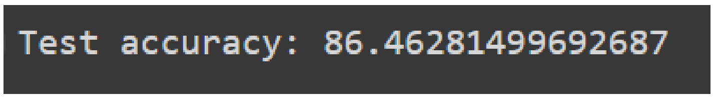
Laia Tomàs 252294
Manvir Kaur 253852
Quim Ribas 251754

REPORT LAB3

EXERCISE 1

In this first exercise we are asked to obtain the maximum test accuracy possible in the SVHN dataset. For this purpose, we are asked to explore some different strategies.
Firstly, here are the results with just the model that was provided in the examples.



```
Test accuracy: 86.46281499692687
```

Figure 1

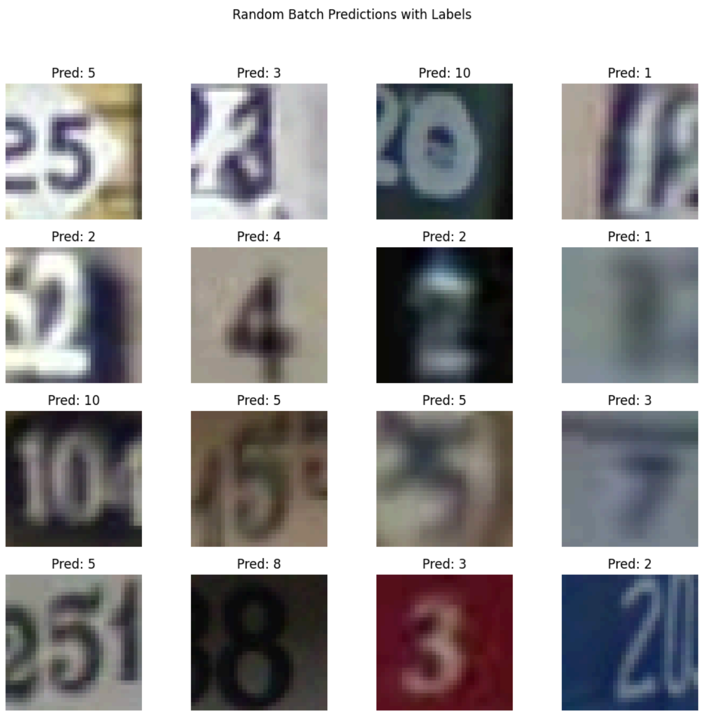
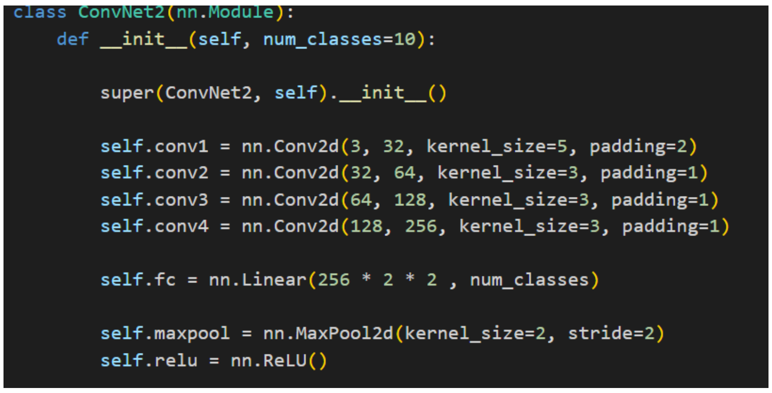


Figure 2

Now, exploring the strategy of adding more layers, we have added two more convolutional layers as seen in Figure 3. For this, we have kept the Adam optimizer.

