

**MACHINE LEARNING 2019**

**Prof. Paola Velardi**

**SAPIENZA UNIVERSITA DI ROMA**

# **FEATURE EXTRACTION USING DEEP CONVOLUTIONAL NEURAL NETWORKS FOR OFFLINE SIGNATURE VERIFICATION**

## Muhammad Aimal – 1848097

## Manoochehr Joodi Bigdello – 1860273

# Table of Contents

1. Motivation
2. Neural Networks and Deep Learning
3. Support Vector Machines
4. Multi-Output CNN Model Implementation
5. SVM Implementation
6. Conclusion
7. References

# **Motivation**

Signature verification is still one of the most used methods for identity verification. Being an easy to implement system, it also comes with its drawbacks, one of them being forgeries. Each system has a different sensitivity towards forgeries, e.g. a successfully forgery with in a bank may cause the loss of millions where as it may not do much harm in other organizations.

In this project we were motivated to tackle the problem of **Skilled Forgeries** with the use of Deep learning and Support Vector Machines to see if we can extract some features that may be able to distinguish not only between two different users but also between the two class of users i.e. a genuine signature or a forged signature image.

# **Neural Networks and Deep Learning**

* **Neural Networks**

Neural Networks or Artificial Neural Networks is a machine learning model based on the biological neural networks present in the brains of humans.

A neural network learns without being explicitly using previous examples or data and tries to construct a model that is able to correctly predict future unseen data.

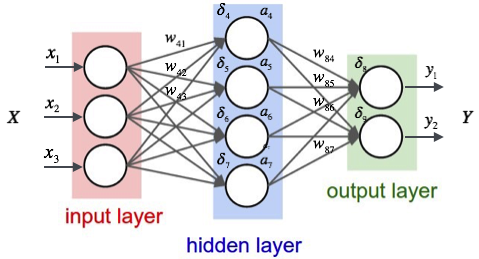


Figure 1: Artificial Neural Network

An artificial neural network is formed by connecting nodes called **artificial neurons** modeled after the real neurons present in the biological brain and each connection between nodes in subsequent layers model the synapses in the biological brain. The connection between the neurons are called edges. Each edge has a weight associated with it and each neuron performs a weighted sum of the inputs from nodes in the previous layers and then passes the sum through an **activation function** aka threshold or bias.

Weights on the edges are updated an in iterative manner. This weight update rule is based on the use of **Gradient Descent.** The algorithm used for updating the weights iteratively is called **Backpropagation.**

**Output** is calculated at the output layer again by the weighted sum of all the outputs of the nodes in the previous layer and then the error is calculated by comparing the calculated output and the real output of the training example that was fed in to generate the activations.

Several **Error/Loss Functions** are available for the calculation of error at the output layer one of the most common and the one that we’ve used in our network (discussed later) is the **Cross-Entropy Function.**

* **Backpropagation**

In backpropagation the error computed at the output is backpropagated throughout the network using gradient descent. Each weight gets an update respective to the derivative of the error function w.r.t the weight. It utilizes the chain-rule to iteratively compute derivatives at each layer and then the weights are updated accordingly.

Equation 1: Mean Squared Error Function

So, if we use the mean squared error function to calculate the error/loss on the output node then the change in each weight can be calculated using the following equation.

Equation 2: Weight Update Rule

* **Loss Function (Cross-Entropy)**

Equation 3: Cross-Entropy Loss

Cross-entropy loss aka log loss is a function that calculates the probability value for each node in the output layer in a neural network. The cross-entropy loss increases as the predicted probabilities diverges from the actual values for the probabilities. A log loss of 0 would mean a 100% accurate model.



Figure 2: Cross-Entropy Loss Function

* **Deep Learning**

A neural consists of three layers, an input layer, a hidden layer and an output layer. In deep learning or deep neural networks, multiple hidden layers are used to generate even more complex networks able to tackle even more challenging problems. A deep neural network is able to find even non-linear separation between different classes.

