Tutorial

Healthc Inform Res. 2019 April;25(2):131-138.
https://doi.org/10.4258/hir.2019.25.2.131
pISSN 2093-3681 • eISSN 2093-369X

HIR
Healthcare Informatics Research

Development of Artificial Intelligence to Support Needle Electromyography Diagnostic Analysis

Sangwoo Nam[1]*, Min Kyun Sohn[2,3]*, Hyun Ah Kim[2], Hyoun-Joong Kong[4], Il-Young Jung[2,3]

[1]Department of Biomedical Engineering, Chungnam National University Graduade School, Daejeon, Korea
[2]Department of Rehabilitation Medicine, Chungnam National University Hospital, Daejeon, Korea
[3]Department of Rehabilitation Medicine, Chungnam National University College of Medicine, Daejeon, Korea
[4]Department of Biomedical Engineering, Chungnam National University College of Medicine, Daejeon, Korea

Objectives: This study proposes a method for classifying three types of resting membrane potential signals obtained as images through diagnostic needle electromyography (EMG) using TensorFlow-Slim and Python to implement an artificial-intelligence-based image recognition scheme. **Methods:** Waveform images of an abnormal resting membrane potential generated by diagnostic needle EMG were classified into three types—positive sharp waves (PSW), fibrillations (Fibs), and Others—using the TensorFlow-Slim image classification model library. A total of 4,015 raw waveform data instances were reviewed, with 8,576 waveform images subsequently collected for training. Images were learned repeatedly through a convolutional neural network. Each selected waveform image was classified into one of the aforementioned categories according to the learned results. **Results:** The classification model, Inception v4, was used to divide waveform images into three categories (accuracy = 93.8%, precision = 99.5%, recall = 90.8%). This was done by applying the pretrained Inception v4 model to a fine-tuning method. The image recognition model was created for training using various types of image-based medical data. **Conclusions:** The TensorFlow-Slim library can be used to train and recognize image data, such as EMG waveforms, through simple coding rather than by applying TensorFlow. It is expected that a convolutional neural network can be applied to image data such as the waveforms of electrophysiological signals in a body based on this study.

Keywords: Artificial Intelligence, Deep Learning, Electromyography, Convolutional Neural Network, Classification

Submitted: March 28, 2019
Revised: April 20, 2019
Accepted: April 22, 2019

Corresponding Author
Il-Young Jung
Department of Rehabilitation Medicine, Chungnam National University College of Medicine, 266 Munhwa-ro, Jung-gu, Daejeon 35015, Korea. Tel: +82-42-338-2460, E-mail: 102onez@cnuh.co.kr (https://orcid.org/0000-0001-8204-8195)
*These authors contributed equally to this work.

I. Introduction

Electrodiagnostic studies include nerve conduction studies, needle electromyography (EMG), and evoked potential studies, among others. Testing electrical signals from nerves, muscles, and neuromuscular junctions can help to clarify electrophysiologic findings and abnormalities [1]. The electrodiagnostic studies obtained results through a combination of protocolized test procedures, expert decision-making, and by referring to quantified results. Needle EMG, one of the tests that make up electrodiagnostic studies, involves inserting a needle electrode into a muscle to record generated electrical signals. A specialist checked whether these signals are normal or abnormal to estimate electrophysiological information on nerves and muscles, such as the period after

damage, injury lesion, and amount of injury.

Artificial intelligence (AI) has been applied in healthcare and medicine for many years [2]. Specifically, deep learning techniques can be applied to diagnostic needle EMG owing to the need for repetitive testing in each muscle, the importance of quantitative definitions to discern normal and abnormal EMG outcomes, and the subjective nature of interpreting EMG results. Moreover, the criteria for normal and abnormal findings are defined quantitatively, thus enabling high-quality data to be selected. Further, an examiner's expertise and interpretation can influence the diagnosis; therefore, AI can be used to improve the decision-making accuracy.

In this study, we retrospectively collect needle EMG waveforms with abnormal spontaneous activities (ASAs) of the resting membrane potential and describe how to use the TensorFlow-Slim API to conduct AI-based EMG signal imaging recognition.

