Image Classification on Animals for Computer Vision Competition

Camille Ko, Lipin Guo
The University of Adelaide, Australia

Abstract

Our task is to classify animals in the animal dataset which have has 6,270 images categorized into 151 different labeled classes. The accuracy of base model is 37.53%, and the efficiency is 0.69G. After experiments, we find the best model is MobileNetV3_large. Our best accuracy is 98.28% and the efficiency is 0.43G. The performance improves 60.75% of accuracy and decreases 0.26G.

1. Introduction: Base Model

The main task of this competition is to classify animals in the provided dataset with convolutional neural networks (CNN) and to achieve accuracy > 95% and efficiency <0.69G for the highest marks as possible. In total there are 24 experiments ran with different models and parameters. To begin with, base model is introduced and can be served as the benchmark and control model.

1.1. Dataset Preparation

The animal dataset provided has 6,270 images categorized into 151 different labeled classes. The number of images in each class varies from 26 (for stegosaurus-stenops) to 60 (for vulpes-vulpes). By random splitting, there are 5,330 training, 313 validation and 627 test images.

1.2. Data Augmentation

To incorporate more variations for each epoch and improve generalization to unseen samples (different kinds of test data), data augmentation, is a technique to increase the diversity of training set by apply random transformations [1], can be applied in the pre-processing step. In this specific task, the default transformations to base model are:

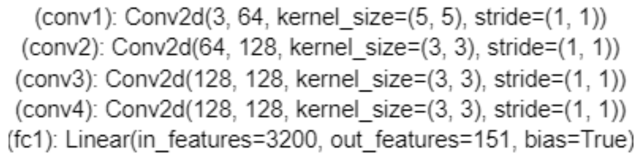
(i) resize (from 224 to 112 pixels)
(ii) horizontally flip the given image randomly with a probability of 0.5 [2]
(iii) crops the resized image from the centre
(iv) converts the PIL Image which has values in the range of 0-255 to a floating-point Tensor and normalizes them to a range of 0-1, by dividing it by 255
(v) takes in a 3 channel Tensor and normalizes each channel by the input mean and standard deviation for that channel. Mean and standard deviation vectors are input as 3 element vectors. Each channel in the tensor is normalized as T = (T – mean)/(standard deviation)

1.3. Data Loading

The batch size as default is 16. Training, validation, and test images are loaded into train_loader, val_loader, and test_loader respectively. Two subprocesses are used for data loading. The train_loader will reshuffle the data at every epoch. Also. the data loader will copy Tensors into device/CUDA pinned memory before returning them [3].

1.4. Base Model Network Architecture for Training, Validation, Inference

The basic idea of a CNN model is to extract higher representations for the image content. CNN takes the image's raw pixel data, trains the model, then extracts the features automatically for better classification [4]. The default base model is composed of 4 convolutional layers and 1 fully connected layer, and take the respectively input, kernel filter, and stride (the skip of value pools [5]) as below, for convolution operation.



```
(conv1): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1))
(conv2): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
(conv3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1))
(conv4): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1))
(fc1): Linear(in_features=3200, out_features=151, bias=True)
```

The input size is 3*112*112 and the output size is 151. To training the base model, learning rate of 0.001, epoch = 10 are used. For each batch, input images are passed through the model, to get the outputs. The forward pass here fed with ReLU and maxpooling layers. The features wrapped with log_softmax activation function. Then the provided loss criterion - cross entropy is used to compute the loss using the ground truth and the computed outputs. Cross entropy [6] is useful when training a classification problem with multi-classes. The gradients of the loss with respect to the trainable parameters are computed using the backward function to calculate the loss. By the gradient descent the base model could minimize prediction loss and have the predicted output. After the gradients are computed in the backward pass, the parameters are updated using the optimizer's step function. Adam is used here as the optimizer because it can adapt the learning rate for each parameter individually [7].

1.5. Evaluation Metric Function

The outputs are measured by accuracy and also FLOPs. The given accuracy computes the accuracy over the top-3 predictions out of 5 guesses. The higher the accuracy the better the output. On the other hand, FLOPs are the measurement for efficiency. The given FLOPS calculate in the unit of G. The smallest the unit of G, the better the output.

