Cat Breeds Classification

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

1, Problem:

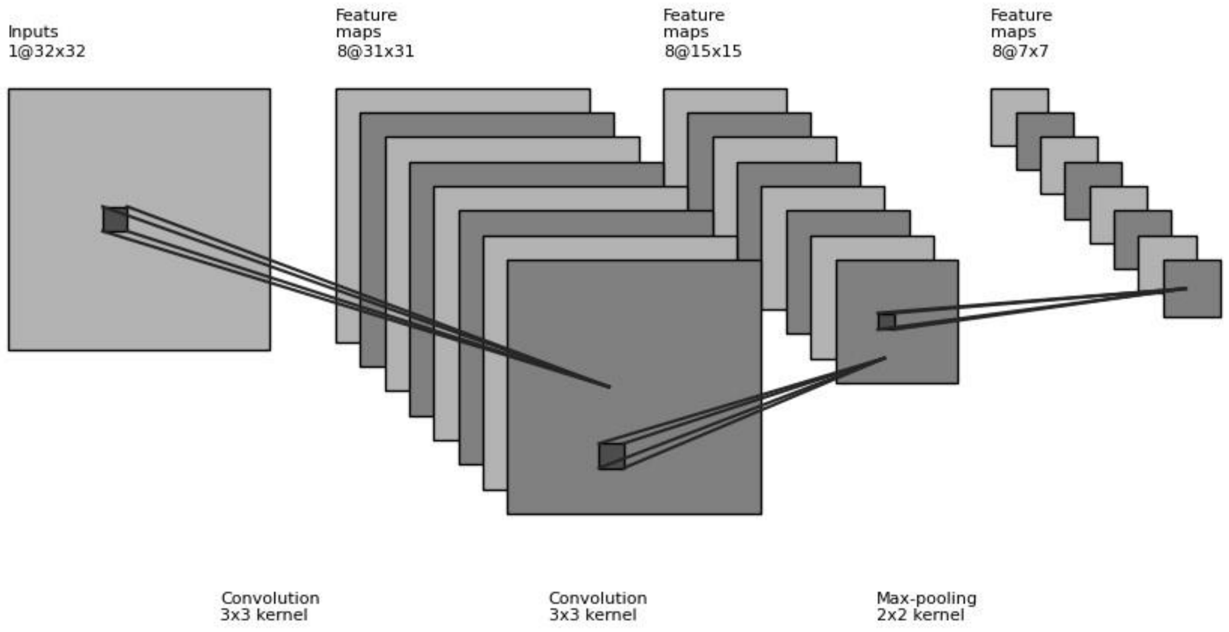
Given images of cats, the model is expected to give a prediction of the cat's breed in 1 of the 4 breeds below:

Munchkin	Persian
Siamese	**Sphynx**

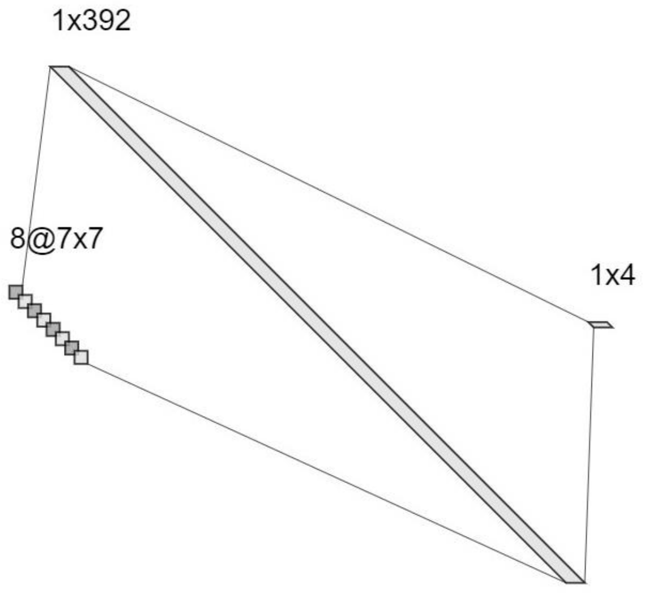
(Munchkin: 0 - Persian: 1 - Siamese: 2 - Sphynx - Hairless Cat: 3)

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:

$$\mathrm{a}_i = \frac{\mathrm{e}^{\mathrm{z}_j}}{\sum_{j=1}^{C} \mathrm{e}^{\mathrm{z}_j}} \qquad \forall \mathrm{i} = 1, 2, \ldots, \mathrm{C}$$

