Final report CoE202

Student name: Yeskendir Assankul
Student ID: 20200808
Professor: Young-Gyu Yoon
Teaching Assistant: Seungjae Han

First run:

My first run was with the parameters and code as were given in the initial skeleton code to check whether code works, how it works and what results we get with given code. The given code is as follows:

The transformation of the data is just transforming to Tensor.

```
transform=transforms.Compose([
    transforms.ToTensor()
])
```

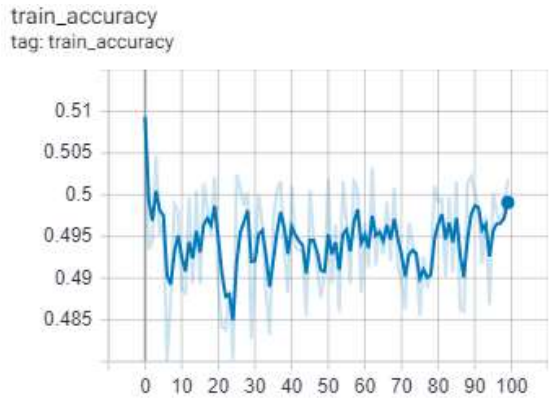
Then the model baseline code is as follows:

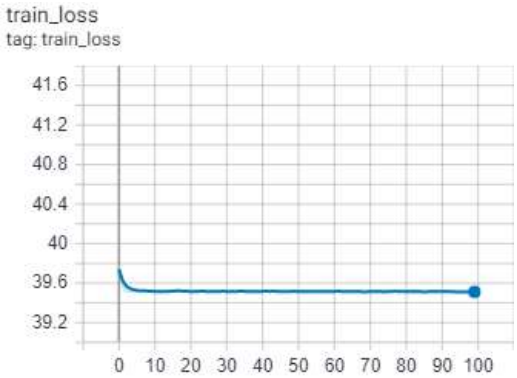
```
class cancer_classifier(nn.Module):
    def __init__(self):
        super(cancer_classifier, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=5, stride=1, padding=2)
        self.conv2 = nn.Conv2d(16, 16, kernel_size=5, stride=1, padding=2)
        self.conv3 = nn.Conv2d(16, 3, kernel_size=3, stride=1, padding=1)
        self.linear1 = nn.Linear(3*64*64, 64)
        self.linear2 = nn.Linear(64, 2)

    def forward(self, x):
        out = F.max_pool2d(F.relu(self.conv1(x)), kernel_size=2, stride=2)
        out = F.max_pool2d(F.relu(self.conv2(out)), kernel_size=2, stride=2)
        out = F.relu(self.conv3(out))
        out = out.reshape([-1, 3*64*64])
        out = self.linear1(out)
        out = self.linear2(out)
        return out
```

Where we can see that it is from the 3 convolutional layers, rgb input (3) and double output (2). First convolutional layer with input layers 3 (RGB), output layers 16 and keras(filter) 5 and padding 2. Second convolutional layer with input layers 16, output layers 16 and keras(filter) 5 and padding 2. Third convolutional layer with input layers 16, output layers 3 and keras(filter) 3 and padding 1. First fully connected layer with input 3*64*64 that was picked by determining the shape x [512,3,64,64], and output 64. Second fully connected layer with input 64 (that is defined from the previous layer) and output 2 as we should recognize 2 states. The number of epochs is 100.

As a result we got:

```
The final validation accuracy is 47.917 [%]
The best validation accuracy was 52.083 [%]
```





Therefore, we identified that the basic model with 3 convolutional layers and 2 fully connected layers do not work for that case. So, we have to have a more powerful model – implement the pretrained neural networks such as Residual (ResNet). Also, to make the model more general (for different, various images) we have to use the Image Augmentation methods, such as rotating or flipping and etc.