2. Methods and Experiments

2.1. Base Model with Different Parameters

The following experiments are performed with the base model with different combinations of parameters changed. The accuracy started from 37.53%, and increased to 53.23% with 4 tricks.

2.1.1 *1 trick: different optimizers*

Experiment	Optimizer	Accuracy
0	Adam	37.53%
1	SGD	3.48%
2	SGD + momentum (0.9)	36.14%

Table 2.1.1. Base model, with batch = 10, epoch = 10, learning rate = 0.001, default data augmentation, loss function = cross entropy, and different optimizers.

From the table 2.1.1, we can see that experiment 0, which uses the base model without any changes, has 37.53% accuracy. The change of optimizer doesn't help to get a better accuracy. That's why we still choose Adam as optimizer when we change other parameters.

2.1.2 *1 trick: different learning rate*

Experiment	Learning Rate	Accuracy
0	0.001	37.53%
1	0.00001	24.27%
2	0.0001	45.83%
3	0.01	1.72%
4	0.1	1.56%

Table 2.1.2. Base model, with batch = 10, epoch = 10, default data augmentation, loss function = cross entropy, optimizer = Adam, and different learning rates.

From the table 2.1.2, we found that when the learning rate is equal to 0.0001, the model gets better accuracy than the original learning rate is 0.001. Learning rate is 0.1 or 0.01, the accuracy is very low, so in the later experiments we would not consider these two learning rates.

2.1.3 *2 tricks: different epoch + learning rates*

Experiment	Epoch	Learning Rate	Accuracy
0	10	0.001	37.53%
1	20	0.001	35.09%
2	10	0.0001	45.83%
3	20	0.0001	44.31%

Table 2.1.3. Base model, with batch = 16, default data augmentation, loss function = cross entropy, optimizer = Adam, different epoch + learning rates.

From the table 2.1.3., we can see that the change of epoch decreases the accuracy. For later experiments, we will use 10 epochs.

2.1.4 *2 tricks: change batch size from 16 to 32 + different learning rates*

Experiment	Batch	Learning Rate	Accuracy
0	16	0.001	37.53%
1	32	0.001	38.34%
2	16	0.0001	45.83%
3	32	0.0001	40.24%

Table 2.1.4. Base model, epoch = 10, default data augmentation, loss function = cross entropy, optimizer = Adam, different batch sizes and learning rates.

From the table 2.1.4, we can see that the changing batch size sometimes doesn't increase the accuracy, but sometimes increases. It needs to be discussed case by case. We will use batch size 16 for later experiments.

2.1.5 *2 tricks: change loss function from cross entropy to multi margin loss + different learning rates*

Experiment	lr	Loss Function	Accuracy
0	0.001	Cross Entropy	37.53%
1	0.001	Multi Margin Loss	32.39%
2	0.0001	Cross Entropy	45.83%
3	0.0001	Multi Margin Loss	34.15%

Table 2.1.5. Base model, batch = 16, epoch = 10, default data augmentation, optimizer = Adam, different loss functions and learning rates.

From the table 2.1.5., we can see that the multi margin loss doesn't increase the accuracy, so we will still use cross entropy for later experiments.

2.1.6 *2 tricks: add batch normalization layer + different learning rate*

Experiment	lr	Batch Normalization	Accuracy
0	0.001	n/a	37.53%
1	0.001	3 batch norm level	47.01%

2	0.0001	n/a	45.83%
3	0.0001	3 batch norm level	51.03%

Table 2.1.6. Base model batch = 16, epoch = 10, default data augmentation, optimizer = cross entropy, loss functions = Adam, different learning rates and 3 extra batch normalization layers.

From the table 2.1.6., we can see that adding 3 batch normalization layers helps increase the accuracy since the structure of neural network is more complex. And best accuracy is when the learning rate is equal to 0.0001, while adding 3 batch normalization layers.

