Assignment 5: Action Recognition with Automatic Model Search

Andrius Jaškauskas (6070981)

Bjartur Hafþórsson (6177417)

I. TASK A: Classification on Stanford-40

Neural Network Architecture

A lot of thought and testing went into designing the network's structure, and the network consists of a total of 9 layers, including the fully connected output layer. The dataset was preprocessed such that all images were loaded in grayscale. This is because for action classification color information is not too important, and dropping the color channels makes for a much faster program. The images were cropped into a square from the middle outwards and then resized to 48x48 pixels, as to reduce the dimensionality of the problem. Each image was normalized to speed up learning.

The first layer is a convolutional layer of 32 3x3 filters. The number 32 was chosen simply because that makes for a decent number of activation maps along with being a power of two, having the possibility of making the program ever so slightly faster. A 3x3 filter was chosen as the images are already quite small. However, since the filter is relatively small, it was decided to have a stride of 1 as to explore all possible 3x3 windows.

The second layer is a batch normalization layer. We include this layer before the activation of the convolutional layer as we are adjusting and scaling the activations beforehand. This layer is included to allow for higher learning rates, as well as reducing overfitting due to its slight regularization effects. After this layer a *ReLU* activation is performed on the now normalized results of the convolutional layer.

The third layer is a max pooling layer of size 2x2, if it were larger it would be too destructive. This layer greatly reduces the dimensionality. We choose max pooling as it is more sensitive to the existence of patterns in the pooled region.

The fourth layer is a dropout layer of 0.5. This means that each neuron has a 50% probability of being ignored. This greatly reduces overfitting, as it makes sure that the network is able to reach its purpose through multiple connections, is less reliant on any single neuron, and will therefore generalize better.

For layers five to eight, we repeat the exact same process again (convolutional layer, batch normalization layer, max pooling layer, dropout layer). This is a good idea since after our previous operations the network now considers higher level features than before during the convolution (since we used pooling to emphasize the larger features of our data), and the dimensionality of the data is again reduced before being passed to the fully connected layer.

Finally, the results are flattened to move the data to a single dimension before being fed to the fully connected (dense) output layer. At first it was tested to include a dense layer of 3200 nodes (the size of the output of the flatten operation) before the final prediction layer of 40, but as it yielded slightly worse results in terms of validation accuracy (although it converged faster), it was later removed. The latter structure was also tested with another fully connected layer of size 256, but as with the previous structure, although converging way faster, it yielded worse results and was therefore removed.

The construction of the network can be seen in `createBasicClassifier()` function.

Network Visualization

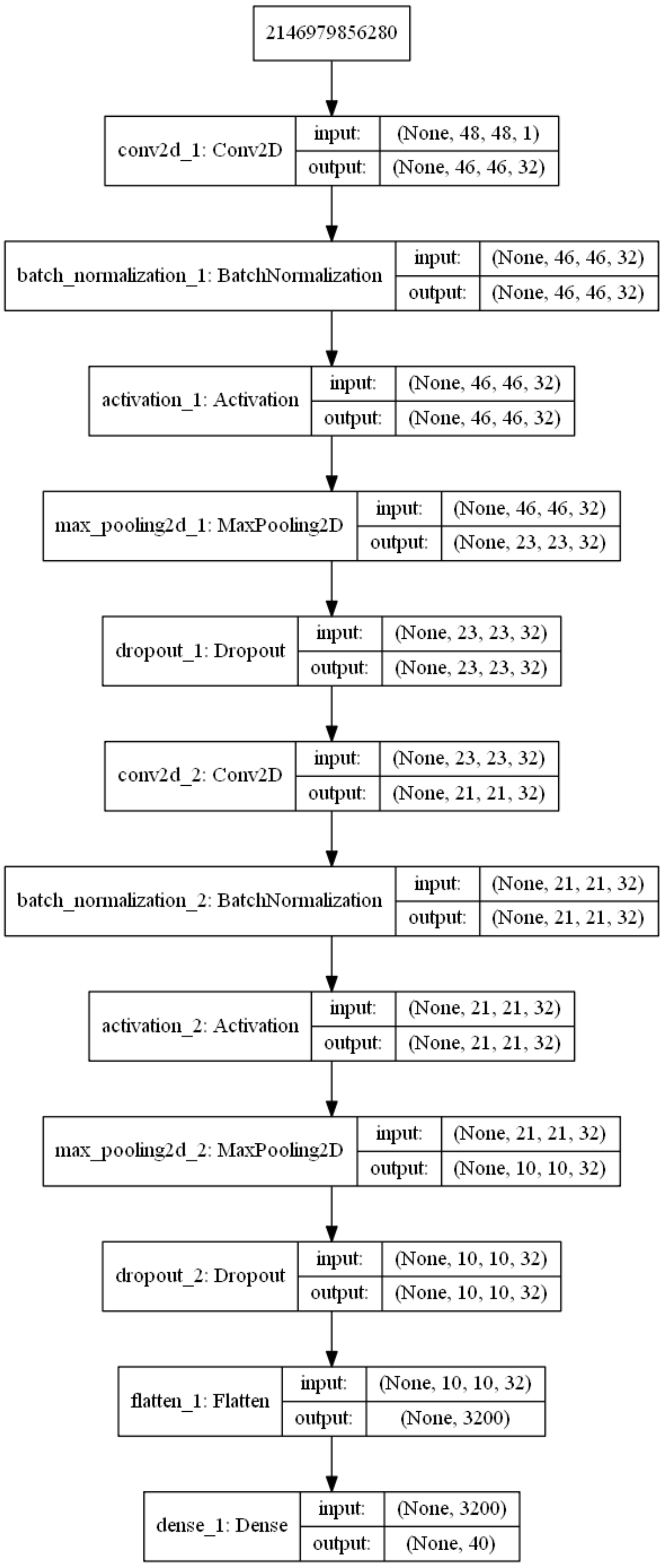

2146979856280
conv2d_1: Conv2D | input: | (None, 48, 48, 1) | output: | (None, 46, 46, 32)
batch_normalization_1: BatchNormalization | input: | (None, 46, 46, 32) | output: | (None, 46, 46, 32)
activation_1: Activation | input: | (None, 46, 46, 32) | output: | (None, 46, 46, 32)
max_pooling2d_1: MaxPooling2D | input: | (None, 46, 46, 32) | output: | (None, 23, 23, 32)
dropout_1: Dropout | input: | (None, 23, 23, 32) | output: | (None, 23, 23, 32)
conv2d_2: Conv2D | input: | (None, 23, 23, 32) | output: | (None, 21, 21, 32)
batch_normalization_2: BatchNormalization | input: | (None, 21, 21, 32) | output: | (None, 21, 21, 32)
activation_2: Activation | input: | (None, 21, 21, 32) | output: | (None, 21, 21, 32)
max_pooling2d_2: MaxPooling2D | input: | (None, 21, 21, 32) | output: | (None, 10, 10, 32)
dropout_2: Dropout | input: | (None, 10, 10, 32) | output: | (None, 10, 10, 32)
flatten_1: Flatten | input: | (None, 10, 10, 32) | output: | (None, 3200)
dense_1: Dense | input: | (None, 3200) | output: | (None, 40)

Figure 1. Neural Network structure, created with *Keras* `plot_model()`.

Training and Testing Accuracies

The training set and testing set were decided based on the splits provided by the dataset owner. The same set is used for testing and validation, although the validation is only done for informative purposes, i.e. it is not used to determine the number of epochs the model should train. The data is fed into the network through a data generator with batch size 64, where we make sure to shuffle if working with the training data. The optimizer used for training is SDG, as it gave the most promising results in addition to being the one proposed in assignment 4.

Figure 2 shows both the training and validation accuracies based on the number of epochs trained. As can be seen, the validation accuracy doesn't improve a whole lot after the first 10 epochs, although it spikes a bit in the 21st epoch. However, the validation accuracy doesn't appear to be actively decreasing with the increase of the training accuracy after this point - but if the model would have been trained until convergence of the training accuracy, the model would very likely have overfitted to the data. It is important to note that there is a slight error in the validation accuracy, as using a data generator with a batch size of 64, given that the data is not divisible with 64, the generator feeds additional data to reach the target size of 64 for the last batch - as the validation set includes 5532 instances, this means that the last batch will include 36 extra instances (that had already been validated before). Looking at the graph, it can be seen that even though the training accuracy keeps rising, the model overfits and does not generalize too well and performs much better on training data than testing data.

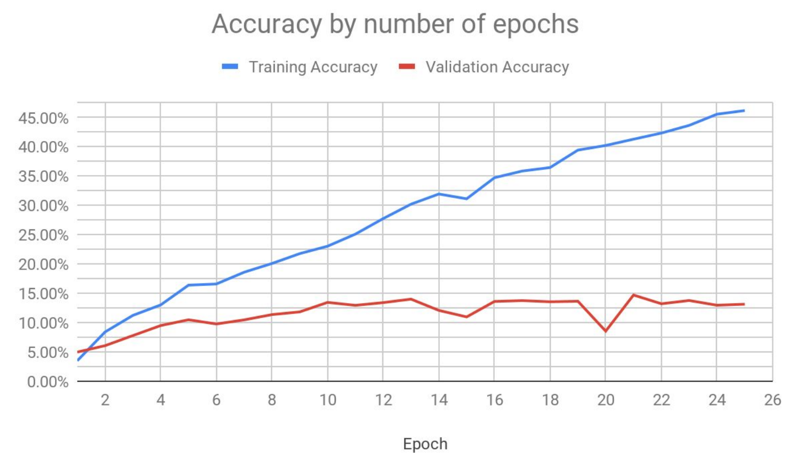


Figure 2. Training and Validation accuracies by number of epochs trained for part A.

Figure 3 shows the training and validation loss by number of epochs, it can be seen that the validation loss converges around the same epoch as the accuracy as was to be expected.

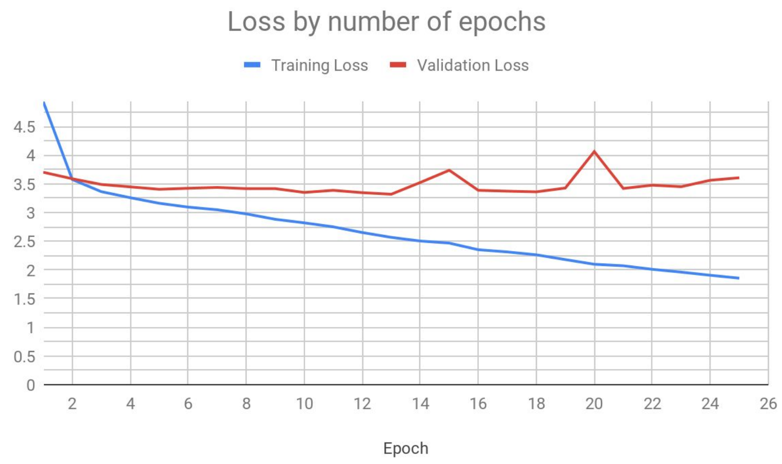
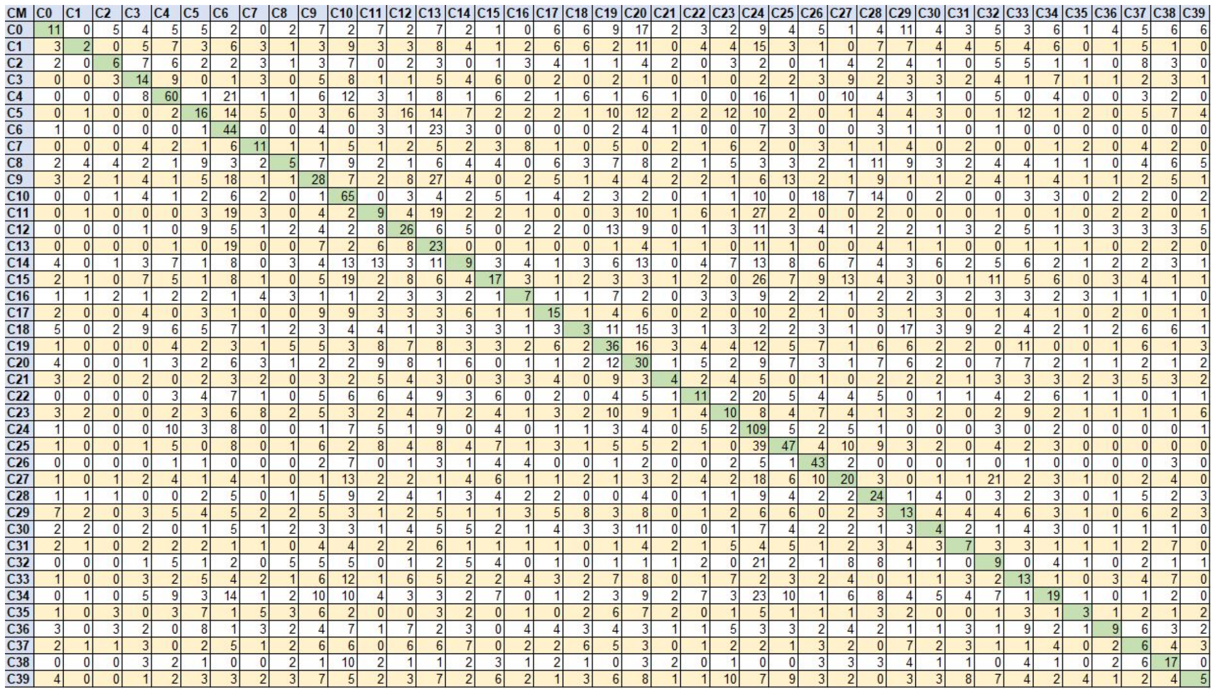


Figure 3. Training and Validation loss by number of epochs trained for part A.