Second run:

Now for the second run and our first model we add changes that consider:

1) Optimize the network design
2) Optimize the hyper-parameters (changing number of epochs)
3) Use an appropriate optimization method (Adam optimizer)
4) Employ data augmentation (hint: rotate & flip)
5) Employ regularization methods
6) Employ ResNet architecture (as was suggested in the final project document)

To implement the changes we at first change our datasets by randomly rotating and flipping them, rotating and cropping, so we implement Image Augmentation – generating new images for training that were generated by changing the existing images:

```
  transform=transforms.Compose([
    transforms.RandomRotation((-45,45), resample=False, expand=False, center=None, fill=None),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomCrop(200),
    transforms.ToTensor()
])
```

We need Image Augmentation to increase the variation of data, so we prepare our model for different possible cases – increase its performance. Here we rotate the image on the angles from -45 to 45 to increase the variety of the images (larger angles does not have a significant effect, as the images have no distinctive upper and lower parts as human or dog). Then we random horizontal flip and vertical flip with standard probability of image being flipped 0.5. And then we randomly crop the image to a size of 200x200 pixels.
I added to the model_baseline.py the function cancer_classifier_function() to class cancer_classifier() to use the pretrained resnet50 model:

```
def cancer_classifier_function():
    import torchvision
    model = torchvision.models.resnet50(pretrained=True).cuda()
    return model
```

I used the resnet 50 as it has the approximately low error from other types of resnet pretrained models. It has 50 layers and makes networks strictly more expressive. “One of the biggest advantages of the ResNet is while increasing network depth, it avoids negative outcomes. So we can increase the depth but we have fast training and higher accuracy”[1]

Model structure	Top-1 error	Top-5 error
resnet18	30.24	10.92
resnet34	26.70	8.58
resnet50	23.85	7.13
resnet101	22.63	6.44
resnet152	21.69	5.94

Also, we changed the number of epochs to 10 as the resnet is very computational model (as it is our first run for pretrained model, we will change that after first trial) and also, we wrote the code for lr_scheduler that is changing lr according to the validation loss; however, we did not included it in training as we do not know whether validation loss is high or low compared to training loss.

```
experiment = "first_run"  # You can change this experiment name
epochs = 10
criterion = nn.CrossEntropyLoss()
model = cancer_classifier.cancer_classifier_function()
optim = torch.optim.Adam(model.parameters())
from torch.optim import lr_scheduler
exp_lr_scheduler = lr_scheduler.ReduceLROnPlateau(optim, 'min')
best_model_state, best_optim_state = None, None

if not os.path.exists(f"{filepath}/{experiment}"):
    os.mkdir(f"{filepath}/{experiment}")
if not os.path.exists(f"{filepath}/{experiment}/visualize"):
    os.mkdir(f"{filepath}/{experiment}/visualize")
```

Also, we used the Adam optimizer as it is an adaptive optimizer working well with noisy gradients. “The Adam algorithm calculates an exponential weighted moving average of the gradient and then squares the calculated gradient. This algorithm has two decay parameters that control the decay rates of these calculated moving averages.”[2]. It works well with noisy gradients, large datasets and requires little memory space.

```
optim = torch.optim.Adam(model.parameters())
```

The results we got are:

```
The final validation accuracy is 57.75 [%]
The best validation accuracy was 96.0 [%]
```

```
2020-12-13 15:37:01 || [0/10], train_loss = 34.9279, train_accuracy = 0.77, valid_loss = 2.5601, valid_accuracy = 0.71
2020-12-13 15:37:50 || [1/10], train_loss = 10.9480, train_accuracy = 0.88, valid_loss = 1.1195, valid_accuracy = 0.89
2020-12-13 15:38:40 || [2/10], train_loss = 10.5160, train_accuracy = 0.88, valid_loss = 2.8704, valid_accuracy = 0.71
2020-12-13 15:39:30 || [3/10], train_loss = 8.5870, train_accuracy = 0.91, valid_loss = 0.8791, valid_accuracy = 0.92
2020-12-13 15:40:20 || [4/10], train_loss = 8.3674, train_accuracy = 0.91, valid_loss = 0.9127, valid_accuracy = 0.91
2020-12-13 15:41:10 || [5/10], train_loss = 7.5670, train_accuracy = 0.92, valid_loss = 4.3854, valid_accuracy = 0.59
2020-12-13 15:42:00 || [6/10], train_loss = 7.7819, train_accuracy = 0.92, valid_loss = 0.6579, valid_accuracy = 0.96
2020-12-13 15:42:50 || [7/10], train_loss = 6.7769, train_accuracy = 0.93, valid_loss = 5.6555, valid_accuracy = 0.69
2020-12-13 15:43:40 || [8/10], train_loss = 6.6158, train_accuracy = 0.93, valid_loss = 6.1755, valid_accuracy = 0.68
2020-12-13 15:44:30 || [9/10], train_loss = 7.4117, train_accuracy = 0.92, valid_loss = 12.6059, valid_accuracy = 0.58
End of training, elapsed time : 8.0 min 24.223357439041138 sec.
```