2.1.7 *3 tricks: dropout + add batch normalization layer + different learning rate*

Experiment	lr	Dropout	Batch Normalization	Accuracy
0	0.001	n/a	n/a	37.53%
1	0.0001	n/a	n/a	45.83%
2	0.0001	n/a	3 extra layers	51.03%
3	0.0001	0.5	3 extra layers	53.23%

Table 2.1.7. Base model, batch = 16, epoch = 10, default data augmentation, optimizer = Cross Entropy, loss = Adam, different learning rates, one dropout layer (0.5) and 3 extra batch normalization layers.

Experiment 0 is the original model without changes. Experiment 1 uses 1 trick, experiment 2 uses 2 tricks, and experiment 3 uses 3 tricks. From the table, we can see that the accuracy increases gradually after adding tricks. Tuning parameters such as learning rate, dropout can help improve the accuracy, as well as adding a batch normalization layer to change the architecture.

2.1.8 *4 tricks: data augmentation + dropout + add batch normalization layer + different learning rate*

Experiment	lr	Data Augmentation	Dropout	Batch Normalization	Accuracy
0	0.001	default	n/a	n/a	37.53%
1	0.0001	default	n/a	n/a	45.83%
2	0.0001	default	n/a	3 extra	51.03%
3	0.0001	default	0.5	3 extra	53.23%
4	0.0001	+ rotation	0.5	3 extra	51.82%

Table 2.1.8. Base model, batch = 16, epoch = 10, optimizer = Cross Entropy, loss = Adam, different learning rates, different dropout layer and 3 extra batch normalization layers, and rotation in data augmentation.

From the table 2.1.8, we can see that changing augmentation by adding rotation do not increase the accuracy if we change it with dropout and batch normalization together. However, compared to only adding a batch normalization layer, changing data augmentation at the same time helps increase the accuracy.

2.2. Variety of Models with Same Parameters

Based on the experiments from 2.1, we learnt that the more complex the neural network architecture, the higher the accuracy for most of the cases, along with the optimal parameters we tested throughout different experiments. Therefore, instead of using the relatively simple base model, we will apply different pretrained neural network models in the following sections. ResNet and MobileNet are selected in the following experiments since they have dropout and residual blocks (included batch normalization) in their pretrained architectures. ResNet and MobileNet required the input to be at least 224 pixels. Therefore, the data augmentation for ResNet and MobileNet are set to be

(i) Resize(256),
(ii) CenterCrop(224),
(iii) ToTensor(),
(iv) Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

This data augmentation is referred as default data augmentation when the mentioned model is ResNet or MobileNet.

2.2.1 *2 tricks: different models and optimizers*

Experiment	Model	Optimizer	Accuracy	Efficiency
0	Base	Adam	37.53%	0.69G
1	Base	SGD	3.48%	0.69G
2	ResNet18	SGD	90.94%	3.59G
3	ResNet34	SGD	93.55%	7.28G
4	ResNet50	SGD	92.81%	8.17G
5	MobileNetV2	Adam	80.94%	0.60G
6	MobileNetV2	SGD	86.56%	0.60G
7	MobileNetV3_large	Adam	87.97%	0.43G
8	MobileNetV3_large	SGD	84.69%	0.43G

Table 2.2.1. Base/ ResNet18/ ResNet34/ ResNet50/ MobileNetV2/MobileNetV3_large model, batch = 16, epoch = 10, learning rate = 0.001, default data augmentation, and different optimizers.

Experiment 0 is the base model without changes. As ResNet usually uses SGD as optimizer, so we compare models with SGD as optimizer as well. Without considering the difference of optimizer, we can see that ResNet50 has the best accuracy of 93.55%. However, RestNet50 reaches the highest efficiency of 8.17G. After considering the efficiency, we found that MobileNet has the best efficiency (less than the base model) and great accuracy (close to the best accuracy). So, we decided to use MobileNet as our model to improve performance. Next, we will compare the performance of MobileNetV2 and MobileNetV3 with different parameters to get the best model.

2.2.2 *3 tricks: different models, learning rates, and optimizers*

Experiment	Model	Learning Rate	Optimizer	Accuracy	Efficiency
1	Mobile NetV2	0.001	Adam	80.94%	0.60G
2		0.0001	Adam	96.41%	
3		0.01	SGD	96.72%	
4		0.001	SGD	86.56%	
5		0.0001	SGD	19.27%	
6	Mobile NetV3 _large	0.001	Adam	87.97%	0.43G
7		0.0001	Adam	95.94%	
8		0.01	SGD	97.81%	
9		0.001	SGD	84.69%	
10		0.0001	SGD	85.78%	

Table 2.2.2. MobileNetV2/MobileNetV3_large models, batch = 16, epoch = 10, optimizer = Cross Entropy, different learning rates, and different optimizers.

