


Cat Breeds Classification

-Phạm Thiên Nhật, Trần Giang Anh, Nguyễn Minh Tùng-

1, Problem:

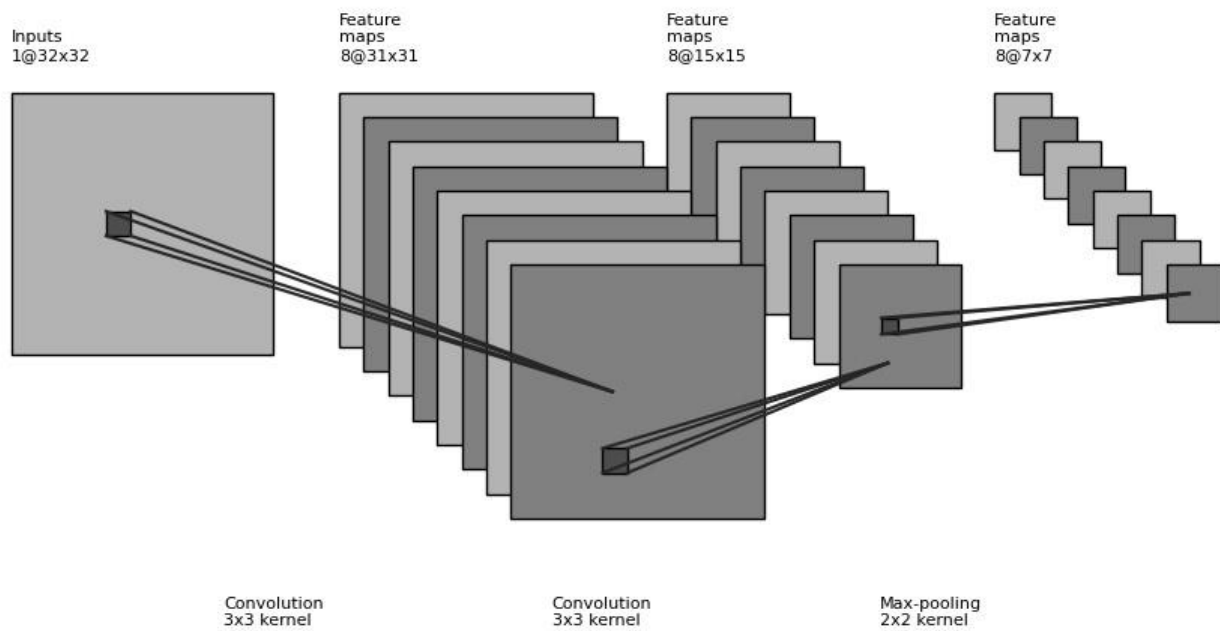
Given images of cats, the model is expected to give a prediction of the cat's breed in 1 of the 4 breeds below:

Munchkin	Persian
	
Siamese	Sphynx
	

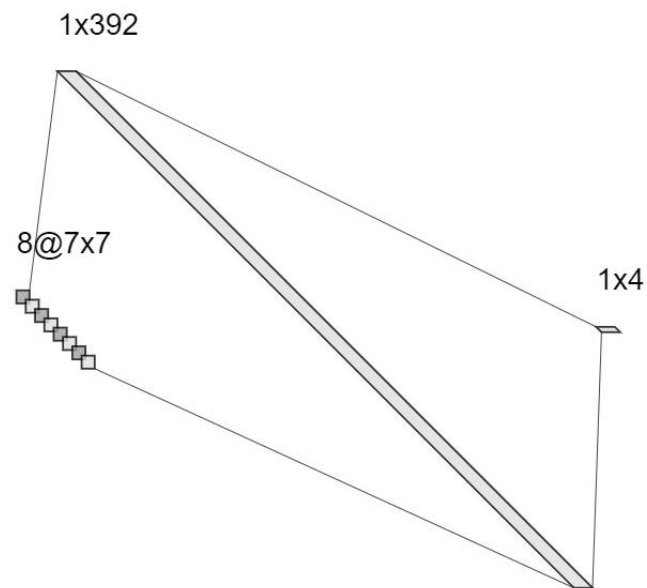
(Munchkin: 0 - Persian: 1 - Siamese: 2 - Sphynx - Hairless Cat: 3)

2, Approach:

In order to solve the problem, we decided to use a ConvNet as it's much more efficient than usual Neural Network at the same task of classification. An illustration an architecture we came up with and designed can be seen below:



(Convolutional Layers Stacking)



Layer	Filter Size	Stride	Activation	Input Shape	Output Shape
Input	-	-	-	64x64	1x64x64
Convolutional	3	2	Leaky ReLU	1x64x64	8x31x31
Convolutional	3	2	Leaky ReLU	8x31x31	8x15x15
Max Pooling	2	2	Leaky ReLU	8x15x15	8x7x7
Dense	-	-	-	1x392	1x4
Output	-	-	-	1x4	1x4

The formula used to calculate the Shape of convolutional output is as below:

$$\left\lfloor \frac{(W - K + 2P)}{S} \right\rfloor + 1$$

W = Input size

K = Filter size

S = Stride

P = Padding (=0 in our case)

We decided to implement Leaky ReLU instead as normal ReLU usually suffer from the “dying ReLU” problem which may kill potential neurons, preventing it from activating again. This is a huge problem as our architecture has a lot neuron.

