Deep Learning for Image Classification: Scientific Glassware

Timothy Mitchell[1] **TMITCHELL8@LUC.EDU**

[1]Department of Mathematics and Statistics, Loyola University of Chicago, Chicago, IL

Abstract

Image classification is a ubiquitous task in machine learning. The best-known method, convolutional neural networks (CNNs), has been shown to deliver state-of-the art results on a variety of benchmarks. This paper described a CNN classifier trained on >10,000 images of scientific glassware and controls. The network architecture, runtime, choice of parameters, and validation performance are discussed. We highlight future directions for research and optimization and discuss some of the challenges of deploying deep learning models on real-world commercial data sets.

Keywords: deep learning, image classification, neural networks, optimization, computer vision

1. Introduction

In machine learning and artificial intelligence, computer vision refers to the broad process of training computers to extract high-level information from digital images. A key example is image classification, in which computers are trained to classify images of different objects. In the simplest case, classes are mutually exclusive and each image captures exactly 1 class.

As recently as 15 years ago, classifying images of cats and dogs, for example, was a cutting-edge problem. In fact, Microsoft researchers designed and published a CAPTCHA, a test that is difficult for robots to solve, using cat and dog images. “Barring a major advance in machine vision,” the authors wrote, “we expect computers will have no better than a 1/54,000 chance of solving it.”

Nevertheless, the CAPTCHA was solved later that year. In “Machine learning attacks against the Asirra CAPTCHA,” Golle and colleagues achieved 82% accuracy on the problem using an ensemble of support vector machines trained on derived features encoding color and texture.

Since that time, CNNs have dominated in image classification competitions. Their innate ability to extract features, without any feature engineering on the part of the user, makes them an attractive solution. In 2012, Alex Krizhevsky reduced the top 5 error on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) from 26.2% to 15.3%, bringing CNNs to the attention of the world. "AlexNet" is considered one of the most influential papers in computer vision, having been cited over 70,000 times (Google Scholar). Since then, CNNs have continued to improve thanks to deeper architectures, better optimizers, transfer learning, bigger data sets, more computing power, open-source libraries, and new techniques for regularization. In 2013, Sermanet et al. easily surpassed 98.9% accuracy on the historic Microsoft cat and dog CAPTCHA.

Here we discuss an original deep learning solution. The object of our analysis was to program a working model for the private biotechnology company Tyccus Scientific. Primary goals included 1.) *Classification* – training a CNN to classify scientific glassware based on physical characteristics; 2.) *Prototyping* – providing a flexible and scalable framework with room to accommodate larger data sets and more sophisticated computer vision tasks; and 3.) *Consultation* – providing evidence-based advice for commercial deep learning applications, from data pipelining to unit testing and model evaluation. As of this writing, the client has developed an intelligent robotic work assistant; version 1 of the robotic gripper measures 80 cm from base to wrist and was prototyped successfully. Our work provides the foundation for the client to relate robotic computer vision to an industry setting.

2. Methods

2.1 Computer Hardware and Software

The model was trained on a Lenovo Thinkpad T440 with 2 processors, 2.9 GHz processing speed, 8 GB RAM, and 4 virtual CPUs running Linux Ubuntu.

Unless noted otherwise, image processing and modeling were implemented in Jupyter Notebook using the Python Keras library and TensorFlow backend.

Table 1: Software and Version

Python	3.6.9
Keras	2.3.1
TensorFlow	2.1.0
Jupyter Notebook	6.0.03
Ubuntu	18.04.4

2.2 Data Collection

Tagged high-resolution images were provided by the client (Tyccus Scientific). Images were obtained by capturing video footage using a high framerate digital camera, then randomly sampling video frames. Objects were presented to the camera in a variety of configurations (rightside-up, upside-down, sideways, at various distances and angles) to provide a representative sample. The environment was consistent across samples. Objects were photographed alone on a flat white surface under neutral diffuse light. This environment was carefully chosen to emulate the production environment. The quality of images was consistent and free from major artifacts, though we did notice some shadows and motion blur in a small number of samples (< 1%).

We discovered that file size could be reduced without sacrificing model performance, so we compressed images by 80% using the `EBImage` library in R, without modifying the original format (JPEG) or aspect ratio (3840 × 2160). This brought the mean file size from 4.28 MB to 625.8 KB, an 85.7% reduction (**Figure 1**).

In addition to client images, we also downloaded 2188 images from the Vector-LabPics data set at `https://www.cs.toronto.edu/chemselfies/`. The images are described in "Computer Vision for Recognition of Materials and Vessels in Chemistry Lab Settings and the Vector-LabPics Dataset" (2020). These images portray research glassware from chemistry laboratories as well as unrelated controls such as beverage containers, food, and drink. A variety of materials, such as liquids, solids, foams, suspensions, powders, gels, and vapors, are represented in various flasks, beakers, bowls, dishes, and pipettes. The images vary in aspect ratio, resolution, and quality. Some images have artifacts such as watermarks and text.

Altogether, there were 9 image classes in the full model. A sample of images is given in the **Appendix**. Images are summarized in **Table 2**.

Table 2: Image Classes

Class	Description	Train	Test	Total
A	Flask, Labeled, Empty, Round Body	1046	184	1230
B	Flask, Labeled, Empty, Round Body	682	120	802
C	Flask, Labeled, Empty, Eliptical Body	603	106	709
D	Flask, Labeled, Empty, Round Body	684	120	804
E	Bottle of Disinfectant, Labeled, Red Cap	319	56	375
F	Bottle of Blue Gels, Unlabeled, Blue Cap	1094	193	1287
G	Bottle of Blue Gels, Unlabeled, Blue Cap	1173	207	1380
J	Plain Drinking Glass, Empty, Unlabeled	1206	212	1418
Vector-LabPics	Glassware & Vessels from Chemistry Labs	1860	328	2188
		8667	**1526**	**10193**

2.3 Image Pre-Processing

Images were downscaled to a 150 × 150 aspect ratio (**Figure 2**) and converted to 3-dimensional arrays: the first dimension for the horizontal position of a pixel; the second dimension for the vertical position of a pixel; and the third dimension, with length 3, for the red, green, and blue channels in RGB color space. In line with common practice, pixel values were min-max normalized.

2.4 Directory Structure

Images were saved to separate subdirectories for each object class to preserve the accuracy of class labels, keep classes separate, and maintain flexibility. Images were shuffled by assigning a random string to each

filename. The default behavior of Keras is to sort images alphabetically during validation, so renaming was necessary to ensure validity.

Images were selected using `.flow_from_directory`, which parses the local directory tree and reads images from each subdirectory. All the pre-processing steps, such as downscaling, converting to arrays, and min-max normalization, were specified at this stage.

2.5 Data Augmentation

Data augmentation refers to random transformations of the original images. For image classification, this includes randomly rotating, cropping, and magnifying source images (**Figure 3**).

A distinction is made between augmenting the training data and augmenting the testing data. Both were used. At train-time, data augmentation involved random rotation, horizontal and vertical shifting, edge cropping, and magnification. The extent of these transformations can be tuned, and we selected moderate parameters, i.e. rotation ≤ 40° and other transformations ≤ 20% of baseline. At test-time, these parameters were reduced by half.

2.6 Prototyping

We originally designed a baseline model to which all other models could be compared, using images of cats and dogs downloaded from Kaggle. The model was based on the VGG architecture described in **Section 2.8**. Since the problem was a binary classification problem, we selected a binary loss function, binary crossentropy. Later, we adapted the code to handle multiple classes.

In small-scale tests, and to test different choices of parameters, we used a **reduced model** with 4 classes {Flask A, Flask B, Flask C, Flask D} and a control {Vector-LabPics}. The **full model** with 9 classes is presented at the end of this paper.