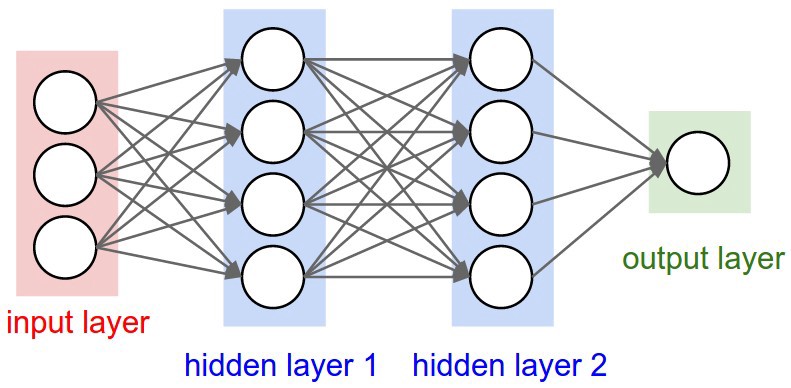


Figure 3: Deep Neural Network

* **Convolutional Neural Networks**

A Convolutional Neural Network (CNN) is an implementation of Deep Neural Networks which are mainly used when working with imagery data.

Convolutional neural networks use multiple hidden layers of different purposes to either extract important features from the image (**convolutional layers**) or manipulate the image for aggregating pixel data (**pooling layers**). Convolutional neural networks allow for the extraction of positional and rotational invariant features from an image and there are usually much better than a general deep network as the object in consideration could be placed anywhere in unseen future images and therefore these positional and rotational invariant features help locate the object with ease.

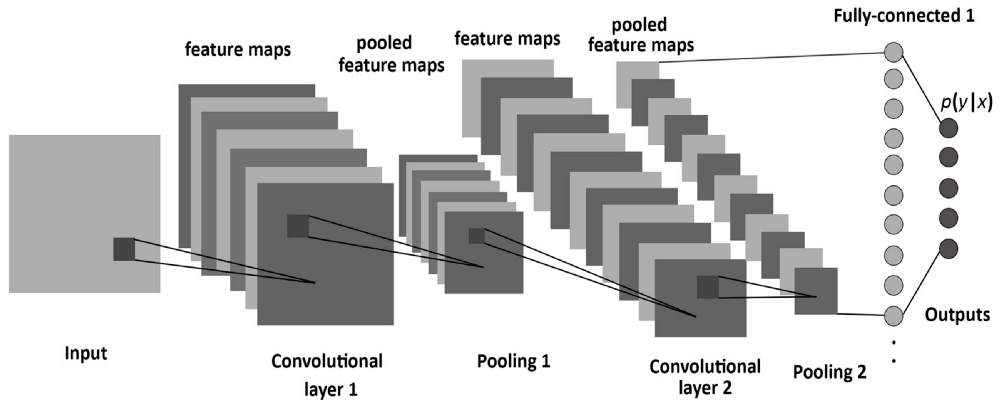


Figure 4: Convolutional Neural Network Architecture

* **Convolutional Layer**

Convolutional layers apply a convolution operation to the input and passes the result to the next layer. A convolutional filter or feature filter is passed over the whole input image and activations are recorded. Each filter/convolutional layer generates a new feature map.

* **Pooling Layer**

Pooling layers are used to aggregate pixel data to help positional and rotational invariant feature extraction.

* **Fully Connected Layer**

Fully connected layers connect every neuron in one layer to another layer.

# **Support Vector Machines**

Support Vector Machines are a type of supervised learning algorithms used in Machine Learning perform that are used to perform classification and regression problems.

Support Vector Machines try to find a linear (if possible) or non-linear hyper-plane that separates the training data and then new unseen examples are mapped onto the learned space and classified using the learned model.

The efficiency of using Support Vector Machines is that it uses a method called the **Kernel Trick** to perform non-linear classification by mapping the input vector into some higher dimensional feature space.

