Animal classification using CNN, ResNet, MobileNet

1. Introduction of the task and the baseline model

Our task is to use machine learning models to perform image classification. The objective of the task is to examine different Convolutional Neural Network (CNN) models and fine-tune the hyperparameters to get the best accuracy while keeping a low computational cost.

Dataset

The dataset contains 6270 RBG images which are divided into 151 labelled classes. The size of each image is 224 pixels wide and 224 pixels high.

Training, Validation, Testing split

The dataset is split into training, validation, and testing set in a ratio of 85%, 5% and 10% respectively. The training, validation, and testing set contain 5330, 313 and 627 images respectively.

Evaluation metrics

The evaluation metrics is the accuracy over top 3 prediction. The output of a CNN model is a set of probability representing the predicted class labels for the input data. If one of the top 3 highest probability predicted classes is the same as the label, we consider it as a correct prediction.

Efficiency calculation

Apart from the accuracy, we also want to evaluate our model base on the computational power. Therefore, we also calculate the model efficiency. The efficiency is defined as the ratio of accuracy to Gflops. Gflops is Giga Floating-Point Operations Per Second, it quantifies the computational power required by calculating the floating-point arithmetic operations within one second. The formula for the efficiency is as followed:

Efficiency = Testing accuracy / Gflops (% per Gflops)

Data Augmentation

- transforms.Resize(112)
- transforms.RandomHorizontalFlip()
- transforms.CenterCrop(112),
- transforms.ToTensor(),
- transforms.Normalize((0.488), (0.2172))
- transforms.RandomRotation(degrees=15)

We use 6 different data augmentation techniques. Resize(112) will resize the image to the width and height of 112 pixels. RandomHorizontalFlip() randomly flips the input image horizontally (left to right) with a default probability (0.5). CenterCrop(112) crops the center portion of the original 224×224 image to a square of size of (112, 112). ToTensor() converts PIL images with range (0 - 255) to tensor with range (0 - 1). Normalize((0.488), (0.2172)) takes in a 3-channel Tensor and normalizes each channel by the input mean and standard deviation for that channel. RandomRotation(degrees=15) applies a random rotation to an image by 15 degrees.

Baseline model architecture

The baseline model is a simple CNN model. Table 1 shows the architecture of the baseline model. It consists of 4 convolutional layer, 4 max pooling layers and 1 fully connected layer. The default hyperparameters are as followed:

- number of epochs = 10
- learning rate = 0.001
- optimizer = Adam

Table 1: The Baseline CNN model architecture

Layer name	Type	Filter size	Output size	Stride
Input	Image		112 x 112 x 3	
conv1	Convolution	5 x 5 x 64	108 x 108 x 64	1
pool1	Max Pooling	2 x 2	54 x 54 x 64	2
conv2	Convolution	3 x 3 x 128	52 x 52 x 128	1
pool2	Max Pooling	2 x 2	26 x 26 x 128	2
conv3	Convolution	3 x 3 x 128	24 x 24 x 128	1
pool3	Max Pooling	2 x 2	12 x 12 x 128	2
conv4	Convolution	3 x 3 x 128	10 x 10 x 128	1
pool4	Max Pooling	2 x 2	5 x 5 x 128	2
fc1	Fully Connected		151	

The baseline model has validation and testing accuracy of 36.8% and 38.8% respectively. Figure 1 shows the validation accuracy, training loss and validation loss of the baseline model with default hyperparameters. We can observe an overfitting problem in the loss curve graph. The overfitting problem will be addressed in the experiment section. The Gflops of the baseline model is 0.69G. The efficiency is 56% per Gflops.

Efficiency = Testing accuracy / Gflops = 38.8 / 0.69 = 56% per Gflops

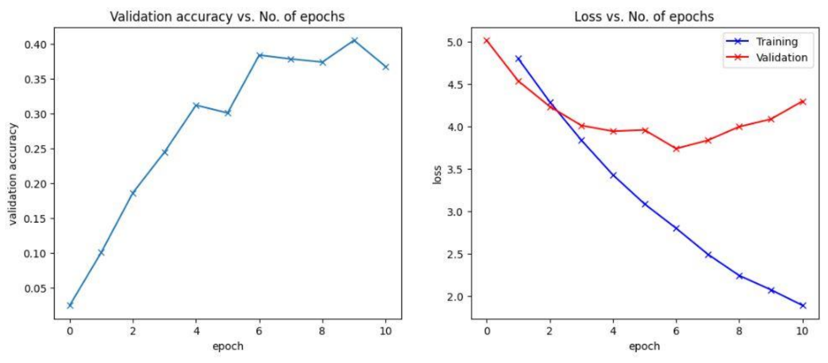


Figure 1: The validation accuracy (left), training and validation loss (right) of baseline model with default hyperparameters

2. Modification of the system to improve performance

We perform 6 experiments to improve the model performance by modifying the systems.