Conclusions

Figure 4 shows the confusion matrix of the best structure (in terms of validation accuracy) found by the network after training for 25 epochs. The class labels from C0 to C39 denote each of the classes in alphabetical order, C0 therefore denotes applauding, C1 blowing bubbles and so forth. The matrix follows the conventional format of showing the actual class on the left side and the predicted class on top.

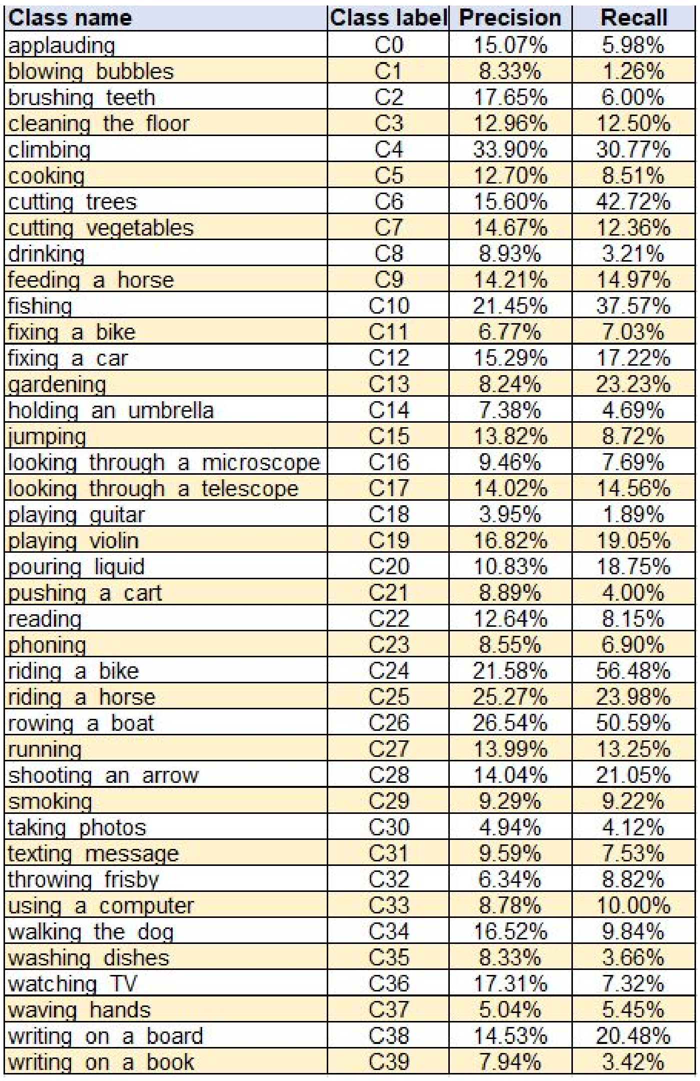


CM	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19
C0	11	0	5	4	5	5	2	0	2	7	2	7	2	7	2	1	0	6	6	9
C1	3	2	0	5	7	3	6	3	1	3	9	3	3	8	4	1	2	6	6	2
C2	2	0	6	7	6	2	2	3	1	3	7	0	2	3	0	1	3	4	1	1
C3	0	0	3	14	9	0	1	3	0	5	8	1	1	5	4	6	0	2	0	2
C4	0	0	0	8	60	1	21	1	1	6	12	3	1	8	1	6	2	1	6	1
C5	0	1	0	0	2	16	14	5	0	3	6	3	16	14	7	2	2	2	1	10
C6	1	0	0	0	0	1	44	0	0	4	0	3	1	23	3	0	0	0	0	2
C7	0	0	0	4	2	1	6	11	1	1	5	1	2	5	2	3	8	1	0	5
C8	2	4	4	2	1	9	3	2	5	7	9	2	1	6	4	4	0	6	3	7
C9	3	2	1	4	1	5	18	1	1	28	7	2	8	27	4	0	2	5	1	4
C10	0	0	1	4	1	2	6	2	0	1	65	0	3	4	2	5	1	4	2	3
C11	0	1	0	0	0	3	19	3	0	4	2	9	4	19	2	2	1	0	0	3
C12	0	0	0	1	0	9	5	1	2	4	2	8	26	6	5	0	2	2	0	13
C13	0	0	0	0	1	0	19	0	0	7	2	6	8	23	0	0	1	0	0	1
C14	4	0	1	3	7	1	8	0	3	4	13	13	3	11	9	3	4	1	3	6
C15	2	1	0	7	5	1	8	1	0	5	19	2	8	6	4	17	3	1	2	3
C16	1	1	2	1	2	2	1	4	3	1	1	2	3	3	2	1	7	1	1	7
C17	2	0	0	4	0	3	1	0	0	9	9	3	3	3	6	1	1	15	1	4
C18	5	0	2	9	6	5	7	1	2	3	4	4	1	3	3	3	1	3	3	11
C19	1	0	0	0	4	2	3	1	5	5	3	8	7	8	3	3	2	6	2	36
C20	4	0	0	1	3	2	6	3	1	2	2	9	8	1	6	0	1	1	2	12
C21	3	2	0	2	0	2	3	2	0	3	2	5	4	3	0	3	3	4	0	9
C22	0	0	0	0	3	4	7	1	0	5	6	6	4	9	3	6	0	2	0	4
C23	3	2	0	0	2	3	6	8	2	5	3	2	4	7	2	4	1	3	2	10
C24	1	0	0	0	10	3	8	0	0	1	7	5	1	9	0	4	0	1	1	3
C25	1	0	0	1	5	0	8	0	1	6	2	8	4	8	4	7	1	3	1	5
C26	0	0	0	0	1	1	0	0	0	2	7	0	1	3	1	4	4	0	0	1
C27	1	0	1	2	4	1	4	1	0	1	13	2	2	1	4	6	1	1	2	1
C28	1	1	1	0	0	2	5	0	1	5	9	2	4	1	3	4	2	2	0	0
C29	7	2	0	3	5	4	5	2	2	5	3	1	2	5	1	1	3	5	8	3
C30	2	2	0	2	0	1	5	1	2	3	3	1	4	5	5	2	1	4	3	3
C31	2	1	0	2	2	2	1	1	0	4	4	2	2	6	1	1	1	1	0	1
C32	0	0	0	1	5	1	2	0	5	5	5	0	1	2	5	4	0	1	0	1
C33	1	0	0	3	2	5	4	2	1	6	12	1	6	5	2	2	4	3	2	7
C34	0	1	0	5	9	3	14	1	2	10	10	4	3	3	2	7	0	1	2	3
C35	1	0	3	0	3	7	1	5	3	6	2	0	0	3	2	0	1	0	2	6
C36	3	0	3	2	0	8	1	3	2	4	7	1	7	2	3	0	4	4	3	4
C37	2	1	1	3	0	2	5	1	2	6	6	0	6	6	7	0	2	2	6	5
C38	0	0	0	3	2	1	0	0	2	1	10	2	1	1	2	3	1	2	1	0
C39	4	0	0	1	2	3	3	2	3	7	5	2	3	7	2	6	2	1	3	6

CM	C20	C21	C22	C23	C24	C25	C26	C27	C28	C29	C30	C31	C32	C33	C34	C35	C36	C37	C38	C39
C0	17	2	3	2	9	4	5	1	4	11	4	3	5	3	6	1	4	5	6	6
C1	11	0	4	4	15	3	1	0	7	7	4	4	5	4	6	0	1	5	1	0
C2	4	2	0	3	2	0	1	4	2	4	1	0	5	5	1	1	0	8	3	0
C3	1	0	1	0	2	2	3	9	2	3	3	2	4	1	7	1	1	2	3	1
C4	6	1	0	0	16	1	0	10	4	3	1	0	5	0	4	0	0	3	2	0
C5	12	2	2	12	10	2	0	1	4	4	3	0	1	12	1	2	0	5	7	4
C6	4	1	0	0	7	3	0	0	3	1	1	0	1	0	0	0	0	0	0	0
C7	0	2	1	6	2	0	3	1	1	4	0	2	0	0	1	2	0	4	2	0
C8	8	2	1	5	3	3	2	1	11	9	3	2	4	4	1	1	0	4	6	5
C9	4	2	2	1	6	13	2	1	9	1	1	2	4	1	4	1	1	2	5	1
C10	2	0	1	1	10	0	18	7	14	0	2	0	0	3	3	0	2	2	0	2
C11	10	1	6	1	27	2	0	0	2	0	0	0	1	0	1	0	2	2	0	1
C12	9	0	1	3	11	3	4	1	2	2	1	3	2	5	1	3	3	3	3	5
C13	4	1	1	0	11	1	0	0	4	1	1	0	0	1	1	1	0	2	2	0
C14	13	0	4	7	13	8	6	7	4	3	6	2	5	6	2	1	2	2	3	1
C15	3	1	2	0	26	7	9	13	4	3	0	1	11	5	6	0	3	4	1	1
C16	2	0	3	3	9	2	2	1	2	2	3	2	3	3	2	3	1	1	1	0
C17	6	0	2	0	10	2	1	0	3	1	3	0	1	4	1	0	2	0	1	1
C18	15	3	1	3	2	2	3	1	0	17	3	9	2	4	2	1	2	6	6	1
C19	16	3	4	4	12	5	7	1	6	6	2	2	0	11	0	0	1	6	1	3
C20	30	1	5	2	9	7	3	1	7	6	2	0	7	7	2	1	1	2	1	2
C21	3	4	2	4	5	0	1	0	2	2	2	1	3	3	3	2	3	5	3	2
C22	5	1	11	2	20	5	4	4	5	0	1	1	4	2	6	1	1	0	1	1
C23	9	1	4	10	8	4	7	4	1	3	2	0	2	9	2	1	1	1	1	6
C24	4	0	5	2	109	5	2	5	1	0	0	0	3	0	2	0	0	0	0	1
C25	5	2	1	0	39	47	4	10	9	3	2	0	4	2	3	0	0	0	0	0
C26	2	0	0	2	5	1	43	2	0	0	0	1	0	1	0	0	0	0	3	0
C27	3	2	4	2	18	6	10	20	3	0	1	1	21	2	3	1	0	2	4	0
C28	4	0	1	1	9	4	2	2	24	1	4	0	3	2	3	0	1	5	2	3
C29	8	0	1	2	6	6	0	2	3	13	4	4	4	6	3	1	0	6	2	3
C30	11	0	0	1	7	4	2	2	1	3	4	2	1	4	3	0	1	1	1	0
C31	4	2	1	5	4	5	1	2	3	4	3	7	3	3	1	1	1	2	7	0
C32	1	1	2	0	21	2	1	8	8	1	1	0	9	0	4	1	0	2	1	1
C33	8	0	1	7	2	3	2	4	0	1	1	3	2	13	1	0	3	4	7	0
C34	9	2	7	3	23	10	1	6	8	4	5	4	7	1	19	1	0	1	2	0
C35	7	2	0	1	5	1	1	1	3	2	0	0	1	3	1	3	1	2	1	2
C36	3	1	1	5	3	3	2	4	2	1	1	3	1	9	2	1	9	6	3	2
C37	3	0	1	2	2	1	3	2	0	7	2	3	1	1	4	0	2	6	4	3
C38	3	2	0	1	0	0	3	3	3	4	1	1	0	4	1	0	2	6	17	0
C39	8	1	1	10	7	9	3	2	0	3	3	8	7	4	2	4	1	2	4	5

Figure 4. Confusion matrix representing correct class predictions on the diagonal.

The accuracy of the model (sum of the diagonal divided by the total number of instances) is **14.64%**. From the figure it can be seen that the most correctly classified class is C24 (riding a bike) with 109 correctly classified instances. The second one is C10 (fishing), with 65 correctly classified instances, and the third is C4 (climbing). C6 (cutting trees), C25 (riding a horse) and C26 (rowing a boat) are all similar and notably higher than the rest of the classes. The notably least correctly classified class is C1 (blowing bubbles), with C18 (playing guitar) and C35 (washing dishes) being tied for the second to third least correctly classified. However, classes C2 (brushing teeth), C8 (drinking), C11 (fixing a bike), C14 (holding an umbrella), C16 (looking through a microscope), C21 (pushing a cart), C30 (taking photos), C31 (texting message), C32 (throwing frisby), C36 (watching TV), C37 (waving hands), and C39 (writing on a book) all have below 10 correctly classified instances and the model therefore obviously struggles with classifying these instances.