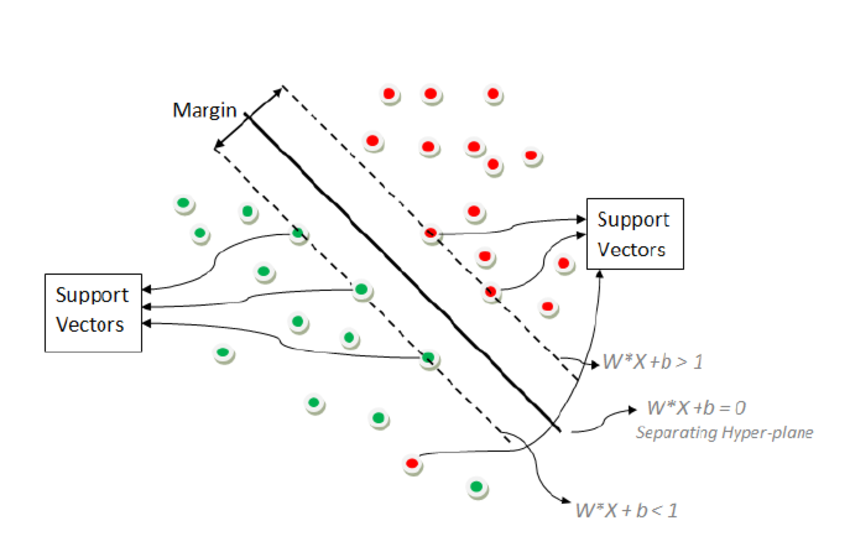


Figure 5: Support Vector Classifier

# **Multi-Output CNN Model Implementation**

In our project we use Deep Convolutional Neural Networks to extract important features from genuine signature images. To initially train our network, our dataset D consists of 24 genuine images for 55 different users representing each user enrolled in our given system, although note that the implementation presented will work with almost any number of users enrolled in a system. Since most of the organizations would have employers in the range of 10-100, training our network to work on a dataset of 55 users with only 24 images per user looks to tackle the problem of not having a large amount of data which we usually require in order to train highly efficient neural networks. We train the network using these 1320 (24 \* 55) genuine signature images with the idea that if our network is able to learn a feature space where all the 55 users are linearly separable then we can assert that our network has learned a good number of visual cues that are found in all signatures. . Table 1 and Figure 6 gives the architecture of the used Convolutional Neural Network. The network creates 5 feature maps after the input is passed after each convolutional layer with regularization of 0.01. After each convolutional layer, batch normalization is applied.