Experiment 1: Baseline model with different learning rate and number of epochs:

In this experiment, we perform hyperparameter tuning to the baseline model. We train the model with different learning rate (0.0001, 0.001, 0.01) and different number of epochs (10, 30, 50). As shown in Table 2, the best learning rate and number of epochs are 0.0001 and 30 respectively, with a validation accuracy of 48.7%. Figure 2 shows the validation accuracy, training loss and validation loss of the baseline model with the best learning rate (0.0001) and number of epochs (30). We can observe an overfitting problem in the loss curve graph. This drives our experiment 2 of adding dropout out layer and batch normalisation to the baseline model in order to reduce overfitting. The Gflops for the baseline model is 0.69G. The testing accuracy of the best learning rate (0.0001) and number of epoch (30) is 47.4%. The efficiency is 69% per Gflops.

Table 2: The validation accuracy of baseline model with different learning rate and number of epochs

Model	Batch	Epochs	Learning rate	Dropout	Batch Normalisation	Optimizer	Validation Accuracy (%)	Gflops
Baseline Model	16	10	0.0001	No	No	Adam	45.9	0.69G
			0.001				36.8	
			0.01				1.3	
		30	0.0001				**48.7**	
			0.001				41.8	
			0.01				1.3	
		50	0.0001				47.5	
			0.001				43.2	
			0.01				1.3	

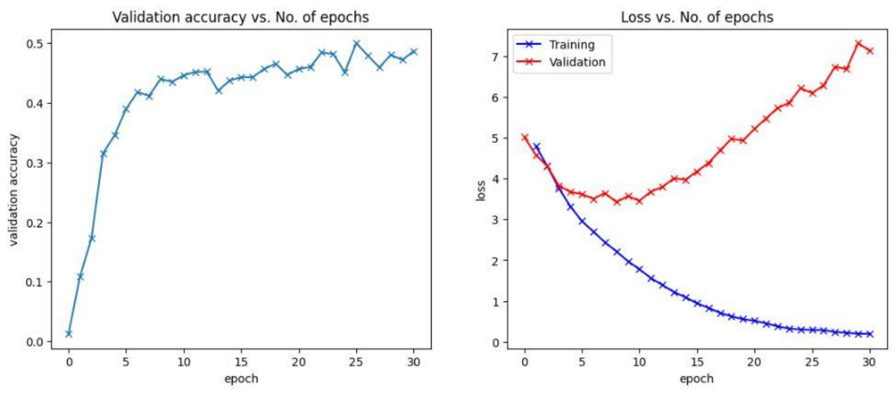


Figure 2: The validation accuracy (left), training and validation loss (right) of baseline model with 30 number of epochs and learning rate of 0.0001

Experiment 2: Baseline model with dropout and batch normalisation:

In this experiment, we add dropout layer and batch normalisation and train the model with different number of epochs (10, 30, 50). As shown in Table 3, the best validation accuracy increases to 55.3% with 50 epochs. Figure 3 shows the validation accuracy, training loss and validation loss of the baseline model with dropout and batch normalisation. We can observe that the overfitting problem is greatly reduced. The dropout and batch normalisation do not change the GFlops of the model (0.69G). The testing accuracy with dropout and batch normalisation in 50 epochs is 58.2%. The efficiency is 84% per Gflops.