2.7 Loss Function

As with other machine learning algorithms, neural networks are an optimization problem. Keras uses backpropagation to calculate the gradient and gradient descent to minimize the loss function.

As we increased the number of image classes, we selected a categorical loss function. Our loss function, categorical crossentropy, or log loss, measures the loss of a prediction by computing the following sum:

$$\text{Loss} = \sum_{i=1}^{\text{output size}} y_i \cdot \log\left(\hat{y}_i\right) \qquad \textbf{(Equation 1)}$$

where $\hat{y}_i$ is the *i*-th scalar value in the model output, y_i is the corresponding target value, and *output size* is the number of image classes, equal to the number of neurons in the ultimate layer of the model.

2.8 Model

The inspiration for the model came from "Very Deep Convolutional Networks for Large-Scale Image Recognition" (2014) by Karen Simonyan and Andrew Zisserman, members of the Visual Geometry Group (VGG) at the University of Oxford. Their model achieved state-of-the-art performance in the ILSVRC 2014 competition, building upon and advancing the architectural principles of AlexNet. We provide a diagram of the model in **Figure 4**.

A "VGG" model relies on several convolution layers followed by a max pooling layer. We summarize the main principles here. First, the convolution layer applies *filters* to the input (**Figure 5**). Mathematically, this is a linear operation representing the dot product between a filter-sized subregion of the input and a 2D matrix of weights. In the beginning, these weights are random, having been sampled from an arbitrary data-generating distribution, but as the model is optimized, the weights are continuously re-adjusted. Over time, filters "learn" to recognize important features by generating dot products that map to the appropriate targets.

Each convolution layer applies multiple filters, and each filter is applied to multiple regions of the input, so that learned features are recognized no matter where in the image they occur; this is called the location-invariance property.

The output of each convolution layer is mapped to an activation function, typically ReLU (rectified linear unit). ReLU is a simple function: $f\left(x\right) = x$ for $x > 0$ (identity), 0 otherwise. Both the function and its derivative are easy to compute, making it suitable for forward propagation and backpropagation, and because ReLU zeroes out negative inputs, it promotes sparsity in the network – reducing the total computation time.

Additionally, ReLU is robust to the vanishing gradient problem because its derivative is always 0 or 1.

A convolution layer and activation function coupled together output a set of *feature maps* encoding important features discovered by filters.

Finally, the max-pooling layer applies a *maximum pooling* operation to each feature map (**Figure 6**). Max pooling is another simple function: an input matrix is subdivided into non-overlapping equal-sized regions, the maximum scalar in each region is extracted, and these are joined together in an output matrix. For example, if the input was a 4×4 matrix, we would subdivide it into four 2×2 subregions. Next we would find the maximum value in each subregion, and join these to make a new 2×2 output matrix. Max-pooling reduces the size of the input without sacrificing too much information, greatly reducing computation time. In fact, max-pooling limits overfitting by reducing the dimensionality of the input and mapping an abstracted form of the input to the next layer.

Typically, three or more of these units (convolution → activation → max-pooling) are chained together in sequence. We refer to these as *convolution blocks.* The advantage of having more than one convolution block is an ongoing topic of research. It is thought that the earlier stages of the network specialize in detecting simple edges, contours, and colors, while deep layers learn higher-order features like objects and textures.

After the convolution blocks, the input is flattened to a 1D vector and mapped to a fully connected layer. In this layer the softmax function normalizes the output to a probability distribution over the predicted output classes. These outputs $\hat{y}_1, \ldots, \hat{y}_n$ always sum to 1, so as $\hat{y}_i \to 1$, $\sum \hat{y}_i^{\,c} \to 0$. Hence, by minimizing the categorical crossentropy (**Equation 1**) we reward only good, confident predictions. The loss is minimized if $y_i = \hat{y}_i$ for all i in $1, \ldots, n$.

To maintain consistency, we used an 85/15 train/test ratio in all our tests. To account for data augmentation, we multiplied the step size by a factor of 3.

2.9 Mini-Batch Size

The *mini-batch size* is the number of images passed to the model at a time. We considered three mini-batch sizes: 8, 32, and 64.

2.10 RMSProp and Adam

We considered two stochastic gradient descent optimizers, RMSProp with default learning rate 0.001 and Adam with default learning rate 0.01.

2.11 ReLU and Hard Swish

In place of ReLU we tested Hard Swish. Hard Swish is a type of activation function based on Swish, which was found to deliver good results on certain computer vision benchmarks (Ramachandran et al. 2017). Hard Swish proposes to reduce the computation time of Swish (Howard et al. 2019) by approximating the computationally expensive sigmoid with a piecewise linear analogue. Since Hard Swish is not available for Keras, it was implemented manually as follows:

$$\text{h-Swish}(x) = \frac{x \cdot \text{ReLU}6(x+3)}{6} \qquad \textbf{(Equation 2)}$$

where

$$\text{ReLU}6(x) = \min(\max(x, 0), 6) \qquad \textbf{(Equation 3)}$$

2.12 Dropout and Batch Normalization

We tested two well-known regularization techniques, dropout (**Figure 7**) and batch normalization ("batch norm"). In the first case, we applied a 50% dropout fraction in the second-to-last layer of the model, before softmax. In the second case, we applied batch norm after each activation (ReLU) and after the flatten layer, 5 times in total. Lastly, we tested a combination, complementing batch norm with a conservative 20% dropout fraction before softmax.

2.13 Stress Test

As a final test of model performance, we considered two additional sources of unseen data: a stock image of a flask retrieved from the web and images of flasks from an different data set provided by the client. This data set contained images of Flask A, Flask B, Flask C, and Flask D. These images portrayed flasks under different lighting conditions and were photographed using a different camera. The lighting was darker and murkier, backgrounds were more busy, and the flasks were generally positioned inside some kind of styrofoam box. Since the ambient environment was entirely different from the training sample, the images seemed like a good test of the model's robustness.

3. Results

3.1 Mini-Batch Size

After training the reduced model for 15 epochs, we did not observe a substantial effect of mini-batch size on accuracy or runtime. We did observe some differences between trials; however, it is difficult to draw conclusions about the effect of one single hyperparameter without averaging together multiple trials, due to the stochastic behavior of neural networks. We decided to use a mini-batch size of 8 (see **Discussion**).

3.2 RMSProp and Adam

After training the reduced model for 30 epochs, we did not observe a substantial difference between Adam or RMSProp with respect to accuracy or runtime. Both optimizers demonstrated excellent performance. We decided to use Adam (see **Discussion**).

3.3 ReLU and Hard Swish

After training the reduced model for around 10 epochs, we found that the manual implementation of Hard Swish was significantly slower than ReLU for both a dropout-based architecture and batchnorm-based architecture, adding hundreds of seconds of training time to each epoch. In the end, Hard Swish was abandoned in favor of ReLU.

3.4 Dropout and Batch Normalization

After training the reduced model for 50 epochs, we found that the reduced model with dropout gave much better performance than the reduced model with batch norm, not only with respect to validation accuracy (**Figure 8**, **Figure 9**) but also the runtime (**Figure 10**).

Preliminary testing suggested that the combination of batch norm with dropout did not produce substantially better results than batch norm alone.

We note that the performance on the validation data is better than the performance on the training data for Figure 8, which seems to defy expectations. To quote the Keras documentation:

> "A Keras model has two modes: training and testing. Regularization mechanisms, such as **Dropout** and L1/L2 weight regularization, are turned off at testing time."

In other words, since our model made judicious use of dropout, the behavior in Figure 8 is not unusual.

3.5 Full Model

For the full model, we chose parameters based on our experiments and literature review up to that point. The full model employed a basic VGG-style architecture (Figure 4) with ReLU activation, a small mini-batch size, Adam, and dropout with a 50% dropout fraction.