II. Methods

This study and the use of the needle EMG waveforms were approved by the Institutional Review Board of Chungnam National University Hospital (No. 2018-09-043). All needle EMG results obtained from January 2014 to July 2018 at a medical center were reviewed retrospectively. All measurements were performed using the Viking IV electrodiagnostic system (Nicolet Instrument Inc., Madison, WI, USA) and a monopolar needle electrode. It was difficult to measure stable membrane potentials for facial muscles, external anal muscles, and paravertebral muscles accurately; therefore, these results were excluded.

Because healthy muscle is electrically silent at rest, only ASAs were collected in denervated muscles. The ASAs found in needle EMG waveforms include positive sharp waves (PSW) and the fibrillation potential (Fibs), commonly associated with nerve and muscle problems, and the motor unit action potential. After individual waveforms were extracted, they were selected, reviewed, and classified by a specialist who had performed electrodiagnostic studies for more than 5 years. A total of 4,015 raw waveform data were reviewed, and 8,576 waveform images were collected for training with a convolutional neural network (CNN).

1. Installation and Settings

In this study, a personal computer with an Intel I7 8700 3.20 GHz processor, 16 GB DDR4 RAM, and NVIDIA GeForce GTX 1070 TI 8 GB is used. The used software consists of Anaconda with Python, TensorFlow, and the TensorFlow-Slim API. After installing Anaconda, at its prompt, we created a virtual environment using the command "conda create -n tensorflowgpu python = 3.5." Once the TensorFlowGPU environment was created, we entered the command "activate tensorflowgpu" at the prompt. Subsequently, we installed TensorFlow-GPU version [3], CUDA Toolkit v9.0 [4,5], and cuDNN [6].

