Optimization of ConvNets for big data computing and binary image classification

Rossi Lorenzo (Student ID 982595)

University of Milan, MSc in Data science and Economics

Project for the Course of

Algorithms for Massive Datasets, Cloud and Distributed Computing

10 December 2022

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

Abstract

In this work different Convolutional Neural Network architectures have been tested for a binary image classification task. Models have been tested with different layer structures and parameters in order to find the best one. In the end, many of the networks showed highly performing results, but the best ones were found with a Convolutional Neural Network where regularization layers of Dropout and Batch Normalization have been added, bringing the highest accuracy and lowest loss among the other architectures.

1 Introduction

This paper will focus on the problem of Image Classification, which consists in the task of attempting to comprehend an entire image as a whole. The goal is to classify the image by assigning it to a specific label. Typically, Image Classification refers to images in which only one object appears and is analyzed.

For this project the task was to develop different Neural Network architectures in order to find the best model for a binary classification task related to comics faces and real faces. The first part of this work will cover the pre-processing of data and how the dataset is structured. Consequently, the theoretical foundations of deep learning will be explained, with a focus on the functioning of Convolutional Neural Network. The following step will be the analysis of the network architectures that has been developed and tested. Finally, the results of the analysis will be discussed, along with some considerations on to improve the performances of the aforementioned models.

2 Dataset and data preparation

The dataset consists in 20.000 images of comics faces and real faces (10.000 for each class), coming from Kaggle[1]. Data preparation and processing were done using the OpenCV library for Python and this part of the work consisted in the following steps: all the images have been converted from JPEG to RGB and scaled down to 100x100 pixels in order to obtain a dataset of images with the same dimensions. Moreover, colors were converted to black and white (gray-scale), because the

[1] https://www.kaggle.com/datasets/defileroff/comic-faces-paired-synthetic-v2

challenge with images having multiple color channels is that we have huge volumes of data to work with which makes the process computationally intensive and, unless color is a principal discriminant in the classification task, it doesn't always lead to better results. Moreover, to improve model performances the gray-scale values have been re-scaled from the [0, 255] range to the [0, 1] range. 80% of the images have been used for training and 20% for testing. In the training dataset, an additional 20% of the images was used as validation set.

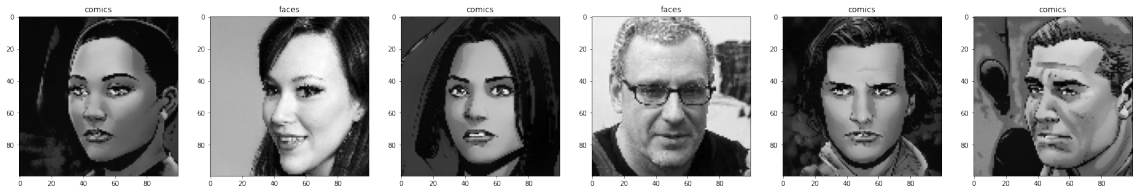


Figure 1: Sample of processed images.

3 Theoretical Framework

3.1 Neural Networks

Neural Networks are a class of predictors inspired by the structure of the brain, which consists (oversimplifying) in a large and complex network of interconnected computing devices, the neurons, through which the brain can perform highly complex computations. The simplest neural network architecture is the feedforward neural network.

The definition of network comes from the fact that they are typically represented as a directed acyclic graph $G = (V, E)$[Fig. 2], where each node (or unit) $j \in V$ computes a function $g(\mathbf{v})$ where $\mathbf{v}$ is the output of the nodes i such that $(i, j) \in E$. The nodes are divided in three subsets: the input nodes, the hidden nodes which have both incoming and outgoing edges and the output nodes.