The model demonstrated remarkable performance on the validation data after only 11 epochs, surpassing 98% validation accuracy (**Figure 11**). The confusion matrix also suggested good sensitivity and specificity (**Figure 12**).

Compared to the reduced model, the full model was more computationally demanding to train. The number of training images was **78%** greater for the full model (8667 vs. 4875). However, the average training time per epoch was **134%** greater (1760 seconds vs. 750 s).

Using the formula *Data Size* × *Data Augmentation Scaling Factor* ÷ *Time per Epoch*, we can calculate the number of images processed per second. This was (8667 × 3) ÷ 1760 = **14.8 images/second** for the full model and (4875 × 3) ÷ 750 = **19.5 images/second** for the reduced model. Interestingly, the full model was slower to process images, in addition to simply having more images. This is probably a result of adding more image classes. As the number of classes increases, the read-write head might be slower to retrieve images; the number of target neurons increases, which means more parameters to estimate; the summation in the softmax calculation has more terms; and the response surface changes, which may affect the calculation of the gradient.

3.6 Stress Test

In contrast to the model performance on the validation data, the model performed poorly on the stress test. While the model was able to correctly classify generic stock photos of flasks (**Figure 13**) as "Other," i.e. not belonging to any one of the target classes, the model failed to correctly classify images of flasks from the second data source (**Figure 14**). Again, these images showed the flasks under different lighting conditions in a different setting. The model also classified these images as "Other," and with high confidence. That is, there was negligible softmax activation in the neurons corresponding to the correct class label.

4. Discussion / Conclusion

Our experiments show two sides of neural networks: their ability to generalize exceptionally well, and their ability to fail completely. We will review each of these in turn.

We observed that convolutional neural networks can deliver state-of-the-art results even on real world data, provided the testing sample resembles the application domain. Our most sophisticated CNN surpassed >99% accuracy on the validation data, even as 9 different classes were considered. In addition, we demonstrated that high-resolution images were unnecessary and that file compression could dramatically reduce the size of the training data. Finally, training the full model for 20 epochs required only 10 hours on a laptop computer.

To verify that the model was validated on unseen data, we completed two simple spot checks. First, we made sure that the number of images in the training sample and the number of images in the validation sample summed to the correct number of files (10193). Next, we confirmed that `train_generator.filepaths` and `validation_generator.filepaths`, where train and test images were sourced, did not overlap. These spot checks provided strong evidence against data leakage.

On the other hand, our model failed to deliver good predictions in the stress test. The simplest explanation is that the model failed to generalize because the new images were unlike the images the model had seen. Data augmentation attempts to resolve this issue by providing the model with a more representative data set, but our augmentation strategy did not make use of changes to brightness and contrast. These images, with their dusky lighting and deep noisy shadows, may have hijacked the model's color-detecting filters. But if the problem is as simple as a non-representative training sample, then adding the second set of images to the training sample might redeem the model. In the future, it would be a good hypothesis to test.

Results of the stress test highlight a key difficulty in commercial deep learning. For robust generalization, the model needs a huge training sample, especially as new object classes become available or the scope of the project develops with time. Data acquisition is the same as chasing a moving target.

It is worth mentioning that some models do pass all the initial tests during inspection, then fail to issue good predictions for various reasons. D'Amour et al. outline a problem called "underspecification" in which models show "divergent behavior when applied to real-world settings" (2019). With respect to deep learning, the authors argue that some models suffer from an inductive bias since the response surface of the loss function can change between train-time and test-time. Similarly, Pearl and Bareinboim introduce the concept of "transportability analysis," the goal of which is explaining "why transfer-learning algorithms fail to converge or perform" in new environments (2011). These ideas are related to the classical concept of overfitting but involve new challenges specific to contemporary machine learning workflows.

Given the results of our analysis, we argue that classifying images of scientific glassware is a constrained and manageable problem. Flasks differ sharply in the threading of the neck, the shape of the body, the colors of caps and stoppers, the writing and patterns on the labels, the label-to-body ratios, size, reflectivity, and other factors. As long as images are representative of the application domain, deep learning can succeed. But when new images break the mould with patterns that are novel, then deep learning is easily fooled.

5. Appendix: A Discussion of Different Hyperparameters

One of the challenges in deep learning is the vast number of hyperparameters. To understand where the model could be improved, here we review the set of hyperparameters and the rationale for our decisions.

We chose Adam (adaptive moment estimation) as our optimizer. According to the authors, "The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients" and "is well suited for problems that are large in terms of data and/or parameters" (Kingma and Ba, 2015). Adam belongs to a class of optimizers with *adaptive learning rate*. Garbin et al. write that "a non-adaptive optimizer . . . can outperform adaptive optimizers, but only at the cost of a significant amount of . . . hyperparameter tuning, while an adaptive optimizer . . . performs well without much tuning" (2020). In the technical report "An overview of gradient descent algorithms" (2016), Ruder finds that Adam gives slightly better results than RMSProp, the most famous alternative. Adam introduces several innovations. By harnessing an adaptive learning rate and computing the moving average of the gradients, Adam has robust stochastic behavior and usually has good results out-of-the-box.

Where accuracy is concerned, batch size is not usually mentioned. However, it has been demonstrated that smaller batch sizes lead to better generalization and faster convergence (Keskar et al. 2016). By limiting the number of samples at each iteration, a smaller mini-batch size contributes noise to the estimate of the gradient. The result is beneficial when combined with the properties of stochastic gradient descent. Masters and Lushchi (2018) found that the best mini-batch size is often 32 or less, and sometimes as small as 2 or 4. However, they cautioned that each data set is different.

Data augmentation is reported to improve performance the most when implemented at test time as well as train time (Thoma 2017). Jason Brownlee documents on his website that aggressive parameters work well, of the kind we implemented (scaling, cropping, *etc*.) However, there is room for improvement in perturbing brightness and contrast.

Another tuning parameter is color, or more formally the dimensionality of the input. Since color encodes important information (for example, two of our image classes had blue caps), removing color can sabotage the model. On the other hand, color can invite overfitting by allowing the model to train on trivial features like the color of the background or ambient light. In a future experiment, it would be useful to investigate black-and-white inputs and their effect on accuracy.

With batch size, and with certain other deep learning parameters, there is a specific motivation for choosing parameters that are powers of 2. By default, TensorFlow supports multithreading and distributes matrix multiplications over multiple cores. Since most computers have 2, 4, or 8 cores (virtual or otherwise), model configurations involving powers of 2 enjoy certain advantages when it comes to speed and efficiency. In deep learning, purely computational concerns need to be balanced against statistical ones.

Image resolution is another important parameter, but it seems to be shrouded in mystery. While there are several rules of thumb, their empirical basis is uncertain. Nevertheless, a few things are clear. Downscaling the resolution speeds up model training and allows for smaller filters to be used. Too much downscaling has a detrimental effect, as images start to lose information. Our model resizes images to 150 × 150, which seems to be in the recommended range of most authors. Many of the competition-winning models submitted to ILSVRC resize images to 256 × 256. In the future, it might be worth trying other aspect ratios.

Some very basic parameters, like the volume of filters, the max-pool size, and the number of convolution blocks seem to involve art as well as science. In general, it has been found that VGG-style architectures perform well, so we did not question their core components. We used three convolution blocks, which is reasonable for a laptop computer and a relatively small data set. While deeper networks might provide better results, they are harder to tune, more expensive to train, and vulnerable to overfitting. For computational reasons, we opted to use 3×3 filters and 2×2 pool size; these values are simplest to calculate.

With respect to the number of filters, we followed a popular convention. The first two convolution blocks apply 32 filters each while the third applies 64 filters. Some practicioners suggest a statistical motivation for this trend. Presumably, the set of all possible low-level features (like lines and edges) is smaller than the set of all high-level features (like flasks), so it makes sense to allocate more filters in deep layers. Additionally, there are computational factors. After multiple max-pooling operations, the inputs are smaller in deeper stages of the network, so we can afford more parameters in the model. We spend some of that computational budget on more filters.