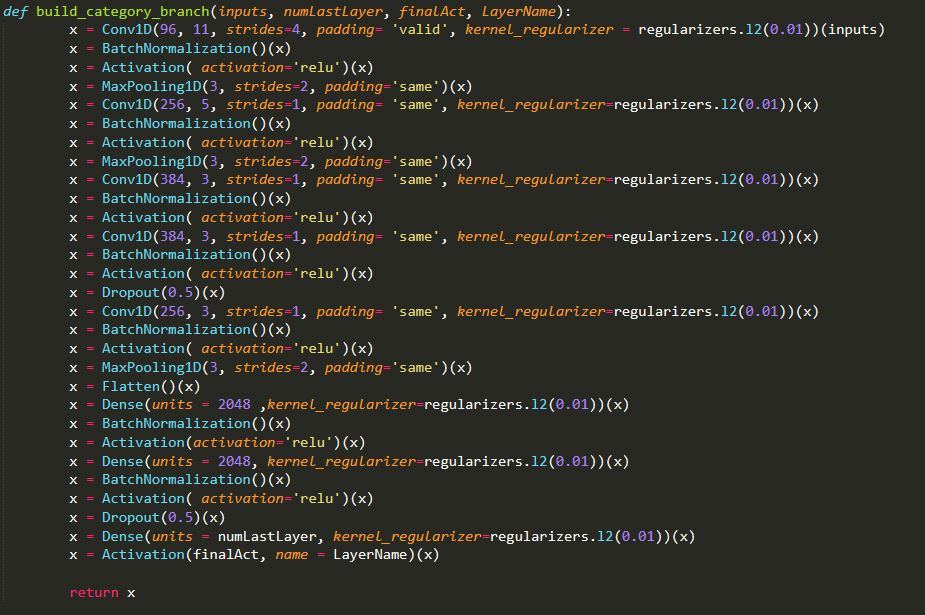


Figure 6: CNN Code Implementation

|  |  |  |  |
| --- | --- | --- | --- |
| CONV 1D | Size = 96 \* 11 | Strides = 4 | Padding = “Valid” |
| Activation Layer | Function = “relu” |  |  |
| Max Pooling Layer 1D | Size = 3 \* 3 | Strides = 2 | Padding = “Same” |
| CONV 1D | Size = 256 \* 5 | Strides = 1 | Padding = “Same” |
| Activation Layer | Function = “relu” |  |  |
| Max Pooling Layer 1D | Size = 3\* 3 | Strides = 2 | Padding = “Same” |
| CONV 1D | Size = 384 \* 3 | Strides = 1 | Padding = “Same” |
| Activation Layer | Function = “relu” |  |  |
| CONV 1D | Size = 384 \* 3 | Strides = 1 | Padding = “Same” |
| Activation Layer | Function = “relu” |  |  |
| Dropout | Rate = 0.5 |  |  |
| CONV 1D | Size = 256 \* 3 | Strides = 1 | Padding = “Same” |
| Activation Layer | Function = “relu” |  |  |
| Max Pooling Layer 1D | Size = 3 \* 3 | Strides = 2 | Padding = “Same” |
| Fully Connected Layer | Units = 2048 |  |  |
| Activation Layer | Function = “relu” |  |  |
| Fully Connected Layer | Units = 2048 |  |  |
| Activation Layer | Function = “relu” |  |  |
| Dropout | Rate = 0.5 |  |  |
| Fully Connected Layer | Units = Number of Users |  |  |
| Activation Layer | Function = “relu or sigmoid” |  |  |

Table 1: Multi-Output CNN Model Architecture

We have setup our network as a multi output network with a single input. One output called the **categoryUser** which contains one node for each enrolled in the system, is used to classify the user given the input image and the other output **categoryGenOrForg** is a single node output that only classifies whether the given input image is a **Genuine** or a **Forgery**.

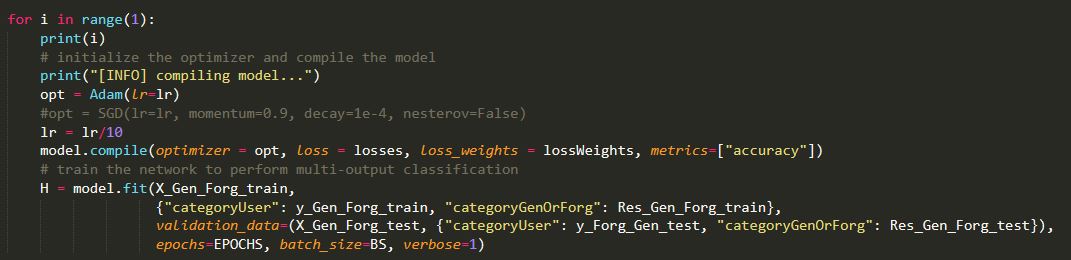


Figure 7: Model Training

Similarly, we have used two different loss functions, one for each ouput, we use the **categorical\_crossentropy** for the **categoryUser** output layer and **binary\_crossentropy**  is used on the **categoryGenOrForg** output layer. We also assign different loss weights to both of the loss functions used.

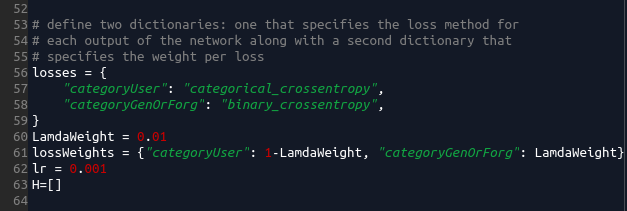


Figure 8. Dictionaries Defining Errors

**Preprocessing:** our training dataset contains images separate images for genuine and forgery for each user (2640 signature of 55 user, 24 genuine and 24 forgery signature for each user). To prepare the dataset for training, we take all the data i.e. the genuine and the forgery images with their labels. Since we have a multi output classification network with a single input, we merge the genuine and forgery images together to make the input vector of signature images where as we make separate label vectors for each output. The label vector for the **categoryUser** layer contains the user labels for each signature image and the label vector for the **categoryGenOrForg** layer is a binary vector having **1** for genuine images and **0** for the forgery images. The data is split accordingly into train, test and validation sets. The input image is first resized to 220 \* 150 and then gray scaled in figure 9 we can see the preprocessing code

