COT 5930 Digital Image Processing Spring 2021
Assignment 3
Christian Garbin

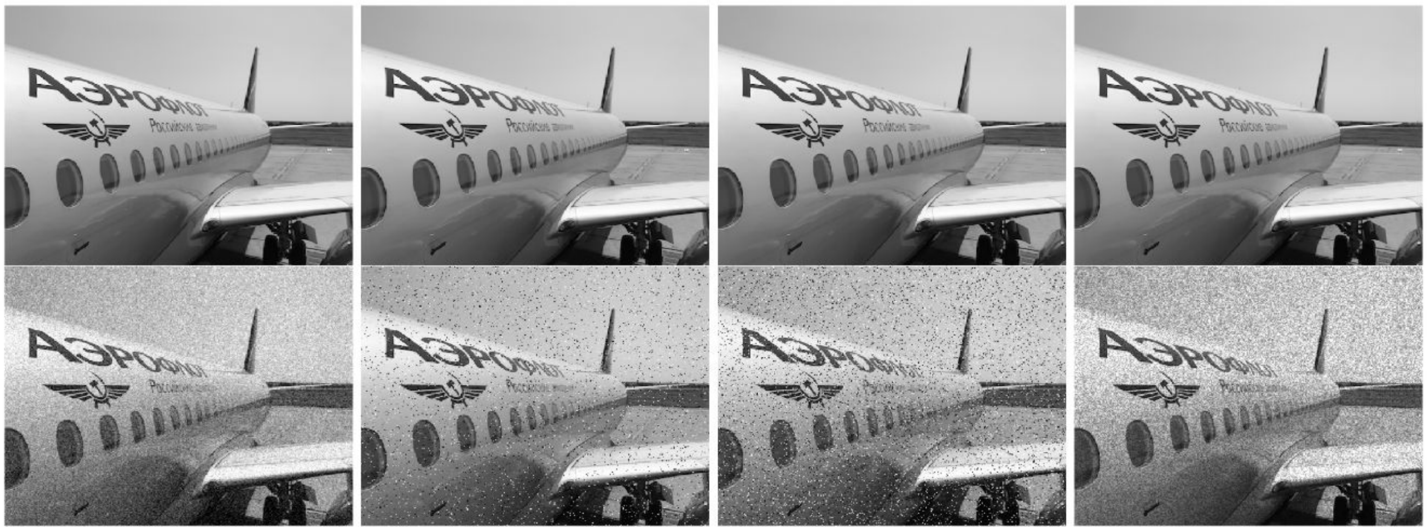
Step 3

Expand "Part 1" of the starter code to include other test images, noise types (and their parameters) and image quality metrics.

In this step we applied four types of noises to a selected grayscale image:

1. Gaussian white noise with the default values (zero mean and 0.01 variance).
2. Salt and pepper noise with density 0.05.
3. Salt and pepper noise with density 0.1.
4. Speckle noise with the default values (zero mean and 0.05 variance).

The following figure shows the resulting images, in the order above from left to right. The top row is the original image (repeated) for comparison.



The following table, extracted from the live script, shows the quality metrics for each picture.

Note that the first row is the original image. Some of the metrics (e.g. MSE, SNR, and SSIM) may not make much sense in that case. It was done this one to simplify the code.

We can make a few observations, based on the metrics shown in the following table:

1. The original image has a significantly lower NIQE, as expected.
2. Speckle noise with the default variance of 0.05 and a 5% salt and pepper noise produce similar quality metrics, except for SSIM and NIQUE. The larger NIQE for the speckle image can be noticed by the much more perturbed sky behind the airplane's tail (last column above).

