Image classification using Convolutional Neural Networks

Mirna Baksa
FIB - Barcelona School of Informatics
Polytechnic University of Catalonia
Email: mirna.baksa@est.fib.upc.edu

Abstract *- This report presents results obtained from experiments with image classification using Convolutional Neural Networks. The experiments were modeled in Keras. The objective was to get the best results possible (in terms of accuracy) while training a CNN on a chosen dataset.*

1 Introduction

Artificial neural networks have found many application in a wide range of disciplines. This report will not explain basic concepts in depth, but will rather focus on the practical aspects of using neural networks.

The main focus will be on using a Convolutional Neural Network for Image Classification. The problem of Image Classification is defined as follows: given a set of images where each image is labeled with a single category, predict the categories for a new set of test images (that have not been seen before) and measure the accuracy of the predictions. Convolutional Neural Networks deal well with this problem due to their structure (which is inspired by the visual cortex in animals).

The purpose of this report is to make coherent use of deep learning techniques and do a proper analysis of obtained results. A series of experimens will be made on a chosen dataset, and there will be an analysis of the impact of different techniques and algorithms on the set problem. The experiments will be modeled in the Keras framework.

2 Dataset

The dataset chosen for the experiments in this report is the Describable Textures Dataset (DTD). DTD is a collection of annotated textural images in the wild, open and available to the community for research purposes.

The motivation for the creation of this dataset comes from the strong power of the human visual system. The task of identifying textures to the most precise details and collecting an immense amount of information in just a few seconds is quite simple to humans. The intention of the authors was to reproduce this capability in machines.

2.1 Structure

The dataset consists of 5640 images categorized in 47 categories. Each category consists of 120 images ranging between 300x300 and 640x640 pixels. At least 90% of each image surface represents the category attributed. Authors of the dataset obtained the images from Google and Flickr by entering image attributes and related terms as search queries. The images were annotated using Amazon Mechanical Turk.

The authors provide 10 splits of the data they used in their work. They propose splitting the data in three equal parts - train, validation and test, with 40 images per class for each split. They provide ground truth annotation for both key and joint attributes.

2.2 Examples

Some simple examples of textures are shown in Figure 1.

Some textures are not so trivial to classify and they could be annotated with more than one attribute (as shown in Figure 2). This is obviously still easy for humans, but poses a problem for a machine and makes it harder to distinguish between classes of textures.

(a) Dotted

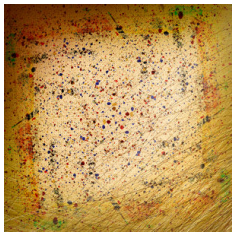
(b) Knitted

(c) Banded

(d) Swirly

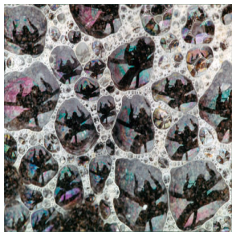
Fig. 1: Examples of simple textures

(a) Zigzagged, braided, fibrous, woven



(b) Sprinkled, marbled, stained

(c) Blotchy, freckled



(d) Blotchy, bubbly, porous

Fig. 2: Examples of more complicated textures

3 Code structure

Code used in the experiments can be found at a GitHub repository [2].

Each experiment is modeled by two files:

1. **exp-#.py**
   # being the experiment number. This source file loads the train and validation data, does the preprocessing needed for that experiment, defines the network structure and begins the training. After training, the weights are stored as a .json file and accuracy and loss plots for both train and validation files are exported to a .pdf file.
2. **validate.py**
   This file loads the network from the previously mentioned .json file, loads the test data and evaluates the model on that data. Metrics are also calculated here. This all could be done in the first file, but it was decided to separate the process in two files so the memory limit on the server is not exceeded. This file is the same for all experiments.

In the repository a file called **splitter.py** can be found. This was used to separate the dataset to train, test and validation folders according to the splits defined by the dataset authors. This is a pretty straightforward process and will not be explained further.

4 Experiments

Seing that this is the first report written in the field of Deep Learning, the decision was to take a simple approach. The problem being the classification of diferrent textures in the dataset, the route taken was to train a Convolutional Neural Network to this task as efficiently and accurately as possible. This task is not easy nor trivial, and it will be shown that different problems arise in the process.