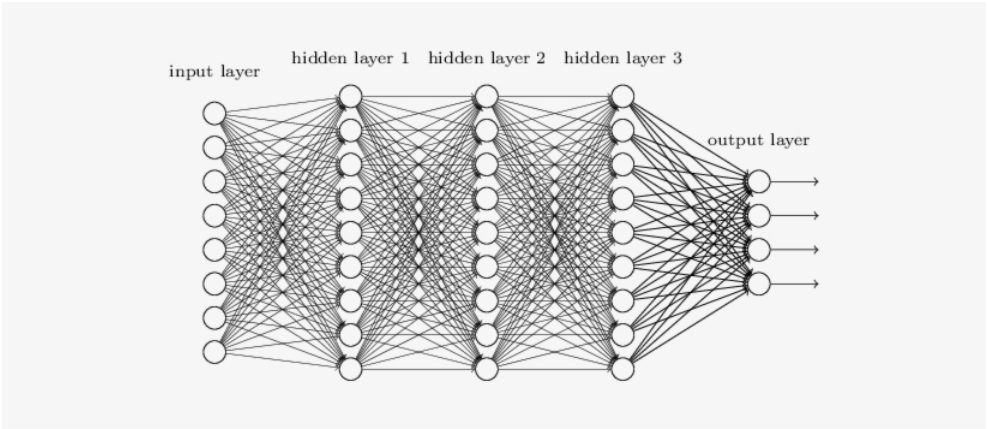


Figure 2: Graphical structure of a network.

Networks are composed by many different functions. For example, we might have three functions connected in a chain to form the following system:

$$g^{(1)}, g^{(2)}, g^{(3)} \rightarrow g(\mathbf{v}) = g^{(3)}(g^{(2)}(g^{(1)}(\mathbf{v}))) \tag{1}$$

Where each of the $g^{(x)}$ is called a "layer", thus having $g^{(1)}$ as first layer, $g^{(2)}$ as second layer and so on. Each layer is composed by many nodes that compute the functions. Moreover, to every edge $(i, j) \in E$ a parameter (called "weight)" $w_{i,j} \in \mathbb{R}$ is associated. Considering all the other edges, this creates a weight matrix $W = |V| * |V|$. This gives the idea of networks as a complex sequence of matrix operations.

The goal of a network is to approximate some function f^* and in order to understand its functioning it is important to recall how linear models work. In the case of a linear classifier $y = f^*(\mathbf{x})$ the function maps $\mathbf{x}$ to a class y. However, linear models have the defect to depend only on linear

functions. Neural Networks solve the problem of learning from non-linearity by applying a non-linear transformation to $\mathbf{x}$:

$$y = g(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + \mathbf{c}) \tag{2}$$

Where $\mathbf{x}$ is the input, $\mathbf{w}$ is the weight (or parameter), $\mathbf{c}$ is the (possible) bias and σ is a non-linear function such that $\sigma : \mathbb{R} \to \mathbb{R}$, and it is called "activation function". There is broad range of activation functions which have to be chosen carefully depending on the task. Each layer has its own activation function. In this work we'll refer to two of the most used functions:

ReLU: denoted by $f(z) = max(0, z)$. It is mainly used in input and hidden layers and its advantages comes from the fact that the gradients remain large and consistent in the nodes where it is applied.

Sigmoid: denoted by $\sigma(z) = 1/(1 + e^{-z})$. It is commonly used to produce the parameter of a Bernoulli distribution because its range is (0,1). It's mainly applied in binary classification where the task is to determine whether a data point belongs to a class or not and because of this it is usually applied in the output nodes.

Basically, when building a Neural Network one must take into consideration how many layers to include, the number of nodes per layer, how these layers are connected, the choice of the activation functions, i.e. a set of many parameters and structural components that need to be tuned finely in order to find the best learning model.

3.2 Convolutional Neural Networks

Convolutional Networks (or ConvNets) are a special kind of neural networks used to process data that has a grid-like topology, such as forecasting time series (which can be considered as a 1-D grid of samples at regular intervals of time) or classifying images, interpreted as a 2-D grid of pixels.

3.2.1 Convolutional Layers

The name of this particular structure of neural networks comes from the "convolution", a mathematical linear operation that the network employs instead of general matrix multiplication inside the layers. Convolution is used to obtain more accurate estimates by averaging The operation is denoted by the following equation:

$$s = (x * w) \tag{3}$$

Where x is the input and w is the kernel and they are usually multidimensional arrays of data and parameters respectively. The kernel, or feature detector, can be set to different dimensions, for example, 3x3. The kernel works as a filter and to perform convolution, it goes over the input image (in case of image classification), doing matrix multiplication element after element [Fig. 3]. Usually, these operations are performed with respect to t (in this case we write $s(t) = (x * w)(t)$), which is an index and integer value. It can be the position of a pixel or a date of measurement and in this case the kernel performs a weighting operation that gives more weight to the values that are "closer" to the one we're estimating.

