INFOMCV 2017/2018

Assignment 4 Report – Action Classification in Video

Stephan Wells

Game & Media Technology

Graduate School of Natural Sciences

Utrecht University

Utrecht, Netherlands

s.b.wells@students.uu.nl

6157114

Erik Scerri

Game & Media Technology

Graduate School of Natural Sciences

Utrecht University

Utrecht, Netherlands

e.scerri@students.uu.nl

6158811

# Overview

**Objectives Met:**

* **Training a CNN (10)**
* **Testing a video on a single frame (10)**
* **K-Fold Cross Validation (10)**
* **Performance Measurement (10)**
* ***Train on mirrored frames (5)***
* ***Test performance using different sets of parameters (10)***
* ***Test performance on different network topologies (5)***
* ***Test performance on your test sets and own recordings (20)***
* ***Test performance on all frames of a test video (5)***

**References:**

* Lecture slides.
* OpenCV documentation: <https://docs.opencv.org/master/>
* TensorFlow documentation: <https://www.tensorflow.org/api_docs/>
* 3Blue1Brown’s neural network explanations: <https://www.youtube.com/watch?v=aircAruvnKk>

**Division of Work:**

* Stephan Wells:
  + Coded functionality for confusion matrices.
  + Implemented cross validation.
  + Worked on tuning parameters.
  + Wrote the report.
  + Performed general research & debugging.
  + Recorded videos for test data.
* Erik Scerri:
  + Handled file I/O and processing.
  + Set up the convolutional neural network.
  + Worked on tuning parameters.
  + Wrote the report.
  + Performed general research & debugging.
  + Recorded videos for test data.

# Implementation

The purpose of this section is to detail the implemented functionality through explanations and code snippers and to give an overview of the training/testing pipeline developed for the assignment.

The project was built in Python 3.6, using OpenCV and TensorFlow. Work was done on the following files:

* main.py: Contains the code that processes video frames and calls training/evaluation methods.
* cnn.py: Contains the TensorFlow setup of the CNN, including the layer topology.
* validation.py: Contains code that performs cross validation and generates performance measures.

**Data Loading/Processing:**

Loading of video frames is done through OpenCV in Python. The default method is to load the middle frame from the video, using getFrameMat(path). Functionality to load a specific frame (or multiple frames) from a video was also added.

1. **def** getFrameMat(path):
2. vid = cv.VideoCapture(path)
3. count = vid.get(cv.CAP\_PROP\_FRAME\_COUNT)
4. vid.set(cv.CAP\_PROP\_POS\_FRAMES, count / 2)
5. \_, frame = vid.read()
6. vid.release()
7. **return** frame

The main data loading method, getFrameMats(basepath) repeatedly calls the above methods for all files in the given base directory. The basepath parameter is expected to point to a folder which contains 5 subfolders, one per Action. Loaded frames are **reshaped to a standard 90x90** image dimension. Note that this method is also where **loaded frames are horizontally flipped** to serve as an augmentation to the training data. This method returns two lists, one is the list of frames (90x90x3 matrices), and one is the list of labels corresponding to each image (their class.)

1. for filename in os.listdir(datapath):
2. frame = getFrameMat(datapath + filename)
3. mat = cv.resize(frame, (90, 90))
5. mat\_orig = np.divide(np.float32(mat), 255)
6. labels.append(action.value)
7. mats.append(mat\_orig)
9. mat\_flip = cv.flip(mat\_orig, 1)
10. labels.append(action.value)
11. mats.append(mat\_flip)

**Cross Validation:**

validation.py deals with all of the code related to cross validation, evaluation, and performance measures. k-fold cross validation was handled by the crossValidation() method, which takes a single parameter (*k*) and splits the training data in a uniformly random manner into *k* folds. It then runs through *k* iterations, where a different fold is used for validation each time and the rest of the folds are concatenated for the training data.

The below code snippet from the aforementioned method shows how the fold segregation functions.

1. end\_offset = 1 **if** i < (k – 1) **else** 0
2. left\_data = data[0:fold\_size \* i]
3. left\_labels = labels[0:fold\_size \* i]
4. right\_data = data[fold\_size \* (i + 1) + end\_offset:]
5. right\_labels = labels[fold\_size \* (i + 1) + end\_offset:]

**Performance Measures:**

Four performance measures were calculated: precision, recall, F-score, and confusion matrices. The respective methods for these performance measures are all in validation.py. For each fold iteration, the iteration’s confusion matrix is calculated, and its values are added to a cumulative confusion matrix in the following code snippet found in crossValidation().