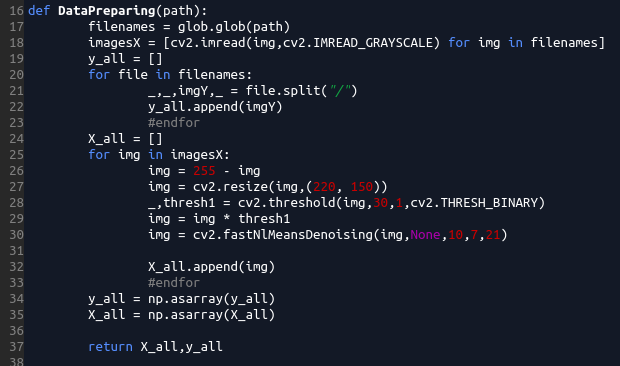


Figure 9. Preprocessing phase code

We fork two branches of the above network for each output with just the activation function and the number of nodes in the last (output) layer being different. We use the **softmax** activation function for the last layer of the **categoryUser** branch and **sigmoid** activation function for the last layer of the **categoryGenOrForg** branch.

Training is done by running the network over the dataset consisting of the genuine and the forgery images for 20 epochs for 5 iterations. The learning rate is initially set to 0.001 and decays by a rate of 1/10 after each iteration so the network is trained for 100 epochs. After running our model on dataset we saw that our model learn in first 40 epoch and after that somehow overfitting, so we retrain our model with just 3 iteration (20 epoch each) and we reach to 89% accuracy in user and 74% accuracy in type.

