Using Convolutional Neural Networks for Emoticon Classification

K. Burnik* and D. Bjelobrk Knežević**
Zagreb University of Applied Sciences, Zagreb, Croatia
* kristijanburnik@gmail.com, ** dunja.bjelobrk@tvz.hr

Abstract—The problem of automatic classification of photographs, graphics and spatial data is increasingly being addressed via advanced machine learning methods and related models like Convolutional Artificial Neural Networks (CANN). In the scope of this work, we conducted a study of application of the CANN to an example of emoticon classification.

In this paper, chosen methods are described as well as solutions for data collection and processing. Methods for training and evaluation of the model are offered. The presented results show successful application of the model.

Keywords—convolutional neural networks, classification, emoticons, image processing, computer vision

I. INTRODUCTION

Automatic classification of imagery has shown numerous applications in the fields of scientific research and is actively being applied in various industries with a particularly large impact on big data and automated image processing.

Convolutional Artificial Neural Networks have been around since the 1980s [1], although at the time considered computationally ineffective, in recent years with hardware performance improvements, introduction of task-specific hardware (GPUs and more recently TPUs) and distributed computation, CANNs have truly been revived and improved to the point of becoming state-of-the-art models for image classification [2].

In our study, we addressed the problem of recognizing rendered images of emoticons (e.g. smiley face, frowning face, winking face, etc.) using a CANN architecture and tuning it for best possible performance on our curated datasets.

We explored multiple variations of the network architecture, choices of hyperparameters, various regularization methods, techniques for artificial dataset expansion and otherwise curated the datasets to find a suitable model for the task.

The emoticon datasets were gathered from multiple sources [3], [4], including Google image search and even by generating emoticon images using a simple program we had developed for the very purpose.

As supervised models are evaluated, the resulting accuracy is inevitably a mixture of the model fitness and the quality of the dataset, hence the data collection and model searching are often two interleaving processes which also involve numerous experiments. Such is the case in our study as well. Nevertheless, we find it more comprehensive for the reader to present these processes separately.

II. CURATING THE DATASETS

The initial emoticon samples were collected from the emoji-pedia.org [3] website along with their meta data (class labels).

Although 33 classes in total were prepared, there were only 12 samples per class on average which is insufficient for typical dense and convolutional models (e.g. the MNIST dataset [5] contains 7000 samples per each decimal digit class).

Given the small ratio of samples per class to the number of classes, the samples were curated to form a dataset called COMMON_6 containing only six classes of our choosing as shown in Figure 1.

![Samples from the COMMON_6 dataset](image)

Fig. 1. Samples from the COMMON_6 dataset. Each image represents one sample of their associated emoticon class, from left to right: astonished-face, crying-face, fearful-face, slightly-frowning-face, slightly-smiling-face and winking-face. These particular samples were collected from [3] and were produced by Apple Inc.

Similar to the common six, equivalent subsets of emoticons called COMMON_3 and COMMON_4 were prepared to evaluate how the number of classes affects the model performance.

Due to the foreseen low performance of initial models evaluated on COMMON_6 (50-68% accurate), the dataset was extended with additional samples obtained via Google image search, producing the COMMON_6 EXTRA dataset which on average contains 25 samples per class.

Furthermore, to allow for emoticon detection (i.e. whether a square in an image contains an emoticon) rather than just classification (i.e. which emoticon is displayed in the square image), three additional background classes were generated as shown in figure 2. This combined dataset was appropriately named COMMON_6 EXTRA_WITH_NOISE, where the terms "noise" and "background" are used interchangeably.

![Samples from the background classes](image)

Fig. 2. Samples from the background classes. Each image represents one sample of their associated background class, from left to right: blank, circle and noise. The rationale for introducing background classes is purely based on intuition and experiments. The blank class should help with recognizing blank portions of a screenshot. The circle should amplify the differences in the face expression rather than the Mostly generic emoticon head. The noise class is to separate a meaningful set of shapes from a random patch of pixels.
A. Storing, parsing and preprocessing

Each dataset is stored in a directory structure where the class sub-folder is named after the associated label (e.g. "common-6/slightly-smiling-face/01.png") and all of the available samples of the same class reside together.