Table 3: The validation accuracy of baseline model with dropout and batch normalisation

Model	Batch	Epochs	Learning rate	Dropout	Batch Normalisation	Optimizer	Validation Accuracy (%)	Gflops
Baseline Model with dropout and batch normalisation	16	10	0.0001	Yes	Yes	Adam	31.2	0.69G
		30					54.1	
		50					**55.3**	

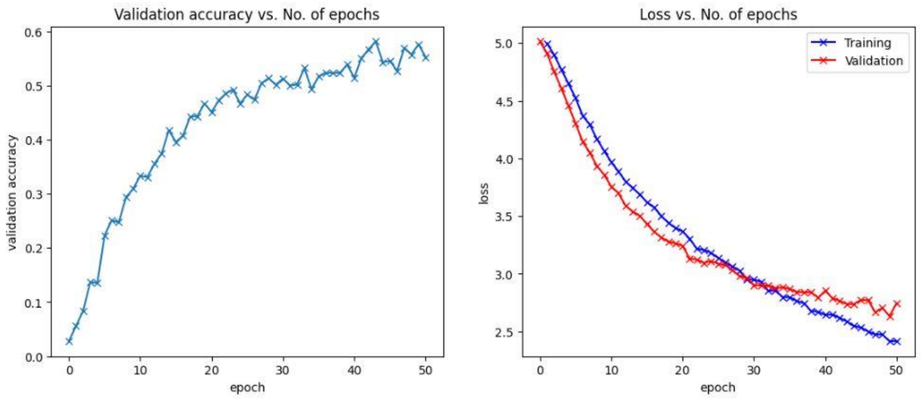


Figure 3: The validation accuracy (left), training and validation loss (right) of baseline model with dropout and batch normalisation (50 number of epochs and learning rate of 0.0001)

Experiment 3: Baseline model with different optimizers:

In order to further increase the accuracy of the model, we try different optimizers, namely Stochastic gradient descent (SGD) and Root Mean Squared Propagation (RMSprop). As shown in Table 4, the Adam optimizer gives the highest validation accuracy of 55.3%. It has a testing accuracy of 58.2%. The Gflops (0.69G) does not change with different optimizers. The efficiency remains 84% per Gflops.

Table 4: The validation accuracy of baseline model with different optimizers

Model	Batch	Epochs	Learning rate	Dropout	Batch Normalisation	Optimizer	Validation Accuracy (%)	Gflops
Baseline Model with dropout and batch normalisation	16	50	0.0001	Yes	Yes	**Adam**	**55.3**	0.69G
						SGD	12.1	
						RMSprop	49.6	

Experiment 4: Deeper and wider baseline model

In this experiment, we first increase the depth of the baseline model. Then we increase both the depth and the width of the baseline model. As shown in Table 5, the Deeper Baseline Model gives the best validation accuracy of 59.4%. The Deeper Baseline Model has testing accuracy of 64.3% and the Gflops of 1.16G. The efficiency is 55% per Gflops.

Table 5: The validation accuracy of deeper and wider baseline model

Model	Batch	Epochs	Learning rate	Dropout	Batch Normalisation	Optimizer	Validation Accuracy (%)	Gflops
Deeper Baseline Model	16	50	0.0001	Yes	Yes	Adam	**59.4**	1.16G
Wider and Deeper Baseline Model							58.9	4.45G

Transfer Learning

In order to further increase the accuracy and reduce the required computational power, we attempt the transfer learning technique. We use Pre-trained ResNet and Pre-trained MobileNet. ResNet has the skip connection which has been proven to give higher performance. The MobileNet uses depth-wise and point-wise convolutions that reduces the computational power. The detail explanation will be presented in the coming section.

Experiment 5: Pre-trained ResNet

In this experiment, we examine the ResNet 50, ResNet 101 and ResNet 152 with different learning rate (0.0001, 0.001, 0.01). As shown in Table 6, ResNet 50 with learning rate = 0.0001 gives the best validation accuracy of 89.1%. It has testing accuracy of 89.2% and the Gflops of 2.15G. The efficiency is 41% per Gflops.

Table 6: The validation accuracy of Pre-trained ResNet models

Model	Batch	Epochs	Learning rate	Dropout	Batch Normalisation	Optimizer	Validation Accuracy (%)	Gflops
ResNet 50	16	10	0.0001	No	Yes	Adam	**89.1**	2.15G
			0.001				43.4	
			0.01				17.9	
ResNet 101			0.0001				87.8	4.01G
			0.001				43.0	
			0.01				6.3	
ResNet 152			0.0001				87.8	5.87G
			0.001				47.5	
			0.01				4.1	

Experiment 6: Pre-trained MobileNet

In this experiment, we examine the MobileNet V2 and MobileNet V3 models with with different learning rate (0.0001, 0.001, 0.01). As shown in Table 7, MobileNet V2 with learning rate 0.0001 gives the best validation accuracy of 90.3%. It has a testing accuracy of 88.7% and the Gflops of 0.16G. The efficiency is 554% per Gflops. However, since the required Gflops for MobileNet V3 (0.12G) is lower than MobileNet V2 (0.16G), we also want to examine the accuracy of MobileNet V3. MobileNet V3 with learning rate 0.0001 has validation and testing accuracy of 89.1% and 90.3% respectively. The efficiency is 752% per Gflops.