After comparing the accuracy and efficiency, we found that MobileNetV3_large has the best performance - 97.81% accuracy and 0.43G efficiency when the learning rate is 0.01 and the optimizer is SGD.

2.3. Best Model – MobileNetV3 with Different Parameters

Compared all the pretrained models with base model, MobileNetV3_large gave the best performance regarding to accuracy and efficiency. Therefore, in the following experiments, we will continue to test out different parameters with MobileNetV3_large to achieve the best result. The final best result is 98.28% accuracy with 0.43G efficiency using MobileNetV3_large models, batch = 32, epoch = 20, optimizer = SGD, default data augmentation, learning rates = 0.1 and loss function = cross entropy.

2.3.1 *2 tricks: batch size + different learning rate*

Experiment	Batch	Learning Rate	Accuracy	Efficiency
0	16	0.01	97.81%	0.43G
1	16	0.1	90.16%	
2	32	0.01	97.66%	
3	32	0.1	97.81%	

Table 2.3.1 MobileNetV3_large model, epoch = 10, default data augmentation, optimizer = SGD. Different batch size and learning rates.

From the table 2.3.1, we can see that the best accuracy is 97.81%. However, when the batch size is 32, the model has great accuracy under different learning rates. So later experiments will use batch size which is equal to 32.

2.3.2 *2 Tricks: different learning rates + loss functions*

Experiment	lr	Loss Function	Accuracy	Efficiency
1	0.01	Cross Entropy	97.66%	0.43G
2	0.01	Mutli Margin Loss	91.61%	
3	0.1	Mutli Margin Loss	96.41%	

Table 2.3.2. MobileNetV3_large models, batch = 32, epoch = 10, default data augmentation and optimizer = SGD. Different learning rates, and loss function.

From the table 2.3.2, we can see that the change of loss function doesn't increase the accuracy, but the change of learning rate increases the accuracy.

2.3.3 *3 tricks: loss function + optimizer + learning rate*

Experiment	lr	Loss	Optimizer	Accuracy	Efficiency
1	0.01	Cross Entropy	SGD	97.66%	0.43G
2	0.01	Mutli Margin Loss		91.61%	
3	0.1	Mutli Margin Loss		96.41%	
4	0.1	Mutli Margin Loss	SGD + momentum (0.9)	2.76%	

Table 2.3.3. MobileNetV3_large models, batch = 32, epoch = 10, default data augmentation. Different learning rates, optimizers and loss function.

From the table 2.3.3, we can see that changing the optimizer decreases the accuracy rapidly, so SGD is the appropriate optimizer for MobileNetV3. We don't need to change the optimizer.

2.3.4 *2 Tricks: epoch number+ learning rate*

Experiment	Epoch	Learning Rate	Accuracy	Efficiency
0	10	0.01	97.66%	0.43G
1	10	0.1	97.81%	
2	20	0.01	97.66%	
3	20	0.1	98.28%	

Table 2.3.4. MobileNetV3_large models, batch = 32, optimizer = SGD, loss = cross entropy, default data augmentation. Different learning rates and epoch.

After changing the epoch to 20, we get our best model with 98.28%, accuracy and 0.43G FLOPS, when the batch size is 32, loss function is cross entropy, optimizer is SGD.

