Classifiying Transmission Electron Microscopy Virus Textures

Alan Do-Omri 260532985
Kian Kenyon-Dean 260564475
Genevieve Fried 260564432

December 11, 2015

Abstract

On a daily basis doctors are tasked with determining the identity of viruses from blood samples. Even with advanced microscopy techniques, viruses can be difficult to discern by the expert human eye. Unfortunately, Misclassifying can have grave implications, not only for the patients in question but also for those who come into contact with him or her. Fortunately, virus classification is an image classification problem, and machine learning methods present exceedingly high accuracy rates for image classification tasks and don't require expert knowledge to do so. Here we present a successful application of convolutional neural networks to the problem of virus classification. Using the Virus Texture Dataset from Uppsala University, we classify viruses samples into one of 15 categories and compare its performance to previous work done on the same task.

1. Introduction

Identifying viral pathogens is a continuous endeavor medical professionals essential to public health.

2. Dataset

3. Methods

Convolutional Neural Networks (CNN) are biologically inspired variants of Multi-Layered Perceptrons that can encode non-linear features with atleast one hidden layer, and are specifically architected for image classification. Image processing with a Neural Network is computationally infeasible given how quickly parameters scale. CNN's on the other hand constrain their architecture to the nature of images. A neuron in the nth layer is not fully connected to all neurons before it, rather,

They work as follows.

They're constrained to work with images, which specifically oriented towards image processi Image processing with a fully connected Neural Network

Unlike a fully connectd Neural Network, CNN's a Neural Network, image classification is infeasibly computationally expensive, and CNN's CNN's are made up of several layers: input, convolutional, pooling and fully connected.

4. Related Work

In the work of Kylberg et al. [4], texture analysis is performed on a dataset composed of 22 different virus samples and the resulting feature vector is fed into a Random Forest classifier. The authors first compare the performance of different texture analysers, such as Local Binary Patterns, Radial Density Profile and their respective variants before proceeding to detail their results. We will briefly explain their feature extractors and results here.

4.1. Local Binary Pattern (LBP)

Local Binary Pattern works such that, given an image, for each pixel p_i in it we sample n equally-spaced

points on the circle of radius r with center p_i – an example of sampling is shown in figure 1 – and construct a vector $v(p_i)$ such that its ith entry is a 1 if the ith sampled pixel has a value bigger than p_i and 0 otherwise.

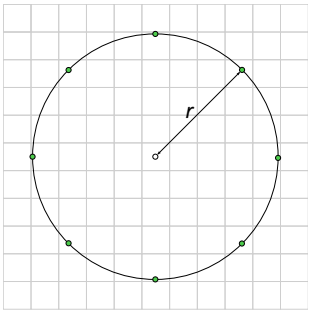


Figure 1: Example of LBP sampling: The green points are the neighbouring sample points at distance r from the central white point. In this case, we are sampling with $n = 8$ neighbour points.

A sequence of 0s and 1s are constructed $v(p_i)$ to form a binary number v_{p_i}, hence the name. Once we have the v_{p_i} for all pixels, we construct a histogram counting the number of appearances of each value v_{p_i}. The histogram forms the feature vector associated with the given image. Kylberg et al. [4] denote this feature extraction method by $\text{LBP}_{n,r}$, where n is the number of sampled points and r is the radius, as described above. The resulting histogram is represented in a vector of counts with 2^n elements.

Rotational Invariant LBP In order to reduce the size of the feature vector, Kylberg et al. [4] mention a modification of LBP in the following sense: instead of creating v_{p_i} by interpreting the vector $v(p_i)$ and using v_{p_i} as is, rotate the number v_{p_i} bitwise until you get the smallest possible number. For example, the number 110 becomes 011. They name this technique the rotational invariant and denote it with $\text{LBP}^{\text{ri}}_{n,r}$.