Table 7: The validation accuracy of Pre-trained MobileNet models

Model	Batch	Epochs	Learning rate	Dropout	Batch Normalisation	Optimizer	Validation Accuracy (%)	Gflops
MobileNet V2	16	10	0.0001	No	Yes	Adam	**90.3**	0.16G
			0.001				73.5	
			0.01				3.4	
MobileNet V3			0.0001				**89.1**	0.12G
			0.001				78.5	
			0.01				1.3	

Final model

For baseline models comparison, the best accuracy model is Deeper Baseline Model with testing accuracy of 64.3%, Gflops of 1.16G and efficiency of 55% per Gflops. While the most efficient model is Baseline model with dropout and batch normalisation, which has testing accuracy of 58.2%, Gflops of 0.69G and efficiency of 84% per Gflops.

Efficiency = Testing accuracy / Gflops = 58.2 / 0.69 = 84% per Gflops

For pre-trained models comparison, the best accuracy and most efficient model is MobileNet V3 with testing accuracy of 90.3%, Gflops of 0.12G and efficiency of 752% per Gflops.

Efficiency = Testing accuracy / Gflops = 90.3 / 0.12 = 752% per Gflops

The overall best model is MobileNet V3 with testing accuracy of 90.3% and efficiency 752% per Gflops.

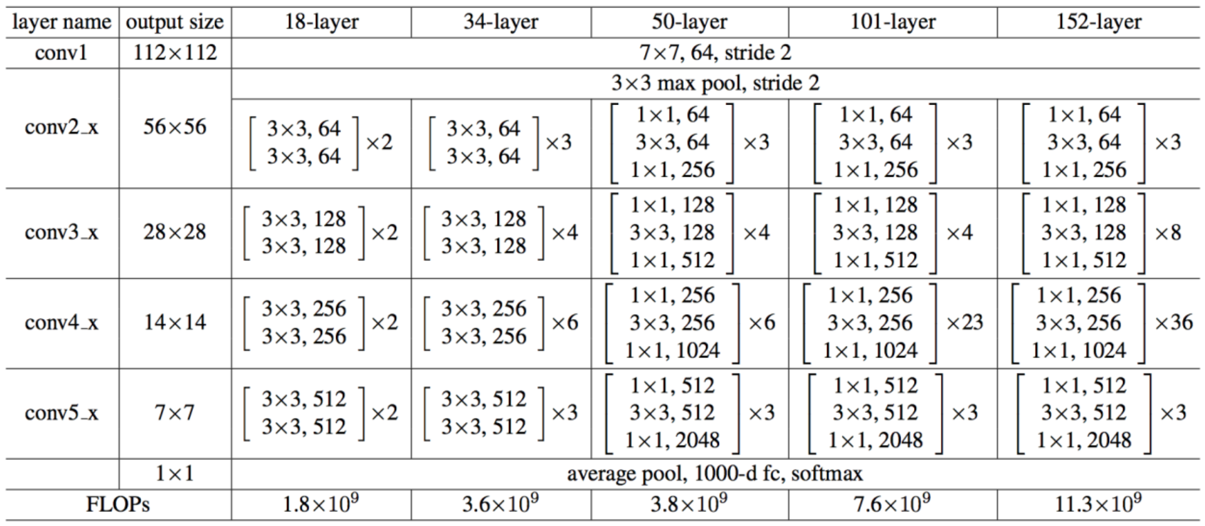
3. Explanation of the methods for reducing the computational cost and improve the trade-off between accuracy and cost

The performance of baseline models is optimised in three steps. First, we perform hyperparameters tuning (learning rate and number of epochs). The result shows overfitting issue. Second, we add batch normalisation and dropout layer. Batch normalisation normalizes the layer's output by re-centring and re-scaling to give the same mean and variance. The dropout layer randomly selects a fraction of neurons to be ignored and not updated in that training step. The result shows an increased accuracy and a reduced overfitting. Both hyperparameter tuning and adding batch normalisation and dropout do not increase the computational cost, but result in a higher accuracy. Third, we further increase the accuracy by changing the optimizer and increasing the depth and width of the baseline model. For the optimizers, among Adam, SGD and RMSprop, the default optimizer Adam gives the highest accuracy. For deeper and wider baseline models, they give a higher accuracy but also cause higher computational cost. To optimize the trade-off between accuracy and cost, the baseline model with batch normalisation and dropout is the best model with the highest efficiency.