Other variations include stacking multiple convolution layers, placing dropout layers after each activation, doubling the number of filters after each successive convolution block, and having more than one fully connected layer. These would be interesting variants to test.

The decision to try Hard Swish instead of ReLU was based on empirical findings. Swish was proposed to give good results on a variety of image recognition benchmarks (Ramachandran et al. 2017). Hard Swish was then proposed to speed up the computation of Swish (Howard et al. 2019). While we would have liked to investigate these properties, we found that Hard Swish was too slow, adding hundreds of seconds of training time to each epoch. In the long run, there are more important hyperparameters to tune anyway. We fell back to using ReLU, which takes advantage of the low-level TensorFlow API.

The decision to try batch normalization was also based on empirical findings. Batch norm has become a staple in competition-winning neural networks, and it tends to outperform dropout on large-scale computer vision problems. It has been shown that batch norm improves performance and accelerates convergence by reducing

the time to locate an optimum. Batch norm also seems to have a regularizing effect as it adds some noise to the inputs. While the source of these benefits is the subject of some debate, Santurkar et al. propose that batch norm makes the estimate of the gradient more stable by smoothing the response surface of the loss function (2018). Functionally, batch norm is a simple calculation involving re-centering and re-scaling pixel values across mini-batches.

We observed that batch norm was 37% slower than dropout. This is reasonable. Gitman and Ginsburg (2017) report that batch norm consumed 25% of total training time for their deep learning model. So while batch norm can accelerate learning by reducing the time to locate the optimum, the actual training time per epoch increases.

Dropout and batch norm do not work well together in practice. This was pointed out by the original authors of the batch norm paper (2015) and was reviewed more recently by Garbin et al. (2020), who identified a narrow set of conditions in which both techniques might work well together: namely, placing a dropout layer with a 20% dropout fraction at the end of the model after all the batch norm layers.

Given the tremendous volume of empirical support for batch norm, it came as a surprise when batch norm had poor performance in our experiments. We are not sure why this is the case, but it may have something to do with the size of the data. Ian Goodfellow wrote, "Dropout is mostly a technique for regularization. It introduces noise into a neural network to force the neural network to learn to generalize well enough to deal with noise . . . Batch normalization is mostly a technique for improving optimization . . . When you have a large dataset, it's important to optimize well, and not as important to regularize well, so batch normalization is more important for large datasets" (Quora post, 2016). If batch norm were combined with another regularization technique, it might give better results.

Works Cited

Brownlee, Jason. "How Do Convolutional Layers Work in Deep Learning Neural Networks?" *Machine Learning Mastery*, 16 Apr. 2019, machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/.

---. "How to Classify Photos of Dogs and Cats (with 97% Accuracy)." *Machine Learning Mastery*, 16 May 2019, machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/.

"Categorical Crossentropy Loss Function | Peltarion Platform." *Peltarion.com*, peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy.

D'Amour, Alexander, et al. "Underspecification Presents Challenges for Credibility in Modern Machine Learning." *ArXiv:2011.03395 [Cs, Stat]*, 24 Nov. 2020, arxiv.org/abs/2011.03395. Accessed 3 Dec. 2020.

Elson, Jeremy, et al. "Asirra: A CAPTCHA That Exploits Interest-Aligned Manual Image Categorization." *Www.Microsoft.com*, 1 Oct. 2007, www.microsoft.com/en-us/research/publication/asirra-a-captcha-that-exploits-interest-aligned-manual-image-categorization/.

Eppel, Sagi, et al. "Computer Vision for Recognition of Materials and Vessels in Chemistry Lab Settings and the VectorLabPics Dataset." *Figshare*, 20 Apr. 2020, chemrxiv.org/articles/Computer_Vision_for_Recognition_of_Materials_and_Vessels_in_Chemistry_Lab_Settings_and_the_VectorLabPics_Dataset/11930004. Accessed 3 Dec. 2020.

Golle, Philippe. "Machine Learning Attacks against the Asirra CAPTCHA." *Proceedings of the 15th ACM Conference on Computer and Communications Security - CCS '08*, 2008, 10.1145/1455770.1455838.

Ioffe, Sergey, and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." *ArXiv.org*, 2015, arxiv.org/abs/1502.03167.

Keskar, Nitish Shirish, et al. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima." *ArXiv:1609.04836 [Cs, Math]*, 9 Feb. 2017, arxiv.org/abs/1609.04836.

Kingma, Diederik P, and Jimmy Ba. "Adam: A Method for Stochastic Optimization." *ArXiv.org*, 2014, arxiv.org/abs/1412.6980.

Masters, Dominic, and Carlo Luschi. *REVISITING SMALL BATCH TRAINING FOR DEEP NEURAL NETWORKS*.

"Max-Pooling / Pooling - Computer Science Wiki." *Computersciencewiki.org*, 2018, computersciencewiki.org/index.php/Max-pooling_/_Pooling.

"Papers with Code - Searching for MobileNetV3." *Paperswithcode.com*, 2019, paperswithcode.com/paper/searching-for-mobilenetv3. Accessed 3 Dec. 2020.

Pearl, Judea, and Elias Bareinboim. *Transportability of Causal and Statistical Relations: A Formal Approach* *.

Ramachandran, Prajit, et al. "Searching for Activation Functions." *ArXiv.org*, 2017, arxiv.org/abs/1710.05941.

Ruder, Sebastian. "An Overview of Gradient Descent Optimization Algorithms." *ArXiv.org*, 2016, arxiv.org/abs/1609.04747.

Santurkar, Shibani, et al. "How Does Batch Normalization Help Optimization?" *ArXiv:1805.11604 [Cs, Stat]*, 14 Apr. 2019, arxiv.org/abs/1805.11604. Accessed 3 Dec. 2020.

Sermanet, Pierre, et al. "OverFeat: Integrated Recognition, Localization and Detection Using Convolutional Networks." *ArXiv:1312.6229 [Cs]*, 23 Feb. 2014, arxiv.org/abs/1312.6229.

Thoma, Martin. *Analysis and Optimization of Convolutional Neural Network Architectures*. 2017.

"What Is the Difference between Dropout and Batch Normalization? - Quora." *Www.Quora.com*, www.quora.com/What-is-the-difference-between-dropout-and-batch-normalization. Accessed 3 Dec. 2020.

Images

Figure 1: File Compression

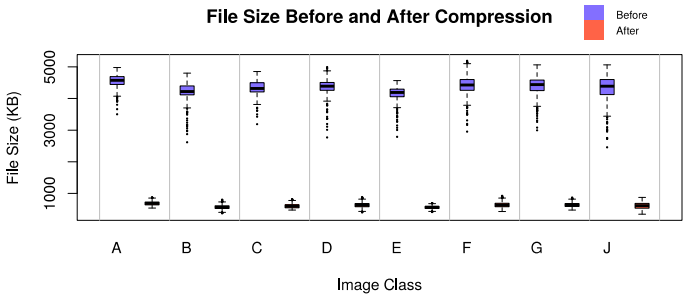


Figure 1: Image sizes in kilobytes (KB) before (*blue*) and after (*red)* 80% file compression, classes A through J.

Figure 2: Downsizing Images

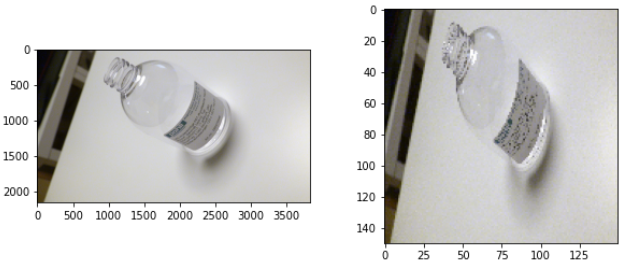


Figure 2 (above): Sample image before (*left*) and after downsizing to a standard aspect ratio.