At first, only basic image preprocessing will be used. The images will only be downsized to the smallest - recall from the dataset description that the smallest dimension is 300x300 pixels. Later on, the experiments will try to use data augmentation to get more info from the dataset and make the training easier and more robust.

Each experiment will train the data in batches, for a fixed number of epochs (hopefully an overfit at that point). Even though the authors suggested a 40 - 40 - 40 images for the train - validation - test split, here a compromise needed to be made. 40 images is a little data for the training of a Convolutional Neural network, so the data was split in 80 - 20 - 20 ratio by moving 20 images from both the validation and test sets to the training set. 80 images is still less data than hoped for, but there are some techniques that make up for the lack of data.

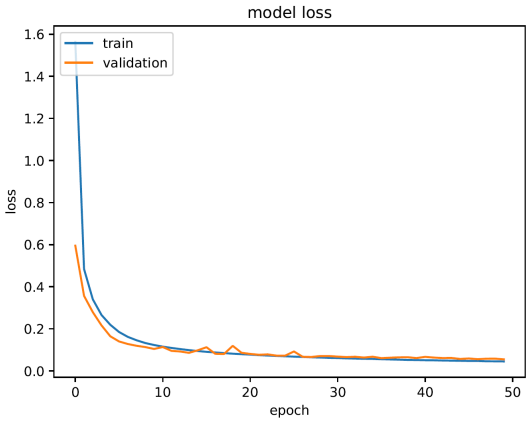
For this report, it was decided not to change the loss function and optimizer throughout experiments, but to put the focus on network structure and additional tools such as data augmentation, dropout etc. Therefore, in each experiment we use categorical cross-entropy as the loss function and stohastic gradient descent as the optimizer. Also, the output layer activation function will always be the softmax function. By fixing this parameters we hope that the focus is on the network itself.

4.1 Experiment #1 - a comparison to MNIST

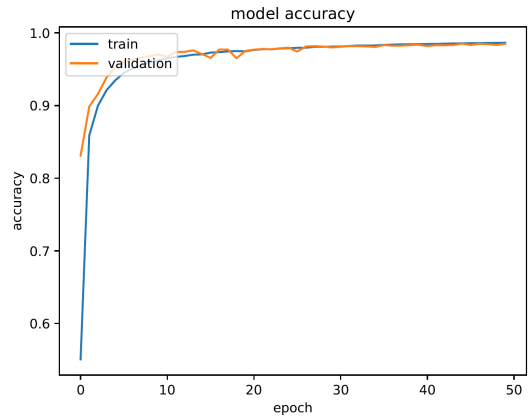
First, an extremely simple network was trained on the dataset just to see the results. This experiment was made in order to compare the behaviour of the same network on the MNIST dataset and on the DTD dataset used in this report.

Network structure was taken from the example code from the Deep Learning course [1]. The network consists of two 2D convolutional layers, the first one of dimensionality 64 and the second of 32. After each convolutional layer a max pooling layer was inserted. The end of the network consisted of a set of two fully connected layers, the first one of dimensionality 16, and the last one specific to the dataset (the output layer must match the number of classes - this was the only thing changed in the source code of the example).

As expected, the network did well on the MNIST dataset. As it can be seen in figure 3, satisfactory results were achieved around the 15th epoch. Despite that, the training continued for 50 epochs in total so the results on the DTD dataset were easier to compare. Ideally, this results are desirable - the validation curve approximately follows the training curve in both accuracy and loss plots, which means we have been successful in avoiding overfitting the training set. Some metrics are shown in table 1.



(a) Loss



(b) Accuracy

Fig. 3: Training results on MNIST

Table 1: Metrics of experiment 1 on MNIST

	precision	recall	f1-score	support
avg/total	0.98	0.98	0.98	10000

The results on DTD are expectedly worse (as shown in 4). The dataset is far more complex and such a simple network overfits on the training data but achieves inferior results on the validation set. The model also did not do well on the test set, with the test loss of 11.17 and test accuracy of only 6%. Some other metrics are shown in table 4.