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:

$$CE = -\log(\mathrm{a}_i)$$

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:

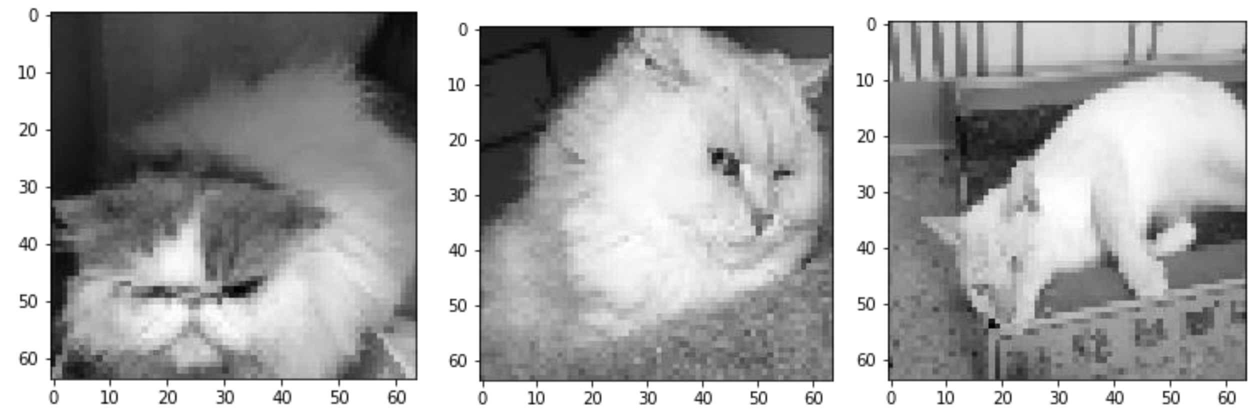
$$f(x) = \begin{cases} x & for\ x \geq 0 \\ \alpha x & for\ x < 0 \end{cases}$$

Which mean the derivative of it is:

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

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:

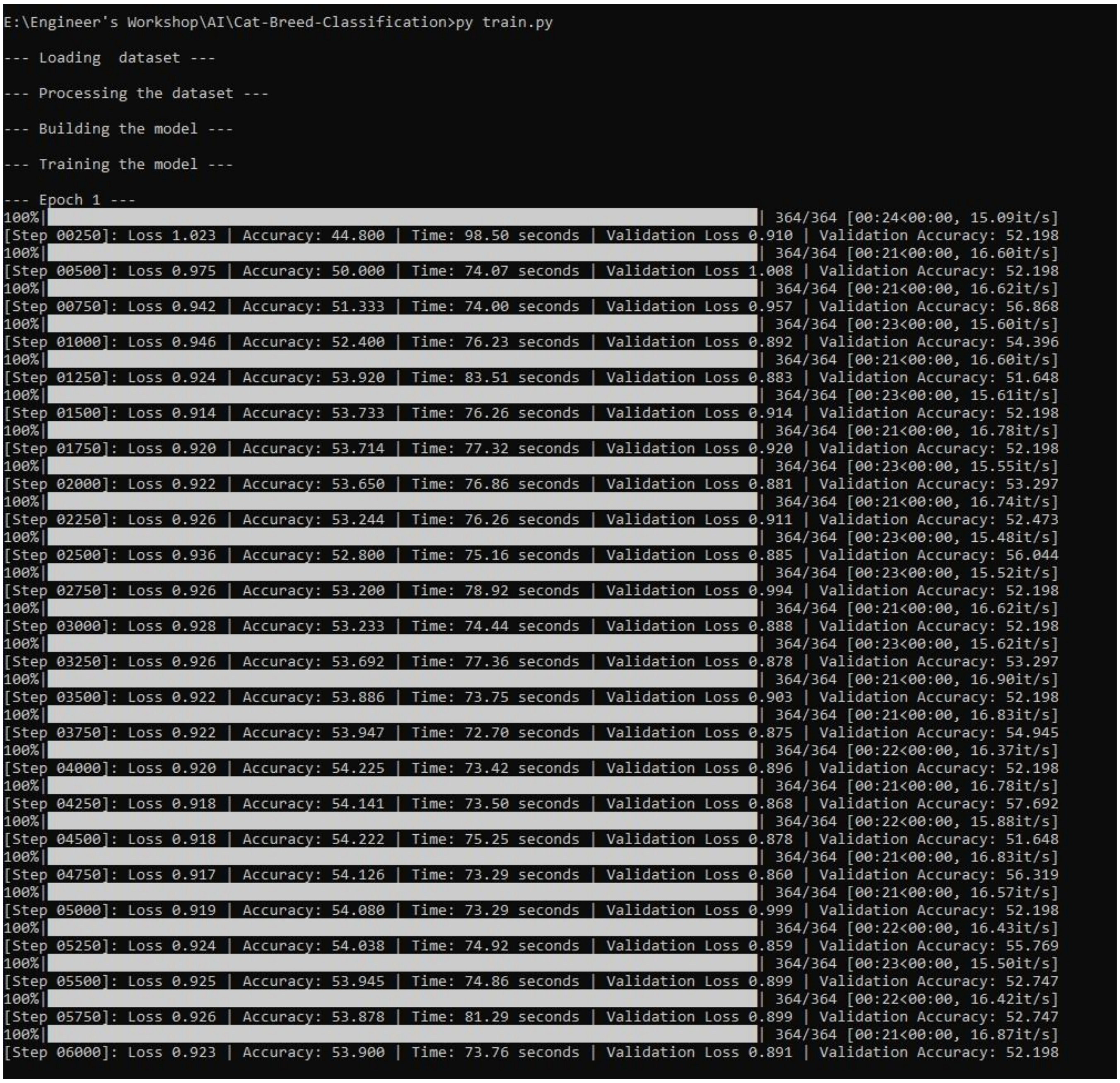
- Train batch: 85% = 6201
- Validation batch: 5% = 364
- Test batch: 10% = 731

