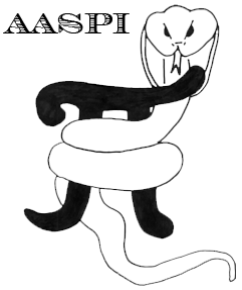


IMAGE CLASSIFICATION WITH CONVOLUTIONAL NEURAL NETWORKS – PROGRAM cnn_image_classification

Contents

Overview

Recent convolutional neural network (CNN) research has yielded significant improvements and unprecedented accuracy in image classification. The annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC, Russakovsky et al. (2015)) is an international competition in computer vision, that provides the benchmark for the field. Specific CNN architectures have been the leading approach for several years now (e.g., Szegedy et al., 2014; Chollet, 2016; He et al., 2016; Huang et al., 2016; Sandler et al., 2018). Eventually, the success in the "pure" computer vision field reached distinct research communities. Researchers noted that the parameters learned by the layers in many CNN models trained on images, exhibit a common behavior – layers closer to the input data tend to learn general features, such as edge detecting/enhancing filters or color blobs, then there is a transition to more specific dataset features, such as faces, feathers, or object parts (Yosinski et al., 2014; Yin et al., 2017). These general-specific CNN layer properties are important points to be considered for the implementation of transfer learning (Caruana, 1995; Bengio, 2012; Yosinski et al., 2014). In transfer learning, first a CNN model is trained on a base dataset for a specific task, then we repurpose the learned features (the model parameters), or transfer them, to a second target CNN to be trained on a different target dataset and task (Yosinski et al., 2014). Figure 1 shows a simple flowchart for the transfer learning and process. This program uses transfer learning with images provided by the users. The CNN models were trained on the ILSVRC dataset (the base dataset) and are used as feature extractors to classify the new images provided. This application falls in the realm of supervised machine learning, as the user provides data and labels and the application tries to find a relationship between them.

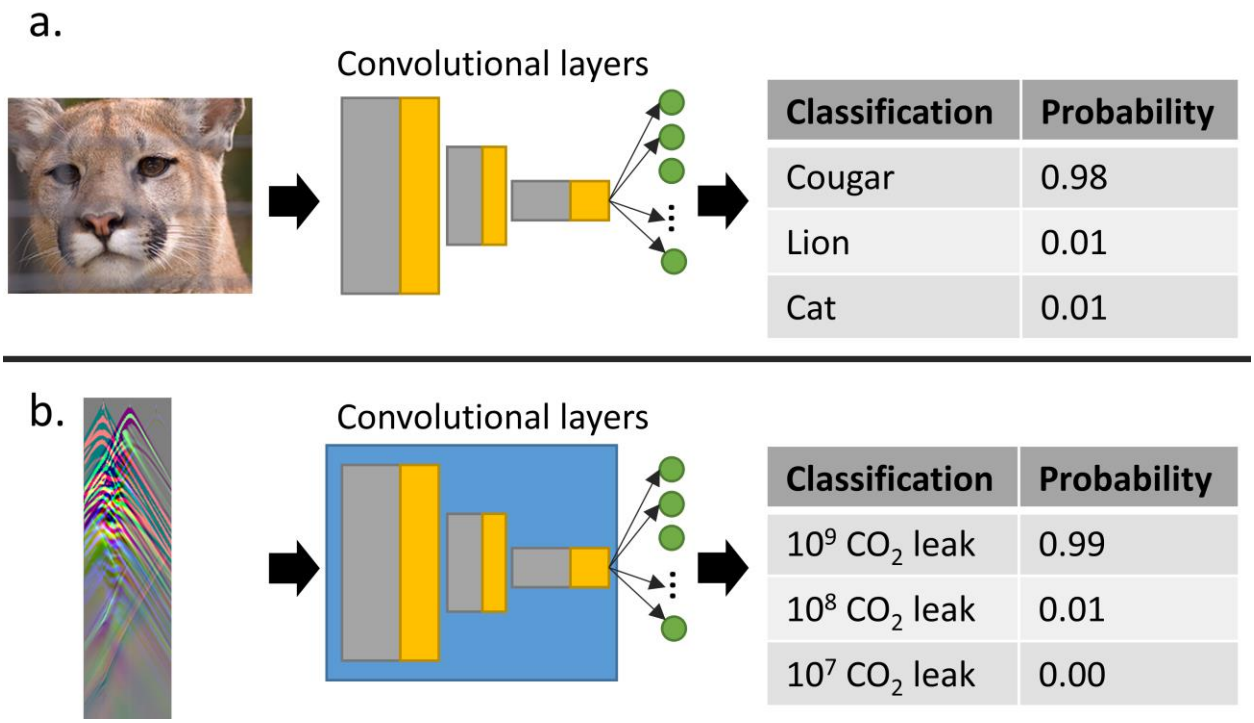


Figure 1. Visual representation of the transfer learning process. We generically represent convolutional and pooling layers with gray and golden rectangles whereas green circles represent densely connected neurons, commonly used as classification layers. Base task (a.) in this case represents an image from the ILSVRC going through a generic CNN model (convolutional layers and classification layers) trained on the same dataset. The CNN model then outputs the probability of the image belonging to one of the thousands of classes of the ILSVRC. For the transfer learning process, we copy the parameters learned by the convolutional layers on the base task using the blue rectangle as a feature extractor, in this example, a pseudo-RGB seismic image. We then train a new classification layer based on the extracted features. The final classification (amount of CO_2 leaked) depends on the secondary task.

Computation flow chart

To execute the transfer learning process using **cnn_image_classfication**, the user needs to provide a root folder containing examples of each one of the classes to be classified in subfolders (Figure 2). We suggest at least 100 examples (pictures) for each one of the classes. The flow chart on the following page shows an overview of the complete process.

