

# NATIONAL CHIAO TUNG UNIVERSITY

**Institute of Computer Science and Engineering**


# Deep Learning

## Homework 2


Author: Alessandro Saviolo

Student Number: 0845086


**3rd December 2019**

# Summary

# 1. MNIST

## 1.1 Model

The CNN architecture implemented for the task of image recognition using the MNIST dataset [11] is presented in Table 1.1. The first two layers of the model apply the convolutional operation to the input image (by sliding over the image first vertically and then horizontally). Note that while the size of the filter is equal to 3, which is a standard choice, the stride size is set to 2. This choice is based on two considerations. First, choosing a stride equal to 2 let us downsample the image smartly, making the architecture more expressive than the equal one using pooling layers to downsample [1]. In particular, by using a convolutional layer instead of a pooling layer, the CNN is capable to learn certain properties that are lost by the latter. This is due to the fact that pooling is a fixed operation, while convolution can be learned. The down side is that this choise also increases the number of trainable parameters (the pooling layer has no parameters to learn). Second, due to the complexity of the dataset, even a simple architecture can obtain an high accuracy on MNIST (this is the main reason to choose a 3-layer architecture). The complexity of the dataset is also the main reason for using the first consideration.

The choice of fixing the size of the kernel to be small is to take advantage of two benefits: weight sharing and reduction in computational costs. Since we use the same kernel for different set of pixels in an image, the same weights are shared across these pixel sets as we convolve on them. And, as the number of weights are less than a fully connected layer, we have lesser weights to backpropagate on.

Each convolutional layer is followed by Leaky ReLU activation function, using *alpha* equal to 0.01. Leaky ReLU is preferred to standard ReLU to deal with the "Dead ReLU problem" [6]. ReLU units can be fragile during training and can "die". For example, a large gradient flowing through a ReLU neuron can cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. That is, the ReLU units can irreversibly die during training since they can get knocked off the data manifold.

| Layer | Input shape | Output shape | Stride | Size | Activation |
|---|---|---|---|---|---|
| Convolutional | (1, 28, 28) | (8, 13, 13) | 2 | 3 | Leaky ReLU |
| Convolutional | (8, 13, 13) | (8, 6, 6) | 2 | 3 | Leaky ReLU |
| Dense | (8, 6, 6) | (10,) | - | - | - |

Table 1.1: Convolutional Neural Network architecture for MNIST dataset.

The hyper parameters tuned to train the aforementioned CNN model are listed in Table 1.2. The number of iterations is equal to the training set size, since both the number of epochs and the batch size used are equal to 1.

| Hyper parameter | Value |
|---|---|
| # Iterations | 55000 |
| Learning Rate | 0.01 |
| Optimizer | SGD |
| Lambda | 0 |

Table 1.2: Hyper parameters tuned to train the Convolutional Neural Network. Lambda refers to the regularization term for L2 regularization. Since it is equal to 0, no regularization is applied. By fixing these hyper parameters, the model requires about 113 seconds to perform 1000 iterations on Google Colab CPU [8].

By using the architecture described in Table 1.1 and training it by fixing the hyper parameters listed in Table 1.2, the CNN reaches an accuracy of 86.589% on the training set and an accuracy of 87.182% on the test set (see Appendix A.2 for training outputs). The training accuracy and learning curve produced by the training process are illustrated, resp., in Figure 1.1 and Figure 1.2. The error considered is the cross entropy error.


Figure 1.1: Training accuracy produced by the regularized model while training on MNIST dataset. The error considered is the cross entropy error. The validation accuracy is computed every 1000 iterations of training. At each validation step, the current model forward propagates the entire validation set. Note that without proper regularization, the model tends to overfit the data, making a rapid search among the solutions space and getting stuck in a local minimum.

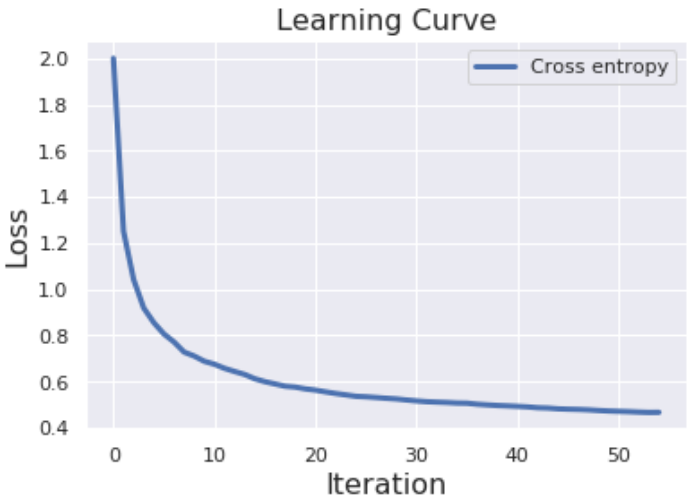



Figure 1.2: Learning curve produced by the model while training on MNIST dataset. The error considered is the cross entropy error.

The weights of the aforementioned trained model are randomly initialized from a univariate Gaussian distribution of mean 0 and variance 1 and divided by 10 (to keep them small). The distribution of the weights after the training process is illustrated in Figure 1.3. We can observe that even after training the weights still represent a Gaussian-like distribution.

In order to control the weight growth, it is possible to use L2 regularization. The usefulness of such technique is proved in Section 1.4.

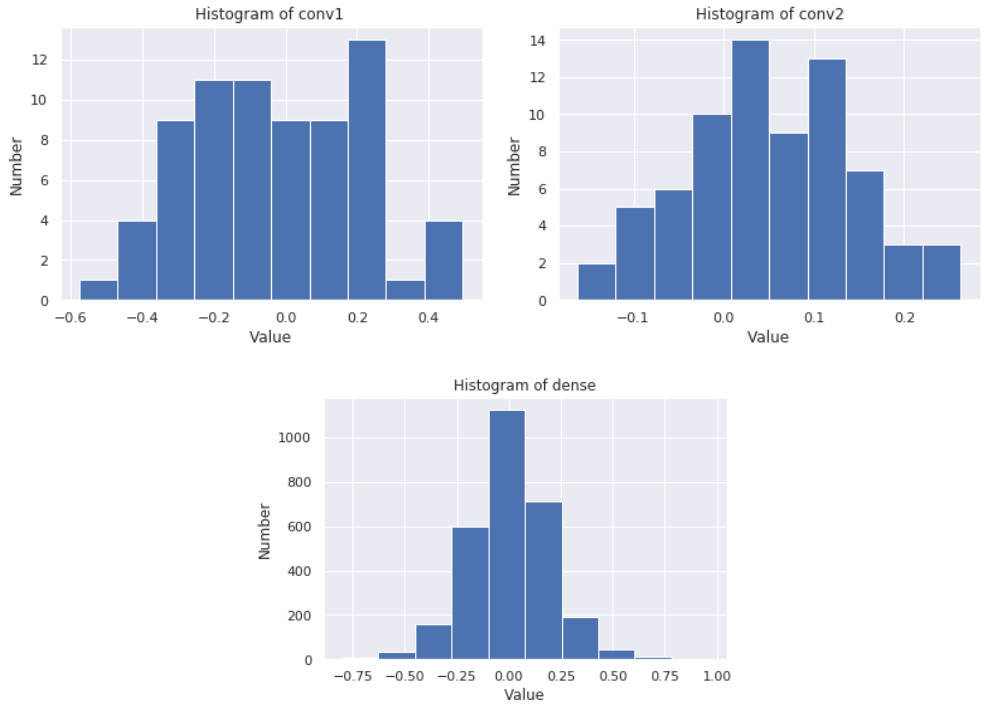



Figure 1.3: Distribution of the weights in the different layers. Note that even after training the weights still represent a Gaussian-like distribution.

## 1.2 Results

After training, the model can be used to recognize digits. Examples of the results produced by such model on the test set are illustrated in Figure 1.4. In some cases, the model's prediction is wrong (see Figure 1.5). This may happen because the model is not complex enough to capture the in-depth features of the training digits (i.e., underfitting). In Section 1.3 such features are shown and discussed. In general, better results can be achieved by adding more convolutional layers to the model. On the other hand, making the model more complex makes the computational time increase substantially (see Appendix A.2 for training outputs).

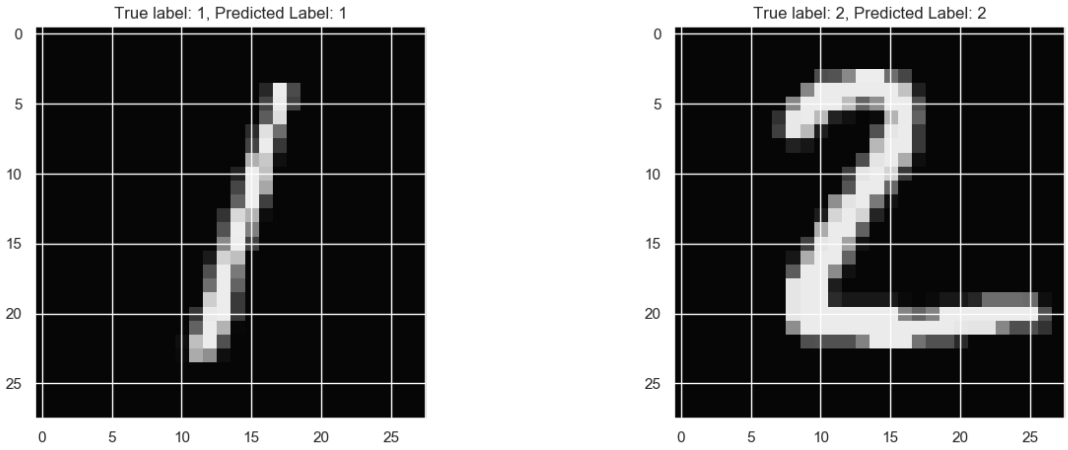



Figure 1.4: Examples of correctly classified digits taken from the test set.