Table 2: Metrics of experiment 1 on DTD

	precision	recall	f1-score	support
avg/total	0.02	0.02	0.02	940

4.2 Experiment #2 - a ”smarter” approach

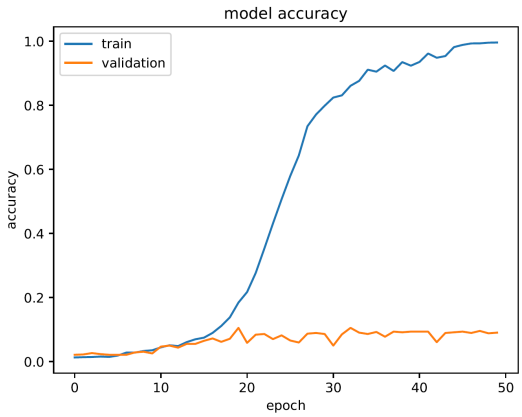
After the behaviour of a simple network on the DTD dataset was shown, the next step was to try to achieve satisfactory results on the DTD dataset.

A network structure used had to be adjusted to the problem (the dataset) - i.e. it had to be more complex then the structure in experiment 1. This all sounded very straightforward and logical in theory, but proved to be more difficult in practice. What was changed was e.g. the number of convolution layers, the number of fully connected layers, the dimensionality of the layers, number of epochs etc.

After quite a few iterations of the good old try - and - fail method, most of the results were similar to those in figure 5, if not worse. The network used to generate graphs in figure 5 was of the following structure: three convolutional layers of dimensionality 64, each followed by a Max Pooling layer. The fully connected part of the network consisted of a layer of dimensionality 128 and an output layer of dimensionality 47.

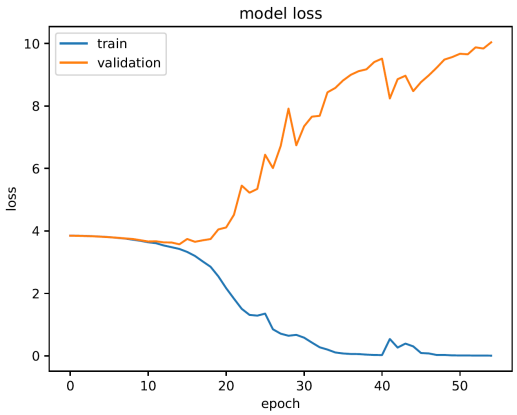
It is very obvious that these experiments were not very successful. The network was still overfitting on the

(a) Loss

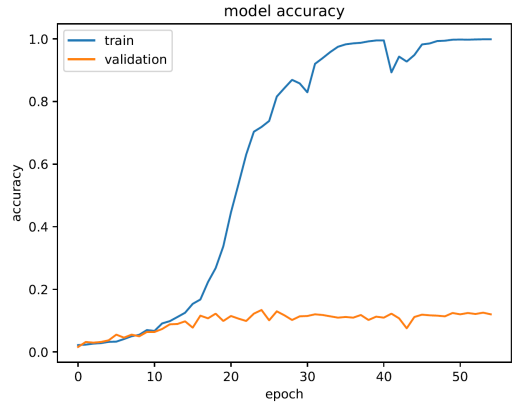


(b) Accuracy

Fig. 4: Experiment 1 results on DTD



(a) Loss



(b) Accuracy

Fig. 5: Experiment 2 results on DTD

training set as it did in Experiment #1, the only difference being that it took a lot more time for the training to finish due to the complexity of the network. The validation set accuracy was a little better than in the previous experiment, but none of the iterations achieved test set accuracy bigger than 15%, which is not a satisfactory result. All in all, the conclusion of this experiment was that the approach indeed needed to be smarter - it was time to pull out the ”big guns”.

4.3 Experiment #3 - data augmentation

It was already mentioned that the dataset is relatively small. With a 120 images for each class and 47 classes in total, it was reasonable to presume that this was the reason for poor perfomance. Luckily, there are mechanisms of dealing with lack of data, since this is actually a common problem.

An interesting example of the problem is CANDLE [3] - CANcer Distributed Learning Environment [1] - a project which focused on building a scalable deep neural network to solve problems such as understanding the mechanisms that lead to cancer, predict drug response and determine optimal cancer treatment strategies. Their published dataset consists of only 1000 clinical reports, due to the privacy and delicacy of such reports (patient personal data, medical history etc.). This is a real life example of the lack of data for training a model which is quite more important and sensitive than our problem of classifying textures.