Uniform LBP To further reduce the size of the histogram, Kylberg et. al also restrict the values of v_{p_i} to only numbers that have 2 or less transitions from 0 to 1 or from 1 to 0, and they call this variant *uniform binary patterns with at most 2 spatial transitions*, denoted $\text{LBP}^{\text{u2}}_{n,r}$. For example, 01010 has 2 transitions from 0 to 1 and 2 transitions from 1 to 0 which makes 4 transitions in total. This version then $\text{LBP}^{\text{u2}}_{n,r}$ does not count the number 01010. On the other hand, 00111 has only 1 transition from 0 to 1 so it is accepted.

Gaussian Filtered LBP Finally, Mäenpää and Pietikäinen [5] talk about an extension to the LBP that is also used in the work of Kylberg et al. [4], which consists of sampling neighbours about a central pixel at different radii. Instead of sampling these points equidistantly in a circle, the authors sample according to a Gaussian distribution. Figure 3 shows an example of such a sampling. Kylberg et al. [4] denote it as $\text{LBPF}^{\text{ri}}_{n_1,r_1} + {}^{\text{ri}}_{n_2,r_2} + \cdots + {}^{\text{ri}}_{n_j,r_j}$ for the different numbers of sample points $n_1, n_2, \cdots, n_j$ and different radii distances $r_1, r_2, \cdots, r_j$.

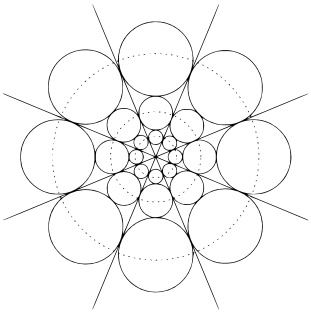


Figure 2: Example of LBPF sampling: In the simple LBP, the points would be sampled at equal distances around the dotted circles for one radius as in figure 1 but with this extension, the points are sampled according to a Gaussian distribution inside the solid black circles at multiple radii. In this picture, the number of neighbours also vary with the radii. This image comes from Mäenpää and Pietikäinen [5].

4.2. Radial Density Profile (RDP)

This method is a way to get the mean intensity in a ring around each pixel in the image. Kylberg et al. [4] defines the *radial mean intensity* f around a center pixel q_c as

$$f(q_c, r) = \frac{1}{|N|} \sum_{q \in N} I(q)$$

where $I(q)$ is the pixel value for pixel q and $N = \left\{q : \|q - q_c\|_2 \in \left(r - \frac{1}{2}, r + \frac{1}{2}\right]\right\}$ is the set of pixels in a ring around q_c of width 1 at distance r from q_c. Figure 3 shows an example of what the set N may look like. Following the notation from Kylberg et al. [4], let $\bar{f}_{q_c}$ be the mean of the set $\{f(q_c, i)\}_{i=1,\cdots,n}$. Then they define the radial density profile for that pixel q_c as $\text{RDP}_n = \left[f(q_c, 1) - \bar{f}_{q_c}, f(q_c, 2) - \bar{f}_{q_c}, \cdots, f(q_c, n) - \bar{f}_{q_c}\right]$.

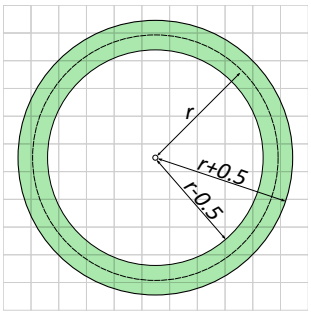


Figure 3: Example of RDP sampling: The green zone is the set of sampled pixels for a given radius r.

It is not explained how the pixel features are combined to make the feature vector for the whole image but we would imagine that the feature vector for the whole image is either a concatenation of the pixel features or the concatenation of the mean of each pixel feature.

Fourier RDP One variation Kylberg et al. [4] introduced is applying the same RDP technique on the image after it undergoes a Fourier transform and looking at it with respect to a base e logarithmic scale. They denote it with FRDP_n.