Splitting the data for training, testing and validation as well as image augmentation and preprocessing are all performed dynamically - prior to starting the training and evaluation loop.

Although this sacrifices run-time memory and initialization time (negligible compared to training time) rather than storage space, it provides more flexibility for experimenting and introduces a convenient means of selecting hyperparameters while the source data remains pure. Furthermore, caching can be utilized for subsequent runs to decrease the initialization time.

The size of the output layer of the model is directly tied to the number of classes advertised by the target dataset (i.e. by the number of sub-folders) making the model architecture implicit to the provided dataset.

B. Expanding the dataset

Given a very low number of samples per class (e.g. 25), the performance of initial models was unsurprisingly low. It proved increasingly harder to collect additional samples via image search, therefore we explored the possibility to generate emoticons with parameterized traits (e.g. size and shape of eyes, mouth, eyebrows, etc.).

![Generated samples](image)

The generated samples covered three classes of emotions: happy, sad and surprised, enough to evaluate existing models and compare the performance of the GENERATED dataset vs. the COMMON_3 equivalent (slightly-smiling-face, slightly-frowning-face and fearful-face).

It is fairly straightforward to generate simple variations of emoticons, however the actual samples obtained from [3] and other sources display more variety in style which is more difficult to reproduce (generate) using a classical approach.

Although it is out of scope of this study, we would argue that using Generative Adversarial Networks [6] to bootstrap the generative process would be worth exploring as a possible improvement.

C. Artificial expansion via augmentation

Another venue was explored which would allow the existing COMMON_6 dataset (and its expansions) to be further - artificially expanded. Although affine transformations such as translation, rotation, flipping and skewing were considered, a more novel approach of using elastic transformations [7] for image augmentation was favored.

Applying these kinds of transformations to produce image variations comes with multiple benefits. The trained model has better generalization due to the increased variance and never has to observe the original images, yet still produces a very low error on them (comparable to loss). Therefore, the unaltered training images (which are never observed during training) can instead be used as a validation set to prevent over-training the model.

Elastic transformations also produce an effect similar to the affine transformations when observed on local segments of the image (e.g. a slightly rotated eyebrow).

To the human observer, the augmented images may appear as if smudged or observed through rippled water, yet still recognizable. The intensity of the transformation should be controlled so as to preserve the traits and keep the emoticon distinguishable from other classes.

![Random elastic transformation](image)

III. CHOOSING THE CLASSIFIER

The goal of the study was to develop a convolutional model which can successfully classify at least six emoticon types and which is no less than 85% accurate on the evaluation dataset. As mentioned via introduction, the process of finding a suitable model coincides with curating the datasets to provide sufficient description for the model to learn from the data.

A number of iterations in which either the model or data was adjusted to increase performance were conducted. Within the constrained scope of our study, the experiments have shown that using two convolutional groups and three dense groups while training on six emoticon and three background classes produce the best overall results.
A convolutional group is comprised of the convolutional layer, a max-pooling layer and an optional leaky rectified linear unit layer (commonly known as leaky ReLU) as depicted by figure 6.

Although the convolutional groups also have an optional dropout layer, our experiments and recent studies [8] have shown that they should be excluded.

A fully-connected (i.e. dense) group computes the matrix product of the inputs and the neural weights with added biases. Two optional layers can be applied: a leaky ReLU for activation and a dropout layer for regularization.

The boundary between the convolutional groups and the dense groups is occupied by a simple “unrolling” operation which reshapes the multidimensional outputs of the last convolutional layer to a one-dimensional vector.

The very last group of the network omits the ReLU layer and contains exactly N outputs (i.e. logits) where N implies the number of distinct classes.

IV. Training

The training program was designed with experimentation in mind - the choice of dataset, model architecture and other hyperparameters are all configurable either through a file or directly using command line arguments.

Implicit parameters, such as the output layer size which is tied to the dataset class number are stored in the model configuration for running post-training predictions and producing evaluation reports.