To get back on track with the report, hopefully data augmentation will help to get more information for the network to train on. To get more data, minor alterations such as flips, translations or rotations of images can be applied. The neural network would consider those images as distinct due to its property called invariance - Convolutional Neural Networks are invariant to viewpoint, size, translation or illumination.

To get more data on our dataset, in experiment #3 we applied next transformations: rotation, vertical and horizontal translations, shearing, zooming and horizon-

[1] This project was analysed in another class of mine and it is here mentioned simply as an example. I have no associations with it whatsoever.

tal flip defined as seen in snippet 1. The network used in this experiment was of the same structure as the network in experiment #2, so the effect of data augmentation can be more transparent - if the network performance approves, we know that it is because of the augmenation and not the changed network structure. The one thing that was added to the network was a dropout layer (with 50% dropout rate) after the first fully connected layer - this should also help in preventing overfitting.

Listing 1: Augmentation

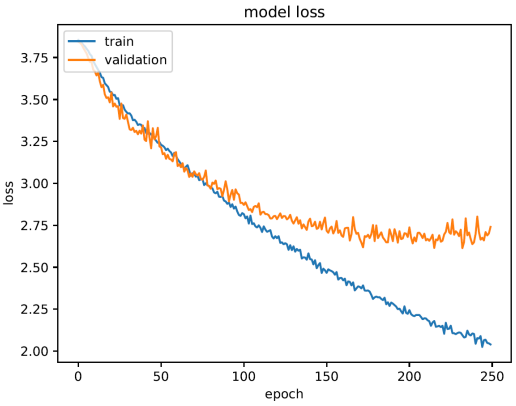
```
ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```

The results were not as successful as hoped for, but it was the inexperience that led to believing that data augmentation will solve all the problems with the dataset. What can be seen from figure 6, the network stopped overfitting on the training set, achieving only the accuracy of approximately 50% on the training set after 250 epochs. Performance on the validation set improved - in the end the network had accuracy of approximately 30%. The 250 epochs took almost 5 hours, so the training was stopped there - but it can be seen from the graphs that the validation set loss function stopped dropping somewhere around the 150th epoch, so it is doubtful that training the network further would lead to better results.

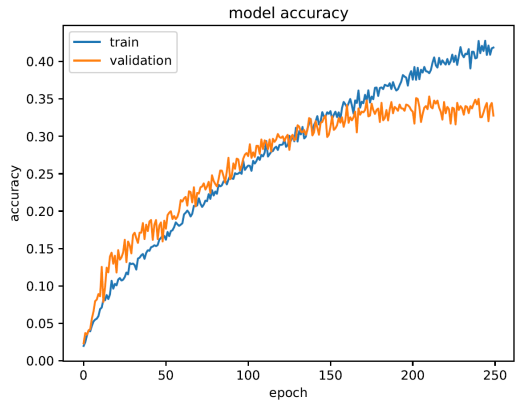
Since this was more successful than the previous experiments, it was decided to repeat the experiment with data augmentation but with a bigger network. The convolutional layers now had dimensionality of 128 and the dense layer had dimensionality 256. With this it was hoped that the network had bigger capacity and that it will be able to store more informations about the features of the textures.

Although we got a 10% more accurate network on the validation set (almost 40%), from the graph in figure 7 it can be seen that the model loss stopped dropping around the 150th epoch and that any further training was in vain.

Lastly, an experiment was again repeated but now an additional convolutional layer was added resulting in 4 convolutional + max pooling layers in total. The



(a) Loss