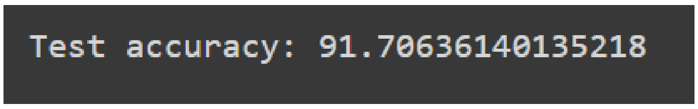


```
class ConvNet2(nn.Module):
    def __init__(self, num_classes=10):

        super(ConvNet2, self).__init__()

        self.conv1 = nn.Conv2d(3, 32, kernel_size=5, padding=2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, padding=1)

        self.fc = nn.Linear(256 * 2 * 2 , num_classes)

        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.relu = nn.ReLU()
```

Figure 3

Laia Tomàs 252294
Manvir Kaur 253852
Quim Ribas 251754

The results are the following:



```
Test accuracy: 91.70636140135218
```

Figure 4

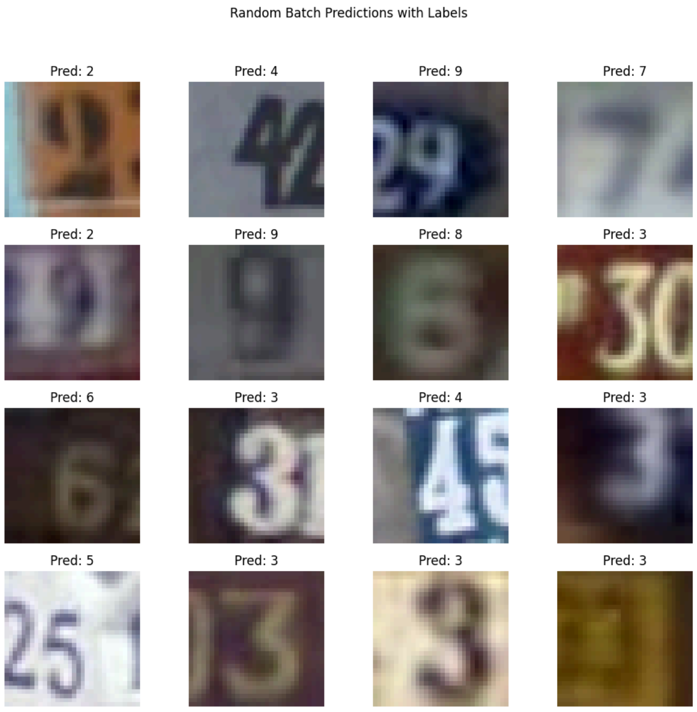
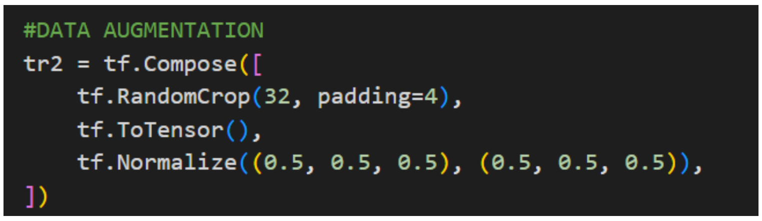


Figure 5

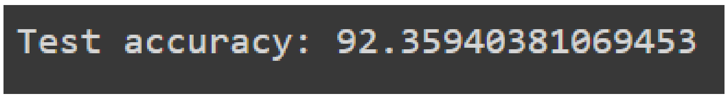
Next step, we followed the strategy of data augmentation. For this, we used random cropping with padding in the test loader creation as you can see in Figure 6.



```
#DATA AUGMENTATION
tr2 = tf.Compose([
    tf.RandomCrop(32, padding=4),
    tf.ToTensor(),
    tf.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])
```

