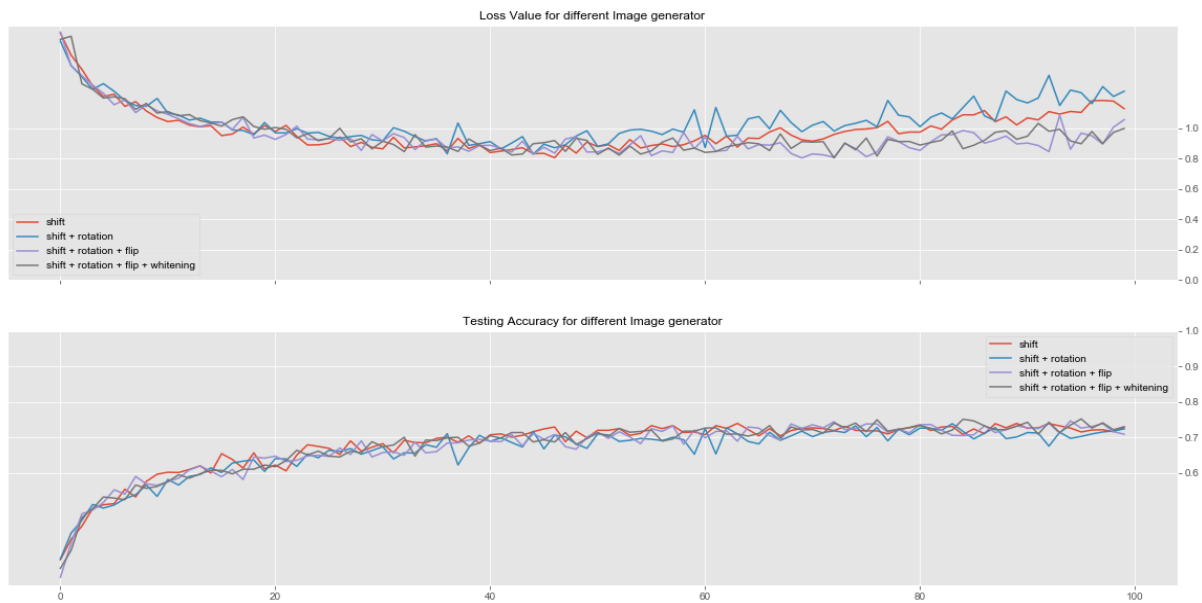


Image Generator	Best Accuracy
Shift	73.92%
Shift + rotation	73.84%
Shift + rotation + flip	74.63%
Shift + rotation + flip + whitening	75.18%

As we are stacking the parameters, we can see the accuracy did prove and can have best performance when we are using all four features.

L2 Regularizer

Here we mainly tuning for L2 regularizer inside our modified ModelV2:

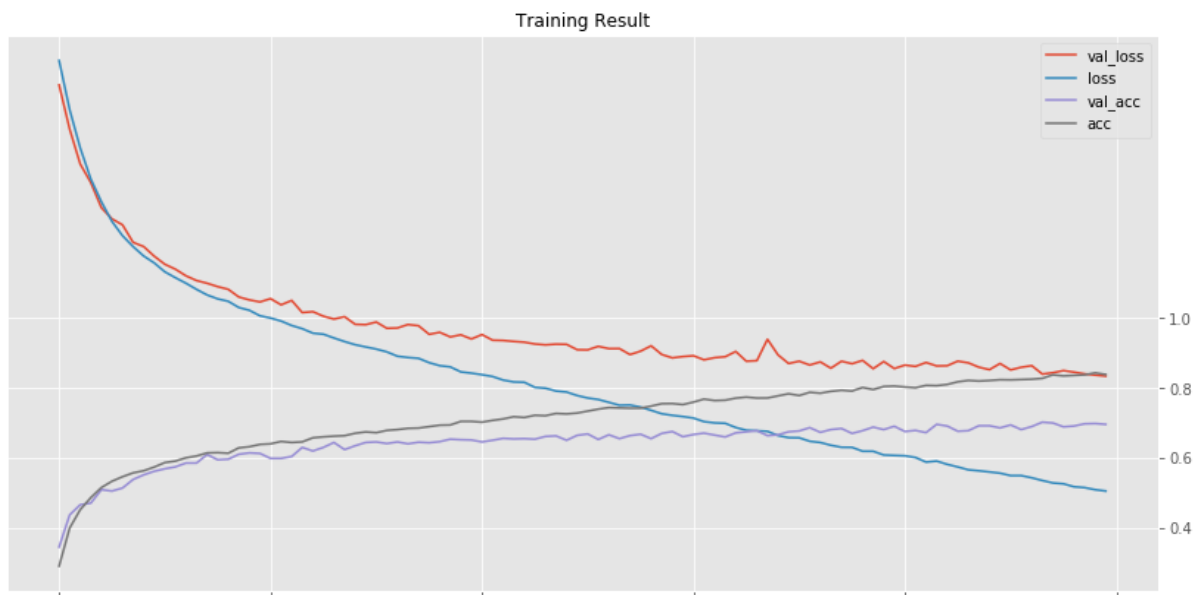


L2 Regularizer	Best Accuracy
0.01	70.01%
0.001	67.42%
0.0001	67.42%
0.00001	67.74%

Form the diagram, 0.01 can get the best performance but it is not so stable for training, so we picked 0.00001.