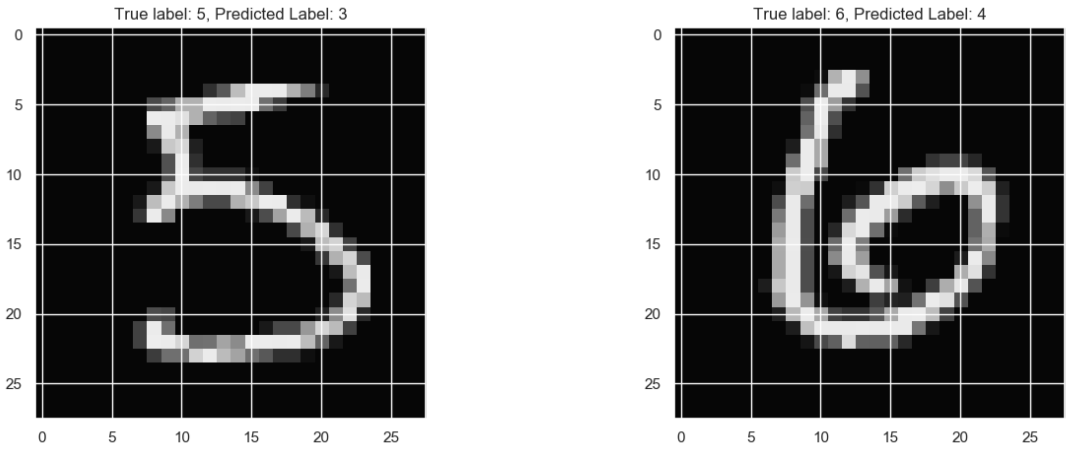
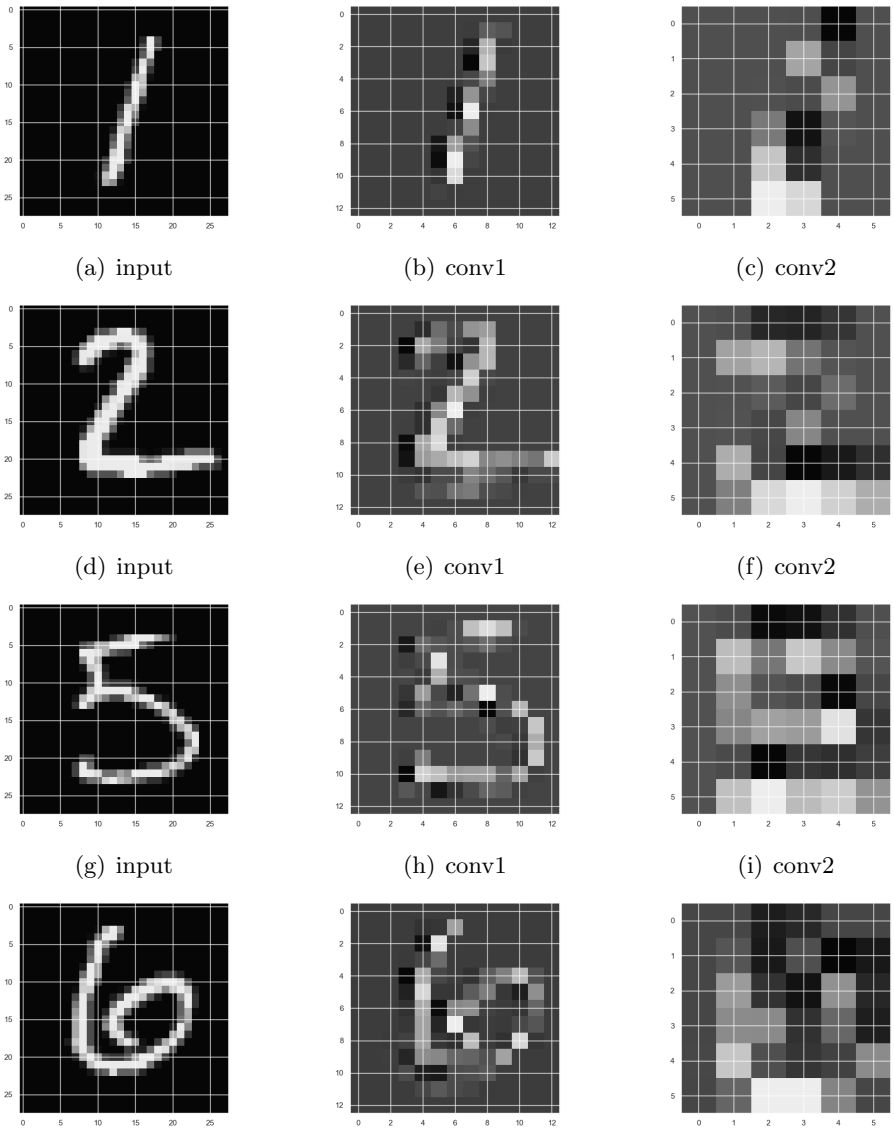



Figure 1.5: Examples of missclassified digits taken from the test set.

## 1.3 Feature Maps

The different feature maps produced by the model across the layers are illustrated in Figure 1.6. The feature maps capture the result of applying the filters to an input image. By visualizing the feature maps we can gain some understanding of what features the model detects. This is important to fine tune the model architecture, since we can observe if some parts of our desired object (i.e., digit) are detected among all the layers or if the activations die out at a certain layer. In general, early layers of the network detect low-level features (e.g., colours, edges) and the deeper layers of the network detect high-level features (e.g., shapes, objects).



(a) input (b) conv1 (c) conv2

(d) input (e) conv1 (f) conv2

(g) input (h) conv1 (i) conv2

(j) input (k) conv1 (l) conv2

Figure 1.6: Feature maps produced by the model in different layers. Digit “1” and digit “2” are correctly classified by the model. To address this fact, in subfigures (c) and (f) we can observe that the model is capable of detecting in-depth features of the corresponding digits. Digit “5” and digit “6” are missclassified by the model. This is due to the low-level features (subfigures (i) and (l)) that the model has failed to learn. Better results can be achieved by increasing the number of filters in the convolutional layers, or by increasing the depth of the model.

## 1.4 Regularization

L2 is one of the most common type of regularization. It updates the general cost function by adding another term known as the regularization term (i.e., *lambda*). Due to the addition of this regularization term, the values of weight matrices decrease because it assumes that a neural network with smaller weight matrices leads to simpler models (i.e., it reduces overfitting to quite an extent).

By applying L2 regularization to the aforementioned model, we can control the weight growth and make the learning process more stable. As a matter of fact, by comparing Figure 2.10, Figure 2.11, Figure 1.1 and Figure 1.2, we can see that the training accuracy curve grows faster in the regularized case. This is due to the regularization term which makes the training process for stable and effective.

By using the architecture described in Table 1.1 and training it by fixing the hyper parameters listed in Table 1.2 with *lambda* equal to 0.01, the regularized model reaches an accuracy of 87.259% on the training set and an accuracy of 89.640% on the test set (see Appendix A.2 for training outputs).

In Figure 1.9 can observe that the weights form a Gaussian-like distribution, where the tails of the distribution are shallow. In particular, by comparing Figure 1.3 and Figure 1.9 we can observe that the regularization helps to clip part of the tail without affecting the final accuracy.


Figure 1.7: Training accuracy produced by the regularized model while training on MNIST dataset. The error considered is the regularized cross entropy error. We can observe that the model is capable of learning the digit recognition task by going through a stable training process. This is due to the L2 regularization. The validation accuracy is computed every 1000 iterations of training. At each validation step, the current model forward propagates the entire validation set.

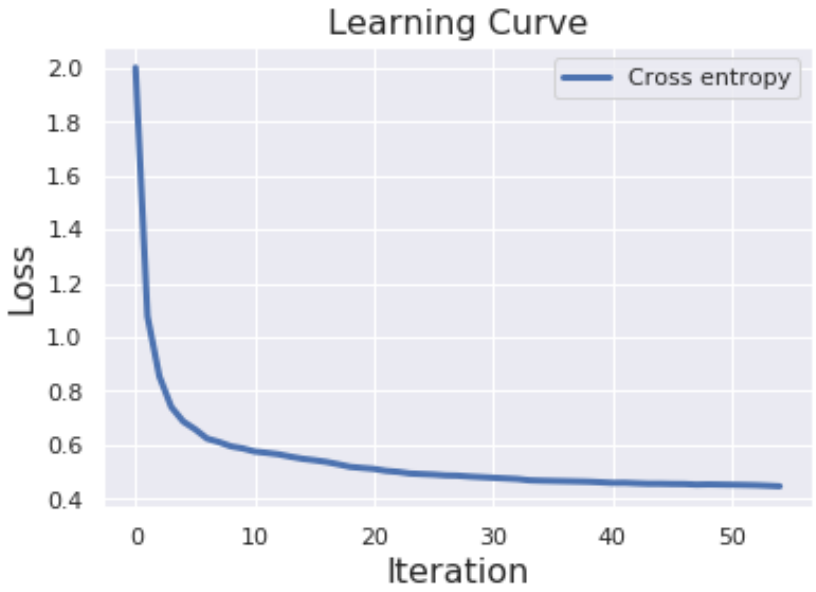



Figure 1.8: Learning curve produced by the regularized model while training on MNIST dataset. The error considered is the regularized cross entropy error.