For comprehensiveness, the parameters have been categorized into three groups. Firstly, the ones which affect the model architecture (i.e. model selection) as shown in table I. The image size affects the input layer while the dataset selection affects the output layer. Optional layers for ReLU, max-pooling and dropout are controlled via binary switches for convolutional (C) and dense (D) groups:

\[ C_i, D_j ∈ \{0, 1\} : D_1D_2D_3C_2C_1 \]  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>–dataset</td>
<td>Dataset selection - implicitly affects the trained model’s architecture.</td>
<td>COMMON_3 COMMON_4 COMMON_6...</td>
</tr>
<tr>
<td>–image-size</td>
<td>Target image size - the square side length</td>
<td>Integer, e.g. 24</td>
</tr>
<tr>
<td>–num-channels</td>
<td>Number of color channels for the input image conversion.</td>
<td>1 or 3</td>
</tr>
<tr>
<td>–filter-size</td>
<td>Size of the filters in the convolutional layers</td>
<td>Integer, e.g. 8</td>
</tr>
<tr>
<td>–pooling</td>
<td>Switches for max-pooling in convolutional layers.</td>
<td>Integer, e.g. 3 (00011₂) for both convolutions</td>
</tr>
<tr>
<td>–relu</td>
<td>Switches for ReLU activations.</td>
<td>Integer, e.g. 12 (01100₂) for the first two dense layers</td>
</tr>
<tr>
<td>–dropout</td>
<td>Switches for dropout layers</td>
<td>Integer, e.g. 4 (00100₂) for after the first dense layer</td>
</tr>
<tr>
<td>–dropout-rate</td>
<td>Rate for the dropout layers</td>
<td>(0.0, 1.0), e.g. 0.7</td>
</tr>
</tbody>
</table>

Secondly, the data transformations prior to training are controlled by arguments described in table II. The image augmentation using elastic transformations and the train/test split are both tied to the random number generator (RNG), which can be explicitly set for reproducibility.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>–random-seed</td>
<td>Initial value for the RNG - affects data splitting and other pseudo-random preprocessing.</td>
<td>Integer, e.g. 2719411</td>
</tr>
<tr>
<td>–split-ratio</td>
<td>Ratio for the train/test data split.</td>
<td>(0.0, 1.0), e.g. 0.7</td>
</tr>
<tr>
<td>–expansion-factor</td>
<td>Number of augmented images to generate via elastic transformations, when set to 0 originals are used.</td>
<td>Non-negative integer, e.g. 100</td>
</tr>
</tbody>
</table>

Finally, the training process is configurable with training hyperparameters detailed in table III. These parameters are exposed by the momentum optimizer which was selected for training.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>–batch-size</td>
<td>Number of samples for each training step</td>
<td>Integer, e.g. 20</td>
</tr>
<tr>
<td>–momentum</td>
<td>Training hyperparameter</td>
<td>Small positive decimal, e.g. 0.6</td>
</tr>
<tr>
<td>–learning-rate</td>
<td>Training hyperparameter</td>
<td>Small positive decimal, e.g. 0.01</td>
</tr>
<tr>
<td>–training-steps</td>
<td>Number of steps per epoch for periodic evaluation</td>
<td>Integer, e.g. 100</td>
</tr>
</tbody>
</table>
A. The training process

The train/test split occurs prior to the optional train set expansion and the test images are only observed during evaluation while the model is frozen (i.e. backpropagation may not occur).

Per convention, an epoch implies that the model has observed the entire training dataset. Due to the cumulative effect of backpropagation, the model weights are readjusted to optimize the loss after each observed batch.

The output layer is evaluated for loss using the sparse softmax cross-entropy between the logits (measured outputs) and the ground truth labels. This implies that the output classes are mutually exclusive as documented in [9].

The model evaluation occurs after each 100 epochs and reports the training error (loss), validation set error and the test set error. If the training is run on an expanded dataset using only the augmented images, then (as described in II-C) the validation set is comprised of the original images from the training portion of the train/test split.

After running initial training experiments, the stopping conditions have been determined empirically. The training process will run at least 500 epochs and afterwards (by checking every 100 epochs) will stop if either:

1) the validation set accuracy has surpassed 99.9%
2) the exact same validation set accuracy was subsequently reported more than twice
3) the training ran for more than 5000 epochs.

B. Tracking the performance

For each training session, the model performance metrics were tracked in real time using TensorBoard [10]. Naturally, the main expectation was loss decrease (fig. 7) and accuracy increase (fig. 8) over time, providing confidence that the model is capable of converging.