Image	MSE	PeakSNR	SNR	SSIM	BRISQUE	NIQE	PIQE
'Original'	0	Inf	Inf	1.0000	25.1869	3.6370	52.6345
'Gaussian'	617.9616	20.2212	16.4431	0.2428	47.2423	16.0111	70.8140
'5% Salt & Pepper'	1.0162e+03	18.0612	14.2831	0.3110	47.0461	20.1259	74.8934
'10% Salt & Pepper'	2.0414e+03	15.0316	11.2535	0.1676	43.4677	23.7456	76.9286
'Speckle'	1.1803e+03	17.4110	13.6329	0.2324	44.8168	58.4251	73.2711

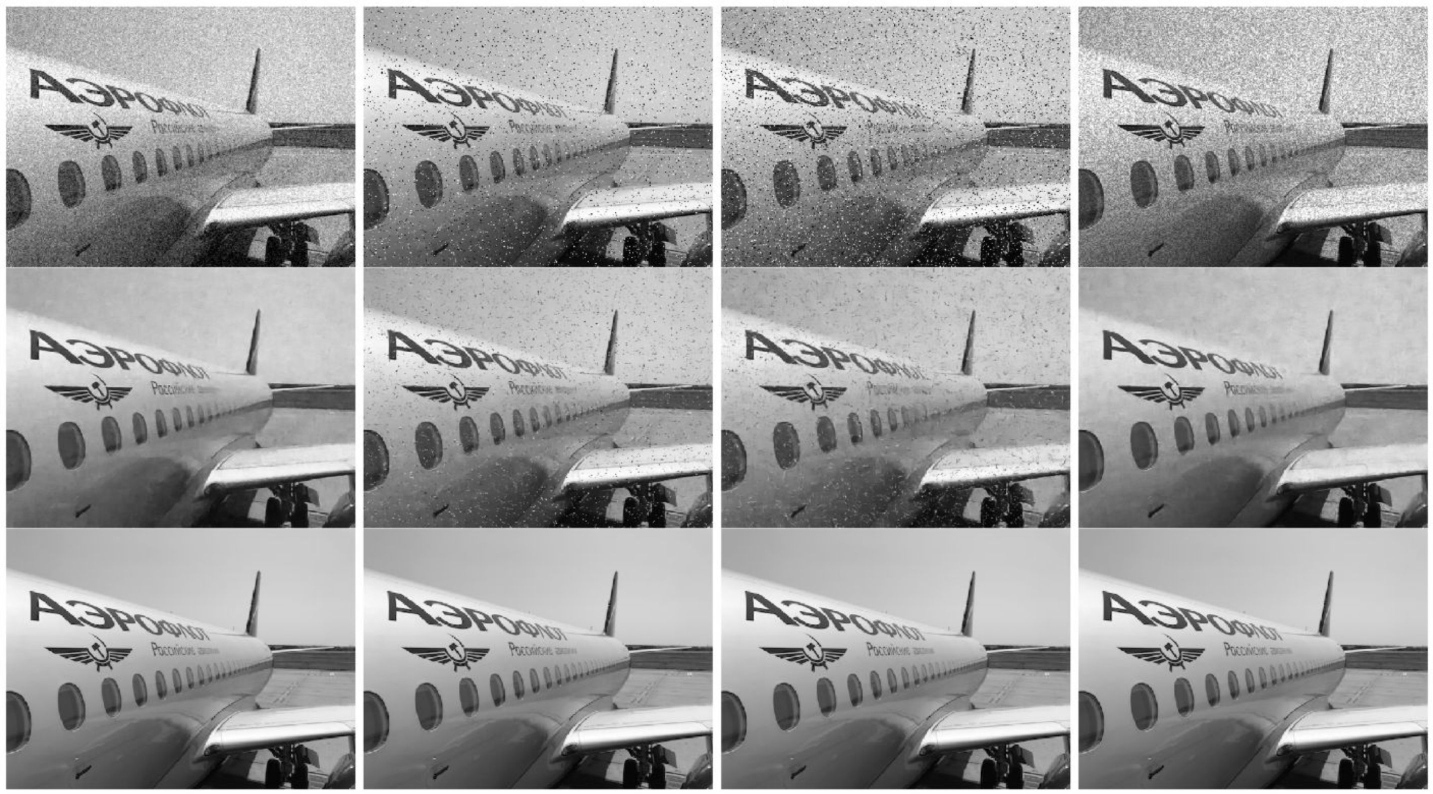
Step 5

Write code to apply the DnCNN-based denoising technique of "Part 2" to at least 2 (two) different test images of your choice, using both Gaussian and salt-and-pepper noise, and compare their results using at least 2 (two) image quality metrics.
Summarize your results in a table (see Guidelines for additional information).

In this step we denoise the noisy images created in step 3:

1. Gaussian white noise with the default values (zero mean and 0.01 variance).
2. Salt and pepper noise with density 0.05.
3. Salt and pepper noise with density 0.1.
4. Speckle noise with the default values (zero mean and 0.05 variance).

The following figure shows the noisy image (top row), denoised image (middle row), and the original image (repeated in the bottom row).



The following table, extracted from the live script, shows the quality metrics for each picture. The first row of the resulting table is the original image. The following rows show one of the noisy images, followed by its denoised version (prefixed with "(D)").

We can make a few observations, based on the metrics shown in the following table:

1. From the MSE values: Gaussian and speckle noise recovered significantly better than salt and pepper.
2. This is also visible in the SSIM values, where salt and pepper are in the mid 0.5 range, while Gaussian and speckle are over 0.8.
3. Interestingly, the subjective PIQE index indicates that the denoised Gaussian and speckle images are of better quality than the original image. Something to investigate later, if time permits.

The observation above can be confirmed visually in the image montage. The Gaussian image is the best denoised version ("best" = subjective user perception). All other images have readily noticed artifacts, especially the salt and pepper ones.

It seems that the neural network is better at recovering "natural" noise than structured noise.