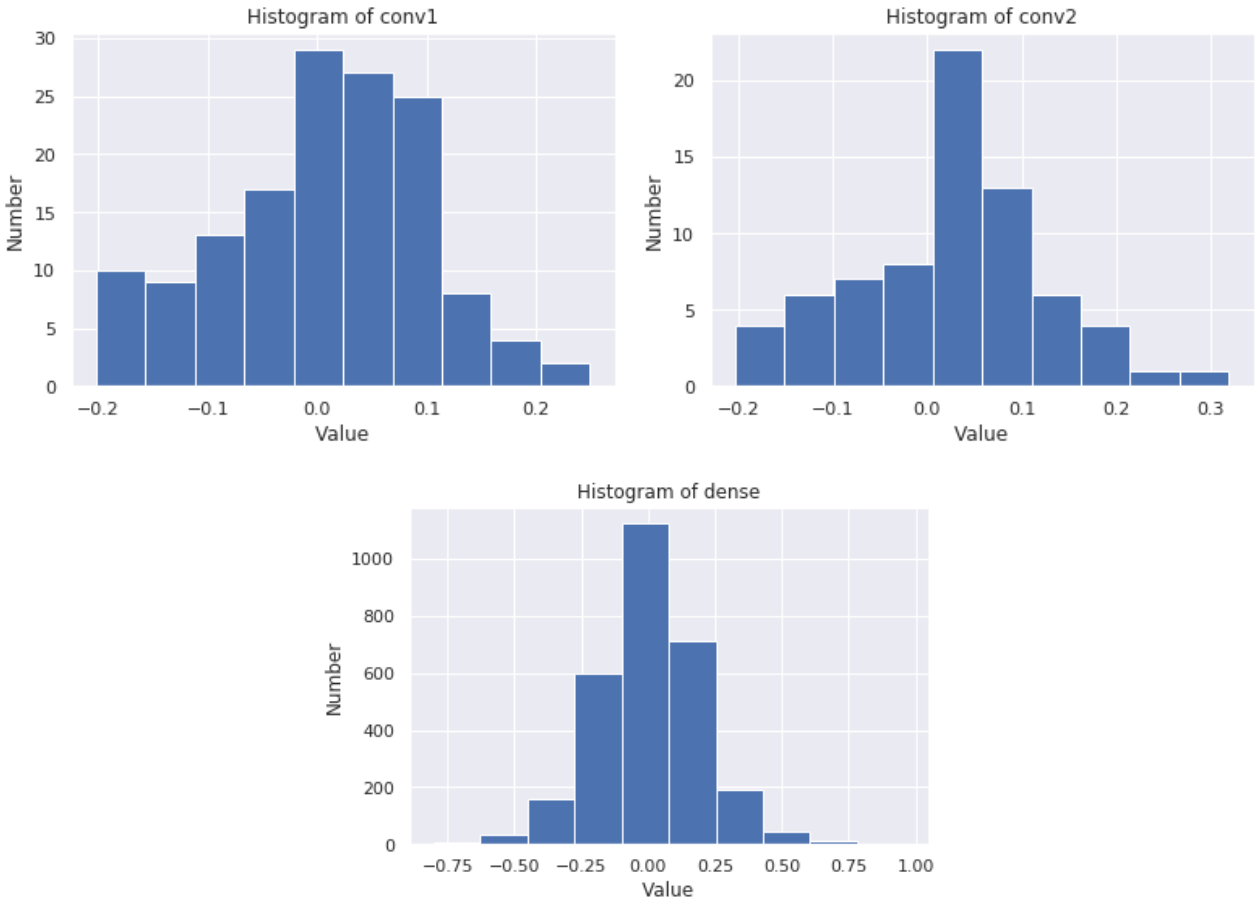



Figure 1.9: Distribution of the weights in the different layers.

# 2. CIFAR10

## 2.1 Model

The model described in Section 1.1 returns very poor results if it is used for the task of image recognition on a more complex dataset, such as CIFAR-10 dataset [10]. As a matter of fact, if we try to train the model on CIFAR-10 dataset, we obtain an accuracy of 28.32% on the test set (see Figure 2.1).

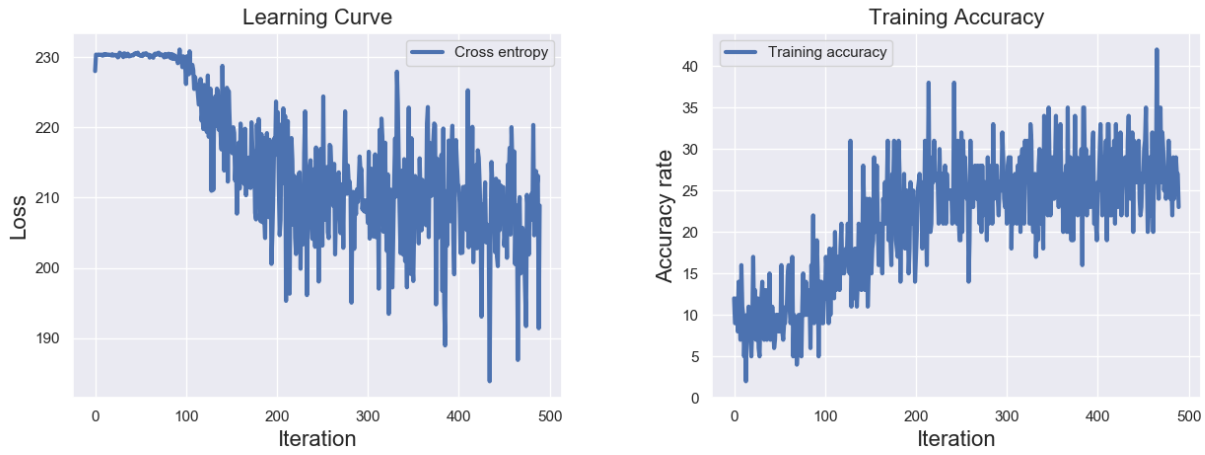



Figure 2.1: Learning curve and training accuracy produced by the model described in Section 1.1 while training on CIFAR-10 dataset. The error considered is the cross entropy error. The model is not capable of learning the task due to its architecture which is too simple.

In order to achieve good results on CIFAR-10 dataset, we need to increase the complexity of the model, by adjusting its depth and by tuning the hyper parameters. The CNN architecture implemented for the task of image recognition using the CIFAR-10 dataset is presented in Table 2.1.

The hyper parameters tuned to train the aforementioned model are listed in Table 2.2. The number of iterations is equal to the training set size, since both the number of epochs and the batch size used are equal to 1. Note that the learning rate has been reduced since the model for CIFAR-10 dataset is more complex (i.e., more parameters to train).

| Layer | Input shape | Output shape | Stride | Size | Activation |
|---|---|---|---|---|---|
| Convolutional | (3, 32, 32) | (32, 30, 30) | 1 | 3 | Leaky ReLU |
| Convolutional | (32, 30, 30) | (32, 28, 28) | 1 | 3 | Leaky ReLU |
| MaxPooling | (32, 28, 28) | (32, 14, 14) | 2 | 2 | - |
| Convolutional | (32, 14, 14) | (64, 12, 12) | 1 | 3 | Leaky ReLU |
| Convolutional | (64, 12, 12) | (64, 10, 10) | 1 | 3 | Leaky ReLU |
| MaxPooling | (64, 10, 10) | (64, 5, 5) | 2 | 2 | - |
| FullyConnected | (64, 5, 5) | (256,) | - | - | Leaky ReLU |
| Dense | (256,) | (10,) | - | - | - |

Table 2.1: Convolutional Neural Network architecture for CIFAR10 dataset.

| Hyper parameter | Value |
|---|---|
| # Iterations | 50000 |
| Learning Rate | 0.005 |
| Optimizer | SGD |
| Lambda | 0 |

Table 2.2: Hyper parameters tuned to train the Convolutional Neural Network. Lambda refers to the regularization term for L2 regularization. Since it is equal to 0, no regularization is applied.

The computational time required to train the model with the architecture described in Table 2.1 and by fixing the hyper parameters listed in Table 2.2 is unfeasible for CPU computing. In particular, the computational time required to train the model for 100 iterations is abound 88 seconds using Google Colab CPU. Since we use a batch size equal to 1 and the training set size is 50000, the time required to train the model for 1 epoch is about 13 hours of CPU computation, which is unfeasible for the scope of this project. To address this fact, I decided to use Keras [9] to implement the exact same model and to train it by using GPU computation.

Switching from CPU to GPU computing makes a huge difference in the computation time required. In particular, using GPU and Keras libraries we are capable to train the model for one epoch (i.e., 50000 iterations) in less then 90 seconds. The same time required for computing just 100 iterations using the CPU!

By using the architecture described in Table 2.1 and training it for 10 epochs with the hyper parameters listed in Table 2.2 (where # Iterations refers to the number of iterations per epoch), the model reaches an accuracy of 55.119% on the test set.

The learning curve and the training accuracy produced by the training process are illustrated in Figure 2.2. The error considered is the cross entropy error. We can observe that the loss and accuracy curves are improving but at the same time they are constantly fluctuating. This behaviour can be reduced by applying regularization techniques, such as L2 regularization (Section 2.4).

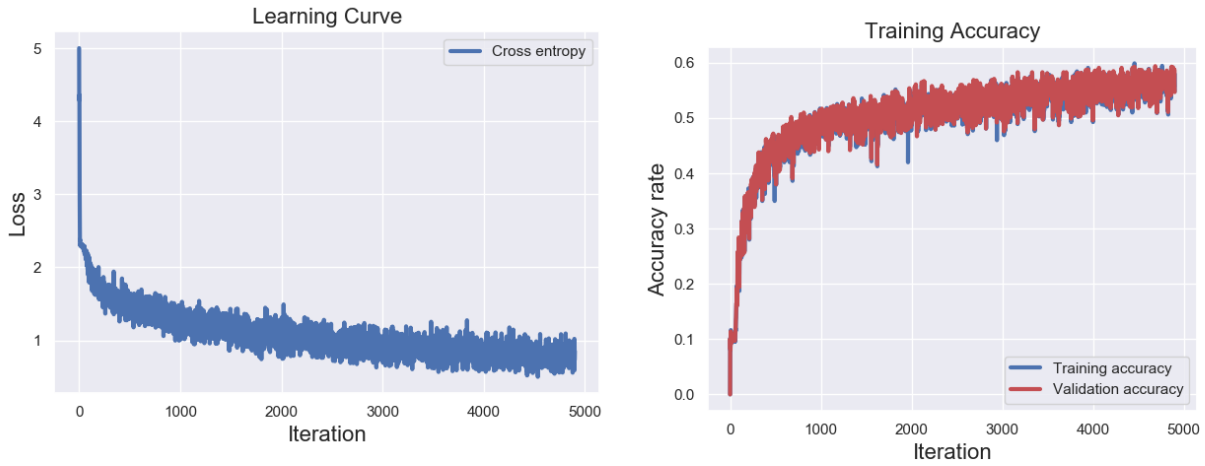



Figure 2.2: Learning curve and training accuracy produced by the model while training on CIFAR-10 dataset. The error considered is the cross entropy error.

The weights of the aforementioned trained model are randomly initialized from a univariate Gaussian distribution of mean 0 and variance 1 and divided by 10 (to keep them small). The distribution of the weights after the training process is illustrated in Figure 2.3. The same observations stated in Chapter 1 apply also to this case.