We implemented each layer type as a class, all has 2 common functions:

1. **forward()** : Perform forward propagate used when predicting, training, returning the output after processing the input.
2. **backward()** : Perform backward propagate which is used mostly in training, it update the weights of the layer based on the loss value returned to the layer, then return the loss value to the layer before it.

While it is true that the model is still too simple with only 2 Convolutional Layers and a Max Pooling Layer, this is a good trade-off for the computation speed which is way faster than those of high complexity. The choice of this model design is also due to the lack of time and hardware potential to train complex model.

The choice of fixing the size of the kernel to be 3x3 is to reduce the computational costs, while choosing a stride equal to 2 let us downsample the image smartly, making the architecture more expressive than abusing the use of pooling layers to downsample.

We also implemented L2 regularization, although it is not used yet as there are still bugs (?). The basic idea of L2 can be described as below:

$$Cost\ function = Loss + \frac{\lambda}{2m} \sum \|w\|^2$$

L2 regularization basically forces the weights to decay towards zero (but not exactly zero), hence the term “weight decay”.

In the Dense Layer, we used SoftMax function to calculate the output. The SoftMax we implemented can be described as below:

$$a_i = \frac{e^{z_j}}{\sum_{j=1}^C e^{z_j}} \quad \forall i=1, 2, \dots, C$$

Where C equals number of total classes and Z_j is the score of that class.

As for the loss, we used Categorical Cross Entropy, which combined with SoftMax can be described as below:

$$CE = -\log(a_i)$$

Hence, when combined with L2, we have the following Cost function:

$$Cost\ function = CE + \lambda \sum \|w\|^2$$

Where w stands for the weight of that layer.

As for the Convolutional Layer, each of them is activated through Leaky ReLU as mentioned above, which can be explained as:

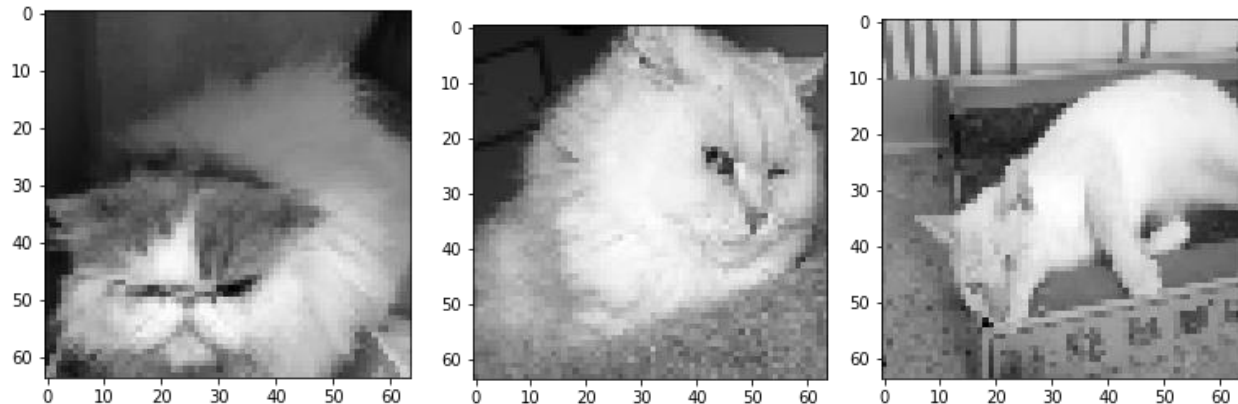
$$f(x) = \begin{cases} x & \text{for } x \geq 0 \\ \alpha x & \text{for } x < 0 \end{cases}$$

Which mean the derivative of it is:

$$\frac{df(x)}{dx} = \begin{cases} 1 & \text{for } x > 0 \\ \alpha & \text{for } x < 0 \end{cases}$$

3, Dataset:

The data we used was mined from petfinder.com using their APIs. The data contained a total of 7296 images total, divided into 4 breeds of cat as mentioned earlier. The data are fitted and cropped into a size of 64x64, and then grayscaled to reduce the dimension to 1. This is to reduce the data complexity and speed up computation task.



(Example of images transformed)

Each cat breeds are then labelled numerically, indexing from 0 to 4.

(Munchkin: 0 - Persian: 1 - Siamese: 2 - Sphynx - Hairless Cat: 3)

Then, we grouped the data into arrays, which is then through the usage of [Pickle](#) library, will be transformed into a single file for caching, which reduce the disk usage tremendously as it tends to become a bottleneck when training.

We divided the data into 2 files separately, one contains the images, and one contains the labels. The final shape of each array stored in each file is as below:

- Images: (7296, 64, 64)
- Labels: (7296,)

We then sliced the data into each batch. 7296 images are divided into:

- Train batch: 85% = 6201
- Validation batch: 5% = 364
- Test batch: 10% = 731

When inside the Input Layer, the data is then further sliced into each image solely, which an added dimension added which represent the channel of the image. In our case, as it's grayscaled, the channel value at input layer is 1; else it's 3 for RGB.