Figure 6

These are the corresponding results:



```
Test accuracy: 92.35940381069453
```

Figure 7

Laia Tomàs 252294
Manvir Kaur 253852
Quim Ribas 251754

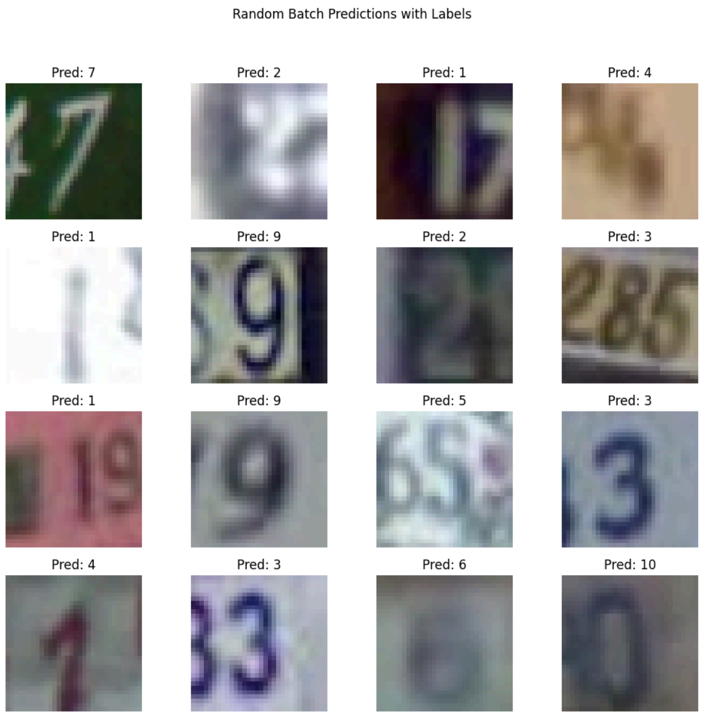
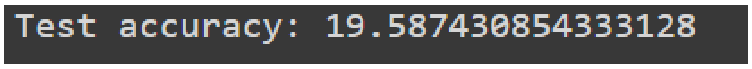


Figure 8

Following, we tried changing the optimizer to SGD with momentum = 0.9, along with the data augmentation process. However, we got really bad results:



```
Test accuracy: 19.587430854333128
```

Figure 9

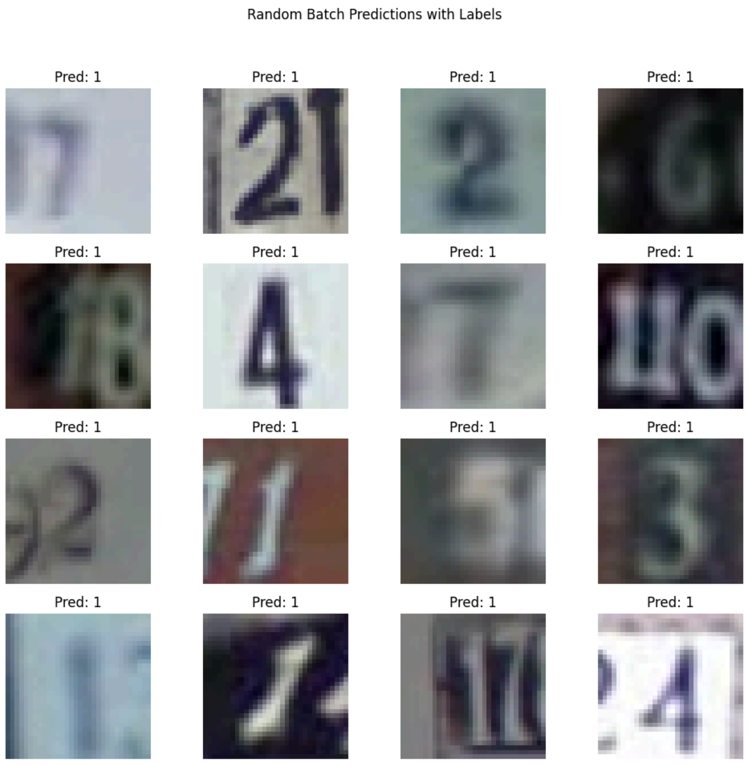
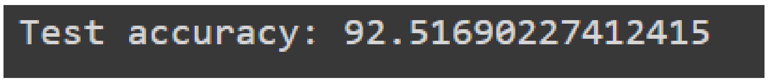