Class name	Class label	Precision	Recall
applauding	C0	15.07%	5.98%
blowing_bubbles	C1	8.33%	1.26%
brushing_teeth	C2	17.65%	6.00%
cleaning_the_floor	C3	12.96%	12.50%
climbing	C4	33.90%	30.77%
cooking	C5	12.70%	8.51%
cutting_trees	C6	15.60%	42.72%
cutting_vegetables	C7	14.67%	12.36%
drinking	C8	8.93%	3.21%
feeding_a_horse	C9	14.21%	14.97%
fishing	C10	21.45%	37.57%
fixing_a_bike	C11	6.77%	7.03%
fixing_a_car	C12	15.29%	17.22%
gardening	C13	8.24%	23.23%
holding_an_umbrella	C14	7.38%	4.69%
jumping	C15	13.82%	8.72%
looking_through_a_microscope	C16	9.46%	7.69%
looking_through_a_telescope	C17	14.02%	14.56%
playing_guitar	C18	3.95%	1.89%
playing_violin	C19	16.82%	19.05%
pouring_liquid	C20	10.83%	18.75%
pushing_a_cart	C21	8.89%	4.00%
reading	C22	12.64%	8.15%
phoning	C23	8.55%	6.90%
riding_a_bike	C24	21.58%	56.48%
riding_a_horse	C25	25.27%	23.98%
rowing_a_boat	C26	26.54%	50.59%
running	C27	13.99%	13.25%
shooting_an_arrow	C28	14.04%	21.05%
smoking	C29	9.29%	9.22%
taking_photos	C30	4.94%	4.12%
texting_message	C31	9.59%	7.53%
throwing_frisby	C32	6.34%	8.82%
using_a_computer	C33	8.78%	10.00%
walking_the_dog	C34	16.52%	9.84%
washing_dishes	C35	8.33%	3.66%
watching_TV	C36	17.31%	7.32%
waving_hands	C37	5.04%	5.45%
writing_on_a_board	C38	14.53%	20.48%
writing_on_a_book	C39	7.94%	3.42%

Figure 5. Precision and Recall per class.

The individual precision and recall of each class, along with the class name, can be seen in figure 5. In terms of recall, as expected, it can be seen that there is a high correlation between the number of most correctly classified instances and the recall of the corresponding class. It is notable that although C26 (rowing a boat) has only 43 correctly classified instances it has the second highest recall of 50.59%, indicating that the number of training samples for this class is slightly lower than that of some of the others. Apart from this, things are mostly as expected, with C24 (riding a bike) leading in terms of recall, and C1 (blowing bubbles) sitting at the bottom. The bad performance in terms of C1 is probably due to the bubbles themselves being lost during the rescaling of the images. In a similar manner, the success of C24 (riding a bike) and C26 (rowing a boat) might be due to the fact that these classes contain large objects that are still easily visible when the images have been rescaled.

In terms of precision, C4 (climbing) scores the highest with 33.90%. It can be seen that the precision of the top classes in terms of recall is quite significantly lower than their recall, the precision values indicate that the model often guesses that instances belong to these classes, even when they do not. All precisions are relatively low, which means that there is no class that the model is particularly good at distinguishing from all other classes.

II. TASK B: Using additional methods

Custom Learning Rate

Previously, the learning rate has been static at 0.01. When designing the custom learning rate function, the idea was to keep the learning rate relatively high at first to train quickly, but then decrease it over time to prevent overfitting and allow for a bit more exploration before convergence. The resulting equation can be seen here:

$$\eta = \gamma * e^{(-\kappa * \varepsilon)}$$

Where η is the new learning rate, γ is the initial learning rate, κ is the learning rate decrease rate, and ε is the number of the current epoch minus one (so the first epoch is zero, the second is one and so forth).

Due to the decreasing learning rate, the model does not converge as fast, and therefore more epochs were needed for training, that is why the number of epochs was increased for these experiments. After testing various values, the best pair of initial learning rate and decrease rate found was $\gamma = 0.0125$ and $\kappa = 0.035$. This guarantees starting the learning rate a bit high (25% higher than the static one from part A) and not decreasing it too much each epoch, as the network should train relatively fast for the first few epochs. A graph of the learning rate function with these values for a 50 epoch training session can be seen in Figure 6. The learning rate decay function implementation can be found in `expDecay()`.

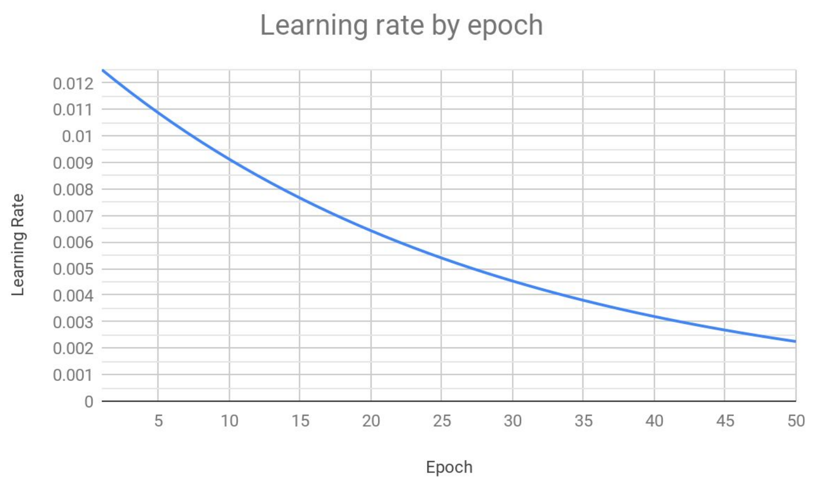


Figure 6. Custom exponential learning rate decay function with $\gamma = 0.0125$ and $\kappa = 0.035$.

The learning rate therefore decreases from 0.0125 to 0.00225 over the course of these 50 epochs. This results in a validation accuracy of around **15.71%**, an improvement of slightly over 1% from part A. Figure 7 shows the accuracy of this network over the whole training period, and Figure 8 shows the loss of the network over the same period. To train with the custom learning rate, `createBasicClassifier()` is called with the `custom_learning_rate` flag set to `True`.

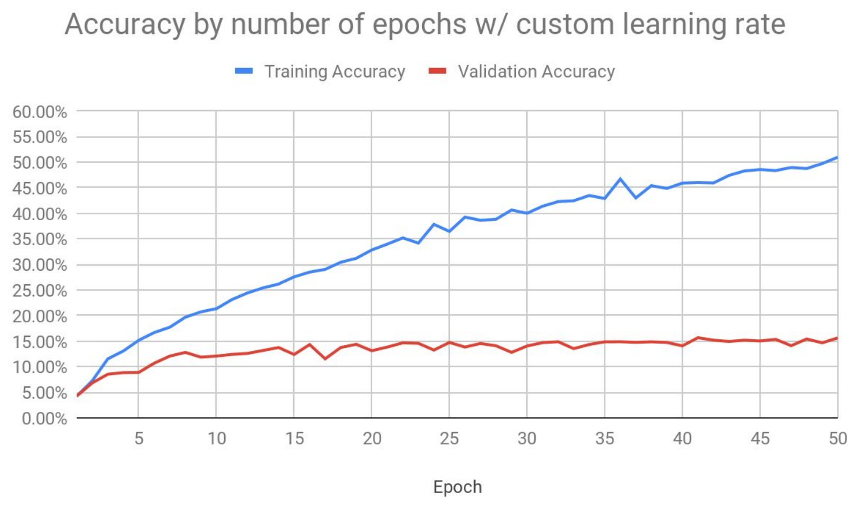


Figure 7. Training and Validation accuracies by number of epochs trained by applying custom learning rate decay function.

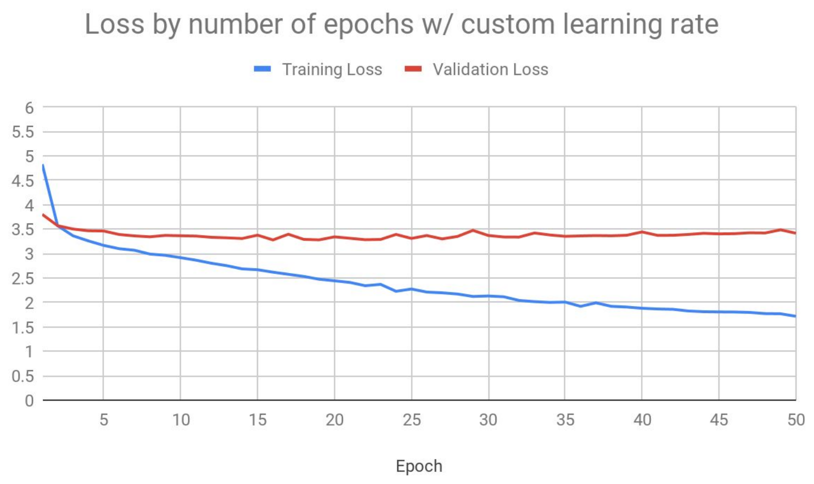


Figure 8. Training and Validation loss by number of epochs trained by applying custom learning rate decay function.

Weight Decay

Various parameter values of both *Lasso* regularization and *Ridge* regularization were tested on the model after applying the custom learning rate. The regularizations were applied to various combinations of the convolutional layers along with the fully connected layer. The configuration that gave the best results was simply applying it to all three layers. *Ridge* regularization turned out to work better than *Lasso*, and the best parameter value found was 0.01. Anything above this value turned out to be too destructive, while smaller values did not provide a whole lot of change. Adding the regularization managed to slightly reduce overfitting, resulting in a testing set accuracy increase of around 1%, i.e. around **16.79%**. Figure 9 shows the accuracy of this network over the whole training period, and Figure 10 shows the loss of the network over the same period. To train with the *Ridge* regularization, `createBasicClassifier()` is called with the `regulizer` flag set to `True`.

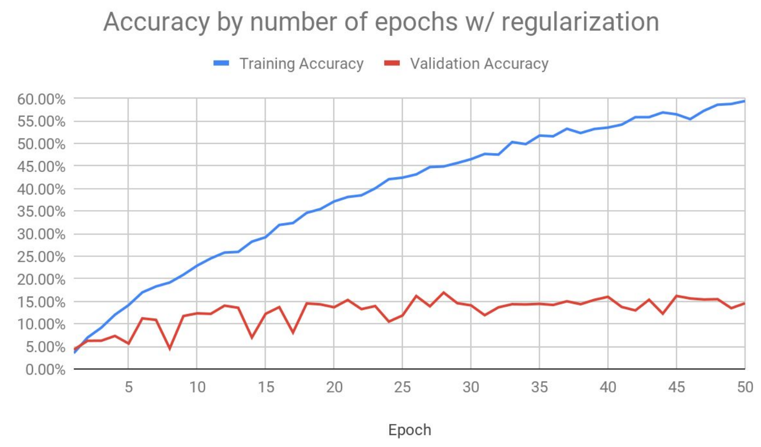


Figure 9. Training and validation accuracies by number of epochs trained with the *Ridge* regularization.

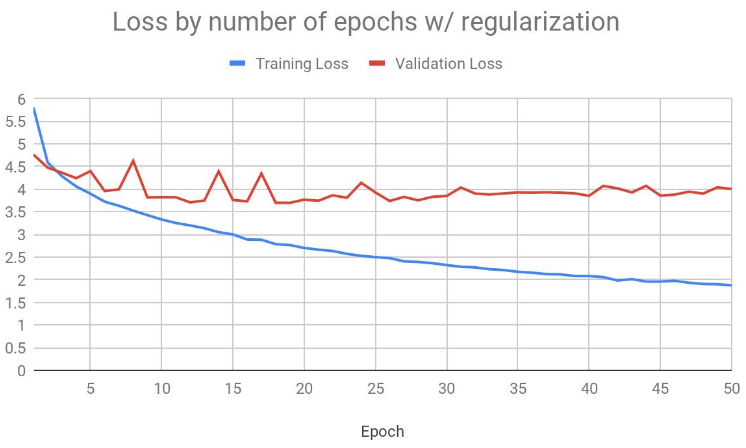


Figure 10. Training and validation loss by number of epochs trained with the *Ridge* regularization.

Transfer Learning

The network chosen was *VGG19*, as this network fits the architecture well by supporting small images of size 48x48. However, as the this network expects 3 channels, color had to be included for the network to function. Since this network has a lot of parameters it is not expected to be the most optimal one, but after trying to get various networks to work without success, *VGG19* finally did the trick for us. Loading the images in grayscale was therefore omitted from the preprocessing steps from this point onwards, and all images were loaded in color. In order for the network to function, a dense layer of size 40 (the total number of classes) had to be added at the end of the network such that it would be able to classify the data. The exponential learning rate and regularization did not give reliable improvements when tested on this part and were therefore excluded.

Running this configuration gave a validation accuracy of **22.85%**, over 6% more than from the previous network. It was also tested to add two additional dense layers before the last one, the former of size 512 (the number of parameters returned by the network) and the latter of size 256. This did not give any improvements in terms of validation accuracy and was therefore removed. Figure 11 shows the accuracy of this network over whole the training period of 50 epochs, and figure 12 shows the loss of the network over the same period.