Figure 3: Data Augmentation

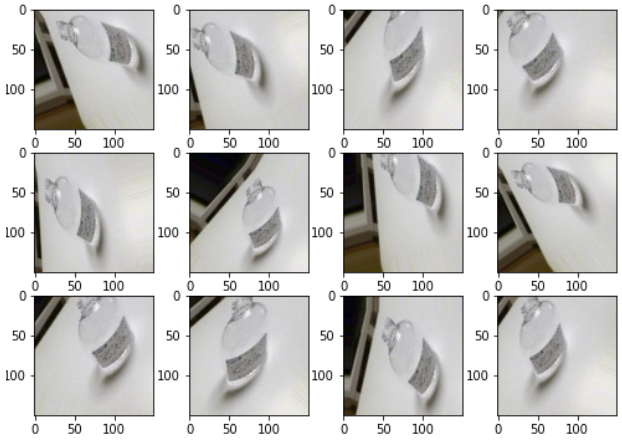


Figure 3 (right): Random sample of 12 images generated from data augmentation at train-time.

Figure 4: Model Diagram of Convolutional Neural Network Architecture

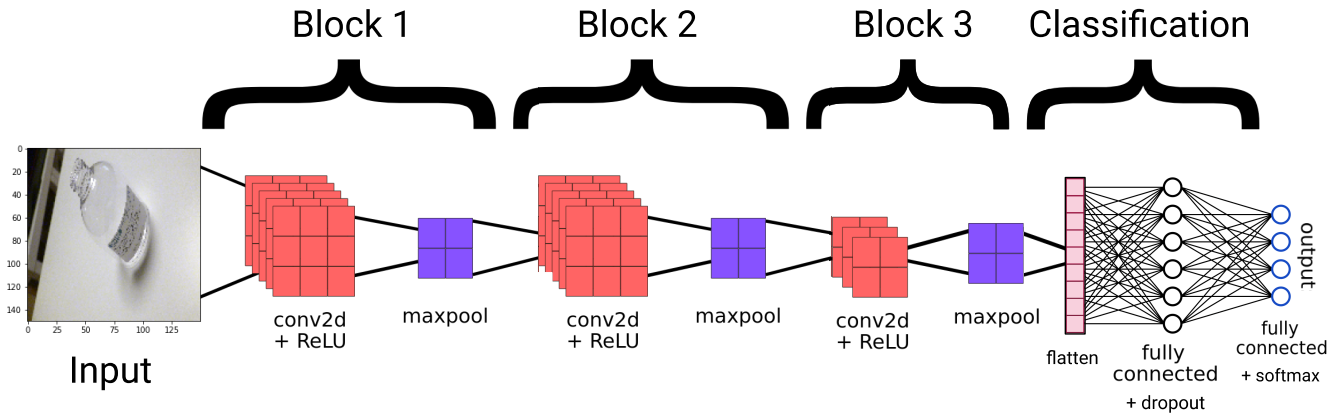


Figure 4: Model diagram of CNN architecture. 8 images are passed to the model in each mini-batch. Images are converted to arrays, min-max scaled, and downsized to a 150 × 150 aspect ratio. Three convolution blocks are chained together in sequence. Each has a convolution layer layer with multiple 3×3 filters followed by a ReLU activation and a 2×2 max-pooling layer. The number of filters is 32 for the first two convolution layers and 64 for the last convolution layer. Then, the input is flattened and passed to a fully-connected layer with 64 neurons. Dropout is applied with a 50% dropout fraction. The last layer has a number of neurons equal to the number of image classes. Here the softmax activation function normalizes the output to a probability distribution over the predicted class probabilities.

Figure 5: Convolution Filters

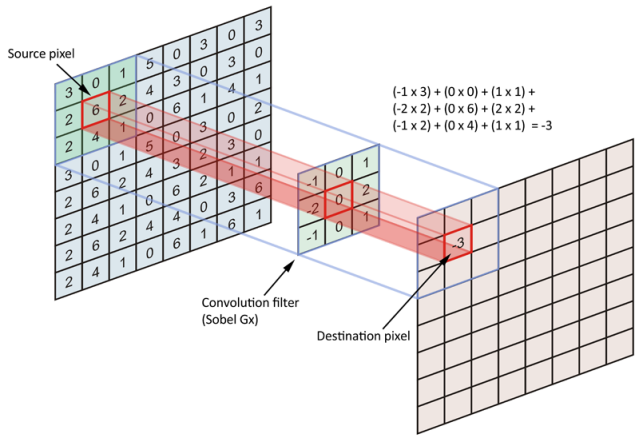


Figure 5: A convolution is the dot product between a 3×3 subregion of the input and a 3×3 matrix of trainable weights called the *filter*. The filter slides over the input, one pixel at a time, generating a new matrix with the same number of dimensions. One convolution layer learns multiple filters.

Figure 6: Max-pooling with a 2×2 pool size is applied to the input, reducing dimensionality while preserving key information.

Figure 7: In dropout, a pre-specified fraction of neurons is randomly eliminated to thin the network and reduce overfitting.

Figure 6: Max-Pooling

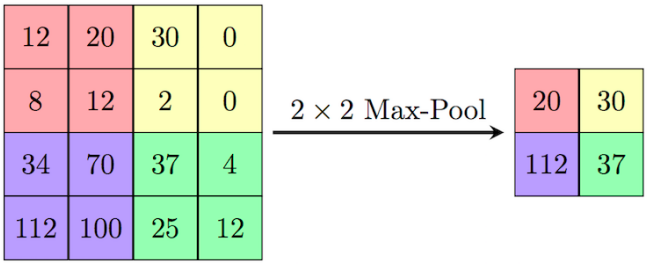


Figure 7: Dropout

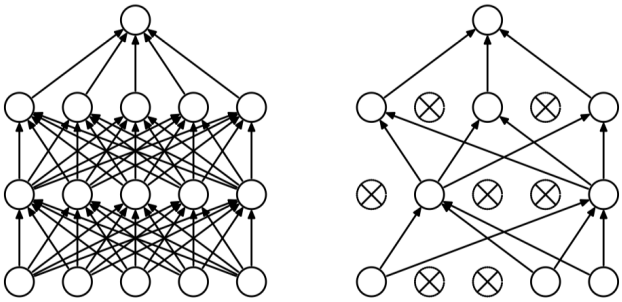


Figure 8: Model Accuracy (Dropout)

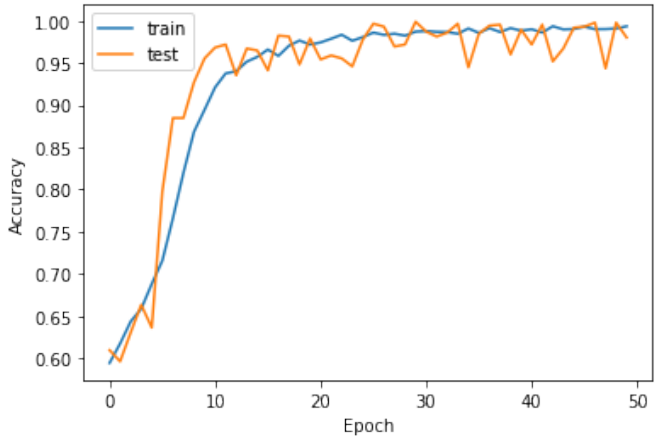


Figure 9: Model Accuracy (Batch Norm)

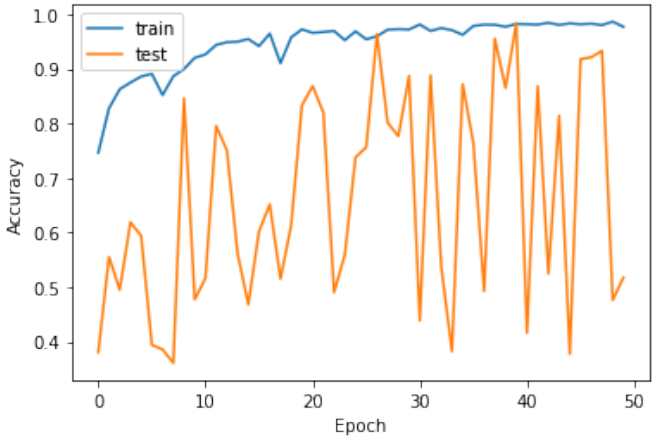


Figure 8: Accuracy of the reduced model {A, B, C, D, and Vector-LabPics} with dropout. (*Blue*: performance on the training data. *Orange*: performance on the validation data.) With dropout, the average validation accuracy during the last 15 epochs was 98.08% (High: 99.76%; Low: 94.35%).

