

Introduction to Transfer Learning

Classifying Photographs vs. Drawings



Overview

Problem

Here we show how to develop a deep learning algorithm to classify images, given any small labeled dataset, using transfer learning. To demonstrate the procedure, we train a classifier to identify whether an input image is photograph of a real object or whether it is a painting/drawing.

Transfer Learning

Humans are able to learn very quickly to identify something after being shown only a few examples because we can utilize our existing knowledge about the world to solve new problems. This ability to transfer knowledge learned for one task in one domain to another task in a different domain is called transfer learning.

Typically machine learning algorithms do not generalize well to situations very different from the data that is used for training. Furthermore, they typically require large amounts of labeled data to achieve good performance when trained from the ground up.

However, convolutional neural nets trained on images learn to detect simple features such as edges, corners, color blobs, etc., in their early layers. These features are useful for all kinds of image processing tasks. Therefore, we can reuse these nets already trained on a large dataset of images such as ImageNet, and fine-tune them to build a classifier for a new problem using only a few labeled elements.

Data

A dataset of photographs and drawings can be easily obtained by searching on the web (e.g. Google Image Search) or using the function `WeblImageSearch`. Here, we have already scraped some photographs and drawings which are shown below.

Photos

Getting 150 images of photographs by partitioning the image below and labeling each of them “Photo”.

```
photos = Flatten@ImagePartition[
```

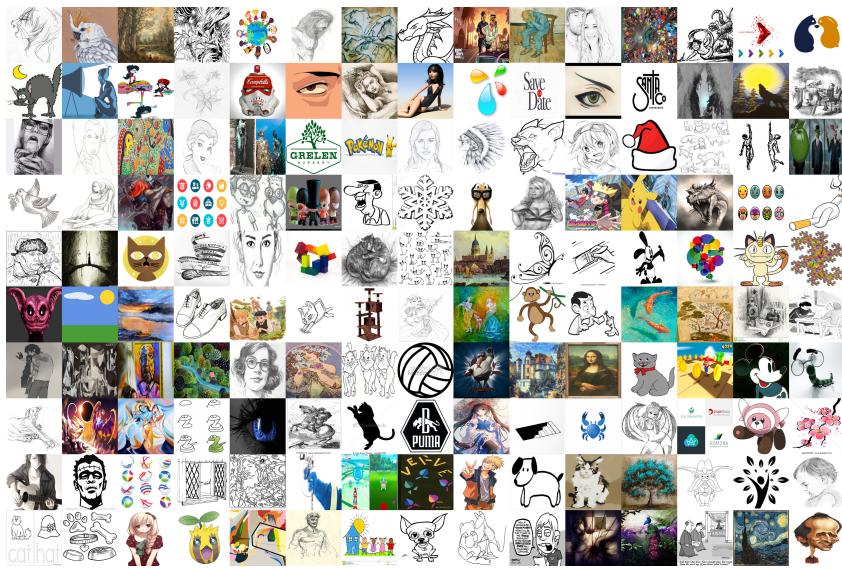


```
, {224, 224}] → "Photo" // Thread;
```

Drawings

Getting 150 images of drawings from the image below and labeling them “Drawing”

```
drawings = Flatten@ImagePartition[
```



```
, {224, 224}] → "Drawing" // Thread;
```

Splitting into Training and Testing Data

Randomly shuffling and then splitting the dataset into 200 images for training and 100 images for testing.

```
{trainingData, testData} = TakeDrop[RandomSample@Join[drawings, photos], 200];
```

Having a separate test set is essential since it ensures that the algorithm is not simply memorizing the image shown to it without generalizing to unseen images.

Designing Net

Pre-trained Net Model

We download the convolutional neural net used in *ImageIdentify*(<https://www.imageidentify.com>), which is already trained on millions of images to classify objects belonging to thousands of different classes.

```
preTrainedNet = NetModel["Wolfram ImageIdentify Net for WL 11.1"]
```

NetChain [Input	image
	conv_1	3-tensor (size: 3 × 224 × 224)
	bn_1	3-tensor (size: 64 × 112 × 112)
	relu_1	3-tensor (size: 64 × 112 × 112)
	pool_1	3-tensor (size: 64 × 55 × 55)
	conv_2_red	3-tensor (size: 64 × 55 × 55)
	bn_2_red	3-tensor (size: 64 × 55 × 55)
	relu_2_red	3-tensor (size: 64 × 55 × 55)
	conv_2	3-tensor (size: 192 × 55 × 55)
	bn_2	3-tensor (size: 192 × 55 × 55)
	relu_2	3-tensor (size: 192 × 55 × 55)
	pool_2	3-tensor (size: 192 × 27 × 27)
	3a	3-tensor (size: 256 × 27 × 27)
	3b	3-tensor (size: 320 × 27 × 27)
	3c	3-tensor (size: 576 × 14 × 14)
	4a	3-tensor (size: 576 × 14 × 14)
	4b	3-tensor (size: 576 × 14 × 14)
	4c	3-tensor (size: 608 × 14 × 14)
	4d	3-tensor (size: 608 × 14 × 14)
	4e	3-tensor (size: 1056 × 7 × 7)
	5a	3-tensor (size: 1024 × 7 × 7)
	5b	3-tensor (size: 1024 × 7 × 7)
	global_pool	PoolingLayer
	linear	LinearLayer
	softmax	SoftmaxLayer
	Output	Output

This is a convolutional neural network architecture called *Inception*, which is very fast to train and evaluate.

