Dataset

The dataset we generated comprises a set of clean images and a set of augmented pictures, for a total of 83,904 samples, comprising 2,622 basic images of all characters and fonts, and 81,282 augmented pictures. These are divided into labels: Font, Character, Bold and Italics, with 11 classes for the Font label (i.e. 11 possible fonts), 94 classes for the Character label, and binary values for both the Bold and Italics labels. This subdivision results from the data-cleaning and image pre-processing performed in the *data_generator.py*, where endings and file types of the font names are removed (e.g. "_B_I" and ".ttf"), thus reducing the number of fonts from 28 to 11. The rationale behind this choice concerns the use of the binary variables Bold and Italics, whose purpose is to identify whether a character is in bold or italic format, making the storage of this information in the font names redundant.

Data Augmentation

The Data Augmentation was performed using the *keras* and *scipy* Python libraries, and by adding some extra noise through our functions contained in *transform_image.*py*.* The main augmentation step is the use of the function *ImageDataGenerator* from the *keras* library, which implements rotations, zooms, rescaling and other minor changes to the pictures, and then applies our functions as the *preprocessing_function* parameter of the *ImageDataGenerator*. The modifications of the images are defined by the functions *add_line, add_filter* and *ruined_pic* contained in the *.py* file aforementioned, and randomly add lines (vertical, horizontal or diagonal), add a random filter from the *scipy* library, and add/remove pixels, respectively. The filter used are imported from the *scipy* library and alter the images by changing the colours of their pixels, either by switching the colour of the background and that of the character, or by darkening the background and the outlines of the characters and symbols. These functions are later called in the function *modify,* which chooses with a certain probability which transformation to apply. Since our aim was to build a balanced augmented dataset, we decided to apply 31 permutations to each image, thereby increasing the pool of images to train the model with (including the same number of images for all possible fonts and characters).

Model and Performance

The convolutional neural network takes as input an array of shape (64, 64, 1), i.e. the shape of the individual images in the dataset. The first step is to perform the Zero Padding, which consists in adding a black frame around the image with a width of 3 pixels (each with value 0). The reason behind this is to avoid penalising the pixels in the centre of the image, which usually happens due to the structure of the convolution layer itself: by creating a filter which skims the image matrix, the contours of the image would tend to be analysed a lot more than the centre, thereby decreasing the precision of the neural network in recognising specific features. The subsequent six layers consist in three blocks of a Convolution and Max Pooling layer each. The former begins with 32 5x5 filter matrices with strides (2, 2) and a Max Pool with 2x2 matrices and strides (2, 2), and the latter increase the number of filters used in the convolution to 64, and decrease the dimension of the filter matrices to 3x3. The activation functions used in the convolution layers are *relu* and the *kernel_initializer* is *glorot_normal*. After the three blocks of Convolution and Max Pool we used a Flatten function, which modifies the dimension of the data in order to begin the training of the neural network model. To this end, we also added a Dropout function removing 50% of the nodes at random. The four main blocks that make up the classes of the predictions, i.e. Font, Character, Bold and Italics (though not in this order). All hidden layers of all blocks use *relu* as activation function and *glorot_normal* as *kernel_initializer.* The first block, Italics, consists of one hidden layer with 256 nodes, followed by a Dropout of 40% of these nodes, and the output layer with one node and activation *hard_sigmoid,* which proved to be faster than the *sigmoid.* The Bold block is built in an identical manner to the Italics block and immediately follows it. Then the Character block is made up of 128 nodes, with a Dropout of 40% of the nodes and an output layer using the *softmax* activation function with 94 nodes (i.e. all possible characters). Finally, the Font block is identical to the Character, with the only difference being that it uses 64 nodes in its hidden layer.

It is important to notice that the Dropout layers are not directly connected to the output layers: although they do take as input the hidden layers of each block, they are not taken as input of the output layer immediately after, but rather, they are the input of the first hidden layer of the following block.

This model had the following results of the accuracy of each class: 0.755 for Character, 0.413 for Font, 0.801 for Bold and 0.769 for Italics, making the total accuracy equal to 0.6644 on the training set. On the other hand, in the 33% of the hidden test we obtained: 0.460 for Character, 0.298 for Font, 0.739 for Bold and 0.708 for Italics, for a total of 0.517, indicating a bit of overfitting especially for the Character class.