Figure 9: Accuracy of the reduced model with batch normalization. After 50 epochs, the model never achieved consistent performance on the validation data, overfitting the training data instead.

Figure 10: Runtime (Dropout vs. BatchNorm)

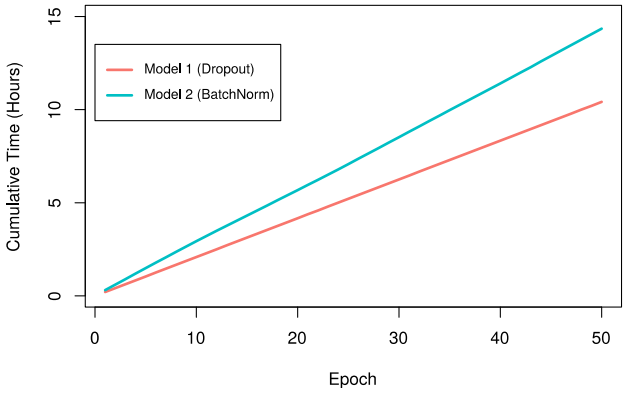


Figure 10: Runtime (in hours) to train the reduced model for 50 epochs. *Red*: reduced model with dropout. *Blue*: reduced model with batch norm. Training the reduced model required 10.4 hours with dropout (around 750 seconds per epoch) compared to 14.3 hours with batch norm (around 1033 seconds per epoch). Batch normalization was 37% slower.

Figure 11: Model Accuracy (Full Model)

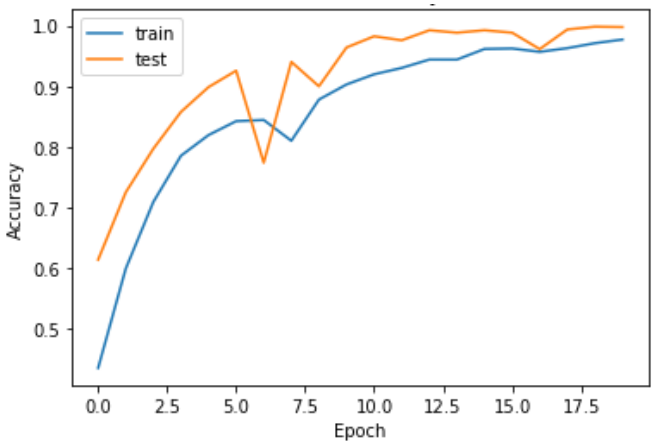


Figure 12: Confusion Matrix (Full Model)

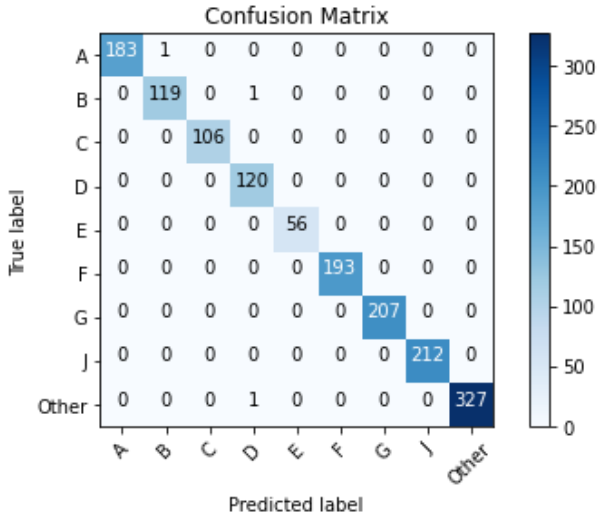


Figure 11: Accuracy of the full model {A, B, C, D, E, F, G, J, and Vector-LabPics} over 20 epochs. The average validation accuracy during the last 8 epochs was 98.88% (High: 99.80%; Low: 96.11%). It is possible that training the model for a longer duration would yield even better accuracy.

Figure 12: Confusion matrix for the same model demonstrating sensitivity and specificity for all 9 classes. All but 3 of the observations were correctly classified.

Figure 13: Stress Test

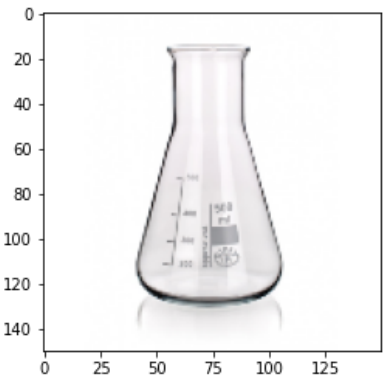


Figure 14: Stress Test

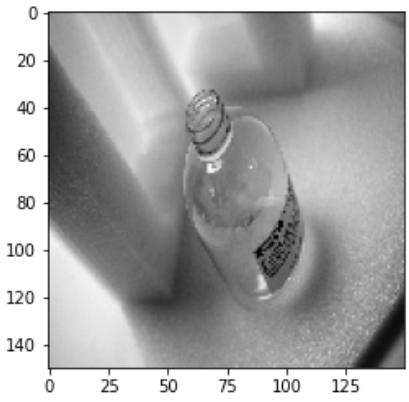
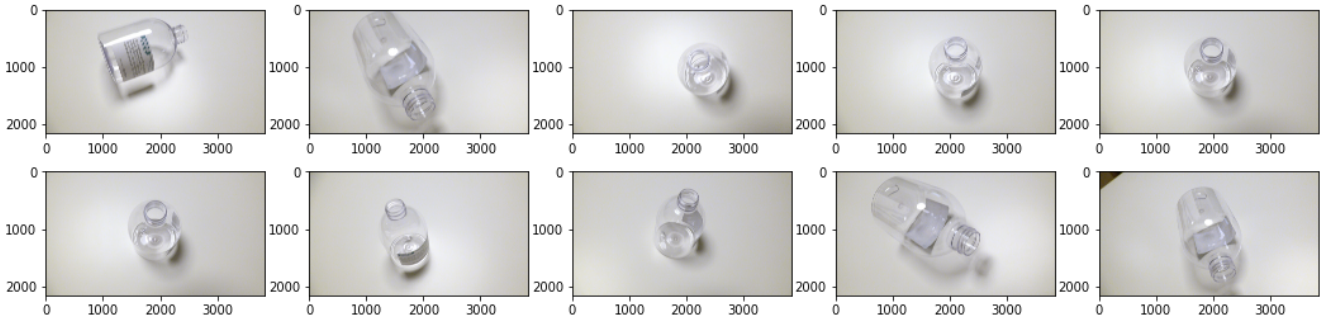


Figure 13 (left): Generic stock photo of an Erlenmeyer flask from the web. The model correctly classified this is "Other," i.e. the same image class as Vector Lab-Pics.