3. Computational cost, Accuracy, Efficiency

3.1. Accuracy-efficiency Graph among Models

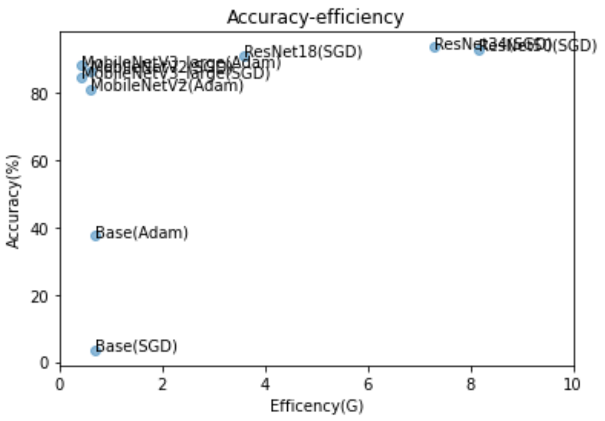


Figure 3.1. Accuracy-efficiency Graph of different models from experiments

Accuracy-efficiency graph demonstrates the relationship of accuracy and efficiency from a model. We plot all the models we tested in experiments into one accuracy-efficiency curve and realized that there is strong evidence for trade off between these two factors.

Using accuracy as the single measurement on classification are not ideal. As the model complexity increases, the accuracy will usually go along the same direction. However, that also means a longer computation cost. Therefore, finding a balance between accuracy and efficiency is important and practical to solve classification task in real life scenario.

Out of all the model we tested in experiments, MobileNetV3_large returns the best performance. From the graph above, MobileNet are positioned at the top left corner as they have higher accuracy and low efficiency. ResNet have very high accuracy yet their computational costs are also high. They are positioned at the top right-hand corner on the graph.

On the other hand, base model with Adam and SGD optimizers positioned at the bottom left region on the graph. This indicates that they are under performed compared to the other models.

4. Explanation of Code Implementation

We have tuned different parameters in the experiments. The complete version of code implementation can be found in the attached documents named code_implentation.xlxs and Animal_Classification_best_model.ipynb.

4.1. Transfer Learning

We applied transfer learning in our experiments. Transfer learning is a machine learning method where we reuse a pre-trained model on a new task as a starting point for the model. The pre-trained model already knows how to classify objects and has learned general features like edges, shapes, etc. in images. By applying transfer learning to new tasks, significantly higher performance data can be obtained compared to training with only a small amount of training. We had tried different pre-trained models, such as ResNet18, ResNet34, ResNet50, MobileNetV2 and MobileNetV3_large. And we found the best pre-trained model for our animal classification task is MobileNetV3_large [8].

4.2. Learning Rate

The learning rate is a hyperparameter that controls how much the model changes in response to estimation error each time the model weights are updated. We extensively tune learning rate across our experiments. Choosing a learning rate is challenging, as too small a value may cause the training process to be too long and may get stuck, while too large a value may result in learning a suboptimal set of weights too quickly or with an unstable training process. We had tried different learning rates, such as 0.1,0.01,0.001,0.0001,0.00001 to tune our models [9].

4.3. Loss Function

Loss function computes the distance between the current output of the algorithm and the expected output. It's a method to evaluate how algorithm models the data. In the experiments, we choose cross entropy and multiply margin loss for comparison. Multi margin loss can be used for multi-classification tasks. We tried multi margin loss in the base model and MobileNetV3_large, and we found that cross entropy performed better [10].

4.4. Data Augmentation

In order to get more data, we perform data augmentation, which means make variations to our existing dataset. Transformations such as flips or translations or rotations are applied to data augmentation. CNN can be invariant to translation, viewpoint, size or illumination. CNN can robustly classify objects, so it's suitable for us to try data augmentation. We had tried to change rotation in our base model [11] [12].

4.5. Architectural Changes

Architectural change means construction, alteration, or removal of the CNN structure, such as adding batch normalization layers, change residual layers, change attention blocks, etc.. Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks. In the experiments, we tried adding batch normalization layers in our base model. Also the pretrained MobileNet3 has a batch normalization layers[13].

5. Visualization Results

The graphs we compared here are the base model as the benchmark and MobileNetV3_large as the best performer. The baseline test accuracy of base model returns 37.53% with batch = 16, learning rate = 0.001, loss function = cross entropy, optimizer = Adam. The efficiency of it returns 0.69G. The test accuracy from best performed MobileNetV3_large returns 98.28% and the efficiency is 0.43G with the parameters of epoch = 20, batch = 32, learning rate = 0.1. loss function = cross entropy, optimizer = SGD.