Apart from the optimisation in baseline model, we also perform transfer learning and examine different pre-trained models. Transfer learning is a technique which reuses the pre-trained neural network models that were trained on large datasets for a specific task (Sarkar, D, Bali, R & Ghosh, T 2018, p. 2). In our project, the pre-trained models have already had some knowledge about classifying images based on features like shapes and edges. Transfer learning allows us to initialize the model with the pre-trained weights instead of training a new model from scratch. This greatly increases the performance of the classification models. In our project, we use pre-trained ResNet50, ResNet101, ResNet152, MobileNetV2 and MobileNetV3.

ResNet architecture

Residual Network is a deep convolutional neural network architecture. It has skip connections which allow information to bypass one or more layers in the network (Kaiming, He et al. 2016, p. 2). This avoids the problem of vanishing gradient and increases the model accuracy (Kaiming, He et al. 2016, p. 2). Figure 4 shows the architectures of different types of ResNet. In our project, we use the ResNet50, ResNet101, ResNet152 since deeper networks can learn more features and give better accuracy. The result shows a significant increase in accuracy. However, the computational cost also increases significantly. Therefore, we examine other pre-trained models with lower computational cost such as MobileNet models.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 4: The ResNet architecture of ResNet18, ResNet34, ResNet50, ResNet101, ResNet152 (Kaiming, He et al. 2016, p. 5)

<u>MobileNet architecture</u>

MobileNet is a simple but efficient and not very computationally intensive convolutional neural network for mobile vision applications. MobileNet uses depth-wise convolutions and point-wise convolutions to reduce the number of convolutions required in the network (Howard, AG et al. 2017, p. 2 - 3). Depthwise convolutions apply only a single convolutional filter to each one of the input channels (Howard, AG et al. 2017, p. 2 - 3). Then, the pointwise convolutions combine the output channels from the depthwise convolution using 1x1 convolution (Howard, AG et al. 2017, p. 2 - 3). The overall result is the same as traditional filter convolutions in CNN, but the number of convolutions has greatly reduced. In our project, we attempt to use MobileNet V2 and MobileNet V3 to reduce the computation power while maintaining good performance.

4. Limitations/Conclusions

There are several limitations of CNN models in our project. First, the dataset size is limited. It only contains 6270 images, but includes 151 classes. The amount of data per class is quite low. This can lead to overfitting, where the model memorizes the training data instead of learning meaningful features. Second, the dataset contains a wide variety of animals which increases the complexity of the classification task. Some classes might have subtle visual differences that are hard to distinguish. Third, the computational power is limited. As we modify the CNN architecture to improve the accuracy, the computational cost might also increase, especially for deeper and more complex models. Although using per-train model can increase the accuracy while maintaining relatively low computational cost, sometimes it is difficult to find the suitable pre-trained model for a specific task.

In conclusion, the task of animal image classification using various Convolutional Neural Network (CNN) models is a balance between accuracy and efficiency. We modified the baseline model by hyperparameter tuning, adding batch normalisation and dropout, examining different optimizers and increasing the depth and width of the model architecture. The best baseline model is Baseline model with dropout and batch normalisation, which has testing accuracy of 58.2% and efficiency of 84% per Gflops. Furthermore, we examine two types of pre-trained models including ResNet (50, 101, 152) and MobileNet (V2, V3). The best model is MobileNet V3 with testing accuracy of 90.3% and efficiency 752% per Gflops. Future works include researching for more efficient pre-trained CNN models and modify the architecture to increase the performance and efficiency.

Reference

Sarkar, D, Bali, R & Ghosh, T 2018, Hands-On Transfer Learning with Python : Implement Advanced Deep Learning and Neural Network Models Using TensorFlow and Keras., 1st ed., Packt Publishing, Limited, Birmingham.

Kaiming He, Xiangyu Zhang, Shaoqing Ren & Jian Sun 2016, ‘Deep Residual Learning for Image Recognition’, in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, pp. 770–778.

Howard, AG, Zhu, M, Chen, B, Kalenichenko, D, Wang, W, Weyand, T, … Adam, H 2017, ‘MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications’.