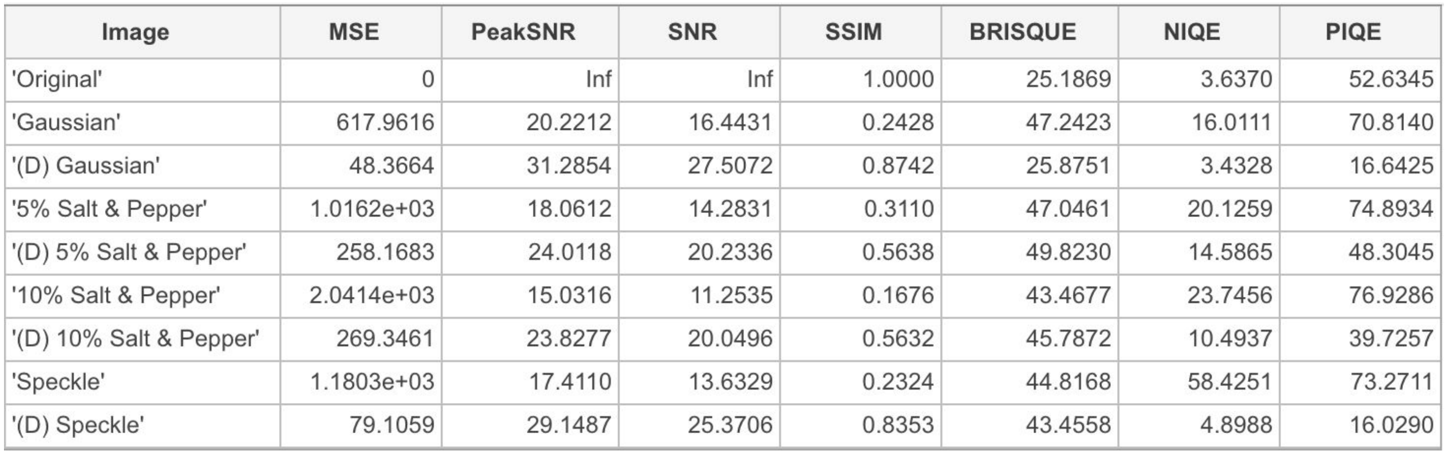


Image	MSE	PeakSNR	SNR	SSIM	BRISQUE	NIQE	PIQE
'Original'	0	Inf	Inf	1.0000	25.1869	3.6370	52.6345
'Gaussian'	617.9616	20.2212	16.4431	0.2428	47.2423	16.0111	70.8140
'(D) Gaussian'	48.3664	31.2854	27.5072	0.8742	25.8751	3.4328	16.6425
'5% Salt & Pepper'	1.0162e+03	18.0612	14.2831	0.3110	47.0461	20.1259	74.8934
'(D) 5% Salt & Pepper'	258.1683	24.0118	20.2336	0.5638	49.8230	14.5865	48.3045
'10% Salt & Pepper'	2.0414e+03	15.0316	11.2535	0.1676	43.4677	23.7456	76.9286
'(D) 10% Salt & Pepper'	269.3461	23.8277	20.0496	0.5632	45.7872	10.4937	39.7257
'Speckle'	1.1803e+03	17.4110	13.6329	0.2324	44.8168	58.4251	73.2711
'(D) Speckle'	79.1059	29.1487	25.3706	0.8353	43.4558	4.8988	16.0290

Step 7

Explore the details of the denoising convolutional neural network (DnCNN) using analyzeNetwork().

Please refer to the live script.

Step 8

***QUESTION 1**: How did the deep-learning-based noise reduction technique perform according to the selected image quality metrics? Did you notice anything unusual?1*

It performed well on noise that has a natural distribution (Gaussian and speckle) and not so well on artificial noise (salt and pepper). This is visible in the MSE for Gaussian and speckle, which are much smaller than the salt and pepper ones.

QUESTION 2*: Did you notice any significant difference between the quality of the denoised images for Gaussian versus salt-and-pepper (a.k.a. speckle) noise? If so, what is your explanation for the differences in quality?*

The denoised Gaussian image is of much better quality.

A possible explanation is that the neural network has been trained with images that have naturally occurring noise. The reference cited by MathWorks (Zhang et al.) mentions only Gaussian noise in their work (e.g. "our DnCNN model is able to handle Gaussian denoising with unknown noise level") and no salt and pepper noise. This makes me believe that the neural network in MathWorks has been specifically trained to handle Gaussian noise as well.

QUESTION 3*: Based on your observations so far, which recommendations would you give to someone who is new to this topic and wants to use deep-learning-based solutions to remove/reduce noise from images for a particular application? Hint: Simpler is often better.*

Attempt the neural network denoiser first in a few sample images. Switch to a more complex solution only if the neural network solution is not satisfactory.