4.3. Results

The first thing to note is that Kylberg et al. [4] made modifications to the images' scales. In one case, one pixel in the image corresponded to 1 nanometer, they refer to that scale as *fixed scale*. In the other case, the radius of the virus is set to take 20 pixels, they refer to that scale as *object scale*. The classifier used was a Random forest with 100 trees (they tried up to 200 with no significant improvement with respect to 100 trees). The different textures Kylberg et al. [4] used were among

1. $\text{LBP}^{\text{ri}}_{8,2}$
2. $\text{LBP}^{\text{riu2}}_{8,2}$
3. $\text{LBPF}^{\text{ri}}_{8,1} + ^{\text{ri}}_{8,2.4} + ^{\text{ri}}_{8,5.4}$
4. $\text{LBPF}^{\text{riu2}}_{8,1} + ^{\text{riu2}}_{8,2.4} + ^{\text{riu2}}_{8,5.4}$
5. RDP_{20}
6. FRDP_{20}

All values are chosen after a grid search. Figure 4 shows their results with respect to the six aforementioned texture extractors and on both fixed and object scale images. They conclude that $\text{LBPF}^{\text{ri}}_{8,1}+^{\text{ri}}_{8,2.4}+^{\text{ri}}_{8,5.4}$ performed the best in fixed scale (median 21% error) whereas FRDP_{20} performed the best in object scale (median 22% error). They also noticed that combining these two texture extractors, they could get an even better result (median 13% error).

5. Baseline Classification Approach

We tested 4 different classifiers using algorithms from the scikit-learn Pedregosa et al. [6] machine learning library. These baseline classifiers provide results that help to reveal the dataset's complexity. We trained and tested these classifiers over the rotated dataset. Not surprisingly, the SVM with an rbf kernel performed the best since it allows projection of the pixel feature space to a higher dimension, thus making it capable of capturing aspects of the data that cannot be observed by the other classifiers. These classifiers clearly have poor results, but they provide a good baseline understanding of the dataset, showing that the image pixel space is not easily separable and requires a stronger non-linear function approximator in order to be properly understood. See the appendix for the confusion matrices of the results.

6. Neural Networks Approach

In order to establish a baseline, we combine ideas from Kylberg et al. [4] and ourselves by attempting to use a neural network with the LBP features. Our neural networks are built using the Lasagne and Theano [1, 2] libraries. In our experiments, we will not rescale the images and we will use $\text{LBP}_{8,2}$, an implementation given by the *Mahotas* computer vision library Coelho [3]. We are not scaling the images because we are later comparing to convolutional neural networks which will not used rescaled images. The values of $8, 2$ come from the best values as checked by Kylberg et al. [4]. *Mahotas*' implementation of LBP is done so that the feature vector is of smaller dimension than the usual dimension of a histogram with 2^N values where N is the number of sampled points. This is achieved by some optimization on their end.

6.1. Results

To normalize the data, we chose to divide the entire dataset by 1.1 times the maximum value of a feature in the training set in order to be more certain that no features in the normalized test set would have a value greater than 1. In all cases, we are using a learning rate of 0.005 and an L2 regularization weight

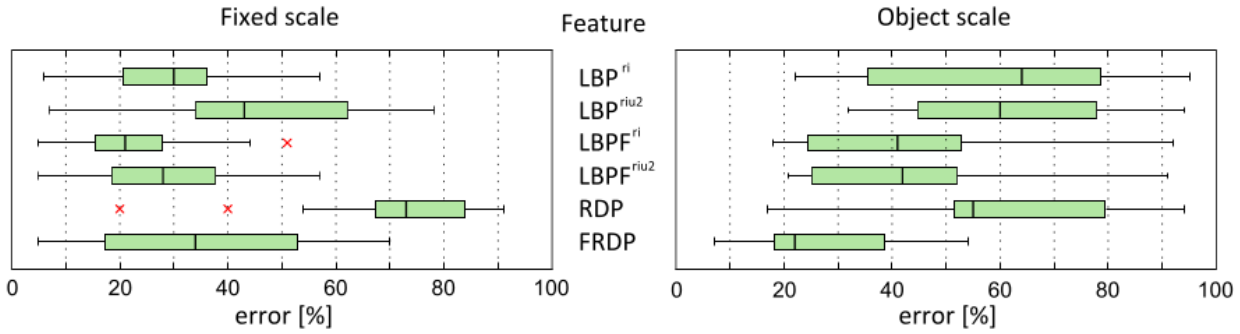


Figure 4: This figure comes from Kylberg et al. [4]. The classification errors are shown for a Random Forest classifier using the 6 different texture extractors as described in section 4.3. The boxes' vertical lines represent the median and the red × represent outliers that are at least 1.5 times the size of the box away from it. The error bars are from the lower to the upper quartile.