Another element that composes a convolutional layer is the stride. It represents the slide of the kernel when shifting from one tile of data points to another on which the convolution is computed. When the stride is 1 then we move the filter one pixel at a time. For computer vision and image processing task, it's unusual to use a stride bigger than 1.

3.2.2 Pooling Layers

Consequently to Convolution there is Pooling: a pooling function is usually applied after the convolution layer, in order to further modify and down-sample the output of the layer. The filter of a pooling layer is always smaller than a feature map. In this case it has been used the Max Pooling function with a 2*2 window, which reports the maximum output in a square-shaped neighborhood [Fig. 4]. Pooling operations help to make representations almost invariant from translation of the

input, reducing computational costs and allowing the learning of high-level pattern by deeper layers of the network.

3.2.3 Flattening and Dense Layers

Blocks of Convolution and MaxPooling layers are usually followed by a Flatten layer, which transforms the 3D or 2D input into a one-dimensional vector. The one-dimensional vector created by the Flatten layer can then pass through one or a series Dense Layers, also referred to as Fully Connected layers. These layers can learn complex relationships between the high-level features learnt by the preceding blocks of Convolution and MaxPooling Layers. This layer provides the model with the ability to finally understand images: there is a flow of information between each input pixel and each output class. The hyperparameters of this layer are the number of nodes and the activation function which, since for this task we're dealing with binary classification, will be Sigmoid in the last Dense layer.

Generalizing, we can conclude that the typical structure of a ConvNet is the following:

1. **Convolutional layer** (including Input layer)
2. **Pooling layer**
3. **Flattening**
4. **Fully connected layers** (including Output layer)

The aforementioned layers are not the only ones that compose ConvNets. There are other two types of layers which are not essential but they help improving learning performances. These layers will be implemented in this work.

3.2.4 Dropout

Dropout layers are placed after the Pooling layer and they function as masks that nullify the contribution of some neurons towards the next layer and leaves unmodified all others. We can apply a Dropout layer to the input vector, in which case it nullifies some of its features; but we can also apply it to a hidden layer, in which case it nullifies some hidden neurons. Dropout layers are important in training ConvNets because they prevent overfitting on the training data. If they are not present, the first batch of training samples influences the learning in a disproportionately high manner. Dropout can also be applied after a Dense layer.

3.2.5 Batch normalization

Batch normalization layers are placed after the Convolutional layer and before the Pooling layer, are proposed as a technique to help coordinate the update of multiple layers in the model by scaling the output of the layer, specifically by standardizing the activations of each input variable per mini-batch, such as the activations of a node from the previous layer (standardization refers to rescaling data to have a mean of zero and a standard deviation of one, e.g. a standard Gaussian). Standardizing the activations of the prior layer means that assumptions the subsequent layer makes about the distribution of inputs during the weight update will not change, or at least not dramatically. This has the effect of stabilizing and speeding-up the training process.

4 Tested Models and Parameters

Note: all the models have been developed and trained using the TensorFlow framework.

4.1 Models

For this work, three ConvNets have been built. The layer composition of the selected networks is the following:

1. **Model 1**: one convolutional input layer and one convolutional hidden layer with 32 nodes, followed by flattening and two Dense layers with 128 and 1 (the output layer) nodes respectively.

2. **Model 2**: one convolutional layers with 32 nodes and two convolutional hidden layers with 32 and 64 nodes respectively, followed by flattening and two Dense layers with 128 and 1 (the output layer) nodes respectively.

3. **Model 3**: one convolutional layers with 32 nodes and three convolutional hidden layers with 32, 64 and 128 nodes respectively, followed by flattening and two Dense layers with 128 and 1 (the output layer) nodes respectively.