1. predicted\_labels = []
2. **for** result **in** results:
3. predicted\_labels.append(result['classes'])
4. conf\_mat = generateConfusionMatrix(eval\_labels, predicted\_labels)
5. overall\_conf\_mat = np.add(overall\_conf\_mat, conf\_mat)

This overall confusion matrix is then averaged after all of the fold iterations are carried out. It is outputted on screen and used to calculate the of the performance measures and output them to file.

1. averaged\_conf\_mat = np.divide(np.float32(overall\_conf\_mat), k)
2. outputConfusionMatrix(averaged\_conf\_mat)
3. generatePerfMeasures(averaged\_conf\_mat)

**The Neural Network:**

The network topology is defined and constructed inside cnn.py, specifically in the method cnn\_model(), which accepts a list of inputs and a list of expected labels, and then defines how these will traverse the network. The topology is defined using standard Tensorflow instructions, notably conv2d(), max\_pooling2d(), and dense().

1. inputLayer = tf.reshape(features["x"], [-1, 90, 90, 3])
2. conv1 = tf.layers.conv2d(
3. inputs=inputLayer,
4. filters=32,
5. kernel\_size=[7, 7],
6. padding="valid",
7. activation=tf.nn.relu)
9. pool1 = tf.layers.max\_pooling2d(inputs=conv1, pool\_size=[2, 2], strides=2)

Important parameters such as **Learning Rate** and **Dropout Rate** are also defined in this file. The model is called for training, evaluation, or prediction, and will return different outputs for either. Notably, for evaluation mode it will return a general accuracy and loss value, and for prediction mode it will return a list of predicted classes.

1. predictions = {
2. "classes": tf.argmax(input=logits, axis=1),
3. "probabilities": tf.nn.softmax(logits, name="softmax\_tensor")
4. }
6. **if** mode == tf.estimator.ModeKeys.PREDICT:
7. **return** tf.estimator.EstimatorSpec(mode=mode, predictions=predictions)

# Results

The initial tests being run here are designed to determine an optimal layout for the neural network. As such, all parameter tuning is done by testing on a validation set using cross validation. This means that accuracy results here are not representative of actual performance on a proper test set but do serve as a guide for improvement in terms of parameter tuning. **All results given here are averages over all folds.**

**Baseline Test:**

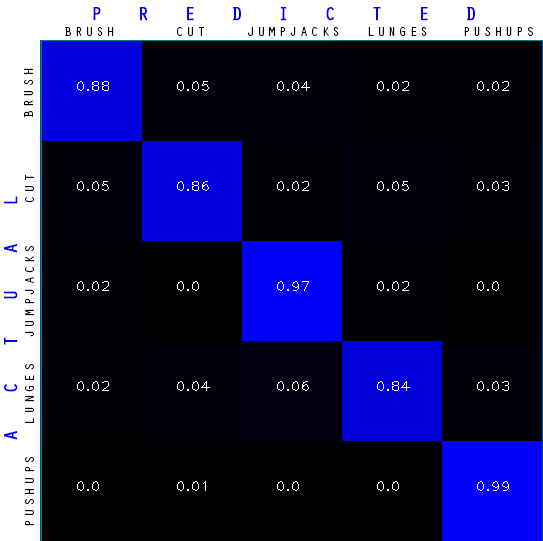
|  |  |
| --- | --- |
| **testA1: Baseline** | |
| *Learning Rate* | 0.001 |
| *Dropout Rate* | 0.3 |
| *Batch Size* | 10 |
| *Num of Epochs* | 10000 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Class** | **0** | **1** | **2** | **3** | **4** |
| *Precision* | 0.878 | 0.862 | 0.967 | 0.843 | 0.992 |
| *Recall* | 0.911 | 0.903 | 0.890 | 0.908 | 0.930 |
| *F-Score* | 0.894 | 0.882 | 0.927 | 0.874 | 0.960 |

|  |  |
| --- | --- |
| *Average Precision* | 0.9084 |
| *Standard Dev.* | 0.0596 |

|  |  |
| --- | --- |
| *Average Recall* | 0.9084 |
| *Standard Dev.* | 0.0130 |

|  |  |
| --- | --- |
| *Average F-Score* | 0.9074 |
| *Standard Dev.* | 0.0319 |