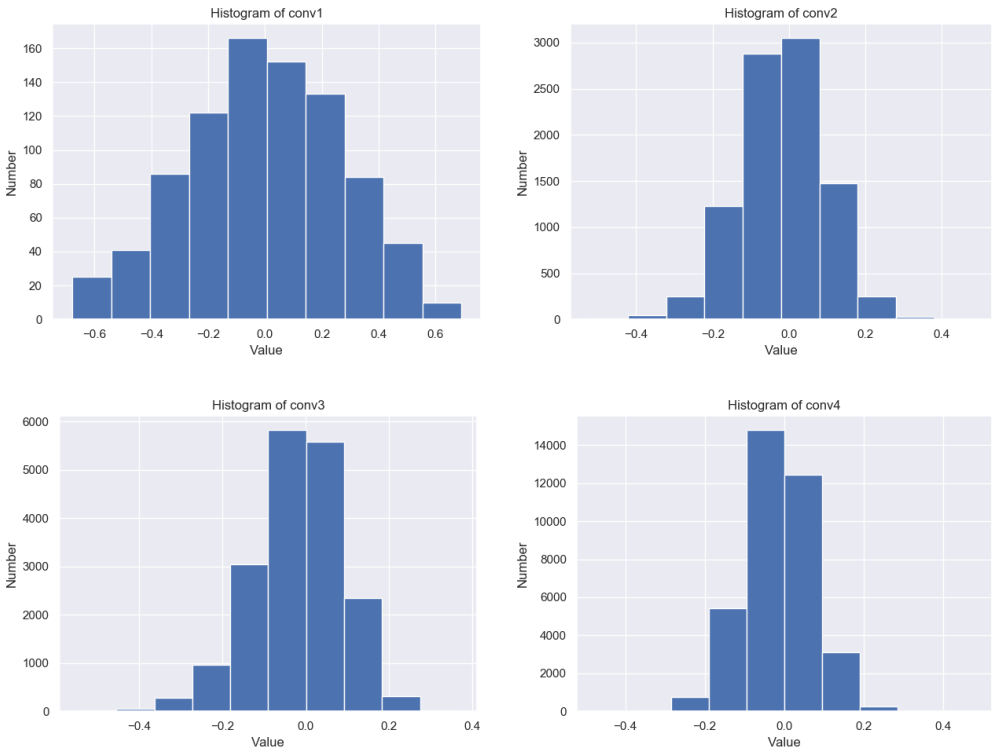



Figure 2.3: Distribution of the weights in the different layers.

## 2.2 Results

After training, the model can be used to recognize CIFAR-10 classes. Examples of the results produced by such model on the test set are illustrated in Figure 2.4. In some cases, the model's prediction is wrong (see Figure 2.5). This may happen because the model is not complex enough to capture the in-depth features (i.e., underfitting). In Section 2.3 such features are shown and discussed. In general, better results can be achieved by adding more convolutional layers to the model. On the other hand, making the model more complex makes the computational time increase substantially.

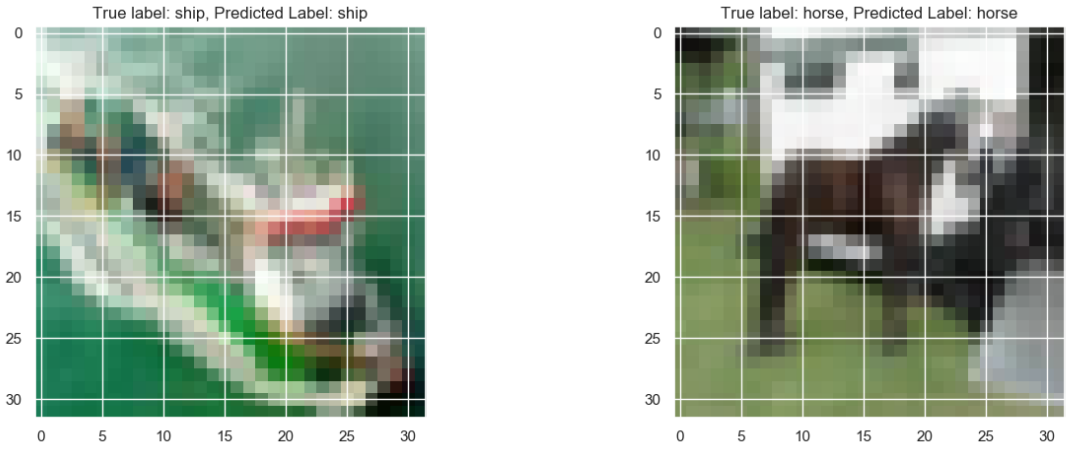



Figure 2.4: Examples of correctly classified images taken from the test set.

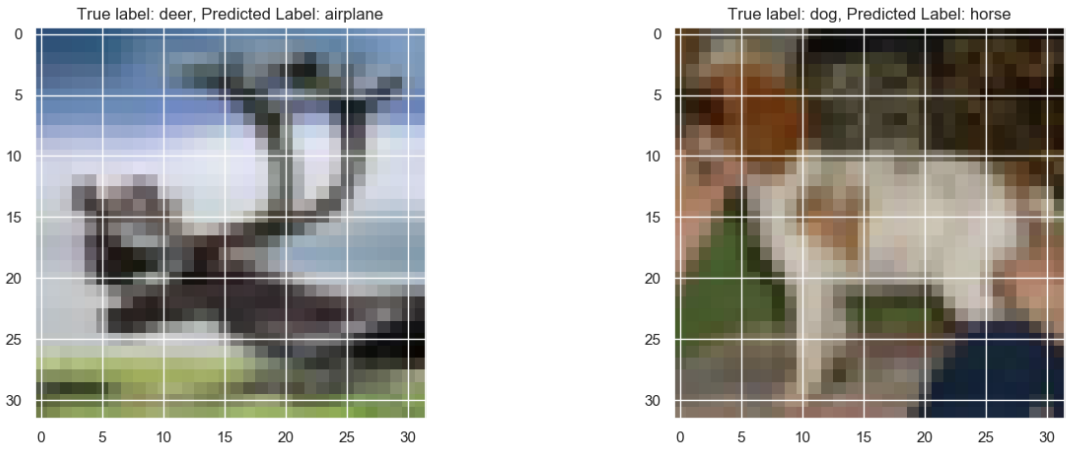



Figure 2.5: Examples of missclassified images taken from the test set.

## 2.3 Feature Maps

The different feature maps produced by the model across the layers are illustrated in Figure 2.6, Figure 2.7, Figure 2.8 and Figure 2.9. The same considerations stated in Section 1.3 still hold in this case.

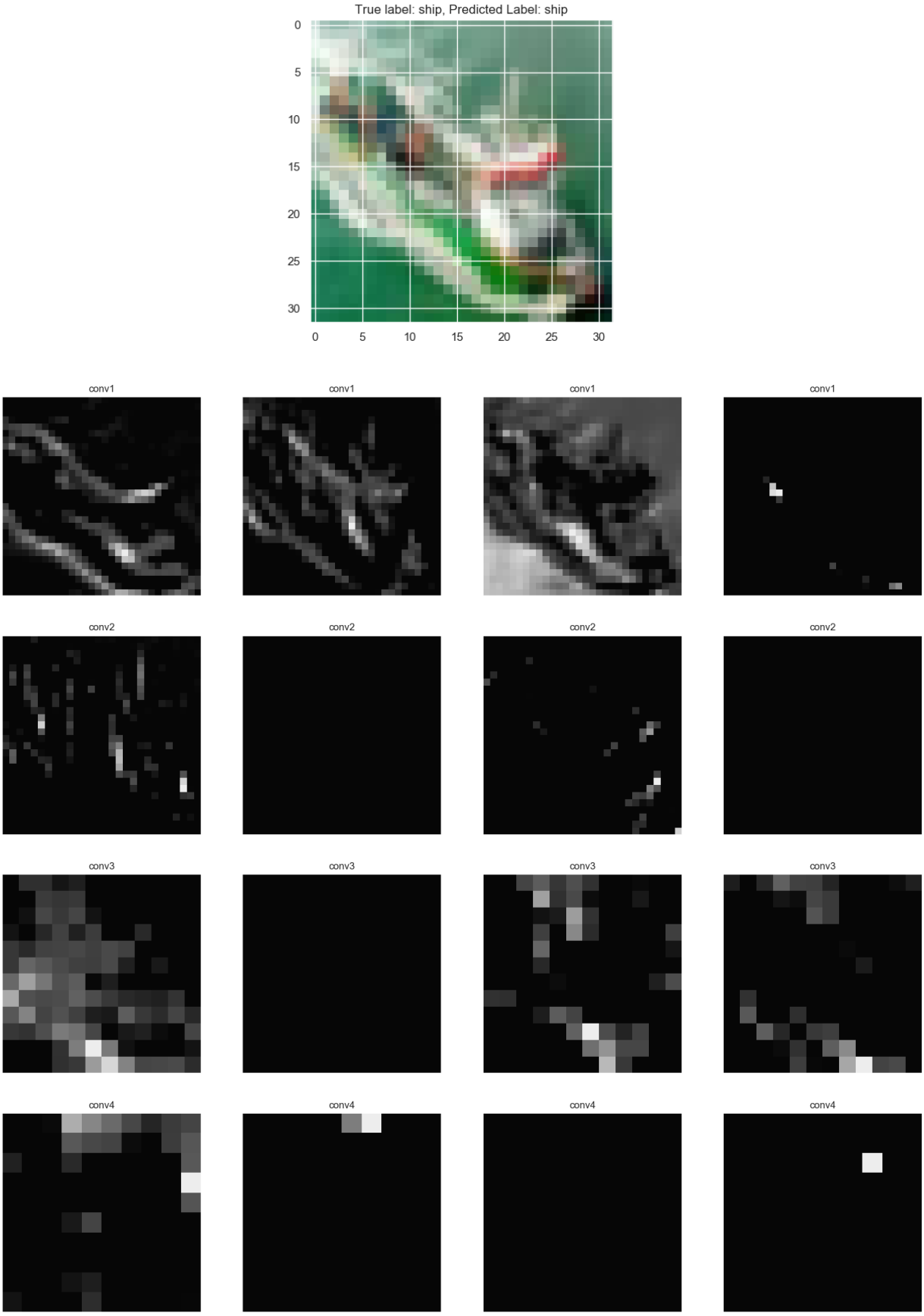



Figure 2.6: Test image and feature maps produced by the model when predicting the image. The model *correctly classifies* the image. Each row represents 4 feature maps produced by a convolutional layer.