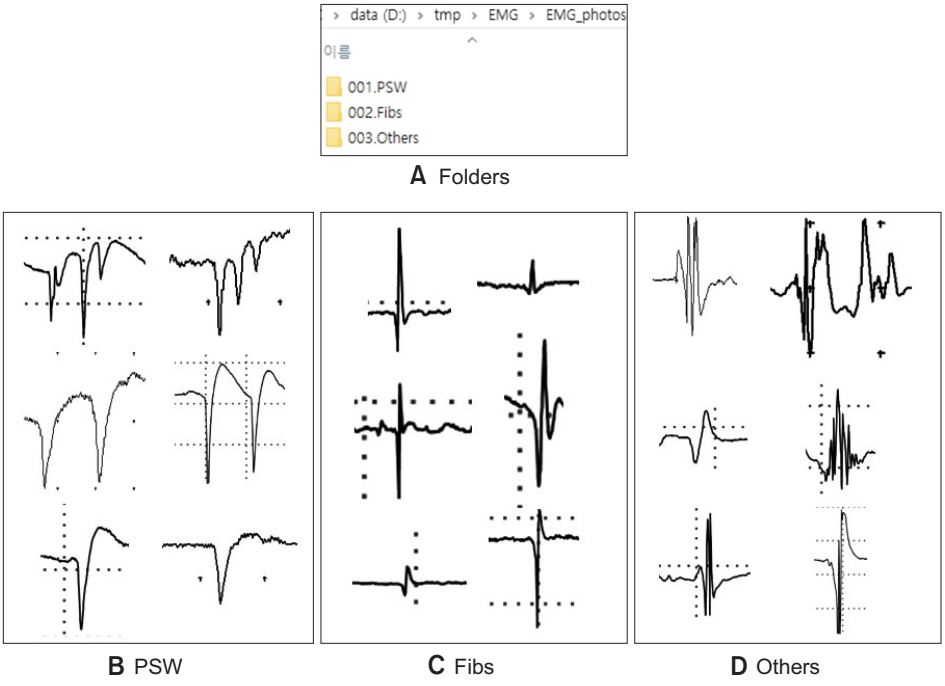


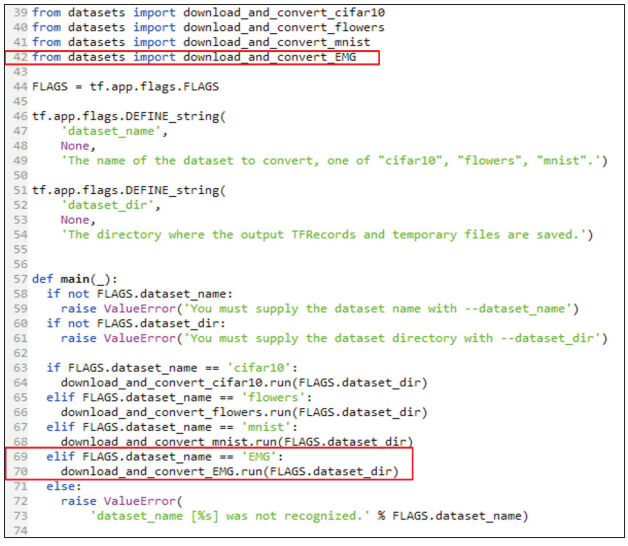
Figure 1. (A) Folders for artificial intelligence learning classified into three categories: (B) positive sharp wave (PSW) samples, (C) fibrillation potential (Fibs) samples, and (D) Other samples (e.g., motor unit action potential).

https://doi.org/10.4258/hir.2019.25.2.131

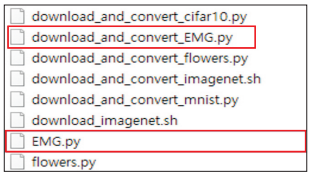
2. Creation of the TFRecord Dataset

We used the TensorFlow-Slim high-level API because it can be used to create desired datasets, train images, and to confirm results easily and quickly [7,8].

To train the desired images, they were initially classified by folder, as shown in Figure 1A, and the API code was modified. Next, we converted thee images into a TFRecord file format for training in TensorFlow. For this purpose, we modified a Download_and_convert_data.py file as shown in Figure 2A. Next, we copied download_and_convert_flowers.py and flowers.py in the Slim\datasets\ folder and renamed them to download_and_convert_EMG.py and EMG.py, respectively, as shown in Figure 2B. In download_and_convert_EMG.py, we modified _NUM_VALIDATION = 350 to

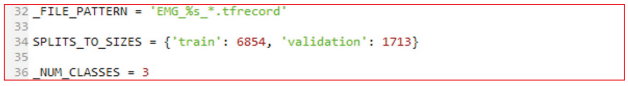


A Modifying "Download_and_convert_data.py"



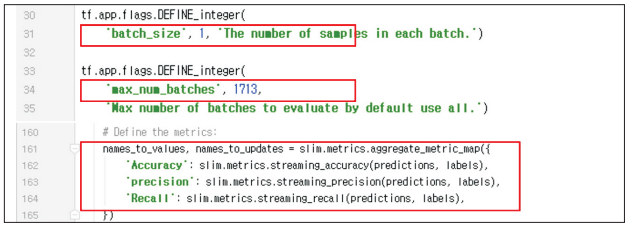
B "Download_and_convert_EMG.py, EMG.py"

C Modifying "Download_and_convert_EMG.py"



D Modifying "EMG.py"

E Modifying "dataset_factory.py"



F Modifying "eval_image_classifier.py"

Figure 2. The electromyography (EMG) waveform image was converted into a TFRecord dataset and the code was modified to proceed with customizing and learning: (A) modifying the code of Download_and_convert_data.py, (B) changing Download_and_convert_flowers.py and flowers.py, (C) modifying the code of Download_and_convert_EMG.py, (D) modifying the code of EMG.py, (E) modifying the code of Dataset_factory.py, and (F) modifying the code of eval_image_classifier.py.

_NUM_VALIDATION = 1713, as shown in Figure 2C, to set the validation data number. In general, 20% of the total data was used for validation. Thus, 1,713 of the 8,567 images were used for validation. Line 83 in Figure 2C set the path of the original data when creating the dataset. Line 102 in Figure 2C shows the code for setting a TFRecord-type output file name, and it was modified to replace Flower with EMG. In this study, data need not be downloaded separately because it was collected and arranged directly; therefore, Line 190 in Figure 2C was commented out. Next, EMG.py was modified as shown in Figure 2D. Lastly, code lines in dataset_factory.py were modified as shown in Figure 2E. We issued the command to create the dataset as described in Appendix 1.