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.

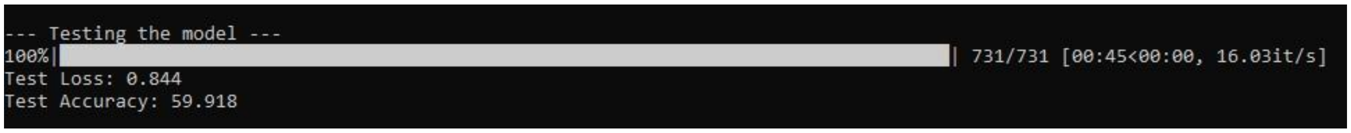
- Images: (64,64) -> Image: (1,64,64)

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.



```
E:\Engineer's Workshop\AI\Cat-Breed-Classification>py train.py

--- Loading  dataset ---

--- Processing the dataset ---

--- Building the model ---

--- Training the model ---

--- Epoch 1 ---
100%|████████████████████████████████████████████████████████████████| 364/364 [00:24<00:00, 15.09it/s]
[Step 00250]: Loss 1.023 | Accuracy: 44.800 | Time: 98.50 seconds | Validation Loss 0.910 | Validation Accuracy: 52.198
100%|████████████████████████████████████████████████████████████████| 364/364 [00:21<00:00, 16.60it/s]
[Step 00500]: Loss 0.975 | Accuracy: 50.000 | Time: 74.07 seconds | Validation Loss 1.008 | Validation Accuracy: 52.198
100%|████████████████████████████████████████████████████████████████| 364/364 [00:21<00:00, 16.62it/s]
[Step 00750]: Loss 0.942 | Accuracy: 51.333 | Time: 74.00 seconds | Validation Loss 0.957 | Validation Accuracy: 56.868
100%|████████████████████████████████████████████████████████████████| 364/364 [00:23<00:00, 15.60it/s]
[Step 01000]: Loss 0.946 | Accuracy: 52.400 | Time: 76.23 seconds | Validation Loss 0.892 | Validation Accuracy: 54.396
100%|████████████████████████████████████████████████████████████████| 364/364 [00:21<00:00, 16.60it/s]
[Step 01250]: Loss 0.924 | Accuracy: 53.920 | Time: 83.51 seconds | Validation Loss 0.883 | Validation Accuracy: 51.648
100%|████████████████████████████████████████████████████████████████| 364/364 [00:23<00:00, 15.61it/s]
[Step 01500]: Loss 0.914 | Accuracy: 53.733 | Time: 76.26 seconds | Validation Loss 0.914 | Validation Accuracy: 52.198
100%|████████████████████████████████████████████████████████████████| 364/364 [00:21<00:00, 16.78it/s]
[Step 01750]: Loss 0.920 | Accuracy: 53.714 | Time: 77.32 seconds | Validation Loss 0.920 | Validation Accuracy: 52.198
100%|████████████████████████████████████████████████████████████████| 364/364 [00:23<00:00, 15.55it/s]
[Step 02000]: Loss 0.922 | Accuracy: 53.650 | Time: 76.86 seconds | Validation Loss 0.881 | Validation Accuracy: 53.297
100%|████████████████████████████████████████████████████████████████| 364/364 [00:21<00:00, 16.74it/s]
[Step 02250]: Loss 0.926 | Accuracy: 53.244 | Time: 76.26 seconds | Validation Loss 0.911 | Validation Accuracy: 52.473
100%|████████████████████████████████████████████████████████████████| 364/364 [00:23<00:00, 15.48it/s]
[Step 02500]: Loss 0.936 | Accuracy: 52.800 | Time: 75.16 seconds | Validation Loss 0.885 | Validation Accuracy: 56.044
100%|████████████████████████████████████████████████████████████████| 364/364 [00:23<00:00, 15.52it/s]
[Step 02750]: Loss 0.926 | Accuracy: 53.200 | Time: 78.92 seconds | Validation Loss 0.994 | Validation Accuracy: 52.198
100%|████████████████████████████████████████████████████████████████| 364/364 [00:21<00:00, 16.62it/s]