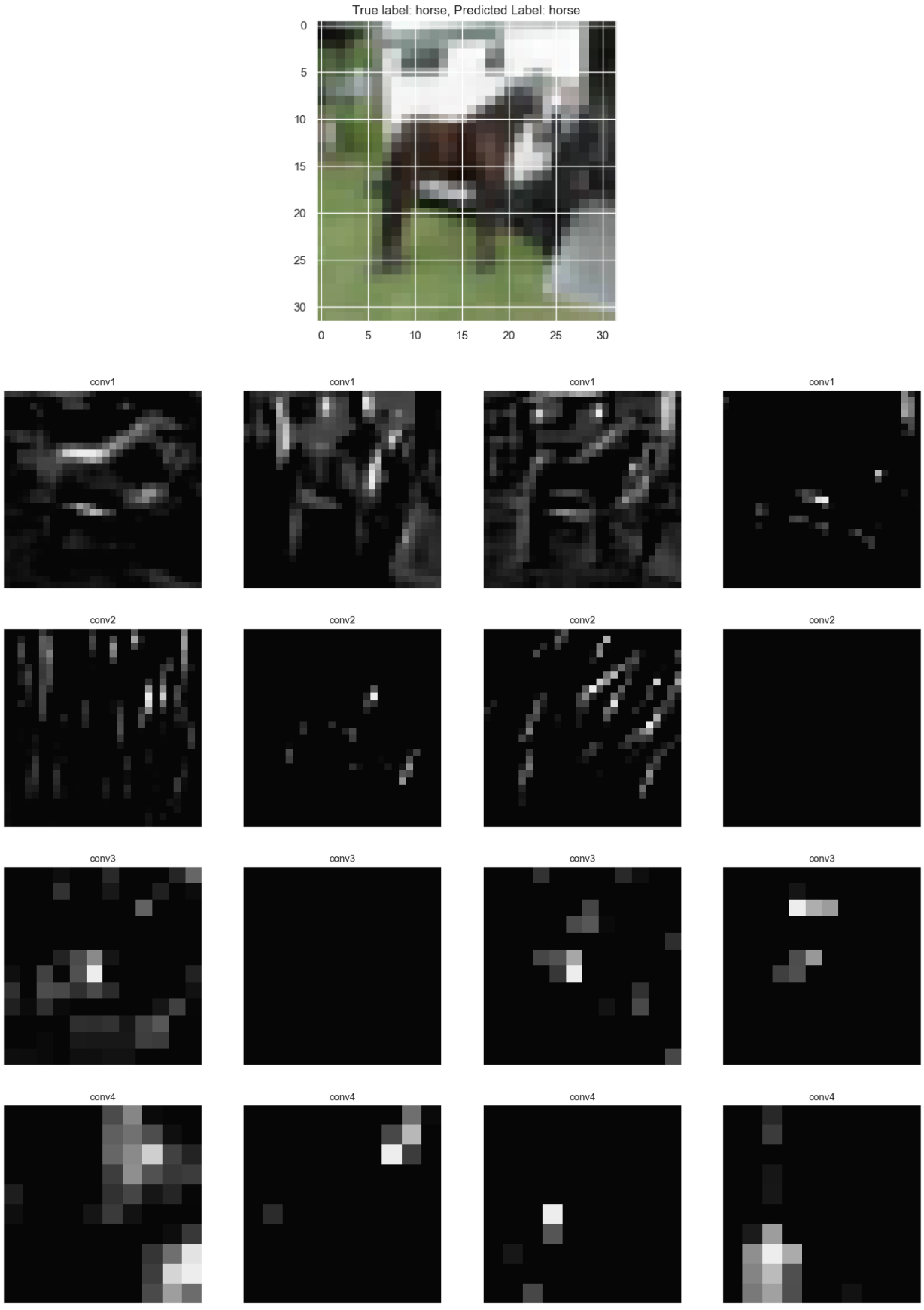



Figure 2.7: Test image and feature maps produced by the model when predicting the image. The model *correctly classifies* the image. Each row represents 4 feature maps produced by a convolutional layer.

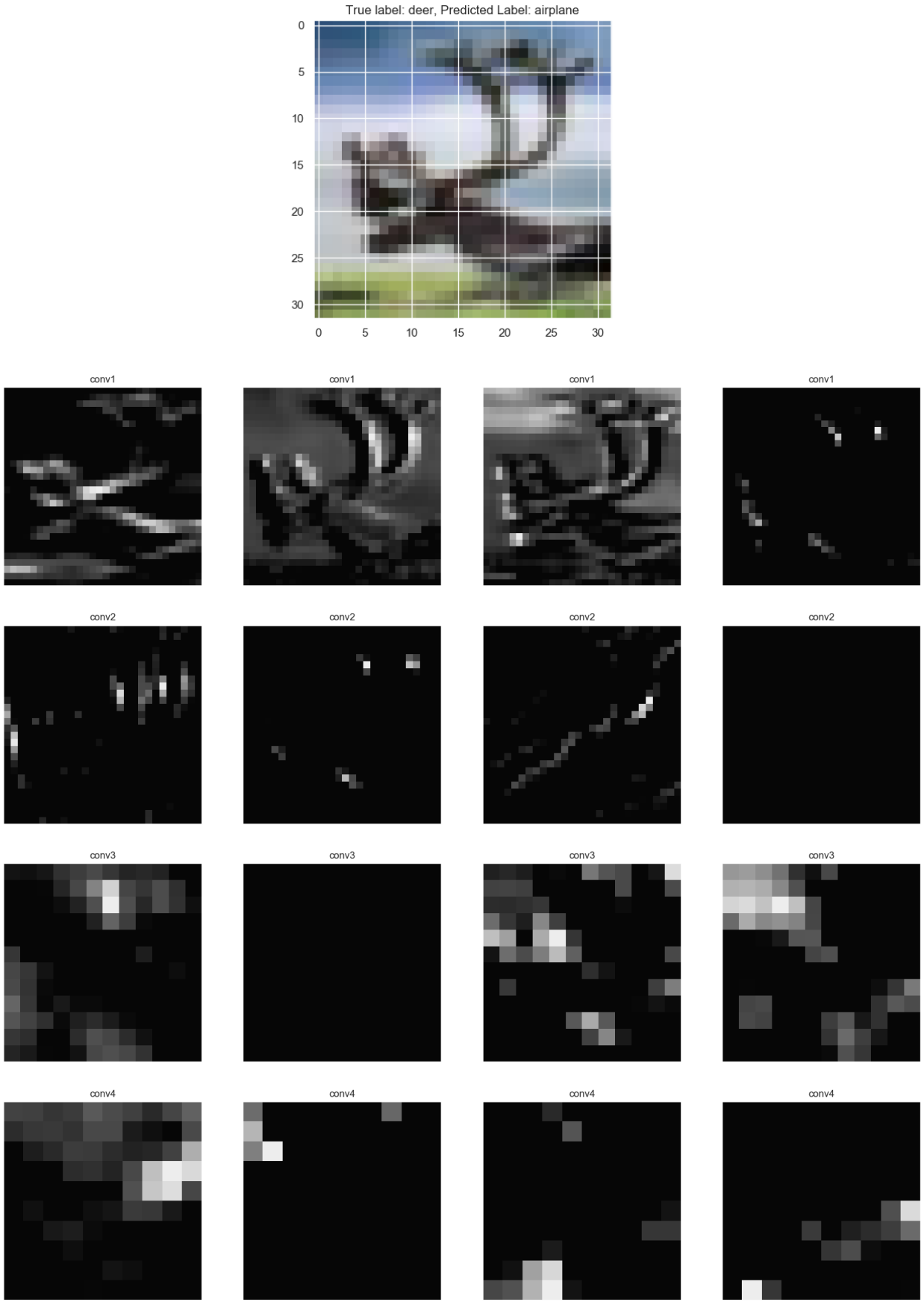



Figure 2.8: Test image and feature maps produced by the model when predicting the image. The model *missclassifies* the image. Each row represents 4 feature maps produced by a convolutional layer.