3. Training

In this study, we trained data by fine-tuning a model from an existing checkpoint. We used the pretrained Inception v4 model, as shown in Figure 3 [9-11]. This model has a Top-1 accuracy rate of 80.2% and a Top-5 accuracy rate of 95.2%. Fine-tuning a model from an existing checkpoint involves two training processes. In the first process, only the last layer in the model was trained, and in the second process, the entire layer was relearned using the trained model obtained in the first process. In the pretrained model, because the last layer consisted of classes that differ from those (i.e., PSW, Fibs, Others) we want to classify, the existing layer structure was removed to classify three EMG images. To train the last layer, we created a my_checkpoints folder in a tmp folder. After copying a downloaded inception_v4.ckpt file into the folder, we trained it as described in Appendix 2. Next, we trained the entire layer as described in Appendix 3. As training progressed, the loss value for each step was outputted, as shown in Figure 4B, and training was performed according to the preset step size, as shown in Figure 4A.

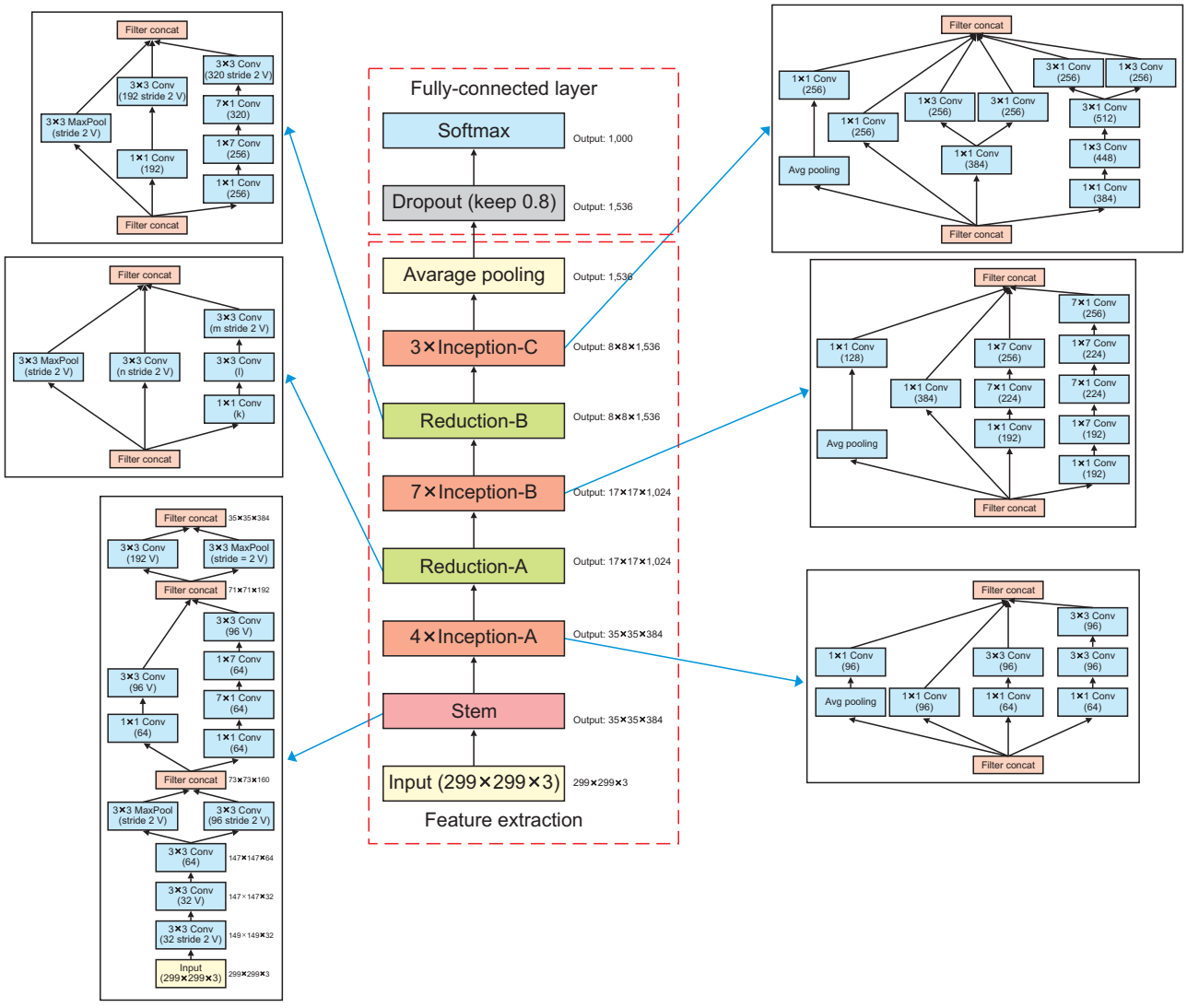
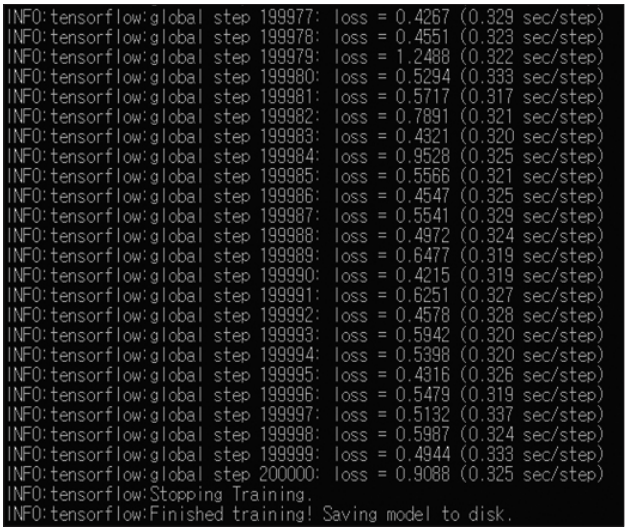