**Train on Mirrored Frames:**

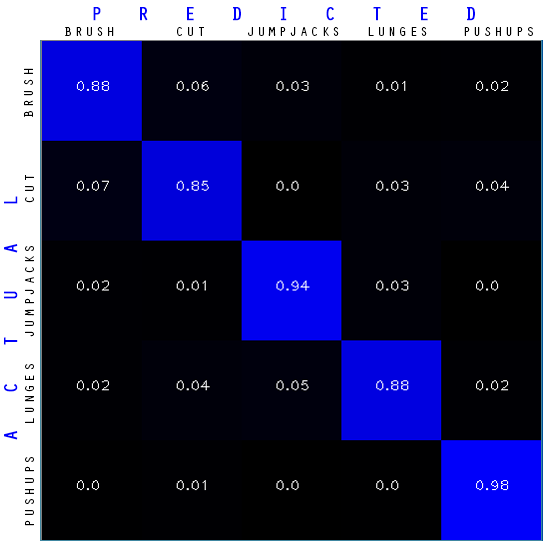
|  |  |
| --- | --- |
| **testA2: Augmented Data** | |
| *Learning Rate* | 0.001 |
| *Dropout Rate* | 0.3 |
| *Batch Size* | 10 |
| *Num of Epochs* | 10000 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Class** | **0** | **1** | **2** | **3** | **4** |
| *Precision* | 0.877 | 0.855 | 0.939 | 0.877 | 0.981 |
| *Recall* | 0.889 | 0.873 | 0.916 | 0.917 | 0.931 |
| *F-Score* | 0.884 | 0.863 | 0.928 | 0.901 | 0.956 |

|  |  |
| --- | --- |
| *Average Precision* | 0.9058 |
| *Standard Dev.* | 0.0469 |

|  |  |
| --- | --- |
| *Average Recall* | 0.9052 |
| *Standard Dev.* | 0.0210 |

|  |  |
| --- | --- |
| *Average F-Score* | 0.9074 |
| *Standard Dev.* | 0.0327 |



**Parameter Tuning – Batch Size:**

Each parameter for tuning is tested with five different values. The best of the five values is carried over to the next test. For Batch Size, the tested values are: **10, 15, 20, 25, 50**. Since Batch Size of 10 with this network setup is already tested above, the test is not repeated here.

|  |  |
| --- | --- |
| **testB1: Batch Size Tuning** | |
| *Learning Rate* | 0.001 |
| *Dropout Rate* | 0.3 |
| *Batch Size* | **15** |
| *Num of Epochs* | 10000 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Class** | **0** | **1** | **2** | **3** | **4** |
| *Precision* | 0.916 | 0.809 | 0.951 | 0.889 | 0.985 |
| *Recall* | 0.865 | 0.908 | 0.928 | 0.922 | 0.929 |
| *F-Score* | 0.890 | 0.856 | 0.939 | 0.905 | 0.956 |

|  |  |
| --- | --- |
| *Average Precision* | 0.9100 |
| *Standard Dev.* | 0.0601 |

|  |  |
| --- | --- |
| *Average Recall* | 0.9104 |
| *Standard Dev.* | 0.0239 |

|  |  |
| --- | --- |
| *Average F-Score* | 0.9092 |
| *Standard Dev.* | 0.0355 |

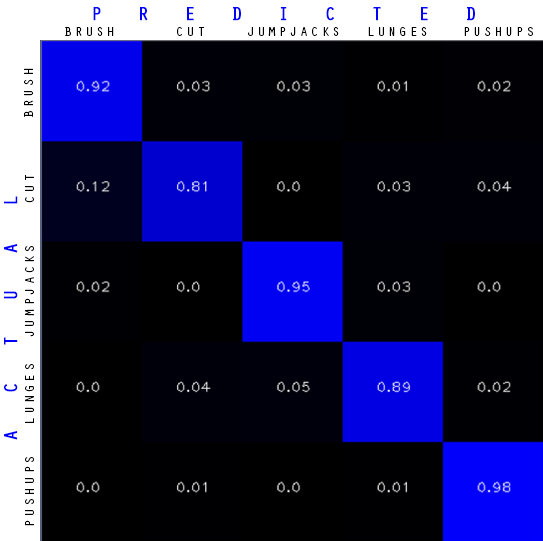
|  |  |
| --- | --- |
| **testB2: Batch Size Tuning** | |
| *Learning Rate* | 0.001 |
| *Dropout Rate* | 0.3 |
| *Batch Size* | **20** |
| *Num of Epochs* | 10000 |