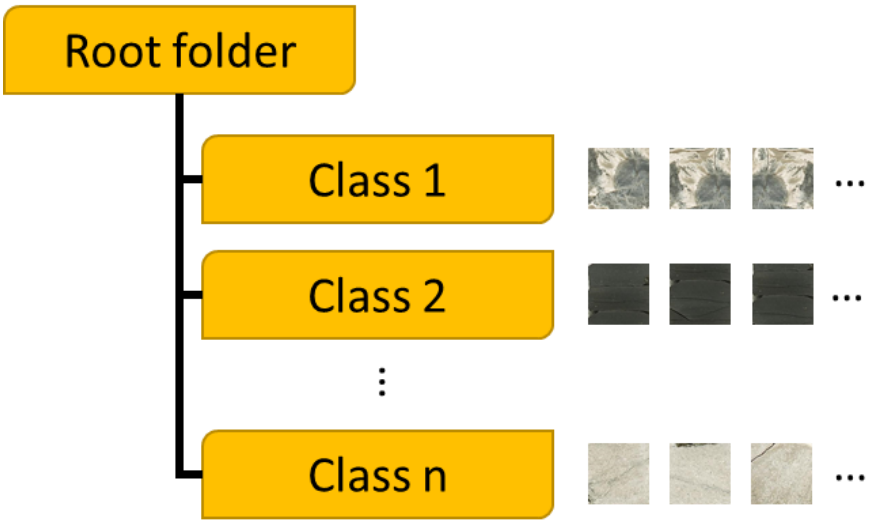
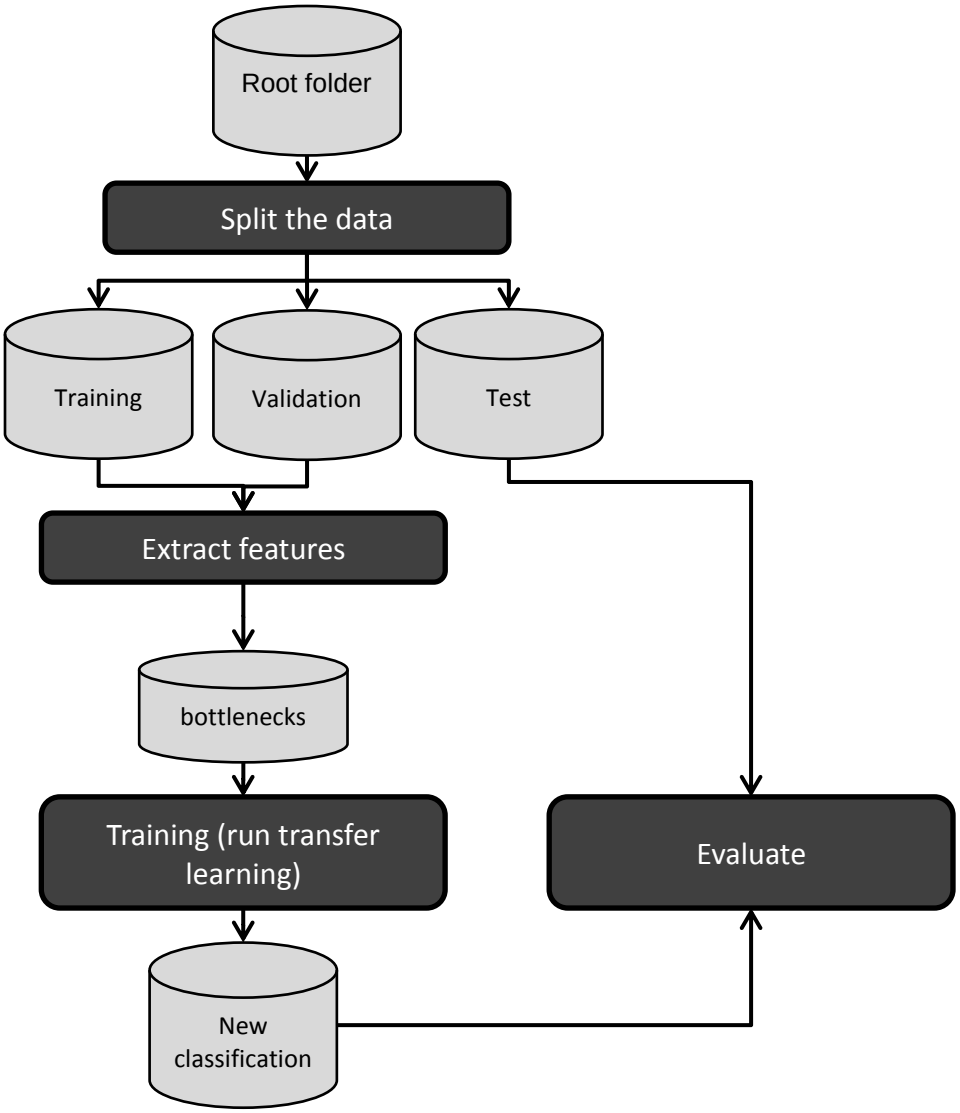


Figure 2. Cartoon showing how the data should be stored for **cnn_image_classification**. Each target class requires its own subfolder containing the corresponding class "examples". We recommend providing at least 100 examples for each class.



Invoking the cnn_image_classification GUI

The **cnn_image_classification** GUI is under the **aaspi_util** *Machine Learning Toolbox* tab. (Figure 3). The program is divided in three tabs: Split Data, Transfer Learning, Test Model. The user needs to specify the root input folder containing subfolders (different classes). Each one of the subfolders should have "examples" of the classes where the examples are provided as in jpg-, gif-, or png-format image files. The program can be used to split the data into training, validation, and test sets (Figure 4). After the data is split, the user computes the bottlenecks (extract the features) and run transfer learning (Figure 5). The last step is to evaluate the classification performed (Figure 6).

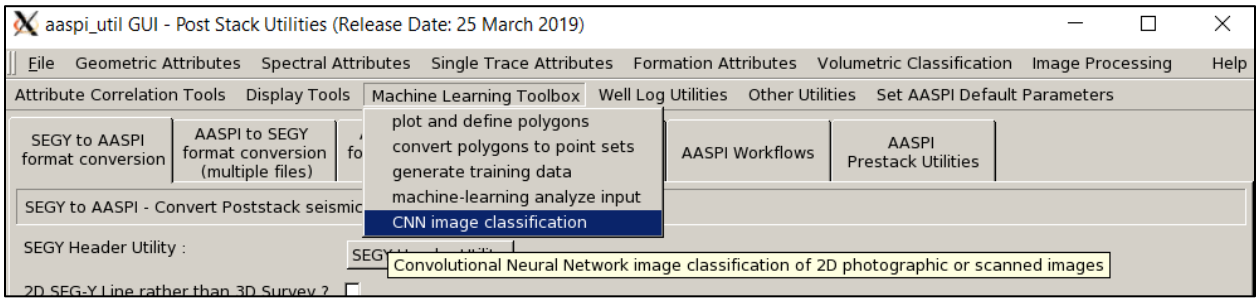


Figure 3**.** The **cnn_image_classification** GUI is under the **aaspi_util** *Machine Learning Toolbox* tab.