of 0.0001, both arbitrarily chosen and the last layer was a 15 units softmax layer. The dataset described in section 2 is shuffled and the training is done with a batch size of 16 by stochastic gradient descent. All images pertaining to the results such as the confusion matrices for the validation and testing set and the learning curves can be found in the Appendix: the first neural network results are found in figures 6 and 7; the second neural network results are found in figures 8 and 9; the third nerual network results are found in figures 10 and 11. The table below shows the results obtained on the test set. The first column represent, in order, the number of units in each of the hidden layers.

Hidden Layers	**Test Error**
256 units	99%
256 and 256 units	52.22 %
256, 128 and 64 units	51.94 %

6.2. Discussion

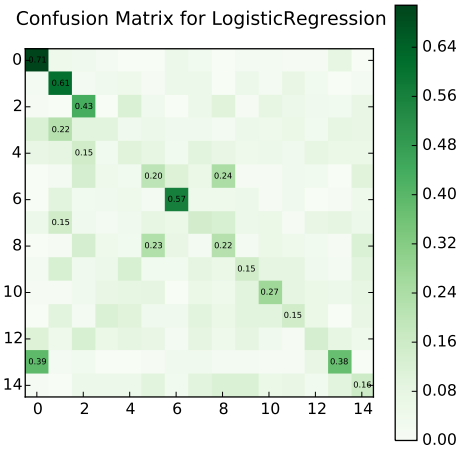
We notice that the neural network with one hidden layer took the longest time to converge whereas the one with three hidden layers took the shortest time. On the other hand, the network with three hidden layers overfitted the quickest. This is to be expected since it has many more parameters than the network with one layer. We also notice that the network with one hidden layer has the biggest error and we think this is because the network does not have enough neurons to learn properly. The three hidden layers network has too many neurons and is overfitting but we think its results can be improved by adding Gaussian noise to neurons and dropping out some of them. This is why the two hidden layers network performs the best.

Compared to the work of Kylberg et al. [4], we can see from Figure 4 that the LBP^{ri} on fixed scale performed at about 30 % error whereas on the object scale it performed at about 63 %. Since we didn't perform any rescaling of the images, we expect the error to be within that interval. Finally, *Mahotas*' implementation of LBP does not necessarily match the implementation of LBP^{ri} so this can induce some error as well.

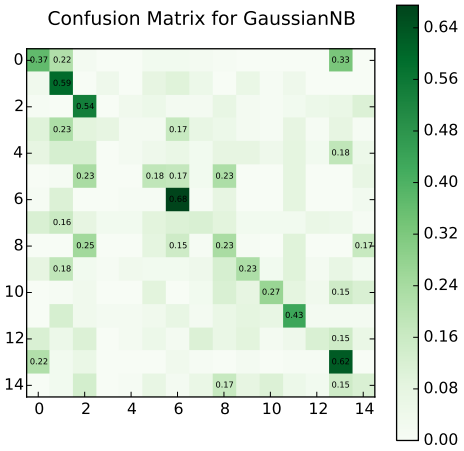
Appendix

A. Baseline Classification Results

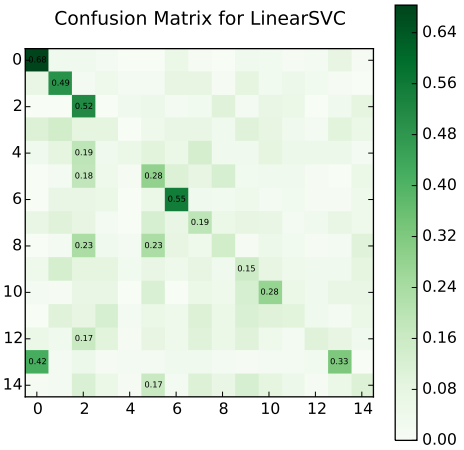
B. Neural Network Trained on LBP - Results