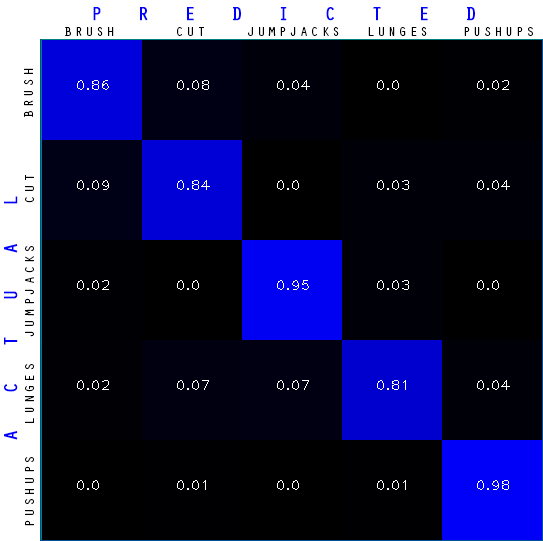
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Class** | **0** | **1** | **2** | **3** | **4** |
| *Precision* | 0.859 | 0.841 | 0.951 | 0.807 | 0.981 |
| *Recall* | 0.871 | 0.841 | 0.897 | 0.910 | 0.918 |
| *F-Score* | 0.865 | 0.841 | 0.923 | 0.855 | 0.949 |

|  |  |
| --- | --- |
| *Average Precision* | 0.8878 |
| *Standard Dev.* | 0.0667 |

|  |  |
| --- | --- |
| *Average Recall* | 0.8874 |
| *Standard Dev.* | 0.0281 |

|  |  |
| --- | --- |
| *Average F-Score* | 0.8866 |
| *Standard Dev.* | 0.0419 |





|  |  |
| --- | --- |
| **testB3: Batch Size Tuning** | |
| *Learning Rate* | 0.001 |
| *Dropout Rate* | 0.3 |
| *Batch Size* | **25** |
| *Num of Epochs* | 10000 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Class** | **0** | **1** | **2** | **3** | **4** |
| *Precision* | 0.893 | 0.836 | 0.959 | 0.850 | 0.977 |
| *Recall* | 0.884 | 0.899 | 0.906 | 0.915 | 0.911 |
| *F-Score* | 0.888 | 0.867 | 0.932 | 0.881 | 0.943 |

|  |  |
| --- | --- |
| *Average Precision* | 0.9030 |
| *Standard Dev.* | 0.0566 |

|  |  |
| --- | --- |
| *Average Recall* | 0.9030 |
| *Standard Dev.* | 0.0109 |

|  |  |
| --- | --- |
| *Average F-Score* | 0.9022 |
| *Standard Dev.* | 0.0298 |

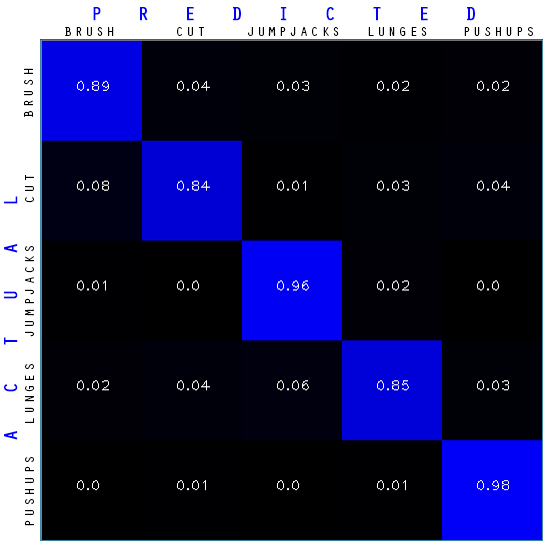
|  |  |
| --- | --- |
| **testB4: Batch Size Tuning** | |
| *Learning Rate* | 0.001 |
| *Dropout Rate* | 0.3 |
| *Batch Size* | **50** |
| *Num of Epochs* | 10000 |

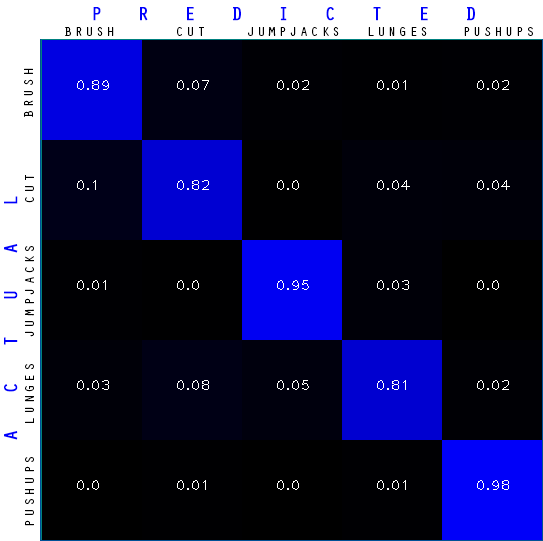
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Class** | **0** | **1** | **2** | **3** | **4** |
| *Precision* | 0.885 | 0.822 | 0.951 | 0.814 | 0.981 |
| *Recall* | 0.864 | 0.834 | 0.927 | 0.894 | 0.932 |
| *F-Score* | 0.874 | 0.828 | 0.939 | 0.852 | 0.956 |