To all the convolutional layers and the first dense layers the ReLU function has been applied, while the Sigmoid function was used on the last Dense layer. Plus, as already explained in 4.2, a 3*3 kernel was applied to all the convolutional layers, as well as a Max Pooling with 2*2 shape layer after each block.

4.1.1 Architecture variants of the models with regularization layers

Each of the three ConvNets have been trained and tested using four different network architectures in which different types of regularization layers (such as dropout and batch normalization) have been progressively added in order to analyze how these additions affect learning performances. The architecture variants are the following:

1. **Base architecture** [Fig. 5.a]: it consists in consecutive convolutional layers (defined in 4.1), with each one of them followed by a pooling layer, and terminating in a Flatten and two Dense layers. For the convolutional layers, the kernel size was set to 3*3 and the stride was set to 1. The pooling layer was set to Max Pooling with a 2*2 window. To each of the convolutional layer have been applied the ReLU activation function, as well as to the first Dense layer (which also presents a different number of nodes depending on the model) while for the final Dense layer (which returns the output) the Sigmoid activation function has been chose, with a number of nodes equal to 1.

2. **Architecture + Droput** [Fig. 5.b]: Dropout layers have been added to the previous architecture, more specifically, a layer with dropout rate of 0.25 have been applied after each pooling layer and another one with dropout rate of 0.5 has been inserted after the first Dense layer.

3. **Architecture + Droput + Batch Normalization** [Fig. 5.c]: this third variant adds one Batch Normalization layer after each convolutional layer (and before the pooling layer) in order to stabilize the learning process and verify the eventual improvements of the models.

4. **Architecture + Droput + Batch Normalization + Additional Dense Layer** [Fig 5.d]: the last architectures adds a second Dense layer with 128 nodes and dropout rate of 0.25 in order to check whether an additional fully connected layer helps improving the overall performance.

Note: the architectures shown in Fig. 7 present the number of layers of model 1. It is an explicative example of how the architecture variants apply to any model presented in this paper.

4.2 Hyperparameters and Compilation

The initial hyperparameters and model compilation setting is the following:

1. **Epochs**: 25

2. **Batch size**: 64

3. **Optimizer**: Adam

4. **Metrics**: Binary Accuracy

5. **Loss**: since we're dealing with binary classification, the choice for the loss function is the Binary Crossentropy, which computes the average of the log losses of predicted class probabilities

(the log is used as a penalization term for the classifier):

$$Loss = -\frac{1}{N}\sum_{i=1}^{N} y_i log(p(y_i)) + (1 - y_i) log(1 - p(y_i)) \tag{4}$$

5 Results

We now discuss the results of the networks. It is clear that all the models manage to reach an high level of accuracy in both training and testing [Tab. 1]. The Base architecture (composed only by Conv2D and Dense layers without regularization) presents the third model (composed by three Conv2D hidden layers) as the most accurate one [Fig. 6], even though the lowest loss is reached with the first model (presenting only one hidden Conv2D layer). In the second architecture variant which include Dropout layers presents Model 3 as the best in terms of accuracy and loss [Fig. 7], while the other two models show similar performances. The third architecture, which includes Batch Normalization layers, has signs of overfitting in Model 3 [Fig. 8]: despite the results, it shows a spike of high validation loss in the last epochs. Model 1 and 2 reach a test accuracy of 100% with even lower loss than in the previous architectures. While this result means that the models can correctly classify the test data, such an high accuracy may be tested with additional training or regularization. Perhaps a K-Fold Cross Validation would lead to lower but more consistent estimates. The fourth architecture shows better results for all the three models when compared to the first two. Similarly to the second architecture, Model 3 has the best results again and it doesn't show any overfitting [Fig. 9], meaning that the additional Dense layer helped in stabilizing the model during learning. However, it is possible to conclude that the best results were obtained by adding regularization layers, such Dropout and Batch Normalization.