- Images: (64,64) -> Image: (1,64,64)

4, Training:

In the training process, we used a **learning rate** of 0.005 and trained for a total of 5 **epochs**, which means a total of 5x7296 **learning steps** total. We also run a **validation** task at each 250 iterations to evaluate the model on the way.

[illegible]

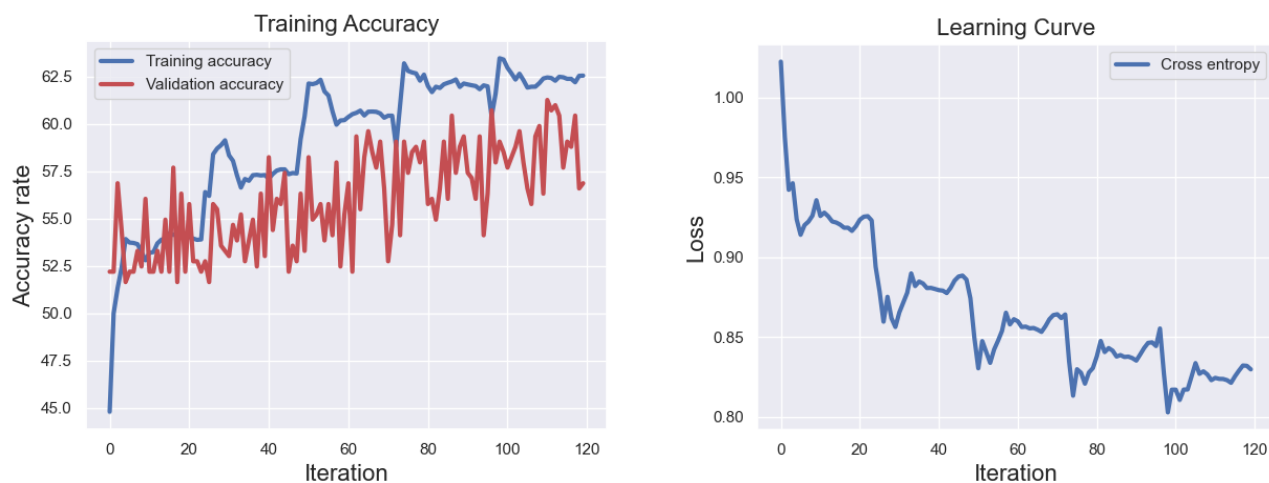
5, Result:

In our case, the CNN reached an accuracy of 62.55% on the training set and an accuracy of 59.918% on the test set.

While it is true that the model is rather simple, the time it took to train for 1 epoch is rather long, almost $\sim 24 \times 80$ seconds.

This is why we have to keep our model as simple as possible, as complex model put a heavy strain on the CPU and took way longer to train.

The Training Accuracy and Learning Curve produced are illustrated as below:



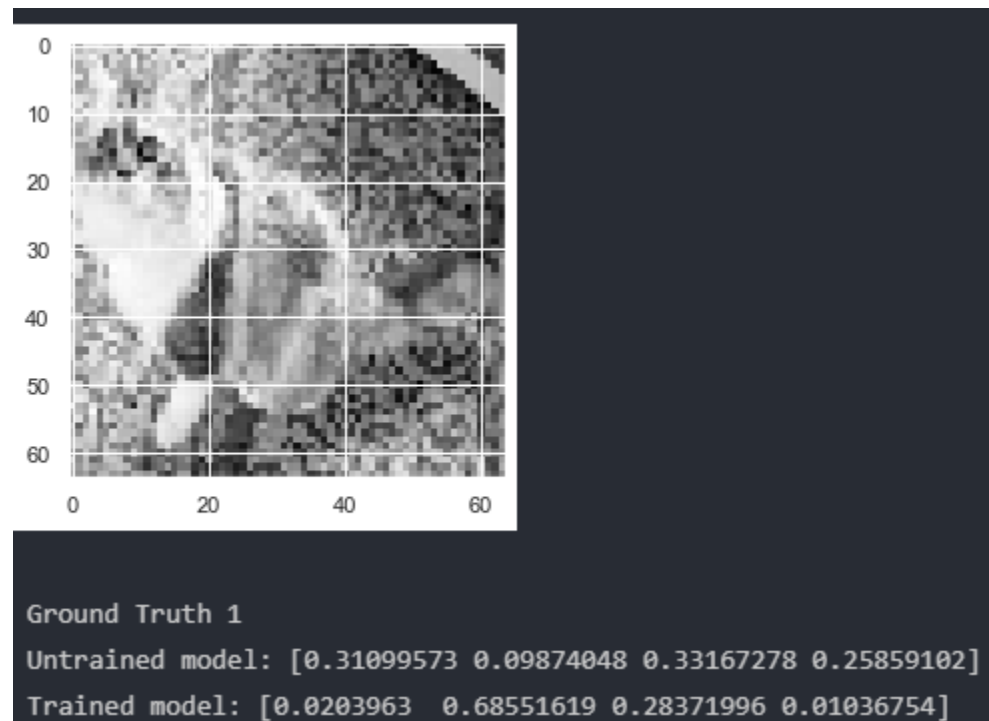
We can observe that the loss and accuracy curves are improving but at the same time they are constantly fluctuating, especially at each new epoch. This behavior can also be reduced by applying regularization. It's also true that we can further optimize the hyperparameter, for example the learning rate in this case, which can be raised from 0.005 to 0.007... etc.