|  |  |
| --- | --- |
| *Average Precision* | 0.8906 |
| *Standard Dev.* | 0.0669 |

|  |  |
| --- | --- |
| *Average Recall* | 0.8902 |
| *Standard Dev.* | 0.0373 |

|  |  |
| --- | --- |
| *Average F-Score* | 0.8898 |
| *Standard Dev.* | 0.0496 |





**Parameter Tuning – Dropout Rate:**

The best Batch Size value out of the tested five is carried over to the next test. A Batch Size of 15 gave the highest average F-Score and was thus chosen to continue into the next batch of tests. For Dropout Rate values are: **0.3, 0.4, 0.5, 0.6, 0.7**. Since Dropout Rate of 0.3 with this network setup is already tested above, the test is not repeated here.

|  |  |
| --- | --- |
| **testC1: Dropout Rate Tuning** | |
| *Learning Rate* | 0.001 |
| *Dropout Rate* | **0.4** |
| *Batch Size* | **15** |
| *Num of Epochs* | 10000 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Class** | **0** | **1** | **2** | **3** | **4** |
| *Precision* | 0.882 | 0.850 | 0.967 | 0.857 | 0.977 |
| *Recall* | 0.901 | 0.898 | 0.882 | 0.917 | 0.937 |
| *F-Score* | 0.891 | 0.873 | 0.922 | 0.886 | 0.956 |

|  |  |
| --- | --- |
| *Average Precision* | 0.9066 |
| *Standard Dev.* | 0.0545 |

|  |  |
| --- | --- |
| *Average Recall* | 0.9070 |
| *Standard Dev.* | 0.0187 |

|  |  |
| --- | --- |
| *Average F-Score* | 0.9056 |
| *Standard Dev.* | 0.0299 |

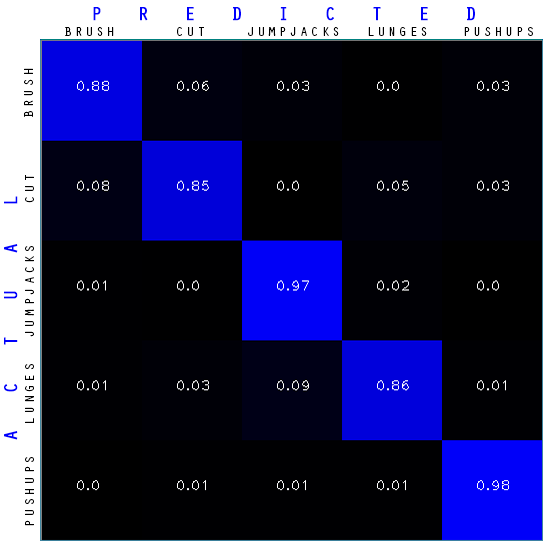
|  |  |
| --- | --- |
| **testC2: Dropout Rate Tuning** | |
| *Learning Rate* | 0.001 |
| *Dropout Rate* | **0.5** |
| *Batch Size* | **15** |
| *Num of Epochs* | 10000 |

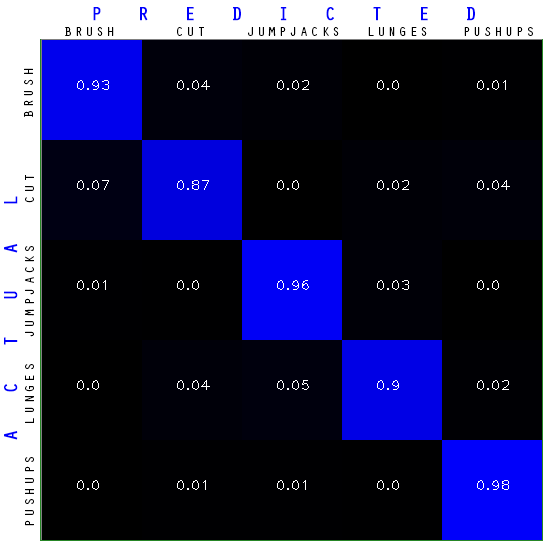
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Class** | **0** | **1** | **2** | **3** | **4** |
| *Precision* | 0.927 | 0.868 | 0.959 | 0.897 | 0.981 |
| *Recall* | 0.920 | 0.907 | 0.921 | 0.942 | 0.942 |
| *F-Score* | 0.924 | 0.887 | 0.940 | 0.919 | 0.961 |