Table 1: Loss and Accuracy of models and architecture variants

Model	Base	Dropout	Batch Norm	Dense +1
Model 1	0.0029, 0.999	0.0032, 0.9992	0.0001, 1.0	0.0031, 0.9995
Model 2	0.0057, 0.9985	0.0031, 0.999	2.87e-05, 1.0	0.0029, 0.9992
Model 3	0.0047, 0.9995	0.0011, 0.9997	0.0048, 0.9982	0.0004, 0.9997

6 Conclusion

This project was aimed to test different Neural Network architectures on a large dataset of images for a binary classification task in order to find the best predictive model. The resulting tests applied to the three ConvNets models confirm that using regularization and optimization techniques such as adding Dropout and Batch Normalization layers can help the network to improve its learning and . To improve the results even more and get more robust estimates, it may be considered to use Data augmentation layers, perform K-Fold Cross Validation, add other convolutions or change the number of nodes in each layer. Finally, one must not forget that hyperparameter tuning is crucial for network functioning. In this case we trained the models using standard hyperparameter values (25 epochs and batch size of 64), but increasing the number of epochs or reducing the batch size may lead to more consistent estimates. Neural Networks are not pre-built models and they must be carefully tuned depending on the task.

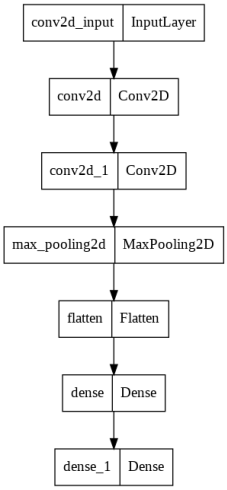
Bibliography

Bengio, Y., Courville, A., Goodfellow, I. (2016). *Deep Learning*, The MIT Press.

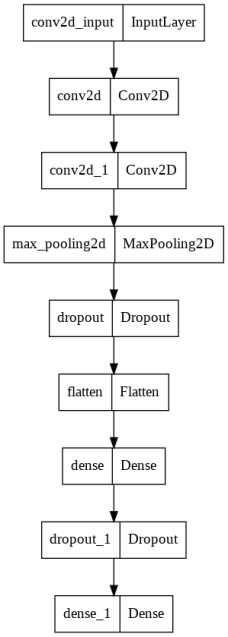
Notes from Stanford CS class: *CS231n: Convolutional Neural Networks for Visual Recognition*[2], Spring 2022, Stanford University.