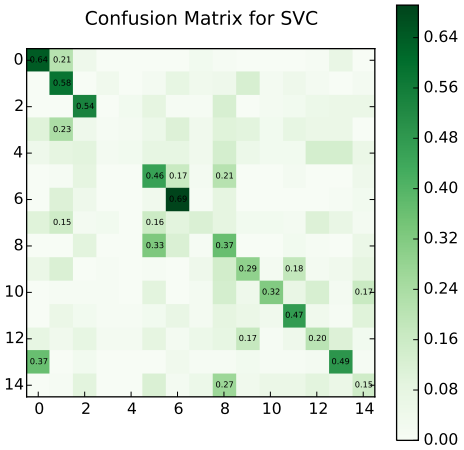
(a) Logistic Regression Confusion Matrix Results



(b) Gaussian Naive Bayes Confusion Matrix Results

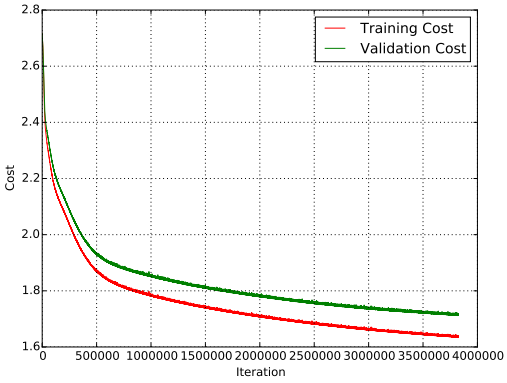


(c) Linear SVM Confusion Matrix Results

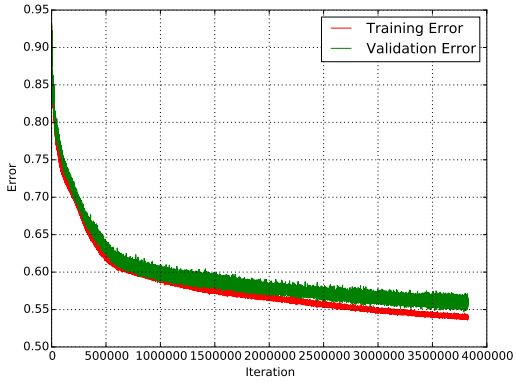


(d) SVM Confusion Matrix Results

Figure 5: Confusion Matrices for the different baseline classifiers.

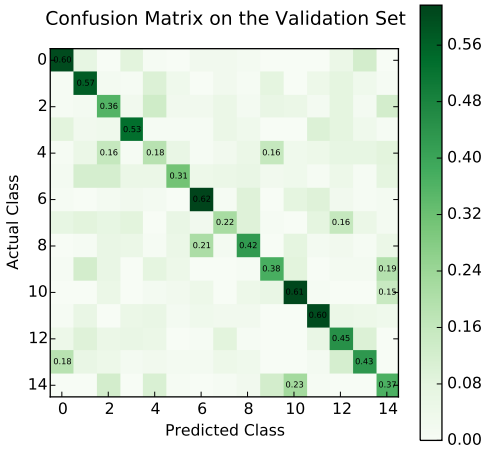


(a) Training versus Validation Cost

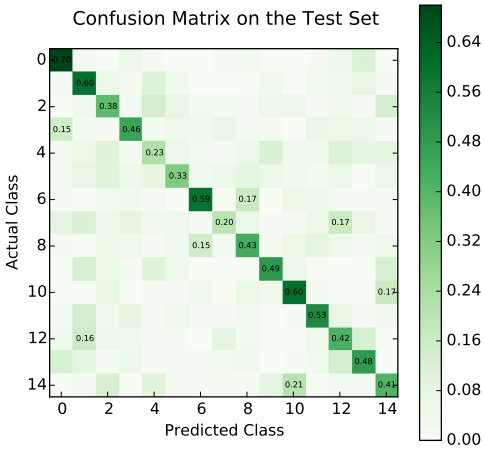


(b) Training versus Validation Error

Figure 6: Learning curves for a feed-forward neural network with one hidden layer of 256 units trained on LBP.