Comparison with Previous Models

The main changes we have done to the model relate to each block of the pipeline. First of all, we added the Zero Padding layer, we increased the size of the filter matrix from 3x3 to 5x5 in the first Convolution and Max Pool block, and added two other blocks of this with the aim of detecting as many features as possible, though lowering their filter matrices to 3x3. We also added the Dropout layers in each label block rather than putting it just after the Flatten layer, and changed the number of nodes used in the hidden layers, which noticeably increase our level of accuracy on the training set and on the partial test set. Furthermore, the number of nodes were chosen in order to maximise the final accuracy and minimise the overfitting occurring in the training of the model. Finally, we decided to change the order of the output layer, as we noticed that by putting Italics as the first layer, the accuracy on both training and partial test sets visibly increased.

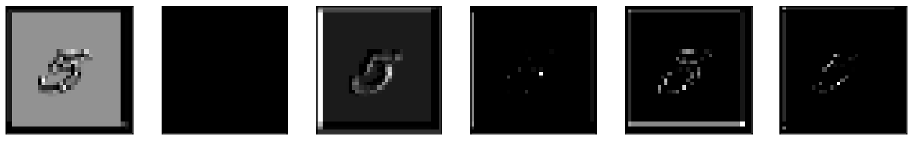
Extra Point 1: Intermediate Output Visualisations

In order to better understand the behaviour of the hidden layers of our model, we first looked at the outputs of each Convolution layer, and then we verified the behaviour of the output layers.

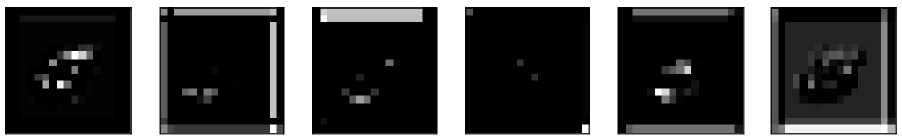
Unfortunately, since we had problems running our scripts using *matplotlib.pyplot* on the cluster, we did not manage to save good quality pictures. These are however very clear when running our code from the script, and we thus suggest you to have a look at those in order to better understand our analysis. Some of them are reported here for clarity.

Although it was not required of us to look into the output of the convolutions, we believed it would provide us with a much deeper understanding of our model. Below, are a small sample of each of the three Convolutions performed by our neural network:

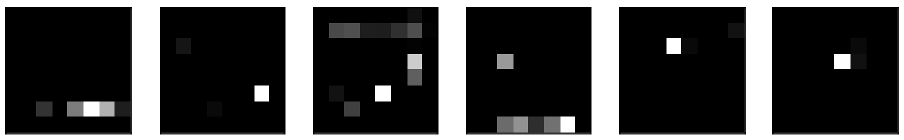
First Convolution



Second Convolution



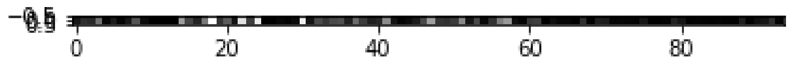
Third Convolution



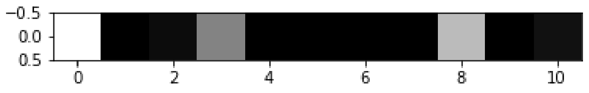
On the other hand, the (intermediate) outputs of the hidden layer assume a much different form: since these categorise the input images into all possible labels in each class, e.g. for all 11 fonts in the Font class, they are represented as a vector divided up into as many blocks as the number of types in the specific category. The colour of this blocks is in the greyscale range, with a pure white representing the final category choice of the (intermediate) output layer, and a darker image a lower probability of the input image belonging to that particular type class.

The resulting images hence vary depending on the number of types in each class: 11 for Font, 94 for Character and 1 for both Bold and Italics (since they are binary variables), implying that the output images of the Bold and Italics class are just a square either completely black or completely white, as shown below:

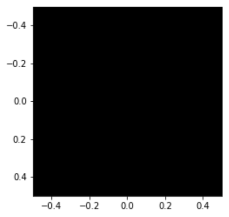
Intermediate Output: Character



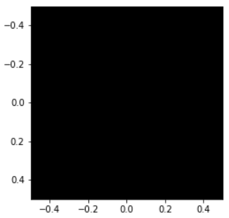
Intermediate Output: Font



Intermediate Output: Bold



Intermediate Output: Italics


−0.4
−0.2
0.0
0.2
0.4
−0.4
−0.2
0.0
0.2
0.4