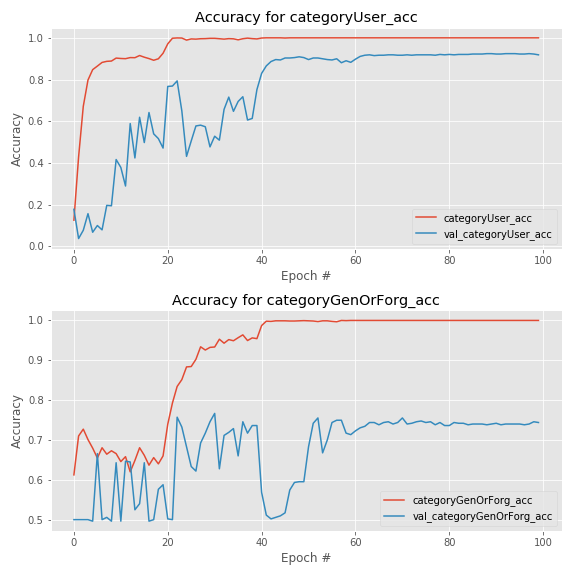


Figure 10. Accuracy of our model with 5 iteration

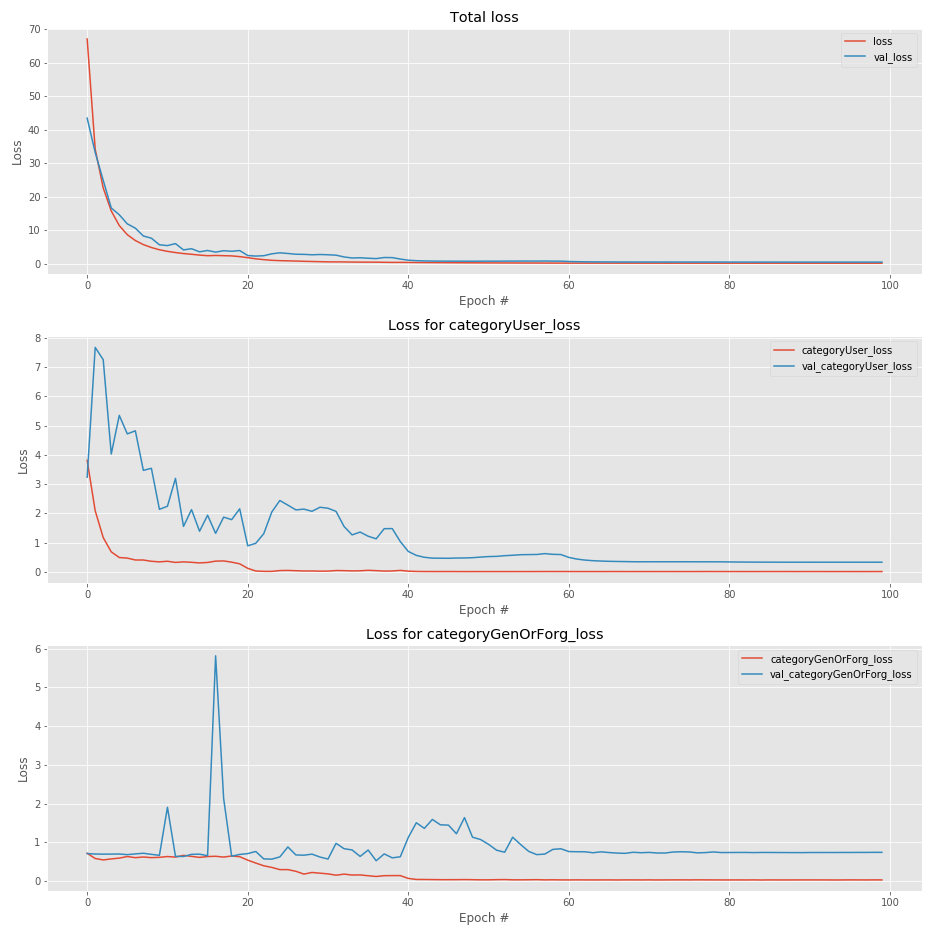


Figure 11. Loss of our model with 5 iteration

# **SVM Implementation**

After we train our model by the convolutional neural network described above with 3 iteration and get our features (2048\*1) we use them to train an SVM model using the features extracted. We configure SVC and try to plot the confusion matrix to show the distribution of TP, TN, FP and FN, we reach to 86% accuracy in user and 69% accuracy in type. By tuning our hyper parameter we couldn’t reach to better results.