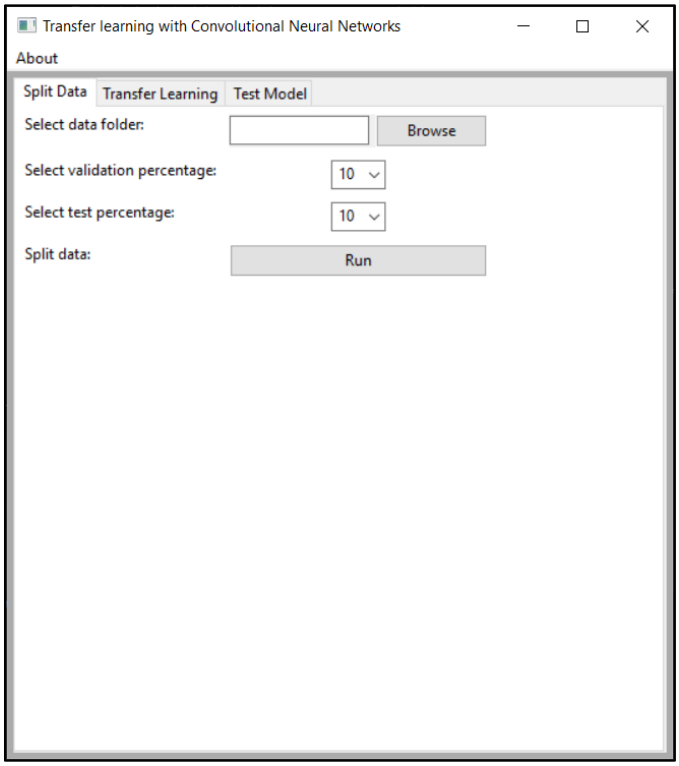


Figure 4. **cnn_image_classification** program *Split Data* tab. Browse to select the root input folder (containing subfolders with classes to be classified), select the percentage of data to be separated for validation and for test. Press "Run" to split the data with the chosen parameters.

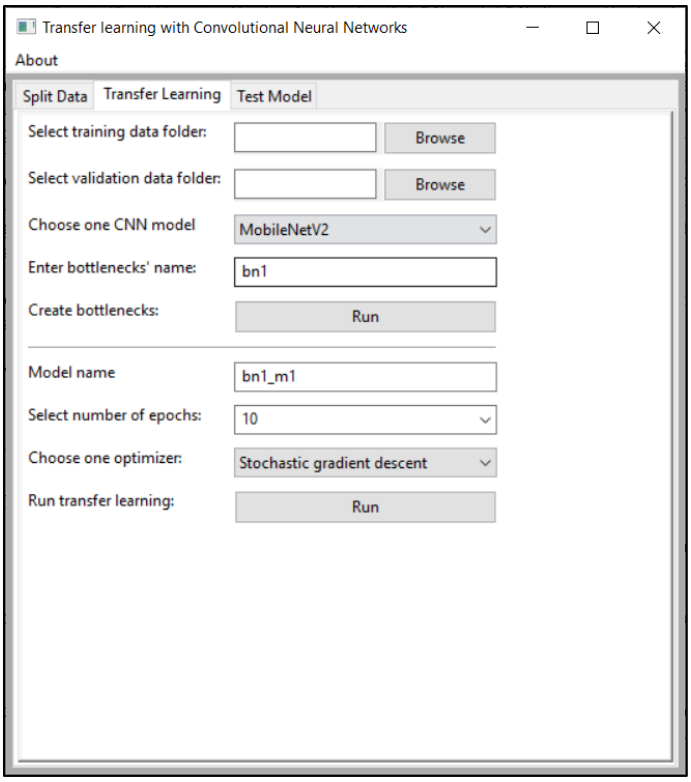


Figure 5. Transfer Learning tab: Specify the training and validation data folders. Select one CNN model from the list. Enter a name for the bottleneck to be created. Click "Run" and the program extracts the features from the selected data using the specified CNN model. Next, choose a model name, select the number of epochs (iterations), select one of the optimizers available, and Run transfer learning. Important note: when "Run transfer learning – Run" button is clicked, the bottlenecks created on the previous step (the name written in the "Enter bottlenecks' name" box) and the used model ("Choose one CNN model") are used to assemble the final retrained model. Therefore, be careful to not create bottlenecks with "model A" and run transfer learning with "model B" (results could still be reasonable with such mistake).