Figure 10

Laia Tomàs 252294
Manvir Kaur 253852
Quim Ribas 251754

Next, we tried a combination of the ConvNet with 5 convolutional layers and the data augmentation process. To this, we obtained really good results:



```
Test accuracy: 92.51690227412415
```

Figure 11

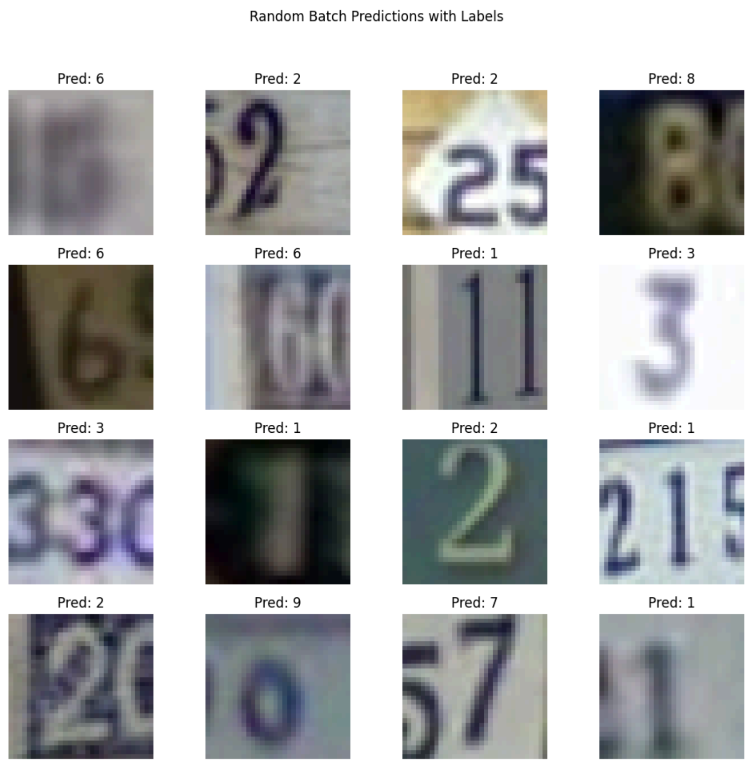
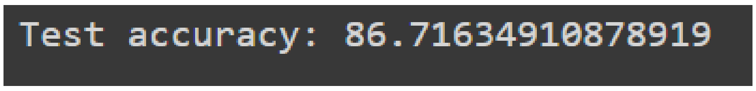


Figure 12

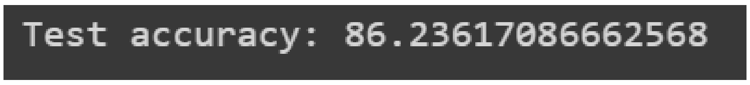
After this, we tried a variation of a ConvNet with 2 convolutional layers and 2 fully connected layers, and even though we didn't get bad results, they could have been better.



```
Test accuracy: 86.71634910878919
```

Figure 13

Finally, we tried to only change the optimizer to the model given in the examples. These were the results:



```
Test accuracy: 86.23617086662568
```