Figure 3. Overall schema for the pure Inception-v4 network used for image recognition.

https://doi.org/10.4258/hir.2019.25.2.131

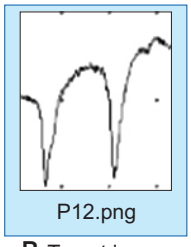


A Training process

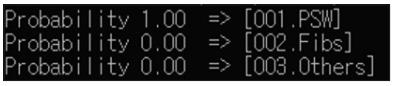
B Total loss

Figure 4. (A) A 200,000-step progress of all layers of model training, and (b) a convergence graph of total loss according to the progress.

A Running image_classification_emg.py for image classification



B Target image



C Result

Figure 5. Verification process using the completed model: (A) running the image_classification_emg.py file using the completed training model, (B) target image excluding validation images, and (C) image classification result.

4. Evaluation

To evaluate the trained model, we modified eval_image_classifier.py as shown in Figure 2F and performed evaluations as described in Appendix 4 to obtain the accuracy, precision, and recall values through the validation TFRecord file. In addition, we wrote codes to implement a function for a user to select data images for classification directly, as described in Appendix 5. The EMG image to evaluate had been selected, and it had been classified into one of three classes (i.e., PSW, Fibs, Others) and displayed with a certain probability, as shown in Figure 5.

III. Results

We applied a CNN-based image classification model using patients' needle EMG waveform signal image data. Figure 5 shows that the image corresponding to the PSW of the EMG signal waveforms can be recognized when it is applied to the trained model, and we found that 100% of cases of PSW were done correctly. Further, the performance evaluation results of the trained model using the validation dataset were as follows: accuracy = 93.8%, precision = 99.5%, and recall = 90.8%.