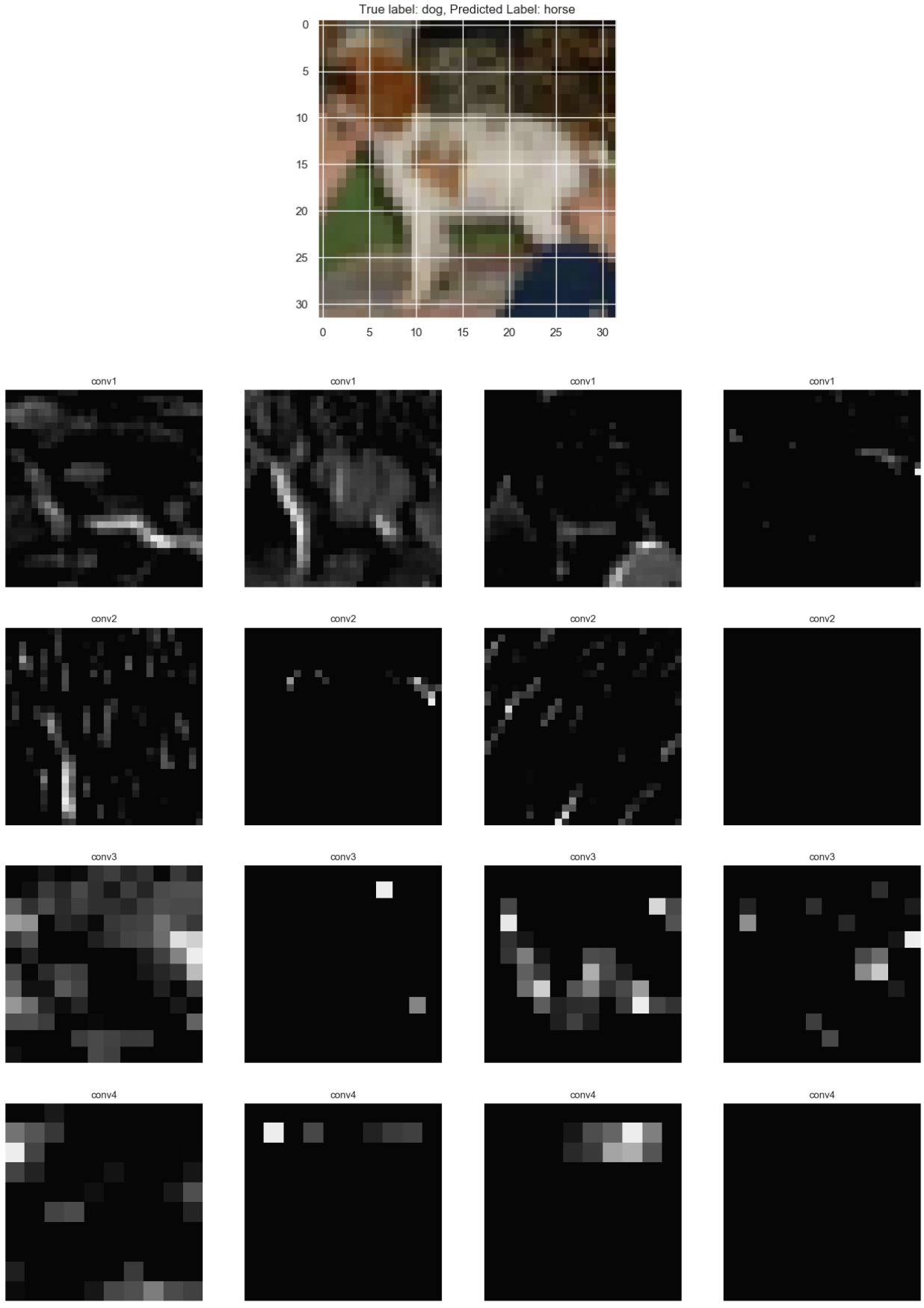



Figure 2.9: Test image and feature maps produced by the model when predicting the image. The model *missclassifies* the image. Each row represents 4 feature maps produced by a convolutional layer.

## 2.4 Regularization

As observed before, by applying L2 regularization to a model, we can control the weight growth and make the learning process more stable.

By using the architecture described in Table 2.1 and training it for 10 epochs with the hyper parameters listed in Table 2.2 and *lambda* equal to 0.01, the regularized model reaches an accuracy of 78.212% on the training set and an accuracy of 75.230% on the test set.

The test accuracy is substantially improved from 55.119% to 75.230% by using the L2 regularization. This demonstrates the importance of regularizing the training process, which makes the learning stable and more accurate.


Figure 2.10: Training accuracy produced by the regularized model while training on CIFAR-10 dataset. The error considered is the regularized cross entropy error. We can observe that the model is capable of learning the image recognition task by going through a stable training process. This is due to the L2 regularization. The validation accuracy is computed every 100 iterations of training. At each validation step, the current model forward propagates the entire validation set.

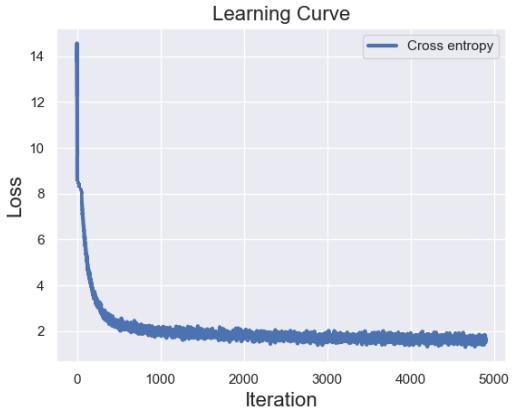



Figure 2.11: Learning curve produced by the regularized model while training on CIFAR-10 dataset. The error considered is the regularized cross entropy error.

In Figure 1.9 we can observe that the weights form a Gaussian-like distribution, where the tails of the distribution are shallow. In particular, by comparing Figure 2.3 and Figure 2.12 we can observe that the regularization helps to clip part of the tail without affecting the final accuracy.

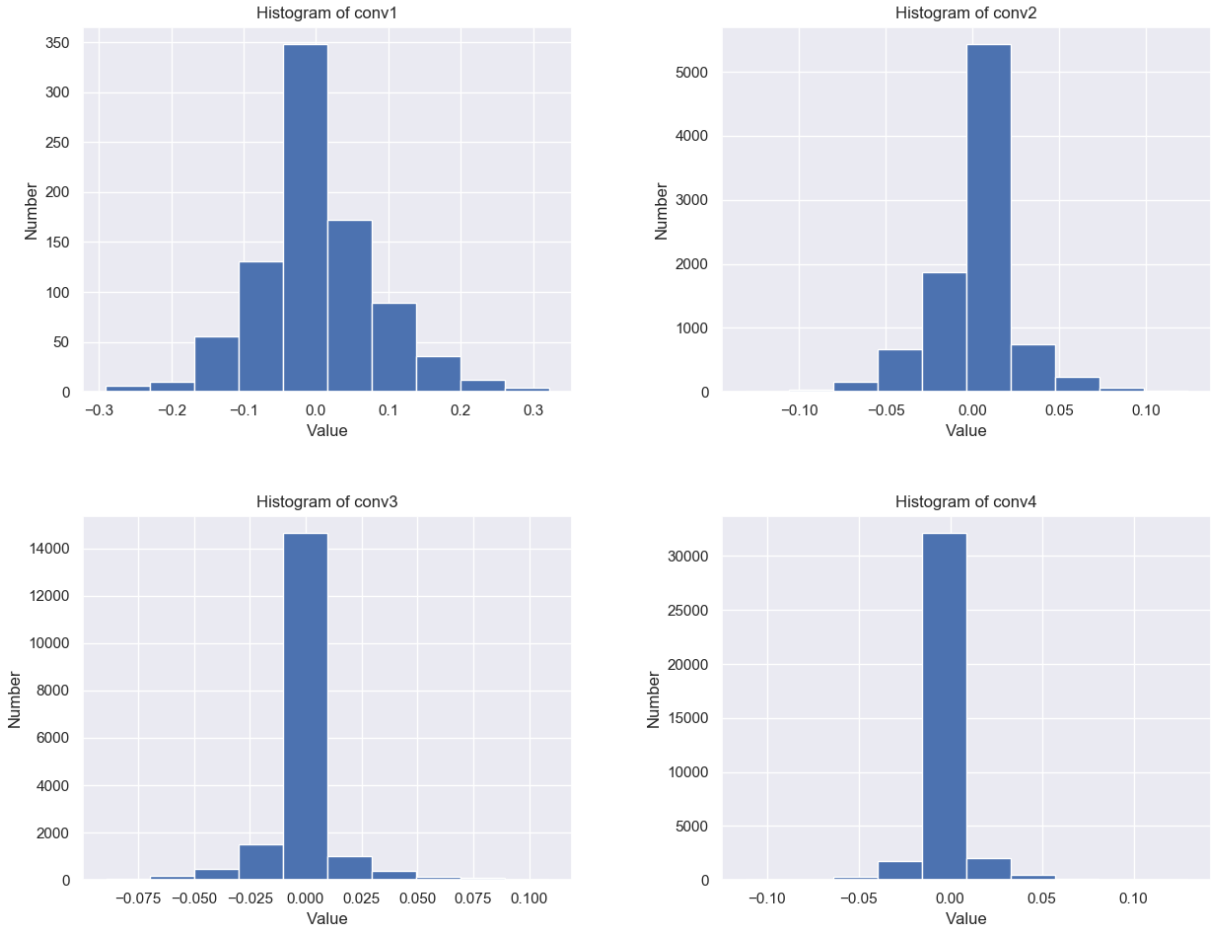



Figure 2.12: Distribution of the weights in the different layers.

## 2.5 Preprocessing

The CIFAR-10 dataset is made of images belonging to 10 different classes. All images have the same square size and are colored, such that each one has shape $32 \times 32 \times 3$. The pixel values for each image in the dataset are unsigned integers in the range $[0, 255]$.

A good starting point to pre process and normalize the pixel values involves first converting the data type from unsigned integers to floats, and then dividing the pixel values by the maximum value (i.e., 255). However, when the algorithm compares data points, the features with the larger scale will completely dominate the other.

A clever way of pre processing the pixel values consists to normalize the data using Min-Max normalization. This type of normalization guarantees that all features have the exact same scale. In particular, the following formula is applied to each image of the dataset:

```
89  def minmax_normalize(x):
90      min_val = np.min(x)
91      max_val = np.max(x)
92      x = (x - min_val) / (max_val - min_val)
93      return x
```

Figure 2.13: Min-Max normalization function implemented for the project (see *inout.py*).

In other words, for every image, the minimum value of that image gets transformed into a 0, the maximum value gets transformed into a 1, and every other value gets transformed into a decimal between 0 and 1.

# 3. Discussion

In this document, I have presented and described the convolutional neural network models build and trained for Homework 2. Each choice made for building and training the models has been discussed and motivated.

The first model is straightforward and obtains rather good results with few training iterations. The reason for this is to be addressed to the low complexity of the dataset (i.e., MNIST).

The second model requires more tuning since the dataset (i.e., CIFAR-10) is more complex, which implies that also the model must be more complex. As a matter of fact, the first model has been tested on this dataset and the results returned by the model are very poor.

The whole convolutional neural network architectures presented in this paper have been implemented from scratch without high-level libraries (e.g., Keras). But, due to the limited available computational power, in order to compute the results requested by the homework, I also implemented the same exact model using Keras libraries. Moreover, since computing a high performance model is out of the purpose of this homework, I have not accurately trained the presented models.

To conclude, the presented models are capable of detecting digits/classes belonging to the respective datasets. The models have been implemented from scratch. For further work, other regularization techniques (e.g., Dropout, Batch Normalization) and optimizers (e.g., Adam) can be implemented.

# APPENDIX

## A.1 GitHub Repository

The code of the project is available on GitHub, in the repository [7].

## A.2 Training Outputs

The training outputs of the model *without* regularization are listed in Figure 1. We can observe that the model is capable of learning the digit recognition task in few thousand of iterations. To achieve even better results, we can keep training the model for more epochs. But, the main bottleneck for running this process is the computational time required. In particular, the model is capable of training for 1000 iterations in about 113 seconds. This means that the total time required by the model to train over the whole training set for 1 epoch is about 110 minutes.

