1. Model architecture:

   Input -> Conv2d1 -> Flatten -> Dense1 -> Dense2 -> output

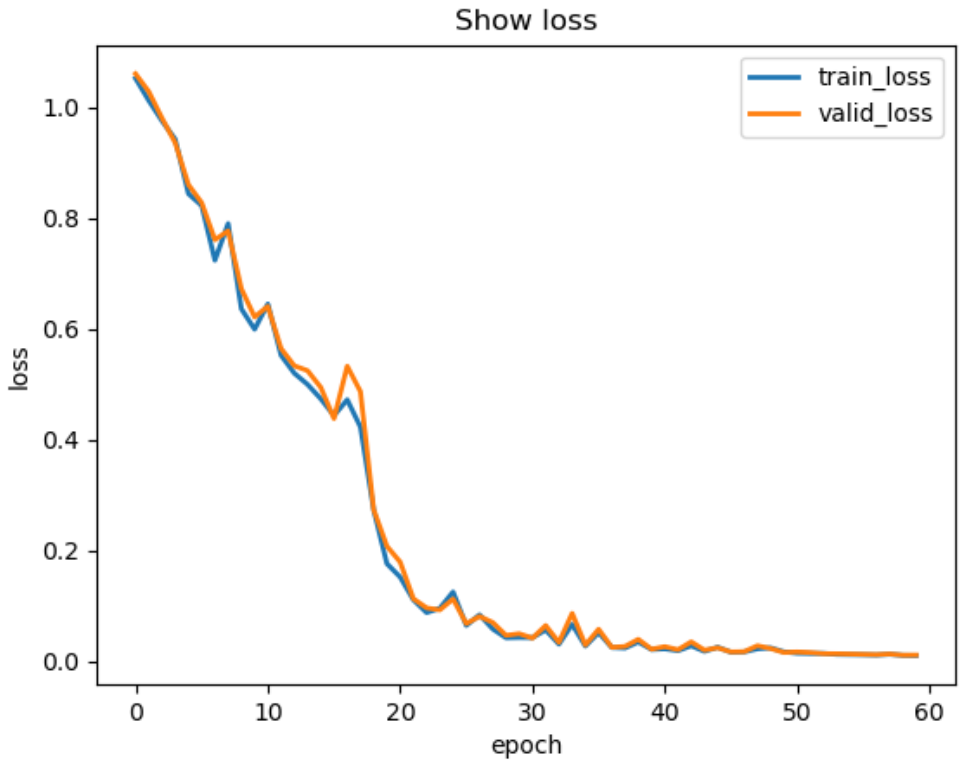
   Conv2d1: Kernel Size = 3. Stride = 1. No Padding. Doesn't use activation function.
   Input channel = 3, Output channel = 2.
   Dense1: output units = 100. Activation function use ReLU
   Dense2: output units = 3. Activation function use Softmax