(b) Accuracy

Fig. 6: Experiment 3 results on DTD

added layer was also of dimensionality 128. By this repetition we can see if there will be any performance change when adding additional layers to the network. Results can be seen in figure 8.

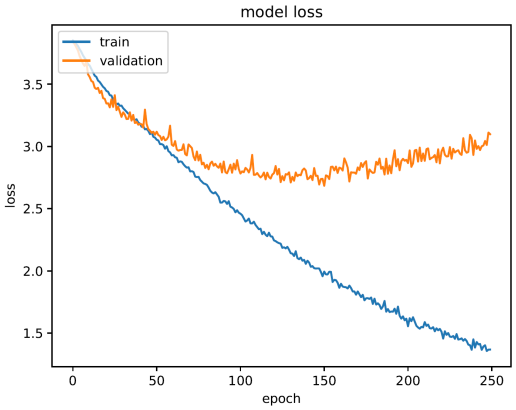
From the graphs we conclude that there was not a significant change in performance of the network to which an additional layer was added. Validation set accuracy remained below 40%, with the loss starting to rise during the last 100 epochs.

Let's look at some additional metrics for the last sub-experiment. This metrics were obtained on the test set.

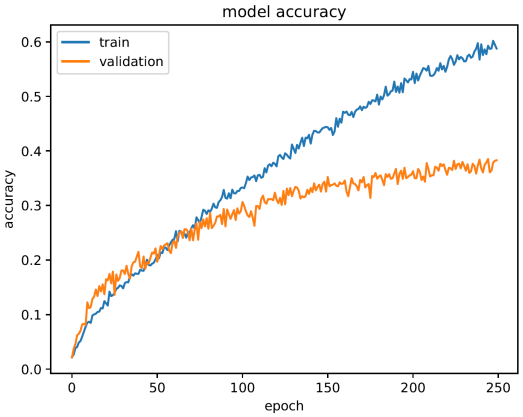
Table 3: Metrics of experiment 3 on DTD

	precision	recall	f1-score	support
avg/total	0.02	0.02	0.02	940

Again, the same metric values as in experiment #2.

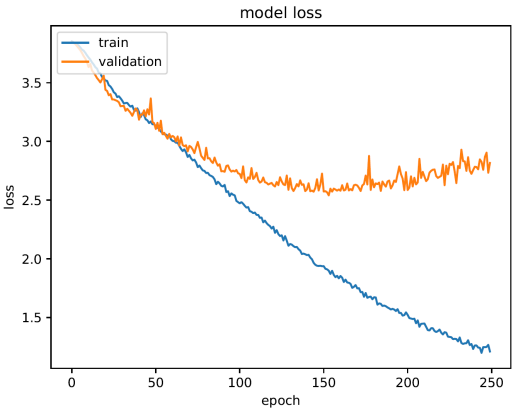


(a) Loss

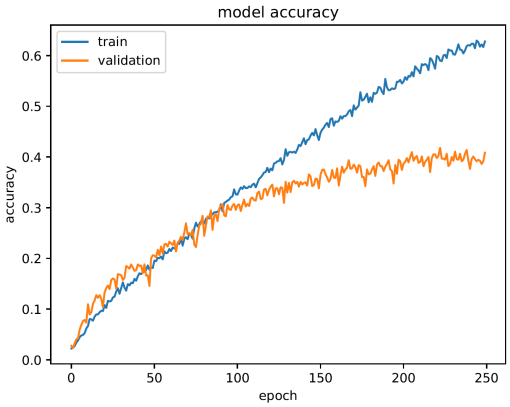


(b) Accuracy

Fig. 7: Experiment 3.1 results on DTD



(a) Loss



(b) Accuracy

Fig. 8: Experiment 3.2 results on DTD

Test accuracy achieved here was around 43%. Let's look at the metrics for some particular classes, just to see if there are some classes where the network was more successful. The metrics shown in figure 9 are only for the classes where the network achieved a metric greater then zero. It is hard to draw any reasonable conclusions from these metrics, as all the metrics have a value lesser than 15% which is not very good.

Note that in this experiment the network was not trained until overfit because it would take too much time and resources, but we can get a quite good idea of network behaviour later on.

All this led to the conclusion that data augmentation did help in some way, but there is still room for improvement.

	precision	recall	f1-score	support
banded	0.05	0.05	0.05	20
blotchy	0.04	0.05	0.05	20
bubbly	0.06	0.10	0.08	20
cobwebbed	0.06	0.05	0.05	20
cracked	0.04	0.05	0.04	20
gauzy	0.04	0.05	0.04	20
grooved	0.05	0.05	0.05	20
interlaced	0.13	0.10	0.11	20
lined	0.08	0.05	0.06	20
porous	0.03	0.05	0.04	20
smeared	0.04	0.05	0.05	20
spiralled	0.08	0.10	0.09	20
sprinkled	0.07	0.10	0.09	20
striped	0.08	0.10	0.09	20
zigzagged	0.03	0.05	0.04	20

Fig. 9: Experiment 3.2 metrics

4.4 Experiment #4

Experiments 1 through 3 are quite simple, but unfortunately none of the other experiments that were ran achieved better results. All in all, an estimate of 40 experiments were ran on this dataset, trying to find the right structure to fit our dataset. Since none of the architectures and combinations of less/more layers, less/-more neurons in fully connected or convolutional layers achieved significantly better results, as a last resort the inspiration was taken from an existing example.

Most of the examples found online that deal with image classification show how to do so on simple datasets such as MNIST or CIFAR, where it is not very difficult to achieve good accuracy. Surely there are ex-

