The basic architecture is convolutional neural network with no doubt because CNN is designed to analyze visual imagery and perform

the best in image classification. From that, we have tried a few CNN based architecture like traditional CNN, ResNet and denseNet.

Among these denseNet performs the most stable and accurate under the same condition. It has some draw backs like slow training and require

more epoch to see result, but it is not of our concern.

Therefore our final choice is denseNet with 3 dense block(5 densely connected convolution layer),

b.choice of loss function and optimizer:

For loss function, we have the choice between Cross-Entropy Loss and Negative Log-Likelihood Loss.

They are useful in classification problems. We suppose that the difference between them is NLLLoss requires Log SoftMax layer as the last layer,

however, the Pytorch Cross-Entropy loss function has these computed inside. Finally, we decided to use NLLLoss.

For optimizer, Adam is used instead of SGD since Adam is less sensitive to learning rate.

     The data we have is relatively small, therefore some data augmentations are applied. we decided to keep the 80\*80 size

and make other changes like crop, rotation, and color adjustment.

They have efficiently reduced overfitting and increase accuracy. The RandomHorizontalFlip function is commonly used on picture classification.

The RandomCrop function allowed more details of cats to be captured. The ColorJitter function improves approximately 2% accuracy (on our DenseNet).

We changed the number of epochs to be 5000 cause our model require around 500 epochs to see the results. Finish all these 5000 epochs take a

lot time so we didn't expect to finish all, instead we add feature that will saves the model to a folder and the output accuracy to a log file.

In this way, we can take a generally long training.

We start at a small batch\_size and keep increasing it, and we found that small batch size speed up the training but result in a unstable model.

And large vatch size slow the training but converge to a more stable model. Therefore we finaily choose batch\_size = 128 as a compromise.

Though it is suggest that we use the variable train\_val\_split to make design decisions aimed to avoid overfitting to the training data,

we didn't really change this variable at the early satge. In early stage, we mainly focus on decide the architecture of our model, change

train\_val\_split won't help with that and we agreed on 0.8 as a reasonable value. Later on, after we decided to use denseNet and some transformation

, we slightly raise this value to prevent overfitting.

Since 20 layer of denseNet is powerful enough and will quickly become overfitting in our previous trails, we applyed sevral method to avoid overfitting.

Add dropout layer is one method that helps a lot, we start by adding a couple around fully connected layers, and add dropout to every transition layer after

find it helps with resolve overfitting. Apply more transformation also helps and has been disscussed in previous answer. Reduce the complixity of the model

also helps to avoid overfitting. The number of layers and independent paramters is an important variable that we tuning a lot, during our pratice,

we have used it on all the different architectures to find a suitable capacity with not result in overfitting.