Step 14

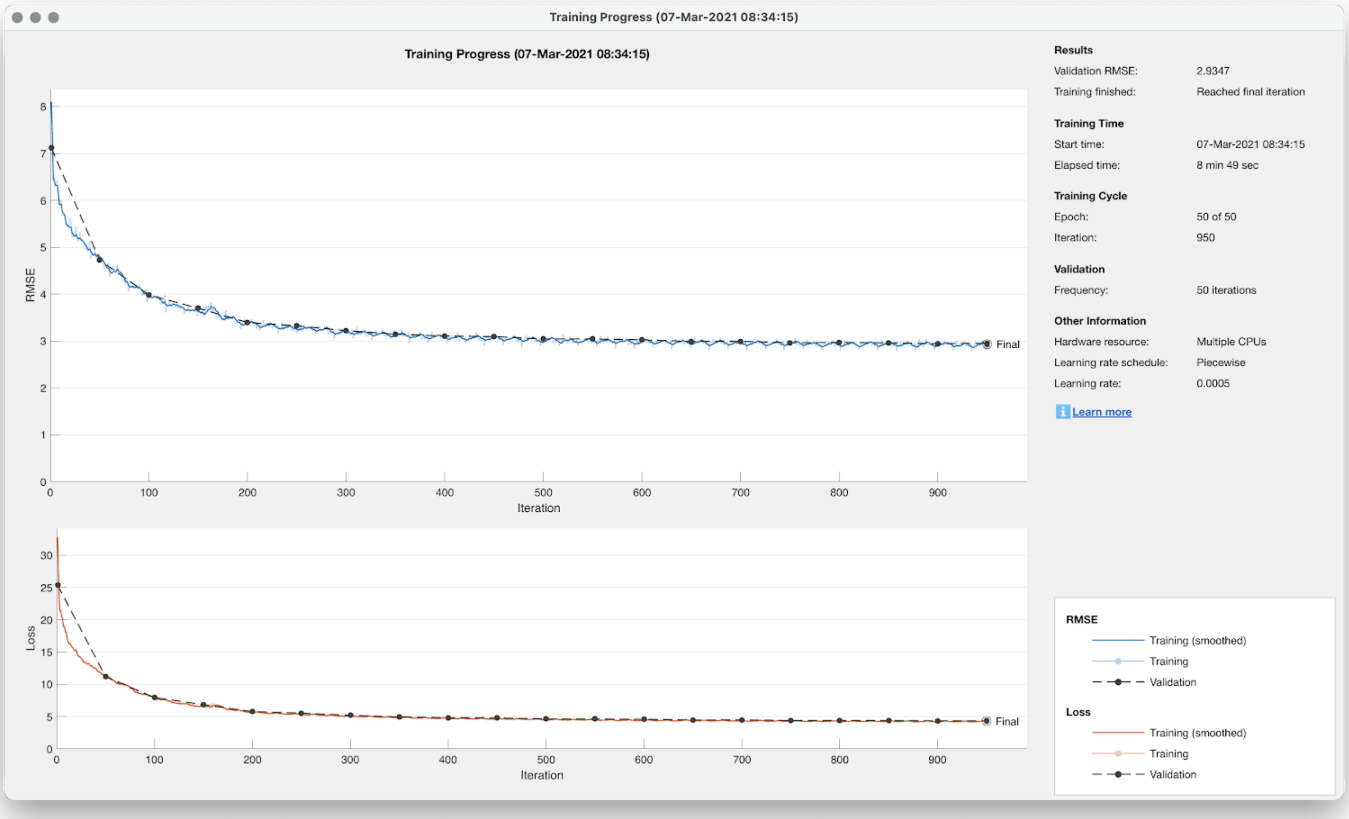
QUESTION 4*: Was the choice of hyperparameters (learning rate, number of epochs, etc.) appropriate?*

The hyperparameters look correct at a first glance. However, because it is not yet overfitting (see next question), it looks like we can either increase the learning rate or train for longer to improve it a bit more.

To test that, we trained the network with a higher initial training rate and a learning scheduler that drops it by half every ten epochs.

```
options = trainingOptions('adam', ...
    'MaxEpochs',50, ...
    'MiniBatchSize',imds.ReadSize, ...
    'ValidationData',dsVal, ...
    'Plots','training-progress', ...
    'ExecutionEnvironment', 'parallel', ...
    'LearnRateSchedule','piecewise', ...
    'InitialLearnRate',0.008, ...
    'LearnRateDropFactor',0.5, ...
    'LearnRateDropPeriod',10, ...
    'Verbose',false);
```

With these hyperparameters, we can get the validation score to 2.935 (figure below), compared with the 3.1 to 3.2 score of the original hyperparameters. Thus, while not the best possible, it is also not too bad. To improve on that we would have to change more important hyperparameters, such as the network architecture (experiment with the number and size of the layers).