amples of dealing with difficult datasets published as scientific articles and papers but a lot of time has to be put in to fully understand the work done in such publications.

A good example was found online where 1191 images of Pokemon needed to be classified in 5 categories [4]. After reading the article, a similar architecure was applied to the DTD dataset to see what results we get.

Figure 10 shows samples from the Pokemon dataset. Each class had around 230 images, and as we can see the dataset is diverse, including illustrations, movie/TV show stills, action figures, toys, etc. The network structure is quite extensive and can be found at link [5]. They achieved 96.84% accuracy on the training set and an amazing 97.07% accuracy on the testing set (11.

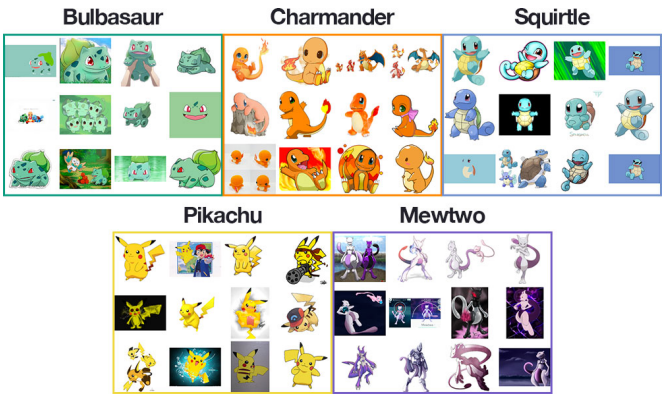


Fig. 10: Pokemon dataset

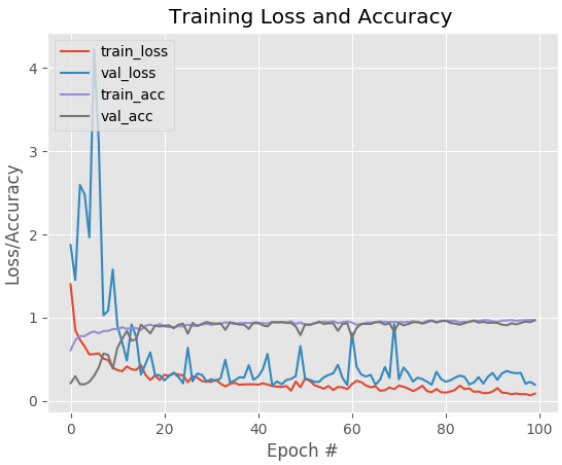


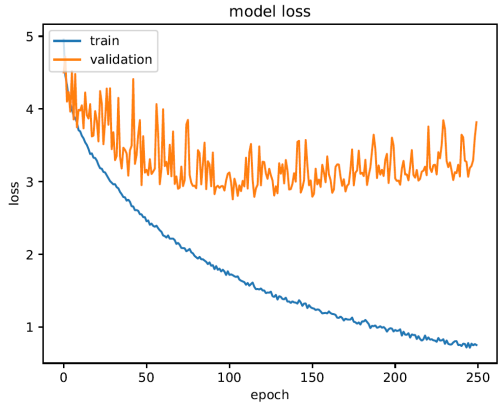
Fig. 11: Training and validation loss/accuracy plot for the Pokemon dataset

Reading this article it was realized that the network structures used in all our experiments were extensively too simple. For only five classes (remember, DTD has 47 classes) the network model consisted of several building blocks with combinations of layers that I did not even know were possible to combine. Furthermore, a quote from the article confirmed the DTD datasets problem:

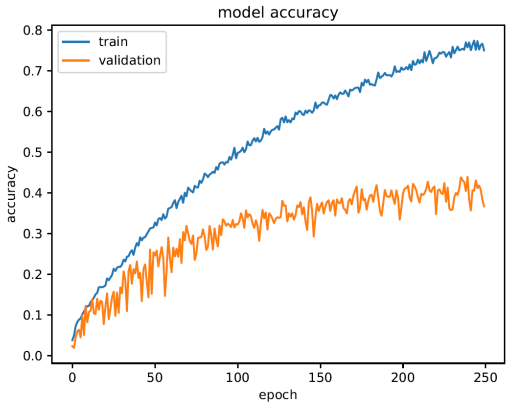
"If you're looking to train a classifier to recognize more Pokemon for a bigger Pokedex, you'll need additional training images for each class. Ideally, your goal should be to have ***500-1,000 images per class*** *you wish to recognize."*

The conclusion after reading this article was that our 80 images per class for 47 classes was indeed the main reason why our networks have been underperforming. Of course, the network structure had to be more complex than the ones used so far.

In this last experiment the network structure from the article was replicated and applied to the DTD dataset. The results can be seen in figure 12.



(a) Loss



(b) Accuracy

Fig. 12: Experiment 4 results on DTD

Again, accuracy around 40% on the validation set, the model did not yet overfit but since the 250 epoch took about 6 hours the experiment was not ran any further. Even though this network did not do significantly better than the one in experiment #3, my opinion is that this experiment is worth looking further into some other time, since it used different combinations of layers, dimensionalities, even including BatchNormalization layers which were not used in previous experiments.

Table 4: Metrics of experiment 4 on DTD

	precision	recall	f1-score	support
avg/total	0.03	0.03	0.03	940

5 Short Discussion