An .h5 file was too large to submit, therefore we provide the statistics of the best epoch: [epoch: 46, training accuracy: 40.08%, training loss: 2.22, validation accuracy: 22.83%, validation loss: 2.91]. It is important to note that the validation accuracy differs slightly from the 22.85% measured from the confusion matrix due to the small bias introduced by the batch size. To train with transfer learning, `createPretrainedClassifier()` is called to initialize the classifier.

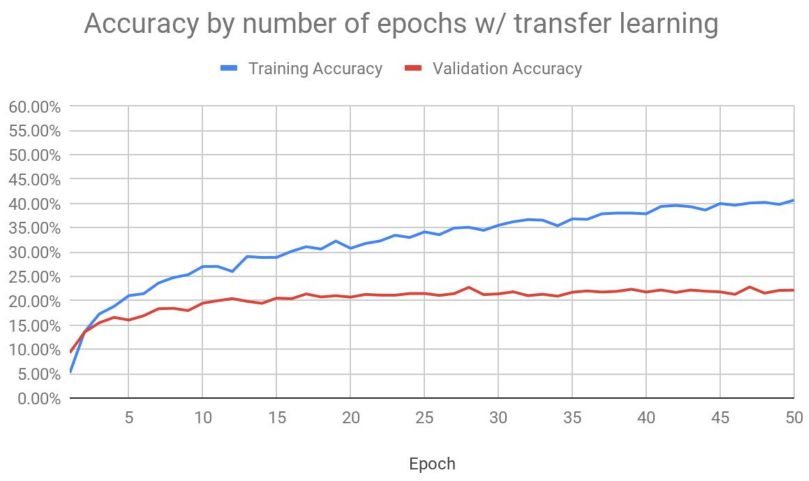


Figure 11. Training and validation accuracy by number of epochs trained with transfer learning.

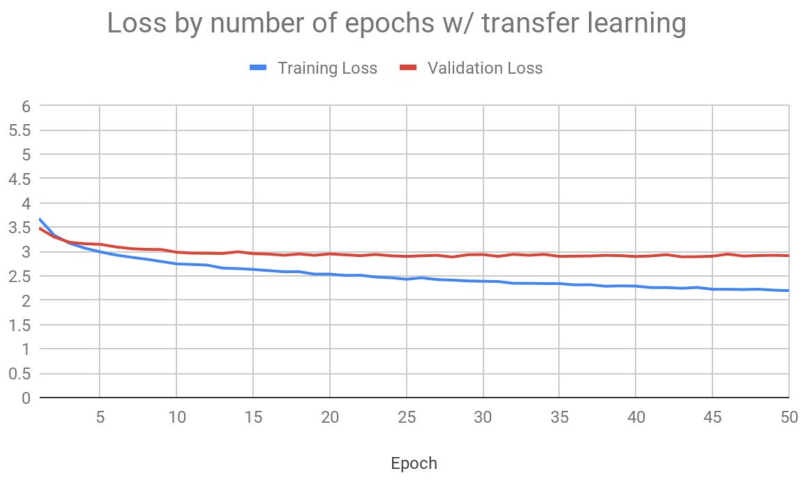


Figure 12. Training and validation loss by number of epochs trained with transfer learning.

Conclusions

A summary of the most important things to note per improvement in terms of why an improvement is observed has already been provided in each of the subsections. However, it has not yet been mentioned which changes these improvements have had on individual classes of the data.

The first improvement to consider is the custom learning rate. Appendix 1 shows the full confusion matrix for this improvement, and Appendix 2 shows precision and recall values for each class. This improvement increased the total network's validation accuracy by 1% to a total of **15.70%**. It is interesting to note that even though the overall accuracy is now higher, part A's most correctly classified class, C24 (riding a bike) had 109 correctly classified instances in part A, but only 69 with the custom learning rate. Much improvement can be seen when looking at instances incorrectly classified as C24 as well. This gives the impression that the new model is slightly more general, giving a more distributed classification and getting its accuracy from a larger distribution of classes. Looking at Appendix 2, and comparing to Figure 5, this appears to be the right assumption as most previously small classes have gained a small increase in accuracy, at the cost of the larger ones, with a few exceptions. The model therefore seems a bit more general in terms of which classes it works well for.

The second improvement is the *Ridge* regularization. Running the network with this improvement gave an additional accuracy increase of 1%, to a total of **16.79%**. Appendix 3 shows the full confusion matrix for this improvement, and Appendix 4 shows precision and recall values for each class. This one is quite interesting, as many of the previously best correctly classified classes from part B1 get even better recall after applying this improvement, but many of the previously worst classified classes also get better. However, some of the middle tier classes have taken a relatively hard hit. Despite this, there are many exceptions to these general trends. Overall, the model gains from this improvement, showing some overall improvements in terms of generalization from the training data to previously unseen data.

Finally, it is time to consider transfer learning. This method gave an accuracy of **22.84%**. As this method trains on a completely different network, vastly different results in terms of correctly classified instances were expected. This turned out to be completely true, comparing the results of Appendix 5 (the transfer learning network confusion matrix) with Appendix 3 (confusion matrix from part B2, i.e. after applying *Ridge* regularization) it can be seen that even though some classes appear to be universally more difficult to classify than others, some appear to be easier in general. For example, the previously best classified class, C24 (riding a bike) from part A and B1 has significantly fewer correctly classified instances when using transfer learning (56.48% recall in part A, 20.21% recall in part B3) as can be seen when comparing Appendix 6 (precision and recall per class with transfer learning) to other precision and recall values previously seen - while C4 (climbing) remains very high in both cases.

It should also be noted that since our architecture uses 48x48 images, getting a high accuracy is harder, as a lot of details are lost when re-scaling the images. It was also tested to run *MobileNetV2*, which has a requirement for images to be of size at least 96x96, as well as having significantly fewer parameters than *VGG19*. Doing so resulted in a validation accuracy of around 38.15%, which is quite the improvement. However, running this architecture took significantly longer, and it was therefore decided to stick with *VGG19* for further experiments.

III. TASK C: Finding the best architecture

Automatic model search

An algorithm was constructed to take the *VGG19* model and test different configurations of a fully connected layer at the end of it. The basic idea was to determine a number of values for a number of features and make the model search all possible permutations of these parameter values to find the best one. However, since trying all possible configurations is horribly time consuming, we wanted to make the algorithm a bit smarter, the final model turned out to be a bit more complex than originally planned.

The tested parameters, and their values, will be described in more detail in the motivation section, for now it is satisfactory to simply state that we used the following feature vector: [`learning_rate`, `regularization_value`, `amount_of_dense_layers`, `amount_of_nodes_per_layer[2]`]. The structure is the following: One for-loop for each feature, each of them running once for each of the possible values. Within the innermost loop, a while-loop trains the current configuration of the model (always train one epoch and then the algorithm makes a decision). This model is trained until no improvement is observed in the validation accuracy. This is observed in a combination of two conditions. If no improvement has been reached for a certain number of epochs in a row (called `break_threshold`), and the validation accuracy has decreased from the previous epoch, the algorithm breaks the loop.

Each time this happens, the weights of the current model are saved, and a new model is created with the next parameters in line using `reconfigureClassifier()`. The weights of this new model are initialized as the weights of the previous model. However, in some cases the whole architecture of the network changes, e.g. when the number of nodes in a layer are changed. When this happens, it is impossible to load the old weights (for a particular layer) due to the fact that the network structure differs. A result of this is that training will take a bit more time to reach decent accuracies again. Therefore, when this happens, the model is allowed to succeed the `break_threshold` by at least five steps, such that the model gets a bit more time to train.

The algorithm is also able to detect when tweaking a parameter is not yielding any improvements. When changing the parameter values, we start at the maximum value and each time the value is changed, it is decreased by a certain step size. If decreasing the parameter does not yield any improvements after being changed two times, the algorithm does not consider decreasing the parameter value any further and breaks out of the current for-loop. The reasoning behind this is that since the parameter values move in a single direction, if they are getting closer to an optimal configuration each run then they would show improvements for each parameter value along the way.

Finally, each time the algorithm finds a new best model, the model is once again allowed to succeed the `break_threshold` by an amount of at least five steps, as it is likely that this configuration has potential to give an even better model than has already been observed.

When experimentation started, we noticed a serious issue that we constantly ran out of GPU memory. This was a consequence of not clearing the *Keras* session and not collecting garbage immediately. The problem was solved by adding a small snippet of code to perform these tasks and keep GPU memory available for training a model with the new configuration.

Motivation

When choosing which features to consider, due to the limited time it was decided to use a static learning rate instead of the custom learning rate decay function. This is because the custom learning rate introduces two parameters instead of one, and each parameter greatly increases the number of permutations the model needs to consider. It is also important to note that to get the most out of loading the previous weights from a file, the different model configurations were placed in the outermost loops, as to maximize the number of consecutive runs with the same network architecture, as weights can only be loaded given that the network architecture is the same as during the previous run.

When choosing which values to consider for the learning rate, the baseline we decided to work around was the value of 0.0125, the same value as was used as initial learning rate in the custom learning rate decay function. A step size of 0.0025 per iteration was incorporated, with 3 considered steps in each direction, so a total of 7 steps including the baseline, as this allowed for exploring values between 0.02 and 0.005. This seemed to us like an interesting interval, but better results were expected for the lower end of the spectrum.

A similar process was used to decide on the regularization values. The baseline of 0.01 was chosen as this has been measured to work best for the network tested in part B2. A step size of 0.0025 was used as it was believed to be neither too big nor too small. Finally, similarly to the learning rate, the number of steps chosen was 7, and the values considered therefore between 0.0175 and 0.0025.

Three layer configurations were examined, the first including a single 40 node prediction layer, the second having the former along with another dense layer before it, and the third with two dense layers before the prediction layer. Any more layers were not expected to yield a great improvement, and due to limited time it was decided not to pursue confirming those expectations. When deciding on possible sizes for these extra layers, the maximum considered value for the first layer was decided to be the same as the number of output nodes from the *VGG19* network, namely 512. From there on it was decided that it should decrease by 128 nodes per tested configuration, with a total of 4 tested configurations. I.e. the tested configurations for this layer is 512 nodes, 384 nodes, 256 nodes, and 128 nodes. This way, three out of the four tested configurations would be a power of two, possibly giving a slight speed increase. For the second layer, it was decided to test the values from 256 to 64, with a step size of 64. I.e. a size of 256 nodes, 192 nodes, 128 nodes and finally 64 nodes. We wanted the number of features to decrease over time, and so the maximum value for this layer had to be smaller than that of the layer before.

For each of the parameters, the initial value is set to be the maximum value in the spectrum, and every time a parameter is changed, it is simply decreased by the amount of the step size. This has the added benefit of new model configurations starting by testing the higher learning rates before moving down to lower ones.

This results in 7 possible learning rate configurations, 7 possible regularization configurations, 3 possible layer architectures, 4 possible configurations of nodes for the first extra layer, and 4 possible configurations for the second extra layer. In total, that allows for a total of 2352 possible configurations, including some redundant layer configurations, due to the fact that the number of nodes in the extra layers are irrelevant when they are not included in the architecture, and the latter calculations are therefore excluded explicitly.

The threshold to break the inner while-loop is set to 2 epochs without improvement (with some exceptions explained above). This was chosen as we do not want to overcommit with training for too long which would result in an overfitted model. Consequently, loaded model weights would not benefit from additional training. The 5 extra epochs were chosen for specific cases when we want to train longer since we believe that this is high enough rate to encounter improvement if it is present.

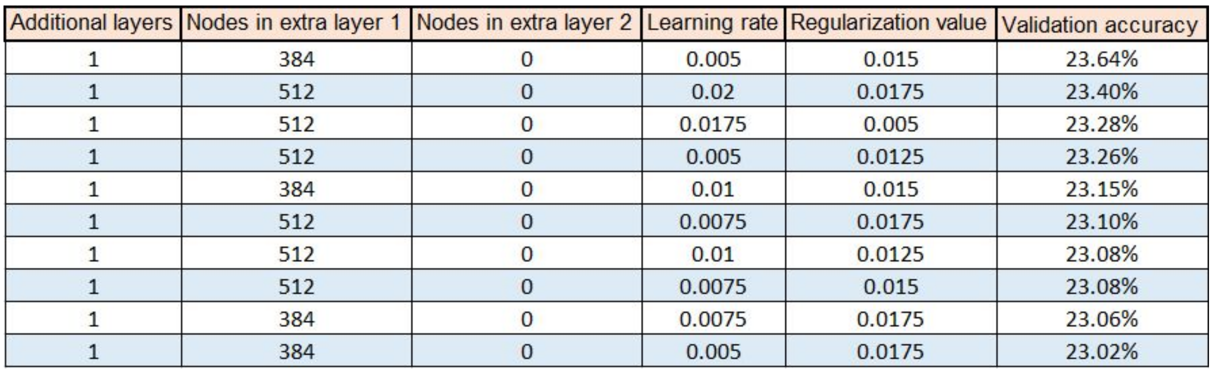
Results & Conclusions

We can definitely state that our implemented method works and performs positively since we were able to test out multiple different feature vectors and find a model which has beaten the one discussed in the transfer learning section of part B.