3.4) MODEL PERFORMANCE

ModelV1



Best accuracy (on testing dataset): 70.24%

	precision	recall	f1-score	support
bird	0.7189	0.7027	0.7107	222
cat	0.6609	0.7476	0.7016	206
dog	0.6114	0.4672	0.5297	229
fish	0.8020	0.8449	0.8229	187
horse	0.6377	0.6875	0.6617	192
pig	0.7661	0.7884	0.7771	241
accuracy			0.7024	1277
macro avg	0.6995	0.7064	0.7006	1277
weighted avg	0.6992	0.7024	0.6984	1277

Total 1277	Predict bird	Predict cat	Predict dog	Predict fish	Predict horse	Predict pig
True bird	156	14	11	20	14	7
True cat	9	154	24	5	6	8
True dog	15	38	107	7	34	28
True fish	16	3	5	158	4	1
True horse	14	15	13	4	132	14
True pig	7	9	15	3	17	190

Data augmentation



Best accuracy (on testing dataset): 75.49%

	precision	recall	f1-score	support
bird	0.7017	0.7523	0.7261	222
cat	0.7188	0.7816	0.7488	206
dog	0.7371	0.6245	0.6761	229
fish	0.8929	0.8021	0.8451	187
horse	0.7202	0.7240	0.7221	192
pig	0.7846	0.8465	0.8144	241
accuracy			0.7549	1277
macro avg	0.7592	0.7551	0.7554	1277
weighted avg	0.7572	0.7549	0.7543	1277

Total 1277	Predict bird	Predict cat	Predict dog	Predict fish	Predict horse	Predict pig
True bird	167	12	11	13	11	8
True cat	15	161	16	1	7	6
True dog	17	30	143	2	15	22
True fish	23	5	0	150	4	2
True horse	8	10	16	1	139	18
True pig	8	6	8	1	14	204

ModelV2

Best accuracy (on testing dataset): 73.92%

	precision	recall	f1-score	support
bird	0.6575	0.7523	0.7017	222
cat	0.7269	0.8010	0.7621	206
dog	0.7546	0.5371	0.6276	229
fish	0.8060	0.8663	0.8351	187
horse	0.7758	0.6667	0.7171	192
pig	0.7453	0.8257	0.7835	241
accuracy			0.7392	1277
macro avg	0.7443	0.7415	0.7378	1277
weighted avg	0.7422	0.7392	0.7354	1277

Total 1277	Predict bird	Predict cat	Predict dog	Predict fish	Predict horse	Predict pig
True bird	167	9	11	21	6	8
True cat	14	165	8	5	6	8
True dog	17	39	123	3	12	35
True fish	21	3	0	162	1	0
True horse	15	8	17	7	128	17
True pig	20	3	4	3	12	199

ModelV3

Best accuracy (on testing dataset): 83.48%

bird	0.7619	0.8649	0.8101	222
cat	0.8918	0.8398	0.8650	206
dog	0.8593	0.7467	0.7991	229
fish	0.9133	0.8449	0.8778	187
horse	0.7193	0.8542	0.7810	192
pig	0.9004	0.8631	0.8814	241
accuracy			0.8348	1277
macro avg	0.8410	0.8356	0.8357	1277
weighted avg	0.8422	0.8348	0.8360	1277