IV. Discussion

This paper presented the overall process for applying a CNN for image recognition using Python and the TensorFlow-Slim library. Specifically, we demonstrated that variable waveform image data could be recognized by the CNN-

based training of medical images. The proposed image recognition method was easily devised to analyze and use various medical images. This tutorial has been prepared for beginners who want to train and analyze images using a CNN; a CNN can play a significant role in accurately interpreting needle EMG diagnostic results if an elaborate waveform recognition algorithm is formulated in the future in combination with automatic waveform recognition and waveform digitizing techniques. If needle EMG is performed under standard conditions using established methods, CNN-based image classification can be used to support needle EMG diagnostic analyses.

In this preliminary study, waveform annotation was done manually. However, automatic waveform recognition and classification would be possible by imaging a report result sheet of medical signals and then training the corresponding image as a single object with an object detection API. As a result, the CNN can be applied to image data such as the waveforms of electrophysiological signals in a body, and the TensorFlow-Slim library can be used to train and recognize image data through simple coding rather than with TensorFlow.

Conflict of Interest

Hyoun-Joong Kong is an editor of Healthcare Informatics Research; however, he did not involve in the peer reviewer selection, evaluation, and decision process of this article. Otherwise, no potential conflict of interest relevant to this article was reported.

Acknowledgments

This work was supported by Chungnam National University Hospital Research Fund of 2017 (No. 2017-CF-015), and supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2019-2018-0-01833) supervised by the IITP (Institute for Information & communications Technology Promotion).

ORCID

Sangwoo Nam (http://orcid.org/0000-0003-2698-7766)
Min Kyun Sohn (http://orcid.org/0000-0002-2548-545X)
Hyun Ah Kim (http://orcid.org/0000-0002-1010-3279)
Hyoun-Joong Kong (http://orcid.org/0000-0001-5456-4862)
Il-Young Jung (http://orcid.org/0000-0001-8204-8195)

References

1. Lindstrom H, Ashworth NL. The usefulness of electrodiagnostic studies in the diagnosis and management of neuromuscular disorders. Muscle Nerve 2018;58(2):191-6.
2. Noorbakhsh-Sabet N, Zand R, Zhang Y, Abedi V. Artificial intelligence transforms the future of health care. Am J Med 2019 Jan 31 [Epub]. http://doi.org/10.1016/j.amjmed.2019.01.017.
3. NVIDIA CUDA Toolkit [Internet]. Santa Clara (CA): NVIDIA Corp.; c2019 [cited as 2019 Jan 15]. Available from: https://developer.nvidia.com/cuda-toolkit.
4. TensorFlow.org. GPU support [Internet]. [place unknown]: TensorFlow.org; c2019 [cited as 2019 Jan 15]. Available from: https://www.tensorflow.org/install/gpu.
5. NVIDIA cuDNN [Internet]. Santa Clara (CA): NVIDIA Corp.; c2019 [cited as 2019 Jan 15]. Available from: https://developer.nvidia.com/cudnn.
6. TensorFlow.org. Install TensorFlow with pip [Internet]. [place unknown]: TensorFlow.org; 2015 [cited at 2019 Apr 15]. Available from: https://www.tensorflow.org/install/pip?hl=ko.
7. Silberman N, Guadarrama S. TensorFlow-Slim image classification model library [Internet]. [place unknown]: GitHub Inc.; 2016 [cited at 2019 Apr 15]. Available from: https://github.com/tensorflow/models/tree/master/research/slim.
8. GitHub Inc. TensorFlow models [Internet]. [place unknown]: GitHub Inc.; c2019 [cited at 2019 Apr 15]. Available from: https://github.com/tensorflow/models.
9. Szegedy C, Ioffe S, Vanhoucke V, Alemi AA. Inception-v4, Inception-ResNet and the impact of residual connections on learning. Proceedings of the 31st AAAI Conference on Artificial Intelligence; 2017 Feb 4-9; San Francisco, CA. p. 4278-84.
10. Pham TC, Luong CM, Visani M, Hoang VD. Deep CNN and data augmentation for skin lesion classification. In: Nguyen N, Hoang D, Hong TP, Pham H, Trawinski B, editors. Intelligent information and database systems. Cham, Switzerland: Springer; 2018. p. 573-82.
11. GitHub Inc. Inception[3] Inception v4, Inception ResNet and the Impact of Residual Connections on Learning(2016) [Internet]. [place unknown]: GitHub Inc.; 2018 [cited at 2019 Apr 15]. Available from: https://github.com/hwkim94/hwkim94.github.io/wiki.