The best model found by the method had an accuracy of **23.64%**, which is around 0.8% higher than what was found in part B3. This model only had an additional dense layer of size 384 nodes before the prediction layer, a learning rate of 0.015 and a regularization value of 0.0175. This structure was surprising to us on two accounts. Adding an additional dense layer had not shown any improvements in our previous experiments, and we therefore had expected the best structure to include no extra layers. Additionally, our intuition told us that the best additional layer to include would have the same number of nodes as the output of the layer before it, which in this case was false, as the layer before had an output of 512 while the layer size was 384. In terms of learning rate, we suspected that it would be low, which it was. The regularization value was a bit higher than what we suspected, as we thought might be a bit too destructive. However, it appears that instead it might have helped by slightly reducing overfitting. The full confusion matrix of the model can be seen in Appendix 7. Additionally, Appendix 8 includes the precision and recall per class. We have also saved the best model into a .h5 file, we have not submitted it as it is too large, but we can provide it on demand.

This was not the only configuration found to be better than those of our previous experiments. In fact, a total of 10 configurations scored a validation accuracy of over **23%**

(22.85% was the accuracy of the best model from part B3). Notably, much to our surprise, every single one of them had an additional dense layer before the prediction layer. The configurations can be seen in Figure 15. As can be seen in the figure, a vast number of different learning rates can be seen, although most of them are in the lower end of the spectrum, as we expected. The regularization values were almost all in the higher end of the spectrum, which surprised us as we thought that might be too destructive. 6 out of the 10 configurations have 512 nodes in the extra layer, but the other 4, including the best one, have 384.



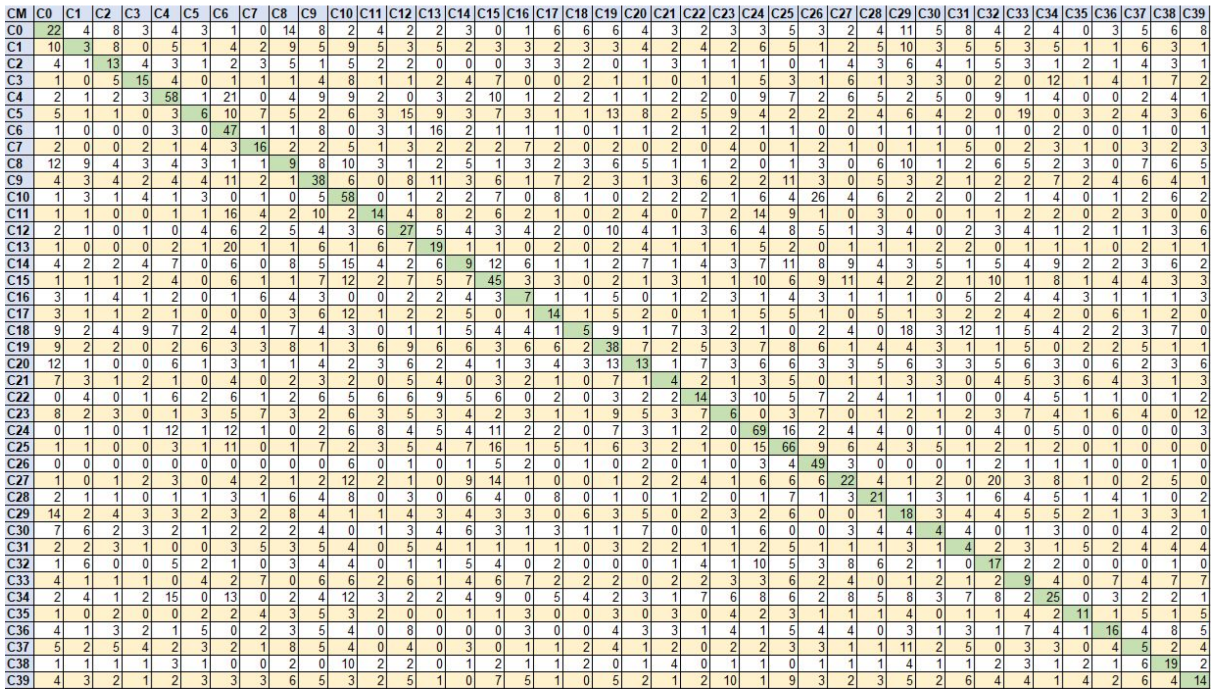
Additional layers	Nodes in extra layer 1	Nodes in extra layer 2	Learning rate	Regularization value	Validation accuracy
1	384	0	0.005	0.015	23.64%
1	512	0	0.02	0.0175	23.40%
1	512	0	0.0175	0.005	23.28%
1	512	0	0.005	0.0125	23.26%
1	384	0	0.01	0.015	23.15%
1	512	0	0.0075	0.0175	23.10%
1	512	0	0.01	0.0125	23.08%
1	512	0	0.0075	0.015	23.08%
1	384	0	0.0075	0.0175	23.06%
1	384	0	0.005	0.0175	23.02%

Figure 15. A table of all models found by the automatic model search scoring an accuracy of over 23%. As none of the configurations had two additional layers, the number of nodes for the second extra layer has been set to 0.

The algorithm did provide some very interesting results. However, there is a lot of room for improvement. First, the number of tested parameter values is rather low. Testing a larger number of values for each of the parameters would give a better indication of what is good and what isn't. Secondly, additional parameters could be tested, for example different layer types, *Lasso* regularization, a custom learning rate decay function and so forth. Thirdly, the time we had for searching was limited, which resulted in a perhaps lower than optimal number of epochs per configuration. The number of redundant permutations could be further reduced as well. Finally, due to time constraints, the algorithm might have been configured to skip a bit too many configurations. However, this is also what allowed us to explore a vast amount of more different configurations in a short amount of time. A possible change for a more even distribution of tested values could be to start every other iteration of the parameters in reverse order. E.g. for one iteration the first tested learning rate would be the largest one in the spectrum, and for each subsequent value it would decrease until reaching the bottom of the spectrum (top-down), and for the next iteration it could be the other way around (bottom-up).

IV. APPENDIX: Supporting information

Extra content that can be used to further verify claims.



CM	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23	C24	C25	C26	C27	C28	C29	C30	C31	C32	C33	C34	C35	C36	C37	C38	C39
C0	22	4	8	3	4	3	1	0	14	8	2	4	2	2	3	0	1	6	6	6	4	3	2	3	3	5	3	2	4	11	5	8	4	2	4	0	3	5	6	8
C1	10	3	8	0	5	1	4	2	9	5	9	5	3	5	2	3	3	2	3	3	4	2	4	2	6	5	1	2	5	10	3	5	5	3	5	1	1	6	3	1
C2	4	1	13	4	3	1	2	3	5	1	5	2	2	0	0	0	3	3	2	0	1	3	1	1	1	0	1	4	3	6	4	1	5	3	1	2	1	4	3	1
C3	1	0	5	15	4	0	1	1	1	4	8	1	1	2	4	7	0	0	2	1	1	0	1	1	5	3	1	6	1	3	3	0	2	0	12	1	4	1	7	2
C4	2	1	2	3	58	1	21	0	4	9	9	2	0	3	2	10	1	2	2	1	1	2	2	0	9	7	2	6	5	2	5	0	9	1	4	0	0	2	4	1
C5	5	1	1	0	3	6	10	7	5	2	6	3	15	9	3	7	3	1	1	13	8	2	5	9	4	2	2	2	4	6	4	2	0	19	0	3	2	4	3	6
C6	1	0	0	0	3	0	47	1	1	8	0	3	1	16	2	1	1	1	0	1	1	2	1	2	1	1	0	0	1	1	1	0	1	0	2	0	0	1	0	1
C7	2	0	0	2	1	4	3	16	2	2	5	1	3	2	2	2	7	2	0	2	0	2	0	4	0	1	2	1	0	1	1	5	0	2	3	1	0	3	2	3
C8	12	9	4	3	4	3	1	1	9	8	10	3	1	2	5	1	3	2	3	6	5	1	1	2	0	1	3	0	6	10	1	2	6	5	2	3	0	7	6	5
C9	4	3	4	2	4	4	11	2	1	38	6	0	8	11	3	6	1	7	2	3	1	3	6	2	2	11	3	0	5	3	2	1	2	2	7	2	4	6	4	1
C10	1	3	1	4	1	3	0	1	0	5	58	0	1	2	2	7	0	8	1	0	2	2	2	1	6	4	26	4	6	2	2	0	2	1	4	0	1	2	6	2
C11	1	1	0	0	1	1	16	4	2	10	2	14	4	8	2	6	2	1	0	2	4	0	7	2	14	9	1	0	3	0	0	1	1	2	2	0	2	3	0	0
C12	2	1	0	1	0	4	6	2	5	4	3	6	27	5	4	3	4	2	0	10	4	1	3	6	4	8	5	1	3	4	0	2	3	4	1	2	1	1	3	6
C13	1	0	0	0	2	1	20	1	1	6	1	6	7	19	1	1	0	2	0	2	4	1	1	1	5	2	0	1	1	1	2	2	0	1	1	1	0	2	1	1
C14	4	2	2	4	7	0	6	0	8	5	15	4	2	6	9	12	6	1	1	2	7	1	4	3	7	11	8	9	4	3	5	1	5	4	9	2	2	3	6	2
C15	1	1	1	2	4	0	6	1	1	7	12	2	7	5	7	45	3	3	0	2	1	3	1	1	10	6	9	11	4	2	2	1	10	1	8	1	4	4	3	3
C16	3	1	4	1	2	0	1	6	4	3	0	0	2	2	4	3	7	1	1	5	0	1	2	3	1	4	3	1	1	1	0	5	2	4	4	3	1	1	1	3
C17	3	1	1	2	1	0	0	0	3	6	12	1	2	2	5	0	1	14	1	5	2	0	1	1	5	5	1	0	5	1	3	2	2	4	2	0	6	1	2	0
C18	9	2	4	9	7	2	4	1	7	4	3	0	1	1	5	4	4	1	5	9	1	7	3	2	1	0	2	4	0	18	3	12	1	5	4	2	2	3	7	0
C19	9	2	2	0	2	6	3	3	8	1	3	6	9	6	6	3	6	6	2	38	7	2	5	3	7	8	6	1	4	4	3	1	1	5	0	2	2	5	1	1
C20	12	1	0	0	6	1	3	1	1	4	2	3	6	2	4	1	3	4	3	13	13	1	7	3	6	6	3	3	5	6	3	3	5	6	3	0	6	2	3	6
C21	7	3	1	2	1	0	4	0	2	3	2	0	5	4	0	3	2	1	0	7	1	4	2	1	3	5	0	1	1	3	3	0	4	5	3	6	4	3	1	3
C22	0	4	0	1	6	2	6	1	2	6	5	6	6	9	5	6	0	2	0	3	2	2	14	3	10	5	7	2	4	1	1	0	0	4	5	1	1	0	1	2
C23	8	2	3	0	1	3	5	7	3	2	6	3	5	3	4	2	3	1	1	9	5	3	7	6	0	3	7	0	1	2	1	2	3	7	4	1	6	4	0	12
C24	0	1	0	1	12	1	12	1	0	2	6	8	4	5	4	11	2	2	0	7	3	1	2	0	69	16	2	4	4	0	1	0	4	0	5	0	0	0	0	3
C25	1	1	0	0	3	1	11	0	1	7	2	3	5	4	7	16	1	5	1	6	3	2	1	0	15	66	9	6	4	3	5	1	2	1	2	0	1	0	0	0
C26	0	0	0	0	0	0	0	0	0	0	6	0	1	0	1	5	2	0	1	0	2	0	1	0	3	4	49	3	0	0	0	1	2	1	1	1	0	0	1	0
C27	1	0	1	2	3	0	4	2	1	2	12	2	1	0	9	14	1	0	0	1	2	2	4	1	6	6	6	22	4	1	2	0	20	3	8	1	0	2	5	0
C28	2	1	1	0	1	1	3	1	6	4	8	0	3	0	6	4	0	8	0	1	0	1	2	0	1	7	1	3	21	1	3	1	6	4	5	1	4	1	0	2
C29	14	2	4	3	3	2	3	2	8	4	1	1	4	3	4	3	3	0	6	3	5	0	2	3	2	6	0	0	1	18	3	4	4	5	5	2	1	3	3	1
C30	7	6	2	3	2	1	2	2	2	4	0	1	3	4	6	3	1	3	1	1	7	0	0	1	6	0	0	3	4	4	4	4	0	1	3	0	0	4	2	0
C31	2	2	3	1	0	0	3	5	3	5	4	0	5	4	1	1	1	1	0	3	2	2	1	1	2	5	1	1	1	3	1	4	2	3	1	5	2	4	4	4
C32	1	6	0	0	5	2	1	0	3	4	4	0	1	1	5	4	0	2	0	0	0	1	4	1	10	5	3	8	6	2	1	0	17	2	2	0	0	0	1	0
C33	4	1	1	1	0	4	2	7	0	6	6	2	6	1	4	6	7	2	2	2	0	2	2	3	3	6	2	4	0	1	2	1	2	9	4	0	7	4	7	7
C34	2	4	1	2	15	0	13	0	2	4	12	3	2	2	4	9	0	5	4	2	3	1	7	6	8	6	2	8	5	8	3	7	8	2	25	0	3	2	2	1
C35	1	0	2	0	0	2	2	4	3	5	3	2	0	0	1	1	3	0	0	3	0	3	0	4	2	3	1	1	1	4	0	1	1	4	2	11	1	5	1	5
C36	4	1	3	2	1	5	0	2	3	5	4	0	8	0	0	0	3	0	0	4	3	3	1	4	1	5	4	4	0	3	1	3	1	7	4	1	16	4	8	5
C37	5	2	5	4	2	3	2	1	8	5	4	0	4	0	3	0	1	1	2	4	1	2	0	2	2	3	3	1	1	11	2	5	0	3	3	0	4	5	2	4
C38	1	1	1	1	3	1	0	0	2	0	10	2	2	0	1	2	1	1	2	0	1	4	0	1	1	0	1	1	1	4	1	1	2	3	1	2	1	6	19	2
C39	4	3	2	1	2	3	3	3	6	5	3	2	5	1	0	7	5	1	0	5	2	1	2	10	1	9	3	2	3	5	2	6	4	4	1	4	2	6	4	14

Appendix 1. Full confusion matrix from part B1 from network trained with custom learning rate function for 50 epochs.