The training outputs of the model *with* regularization are listed in Figure 2. We can observe that the training and validation accuracies and losses improve iteration after iteration. After 1 epoch of training, the trained model has an accuracy equal to 87.182% on the test set, while the trained regularized model has an accuracy equal to 89.640% on the test set. This difference can be addressed to many factors. First, the regularization term makes the training more stable and therefore more accurate. Second, using L2 regularization makes the model less overfit the data (i.e., captures less noise) and this is why the performance of the model on the test set is better.

```
--- Epoch 1 ---
[Step 00000]: Loss 2.000 | Accuracy: 0.000 | Time: 70.93 seconds | Validation Loss 2.304 | Validation Accuracy: 9.500
[Step 01000]: Loss 1.252 | Accuracy: 59.540 | Time: 115.02 seconds | Validation Loss 0.769 | Validation Accuracy: 75.880
[Step 02000]: Loss 1.043 | Accuracy: 68.116 | Time: 114.66 seconds | Validation Loss 0.648 | Validation Accuracy: 80.680
[Step 03000]: Loss 0.919 | Accuracy: 72.076 | Time: 114.66 seconds | Validation Loss 0.670 | Validation Accuracy: 80.720
[Step 04000]: Loss 0.855 | Accuracy: 74.031 | Time: 114.26 seconds | Validation Loss 0.580 | Validation Accuracy: 83.740
[Step 05000]: Loss 0.806 | Accuracy: 75.665 | Time: 114.17 seconds | Validation Loss 0.513 | Validation Accuracy: 84.760
[Step 06000]: Loss 0.772 | Accuracy: 76.854 | Time: 114.57 seconds | Validation Loss 0.528 | Validation Accuracy: 84.500
[Step 07000]: Loss 0.728 | Accuracy: 78.203 | Time: 114.47 seconds | Validation Loss 0.548 | Validation Accuracy: 83.860
[Step 08000]: Loss 0.711 | Accuracy: 78.728 | Time: 115.02 seconds | Validation Loss 0.500 | Validation Accuracy: 84.820
[Step 09000]: Loss 0.688 | Accuracy: 79.447 | Time: 114.57 seconds | Validation Loss 0.423 | Validation Accuracy: 88.200
[Step 10000]: Loss 0.675 | Accuracy: 79.872 | Time: 115.31 seconds | Validation Loss 0.465 | Validation Accuracy: 85.940
[Step 11000]: Loss 0.658 | Accuracy: 80.438 | Time: 115.47 seconds | Validation Loss 0.407 | Validation Accuracy: 88.500
[Step 12000]: Loss 0.644 | Accuracy: 80.943 | Time: 114.77 seconds | Validation Loss 0.402 | Validation Accuracy: 88.220
[Step 13000]: Loss 0.631 | Accuracy: 81.371 | Time: 114.80 seconds | Validation Loss 0.434 | Validation Accuracy: 87.740
[Step 14000]: Loss 0.613 | Accuracy: 81.916 | Time: 114.60 seconds | Validation Loss 0.410 | Validation Accuracy: 88.960
[Step 15000]: Loss 0.600 | Accuracy: 82.295 | Time: 114.88 seconds | Validation Loss 0.398 | Validation Accuracy: 88.820
[Step 16000]: Loss 0.590 | Accuracy: 82.664 | Time: 115.07 seconds | Validation Loss 0.464 | Validation Accuracy: 86.540
[Step 17000]: Loss 0.580 | Accuracy: 82.936 | Time: 114.69 seconds | Validation Loss 0.422 | Validation Accuracy: 87.320
[Step 18000]: Loss 0.577 | Accuracy: 83.140 | Time: 115.23 seconds | Validation Loss 0.378 | Validation Accuracy: 89.260
[Step 19000]: Loss 0.569 | Accuracy: 83.364 | Time: 114.92 seconds | Validation Loss 0.411 | Validation Accuracy: 87.860
[Step 20000]: Loss 0.563 | Accuracy: 83.476 | Time: 115.22 seconds | Validation Loss 0.420 | Validation Accuracy: 89.720
[Step 21000]: Loss 0.556 | Accuracy: 83.691 | Time: 114.93 seconds | Validation Loss 0.372 | Validation Accuracy: 89.680
[Step 22000]: Loss 0.549 | Accuracy: 83.905 | Time: 114.68 seconds | Validation Loss 0.399 | Validation Accuracy: 88.320
[Step 23000]: Loss 0.543 | Accuracy: 84.101 | Time: 114.63 seconds | Validation Loss 0.452 | Validation Accuracy: 88.240
[Step 24000]: Loss 0.537 | Accuracy: 84.301 | Time: 114.46 seconds | Validation Loss 0.396 | Validation Accuracy: 87.920
[Step 25000]: Loss 0.535 | Accuracy: 84.385 | Time: 116.29 seconds | Validation Loss 0.468 | Validation Accuracy: 86.760
[Step 26000]: Loss 0.531 | Accuracy: 84.485 | Time: 115.20 seconds | Validation Loss 0.379 | Validation Accuracy: 89.100
[Step 27000]: Loss 0.528 | Accuracy: 84.589 | Time: 114.39 seconds | Validation Loss 0.407 | Validation Accuracy: 88.940
[Step 28000]: Loss 0.525 | Accuracy: 84.661 | Time: 115.00 seconds | Validation Loss 0.399 | Validation Accuracy: 87.800
[Step 29000]: Loss 0.521 | Accuracy: 84.801 | Time: 114.73 seconds | Validation Loss 0.391 | Validation Accuracy: 88.660
[Step 30000]: Loss 0.517 | Accuracy: 84.917 | Time: 114.36 seconds | Validation Loss 0.399 | Validation Accuracy: 88.380
[Step 31000]: Loss 0.513 | Accuracy: 85.033 | Time: 114.35 seconds | Validation Loss 0.474 | Validation Accuracy: 87.780
[Step 32000]: Loss 0.512 | Accuracy: 85.129 | Time: 114.50 seconds | Validation Loss 0.407 | Validation Accuracy: 88.920
[Step 33000]: Loss 0.510 | Accuracy: 85.210 | Time: 114.43 seconds | Validation Loss 0.419 | Validation Accuracy: 87.180
[Step 34000]: Loss 0.508 | Accuracy: 85.277 | Time: 114.51 seconds | Validation Loss 0.382 | Validation Accuracy: 88.340
[Step 35000]: Loss 0.507 | Accuracy: 85.298 | Time: 114.91 seconds | Validation Loss 0.359 | Validation Accuracy: 89.700
[Step 36000]: Loss 0.502 | Accuracy: 85.428 | Time: 114.44 seconds | Validation Loss 0.502 | Validation Accuracy: 87.980
[Step 37000]: Loss 0.499 | Accuracy: 85.506 | Time: 114.76 seconds | Validation Loss 0.379 | Validation Accuracy: 89.100
[Step 38000]: Loss 0.497 | Accuracy: 85.571 | Time: 114.54 seconds | Validation Loss 0.440 | Validation Accuracy: 86.900
[Step 39000]: Loss 0.495 | Accuracy: 85.644 | Time: 114.58 seconds | Validation Loss 0.378 | Validation Accuracy: 89.400
[Step 40000]: Loss 0.493 | Accuracy: 85.708 | Time: 114.61 seconds | Validation Loss 0.397 | Validation Accuracy: 88.700
[Step 41000]: Loss 0.491 | Accuracy: 85.827 | Time: 114.76 seconds | Validation Loss 0.402 | Validation Accuracy: 87.920
[Step 42000]: Loss 0.487 | Accuracy: 85.943 | Time: 115.16 seconds | Validation Loss 0.359 | Validation Accuracy: 90.380
[Step 43000]: Loss 0.486 | Accuracy: 85.996 | Time: 114.66 seconds | Validation Loss 0.371 | Validation Accuracy: 89.320
[Step 44000]: Loss 0.482 | Accuracy: 86.080 | Time: 114.56 seconds | Validation Loss 0.400 | Validation Accuracy: 88.840
[Step 45000]: Loss 0.481 | Accuracy: 86.111 | Time: 114.83 seconds | Validation Loss 0.542 | Validation Accuracy: 86.600
[Step 46000]: Loss 0.480 | Accuracy: 86.179 | Time: 114.90 seconds | Validation Loss 0.387 | Validation Accuracy: 90.420
[Step 47000]: Loss 0.478 | Accuracy: 86.232 | Time: 114.66 seconds | Validation Loss 0.383 | Validation Accuracy: 88.940
[Step 48000]: Loss 0.475 | Accuracy: 86.302 | Time: 115.02 seconds | Validation Loss 0.391 | Validation Accuracy: 89.480
[Step 49000]: Loss 0.473 | Accuracy: 86.376 | Time: 114.75 seconds | Validation Loss 0.385 | Validation Accuracy: 88.400
[Step 50000]: Loss 0.472 | Accuracy: 86.424 | Time: 114.80 seconds | Validation Loss 0.396 | Validation Accuracy: 88.960
[Step 51000]: Loss 0.470 | Accuracy: 86.465 | Time: 114.41 seconds | Validation Loss 0.439 | Validation Accuracy: 89.140
[Step 52000]: Loss 0.468 | Accuracy: 86.518 | Time: 115.11 seconds | Validation Loss 0.381 | Validation Accuracy: 90.680
[Step 53000]: Loss 0.467 | Accuracy: 86.566 | Time: 114.87 seconds | Validation Loss 0.390 | Validation Accuracy: 89.960
[Step 54000]: Loss 0.467 | Accuracy: 86.589 | Time: 115.07 seconds | Validation Loss 0.405 | Validation Accuracy: 87.840
Train Loss: 0.467
Train Accuracy: 86.589
```

