M5-Visual Recognition
Week 1: PyTorch 101
Group 07

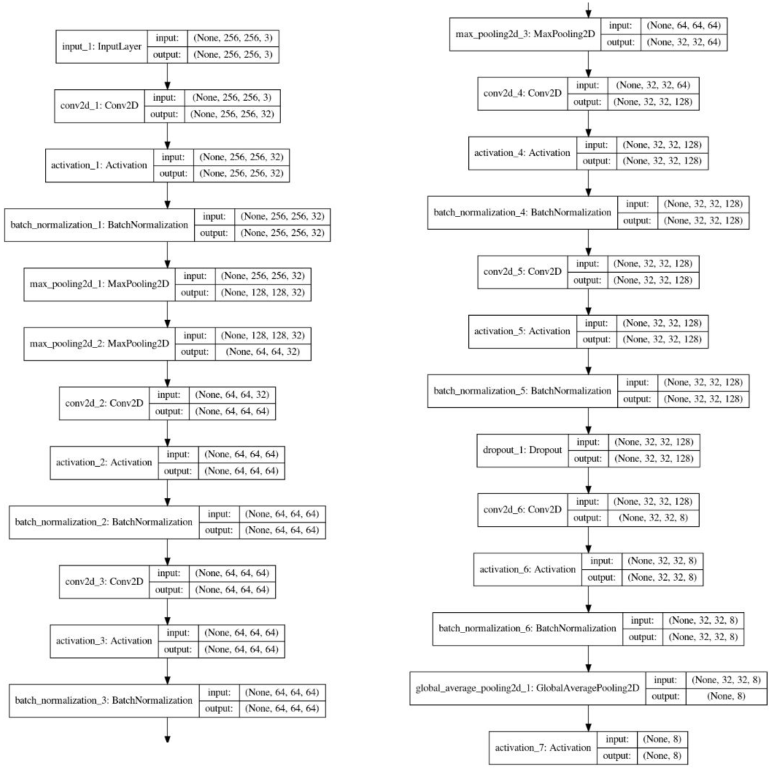
Lali Bibilashvili
Juan Chaves
Carmen García
Mohamed El Atiki

Week 1

- This week focused on getting to know PyTorch by *translating* our last M3 Keras model
- As our group is formed by different M3 groups, we have worked on two models
- Apart from implementing the model itself, we also had to learn how to properly keep track of our performance and log our epochs statistics on Tensorboard
- We have also researched on some useful techniques, like Data Augmentation and Early Stopping

First Model

M3 - Model II



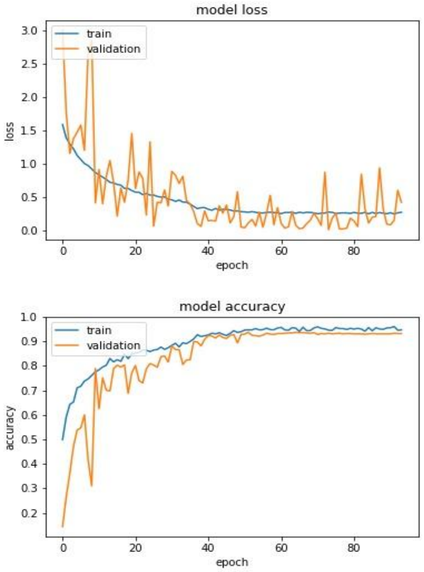
Hyperparameters

Img. Size	256x256
Epochs	100
Batch Size	16
Learning Rate	1e-3

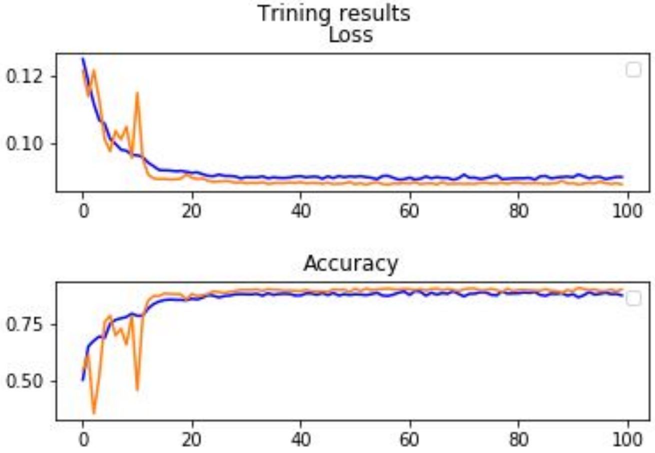
Activation	ReLu
Weight Init	Glorot uniform
Dropout rate	0.2