Results

Validation RMSE:	2.9347
Training finished:	Reached final iteration

Training Time

Start time:	07-Mar-2021 08:34:15
Elapsed time:	8 min 49 sec

Training Cycle

Epoch:	50 of 50
Iteration:	950

Validation

Frequency:	50 iterations

Other Information

Hardware resource:	Multiple CPUs
Learning rate schedule:	Piecewise
Learning rate:	0.0005

QUESTION 5*: Is the network overfitting? Explain.*

No, because the validation error is not increasing during training.

Step 15

Write code to compute additional figures of merit for the same selected image pair.

Please refer to the live script.

Step 16

QUESTION 6*: Which general principle of autoencoders explains why they work so well, in general, for denoising?*

They are specifically trained on the images they need to denoise. They learn the most important aspects of the representation of the clean images when trained against corrupted (noisy) images. Once they learn the representations of the clean vs. noisy images, they can reconstruct (not to the exact detail, but close enough) the clean images.

Appendix

Experiments to speed up trainNetwork

This section documents experiments with MATLAB's Parallel Computing Toolbox to speed up the training process of the neural network in step 4:

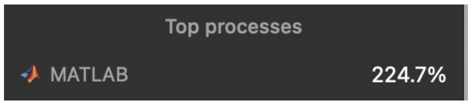
```
autoEncoder = trainNetwork(dsTrain,layers,options);
```

Without parallel execution

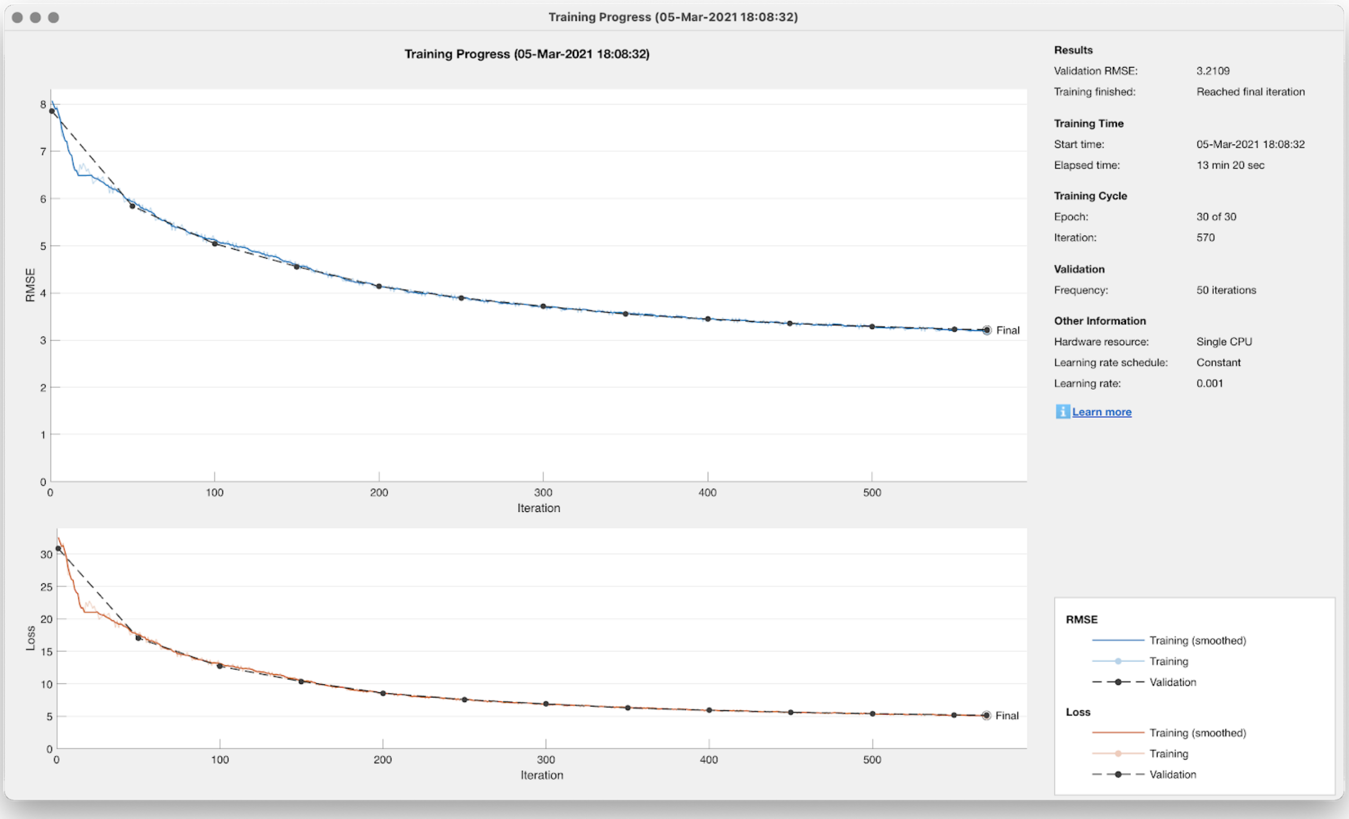
This is the original code.

```
options = trainingOptions('adam', ...
   'MaxEpochs',30, ...
   'MiniBatchSize',imds.ReadSize, ...
   'ValidationData',dsVal, ...
   'Shuffle','never', ...
   'Plots','training-progress', ...
   'Verbose',false);
```