Class name	Class label	Precision	Recall
applauding	C0	12.72%	11.96%
blowing_bubbles	C1	3.90%	1.89%
brushing_teeth	C2	14.44%	13.00%
cleaning_the_floor	C3	18.99%	13.39%
climbing	C4	31.87%	29.74%
cooking	C5	8.33%	3.19%
cutting_trees	C6	19.42%	45.63%
cutting_vegetables	C7	17.58%	17.98%
drinking	C8	6.16%	5.77%
feeding_a_horse	C9	17.84%	20.32%
fishing	C10	21.56%	33.53%
fixing_a_bike	C11	13.86%	10.94%
fixing_a_car	C12	15.52%	17.88%
gardening	C13	12.58%	19.19%
holding_an_umbrella	C14	6.43%	4.69%
jumping	C15	20.55%	23.08%
looking_through_a_microscope	C16	7.37%	7.69%
looking_through_a_telescope	C17	13.46%	13.59%
playing_guitar	C18	9.09%	3.14%
playing_violin	C19	20.54%	20.11%
pouring_liquid	C20	11.50%	8.13%
pushing_a_cart	C21	5.48%	4.00%
reading	C22	12.61%	10.37%
phoning	C23	6.06%	4.14%
riding_a_bike	C24	28.51%	35.75%
riding_a_horse	C25	25.48%	33.67%
rowing_a_boat	C26	26.78%	57.65%
running	C27	16.67%	14.57%
shooting_an_arrow	C28	15.91%	18.42%
smoking	C29	10.65%	12.77%
taking_photos	C30	4.55%	4.12%
texting_message	C31	4.21%	4.30%
throwing_frisby	C32	11.49%	16.67%
using_a_computer	C33	6.34%	6.92%
walking_the_dog	C34	15.43%	12.95%
washing_dishes	C35	17.74%	13.41%
watching_TV	C36	16.84%	13.01%
waving_hands	C37	4.20%	4.55%
writing_on_a_board	C38	14.39%	22.89%
writing_on_a_book	C39	11.86%	9.59%

Appendix 2. Precision and recall values per class from part B1 from network trained with custom learning rate function for 50 epochs.

CM	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23	C24	C25	C26	C27	C28	C29	C30	C31	C32	C33	C34	C35	C36	C37	C38	C39
C0	16	1	5	3	6	2	1	0	2	8	4	8	3	1	0	2	1	5	10	7	13	1	2	8	5	3	3	0	3	16	3	3	4	4	4	0	11	7	5	4
C1	5	3	4	1	9	2	2	2	0	10	13	5	9	1	0	3	2	4	6	2	9	0	2	3	10	6	3	2	6	11	3	0	4	3	2	0	4	5	3	0
C2	3	1	14	2	7	2	3	0	2	1	9	1	2	0	0	0	4	1	4	0	4	0	0	3	2	0	1	1	0	5	3	0	4	6	0	1	5	6	3	0
C3	0	0	5	12	11	0	1	0	0	4	10	2	2	3	1	8	0	2	1	1	0	0	3	2	3	1	3	6	2	5	3	1	2	0	7	1	4	1	5	0
C4	4	0	0	3	88	0	12	1	2	5	14	1	2	4	0	7	1	0	3	0	4	1	0	0	12	0	7	9	2	2	1	0	1	1	2	0	0	3	2	1
C5	3	0	0	0	3	14	5	5	2	2	10	6	21	7	1	3	2	0	4	15	10	0	2	19	3	2	2	1	3	5	2	0	0	14	0	4	3	6	5	4
C6	0	0	0	0	6	1	35	0	0	10	0	8	3	9	1	0	0	0	1	3	4	0	0	3	6	2	3	1	2	0	1	0	1	1	0	0	1	1	0	0
C7	3	2	1	1	1	5	0	5	4	5	5	1	4	3	0	3	2	1	2	6	0	2	1	7	0	0	3	1	0	3	1	0	0	2	0	4	3	4	1	3
C8	10	8	5	1	4	1	0	0	4	12	14	4	5	1	1	3	2	3	3	4	5	2	5	3	3	0	4	0	3	7	1	0	2	10	3	2	8	8	4	1
C9	3	1	1	3	5	5	4	0	0	40	9	6	9	4	0	4	1	8	5	1	5	1	5	2	4	10	4	5	7	1	1	0	2	4	2	5	9	6	5	0
C10	1	0	1	1	4	1	2	0	0	5	79	4	7	1	0	5	0	6	2	0	0	0	3	5	3	0	24	1	3	0	3	0	1	1	0	0	4	2	1	3
C11	3	1	0	0	1	1	10	2	0	7	5	18	8	5	1	5	3	0	1	2	5	0	7	3	21	5	3	0	2	0	0	0	1	2	1	0	2	3	0	0
C12	1	0	1	0	1	7	2	1	2	3	3	9	34	4	1	1	0	3	2	7	6	0	4	8	3	6	10	1	2	4	0	0	2	6	0	1	4	6	1	5
C13	1	0	0	0	1	3	11	1	0	5	5	12	9	12	0	1	0	1	4	0	4	2	1	1	7	3	2	1	1	2	2	0	0	0	1	0	1	3	1	1
C14	4	2	2	3	10	1	2	0	3	9	19	5	7	4	6	7	3	2	4	3	13	1	5	4	9	9	6	9	3	4	7	0	2	4	6	0	6	5	2	1
C15	0	1	1	4	10	1	4	0	0	4	21	4	10	2	1	35	3	2	4	3	3	1	3	2	11	10	16	8	1	1	5	1	3	4	5	1	6	1	2	1
C16	1	1	3	0	1	0	0	3	2	4	0	2	6	0	0	1	12	1	3	5	2	0	2	5	3	2	6	0	2	1	2	1	1	7	0	4	3	4	1	0
C17	1	1	0	0	3	1	0	0	0	5	13	3	2	2	1	0	3	13	3	3	6	0	4	0	6	1	4	0	4	0	3	1	1	6	1	0	5	5	1	1
C18	11	5	5	9	11	3	0	0	1	2	5	1	1	1	3	2	0	2	14	4	8	6	2	6	2	1	2	2	0	10	3	3	0	6	1	1	9	10	6	1
C19	7	1	1	0	7	2	3	1	4	3	5	3	18	3	1	4	1	5	5	34	10	2	5	5	11	2	11	1	4	4	4	0	0	5	0	1	9	6	1	0
C20	7	0	0	0	4	0	2	1	0	2	4	4	8	3	2	2	1	2	4	9	30	1	5	2	6	7	4	2	6	8	4	0	2	10	2	3	8	1	2	2
C21	5	0	1	1	5	3	1	1	1	2	3	2	7	2	0	2	2	3	0	6	2	5	1	4	6	1	2	2	1	0	1	0	1	3	1	6	5	7	1	4
C22	1	3	1	1	9	4	2	1	0	8	6	9	5	4	1	9	0	3	2	1	8	1	14	3	11	3	8	1	1	1	1	0	3	6	0	1	2	0	1	0
C23	3	2	2	1	2	4	3	3	0	1	8	1	11	3	0	4	2	3	4	12	5	1	9	16	2	2	9	3	4	1	1	0	0	11	0	3	4	1	2	2
C24	0	0	0	0	14	2	8	0	0	2	9	9	8	1	0	6	0	0	1	8	5	0	1	1	87	7	4	7	3	1	1	0	4	2	1	0	1	0	0	0
C25	1	2	0	1	9	1	3	0	1	6	5	6	8	3	1	13	0	4	2	5	8	1	2	1	18	62	8	6	8	3	3	0	1	1	1	0	1	1	0	0
C26	0	1	0	0	3	2	0	0	0	1	7	0	1	0	0	2	1	0	0	0	0	0	0	1	2	2	55	2	0	0	0	0	0	1	1	0	1	0	1	1
C27	2	1	0	1	10	0	3	1	1	3	15	1	2	0	1	12	0	0	1	1	1	1	4	4	12	7	10	26	5	1	2	0	15	1	1	0	3	0	3	0
C28	4	1	2	0	7	1	0	0	1	6	13	0	6	1	2	6	1	5	1	0	2	0	2	0	5	4	5	4	15	1	2	0	1	4	2	0	5	3	1	1
C29	9	5	3	1	6	2	3	0	0	4	2	1	4	1	1	1	2	2	11	2	9	0	2	6	5	4	1	0	1	18	3	0	2	7	3	4	9	5	2	0
C30	6	4	0	0	8	3	1	1	2	4	2	1	5	2	3	2	0	2	5	0	7	0	2	3	5	2	3	1	3	3	4	1	1	2	0	0	2	5	2	0
C31	4	1	2	1	1	1	0	1	0	3	7	0	3	2	1	1	2	3	2	2	4	0	1	8	5	4	0	0	3	4	1	2	1	7	1	5	6	2	2	0
C32	2	2	1	0	10	1	1	0	0	2	11	1	3	0	2	4	0	3	0	0	3	1	3	2	11	4	4	8	6	1	0	0	10	2	0	1	3	0	0	0
C33	0	0	2	3	1	6	3	1	0	8	12	4	7	0	1	3	1	3	3	3	1	1	2	6	3	3	6	2	1	0	3	0	0	15	0	2	11	3	8	2
C34	2	1	1	3	19	2	3	1	0	10	15	5	4	3	1	10	1	3	7	3	6	1	6	5	10	10	3	10	3	8	5	1	7	1	18	2	0	3	0	0
C35	0	0	3	0	2	3	1	1	1	4	2	0	2	2	0	1	0	1	1	3	5	1	0	6	3	1	5	2	1	1	0	0	1	6	1	8	3	6	1	4
C36	7	0	3	1	1	2	0	2	0	4	5	0	8	0	0	1	0	2	4	3	5	0	1	4	2	1	5	2	0	2	2	1	1	10	2	3	27	6	3	3
C37	8	3	3	3	2	4	1	0	3	4	9	3	4	0	0	1	0	1	5	2	4	0	5	1	0	1	2	0	0	9	3	2	0	5	1	3	7	5	3	3
C38	0	0	2	2	3	1	0	0	1	0	12	3	1	0	0	2	0	4	3	0	1	1	0	2	0	0	4	2	0	3	0	0	1	2	0	0	9	7	15	2
C39	4	0	2	2	3	3	3	1	2	6	6	4	4	3	1	5	2	1	3	4	3	0	3	16	4	6	4	0	1	2	3	2	5	8	2	5	8	5	1	9

Appendix 3. Full confusion matrix from part B2 from network trained with *Ridge* regularization for 50 epochs.

Class name	Class label	Precision	Recall
applauding	C0	11.27%	8.70%
blowing_bubbles	C1	5.56%	1.89%
brushing_teeth	C2	18.18%	14.00%
cleaning_the_floor	C3	18.75%	10.71%
climbing	C4	28.57%	45.13%
cooking	C5	14.43%	7.45%
cutting_trees	C6	25.93%	33.98%
cutting_vegetables	C7	13.89%	5.62%
drinking	C8	9.76%	2.56%
feeding_a_horse	C9	17.86%	21.39%
fishing	C10	19.95%	45.66%
fixing_a_bike	C11	11.46%	14.06%
fixing_a_car	C12	12.93%	22.52%
gardening	C13	12.37%	12.12%
holding_an_umbrella	C14	17.14%	3.13%
jumping	C15	19.34%	17.95%
looking_through_a_microscope	C16	21.82%	13.19%
looking_through_a_telescope	C17	12.50%	12.62%
playing_guitar	C18	10.00%	8.81%
playing_violin	C19	20.73%	17.99%
pouring_liquid	C20	13.64%	18.75%
pushing_a_cart	C21	14.71%	5.00%
reading	C22	11.76%	10.37%
phoning	C23	8.89%	11.03%
riding_a_bike	C24	27.10%	45.08%
riding_a_horse	C25	31.96%	31.63%
rowing_a_boat	C26	21.24%	64.71%
running	C27	20.16%	17.22%
shooting_an_arrow	C28	13.39%	13.16%
smoking	C29	12.16%	12.77%
taking_photos	C30	4.60%	4.12%
texting_message	C31	10.53%	2.15%
throwing_frisby	C32	11.49%	9.80%
using_a_computer	C33	7.89%	11.54%
walking_the_dog	C34	25.00%	9.33%
washing_dishes	C35	11.27%	9.76%
watching_TV	C36	12.74%	21.95%
waving_hands	C37	3.29%	4.55%
writing_on_a_board	C38	15.46%	18.07%
writing_on_a_book	C39	15.25%	6.16%

Appendix 4. Precision and recall values per class from part B2 from network trained with *Ridge* regularization for 50 epochs.