Truncating Net

The number of neurons in the final linear layer should be equal to the number of classes of objects.

Therefore, we remove the layers at the end which were designed to perform classification on over 4000 images. This truncated net is known as a feature-extractor since it maps any input image into a vector of real numbers encoding useful properties about the image.

```
truncatedNet = Take[preTrainedNet, {1, -3}]
```

NetChain [Input	image
	conv_1	3-tensor (size: 3 × 224 × 224)
	bn_1	3-tensor (size: 64 × 112 × 112)
	relu_1	3-tensor (size: 64 × 112 × 112)
	pool_1	3-tensor (size: 64 × 55 × 55)
	conv_2_red	3-tensor (size: 64 × 55 × 55)
	bn_2_red	3-tensor (size: 64 × 55 × 55)
	relu_2_red	3-tensor (size: 64 × 55 × 55)
	conv_2	3-tensor (size: 192 × 55 × 55)
	bn_2	3-tensor (size: 192 × 55 × 55)
	relu_2	3-tensor (size: 192 × 55 × 55)
	pool_2	3-tensor (size: 192 × 27 × 27)
	3a	3-tensor (size: 256 × 27 × 27)
	3b	3-tensor (size: 320 × 27 × 27)
	3c	3-tensor (size: 576 × 14 × 14)
	4a	3-tensor (size: 576 × 14 × 14)
	4b	3-tensor (size: 576 × 14 × 14)
	4c	3-tensor (size: 608 × 14 × 14)
	4d	3-tensor (size: 608 × 14 × 14)
	4e	3-tensor (size: 1056 × 7 × 7)
	5a	3-tensor (size: 1024 × 7 × 7)
	5b	3-tensor (size: 1024 × 7 × 7)
	global_pool	PoolingLayer
		Output

Attaching Classifier

Now we create a new net by attaching 2 neurons and a Softmax layer to create a classifier for two classes. The softmax layer ensures that the output is the probabilities of each class. We attach a NetDecoder to the output to convert these probabilities to the class names.

```
net = NetChain[{truncatedNet, 2, SoftmaxLayer[]},  
 "Output" → NetDecoder[{"Class", {"Photo", "Drawing"}]]
```

NetChain [Input	image
	1 NetChain[<<22>>]	3-tensor (size: 3 × 224 × 224)
	2 LinearLayer	vector (size: 2)
	3 SoftmaxLayer	vector (size: 2)
	Output	class (uninitialized)

This net takes an image as input and outputs a class which is either “Photo” or “Drawing”.

Training

Now we train this new net on our dataset. This procedure is called fine-tuning since the net contains weights already pre-trained on a larger dataset.

Net Train

```
trainedNet = NetTrain[net, trainingData, TargetDevice → "GPU"]
```

NetChain [<table border="0"> <tr> <td style="padding-right: 20px;">Input</td><td>image 3-tensor (size: $3 \times 224 \times 224$)</td></tr> <tr> <td>1</td><td>NetChain[<<22>>]</td></tr> <tr> <td>2</td><td>LinearLayer</td></tr> <tr> <td>3</td><td>SoftmaxLayer</td></tr> <tr> <td></td><td>Output</td></tr> <tr> <td></td><td>vector (size: 2)</td></tr> <tr> <td></td><td>vector (size: 2)</td></tr> <tr> <td></td><td>class</td></tr> </table>	Input	image 3-tensor (size: $3 \times 224 \times 224$)	1	NetChain[<<22>>]	2	LinearLayer	3	SoftmaxLayer		Output		vector (size: 2)		vector (size: 2)		class]
Input	image 3-tensor (size: $3 \times 224 \times 224$)																	
1	NetChain[<<22>>]																	
2	LinearLayer																	
3	SoftmaxLayer																	
	Output																	
	vector (size: 2)																	
	vector (size: 2)																	
	class																	

Using a GPU speeds up the training by over 15x. If a GPU is not available, the TargetDevice option can be removed or set to “CPU”.

Testing

Classifier Measurements

Computing performance of the fine-tuned net model.