5.1. Training Accuracy Curves

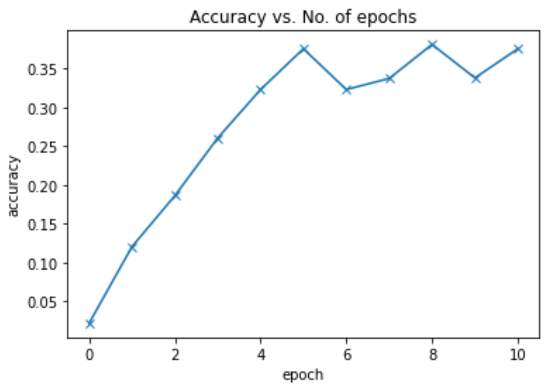


Figure 5.1a. Accuracy curve of Base model with epoch = 10.

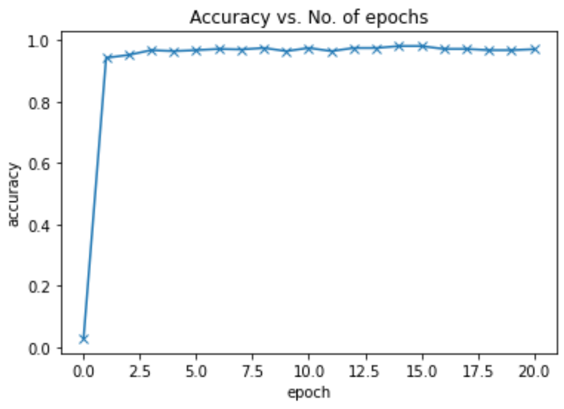


Figure 5.1b. Accuracy curve of MobileNetV3_large model with epoch = 20.

From the above figures, both of the model performance is growing over time. This indicates that the models are learning when the numbers of epochs grow. However, the accuracy from base model fluctuates. This indicates that overfitting happened when the model fit the random samples from test data. Looking at the accuracy curve of MobileNet, the curve reached a plateau around epoch = 2.5. This means the model is not learning anymore.

5.2. Training and Testing Loss Curves

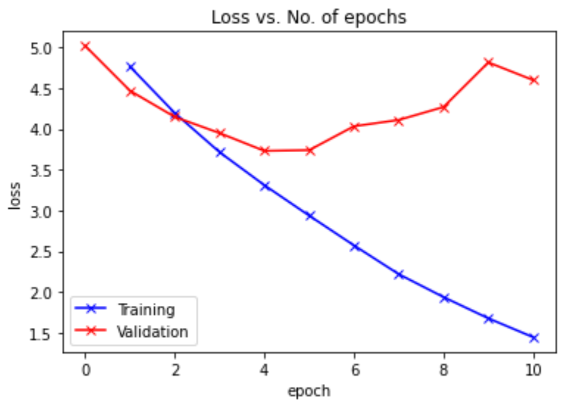


Figure 5.2a. Loss curves of Base model with epoch = 10

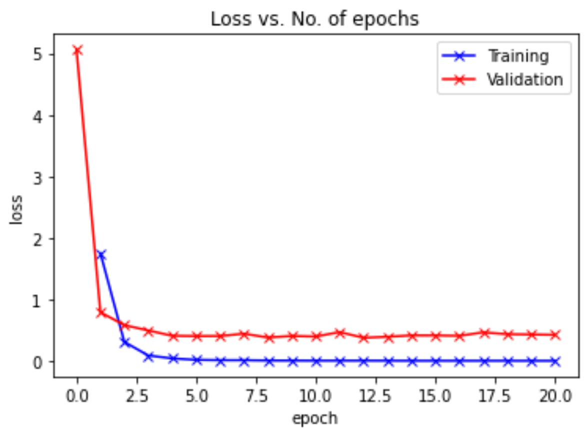


Figure 5.2b. Loss curves of MobileNetV3_large model with epoch = 20.