M3 - Model II

Keras



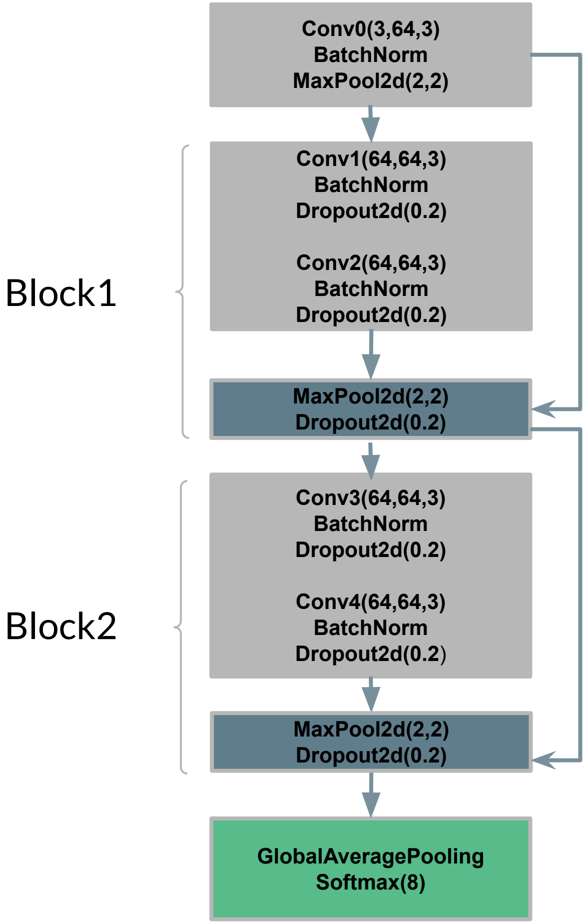
PyTorch



- **Loss** is much **more stable on PyTorch** than on Keras
- Model **performance is preserved**, with very low overfitting in both cases
- For these 100 epochs, **PyTorch had a smaller running time**

Second Model

M3 - Model II



Hyperparameters

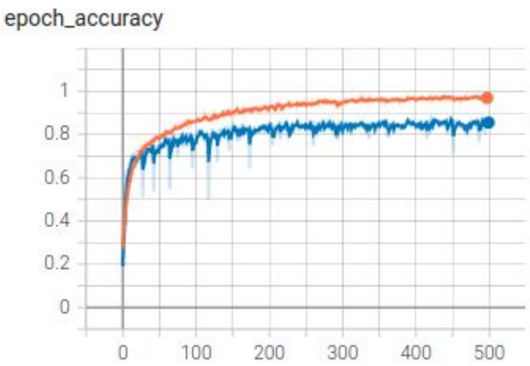
Img. Size	64x64
Epochs	500
Batch Size	32
Optimizer	Adam*
Learning Rate	1e-3

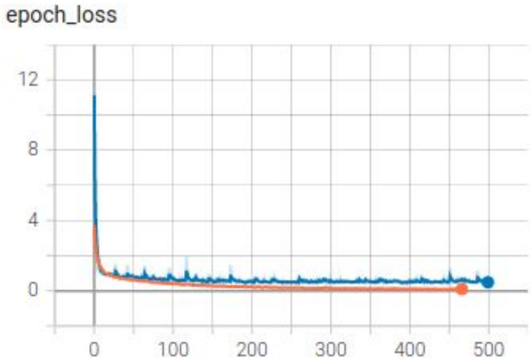
*Adam configuration
- *lr=0.001*
- *betas=(0.9, 0.999)*
- *eps=1e-07*
- *weight_decay=0*
- *amsgrad=False*)