Some extra metrics for debugging are also reported, such as the individual terms of the loss function. For example, the regularization terms are often tracked to ensure that over-fitting is prevented.

Even more insight is provided with visualization - firstly, the input images which are subjected to preprocessing are displayed to ensure that the network receives valid inputs - and secondly, the convolutional filters can also be visualized as depicted in figure 9.

The reported performance curves, although automatically smoothed out, appear jagged due to a relatively small number of measurements. This occurs since the metrics are computed only every 100 epochs so as to speed up training.

V. Deep evaluation and reporting

The early stages of our work have produced more than a few low performing models. Although intuition suggested that the datasets were just too small to expect any significant accuracy, a deeper insight was required into the model behavior once they were expanded and some improvement has been observed.

Tracking the training process and reported accuracies provided only high level black-box observations of the model performance. Those metrics gave no specifics on the failures nor provided the insight into where improvements can be made.

Deeper insight into a model’s behavior was achieved through a reporting tool we had developed.

The confusion matrices provided answers to which classes were often mistaken, accompanied by a display of individual samples for which the classifier made the error, as shown in fig. 10. Both representations were applied to the context of the training set, the testing set and the entire dataset respectively.
The addition of background classes was shown as the leading factor which improved the model accuracy. For comparison, a similar model trained on COMMON_6 EXTRA achieved 31 of 44 matches (70.45%) while the denormalized accuracy (counting all background class errors) of an equivalent model on the 6+4 dataset is averaging over 80%.

Similar accuracies have been reported for expansion factors of 50 and 100 and no extra improvements were observed for larger expansions.

Deeper models such as ResNet-152 [11] were also evaluated, averaging 70% test accuracy.

VII. CONCLUSION

Our study has successfully applied a convolutional neural network for emotion classification and showed significant results (accuracy over 90%) even while being impaired by small datasets.

Using elastic transformations for artificial dataset expansion and introducing background (noise) classes has substantially improved the overall dataset size, variance and consequently the model accuracy.

Experiments suggest that further improvements of CANN models could be achieved primarily through gathering more data and exploring methods for generating emoticons such as using Generative Adversarial Networks.

REFERENCES


VI. THE BEST PERFORMING MODEL

Our best performing model achieved 91.24% accuracy on the testing set of COMMON_6 EXTRA_WITH_NOISE, classifying 125 of 137 (never before observed) images correctly. The model matched 315 of 320 (98.44%) images from the unaltered training images, hence for the entire dataset it successfully recognized 440 of 457 (96.28%) samples.

Although the training set and entire dataset accuracies are usually not considered for performance evaluation, in this case it is useful to report them since the model was trained only on augmented images and never observed the original unaltered samples.

By running a number of experiments through configurable parameters, the best performing model’s architecture and the training hyperparameters have been found and are described in table IV.

As this model achieved the goal of surpassing the 85% accuracy mark, further experiments with hyperparameter optimization were no longer pursued.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset</td>
<td>COMMON_6 EXTRA_WITH_NOISE</td>
</tr>
<tr>
<td>image-size</td>
<td>36</td>
</tr>
<tr>
<td>num-channels</td>
<td>3</td>
</tr>
<tr>
<td>filter-size</td>
<td>8</td>
</tr>
<tr>
<td>pooling</td>
<td>1 (000012)</td>
</tr>
<tr>
<td>relu</td>
<td>12 (011002)</td>
</tr>
<tr>
<td>dropout-rate</td>
<td>0.5</td>
</tr>
<tr>
<td>random-seed</td>
<td>271941</td>
</tr>
<tr>
<td>split-ratio</td>
<td>0.7</td>
</tr>
<tr>
<td>expansion-factor</td>
<td>150</td>
</tr>
<tr>
<td>batch-size</td>
<td>20</td>
</tr>
<tr>
<td>momentum</td>
<td>0.6</td>
</tr>
<tr>
<td>learning-rate</td>
<td>0.01</td>
</tr>
<tr>
<td>training-steps</td>
<td>100</td>
</tr>
</tbody>
</table>

TABLE IV
PARAMETERS OF THE BEST PERFORMING MODEL