Images and Plots

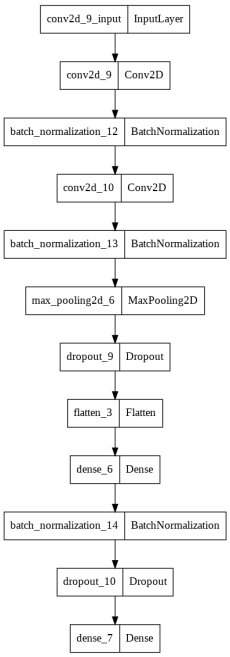
[2]https://cs231n.github.io/



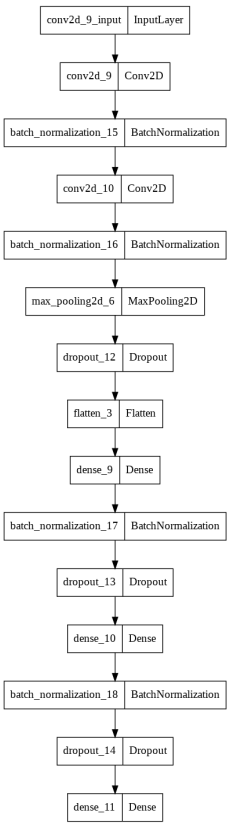
(a) Base network architecture



(b) Network architecture with dropout layers

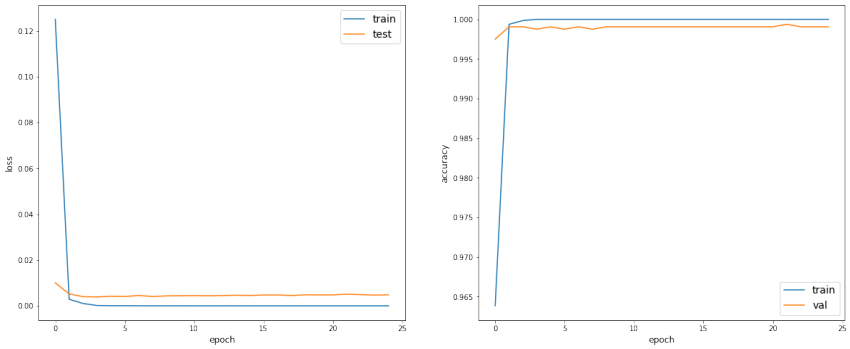


(c) Network architecture with dropout layers and batch normalization

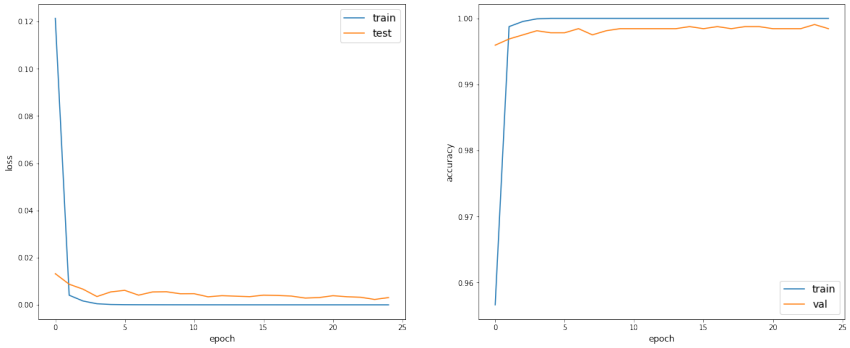


(d) Network architecture with dropout layers, batch normalization and additional Dense layer

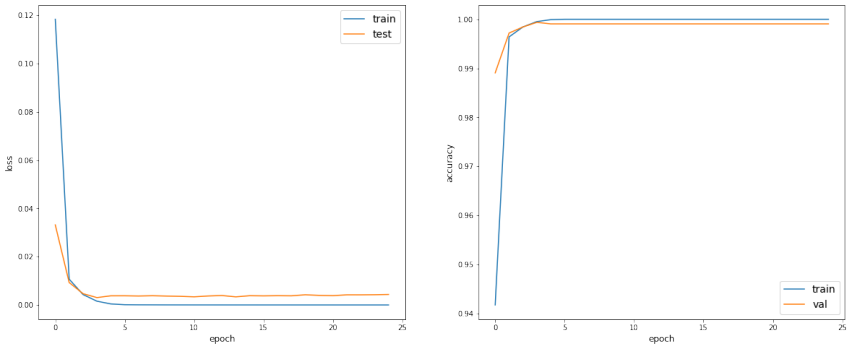
Figure 3: Network architectures variants applied to the models