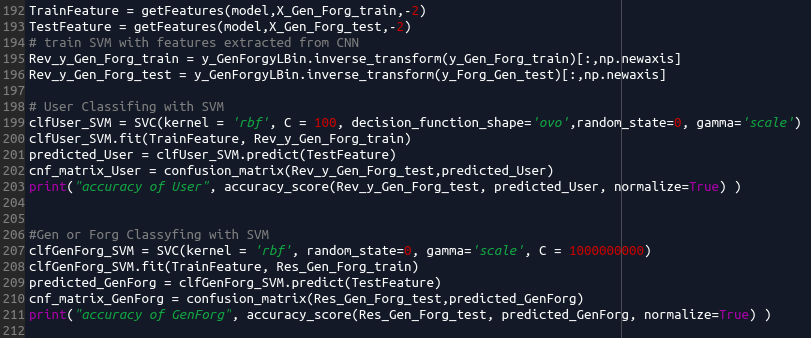


Figure 12. SVM Implementation

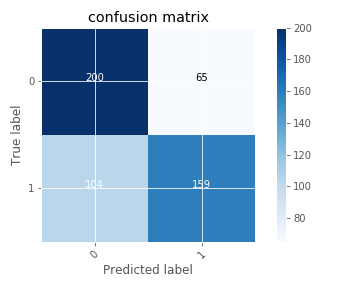


Figure 15. Confusion matrix of SVM model for genuine signature detection

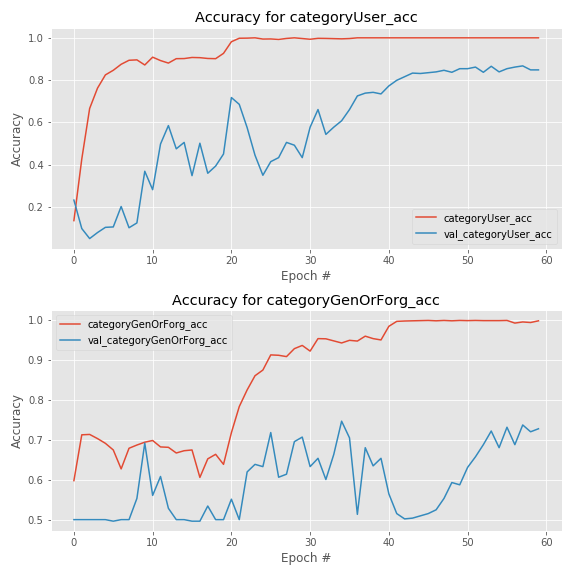


Figure 13: Accuracy Plot for 3 iteration output model

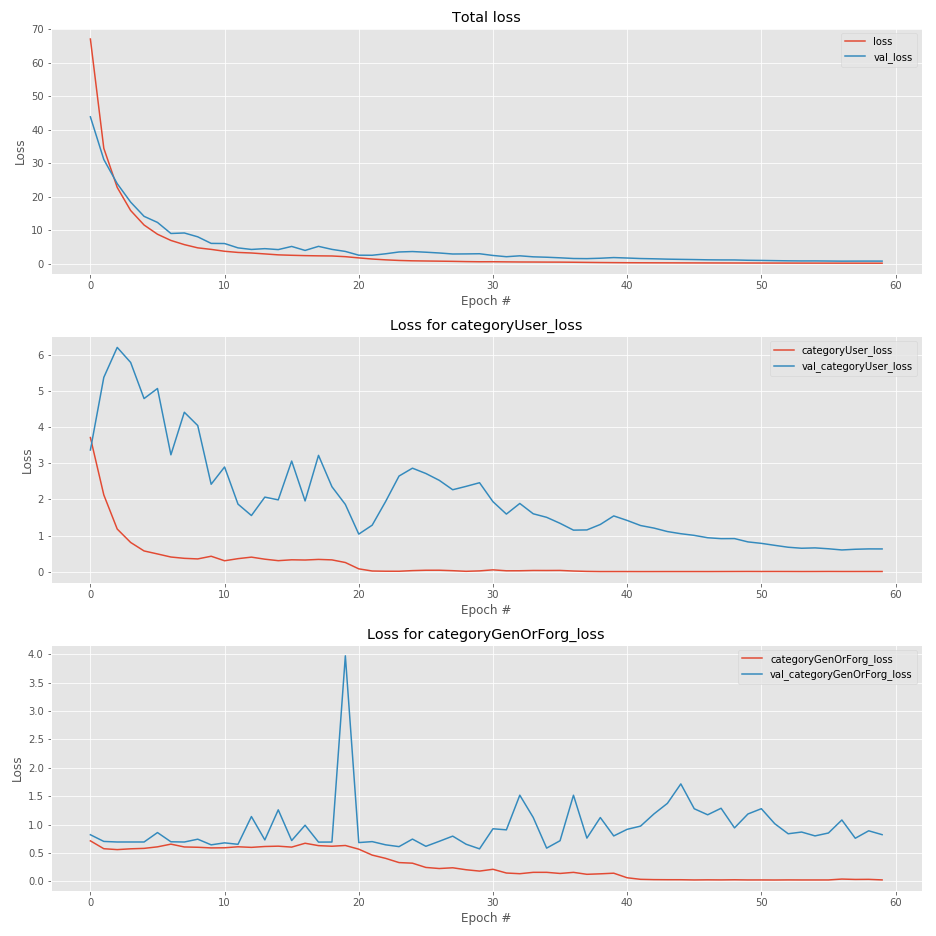


Figure 14: Loss function plot for 3 iteration loss function

# **Conclusion**

In this project we try to distinguish the genuine and forgery signatures of different user with CNN and SVM models. First we train an CNN model and try to understand useful features of each signatures and then we used softmax and sigmoid for understanding the user and type of image (genuine or forgery), we reach to 89% accuracy in user and 74% accuracy in type, Then we model the SVM based on learned features and reach to somehow same accuracy. After training with SVM and CNN we realize that our accuracy is not that much promising, so because we already tune our learning rate and optimization parameters (we use different SGD and Adam settings) we think that maybe we need to tune our regularization factor or try to make our CNN better for reaching to better results.

# **References**

1. Learning features for offline handwritten signature verification using deep convolutional neural networks <https://arxiv.org/pdf/1705.05787.pdf>
2. Code for plotting confusion matrix. <https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html>
3. Keras Documentation [https://keras.io](https://keras.io/)
4. Scikit-learn documentation [https://scikit-learn.org](https://scikit-learn.org/)
5. Numpy Documentation <http://www.numpy.org/>
6. Matplotlib Documentation <https://matplotlib.org/>
7. Convolutional Neural Network Explanation <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>
8. datasets from <https://cedar.buffalo.edu/NIJ/data/>