Appendix 1. Create the TFRecord file using EMG image data

```
Python download_and_convert_data.py
--dataset_name=EMG
--dataset_dir=/tmp/EMG
```

Appendix 2. Training a model from parameter values

```
Python train_image_classifier.py
--train_dir=\tmp\train_inception_v4_EMG_FineTune_logs
--dataset_name=EMG
--dataset_split_name=train
--dataset_dir=\tmp\EMG
--model_name=inception_v4
--checkpoint_path=\tmp\my_checkpoints\inception_v4.ckpt
--checkpoint_exclude_scopes=InceptionV4/Logits
--trainable_scopes=InceptionV4/Logits
--max_number_of_steps=1000
--batch_size=16
--learning_rate=0.01
--learning_rate_decay_type=fixed
--save_interval_secs=60
--save_summaries_secs=60
--log_every_n_steps=1
--optimizer=rmsprop
--weight_decay=0.00004
```

Appendix 3. Fine-tuning a model from an existing checkpoint

```
Python train_image_classifier.py
--train_dir=d:\tmp\train_inception_v4_EMG_FineTune_logs\all
--dataset_name=EMG
--dataset_split_name=train
--dataset_dir=d:\tmp\EMG
--model_name=inception_v4
--checkpoint_path=d:\tmp\train_inception_v4_EMG_FineTune_logs
--max_number_of_steps=200000
--batch_size=10
--learning_rate=0.00005
--learning_rate_decay_type=fixed
--save_interval_secs=60
--save_summaries_secs=60
--log_every_n_steps=1
--optimizer=rmsprop
--weight_decay=0.00004
```

Appendix 4. Fine-tuning a model evaluation

```
Python eval_image_classifier.py -alsologtostderr
--checkpoint_path=d:\tmp\train_inception_v4_EMG_FineTune_logs\all\
--dataset_dir=d:\tmp\EMG
--dataset_name=EMG
```

```
--dataset_split_name=validation
--model_name=inception_v4
```

Appendix 5. EMG signal image classification (image_classification_emg.py)

```
from matplotlib import pyplot as plt
import numpy as np
import os
import tensorflow as tf
from nets import inception
from preprocessing import inception_preprocessing
from tkinter import Tk
from tkinter.filedialog import askopenfilename
Tk().withdraw()
checkpoints_dir = 'D:\\tmp\\train_inception_v4_EMG_FineTune_logs\\all'
slim = tf.contrib.slim
image_size = inception.inception_v4.default_image_size
while True:
    file = askopenfilename()
    if file == "":
        break
    str(file)
    print(file)
    with tf.Graph().as_default():
        image_input = tf.read_file(file)
        image = tf.image.decode_jpeg(image_input, channels=3)
        processed_image = inception_preprocessing.preprocess_image(image, image_size,  image_size, is_training=False)
        processed_images = tf.expand_dims(processed_image, 0)
        with slim.arg_scope(inception.inception_v4_arg_scope()):
           logits, _ = inception.inception_v4(processed_images, num_classes=3, is_training=False)
        probabilities = tf.nn.softmax(logits)
        init_fn = slim.assign_from_checkpoint_fn(os.path.join(checkpoints_dir, 'model.ckpt-200000'), slim.get_model_
         variables('InceptionV4'))
        with tf.Session() as sess:
            init_fn(sess)
            np_image, probabilities = sess.run([image, probabilities])
            probabilities = probabilities[0, 0:]
            sorted_inds = [i[0] for i in sorted(enumerate(-probabilities), key=lambda x: x[1])]
        plt.figure()
        plt.imshow(np_image.astype(np.uint8))
        plt.axis('off')
        names = os.listdir("D:\\tmp\\EMG\\EMG_photos")
        for i in range(3):
            index = sorted_inds[i]
            print('Probability %0.2f%% => [%s]' % (probabilities[index], names[index]))
```

 https://doi.org/10.4258/hir.2019.25.2.131