(a) Model 1

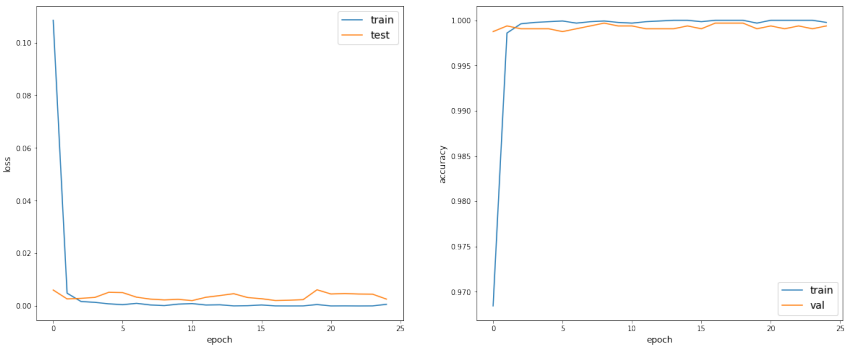


(b) Model 2

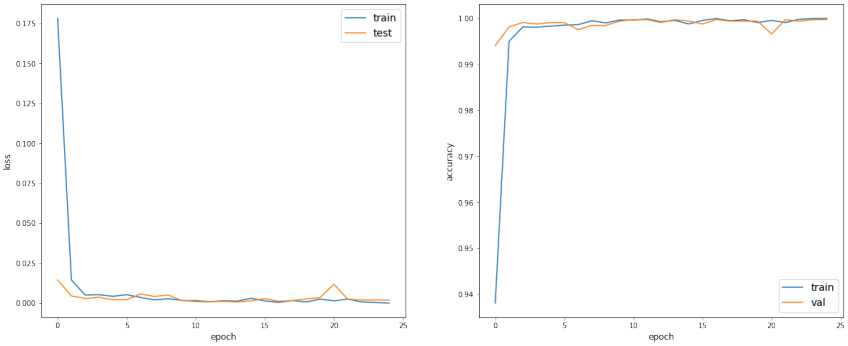


(c) Model 3

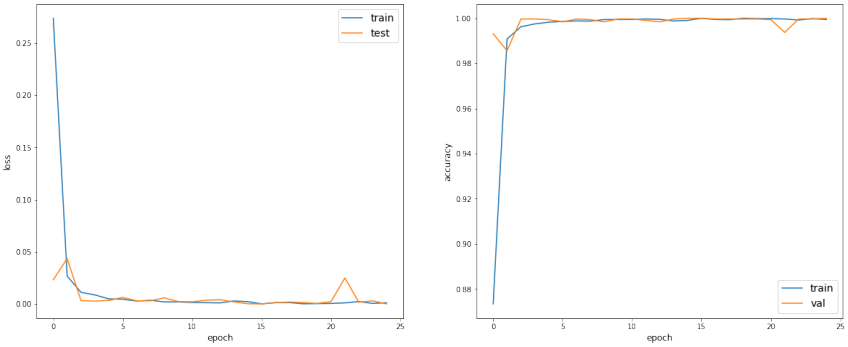
Figure 4: Accuracy and loss of the three models with base architecture.