CM	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23	C24	C25	C26	C27	C28	C29	C30	C31	C32	C33	C34	C35	C36	C37	C38	C39
C0	29	0	4	4	6	1	2	3	2	3	3	1	10	3	1	0	18	7	19	3	13	3	0	1	3	0	1	4	2	3	0	9	5	5	1	1	4	2	1	7
C1	8	13	0	1	5	0	3	1	7	3	8	1	4	16	1	1	7	1	9	5	7	4	1	0	7	2	0	2	3	5	0	6	9	1	3	0	3	0	7	5
C2	4	5	17	11	4	0	0	2	1	2	2	0	2	2	0	0	6	2	10	1	3	2	1	1	0	0	0	0	0	5	0	5	0	1	1	1	2	0	6	1
C3	0	0	2	44	3	2	2	2	2	1	0	3	3	1	0	3	1	2	2	0	0	1	1	0	2	0	0	0	0	1	1	0	1	7	10	2	7	0	4	2
C4	1	0	3	7	131	0	5	0	0	2	6	1	4	3	0	0	3	3	6	0	0	0	0	0	4	0	1	1	0	3	0	0	0	1	5	0	1	0	4	0
C5	5	3	2	4	3	14	2	13	2	5	3	9	18	10	0	0	10	0	8	9	8	6	8	2	2	2	4	2	1	1	0	5	0	11	0	1	3	0	6	6
C6	1	4	0	0	4	2	44	0	0	1	2	4	3	16	1	1	1	0	2	0	3	2	0	0	5	0	0	2	0	0	0	2	1	0	1	0	0	0	1	0
C7	3	0	1	5	0	3	0	11	3	3	1	1	4	2	0	0	4	0	6	1	3	9	0	1	0	0	0	0	0	0	0	3	0	9	0	3	4	0	2	7
C8	5	3	10	6	3	2	1	4	6	4	5	6	8	3	0	0	8	2	11	7	10	10	1	1	0	1	1	1	0	7	0	7	0	8	2	2	2	0	6	3
C9	1	3	1	4	4	0	6	1	3	42	9	10	12	16	6	2	6	6	9	1	8	0	2	1	3	4	3	1	3	0	0	5	1	5	1	0	3	0	0	5
C10	0	1	1	1	2	0	4	0	0	3	85	0	2	10	0	2	1	8	0	1	0	2	1	0	8	1	12	4	1	0	0	0	9	2	4	0	0	2	6	0
C11	6	1	0	0	6	1	8	1	1	4	0	15	23	13	1	0	10	1	2	0	3	3	1	0	5	1	1	1	0	0	1	3	2	5	3	2	1	0	1	2
C12	2	0	1	1	0	2	2	0	1	0	1	13	83	1	1	0	5	3	5	1	3	2	2	3	1	0	1	0	0	1	0	3	0	5	0	0	3	0	1	4
C13	0	3	0	0	0	0	11	1	0	9	3	7	3	42	3	0	0	1	0	0	0	1	1	0	3	0	0	1	0	1	0	0	0	0	4	0	1	1	2	1
C14	1	3	0	14	3	1	4	0	1	5	7	7	25	9	27	2	1	2	10	2	6	4	6	0	5	2	2	7	1	4	0	2	8	2	6	0	4	1	4	4
C15	1	0	0	13	3	0	3	0	1	0	14	2	6	7	0	54	1	5	1	1	0	1	2	0	14	2	9	17	1	0	0	1	9	3	9	0	8	1	4	2
C16	4	0	0	2	1	1	0	5	5	1	1	0	3	0	0	1	33	1	2	2	4	2	1	1	1	0	0	0	0	0	0	8	0	3	0	1	2	0	2	4
C17	3	1	0	5	0	0	1	1	0	5	11	4	6	3	0	2	5	24	4	0	2	0	1	1	0	4	3	1	2	0	0	2	0	3	3	1	0	0	4	1
C18	2	1	5	9	5	0	3	3	3	1	0	2	10	4	4	1	2	6	39	1	3	6	3	0	2	1	1	5	1	3	1	6	3	5	3	1	7	0	6	1
C19	23	1	1	5	2	2	0	2	3	2	4	3	17	4	2	1	10	5	8	39	5	10	4	1	3	1	3	4	1	2	0	7	1	3	1	0	6	1	0	2
C20	9	1	2	6	3	0	5	3	4	4	0	4	11	5	2	2	7	2	12	7	18	6	3	1	6	2	3	4	0	0	0	5	2	6	3	0	7	0	2	3
C21	9	2	5	3	0	1	1	5	1	0	0	3	3	5	2	0	11	1	6	2	3	8	1	1	2	0	1	1	0	0	0	3	0	7	0	0	6	0	1	6
C22	1	1	0	7	12	1	2	1	0	5	7	5	10	4	4	1	1	1	3	0	3	4	14	1	15	1	5	4	1	0	0	0	3	1	7	0	6	1	1	2
C23	5	1	1	4	1	1	1	4	2	3	2	4	9	9	4	0	10	0	9	2	6	6	2	3	2	1	2	2	0	1	1	10	2	13	4	1	1	0	5	11
C24	0	0	0	5	5	0	7	1	1	1	8	12	10	11	2	7	1	5	3	4	6	0	3	2	39	8	5	18	0	2	0	0	11	1	12	0	1	0	2	0
C25	5	6	0	1	2	0	2	1	2	10	12	4	6	13	1	5	2	7	4	5	2	1	3	1	18	36	6	10	5	1	1	0	11	2	7	0	2	0	1	1
C26	0	1	0	2	1	1	0	0	0	0	10	0	3	0	0	0	0	3	0	2	0	0	0	0	4	3	50	3	0	0	0	0	1	0	1	0	0	0	0	0
C27	0	0	0	6	1	0	0	1	2	0	11	3	3	3	2	4	0	3	0	0	4	1	1	0	15	5	8	44	2	0	0	0	15	0	14	0	0	0	2	1
C28	0	2	0	3	1	1	4	1	3	8	12	1	4	4	1	4	1	5	5	0	4	5	3	0	0	2	1	2	13	1	0	1	7	0	7	0	4	0	4	0
C29	12	3	7	2	7	1	3	0	7	2	2	7	9	2	1	1	4	0	22	0	4	2	0	1	2	1	1	0	0	15	0	5	3	4	0	0	5	0	2	4
C30	4	1	3	3	3	2	0	0	1	3	4	1	8	3	2	2	5	6	9	1	5	1	0	0	2	0	0	0	3	4	1	1	3	2	3	0	3	2	2	4
C31	6	4	5	5	1	0	2	1	0	0	3	2	3	2	0	0	9	2	3	3	4	2	1	1	0	1	2	1	1	0	0	9	1	7	2	0	4	0	2	4
C32	0	1	1	5	0	0	1	0	0	0	3	0	1	6	0	0	0	1	2	3	1	0	0	1	2	1	2	7	4	1	0	0	51	2	1	0	1	1	1	2
C33	3	0	1	7	2	1	0	2	2	2	3	3	13	2	0	0	8	2	2	1	1	3	0	2	2	0	1	1	0	0	0	7	0	25	3	1	12	1	6	11
C34	0	2	1	14	8	1	5	3	1	0	13	2	6	15	4	4	0	2	8	0	5	1	7	0	7	7	3	8	4	1	0	1	13	1	38	1	3	2	1	1
C35	3	0	1	7	2	3	0	8	2	3	0	3	2	1	0	0	6	0	5	0	4	7	0	1	0	0	0	0	0	2	0	3	0	7	0	1	3	0	4	4
C36	0	0	1	10	0	0	0	4	0	0	2	2	3	1	2	0	2	1	3	0	2	2	3	0	2	1	0	0	0	0	0	2	0	10	2	0	59	0	5	4
C37	7	2	2	6	6	1	0	1	3	2	4	5	3	1	2	0	1	1	12	1	4	6	1	0	1	0	3	2	1	1	0	7	3	3	3	1	3	1	6	4
C38	2	0	3	6	8	1	2	0	0	2	8	0	1	0	0	0	2	1	10	1	0	1	1	0	0	0	0	0	0	3	0	2	1	4	1	1	1	1	17	3
C39	5	1	2	3	1	0	2	7	1	1	0	2	14	1	2	0	13	3	10	2	3	3	0	3	1	0	0	0	1	1	0	9	4	13	0	4	2	0	2	30

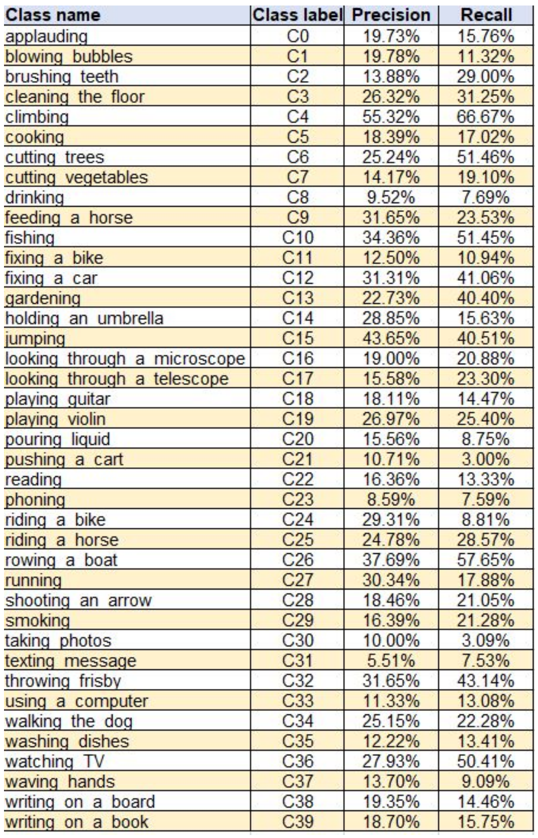
Appendix 5. Full confusion matrix from part B3 from network trained with transfer learning for 50 epochs.

Class name	Class label	Precision	Recall
applauding	C0	17.06%	15.76%
blowing_bubbles	C1	18.57%	8.18%
brushing_teeth	C2	20.48%	17.00%
cleaning_the_floor	C3	18.26%	39.29%
climbing	C4	52.61%	67.18%
cooking	C5	30.43%	7.45%
cutting_trees	C6	31.88%	42.72%
cutting_vegetables	C7	11.83%	12.36%
drinking	C8	8.22%	3.85%
feeding_a_horse	C9	29.58%	22.46%
fishing	C10	31.60%	49.13%
fixing_a_bike	C11	9.87%	11.72%
fixing_a_car	C12	22.55%	54.97%
gardening	C13	16.60%	42.42%
holding_an_umbrella	C14	34.62%	14.06%
jumping	C15	54.00%	27.69%
looking_through_a_microscope	C16	15.35%	36.26%
looking_through_a_telescope	C17	19.20%	23.30%
playing_guitar	C18	13.88%	24.53%
playing_violin	C19	36.11%	20.63%
pouring_liquid	C20	11.25%	11.25%
pushing_a_cart	C21	6.30%	8.00%
reading	C22	17.72%	10.37%
phoning	C23	9.68%	2.07%
riding_a_bike	C24	20.74%	20.21%
riding_a_horse	C25	40.00%	18.37%
rowing_a_boat	C26	37.04%	58.82%
running	C27	27.50%	29.14%
shooting_an_arrow	C28	25.49%	11.40%
smoking	C29	21.74%	10.64%
taking_photos	C30	16.67%	1.03%
texting_message	C31	6.47%	9.68%
throwing_frisby	C32	28.33%	50.00%
using_a_computer	C33	13.37%	19.23%
walking_the_dog	C34	23.03%	19.69%
washing_dishes	C35	4.00%	1.22%
watching_TV	C36	32.07%	47.97%
waving_hands	C37	5.88%	0.91%
writing_on_a_board	C38	12.78%	20.48%
writing_on_a_book	C39	19.74%	20.55%

Appendix 6. Precision and recall values per class from part B3 from network trained with transfer learning for 50 epochs.