Activation	ReLu
Weight Init	Glorot uniform
Dropout rate	0.2

M3 - Model II - Results graphs

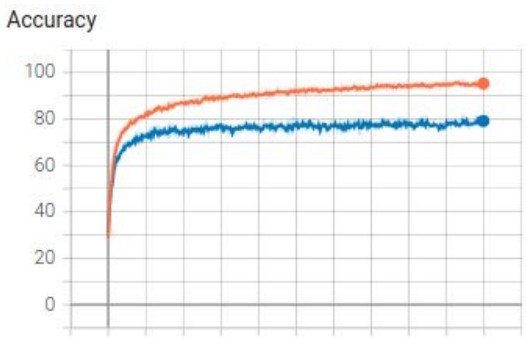
Keras

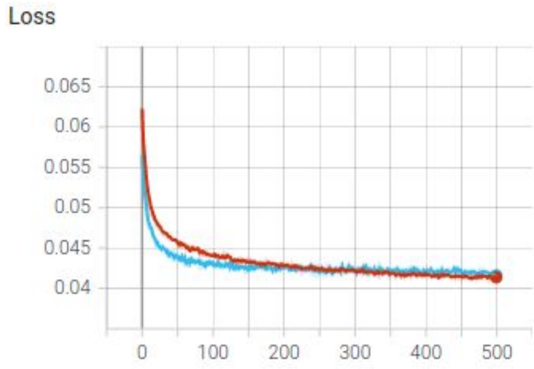




Training
Validation

PyTorch





- Validation **accuracy** is a bit **higher in keras** (85%) than in pytorch (80%).
- **Overfitting** is also slightly **worse in pytorch**
- **Testing accuracy is 88%** in both cases, so we might have a mistake on the validation accuracy calculation
  - As opposed to testing, the validation set is obtained as a random split of the training set. So even specifying shuffle = False, data changes between experiments, unabling repeatability.
- **Accuracy** seems a bit **more stable in pytorch** and the **loss** decreases on a **nicer** manner (keras drastically drops at the beginning and then plateaus, while PyTorch follows a more reasonable pattern)

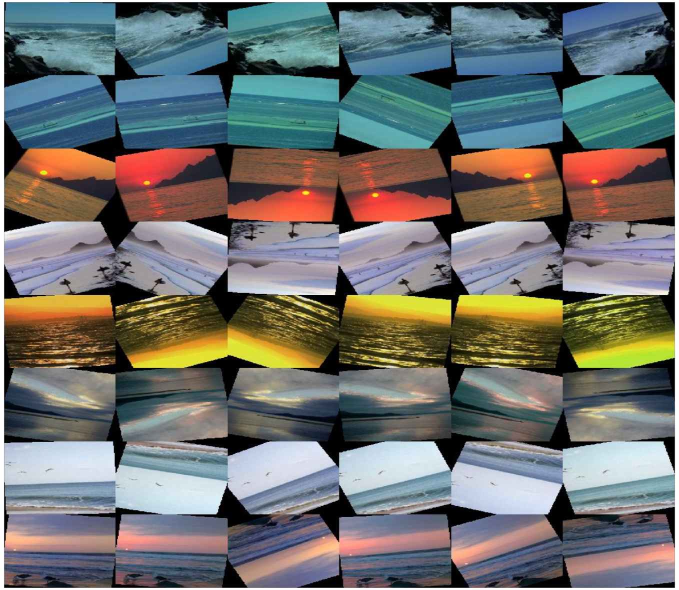
M3 Model II - Comparison II

	Trainable Parameters	Time	Train accuracy	Val accuracy	Test accuracy
Keras	150,664	1805”	98%	85%	88%
PyTorch	150,664	1701”	95%	80%	88%

- We have exactly the same number of trainable parameters in both libraries, which is a good sign
- PyTorch is around 100” faster on these 500 epochs, which is a negligible improvement. Nevertheless, this might become more significant when training bigger models for a larger number of epochs.
- Keras training accuracy is a 3% higher than PyTorch accuracy. This difference is even higher in validation (5%). However, test accuracy is preserved.

Additional investigations

Data Augmentation I



- We additionally investigated on how to perform data augmentation in PyTorch, as it can be a valuable tool in the future

```
transforms = torchvision.transforms.Compose([
    torchvision.transforms.Resize((224,224)),
    torchvision.transforms.ColorJitter(brightness=
0.5,contrast=0.6,hue=.05, saturation=.05),
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.RandomVerticalFlip(),
    torchvision.transforms.RandomCrop(224),
    torchvision.transforms.RandomRotation(30,
resample=PIL.Image.BILINEAR)
])
```