|  |  |
| --- | --- |
| *Average Precision* | 0.9264 |
| *Standard Dev.* | 0.0408 |

|  |  |
| --- | --- |
| *Average Recall* | 0.9264 |
| *Standard Dev.* | 0.0137 |

|  |  |
| --- | --- |
| *Average F-Score* | 0.9262 |
| *Standard Dev.* | 0.0245 |





|  |  |
| --- | --- |
| **testC3: Dropout Rate Tuning** | |
| *Learning Rate* | 0.001 |
| *Dropout Rate* | **0.6** |
| *Batch Size* | **15** |
| *Num of Epochs* | 10000 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Class** | **0** | **1** | **2** | **3** | **4** |
| *Precision* | 0.897 | 0.818 | 0.959 | 0.870 | 0.984 |
| *Recall* | 0.887 | 0.914 | 0.922 | 0.898 | 0.908 |
| *F-Score* | 0.892 | 0.863 | 0.940 | 0.884 | 0.945 |

|  |  |
| --- | --- |
| *Average Precision* | 0.9056 |
| *Standard Dev.* | 0.0600 |

|  |  |
| --- | --- |
| *Average Recall* | 0.9058 |
| *Standard Dev.* | 0.0122 |

|  |  |
| --- | --- |
| *Average F-Score* | 0.9048 |
| *Standard Dev.* | 0.0322 |

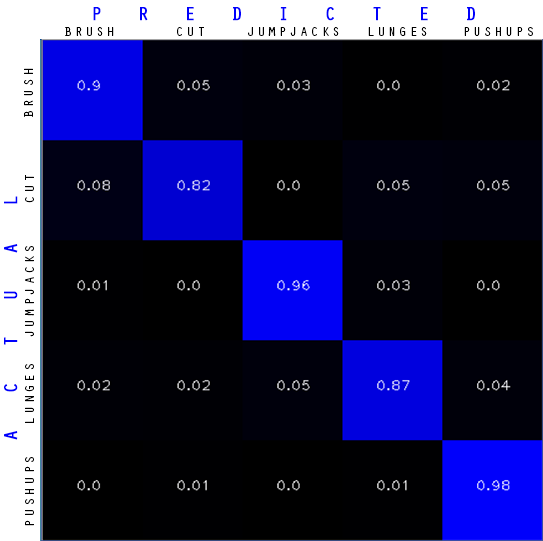
|  |  |
| --- | --- |
| **testC4: Dropout Rate Tuning** | |
| *Learning Rate* | 0.001 |
| *Dropout Rate* | **0.7** |
| *Batch Size* | **15** |
| *Num of Epochs* | 10000 |

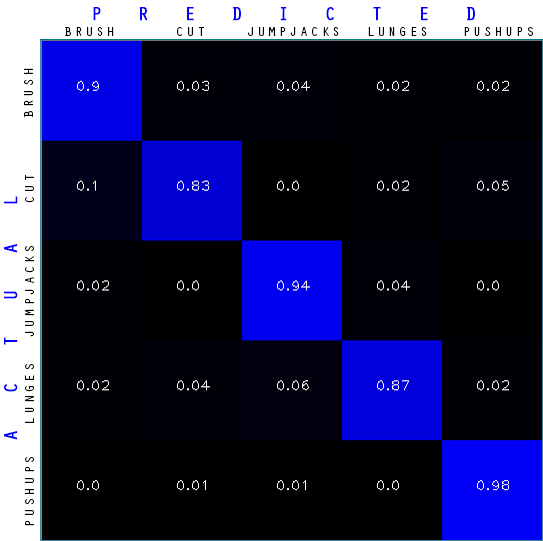
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Class** | **0** | **1** | **2** | **3** | **4** |
| *Precision* | 0.897 | 0.818 | 0.959 | 0.87 | 0.984 |
| *Recall* | 0.887 | 0.914 | 0.922 | 0.898 | 0.908 |
| *F-Score* | 0.892 | 0.863 | 0.94 | 0.884 | 0.945 |

|  |  |  |
| --- | --- | --- |
| *Average Precision* | 0.9060 | 0.9264 |
| *Standard Dev.* | 0.0490 | 0.0408 |

|  |  |  |
| --- | --- | --- |
| *Average Recall* | 0.9042 | 0.9264 |
| *Standard Dev.* | 0.0220 | 0.0137 |

|  |  |  |
| --- | --- | --- |
| *Average F-Score* | 0.9044 | 0.9262 |
| *Standard Dev.* | 0.0270 | 0.0245 |