This model can be further improved by training it more, but it is rather inefficient due to the complexity of the dataset. To tackle this, a more complex model using optimized library such as Keras, PyTorch is recommended as it is much more efficient due to the usage of GPU to compute.

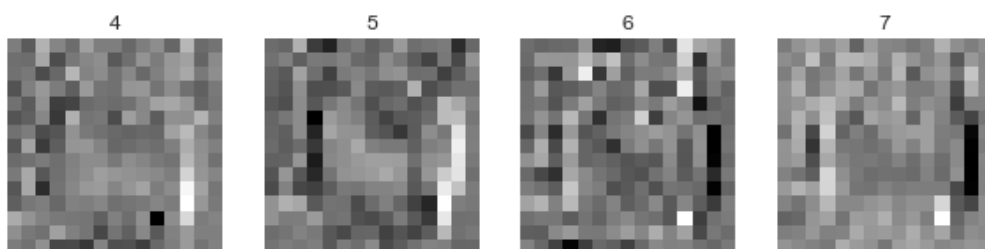
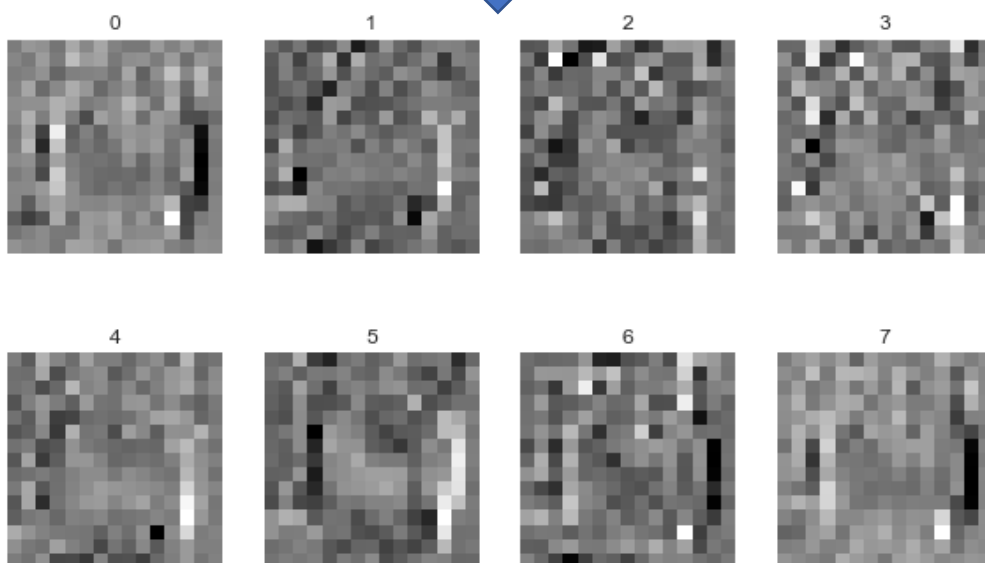
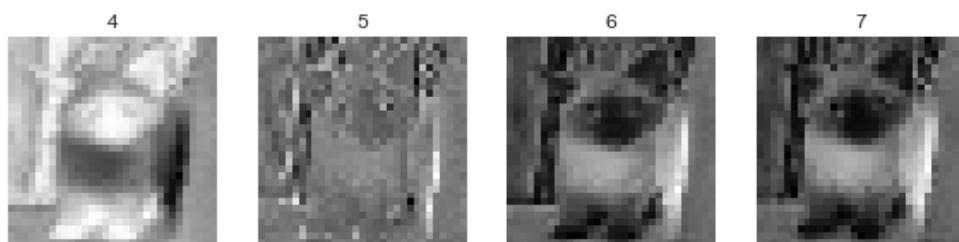
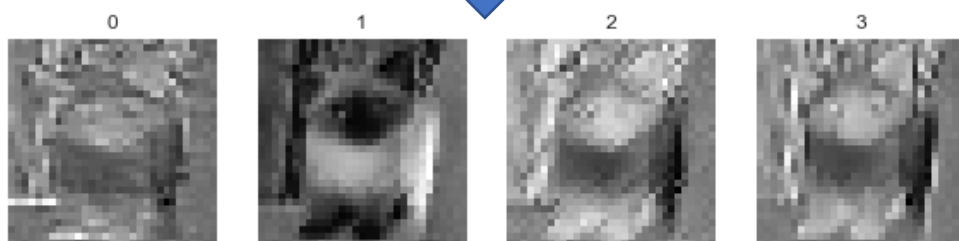
Example of a confused detection:



Example of a correct detection:



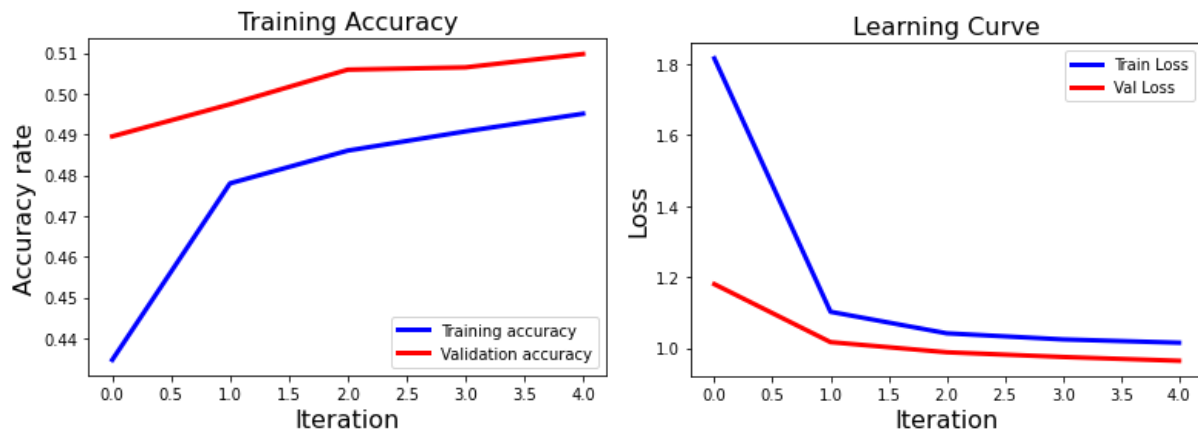
(Munchkin: 0 - Persian: 1 - Siamese: 2 - Sphynx - Hairless Cat: 3)



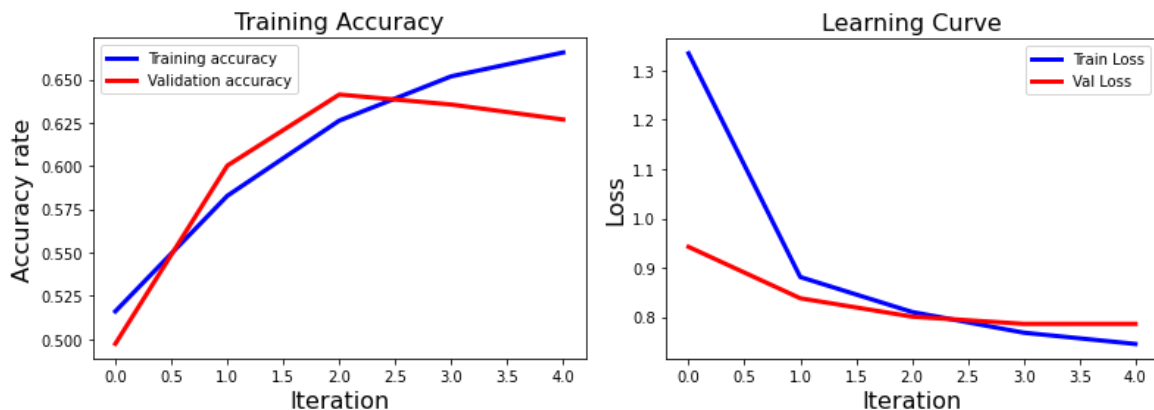
6, Improvements:

a, Learning rate:

As mentioned above, one of the issues with the model we deduced was the learning rate itself. Because the learning rate was high, the loss during the training process was very unstable. To overcome this, we decided to lower its value down.



By lowering the learning rate from 0.005 to 10^{-5} , we can see that the loss and learning curve was a lot more stable. However, the speed at which it is learning is still very slow. So, we decided to raise it a bit higher, to 10^{-4} .



Not only was the learning curve a lot more stable, the accuracy is also improving at a much faster rate than before. Therefore, the most optimized learning rate is at 10^{-4} .