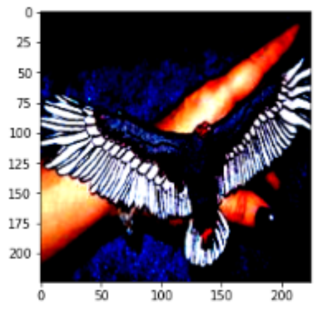
Both of the training loss curves reach lower error over time. But From figure 5.2a, we can see that the gap between training and validation are setting apart after epoch reached 4, this means that the model starts to overfit. On the other hand, the loss curves of training and validation from MobileNetV3_large go down at the same direction, meaning the model has good ability to fit new data.

5.3. Failure Case Studies

We extracted some of the misclassified images in the model and concluded that deformation, background clutter and illumination are the main challenges that the model fails to classify images of animals.

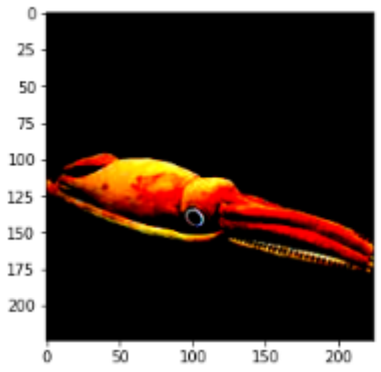
5.3.1 *Deformation*

```
label: 26 cathartes-aura
predicted: 16 betta-splendens
<matplotlib.image.AxesImage at 0x7fdd3c30c210>
```



From the above image, we can see that cathartes-aura(eagle) are predicted as betta-splendens (fish)by our model. They are completely different animals. But the image of the eagle is captured in different poses, with the wings open, with share the similar features of a fighting fish.

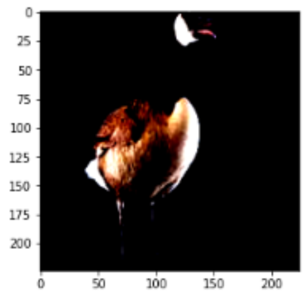
```
label: 13 architeuthis-dux
predicted: 38 correlophus-ciliatus
<matplotlib.image.AxesImage at 0x7fdcf0484410>
```



Again, architeuthis-dux (giant squid)is misclassified as correlophus-ciliatus (lizard). The reason maybe because in the test image, the legs of squid is not open which look different from other training image.

5.3.2 *Background Clutter*

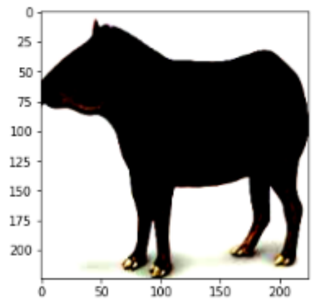
```
label: 21 branta-canadensis
predicted: 134 struthio-camelus
<matplotlib.image.AxesImage at 0x7fdcf0590f50>
```



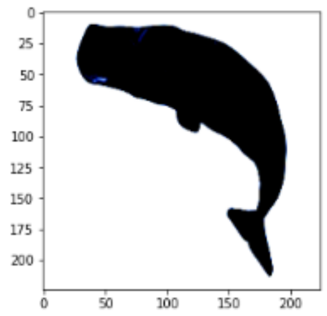
The image branta-canadensis(goose) is misclassified as struthio-camelus (ostrich)since the quality of the test image is poor. The animal itself blends into the background which makes the model hard to extract the key features.

5.3.3 *Illumination*

```
label: 135 tapirus
predicted: 55 equus-caballus
<matplotlib.image.AxesImage at 0x7fdd30183a50>
```



```
label: 117 physeter-macrocephalus
predicted: 143 tursiops-truncatus
<matplotlib.image.AxesImage at 0x7fdcf0620b90>
```



The above two images are the classic examples of illumination. On the top image, tapirus are misclassified as equus-caballus (horse). On the below image, whale is mistakenly predicted as dolphin. The reason of misclassification of these two images are due to the over exposure of the image made the features of tapirus and whale lost the texture and features are fail to extract.

6. Conclusion

6.1. Limitation

6.1.1 *Quality of images in dataset*

Many of the classes are extinct animals. The images of them in the data sets are of fossils, toys or imaginary. Often the time these images are not representative to the classes they are.