**Parameter Tuning – Learning Rate:**

The best Dropout Rate value out of the tested five is carried over to the next test. A Dropout Rate of 0.5 gave the highest average F-Score and the least deviation from the average. It was therefore carried over into this next set of tests. For Learning Rate values are: **0.001, 0.0001, 0.002, 0.005, 0.05**. Since Learning Rate of 0.001 with this network setup is already tested above, the test is not repeated here.

|  |  |
| --- | --- |
| **testD1: Learning Rate Tuning** | |
| *Learning Rate* | **0.0001** |
| *Dropout Rate* | **0.5** |
| *Batch Size* | **15** |
| *Num of Epochs* | 10000 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Class** | **0** | **1** | **2** | **3** | **4** |
| *Precision* | 0.706 | 0.575 | 0.740 | 0.598 | 0.679 |
| *Recall* | 0.662 | 0.679 | 0.587 | 0.691 | 0.708 |
| *F-Score* | 0.684 | 0.622 | 0.654 | 0.641 | 0.694 |

|  |  |
| --- | --- |
| *Average Precision* | 0.6596 |
| *Standard Dev.* | 0.0632 |

|  |  |
| --- | --- |
| *Average Recall* | 0.6654 |
| *Standard Dev.* | 0.0420 |

|  |  |
| --- | --- |
| *Average F-Score* | 0.6590 |
| *Standard Dev.* | 0.0267 |

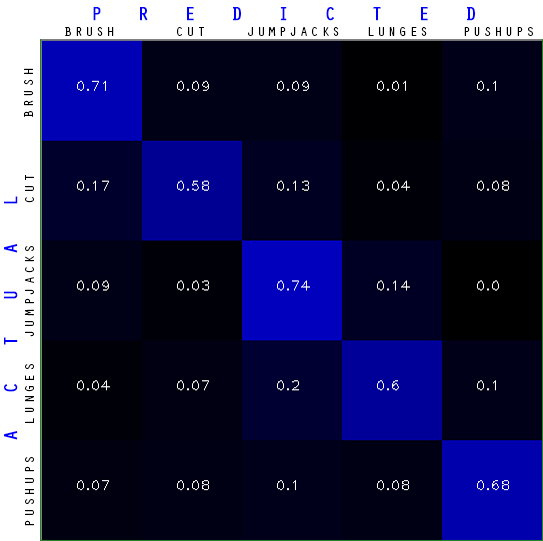
|  |  |
| --- | --- |
| **testD2: Learning Rate Tuning** | |
| *Learning Rate* | **0.002** |
| *Dropout Rate* | **0.5** |
| *Batch Size* | **15** |
| *Num of Epochs* | 10000 |

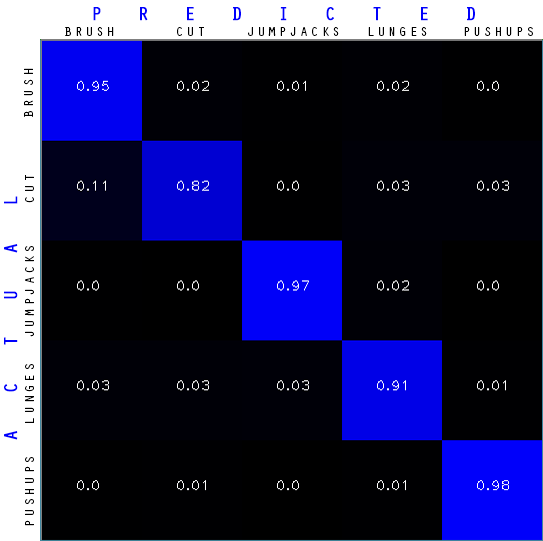
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Class** | **0** | **1** | **2** | **3** | **4** |
| *Precision* | 0.947 | 0.822 | 0.971 | 0.906 | 0.981 |
| *Recall* | 0.867 | 0.934 | 0.957 | 0.916 | 0.957 |
| *F-Score* | 0.905 | 0.874 | 0.964 | 0.911 | 0.969 |