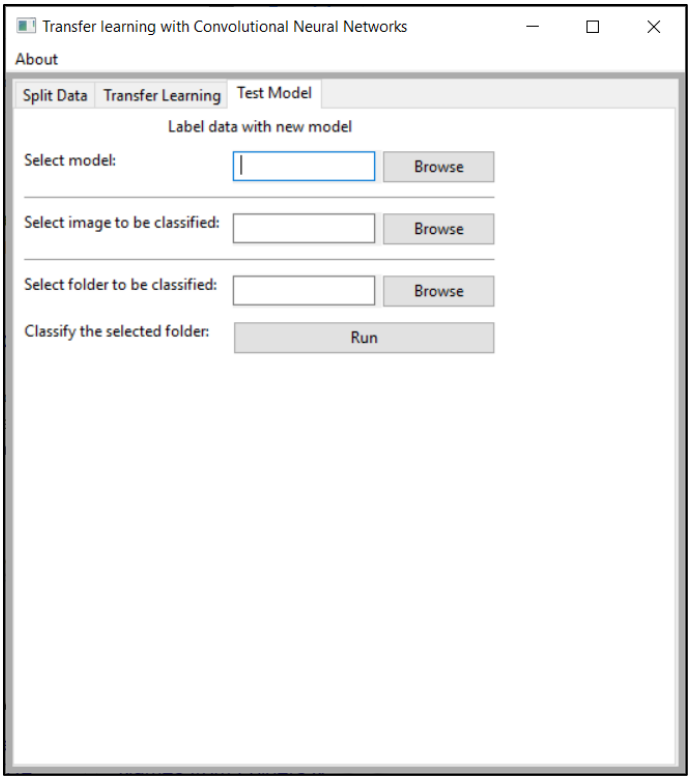
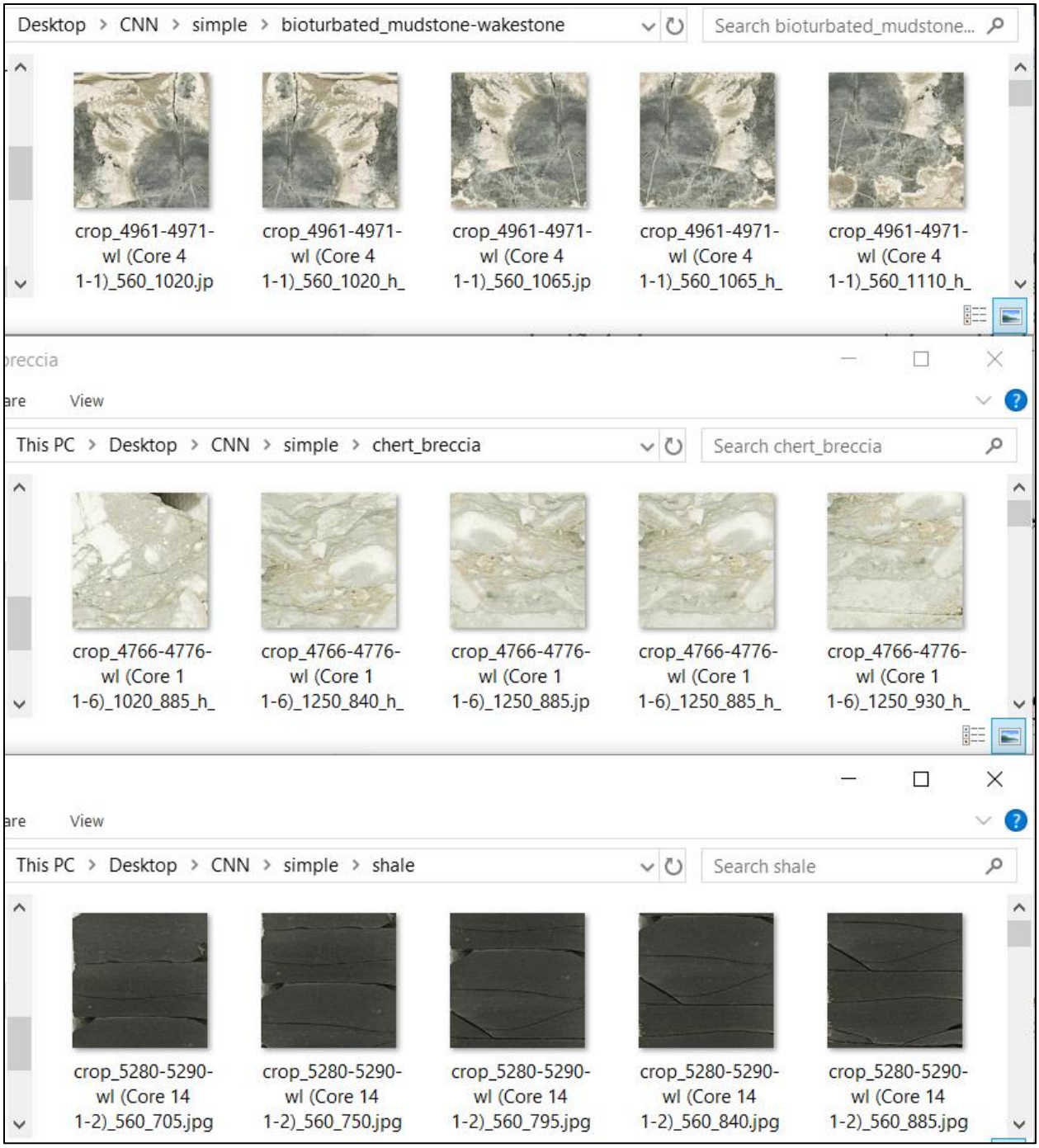


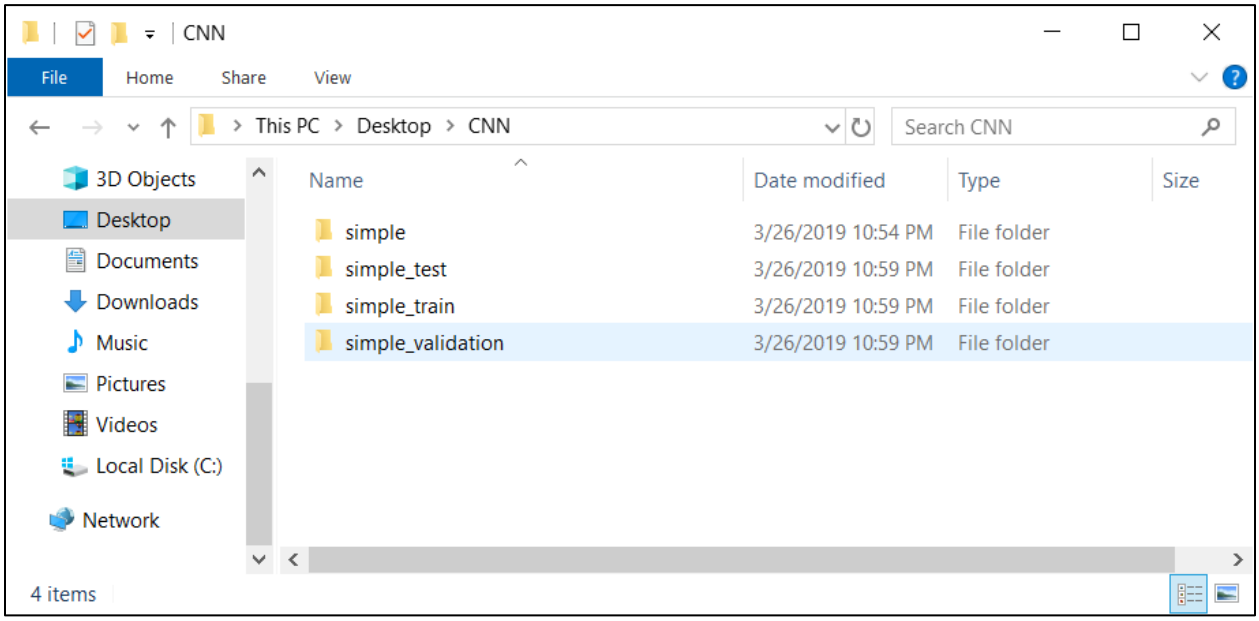
Figure 6. On the final stage, the user can evaluate the retrained model. Select the trained model. The user can then: select a single image to be classified or select a whole folder (with the same structure as before -root and subfolders) to be classified. When selecting a single image to be classified, the program opens a window with the resulting classification. When a folder is classified, a csv file is saved with resulting values.