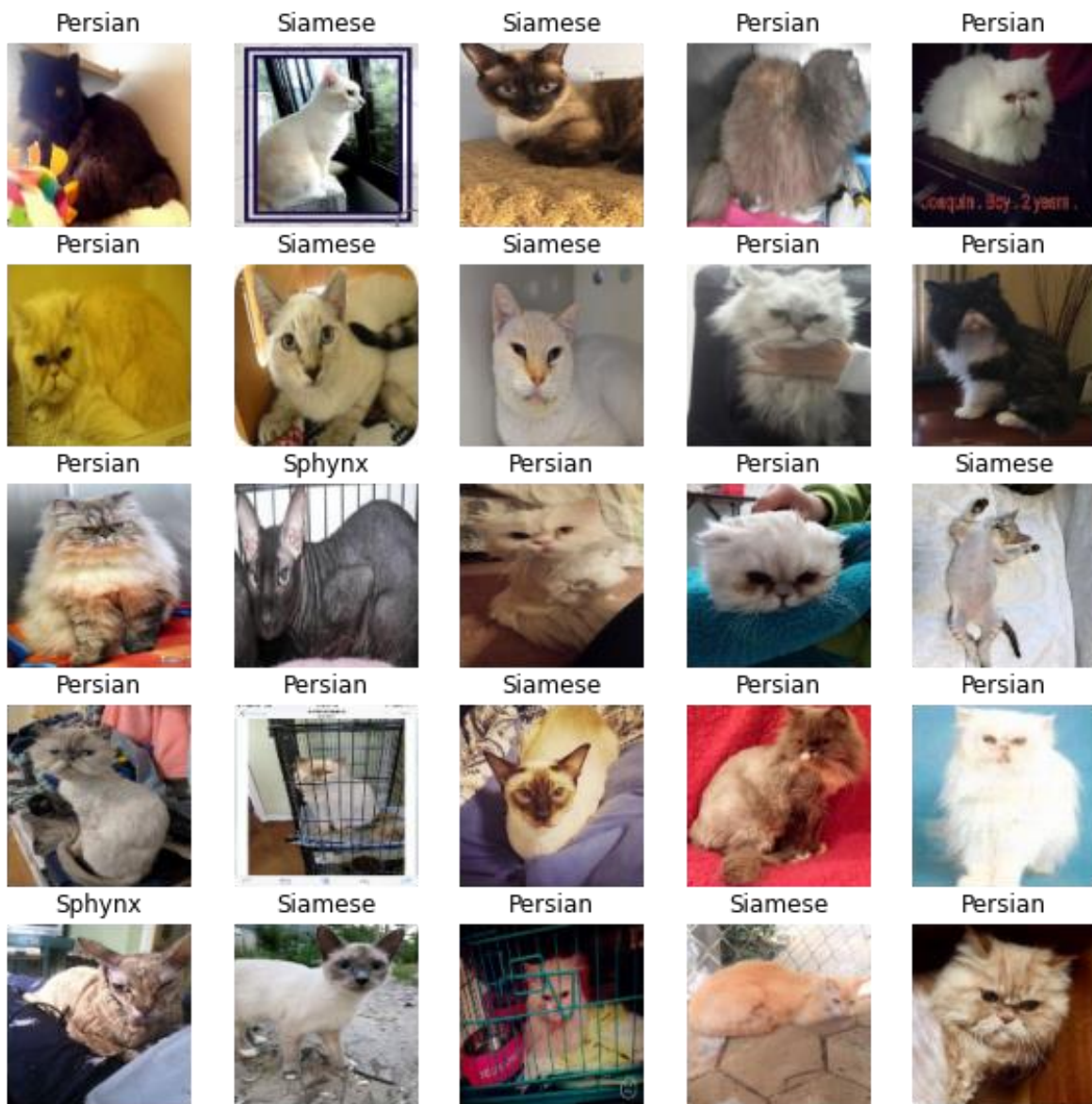
b, Architecture:

As the dataset itself was quite complex for a small and simple CNN to process, we decided to advance the model by increasing the total filters number in Convolutional Layers and total nodes number in the Dense layer. We also reimplemented [L2 regularization](#) with $\alpha = 10^{-4}$ and switched the [initialization method](#) from standard normal distribution to [Kaiming He](#):

$$std = \sqrt{2/N}$$

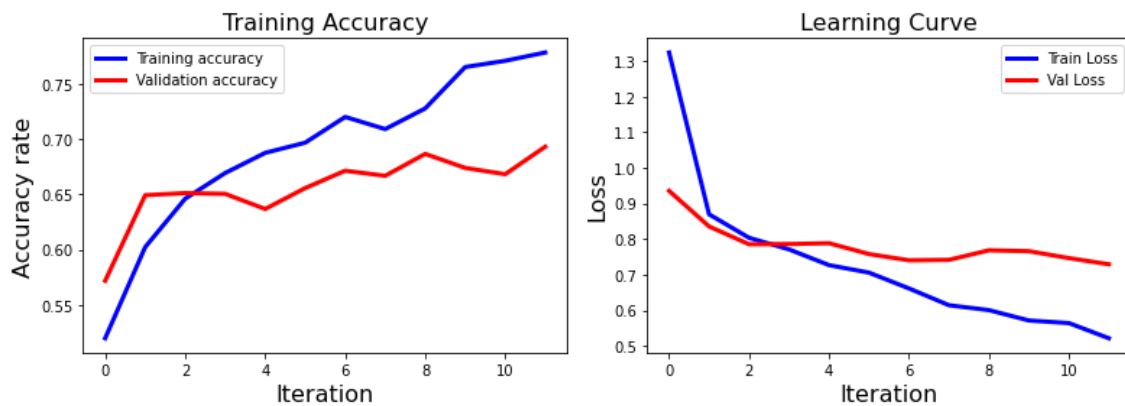
We also decided to raise the dataset resolution to 128x128 while also keeping the original color channel, which is RGB, so that the data will retain more features. This made the new input size:

[\[3x128x128\]](#)



Layer	Filter Size	Stride	Filters	Activation	Input Shape	Output Shape
Input	-	-	-	-	3x128x128	3x128x128
Convolutional	3	2	64	Leaky ReLU	3x128x128	64x63x63
Max Pooling	2	2	-	Leaky ReLU	64x63x63	64x31x31
Convolutional	3	1	192	Leaky ReLU	64x31x31	192x29x29
Max Pooling	2	2	-	Leaky ReLU	192x29x29	192x14x14
Dense	-	-	-	-	37,632	4
Output	-	-	-	-	4	4

Surprisingly, the model training process went very well. The model reached 79.19% accuracy on training set and 66.02% on validation set after only 8 epochs, which is a huge improvement compared to the older model.