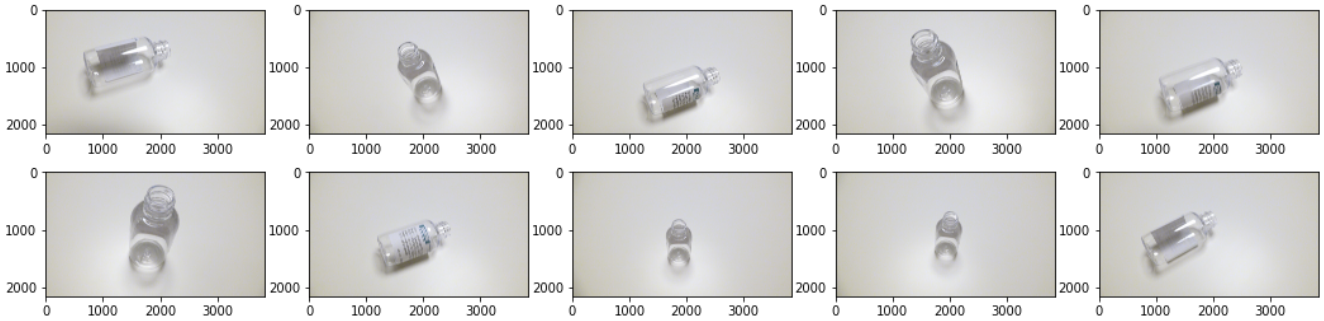
Figure 14 (right): Image of Flask A from a second data source provided by the client. The model failed to correctly classify images from this data set, classifying them as "Other."