Example: Lithofacies classification

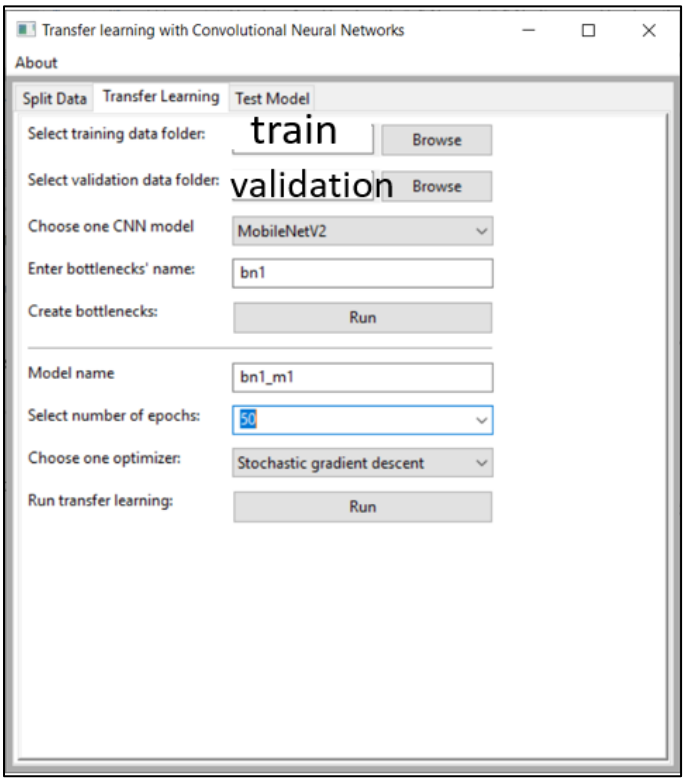
For this tutorial, we selected a few images from Pires de Lima et. al (2019) to show how to use the program. The dataset is composed of four classes (four different lithofacies). The following image shows a simple visualization of part of the dataset. Each one of the classes contains at least 40 examples (images).


Desktop > CNN > simple > bioturbated_mudstone-wakestone
Search bioturbated_mudstone...
crop_4961-4971-wl (Core 4 1-1)_560_1020.jp
crop_4961-4971-wl (Core 4 1-1)_560_1020_h_
crop_4961-4971-wl (Core 4 1-1)_560_1065.jp
crop_4961-4971-wl (Core 4 1-1)_560_1065_h_
crop_4961-4971-wl (Core 4 1-1)_560_1110_h_
breccia
are View
This PC > Desktop > CNN > simple > chert_breccia
Search chert_breccia
crop_4766-4776-wl (Core 1 1-6)_1020_885_h_
crop_4766-4776-wl (Core 1 1-6)_1250_840_h_
crop_4766-4776-wl (Core 1 1-6)_1250_885.jp
crop_4766-4776-wl (Core 1 1-6)_1250_885_h_
crop_4766-4776-wl (Core 1 1-6)_1250_930_h_
are View
This PC > Desktop > CNN > simple > shale
Search shale
crop_5280-5290-wl (Core 14 1-2)_560_705.jpg
crop_5280-5290-wl (Core 14 1-2)_560_750.jpg
crop_5280-5290-wl (Core 14 1-2)_560_795.jpg
crop_5280-5290-wl (Core 14 1-2)_560_840.jpg
crop_5280-5290-wl (Core 14 1-2)_560_885.jpg

After splitting the data (*Split Data* tab), three new folders are added at the same location as the root folder:



I then select the *Transfer Learning* tab and complete it with the newly created _train and _validation folders created. I also select MobileNetV2, enter the name for the bottlenecks:



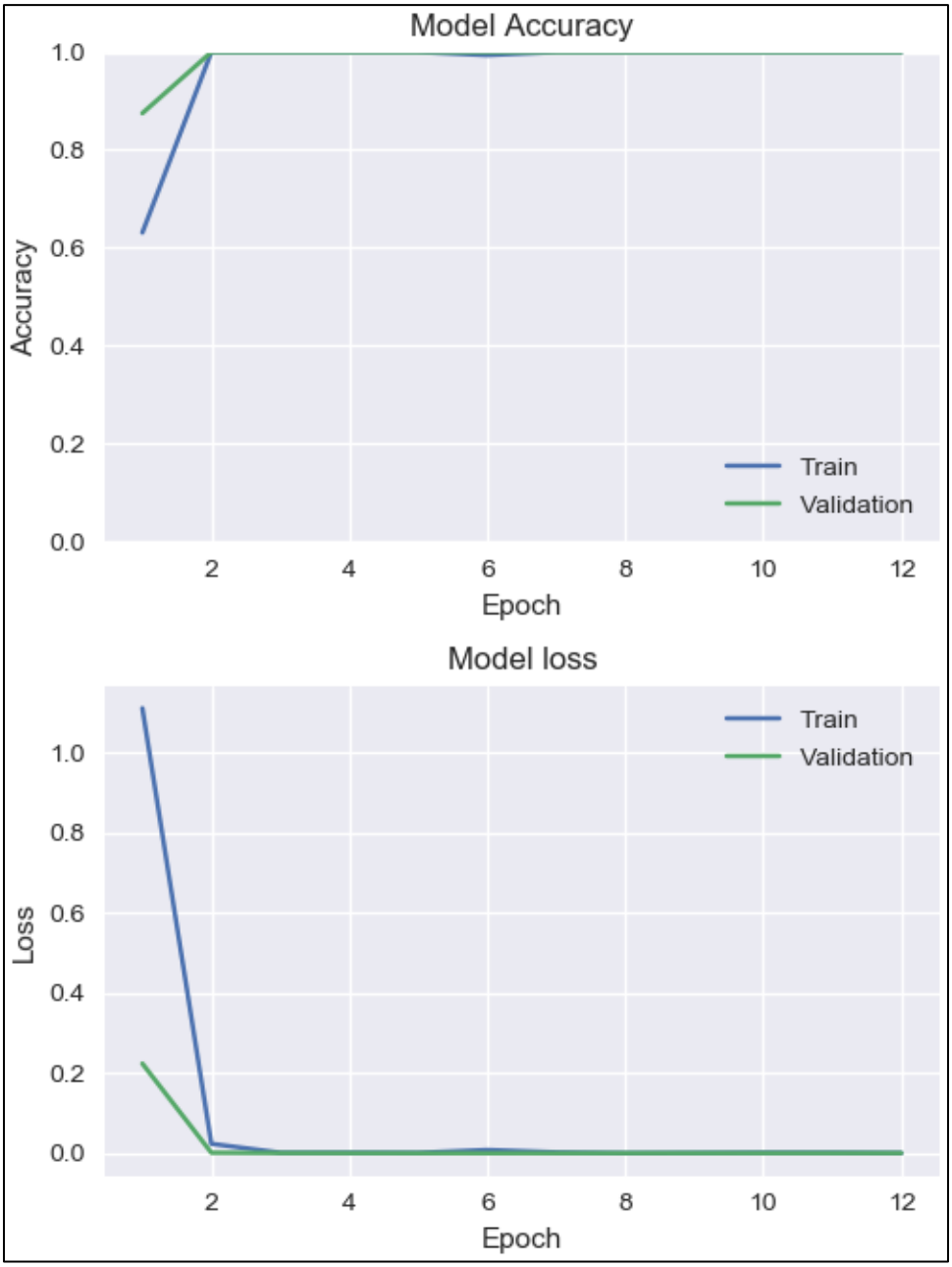
When I click "Create bottlenecks: Run" the program outputs information about its execution:

```
MobileNetV2
bn1
Initiating feature extraction...
2019-03-26 23:05:18.221602: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
2019-03-26 23:05:19.298817: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1432] Found device 0 with properties:
name: GeForce GTX 1050 major: 6 minor: 1 memoryClockRate(GHz): 1.493
pciBusID: 0000:01:00.0
totalMemory: 4.00GiB freeMemory: 3.30GiB
2019-03-26 23:05:19.299956: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1511] Adding visible gpu devices: 0
2019-03-26 23:05:23.073556: I tensorflow/core/common_runtime/gpu/gpu_device.cc:982] Device interconnect StreamExecutor with strength 1 edge matrix:
2019-03-26 23:05:23.073827: I tensorflow/core/common_runtime/gpu/gpu_device.cc:988]      0
2019-03-26 23:05:23.074036: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1001] 0:   N
2019-03-26 23:05:23.079069: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 3018 MB memory) -> physical GPU (devi
MobileNetV2 loaded
Found 160 images belonging to 4 classes.

 1/10 [==>............................] - ETA: 31s
 2/10 [=====>........................] - ETA: 14s
 3/10 [========>.....................] - ETA: 8s
 4/10 [==========>.....................] - ETA: 5s
 5/10 [=============>................] - ETA: 3s
 6/10 [================>.............] - ETA: 2s
 7/10 [==================>...........] - ETA: 1s
 8/10 [====================>.........] - ETA: 1s
 9/10 [=======================>......] - ETA: 0s
10/10 [==============================] - 4s 431ms/step
Found 20 images belonging to 4 classes.

1/1 [==============================] - 0s 122ms/step
Process complete.
```

With the bottlenecks created, I choose a model name, select the number of epochs, an optimizer, and “Run transfer learning – Run”. The program outputs information about its execution again:

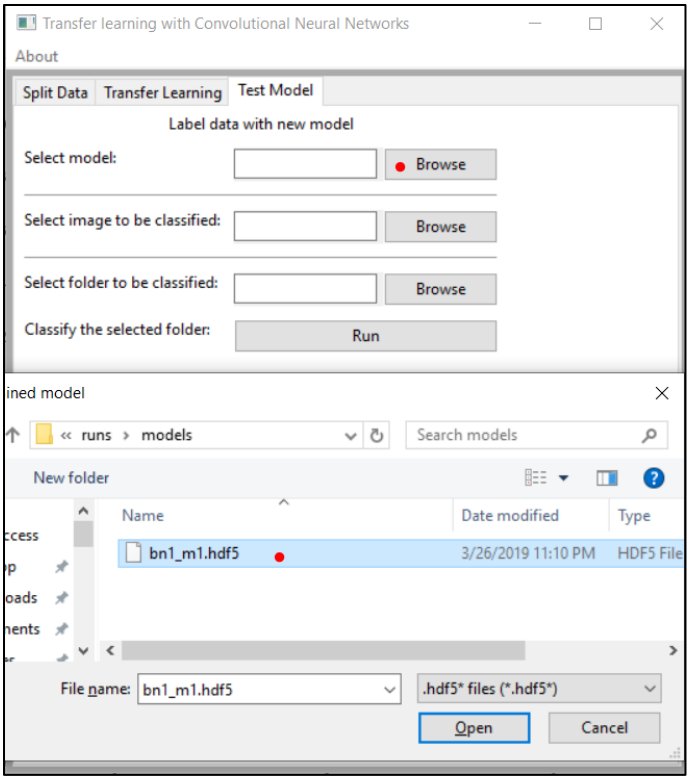
```
bn1
bn1_m1
50
Stochastic gradient descent
Initiating training...
Train on 160 samples, validate on 16 samples
Epoch 1/50

 16/160 [==>............................] - ETA: 13s - loss: 2.2256 - acc: 0.2500
112/160 [====================>.........] - ETA: 0s - loss: 1.4537 - acc: 0.5089
160/160 [==============================] - 2s 11ms/step - loss: 1.1107 - acc: 0.6312 - val_loss: 0.2240 - val_acc: 0.8750
Epoch 2/50

 16/160 [==>............................] - ETA: 0s - loss: 0.0998 - acc: 1.0000
 96/160 [=================>............] - ETA: 0s - loss: 0.0357 - acc: 1.0000
160/160 [==============================] - 0s 667us/step - loss: 0.0238 - acc: 1.0000 - val_loss: 0.0015 - val_acc: 1.0000
Epoch 3/50

 16/160 [==>............................] - ETA: 0s - loss: 0.0051 - acc: 1.0000
 96/160 [=================>............] - ETA: 0s - loss: 0.0021 - acc: 1.0000
160/160 [==============================] - 0s 904us/step - loss: 0.0015 - acc: 1.0000 - val_loss: 8.3534e-04 - val_acc: 1.0000
Epoch 4/50

 16/160 [==>............................] - ETA: 0s - loss: 1.3713e-04 - acc: 1.0000
 80/160 [==============>...............] - ETA: 0s - loss: 0.0017 - acc: 1.0000
160/160 [==============================] - 0s 812us/step - loss: 0.0016 - acc: 1.0000 - val_loss: 6.8110e-04 - val_acc: 1.0000
Epoch 5/50

 16/160 [==>............................] - ETA: 0s - loss: 0.0015 - acc: 1.0000
 96/160 [=================>............] - ETA: 0s - loss: 0.0017 - acc: 1.0000
160/160 [==============================] - 0s 749us/step - loss: 0.0012 - acc: 1.0000 - val_loss: 5.2760e-04 - val_acc: 1.0000
Epoch 6/50
```