On the test set, the model returned a very good result:

```
100%|██████████| 731/731 [00:48<00:00, 15.02it/s]
Test Loss: 0.752
Test Accuracy: 65.45
```

true label: Siamese (0.018)
pred label: Persian (0.973)



true label: Siamese (0.056)
pred label: Persian (0.915)



true label: Persian (0.094)
pred label: Siamese (0.896)



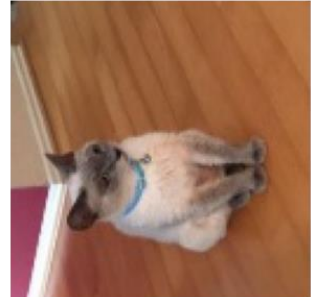
true label: Siamese (0.116)
pred label: Persian (0.865)



true label: Siamese (0.118)
pred label: Persian (0.843)



true label: Siamese (0.145)
pred label: Persian (0.836)



true label: Persian (0.151)
pred label: Siamese (0.836)



true label: Persian (0.158)
pred label: Siamese (0.830)



true label: Persian (0.162)
pred label: Siamese (0.819)



Confused cases

Some of the worst mistaken cases are shown above. As we can see, the model can mistake Persian for Siamese and also Siamese for Persian very easily, as they have a lot of resembling features: long and curly furs, sharp ears, ...

true label: Persian
pred label: Persian (0.625)



true label: Persian
pred label: Persian (0.966)



true label: Persian
pred label: Persian (0.653)



true label: Persian
pred label: Persian (0.597)



true label: Persian
pred label: Persian (0.701)



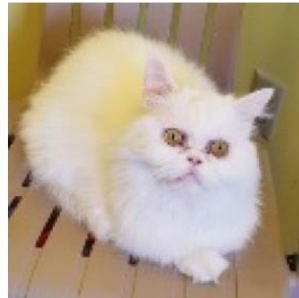
true label: Persian
pred label: Persian (0.508)



true label: Persian
pred label: Persian (0.499)



true label: Persian
pred label: Persian (0.921)



true label: Persian
pred label: Persian (0.859)



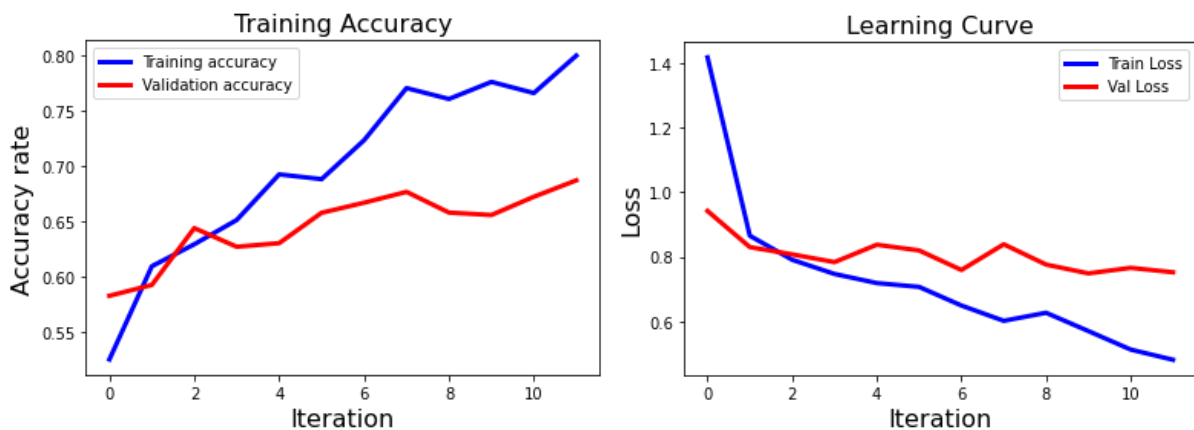
Predictions on test set

So, as creating a more complex model did indeed result in a much better accuracy and loss while halving the time needed to reach it, what if we make it even more complex?