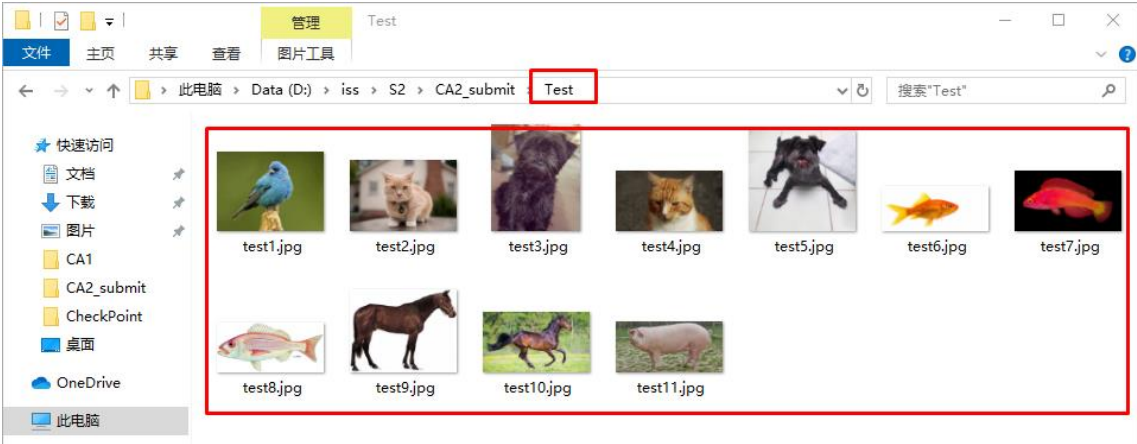
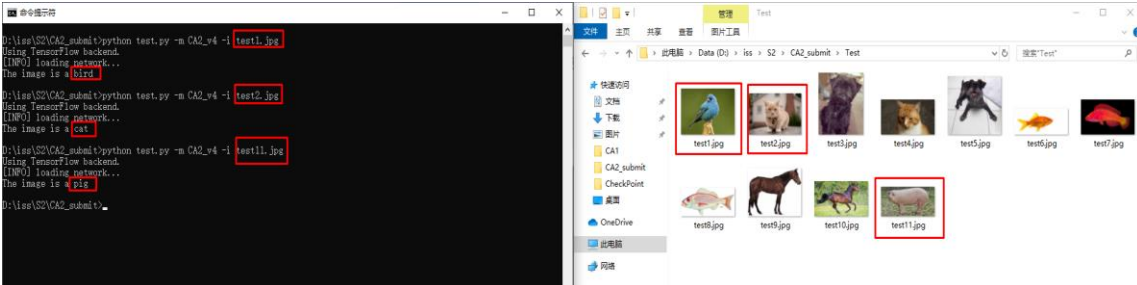
Total 1277	Predict bird	Predict cat	Predict dog	Predict fish	Predict horse	Predict pig
True bird	192	3	5	7	13	2
True cat	12	173	9	2	7	3
True dog	11	9	171	1	26	11
True fish	23	0	0	158	4	2
True horse	7	3	11	2	164	5
True pig	7	6	3	3	14	208

As expected, this model yields the best accuracy. However it is not as high as expected. This is due to the nature of the classification data; Animals can be misclassified easily.

4.1) STEPS TO TEST MODEL

The steps taken to load the final model and classify the image download from internet by one python command line

No	Step
1	<p>Pre-requisition</p> <p>1.1 Check TensorFlow and Keras are installed > pip list</p>  <pre> D:\iss\S2\CA2_submit>pip list Package Version ----- absl-py 0.7.1 astor 0.8.0 cyclor 0.10.0 gast 0.2.2 google-images-download 2.8.0 google-pasta 0.1.7 graphviz 0.12 grpcio 1.23.0 h5py 2.9.0 inline 0.0.1 input 0.0.0 johlib 0.13.2 Keras 2.3.0 Keras-Applications 1.0.8 Keras-Preprocessing 1.1.0 kiwisolver 1.1.0 Markdown 3.1.1 matplotlib 3.1.1 metrics 0.3.3 nltk 3.2.4 numpy 1.17.0 opencv-python 4.1.1.26 pandas 0.25.0 pathlib2 2.3.4 pathspec 0.5.5 pip 19.2.3 preprocessing 0.1.13 protobuf 3.9.1 pydot 1.4.1 Pygments 2.2.0 pyparsing 2.4.2 python-dateutil 2.8.0 pytz 2019.2 PyYAML 5.1.2 scikit-learn 0.21.3 scipy 1.3.1 selenium 3.141.0 setuptools 39.0.1 six 1.12.0 sphinx-rtd-theme 0.2.4 tensorboard 1.14.0 tensorflow 1.14.0 tensorflow-estimator 1.14.0 </pre> <p>1.2 Download the test pictures to Test folder which is under same directory with the test.py</p>

	
2	<p>Run the test script > <code>python test.py -m CA2_v4 -i [image name]</code></p> <div data-bbox="284 798 1414 1081"></div>

Please see the test.py inside the submitted zip file

5.1) CHALLENGES

After we download the data set, how to pick up the valid image into dataset is an issue since we need to manually filter it out and we need make a rule to keep the consistency how do we delete it.

Computer resource may get in the way as the model complexity increases, thus we can get online free platform like Collab which liberate the GPU utilization of our local machine and improve the efficiency of training model. In fact, training can be done over Collab and Locally at the same time to save time on tuning the Hyperparameters.

Tuning model could be a very time-consuming and tedious steps to meet the best performance, although there are many best practises online, it may not suitable for our solution

Overfitting always a very big problem to conquer when you train a deep learning model, we need to know what the key is to generalize our model be able to classify massive real-world scenario.

5.2) FINDINGS AND CONCLUSION

Data augmentation is a very good way to improve our model performance, increase the dataset can be useful but it may not suitable for several scenario such as B scan image

Sometimes if the model hits a bottleneck, we can try training with the SGD optimizer instead of ADAM.

Resnet is a very powerful weapon to help us conquer the challenge. In this assignment we are only using the basic Resnet model, but nowadays, there are other variants being derived from the base model, which might be better.

5.3) REFERENCE

- <https://towardsdatascience.com/deep-learning-for-image-classification-why-its-challenging-where-we-ve-been-and-what-s-next-93b56948fcef>
- <https://courses.lumenlearning.com/suny-biology2xmaster/chapter/features-used-to-classify-animals/>
- <https://medium.com/coinmonks/automated-animal-identification-using-deep-learning-techniques-41039f2a994d>