Appendix

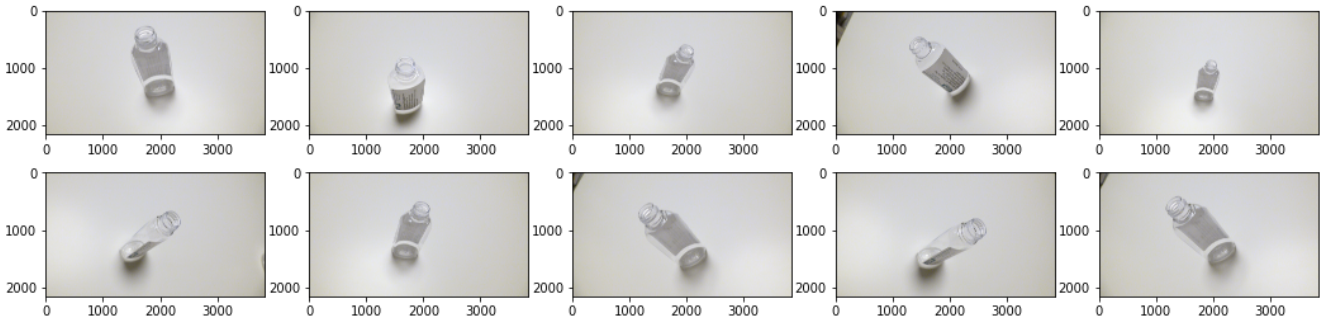
Class A



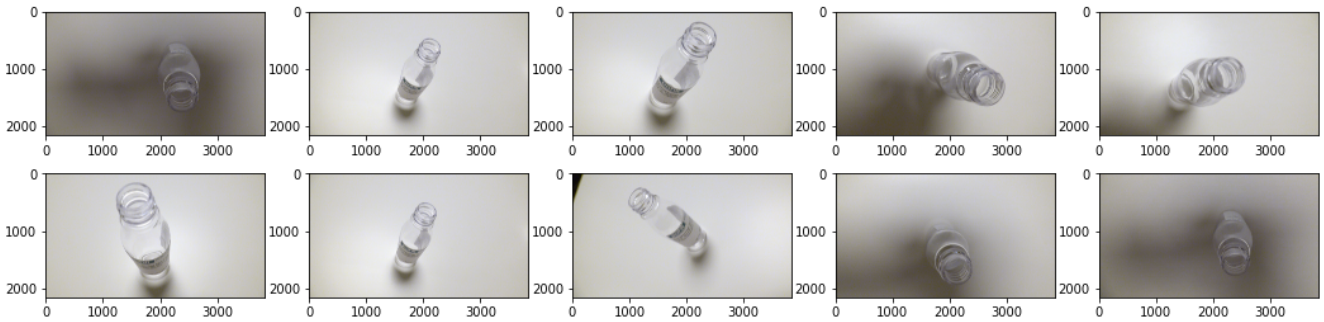
Class B



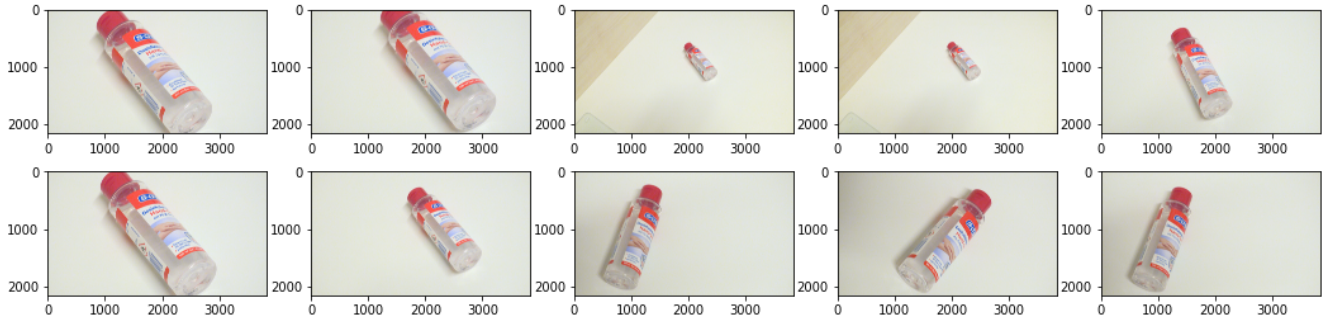
Class C



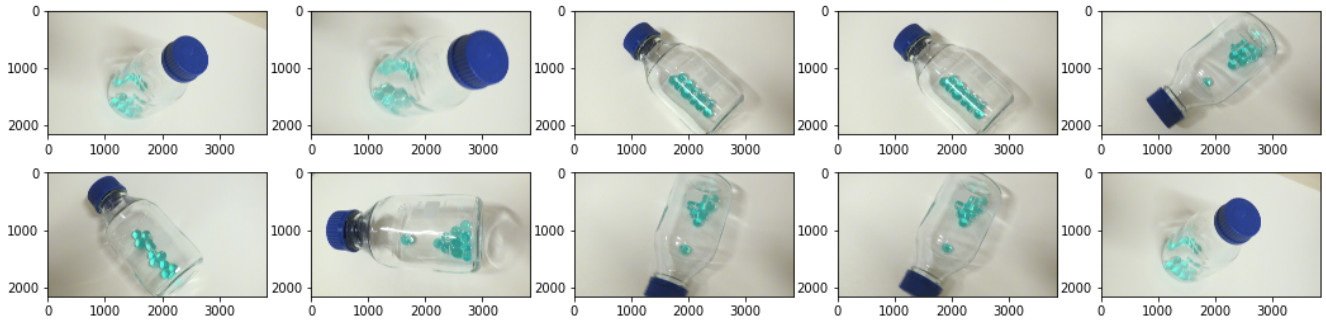
Class D



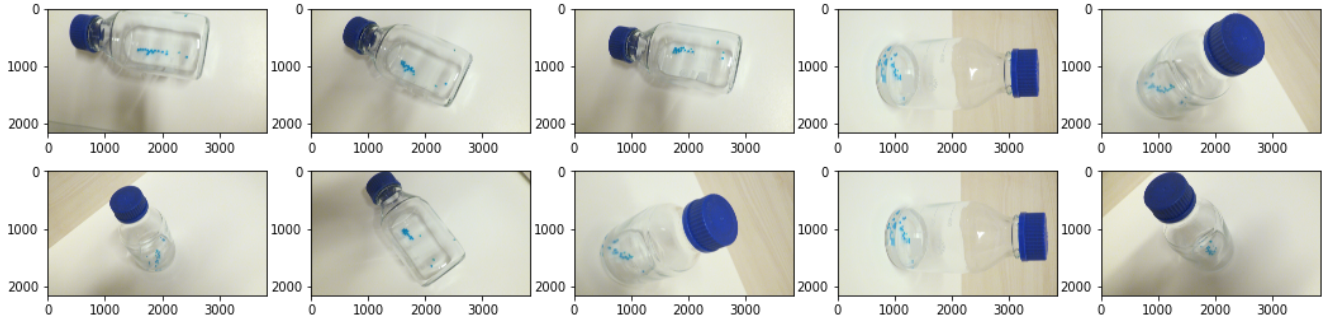
Class E



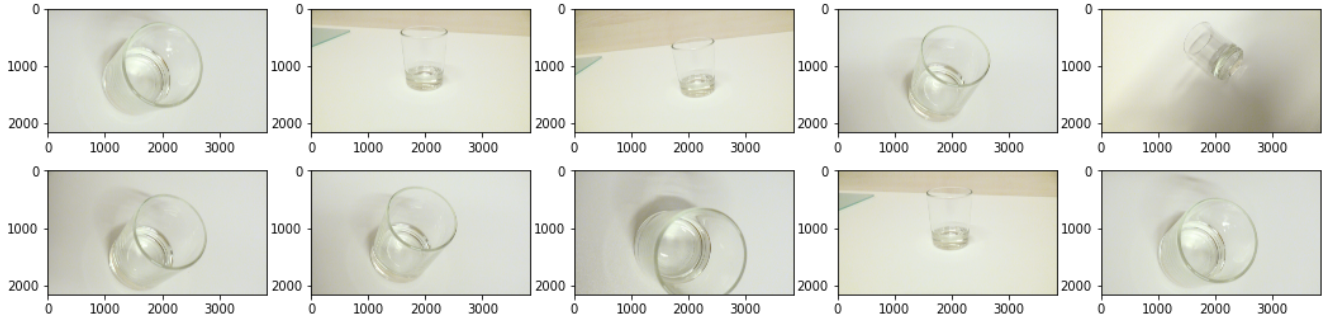
Class F



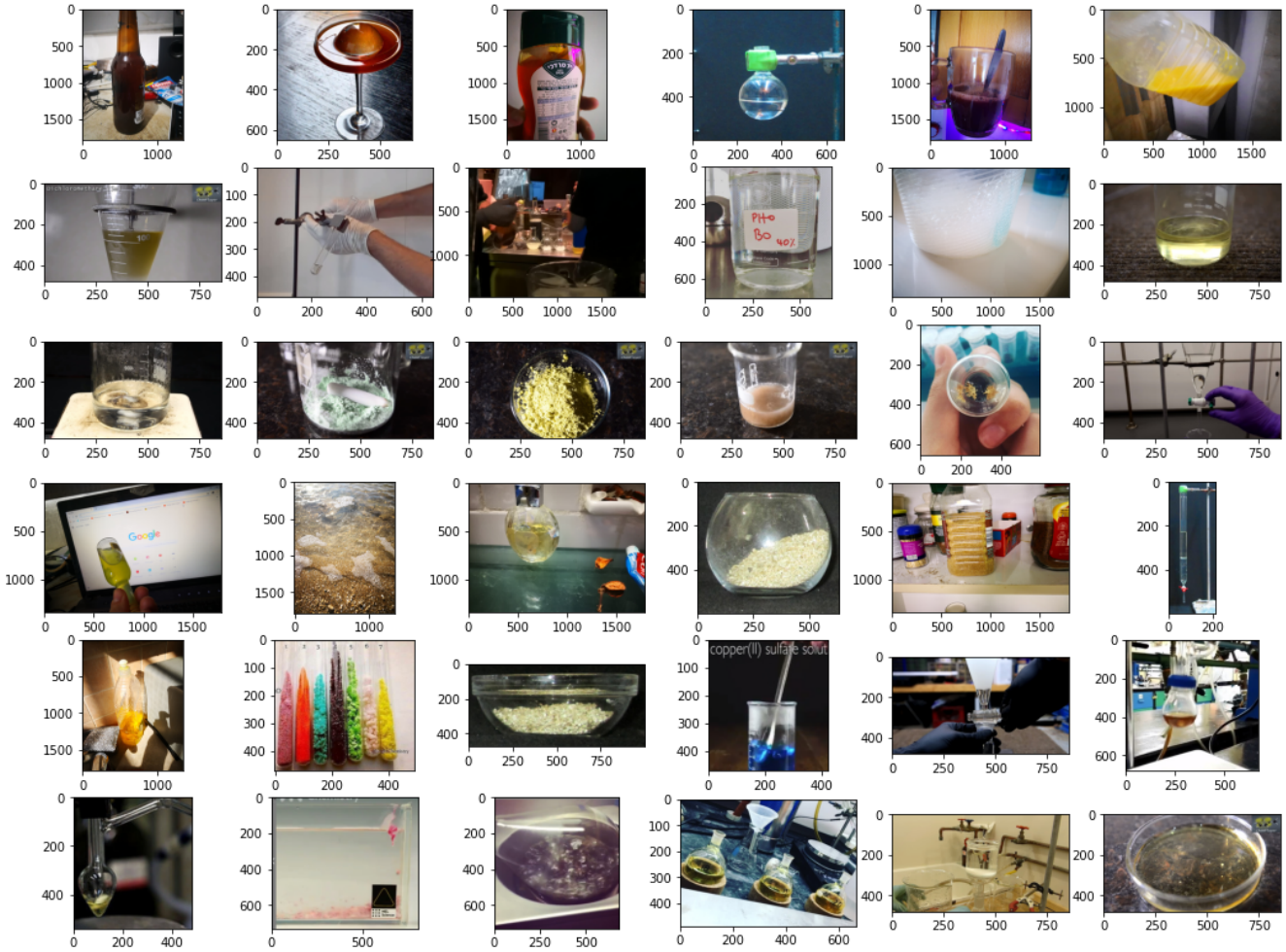
Class G



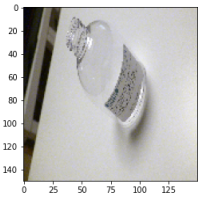
Class J



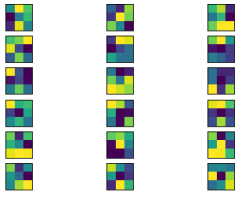
Vector Lab-Pics (Control)



Convolution Filters and Feature Maps

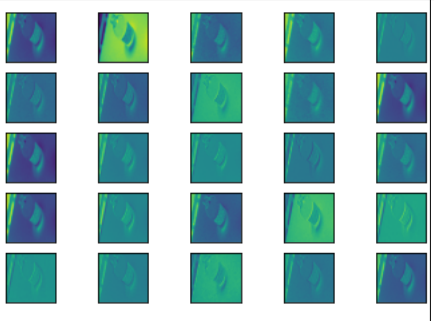


Input Image

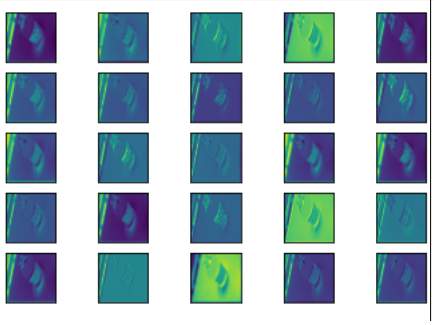


18 sample filters from the first convolution layer, visualized as a heatmap of learned weights

Block 1 Feature Maps



Block 2 Feature Maps



Block 3 Feature Maps