Here, we further added in another Convolutional Layer paired with a Max Pooling Layer, then another Dense Layer at the last part:

Layer	Filter Size	Stride	Filters	Activation	Input Shape	Output Shape
Input	-	-	-	-	3x128x128	3x128x128
Convolutional	3	2	64	Leaky ReLU	3x128x128	64x63x63
Max Pooling	2	2	-	Leaky ReLU	64x63x63	64x31x31
Convolutional	3	1	192	Leaky ReLU	64x31x31	192x29x29
Max Pooling	2	2	-	Leaky ReLU	192x29x29	192x14x14
Convolutional	3	1	288	Leaky ReLU	192x14x14	288x12x12
Max Pooling	2	2	-	Leaky ReLU	288x12x12	288*6*6
Dense	-	-	-	-	10,368	4096
Dense	-	-	-	-	4096	2048
Output	-	-	-	-	2048	4

However, the result from the training process was not that better than before. While it was able to reach 83.98% on the training set and 68.32% on the validation set, which are only raised by a small margin, the model itself become quite complicated. It also seems that the model has already reached a limit and will only be overfitted if training continued.

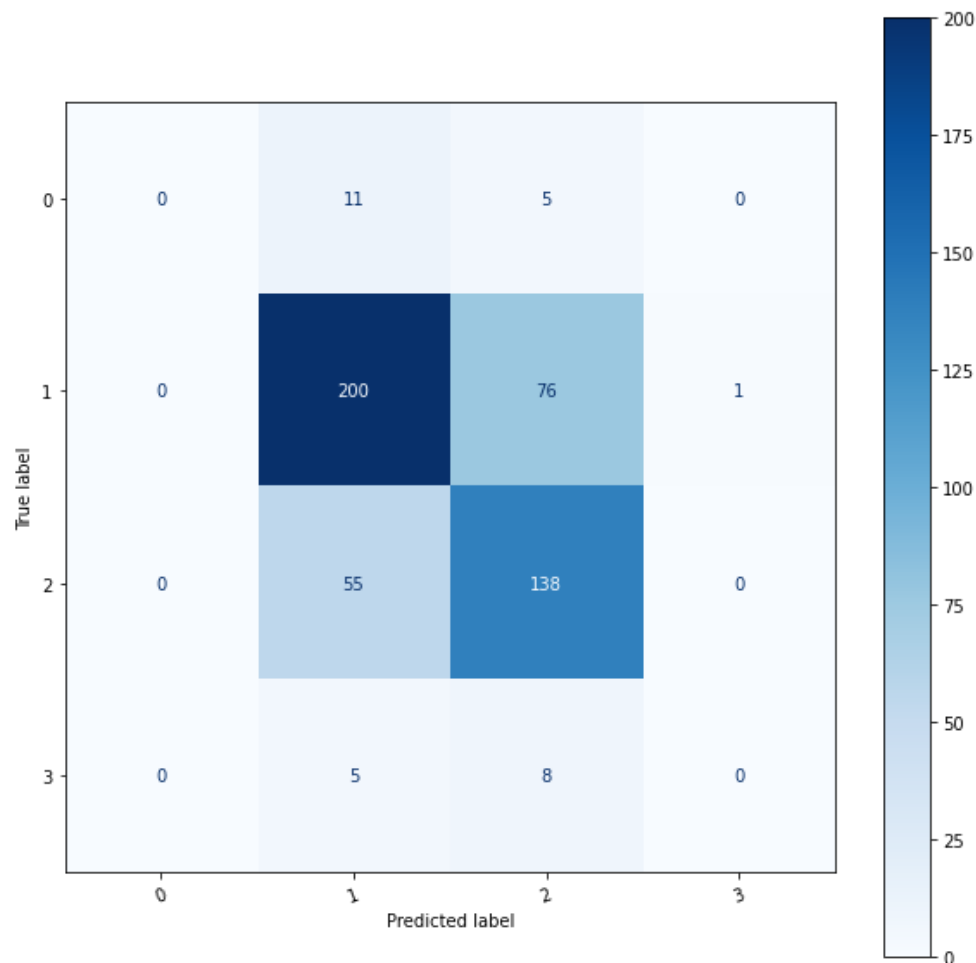


100% | 731/731 [00:48<00:00, 15.02it/s]

Test Loss: 0.705

Test Accuracy: 67.43

c, Data:



With the confusion matrix above, we can easily see that the model does indeed easily mistake class 1 for class 2 and likewise. It's also noteworthy that class 0 and class 3 however, has almost none corrected prediction.

We deduced that the problem lies in the data, and it was undoubtedly correct. The dataset itself has only a small amount of class 0 and class 3, which is most likely an issue when crawling data:

1. Munchkin = 181 images.
2. Persian = 4,018 images.
3. Siamese = 2,888 images.
4. Sphynx = 209 images.

So, to fix this, we crawled additional images for the 2 classes Munchkin and Sphynx.

7, Final words:

In conclusions, the task of classifying cat breeds can be challenging, especially with many identical breeds. However, with even more data available, the room for future improvement is wide and far.

Optimizations are definitely a must to best utilize a CNN, as with good hyperparameters and optimization, a model can converge earlier with better results.

Data used must also be accurate and clear, transparent enough. It's also noteworthy that the ratio between each class in a dataset must be equal or almost equal, thus preventing the problem when one class "outshine" other classes simply because the model was trained on that class more.

We also learnt that a good CNN is not necessarily a deep and complex one, but rather one that is most suitable for the job while still being simple enough to be lightweight and fast.