[Step 03000]: Loss 0.928 | Accuracy: 53.233 | Time: 74.44 seconds | Validation Loss 0.888 | Validation Accuracy: 52.198
100%|████████████████████████████████████████████████████████████████| 364/364 [00:23<00:00, 15.62it/s]
[Step 03250]: Loss 0.926 | Accuracy: 53.692 | Time: 77.36 seconds | Validation Loss 0.878 | Validation Accuracy: 53.297
100%|████████████████████████████████████████████████████████████████| 364/364 [00:21<00:00, 16.90it/s]
[Step 03500]: Loss 0.922 | Accuracy: 53.886 | Time: 73.75 seconds | Validation Loss 0.903 | Validation Accuracy: 52.198
100%|████████████████████████████████████████████████████████████████| 364/364 [00:21<00:00, 16.83it/s]
[Step 03750]: Loss 0.922 | Accuracy: 53.947 | Time: 72.70 seconds | Validation Loss 0.875 | Validation Accuracy: 54.945
100%|████████████████████████████████████████████████████████████████| 364/364 [00:22<00:00, 16.37it/s]
[Step 04000]: Loss 0.920 | Accuracy: 54.225 | Time: 73.42 seconds | Validation Loss 0.896 | Validation Accuracy: 52.198
100%|████████████████████████████████████████████████████████████████| 364/364 [00:21<00:00, 16.78it/s]
[Step 04250]: Loss 0.918 | Accuracy: 54.141 | Time: 73.50 seconds | Validation Loss 0.868 | Validation Accuracy: 57.692
100%|████████████████████████████████████████████████████████████████| 364/364 [00:22<00:00, 15.88it/s]
[Step 04500]: Loss 0.918 | Accuracy: 54.222 | Time: 75.25 seconds | Validation Loss 0.878 | Validation Accuracy: 51.648
100%|████████████████████████████████████████████████████████████████| 364/364 [00:21<00:00, 16.83it/s]
[Step 04750]: Loss 0.917 | Accuracy: 54.126 | Time: 73.29 seconds | Validation Loss 0.860 | Validation Accuracy: 56.319
100%|████████████████████████████████████████████████████████████████| 364/364 [00:21<00:00, 16.57it/s]
[Step 05000]: Loss 0.919 | Accuracy: 54.080 | Time: 73.29 seconds | Validation Loss 0.999 | Validation Accuracy: 52.198
100%|████████████████████████████████████████████████████████████████| 364/364 [00:22<00:00, 16.43it/s]
[Step 05250]: Loss 0.924 | Accuracy: 54.038 | Time: 74.92 seconds | Validation Loss 0.859 | Validation Accuracy: 55.769
100%|████████████████████████████████████████████████████████████████| 364/364 [00:23<00:00, 15.50it/s]
[Step 05500]: Loss 0.925 | Accuracy: 53.945 | Time: 74.86 seconds | Validation Loss 0.899 | Validation Accuracy: 52.747
100%|████████████████████████████████████████████████████████████████| 364/364 [00:22<00:00, 16.42it/s]
[Step 05750]: Loss 0.926 | Accuracy: 53.878 | Time: 81.29 seconds | Validation Loss 0.899 | Validation Accuracy: 52.747
100%|████████████████████████████████████████████████████████████████| 364/364 [00:21<00:00, 16.87it/s]
[Step 06000]: Loss 0.923 | Accuracy: 53.900 | Time: 73.76 seconds | Validation Loss 0.891 | Validation Accuracy: 52.198
```



```
--- Testing the model ---
100%|████████████████████████████████████████████████████████████████████| 731/731 [00:45<00:00, 16.03it/s]
Test Loss: 0.844
Test Accuracy: 59.918
```