The main thing I learned from this report is how much I yet have to learn. In my opinion, deep learning is a field where experience plays an important role. An experienced researcher would have known from the beggining how challenging the dataset was and would have scaled the problem accordingly. I feel it is important to have an instinct, a vague feeling of what is right or wrong for a certain problem - this kind of instinct comes with experience and knowledge and time, all of which I did not have. Of course, this is not an excuse and I should have done even more research before even beggining to run any experiments. Also, if I was doing the same thing again I would structure the experiments differently and try to focus on some deep learning concepts in depth instead of focusing on network accuracy. The last thing I feel sorry for is not beeing able to analyse network metrics (precision, recall, f1 or maybe some introduce some new measures) since my network was underperforming and all metrics were equally bad.

6 Conclusion

All in all this report contains 4 experiments. First experiment shows the comparison of the same network on MNIST and DTD datasets. The second experiment tried to extend network structure from experiment #1 in order to adjust to the DTD dataset. The third experiment introduces data augmentation to help with the small dimenstionality of the DTD dataset. The fourth experiment replicated an article found online and compared results on the DTD dataset versus the dataset used in the article.

In summation the results were not satisfactory - no accuracy greater than 40% was achieved on the validation set, while we could easily achieve overfitting on the training set. Two main reasons for the poor performance are the dimensionality of the dataset and the my inexperience because of which I did not predict well how complicated the problem really was.

6.1 Extensions

A lot more can be done with this dataset. The first thing would probably be to try to extend experiment #4, adjust the network to the DTD dataset and see all the ways different layers can be combined in order to get higher performance. Furthermore, the report did not use regularizers in any of the experiments and it would be interesting to see how regularization would affect network performance. Lastly, there were a lot of factors in this report that were kept fixed - activation functions, batch size, optimizers, loss function etc, and it would certainly be interesting to explore how different factors have different effect on network performance.

References

[1] MNIST CNN Example, https://github.com/UPC-MAI-DL/UPC-MAI-DL.github.io/blob/master/_codes/1.FNN-CNN/mnist_cnn_example.py

[2] Report codes, https://github.com/mirnabaksa/Deep-Learning/tree/master/CNN

[3] CANDLE https://candle.cels.anl.gov/

[4] Keras and Convolutional Neural Networks https://www.pyimagesearch.com/2018/04/16/keras-and-convolutional-neural-networks-cnn

[5] Pokemon dataset network structure https://www.pyimagesearch.com/wp-content/uploads/2018/04/smallervggnet_model.png