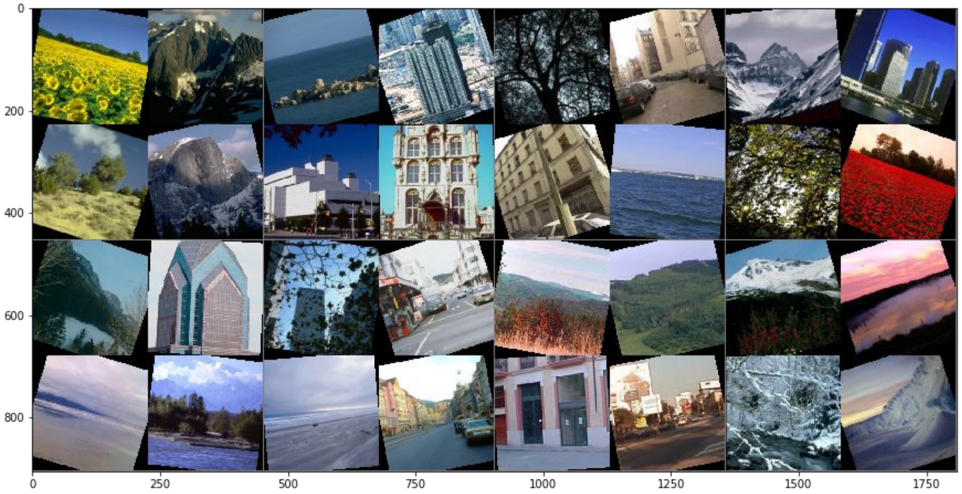
```
dataset = torchvision.datasets.ImageFolder(images_folder,
      transform=transforms)
```

Data Augmentation II

Probe architecture for data augmentation:

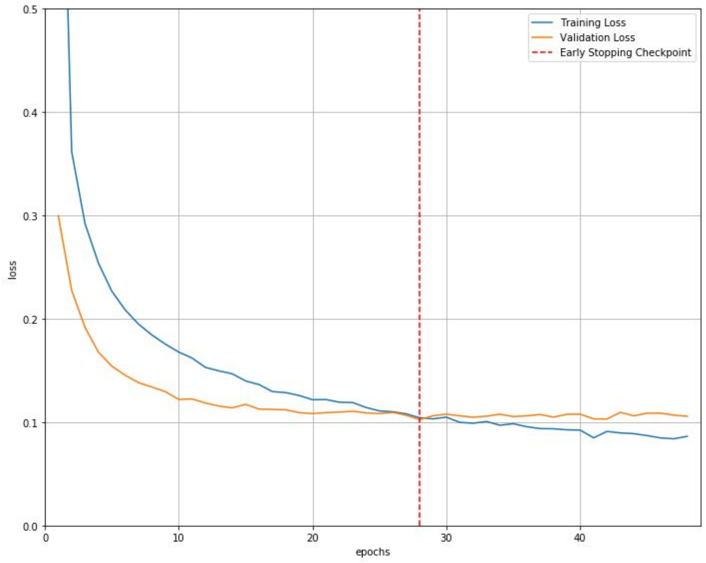
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        #x = x.view(x.size(0), 16*224*224)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```




```
Accuracy of the network on the 1881 test images: 71 %
epoch = 100
```

Early stopping



- Early stopping is a form of regularization used to avoid overfitting on the training dataset. It keeps track of the validation loss, if the loss stops decreasing for several epochs in a row the training stops.
- While not implemented by default on PyTorch, there are many community implementations such as this one.
- We set the patience argument in the EarlyStopping class to how many epochs we want to wait after the last time the validation loss improved before breaking the training loop

Docker image

- Though not directly related to the project, we have taken this opportunity to investigate on how to use Docker in order to have a more versatile application
- This was more relevant when using Keras, as we had some trouble in the past with finding the best configuration of Cuda + Cudnn + Tensorflow + Keras
- Torch seems to work on GPU more easily, but we still found learning about Docker useful

[Summary]: Keras vs Tensorflow vs Pytorch

- Our M3 model was translated to PyTorch quite straightforwardly
- PyTorch is slightly less user-friendly than Keras, but seems much more versatile and customizable
- Debugging is also probably easier on PyTorch
- GPU implementation was **significantly** easier with PyTorch than with Keras
- Pytorch's documentation is as good or better than Keras, and it seems to have a very active community.

Keras is most suitable for:

- ☑ Rapid Prototyping
- ☑ Small Dataset
- Multiple back-end support

TensorFlow is most suitable for:

- Large Dataset
- High Performance
- Functionality
- Object Detection

PyTorch is most suitable for:

- ☑ Flexibility
- ☑ Short Training Duration
- Debugging capabilities

*https://www.edureka.co/blog/keras-vs-tensorflow-vs-pytorch/

Thank You