   Loss function: CrossEncropy

Testing accuracy: 100%

```
epoch: 23, train_loss: 0.087671, validation_loss: 0.096168, val_acc: 99.5465
epoch: 24, train_loss: 0.095616, validation_loss: 0.092597, val_acc: 99.3197
epoch: 25, train_loss: 0.125055, validation_loss: 0.112304, val_acc: 98.6395
epoch: 26, train_loss: 0.064448, validation_loss: 0.067812, val_acc: 99.5465
epoch: 27, train_loss: 0.083790, validation_loss: 0.080882, val_acc: 99.7732
epoch: 28, train_loss: 0.058133, validation_loss: 0.069860, val_acc: 99.7732
epoch: 29, train_loss: 0.042174, validation_loss: 0.046567, val_acc: 100.0000
epoch: 30, train_loss: 0.042725, validation_loss: 0.049608, val_acc: 100.0000
epoch: 31, train_loss: 0.042437, validation_loss: 0.041604, val_acc: 99.5465
epoch: 32, train_loss: 0.056389, validation_loss: 0.064385, val_acc: 99.5465
epoch: 33, train_loss: 0.030676, validation_loss: 0.033865, val_acc: 100.0000
epoch: 34, train_loss: 0.066368, validation_loss: 0.086393, val_acc: 97.5057
epoch: 35, train_loss: 0.027242, validation_loss: 0.029227, val_acc: 100.0000
epoch: 36, train_loss: 0.051530, validation_loss: 0.057629, val_acc: 99.5465
epoch: 37, train_loss: 0.024611, validation_loss: 0.024915, val_acc: 100.0000
epoch: 38, train_loss: 0.023649, validation_loss: 0.026643, val_acc: 100.0000
epoch: 39, train_loss: 0.034619, validation_loss: 0.039441, val_acc: 100.0000
epoch: 40, train_loss: 0.021076, validation_loss: 0.021599, val_acc: 100.0000
epoch: 41, train_loss: 0.022225, validation_loss: 0.026255, val_acc: 100.0000
epoch: 42, train_loss: 0.018902, validation_loss: 0.020702, val_acc: 100.0000
epoch: 43, train_loss: 0.027591, validation_loss: 0.034886, val_acc: 100.0000
epoch: 44, train_loss: 0.017677, validation_loss: 0.019919, val_acc: 100.0000
epoch: 45, train_loss: 0.025247, validation_loss: 0.024235, val_acc: 100.0000
epoch: 46, train_loss: 0.016168, validation_loss: 0.016670, val_acc: 100.0000
epoch: 47, train_loss: 0.016714, validation_loss: 0.017081, val_acc: 100.0000
epoch: 48, train_loss: 0.022249, validation_loss: 0.027860, val_acc: 100.0000
epoch: 49, train_loss: 0.024047, validation_loss: 0.022878, val_acc: 100.0000
epoch: 50, train_loss: 0.016829, validation_loss: 0.015719, val_acc: 100.0000
epoch: 51, train_loss: 0.014382, validation_loss: 0.016627, val_acc: 100.0000
epoch: 52, train_loss: 0.013923, validation_loss: 0.015571, val_acc: 100.0000
epoch: 53, train_loss: 0.014028, validation_loss: 0.013563, val_acc: 100.0000
epoch: 54, train_loss: 0.012250, validation_loss: 0.013373, val_acc: 100.0000
epoch: 55, train_loss: 0.011894, validation_loss: 0.012883, val_acc: 100.0000
epoch: 56, train_loss: 0.011651, validation_loss: 0.012329, val_acc: 100.0000
epoch: 57, train_loss: 0.011127, validation_loss: 0.011836, val_acc: 100.0000
epoch: 58, train_loss: 0.012487, validation_loss: 0.012912, val_acc: 100.0000
epoch: 59, train_loss: 0.010574, validation_loss: 0.010858, val_acc: 100.0000
epoch: 60, train_loss: 0.010219, validation_loss: 0.011017, val_acc: 100.0000
test_loss: 0.077449, test_acc: 100.0000
```

2. Plot training loss and validation loss



3. Discussion:

Do a lots of model testing. Use two convolution layer but the accuracy may not be better than only use one. Or after convolution use ReLU still not be better. I think model shouldn't be too complicated. This model architecture does best performance in my experiment.

Also, in the previous homework1 and homework2. My code looks dirty and get bad score, so this time I split layer to a file, functions to a file, model to a file, make main program more clean. In addition, I use a lots of class and some special check to make code flexible.

Major problem is how to do forward convolution faster. First, I iterate all batch, do a lots of for loops and convolution is very slow. And this make training so slow. So I simultaneously do all batch data, think more complicated and make forward faster only use numpy sum and add newaxis.

Backward and Padding are also a hard work, and padding make code trickier. And see other people article to understanding backward is same as forward convolution. Compute dx is just rotated weight, convolution with next layer gradient add padding and get current layer gradient. Compute dw is just input data * next layer gradient, and then convolution.

Finally, hope my code is clean enough to understand.