MATLAB's CPU usage indicates that it is doing some parallel work (using more than 100% of CPU, i.e. more than one core).


Top processes
MATLAB 224.7%

The training process takes 13 minutes and 20 seconds, resulting in a validation RMSE of 3.1209.



Training Progress (05-Mar-2021 18:08:32)

Results	
Validation RMSE:	3.2109
Training finished:	Reached final iteration
Training Time	
Start time:	05-Mar-2021 18:08:32
Elapsed time:	13 min 20 sec
Training Cycle	
Epoch:	30 of 30
Iteration:	570
Validation	
Frequency:	50 iterations
Other Information	
Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.001

RMSE
- Training (smoothed)
- Training
- Validation

Loss
- Training (smoothed)
- Training
- Validation

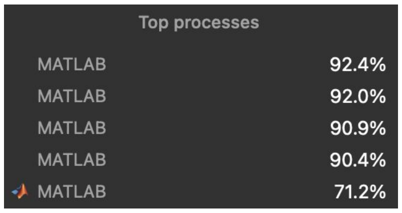
With parallel execution

Configuration: Parallel Computing Toolbox with 4 workers.

Must also remove "`'Shuffle', 'never'`" to enable parallel execution.

```
options = trainingOptions('adam', ...
    'MaxEpochs',30, ...
    'MiniBatchSize',imds.ReadSize, ...
    'ValidationData',dsVal, ...
    'Plots','training-progress', ...
    'ExecutionEnvironment', 'parallel', ...
    'Verbose',false);
```