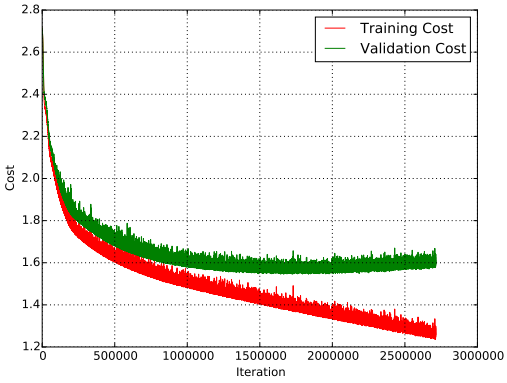


(a) Confusion Matrix on the Validation Set

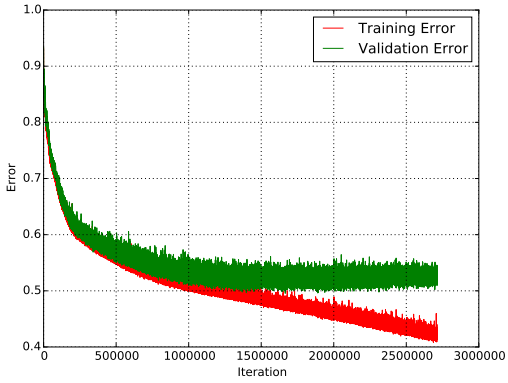


(b) Confusion Matrix on the Test Set

Figure 7: Confusion Matrices for a feed-forward neural network with one hidden layer of 256 unit trained on LBP.

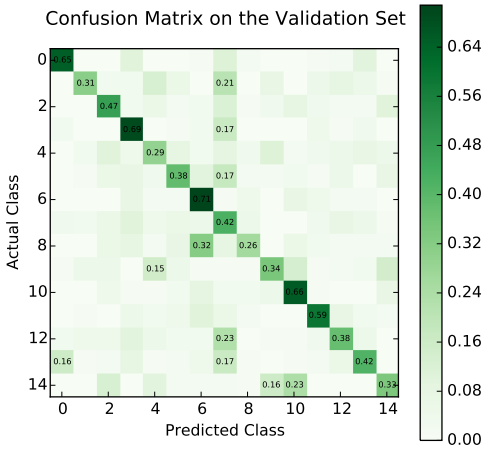


(a) Training versus Validation Cost

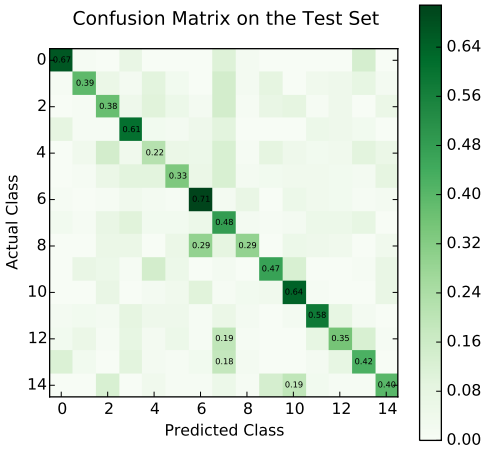


(b) Training versus Validation Error

Figure 8: Learning curves for a feed-forward neural network with two hidden layers of 256 units each trained on LBP.