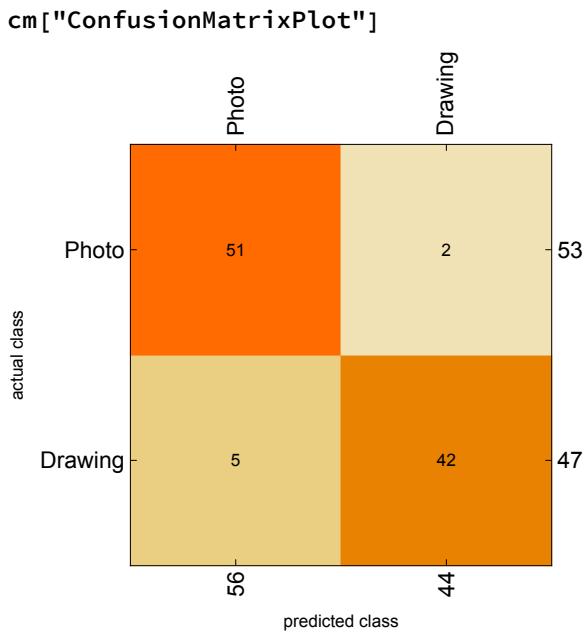
```
cm = ClassifierMeasurements[trainedNet, testData]
```

ClassifierMeasurementsObject [ Classifier: Net Number of test examples: 100 Number of classes: 2 Accuracy: 0.9300 ± 0.026]
--------------------------------	--	---

The accuracy is shown above in the ClassifierMeasurementsObject. This is an object from which you can query all kinds of properties about how the model performs on the test set.

Confusion Matrix

A confusion matrix shows the distribution of predictions on the test set across each class. The diagonal entries show the number of correctly classified elements.



Misclassified Examples

All the examples in the test set which were classified incorrectly are shown below.

```
cm["MisclassifiedExamples"][[;;, 1]] // GraphicsRow
```



Comparisons

Net Train without Transfer Learning

We reset the net model with random initialization to erase the pre-trained knowledge. Then we train the net in the same manner as before for comparison.

```
trainedNet = NetTrain[NetInitialize[net, All], trainingData, TargetDevice -> "GPU"]
```

```
NetChain [ Input image  
          1 NetChain[<<22>>] 3-tensor (size: 3 × 224 × 224)  
          2 LinearLayer vector (size: 2)  
          3 SoftmaxLayer vector (size: 2)  
          Output class ]
```

Computing accuracy on the test set.

```
cm = ClassifierMeasurements[trainedNet, testData]
```

```
ClassifierMeasurementsObject[  Classifier: Net  
Number of test examples: 100  
Number of classes: 2  
Accuracy: 0.730±0.045 ]
```

We can see that without transfer learning the net model performs much worse than before on the same dataset.

Classify with Raw Pixels

Training the built-in Classify functions on a vector of raw pixels without using any feature extractors for images.

```
classifyRaw = Classify[  
  MapAt[Flatten@*ImageData, trainingData, {;; , 1}], PerformanceGoal -> "Quality"]
```

```
ClassifierFunction[  Input type: NumericalSequence  
Classes: Drawing, Photo  
Method: NaiveBayes  
Number of training examples: 200 ]
```

Computing the accuracy with the raw pixels as inputs for each image in the test set.

```
ClassifierMeasurements[classifyRaw, MapAt[Flatten@*ImageData, testData, {;; , 1}]]
```

```
ClassifierMeasurementsObject[  Classifier: NaiveBayes  
Number of test examples: 100  
Number of classes: 2  
Accuracy: 0.8300±0.038 ]
```

Classify with Image Features

Training Classify on the same training set, which by default uses a built-in feature extractor for images.

```
classifyFeatures = Classify[trainingData,  
  FeatureExtractor -> "ImageFeatures", PerformanceGoal -> "Quality"]
```

```
ClassifierFunction[  Input type: Image  
Classes: Drawing, Photo  
Method: RandomForest  
Number of training examples: 200 ]
```

Computing the accuracy on the same test set.

```
ClassifierMeasurements[classifyFeatures, testData]
```

```
ClassifierMeasurementsObject[  Classifier: RandomForest  
Number of test examples: 100  
Number of classes: 2  
Accuracy: 0.8600±0.035 ]
```

We achieve a slightly lower accuracy with the built-in Classify function compared to our fine-tuned net but higher accuracy compared to Classify with raw pixels. This is because Classify internally uses a pre-

trained net model to extract features from the images before training the classifier. The difference is that we have fine-tuned every layer in the net, whereas Classify trains a machine learning algorithm on the outputs of the net, keeping each layer fixed.

Conclusion

We have introduced the concept of transfer learning and demonstrated how it can be used for the simple problem of classifying whether an input image is a photograph or not. We achieved 93% accuracy with a small training dataset containing only 100 images of each class. Furthermore, we show that the transfer learning achieves significantly better results compared to training the net from scratch or using other machine learning methods.

This technique can be applied to many kinds of image classification problems, even when the new dataset is completely different from the original dataset used to pre-train the net models.

Further reading: <https://cs231n.github.io/transfer-learning/>