Figure 14

Laia Tomàs 252294
Manvir Kaur 253852
Quim Ribas 251754

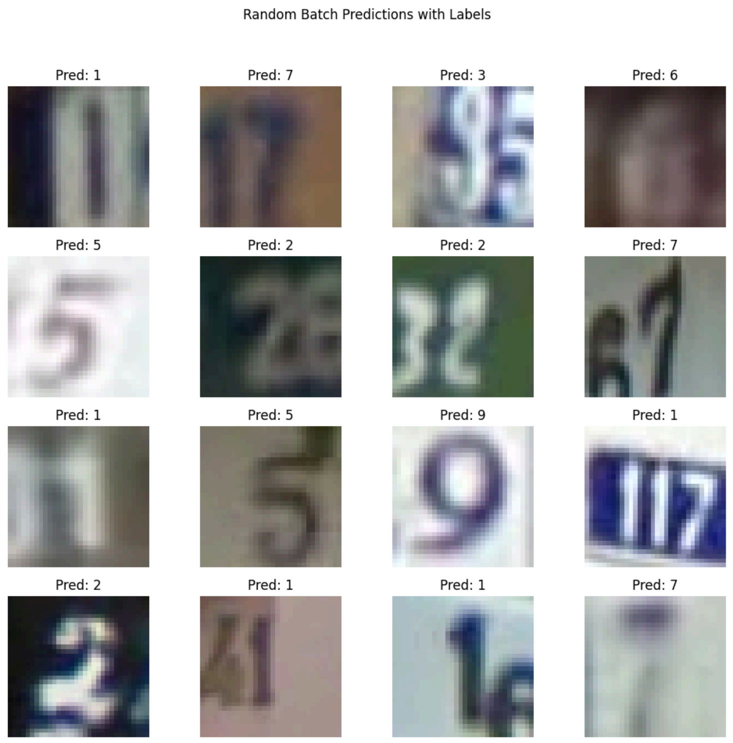


Figure 15

To conclude this first exercise, we can state that the best strategy to achieve the greatest accuracy was with the ConvNet of 5 convolutional layers and the data augmentation process, together with the Adam optimizer. Using more convolutional layers allows the network to learn more complex features. Early layers might learn basic edges and textures, while deeper layers can learn more abstract concepts like shapes and object parts. About the data augmentation process, it helps the model generalize better to new, unseen data by providing more varied training examples. Generally, this prevents overfitting. Finally, the Adam optimizer adjusts the learning rate for each parameter individually based on estimates of the first moment (mean) and second moment (uncentered variance) of the gradients.

Laia Tomàs 252294
Manvir Kaur 253852
Quim Ribas 251754

EXERCISE 2:
In this exercise we were asked to build our own CNN architecture to predict digital numbers on the SVHN dataset. The goal was to find the maximum possible accuracy, with the constraint that the number of parameters should be less than 150k. It wasn't allowed to change training hyper-parameters such as learning rate, batch size or number of epochs. We could only modify the architecture definition.

To improve the model's accuracy and performance, we employed several techniques and architectural choices, each with its rationale and observed benefits. Below is a comprehensive discussion of the approaches we tried, why we chose them, and which ones proved to be more beneficial.

Convolutional Layers

Approach: We introduced multiple convolutional layers to extract features from the input image.
Rationale: Convolutional layers are fundamental for capturing spatial hierarchies and patterns in image data.
Benefit: The model learned low-level features (like edges) in early layers and more complex features (like shapes and objects) in deeper layers.

Batch Normalization

Approach: Applied batch normalization after each convolutional layer.
Rationale: Batch normalization stabilizes and accelerates the training process by normalizing the input to each layer, ensuring a more stable distribution of activations throughout the network.
Benefit: Faster convergence during training and improved performance, as it helps mitigate issues like vanishing/exploding gradients.

ReLU Activation Function

Approach: Used ReLU activation after each batch normalization layer.
Rationale: ReLU introduces non-linearity into the model, which is essential for learning complex patterns.
Benefit: Efficient and effective in training deep networks by reducing the likelihood of vanishing gradients.

Max Pooling

Approach: Placed max pooling layers after certain convolutional blocks.
Rationale: Max pooling reduces the spatial dimensions of the feature maps, which reduces the computational load and helps in achieving spatial invariance.
Benefit: Improved model efficiency and robustness to spatial variations in the input data.

Depthwise Separable Convolution

Approach: Replaced some standard convolutional layers with depthwise separable convolutions.

Rationale: This technique significantly reduces the number of parameters and computational cost without compromising on performance.
Benefit: The model became lighter and faster, making it suitable for deployment on devices with limited computational resources.