We can see here that the validation accuracy is fluctuating, while train accuracy is continuing increasing.

Therefore, in our next run we change the several things:

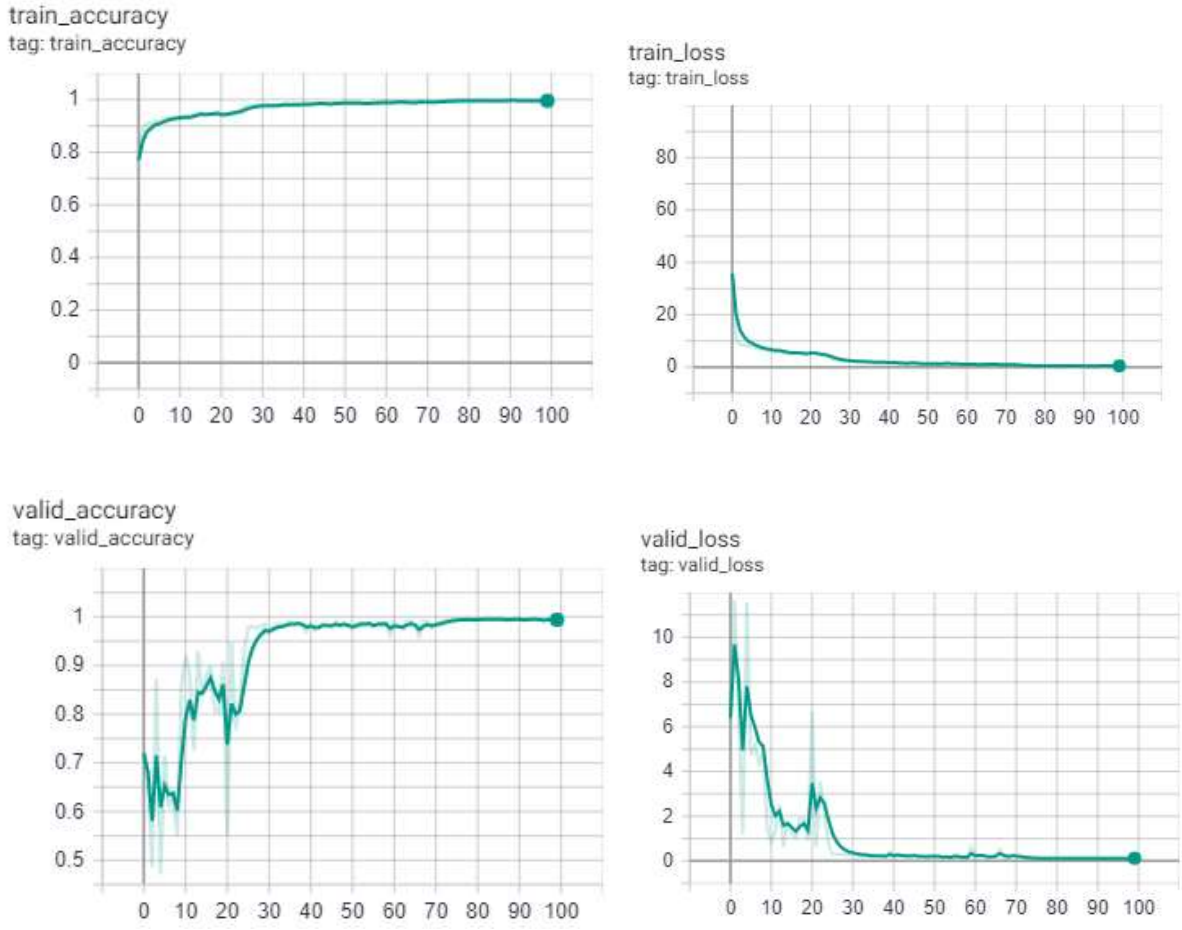
1) we add the lr_scheduler `lr_scheduler.ReduceLROnPlateau(optim, 'min')` to decrease the validation loss. “`ReduceLROnPlateau` allows dynamic learning rate reduction based on some validation measurements. Reduce learning rate when a metric has stopped improving. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This scheduler reads a metrics quantity and if no improvement is seen for a ‘patience’ number of epochs, the learning rate is reduced.”[3] We wrote `exp_lr_scheduler.step(valid_total_loss)` after each epoch.
2) Also, we increased the number of epochs to 100 as the train accuracy was continuously increasing and the overfitting did not occured.