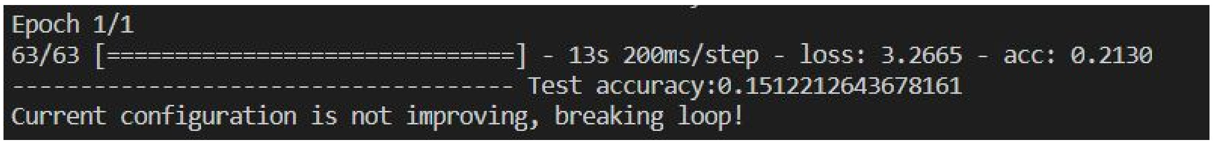
CM	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23	C24	C25	C26	C27	C28	C29	C30	C31	C32	C33	C34	C35	C36	C37	C38	C39
C0	29	2	13	1	3	3	2	1	8	3	2	1	3	5	2	0	9	5	9	4	7	0	0	4	2	3	2	2	7	10	1	8	4	6	0	5	7	7	0	4
C1	7	18	5	1	3	6	7	1	9	1	7	3	1	10	0	1	3	2	5	7	5	1	2	1	2	4	0	1	5	13	0	3	6	2	0	2	5	5	1	4
C2	1	4	29	4	7	0	1	1	5	1	0	0	1	3	0	0	3	3	8	2	2	0	0	0	0	0	0	0	1	9	0	5	0	1	1	1	2	1	1	3
C3	0	0	5	35	4	2	4	1	1	3	1	3	0	0	1	3	0	3	0	0	0	0	3	0	0	2	0	0	1	1	1	2	1	8	9	5	10	0	3	0
C4	0	0	6	3	130	0	13	0	0	1	7	0	1	1	1	3	0	4	6	2	2	0	0	1	2	0	2	0	0	5	1	0	0	0	2	0	0	2	0	0
C5	4	1	6	2	1	32	4	14	4	4	5	4	7	3	1	1	6	0	5	15	4	1	5	6	1	4	3	1	2	4	2	4	0	9	0	10	5	1	2	5
C6	0	5	1	0	6	4	53	0	0	1	1	1	2	8	1	2	1	0	0	4	1	0	0	1	1	4	0	0	2	0	0	2	0	0	1	0	0	1	0	0
C7	3	1	6	1	1	6	0	17	5	2	1	2	2	2	1	1	1	1	1	2	1	1	1	4	0	0	0	0	0	1	0	2	0	3	1	8	4	0	0	7
C8	2	4	17	4	3	3	2	6	12	4	3	2	5	3	0	1	3	2	3	12	4	2	1	3	1	3	0	0	0	15	0	7	1	6	2	5	6	2	2	5
C9	1	4	4	3	1	3	8	2	2	44	7	5	5	10	3	2	2	12	3	2	3	1	5	6	0	7	1	1	14	10	0	6	1	0	1	0	3	1	0	4
C10	1	2	0	0	2	1	5	0	0	3	89	0	1	6	0	6	0	11	0	0	0	0	2	0	2	10	13	3	5	1	0	1	3	0	3	0	0	1	2	0
C11	5	1	0	0	5	6	13	2	1	7	0	14	11	10	3	2	5	3	1	3	1	0	5	3	1	2	3	0	2	0	1	4	1	4	2	4	1	1	0	1
C12	2	2	3	0	0	9	6	0	3	0	1	10	62	2	0	1	4	2	3	1	1	1	2	6	0	3	2	0	0	3	1	3	0	5	0	1	3	3	1	5
C13	0	3	0	0	1	2	14	2	1	9	3	6	2	40	2	2	0	1	0	0	1	0	2	2	0	0	0	1	0	1	0	0	1	0	2	0	0	1	0	0
C14	2	2	4	7	2	7	6	0	2	6	8	5	14	6	30	8	0	6	6	3	2	0	8	6	0	5	3	4	3	6	1	2	6	0	8	0	7	1	4	2
C15	0	0	1	7	3	1	7	1	1	1	13	2	1	4	2	79	1	7	0	0	0	0	4	0	3	13	3	10	2	0	0	0	5	2	9	0	8	3	1	1
C16	2	0	3	1	1	5	0	10	6	1	0	0	2	0	0	1	19	1	1	3	2	0	2	3	0	2	0	0	1	1	1	5	0	3	1	4	3	1	1	5
C17	3	3	4	2	0	2	2	1	1	4	8	4	3	0	3	2	1	24	1	2	1	1	3	2	0	4	5	1	2	0	0	2	0	3	4	2	0	1	1	1
C18	3	2	15	7	4	1	2	2	5	2	1	2	5	4	6	0	1	9	23	3	2	1	5	2	0	2	0	2	2	11	1	4	2	7	5	1	6	4	4	1
C19	28	1	3	3	1	10	1	3	6	2	1	5	8	4	3	1	1	5	3	48	2	4	4	5	1	3	3	0	1	3	0	5	1	4	4	2	5	2	0	3
C20	8	4	4	3	1	8	7	4	4	5	0	4	3	4	3	2	4	3	5	10	14	2	1	4	2	4	4	6	3	6	0	4	1	5	5	2	7	1	1	2
C21	4	1	7	3	1	9	2	7	4	0	1	1	2	5	1	0	1	2	2	5	1	3	1	6	0	1	1	0	1	3	0	3	0	5	0	2	8	3	0	4
C22	1	2	2	2	10	5	6	0	0	4	6	2	5	2	7	2	1	2	1	1	2	0	18	5	6	6	5	2	3	2	0	0	2	2	8	2	8	1	1	1
C23	8	2	6	3	1	5	2	5	4	3	4	1	7	7	6	0	6	1	2	3	6	3	0	11	1	4	1	1	2	3	2	5	1	12	4	1	1	1	2	8
C24	1	0	0	0	6	4	10	3	2	3	8	11	6	8	4	13	0	5	3	5	3	0	5	4	17	20	6	10	5	2	1	1	12	3	11	0	0	1	0	0
C25	4	3	0	0	2	3	4	1	4	5	10	2	4	7	5	12	0	5	1	6	0	0	4	2	4	56	5	5	7	3	4	1	10	4	9	1	2	0	1	0
C26	0	0	0	0	1	3	1	0	0	0	10	0	1	0	0	2	0	3	0	2	0	0	3	0	0	4	49	2	0	0	0	0	0	0	1	0	1	2	0	0
C27	0	0	0	1	4	1	1	1	2	0	11	2	0	3	2	11	0	4	1	1	4	0	4	1	5	21	8	27	3	1	1	0	13	0	14	1	0	2	0	1
C28	1	3	0	4	1	1	4	2	3	4	15	2	4	0	1	4	0	4	1	2	3	1	3	0	0	4	1	1	24	1	1	1	5	0	7	0	4	0	2	0
C29	5	5	14	1	5	2	4	1	12	2	0	5	4	2	1	1	1	0	7	4	1	1	1	4	0	4	0	0	0	30	0	2	3	2	1	3	6	1	3	3
C30	5	4	4	1	1	1	1	1	3	3	2	1	3	2	1	2	4	8	6	1	2	1	3	2	1	3	0	0	4	8	3	3	1	0	1	0	3	3	1	4
C31	5	3	10	3	1	0	3	4	3	1	2	2	2	1	0	2	5	2	1	5	0	0	1	2	0	1	1	0	2	4	0	7	1	6	2	1	4	0	1	5
C32	0	2	2	2	0	1	1	1	0	0	3	0	1	4	1	5	0	1	0	2	0	0	0	1	1	7	1	4	11	1	0	1	44	1	1	0	0	2	0	1
C33	1	0	7	6	1	4	0	1	2	1	3	3	8	1	0	0	4	2	0	4	5	0	0	7	1	1	1	0	1	2	1	7	0	17	0	5	19	1	3	11
C34	1	1	2	9	6	6	7	1	1	1	13	0	2	7	7	7	0	4	2	2	0	0	6	2	4	12	4	3	10	5	3	2	9	2	43	1	5	3	0	0
C35	1	0	2	3	1	7	0	6	2	3	0	3	2	0	0	0	4	1	0	2	2	2	0	4	0	0	0	0	1	4	0	4	0	2	0	11	5	0	5	5
C36	0	0	3	4	0	3	0	5	0	0	1	0	2	0	2	0	2	3	2	0	0	1	5	1	0	2	0	0	1	0	1	2	0	10	3	1	62	1	4	2
C37	5	5	7	2	5	3	2	2	3	2	4	3	0	1	2	1	0	1	6	5	2	0	0	1	0	3	2	2	0	8	0	8	1	2	4	1	3	10	2	2
C38	1	1	6	3	9	1	2	1	2	1	8	0	1	0	1	0	1	2	3	1	1	0	1	4	0	0	0	0	1	5	3	2	1	2	2	2	3	0	12	0
C39	3	0	8	2	1	4	3	10	3	2	0	1	5	1	1	1	7	0	6	4	3	1	0	12	0	2	1	0	1	1	0	9	3	12	0	6	6	3	1	23

Appendix 7. Full confusion matrix for best model found by automatic model search algorithm.

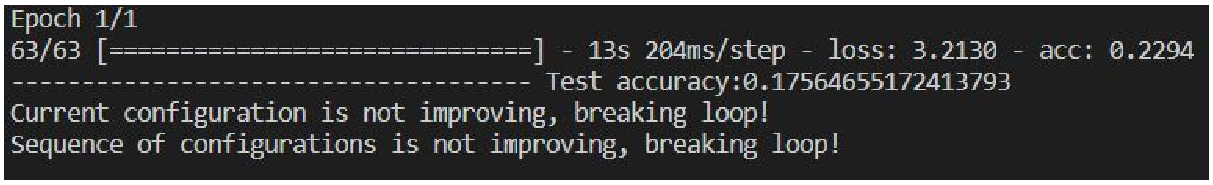


Class name	Class label	Precision	Recall
applauding	C0	19.73%	15.76%
blowing_bubbles	C1	19.78%	11.32%
brushing_teeth	C2	13.88%	29.00%
cleaning_the_floor	C3	26.32%	31.25%
climbing	C4	55.32%	66.67%
cooking	C5	18.39%	17.02%
cutting_trees	C6	25.24%	51.46%
cutting_vegetables	C7	14.17%	19.10%
drinking	C8	9.52%	7.69%
feeding_a_horse	C9	31.65%	23.53%
fishing	C10	34.36%	51.45%
fixing_a_bike	C11	12.50%	10.94%
fixing_a_car	C12	31.31%	41.06%
gardening	C13	22.73%	40.40%
holding_an_umbrella	C14	28.85%	15.63%
jumping	C15	43.65%	40.51%
looking_through_a_microscope	C16	19.00%	20.88%
looking_through_a_telescope	C17	15.58%	23.30%
playing_guitar	C18	18.11%	14.47%
playing_violin	C19	26.97%	25.40%
pouring_liquid	C20	15.56%	8.75%
pushing_a_cart	C21	10.71%	3.00%
reading	C22	16.36%	13.33%
phoning	C23	8.59%	7.59%
riding_a_bike	C24	29.31%	8.81%
riding_a_horse	C25	24.78%	28.57%
rowing_a_boat	C26	37.69%	57.65%
running	C27	30.34%	17.88%
shooting_an_arrow	C28	18.46%	21.05%
smoking	C29	16.39%	21.28%
taking_photos	C30	10.00%	3.09%
texting_message	C31	5.51%	7.53%
throwing_frisby	C32	31.65%	43.14%
using_a_computer	C33	11.33%	13.08%
walking_the_dog	C34	25.15%	22.28%
washing_dishes	C35	12.22%	13.41%
watching_TV	C36	27.93%	50.41%
waving_hands	C37	13.70%	9.09%
writing_on_a_board	C38	19.35%	14.46%
writing_on_a_book	C39	18.70%	15.75%

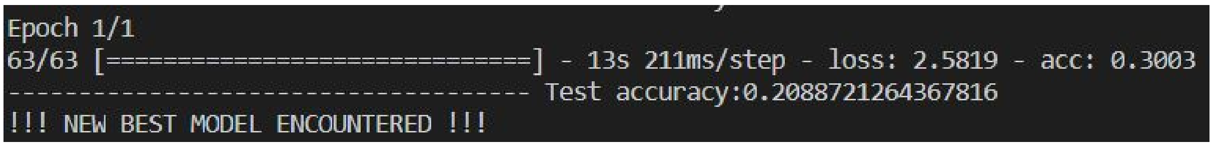
Appendix 8. Precision and recall for best found model by automatic model search algorithm.



```
Epoch 1/1
63/63 [==============================] - 13s 200ms/step - loss: 3.2665 - acc: 0.2130
------------------------------------ Test accuracy:0.1512212643678161
Current configuration is not improving, breaking loop!
```



```
Epoch 1/1
63/63 [==============================] - 13s 204ms/step - loss: 3.2130 - acc: 0.2294
------------------------------------ Test accuracy:0.17564655172413793
Current configuration is not improving, breaking loop!
Sequence of configurations is not improving, breaking loop!
```



```
Epoch 1/1
63/63 [==============================] - 13s 211ms/step - loss: 2.5819 - acc: 0.3003
------------------------------------ Test accuracy:0.2088721264367816
!!! NEW BEST MODEL ENCOUNTERED !!!
```

Appendix 9. Some screenshots from the automatic model search algorithm in action.