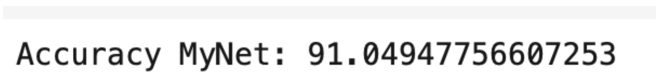
Residual Connections

Approach: Implemented residual connections in deeper layers of the network.
Rationale: Residual connections help in mitigating the vanishing gradient problem by allowing gradients to flow through the network more effectively.
Benefit: Enabled the training of deeper networks by improving gradient flow, which resulted in better performance.

Fully Connected Layer and Dropout

Approach: Used a fully connected layer with dropout before the final output layer.
Rationale: The fully connected layer maps the learned features to the output classes. Dropout prevents overfitting by randomly dropping units during training.
Benefit: Enhanced generalization of the model, leading to better performance on unseen data.

Overall, the combination of these techniques and architectural choices has shown to be highly beneficial in improving the model's accuracy and performance as we obtained an accuracy of 91% as we can see in figure 16.



```
Accuracy MyNet: 91.04947756607253
```

Figure 16

Laia Tomàs 252294
Manvir Kaur 253852
Quim Ribas 251754

EXERCISE 3:

For the solution of this exercise we needed to apply the implemented network architecture designed in exercise 2 to solve the transfer learning problem.

As the code given is structured, we first initialize the data for the training step and then we train the model with the big dataset (the one containing numbers from 1 to 8).

Once this pretrained step is completed, now we need to adjust our model to the corresponding task, which is to classify the dataset into 0s and 9s. Under this goal, and differently from the examples, we need to fine-tune not only the last layer but also a larger number of parameters.

For this reason, we are still going to modify the last layer as we need to remove the full connectivity in order to transform this model into a binary classifier. However, we are going to change more parameters. We took the given code for changing only the last layer and modified accordingly to adjust more parameters from other layers.

We computed three cases, with their respective accuracies:

Case	Layer modified	Accuracy obtained
1	Last 2	85.085
2	Last 3	88.986
3	All	92.080

Table 1. Table with each case of parameter fine-tuning

Moreover, in Figure 17 we can see the predicted labels: 1 corresponding to number 9, and label 2 corresponding to number 0.

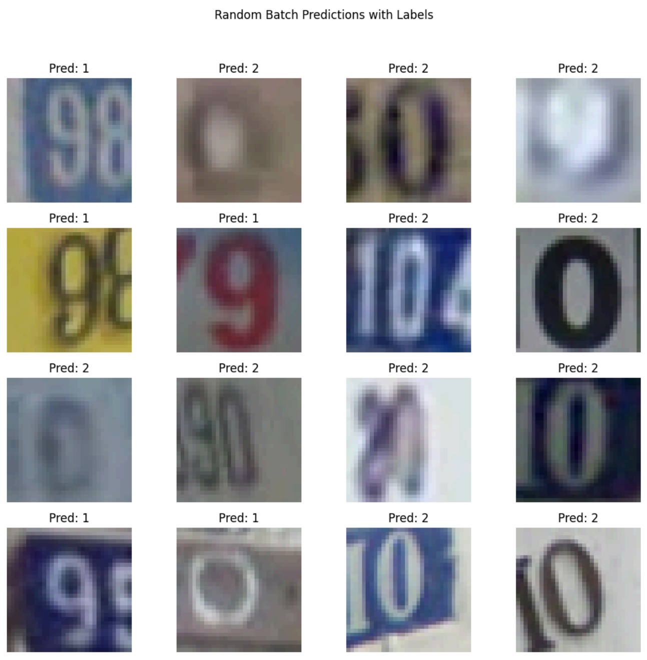


Figure 17

Laia Tomàs 252294
Manvir Kaur 253852
Quim Ribas 251754

By analyzing the results shown in Table 1, we can observe how higher accuracy is related directly to the number of fine-tuned layers. The more parameters modified, the larger accuracy obtained. Initially, limited performance suggests the model is struggling to adapt only the last layer to the new task. However, by unfreezing and fine-tuning additional layers, the model gains access to a wider range of task-specific features from the pre-trained weights. This depth of adaptation across multiple layers enhances the model's ability to discern patterns in the data.