After training, the program outputs the results in a graph:



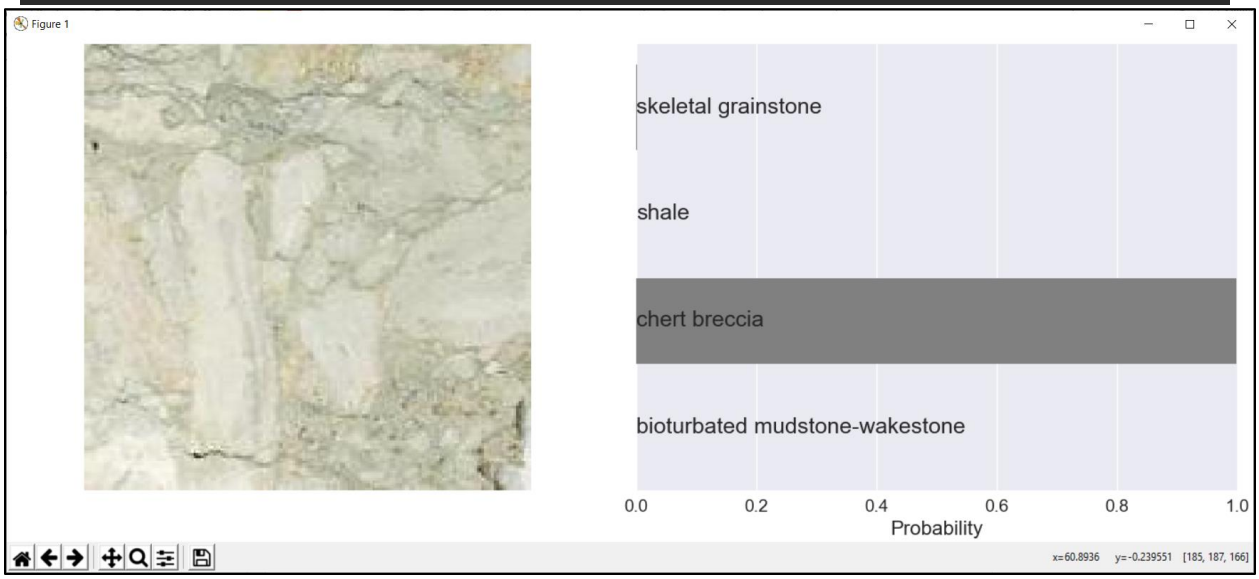
Because this is a very easy dataset for the CNN, the results are perfect. Train and validation scores overlap each other (indicating that there's no over fitting), accuracy is 1.0 (all examples were classified according to the provided classes) and loss is very low (0.0 loss indicates "perfect" results).

With the new model trained, I can evaluate its results on the test set (when the program runs, a folder is created under runs/models, models are saved in that folder):

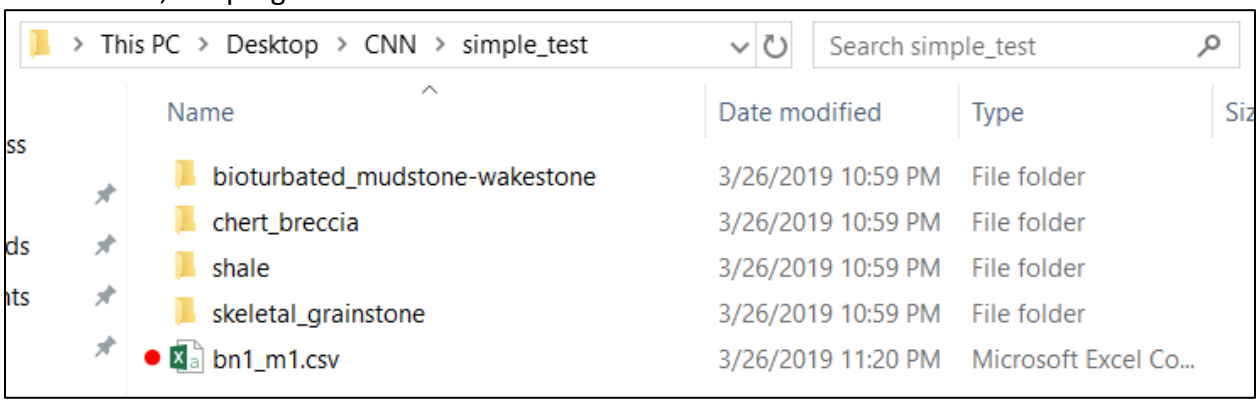


I then select a single image from the test set and the program outputs the results:

Class	Probability
bioturbated_mudstone-wakestone	0.0008
chert_breccia	0.9981
shale	0.0000
skeletal_grainstone	0.0012



When I select the test folder under "Select folder to be classified", and "Classify the selected folder: Run", the program saves a csv file in the test folder location:



The csv file contains the probability assigned for each one of the classes for each one of the files found in the test folder, as well as the file name.

	bioturbated_mudstone-wakestone	chert_breccia	shale	skeletal_grainstone	file
0	1.00	0.00	0.00	0.00	bioturbated_mudstone-wakestone\crop_4961-4971-wl (Core 4 1-1)_1250_615_h_flip.j
1	1.00	0.00	0.00	0.00	bioturbated_mudstone-wakestone\crop_4961-4971-wl (Core 4 1-1)_560_1020.jpg
2	1.00	0.00	0.00	0.00	bioturbated_mudstone-wakestone\crop_4961-4971-wl (Core 4 1-1)_560_1245.jpg
3	1.00	0.00	0.00	0.00	bioturbated_mudstone-wakestone\crop_4961-4971-wl (Core 4 1-1)_790_345_h_flip.jp
4	0.00	1.00	0.00	0.00	chert_breccia\crop_4766-4776-wl (Core 1 1-6)_1250_885.jpg
5	0.00	1.00	0.00	0.00	chert_breccia\crop_4776-4780-wl (Core 1 1-7)_330_1200_h_flip.jpg
6	0.00	1.00	0.00	0.00	chert_breccia\crop_4776-4780-wl (Core 1 1-7)_330_615_h_flip.jpg
7	0.00	1.00	0.00	0.00	chert_breccia\crop_4776-4780-wl (Core 1 1-7)_330_885_h_flip.jpg
8	0.00	0.00	1.00	0.00	shale\crop_5290-5300-wl (Core 14 1-3)_1020_1155.jpg
9	0.00	0.00	1.00	0.00	shale\crop_5310-5320-wl (Core 14 1-5)_330_1155.jpg
10	0.00	0.00	1.00	0.00	shale\crop_5310-5320-wl (Core 14 1-5)_330_1200.jpg
11	0.00	0.00	1.00	0.00	shale\crop_5310-5320-wl (Core 14 1-5)_330_390.jpg
12	0.00	0.00	1.00	0.00	shale\crop_5310-5320-wl (Core 14 1-5)_560_1020.jpg
13	0.00	0.00	1.00	0.00	shale\crop_5310-5320-wl (Core 14 1-5)_560_345.jpg
14	0.00	0.00	1.00	0.00	shale\crop_5310-5320-wl (Core 14 1-5)_560_705.jpg
15	0.00	0.00	0.00	1.00	skeletal_grainstone\crop_4780-4790-wl (Core 2 1-1)_790_525_h_flip.jpg
16	0.00	0.00	0.00	1.00	skeletal_grainstone\crop_4780-4790-wl (Core 2 1-1)_790_615.jpg
17	0.00	0.00	0.00	1.00	skeletal_grainstone\crop_4780-4790-wl (Core 2 1-1)_790_615_h_flip.jpg
18	0.00	0.00	0.00	1.00	skeletal_grainstone\crop_4930-4940-wl (Core 3 1-7)_560_480_h_flip.jpg
19	0.00	0.00	0.00	1.00	skeletal_grainstone\crop_4930-4940-wl (Core 3 1-7)_560_615.jpg

This transfer learning approach was used in Pires de Lima et al. (2019a, 2019b, 2019c and 2019d).

References

Caruana, R., 1995, Learning Many Related Tasks at the Same Time with Backpropagation, *in* G. Tesauro, D. S. Touretzky, and T. K. Leen, eds., Advances in Neural Information Processing Systems 7: MIT Press, 657–664.

Chollet, F., 2016, Xception: Deep Learning with Depthwise Separable Convolutions: CoRR, abs/1610.0.

He, K., X. Zhang, S. Ren, and J. Sun, 2016, Deep Residual Learning for Image Recognition, in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR): IEEE, 770–778, doi:10.1109/CVPR.2016.90.

Huang, G., Y. Sun, Z. Liu, D. Sedra, and K. Weinberger, 2016, Deep Networks with Stochastic Depth.

Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, 2015, ImageNet Large Scale Visual Recognition Challenge: International Journal of Computer Vision, 115, 211–252, doi:10.1007/s11263-015-0816-y.

Sandler, M., A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, 2018, MobileNetV2: Inverted Residuals and Linear Bottlenecks: ArXiv e-prints.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, 2014, Going Deeper with Convolutions: CoRR, abs/1409.4.

Yin, X., W. Chen, X. Wu, and H. Yue, 2017, Fine-tuning and visualization of convolutional neural networks, in 2017 12th IEEE Conference on Industrial Electronics and Applications (ICIEA): IEEE, 1310–1315, doi:10.1109/ICIEA.2017.8283041.

Yosinski, J., J. Clune, Y. Bengio, and H. Lipson, 2014, How transferable are features in deep neural networks? Advances in Neural Information Processing Systems, 27, 3320–3328.

(Ahead of print/In review/Submitted)

Pires de Lima, R. A., F. Suriamin, K. J. Marfurt, and M. J. Pranter, 2019a, Convolutional neural networks as an aid in core lithofacies classification: Interpretation, accepted, ahead of print.

Pires de Lima, R. A., K. J. Marfurt, D. Duarte Coronado, and A. Bonar, 2019b, Progress and challenges in deep learning analysis of geoscience images: *in* EAGE Annual Meeting Expanded Abstracts (accepted).

Pires de Lima, R. A., A. Bonar, D. Duarte Coronado, K. J. Marfurt, and C. Nicholson, 2019c, Deep convolutional neural networks as a geological image classification tool (in review).

Pires de Lima, R. A., Y. Lin, K. J. Marfurt, 2019d, Transforming seismic data into pseudo-RGB images to predict CO2 leakage using pre-learned convolutional neural networks weights: in SEG Expanded Abstract (submitted).