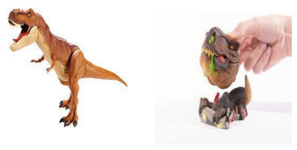


Fig. 6.1.1a. the images of tyrannosaurus-rex (left) and (right)

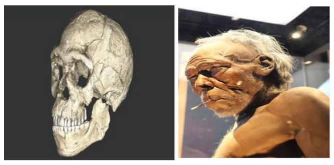


Fig. 6.1.1b. the images of homo-sapiens (left) and (right)

6.1.2 *Imbalance Classes*

The original dataset with 151 classes is slightly imbalance. The number of classes ranging from 26 to 60.

Taking a deep look into the context of each original classes in the dataset, each 151 classes with their biological names seem to have a relatively balanced ratio. However, if re-group these classes into a more layman terminology (for example, bird, big cat, lizard, dinosaurs, etc.), some re-groups are dominated into dataset.

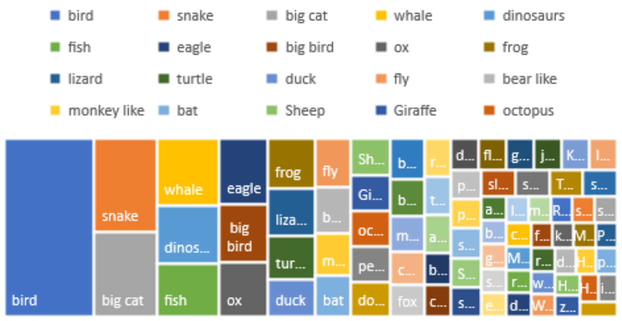


Fig6.1.2. Contribution of each re-groups in dataset (total with 6270 images)

The top 10 of the re-groups are birds (15%), snakes (5%), big-cats (5%), whales (4%), dinosaurs (3%), fishes (3%), eagles (3%), big-birds (3%), ox (3%) and frogs (2%). The least contributing re-groups are sharks, wolves, dogs, hippos, pandas, horses, zebra, human, rats, which all took less than 1% in the dataset. CNN models tend to be more biased toward the majority classes when performing classification task.

6.1.3 *Different shape of animal throughout their life stage*

Butterfly in the dataset covers different stages as we see in the below images. This will bring noises to the model while learning.

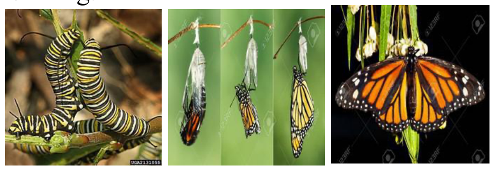


Fig. 6.1.3. the image of different stages of danaus-plexippus (left, middle, right)

6.2. Conclusion

In conclusion, our base model has 37.53% of accuracy and 0.69G of efficiency. To improve performance, we tried to change our base model at the beginning. We tried to tune parameters such as learning rate, loss function and data augmentation. We also changed architecture of base model such as adding batch normalization layers. We improved the base model by getting 53.23% of accuracy and 0.69G of efficiency. Next, we tried using pre-trained models. After many experiments, we found the best pre-trained model was MobileNetV3_large. After tuning parameters, our best accuracy of MobileNetV3_large is 98.28% and the efficiency is 0.43G.

References

[1] https://www.tensorflow.org/tutorials/images/data_augmentation#overview

[2] https://pytorch.org/vision/main/generated/torchvision.transforms.RandomHorizontalFlip.html

[3] https://pytorch.org/docs/stable/data.html#:~:text=Data%20loader.,(collation)%20and%20memory%20pinning.

[4] https://towardsdatascience.com/understanding-cnn-convolutional-neural-network-69fd626ee7d4&sa=D&source=docs&ust=1660550490809372&usg=AOvVaw2HmWvhhVQkU95ompkzZjfl

[5] https://towardsdatascience.com/understanding-cnn-convolutional-neural-network-69fd626ee7d4

[6] https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

[7] https://learnopencv.com/image-classification-using-transfer-learning-in-pytorch/

[8] https://www.v7labs.com/blog/transfer-learning-guide

[9] https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

[10] https://pytorch.org/docs/stable/generated/torch.nn.MultiMarginLoss.html#torch.nn.MultiMarginLoss

[11] https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/

[12] https://pytorch.org/vision/stable/transforms.html

[13] https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/