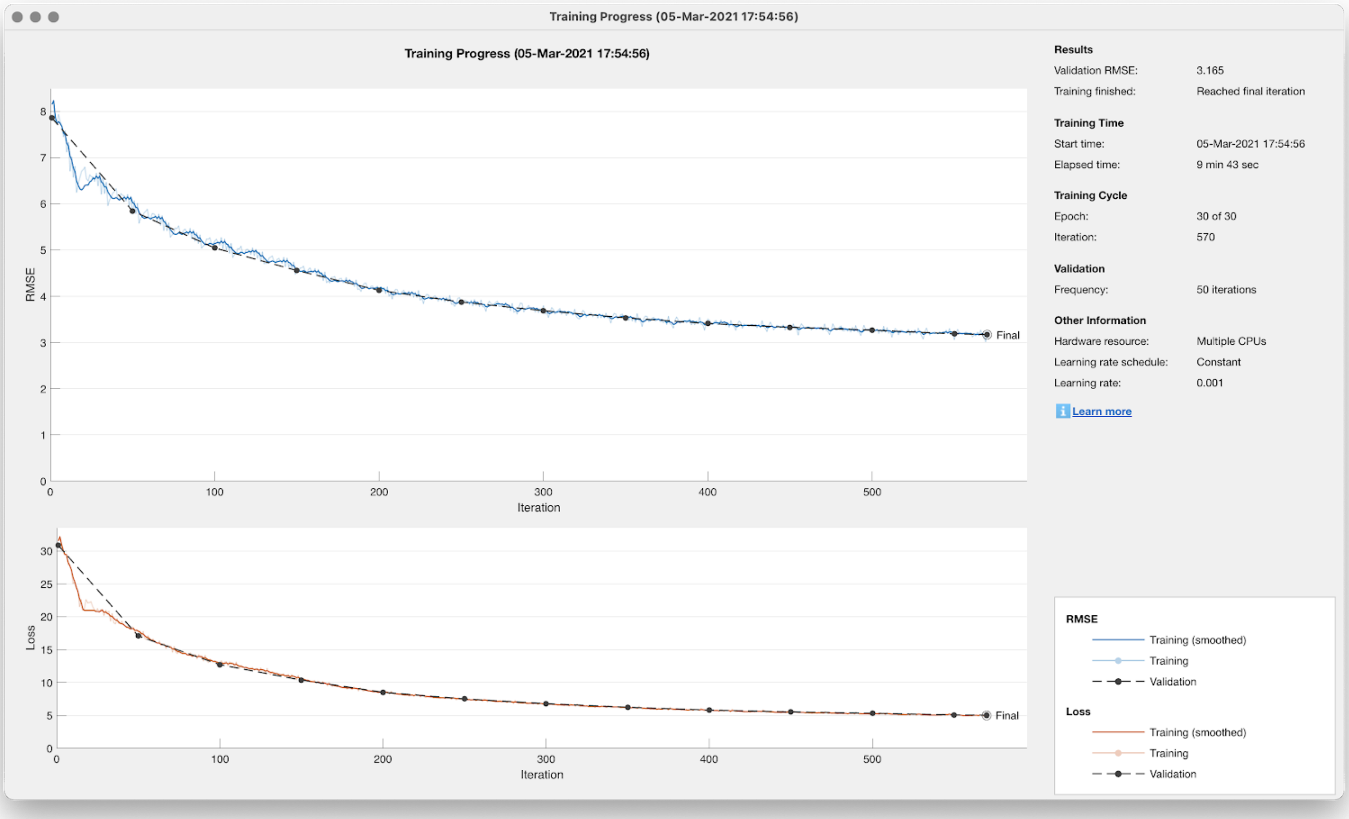
MATLAB's CPU usage now shows four processes, in addition to the main process. Each process is using almost an entire CPU. This is expected when the work can be parallelized.



Top processes

MATLAB	92.4%
MATLAB	92.0%
MATLAB	90.9%
MATLAB	90.4%
MATLAB	71.2%

The training process now takes 9 minutes and 43 seconds, resulting in a validation RMSE of 3.165.


Training Progress (05-Mar-2021 17:54:56)

Training Progress (05-Mar-2021 17:54:56)

Results

Validation RMSE:	3.165
Training finished:	Reached final iteration

Training Time

Start time:	05-Mar-2021 17:54:56
Elapsed time:	9 min 43 sec

Training Cycle

Epoch:	30 of 30
Iteration:	570

Validation

Frequency:	50 iterations

Other Information

Hardware resource:	Multiple CPUs
Learning rate schedule:	Constant
Learning rate:	0.001

Conclusion

Although the reduction of training time was not linearly related to the number of worker threads, it was still significant.

Considering that the code changes to use parallel execution are small, we should always attempt to use the Parallel Computing Toolbox.