# BBM418 – Computer Vision Laboratory

# Programming Assignment 3

# Image Classification with Convolutional Neural Networks

*Burak Özüesen – 21827761*

## *Part 0: Explaining how arranged dataset into train, validation and test:*

```
train_dataset, test_dataset = torch.utils.data.random_split(dataset, (2989, 1494))
```

Thanks to this line of code, we can separate all the data in the file directory that I have given as parameters, according to the parameters given randomly. This function is a feature of the PyTorch library. The numbers given as parameters (2989 and 1494 in this case) were set as the train 1/3 test set at the rate of 2/3. I did not have a validation set.

## *Part 1: Modeling and Training a CNN classifier from Scratch:*

```
self.myModel = nn.Sequential(
    nn.Conv2d(3, 16, kernel_size=3), nn.ReLU(),
    nn.MaxPool2d(2, 2),
    nn.Conv2d(16, 32, kernel_size=3), nn.ReLU(),
    nn.MaxPool2d(2, 2),
    nn.Conv2d(32, 64, kernel_size=3), nn.ReLU(),
    nn.MaxPool2d(2, 2),
    nn.Conv2d(64, 128, kernel_size=3), nn.ReLU(),
    nn.MaxPool2d(2, 2),
    nn.Conv2d(128, 256, kernel_size=3), nn.ReLU(),
    nn.MaxPool2d(2, 2),
    nn.Flatten(),
    nn.Dropout(0.25),
    nn.Linear(9216, 256),
    nn.ReLU(),
    nn.Dropout(0.5),
    nn.Linear(256, 15)
).to(device)
```

This section in my code shows a sequentially linked CNN structure. There are 5 Convutional Layers in total. There are 2 Linear filters and one of them outputs 15, which is our class number. After each convutional layer there is a ReLU and maxPool layer.

```
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
criterion = nn.CrossEntropyLoss()
```

I used the cross entropy loss method as a loss function.

Here is my residual forward function:

```
def forward(self, input):
    output = self.conv1(input)
    output = self.relu(output)
    output = self.pool(output)

    output = self.conv2(output)
    output = self.relu(output)
    output = self.pool(output)

    output = self.conv3(output)
    output = self.relu(output)
    output = self.pool(output)

    output = self.conv4(output)
    output = self.relu(output)
    output = self.pool(output)

    output = self.conv5(output)
    output = self.relu(output)
    output = self.pool(output)

    output = self.flatten(output)
    output = self.linear1(output)
    output = self.relu(output)

    output = self.linear2(output)
    return output
```

Here is the result of residual network with LR = 0.0001:

```
Epoch: [30], Test Loss: 1.8773, Accuracy: 0.42, Time: 50.94 sec
```

## *Part 1.1: Training and Evaluating your model:*

| ***Learning Rate*** | ***Type*** | ***Accuracy*** |
|---|---|---|
| ***0.0001*** | ***Normal*** | ***0.46*** |
| ***0.0001*** | ***Residual*** | ***0.42*** |

| | | |
|---|---|---|
| ***0.001*** | ***Normal*** | ***0.49*** |
| ***0.01*** | ***Normal*** | ***0.14*** |

I tried my models with 4 different parameters. Firstly I compared 0.0001 LR and normal model and residual type model. Normal type has better accuracy and I changed LR to 0.001 and calculated new accuracy which 0.49. After this phase I increased LR one more time and my accuracy decreased very much.

So, I decided to third experiments result is best model according to my workdone.

As note: So I changed some things to make my parameter experiments faster. First of all, instead of training with a validation set, I directly compared the test results. Secondly, I applied dropout addition in all my attempts at the beginning. Finally, I replaced my parameter experiments with a straightforward greedy approach.

## *Q - What is dropout and why it is important?*

***A -*** To summarize, we can say that it is used to forget some neurons in order to prevent overfitting during training. We must remember that if our network is too large, if we are training too long, or if we have too few data, you run the risk of over-learning.

## *Part 2: Transfer Learning with CNNs:*

## *Q – What is fine-tuning? Why should we do this? Why do we freeze the rest and train only FC layers?*

***A -*** Fine-tuning, in general, means making small adjustments to a process to achieve the desired output or performance. Fine-tuning deep learning involves using weights of a previous deep learning algorithm for programming another similar deep learning process. Whenever we are given the task of training a deep learning neural

network, we usually think of training it from scratch. Training a neural network is a time and resource-intensive process. The neural network needs to be fed tons of data for it to actually work as intended. Gathering the data for the neural network can take long periods of time. With fine-tuning, the deep learning neural networks already have most of the data available for the new model from previous ones. Thus, a lot of time and resources are saved when fine-tuning deep learning models is carried out. Once you've added or removed layers depending upon the data required, you must then freeze the layers in the new model. Freezing a layer means the layer doesn't need any modification to the data contained in them, henceforth. The weights for these layers don't update when we train the new model on the new data for the new task.

```
model = models.resnet18(pretrained=True)
for param in model.parameters():
    param.requires_grad = False
```

## *Part 2.2: Results and comparison of tests made with the RestNet model*

```
Epoch: [30], Test Loss: 2.7339, Accuracy: 0.07, Time: 87.11 sec
```

```
Epoch: [30], Test Loss: 2.8761, Accuracy: 0.07, Time: 84.43 sec
```

In the screenshots here, the first result is that all layers are frozen, while in the second screenshot the last layer is not frozen. However, there is no visible difference between the results.

| ***Learning Rate*** | ***Type*** | ***Accuracy*** |
|---|---|---|
| ***0.001*** | ***Normal*** | ***0.49*** |
| ***0.001*** | ***RestNet (All frozen)*** | ***0.07*** |
| ***0.001*** | ***RestNet (Last layer not frozen)*** | ***0.07*** |

My model, which I have implemented sequentialy, has a big advantage with equal learning speed .It obtained more efficient results than the RestNet model.

## *References:*

**[1]** https://medium.com/swlh/deep-learning-for-image-classification-creating-cnn-from-scratch-using-pytorch-d9eeb7039c12

**[2]** https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

**[3]** https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/

**[4]** https://github.com/aladdinpersson/Machine-Learning-Collection/blob/master/ML/Pytorch/Basics/pytorch_pretrain_finetune.py