(a) Model 1

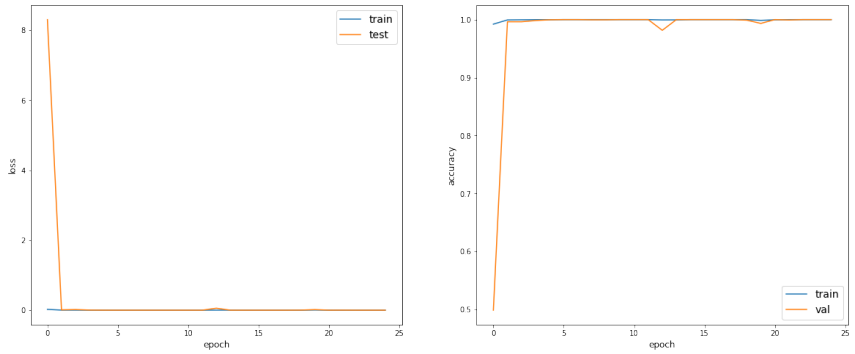


(b) Model 2

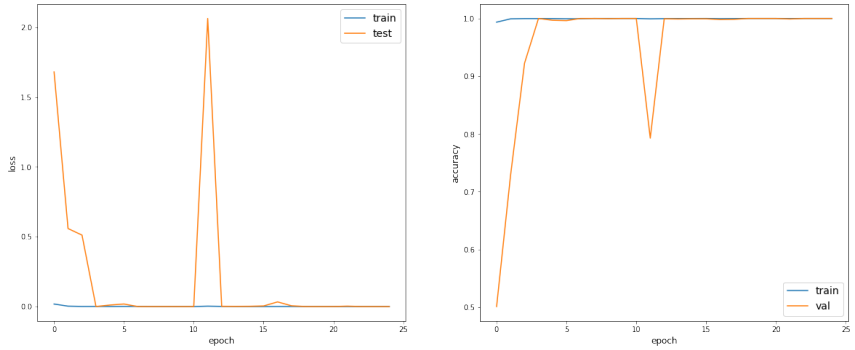


(c) Model 3

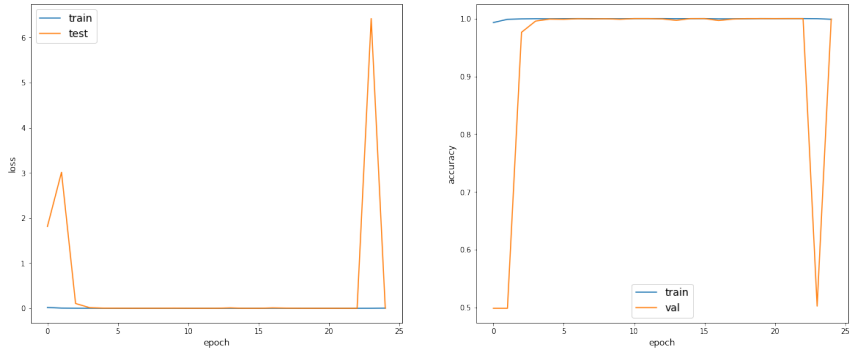
Figure 5: Accuracy and loss of the three models with dropout layers.



(a) Model 1

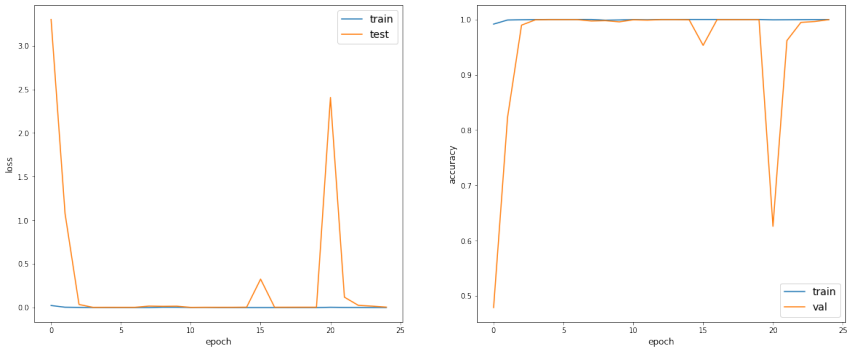


(b) Model 2

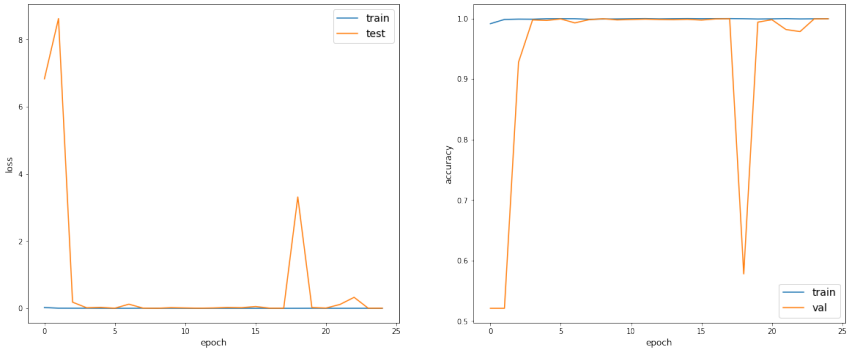


(c) Model 3

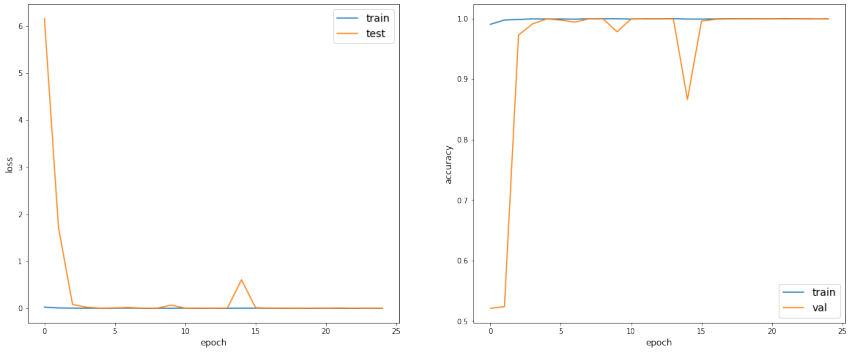
Figure 6: Accuracy and loss of the three models with dropout layers and batch normalization layers.



(a) Model 1



(b) Model 2



(c) Model 3

Figure 7: Accuracy and loss of the three models with dropout layers, batch normalization and additional Dense layer.