Third run:

In the third run we implemented the changes above: added the scheduler `ReduceLROnPlateau` and increased the number of epochs to 100 and ran the training. The results are as follows:

```
2020-12-13 17:12:04 || [87/100], train_loss = 0.4868, train_accuracy = 1.00, valid_loss = 0.1181, valid_accuracy = 0.99
2020-12-13 17:12:53 || [88/100], train_loss = 0.5052, train_accuracy = 0.99, valid_loss = 0.1175, valid_accuracy = 1.00
2020-12-13 17:13:43 || [89/100], train_loss = 0.4158, train_accuracy = 1.00, valid_loss = 0.1150, valid_accuracy = 1.00
2020-12-13 17:14:32 || [90/100], train_loss = 0.3887, train_accuracy = 1.00, valid_loss = 0.1162, valid_accuracy = 1.00
2020-12-13 17:15:22 || [91/100], train_loss = 0.3786, train_accuracy = 1.00, valid_loss = 0.1213, valid_accuracy = 0.99
2020-12-13 17:16:11 || [92/100], train_loss = 0.4883, train_accuracy = 1.00, valid_loss = 0.1223, valid_accuracy = 1.00
2020-12-13 17:17:01 || [93/100], train_loss = 0.3752, train_accuracy = 1.00, valid_loss = 0.1156, valid_accuracy = 1.00
2020-12-13 17:17:51 || [94/100], train_loss = 0.5156, train_accuracy = 0.99, valid_loss = 0.1192, valid_accuracy = 1.00
2020-12-13 17:18:40 || [95/100], train_loss = 0.5661, train_accuracy = 0.99, valid_loss = 0.1164, valid_accuracy = 0.99
2020-12-13 17:19:30 || [96/100], train_loss = 0.5095, train_accuracy = 0.99, valid_loss = 0.1247, valid_accuracy = 0.99
2020-12-13 17:20:19 || [97/100], train_loss = 0.3791, train_accuracy = 1.00, valid_loss = 0.1187, valid_accuracy = 1.00
2020-12-13 17:21:09 || [98/100], train_loss = 0.3820, train_accuracy = 1.00, valid_loss = 0.1193, valid_accuracy = 0.99
2020-12-13 17:21:58 || [99/100], train_loss = 0.5447, train_accuracy = 0.99, valid_loss = 0.1205, valid_accuracy = 1.00
End of training, elapsed time : 82.0 min 49.336244344711304 sec.
```

Here we can see that the train accuracy and validation accuracy has reached 0.99-1.00 values! The tensorboard shows the result:



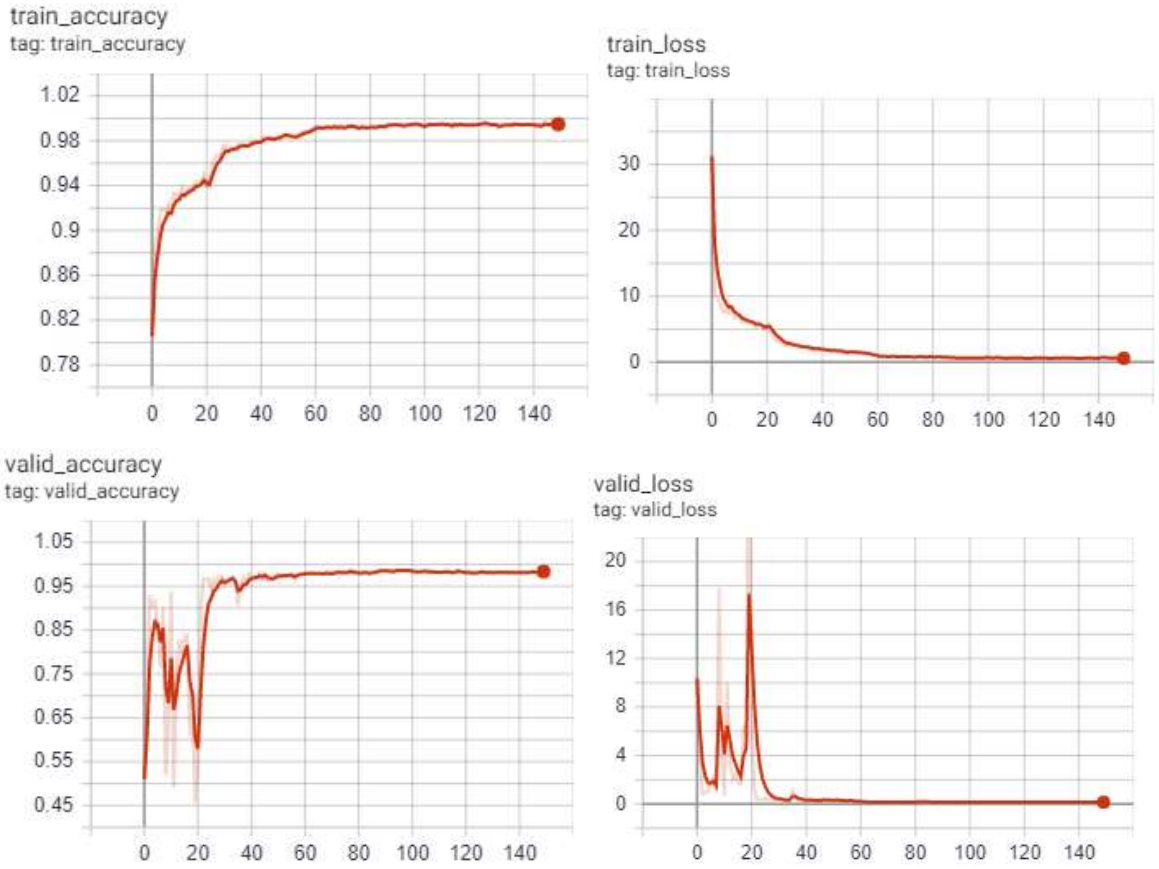
However, we still have some tendency for training loss to decrease and accuracy to increase by the epoch. Therefore, I conducted another run with changed parameters for the number of epochs. I increased it to 150.

Fourth run:

We used the same model with changed parameters of epochs to 150. And the results are as follows:

```
2020-12-13 21:20:12 || [125/150], train_loss = 0.6464, train_accuracy = 0.99, valid_loss = 0.1586, valid_accuracy = 0.98
2020-12-13 21:21:01 || [126/150], train_loss = 0.5327, train_accuracy = 0.99, valid_loss = 0.1565, valid_accuracy = 0.98
2020-12-13 21:21:49 || [127/150], train_loss = 0.6940, train_accuracy = 0.99, valid_loss = 0.1619, valid_accuracy = 0.98
2020-12-13 21:22:38 || [128/150], train_loss = 0.6153, train_accuracy = 0.99, valid_loss = 0.1604, valid_accuracy = 0.98
2020-12-13 21:23:26 || [129/150], train_loss = 0.5997, train_accuracy = 0.99, valid_loss = 0.1402, valid_accuracy = 0.98
2020-12-13 21:24:14 || [130/150], train_loss = 0.5574, train_accuracy = 0.99, valid_loss = 0.1692, valid_accuracy = 0.98
2020-12-13 21:25:02 || [131/150], train_loss = 0.6134, train_accuracy = 0.99, valid_loss = 0.1629, valid_accuracy = 0.98
2020-12-13 21:25:50 || [132/150], train_loss = 0.5574, train_accuracy = 0.99, valid_loss = 0.1616, valid_accuracy = 0.98
2020-12-13 21:26:38 || [133/150], train_loss = 0.5048, train_accuracy = 1.00, valid_loss = 0.1486, valid_accuracy = 0.98
2020-12-13 21:27:27 || [134/150], train_loss = 0.6394, train_accuracy = 0.99, valid_loss = 0.1576, valid_accuracy = 0.98
2020-12-13 21:28:16 || [135/150], train_loss = 0.5918, train_accuracy = 0.99, valid_loss = 0.1462, valid_accuracy = 0.98
2020-12-13 21:29:04 || [136/150], train_loss = 0.7470, train_accuracy = 0.99, valid_loss = 0.1621, valid_accuracy = 0.98
2020-12-13 21:29:53 || [137/150], train_loss = 0.6202, train_accuracy = 0.99, valid_loss = 0.1489, valid_accuracy = 0.98
2020-12-13 21:30:41 || [138/150], train_loss = 0.5556, train_accuracy = 0.99, valid_loss = 0.1463, valid_accuracy = 0.98
2020-12-13 21:31:29 || [139/150], train_loss = 0.5465, train_accuracy = 0.99, valid_loss = 0.1576, valid_accuracy = 0.98
2020-12-13 21:32:18 || [140/150], train_loss = 0.5972, train_accuracy = 0.99, valid_loss = 0.1465, valid_accuracy = 0.98
2020-12-13 21:33:07 || [141/150], train_loss = 0.8337, train_accuracy = 0.99, valid_loss = 0.1541, valid_accuracy = 0.98
2020-12-13 21:33:55 || [142/150], train_loss = 0.7014, train_accuracy = 0.99, valid_loss = 0.1560, valid_accuracy = 0.98
2020-12-13 21:34:44 || [143/150], train_loss = 0.6981, train_accuracy = 0.99, valid_loss = 0.1508, valid_accuracy = 0.98
2020-12-13 21:35:32 || [144/150], train_loss = 0.4624, train_accuracy = 1.00, valid_loss = 0.1485, valid_accuracy = 0.98
2020-12-13 21:36:21 || [145/150], train_loss = 0.5629, train_accuracy = 0.99, valid_loss = 0.1574, valid_accuracy = 0.98
2020-12-13 21:37:09 || [146/150], train_loss = 0.7473, train_accuracy = 0.99, valid_loss = 0.1581, valid_accuracy = 0.98
2020-12-13 21:37:58 || [147/150], train_loss = 0.3919, train_accuracy = 1.00, valid_loss = 0.1456, valid_accuracy = 0.98
2020-12-13 21:38:46 || [148/150], train_loss = 0.6477, train_accuracy = 0.99, valid_loss = 0.1586, valid_accuracy = 0.98
2020-12-13 21:39:35 || [149/150], train_loss = 0.5359, train_accuracy = 0.99, valid_loss = 0.1360, valid_accuracy = 0.98
End of training, elapsed time : 126.0 min 20.221678733825684 sec.
```

As we can see here the validation loss stopped changing at 136 epoch; however, the validation loss decreased. Therefore, we can say that this run was successful. The results from TensorBoard:



We can see from these graphs that our model successfully recognizes the images and the cancer cells - what was our goal. Then we evaluated it with eval.py and it shown results:

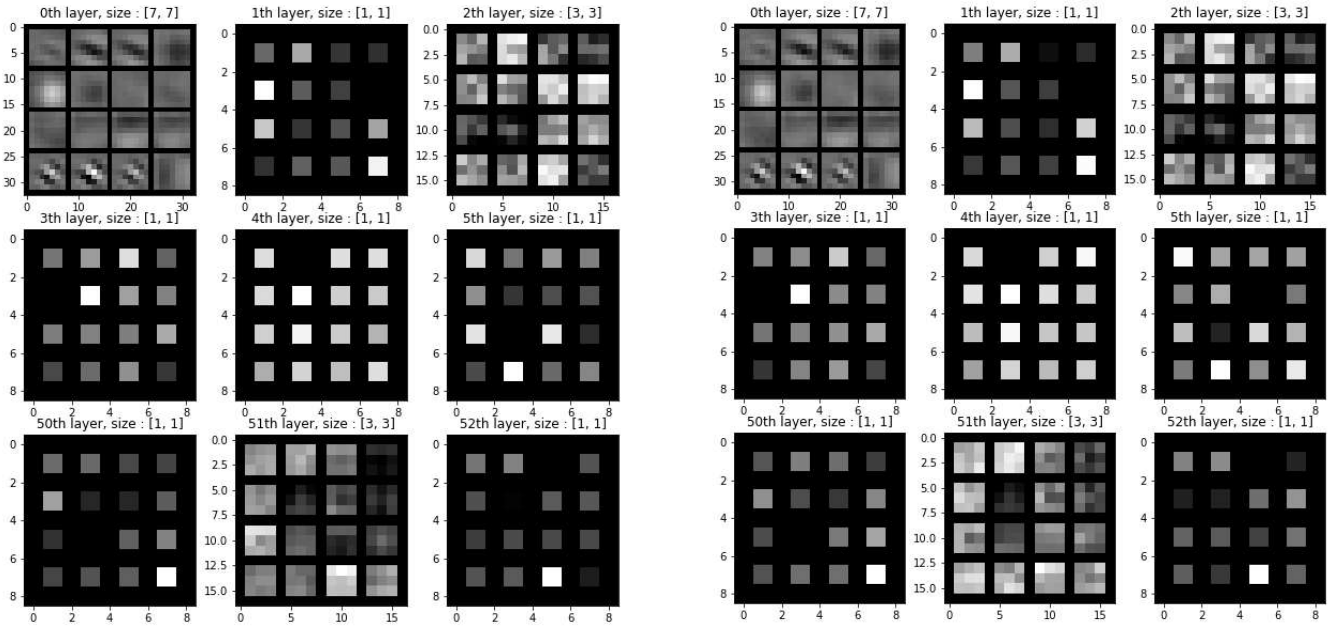
```
[0, 0, 1, 1, 1]
```

After submitting it to Kaggle it showed accuracy for testing samples 0.99346. Therefore, we can say that our model designed well and successfully applies to various cases.

The final version of the model and parameters that I was using are:

1) Adding the Image Augmentation methods: horizontal and vertical flipping, rotating, cropping. We need Image Augmentation to increase the variation of data, so we prepare our model for different possible cases – increase its performance. More detailed reason in part “Second run”.
2) The number of epochs that we used was changed to 150 (from 10 to 100, from 100 to 150). We started with a number of epochs equal to 10. At first, the training accuracy was continuously increasing; therefore, we increased it from 10 to 100. After the “Third run” we increased it to 150 as there was still a tendency of accuracy to increase. More detailed reasons in “Second run” and “Third run”.
3) We used the pretrained method Residual neural network with 50 layers. We implemented it as the model with 3 convolutional layers was not handling the data and showed maximal accuracy 55%. More detailed reasons in “First run” and “Second run”.
4) We used the `ReduceLROnPlateau` learning rate scheduler. `ReduceLROnPlateau` allows dynamic learning rate reduction based on some validation measurements. More detailed reasons in “Third run”.
5) We changed the batch_size to 100 from 32. To make the model converge faster and give better performance. More detailed reasons in “Third run”.
6) Using Adam optimizer. It works well with noisy gradients, large datasets and requires little memory space. More detailed reasons in “Second run”.

All layers before and after training, correspondingly:



References:

1. https://medium.com/@bakiiii/microsoft-presents-deep-residual-networks-d0ebd3fe5887
2. https://www.tech-quantum.com/adam-optimization-algorithms-in-deep-learning/
3. https://rdrr.io/cran/kerasR/man/ReduceLROnPlateau.html
4. https://pytorch.org/docs/stable/optim.html