|  |  |
| --- | --- |
| *Average Precision* | 0.9254 |
| *Standard Dev.* | 0.0578 |

|  |  |
| --- | --- |
| *Average Recall* | 0.9262 |
| *Standard Dev.* | 0.0333 |

|  |  |
| --- | --- |
| *Average F-Score* | 0.9246 |
| *Standard Dev.* | 0.0365 |





|  |  |
| --- | --- |
| **testD3: Learning Rate Tuning** | |
| *Learning Rate* | **0.005** |
| *Dropout Rate* | **0.5** |
| *Batch Size* | **15** |
| *Num of Epochs* | 10000 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Class** | **0** | **1** | **2** | **3** | **4** |
| *Precision* | 0.931 | 0.858 | 0.939 | 0.917 | 0.996 |
| *Recall* | 0.906 | 0.917 | 0.948 | 0.904 | 0.966 |
| *F-Score* | 0.918 | 0.887 | 0.944 | 0.910 | 0.981 |

|  |  |
| --- | --- |
| *Average Precision* | 0.9282 |
| *Standard Dev.* | 0.0442 |

|  |  |
| --- | --- |
| *Average Recall* | 0.9282 |
| *Standard Dev.* | 0.0246 |

|  |  |
| --- | --- |
| *Average F-Score* | 0.9280 |
| *Standard Dev.* | 0.0322 |

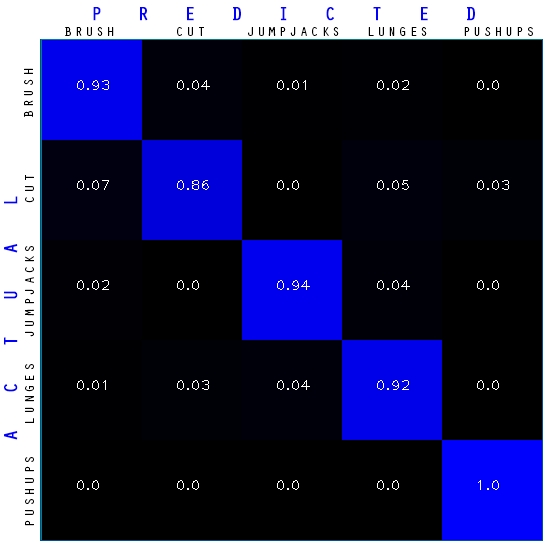
|  |  |
| --- | --- |
| **testD4: Learning Rate Tuning** | |
| *Learning Rate* | **0.05** |
| *Dropout Rate* | **0.5** |
| *Batch Size* | **15** |
| *Num of Epochs* | 10000 |

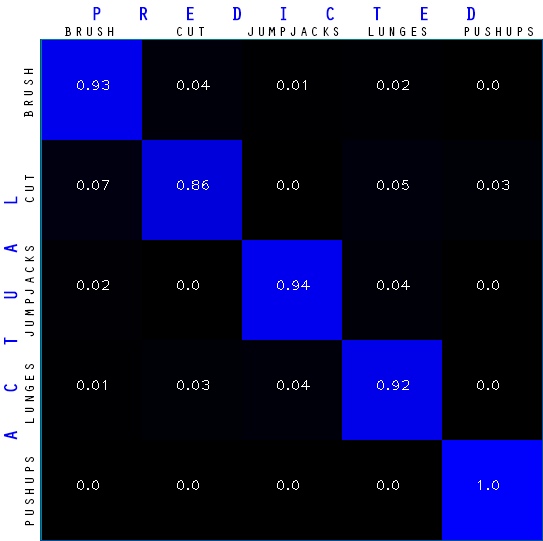
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Class** | **0** | **1** | **2** | **3** | **4** |
| *Precision* | 0.931 | 0.858 | 0.939 | 0.917 | 0.996 |
| *Recall* | 0.905 | 0.917 | 0.948 | 0.903 | 0.966 |
| *F-Score* | 0.918 | 0.887 | 0.944 | 0.911 | 0.981 |

|  |  |  |
| --- | --- | --- |
| *Average Precision* | 0.9282 | 0.9264 |
| *Standard Dev.* | 0.0442 | 0.0408 |

|  |  |  |
| --- | --- | --- |
| *Average Recall* | 0.9278 | 0.9264 |
| *Standard Dev.* | 0.0250 | 0.0137 |

|  |  |  |
| --- | --- | --- |
| *Average F-Score* | 0.9282 | 0.9262 |
| *Standard Dev.* | 0.0320 | 0.0245 |





# Discussion