Figure 1: Outputs produced by the model while training. The model is able to achieve a train accuracy equal to 86.589 and a test accuracy equal to 87.182 after training for 55000 iterations over the training set.

```
--- Epoch 1 ---
[Step 00000]: Loss 2.000 | Accuracy: 0.000 | Time: 83.99 seconds | Validation Loss 2.332 | Validation Accuracy: 9.400
[Step 01000]: Loss 1.078 | Accuracy: 64.735 | Time: 135.43 seconds | Validation Loss 0.677 | Validation Accuracy: 79.120
[Step 02000]: Loss 0.855 | Accuracy: 72.564 | Time: 135.54 seconds | Validation Loss 0.830 | Validation Accuracy: 73.700
[Step 03000]: Loss 0.741 | Accuracy: 76.041 | Time: 135.26 seconds | Validation Loss 0.638 | Validation Accuracy: 82.460
[Step 04000]: Loss 0.687 | Accuracy: 78.180 | Time: 136.28 seconds | Validation Loss 0.548 | Validation Accuracy: 84.760
[Step 05000]: Loss 0.658 | Accuracy: 79.404 | Time: 135.18 seconds | Validation Loss 0.477 | Validation Accuracy: 86.460
[Step 06000]: Loss 0.624 | Accuracy: 80.670 | Time: 135.62 seconds | Validation Loss 0.480 | Validation Accuracy: 86.720
[Step 07000]: Loss 0.611 | Accuracy: 81.331 | Time: 134.59 seconds | Validation Loss 0.513 | Validation Accuracy: 84.800
[Step 08000]: Loss 0.595 | Accuracy: 81.940 | Time: 135.59 seconds | Validation Loss 0.478 | Validation Accuracy: 87.540
[Step 09000]: Loss 0.587 | Accuracy: 82.324 | Time: 135.77 seconds | Validation Loss 0.477 | Validation Accuracy: 86.580
[Step 10000]: Loss 0.576 | Accuracy: 82.642 | Time: 135.25 seconds | Validation Loss 0.460 | Validation Accuracy: 86.640
[Step 11000]: Loss 0.571 | Accuracy: 82.911 | Time: 135.69 seconds | Validation Loss 0.435 | Validation Accuracy: 87.460
[Step 12000]: Loss 0.566 | Accuracy: 83.143 | Time: 136.90 seconds | Validation Loss 0.559 | Validation Accuracy: 84.080
[Step 13000]: Loss 0.557 | Accuracy: 83.394 | Time: 134.36 seconds | Validation Loss 0.503 | Validation Accuracy: 87.820
[Step 14000]: Loss 0.549 | Accuracy: 83.523 | Time: 134.24 seconds | Validation Loss 0.545 | Validation Accuracy: 84.920
[Step 15000]: Loss 0.544 | Accuracy: 83.648 | Time: 134.68 seconds | Validation Loss 0.423 | Validation Accuracy: 87.940
[Step 16000]: Loss 0.537 | Accuracy: 83.845 | Time: 135.33 seconds | Validation Loss 0.404 | Validation Accuracy: 88.920
[Step 17000]: Loss 0.529 | Accuracy: 84.119 | Time: 134.98 seconds | Validation Loss 0.423 | Validation Accuracy: 89.260
[Step 18000]: Loss 0.519 | Accuracy: 84.368 | Time: 135.03 seconds | Validation Loss 0.464 | Validation Accuracy: 88.340
[Step 19000]: Loss 0.515 | Accuracy: 84.532 | Time: 135.22 seconds | Validation Loss 0.415 | Validation Accuracy: 88.540
[Step 20000]: Loss 0.511 | Accuracy: 84.741 | Time: 135.52 seconds | Validation Loss 0.411 | Validation Accuracy: 88.640
[Step 21000]: Loss 0.504 | Accuracy: 84.991 | Time: 134.70 seconds | Validation Loss 0.401 | Validation Accuracy: 88.900
[Step 22000]: Loss 0.501 | Accuracy: 85.132 | Time: 135.04 seconds | Validation Loss 0.393 | Validation Accuracy: 89.160
[Step 23000]: Loss 0.494 | Accuracy: 85.318 | Time: 135.19 seconds | Validation Loss 0.475 | Validation Accuracy: 86.760
[Step 24000]: Loss 0.492 | Accuracy: 85.426 | Time: 135.31 seconds | Validation Loss 0.516 | Validation Accuracy: 85.160
[Step 25000]: Loss 0.490 | Accuracy: 85.557 | Time: 135.43 seconds | Validation Loss 0.388 | Validation Accuracy: 88.960
[Step 26000]: Loss 0.487 | Accuracy: 85.643 | Time: 135.27 seconds | Validation Loss 0.386 | Validation Accuracy: 89.020
[Step 27000]: Loss 0.486 | Accuracy: 85.715 | Time: 135.23 seconds | Validation Loss 0.432 | Validation Accuracy: 88.860
[Step 28000]: Loss 0.483 | Accuracy: 85.818 | Time: 135.59 seconds | Validation Loss 0.449 | Validation Accuracy: 86.960
[Step 29000]: Loss 0.481 | Accuracy: 85.894 | Time: 135.47 seconds | Validation Loss 0.429 | Validation Accuracy: 88.700
[Step 30000]: Loss 0.479 | Accuracy: 86.020 | Time: 135.44 seconds | Validation Loss 0.392 | Validation Accuracy: 89.820
[Step 31000]: Loss 0.476 | Accuracy: 86.107 | Time: 135.41 seconds | Validation Loss 0.418 | Validation Accuracy: 87.560
[Step 32000]: Loss 0.474 | Accuracy: 86.179 | Time: 135.07 seconds | Validation Loss 0.401 | Validation Accuracy: 88.880
[Step 33000]: Loss 0.469 | Accuracy: 86.313 | Time: 135.85 seconds | Validation Loss 0.422 | Validation Accuracy: 89.600
[Step 34000]: Loss 0.468 | Accuracy: 86.392 | Time: 135.51 seconds | Validation Loss 0.448 | Validation Accuracy: 88.560
[Step 35000]: Loss 0.467 | Accuracy: 86.443 | Time: 135.94 seconds | Validation Loss 0.383 | Validation Accuracy: 89.380
[Step 36000]: Loss 0.466 | Accuracy: 86.495 | Time: 135.73 seconds | Validation Loss 0.396 | Validation Accuracy: 89.120
[Step 37000]: Loss 0.465 | Accuracy: 86.533 | Time: 135.14 seconds | Validation Loss 0.424 | Validation Accuracy: 88.660
[Step 38000]: Loss 0.464 | Accuracy: 86.561 | Time: 136.83 seconds | Validation Loss 0.384 | Validation Accuracy: 89.400
[Step 39000]: Loss 0.462 | Accuracy: 86.652 | Time: 135.50 seconds | Validation Loss 0.416 | Validation Accuracy: 89.360
[Step 40000]: Loss 0.460 | Accuracy: 86.750 | Time: 135.42 seconds | Validation Loss 0.429 | Validation Accuracy: 88.680
[Step 41000]: Loss 0.460 | Accuracy: 86.749 | Time: 135.28 seconds | Validation Loss 0.379 | Validation Accuracy: 88.680
[Step 42000]: Loss 0.458 | Accuracy: 86.810 | Time: 134.52 seconds | Validation Loss 0.393 | Validation Accuracy: 89.800
[Step 43000]: Loss 0.457 | Accuracy: 86.865 | Time: 135.42 seconds | Validation Loss 0.410 | Validation Accuracy: 88.140
[Step 44000]: Loss 0.457 | Accuracy: 86.893 | Time: 135.05 seconds | Validation Loss 0.414 | Validation Accuracy: 89.220
[Step 45000]: Loss 0.456 | Accuracy: 86.949 | Time: 135.27 seconds | Validation Loss 0.434 | Validation Accuracy: 87.540
[Step 46000]: Loss 0.455 | Accuracy: 86.957 | Time: 134.96 seconds | Validation Loss 0.419 | Validation Accuracy: 89.560
[Step 47000]: Loss 0.453 | Accuracy: 87.028 | Time: 135.15 seconds | Validation Loss 0.455 | Validation Accuracy: 88.380
[Step 48000]: Loss 0.454 | Accuracy: 87.046 | Time: 135.06 seconds | Validation Loss 0.376 | Validation Accuracy: 89.400
[Step 49000]: Loss 0.454 | Accuracy: 87.053 | Time: 135.24 seconds | Validation Loss 0.387 | Validation Accuracy: 88.140
[Step 50000]: Loss 0.453 | Accuracy: 87.078 | Time: 135.37 seconds | Validation Loss 0.432 | Validation Accuracy: 87.660
[Step 51000]: Loss 0.452 | Accuracy: 87.098 | Time: 134.71 seconds | Validation Loss 0.404 | Validation Accuracy: 88.420
[Step 52000]: Loss 0.451 | Accuracy: 87.146 | Time: 135.06 seconds | Validation Loss 0.391 | Validation Accuracy: 89.500
[Step 53000]: Loss 0.449 | Accuracy: 87.189 | Time: 134.74 seconds | Validation Loss 0.413 | Validation Accuracy: 89.180
[Step 54000]: Loss 0.447 | Accuracy: 87.259 | Time: 135.31 seconds | Validation Loss 0.413 | Validation Accuracy: 88.180
Train Loss: 0.447
Train Accuracy: 87.259
```

Figure 2: Outputs produced by the regularized model while training. The model is able to achieve a train accuracy equal to 87.259 and a test accuracy equal to 89.640 after training for 55000 iterations over the training set.

# Bibliography


[1] Bottou L., and Bousquet O. *The Tradeoffs of Large Scale Learning*. Optimization for Machine Learning. MIT Press, pp. 351–368, Cambridge, 2012.

[2] Nair V., and Hinton G. *Rectified Linear Units Improve Restricted Boltzmann Machines*. ICML. 2010.

[3] Springenberg J., Dosovitskiy A., Brox T., and Riedmiller M. *Striving for Simplicity: The All Convolutional Net*. Department of Computer Science, University of Freiburg, Freiburg (Germany), 2014.

[4] Stigler S. *Francis Galton's Account of the Invention of Correlation*. Statistical Science. pp. 73–79, 1989.

[5] Taddy S. *Stochastic Gradient Descent*. Business Data Science: Combining Machine Learning and Economics to Optimize, Automate, and Accelerate Business Decisions. McGraw-Hill, pp. 303–307, New York, 2019.

# Sitography


[6] CS231n Convolutional Neural Networks for Visual Recognition, last access 1 Dec. 2019, [Online].
*http://cs231n.github.io/neural-networks-1/#actfun*

[7] Github Repository, last access 2 Dec. 2019, [Online].
*https://github.com/AlessandroSaviolo/DeepLearning_0845086_HW2*

[8] Google Colab, last access 3 Dec. 2019, [Online].
*https://colab.research.google.com*

[9] Keras, last access 3 Dec. 2019, [Online].
*https://keras.io/*

[10] The CIFAR-10 dataset, last access 1 Dec. 2019, [Online].
*https://www.cs.toronto.edu/ kriz/cifar.html*

[11] The MNIST database of handwritten digits, last access 1 Dec. 2019, [Online].
*http://yann.lecun.com/exdb/mnist/*