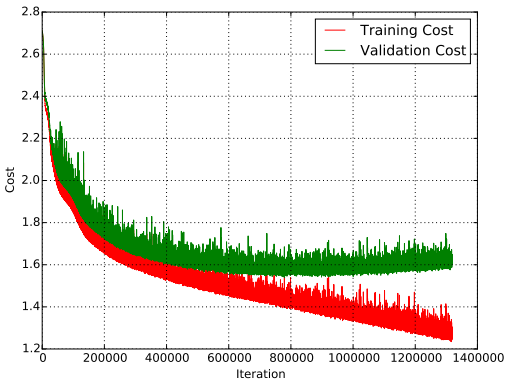


(a) Confusion Matrix on the Validation Set

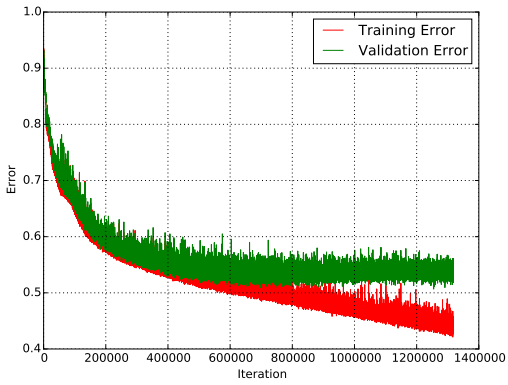


(b) Confusion Matrix on the Test Set

Figure 9: Confusion Matrices for a feed-forward neural network with two hidden layers of 256 units each trained on LBP.

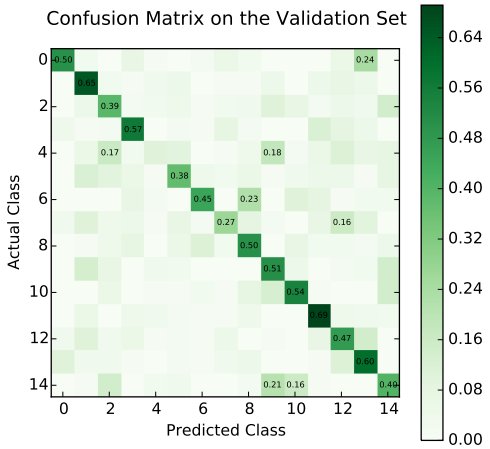


(a) Training versus Validation Cost

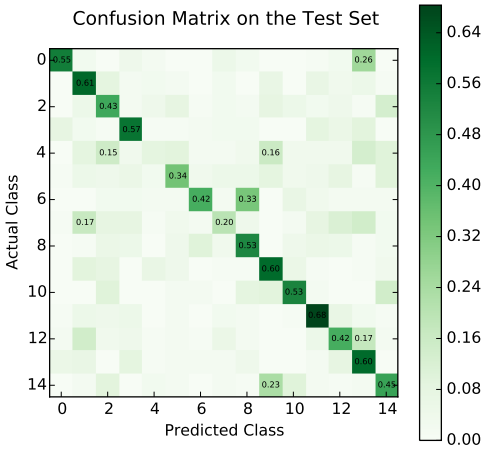


(b) Training versus Validation Error

Figure 10: Learning curves for a feed-forward neural network with three hidden layers of 256, 128, 64 units trained on LBP.



(a) Confusion Matrix on the Validation Set



(b) Confusion Matrix on the Test Set

Figure 11: Confusion Matrices for a feed-forward neural network with three hidden layers of 256, 128, 64 units trained on LBP.

We hereby state that all the work presented in this report is that of the authors.

Alan wrote the sections *Related Work* and *Neural Network Approach*. Alan programmed the feed-forward neural network in `emerald.py` and extracted the Local Binary Patterns from the dataset in `purify_dataset.py`.

Kian created the rotated dataset and wrote its section *Transformed Dataset*, programmed the baseline learners and wrote its section *Baseline Results* and also wrote the *Conclusion*.

Genevieve wrote the *Introduction* and the CNN results and discussion.

Kian and Genevieve both collaborated to writing the CNN with help from Alan's `emerald.py`.

References

[1] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[2] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[3] Luis Pedro Coelho. Mahotas: Open source software for scriptable computer vision. *arXiv preprint arXiv:1211.4907*, 2012.

[4] Gustaf Kylberg, Mats Uppström, and Ida-Maria Sintorn. Virus texture analysis using local binary patterns and radial density profiles. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pages 573–580. Springer Berlin Heidelberg, 2011.

[5] Topi Mäenpää and Matti Pietikäinen. Multi-scale binary patterns for texture analysis. In *Image Analysis*, pages 885–892. Springer, 2003.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.