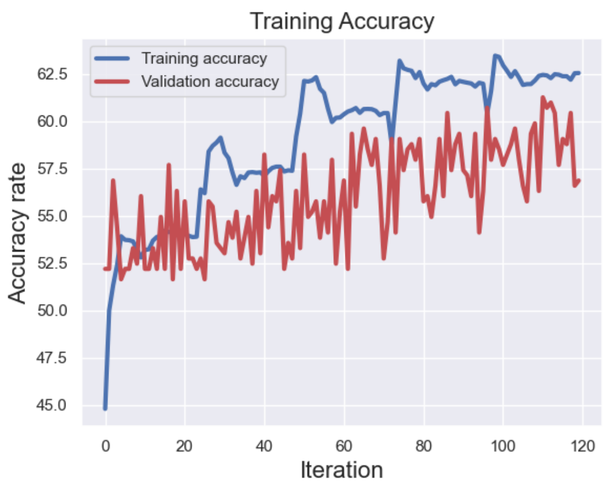
5, Result:

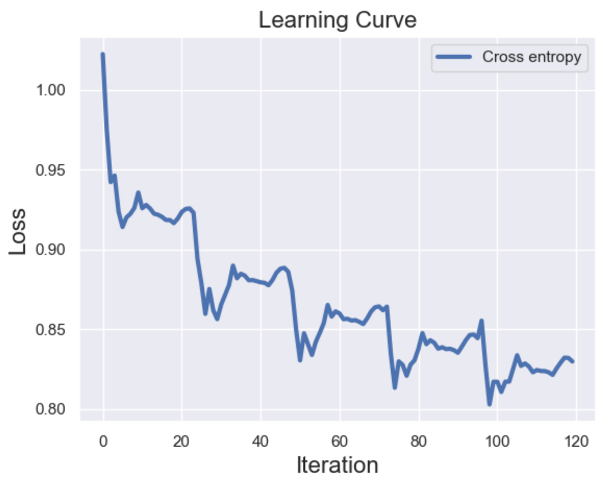
In our case, the CNN reached an accuracy of 62.55% on the training set and an accuracy of 59.918% on the test set.

While it is true that the model is rather simple, the time it took to train for 1 epoch is rather long, almost ~24x80 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:

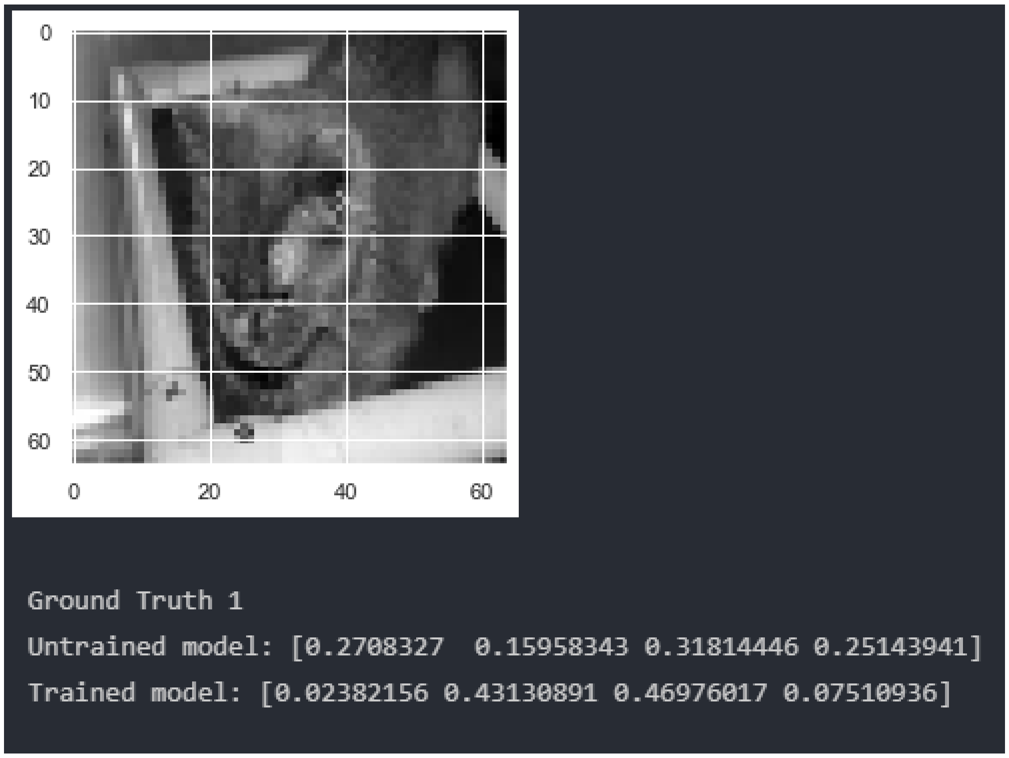




We can observe that the loss and accuracy curves are improving but at the same time they are constantly fluctuating, especially at each new epoch. This behavior can also be reduced by applying regularization. It's also true that we can further optimize the hyperparameter, for example the learning rate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

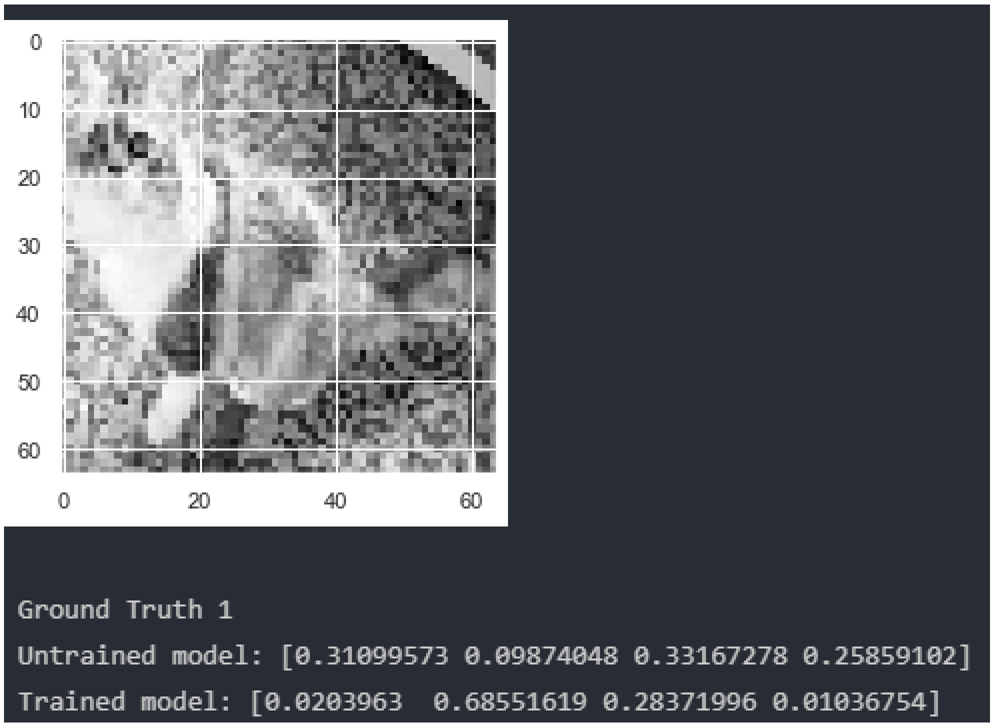
This model can be further improved by training it more, but it is rather inefficient due to the complexity of the dataset. To tackle this, a more complex model using optimized library such as Keras, PyTorch is recommended as it is much more efficient due to the usage of GPU to compute.

Example of a confused detection:



```
Ground Truth 1
Untrained model: [0.2708327  0.15958343 0.31814446 0.25143941]
Trained model: [0.02382156 0.43130891 0.46976017 0.07510936]
```

Example of a correct detection:



```
Ground Truth 1
Untrained model: [0.31099573 0.09874048 0.33167278 0.25859102]
Trained model: [0.0203963  0.68551619 0.28371996 0.01036754]
```

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