Distinguishing Lionel Messi from his almost identical lookalike

Motivation:

If we want to build an image classifier for our own specific task. We might think we need a large collection of data. Surprisingly, a small collection of training images are good enough to produce a reasonable accuracy rate (90–100%).

In many fields (bio-medical images, industrial settings) It could be hard or expensive to collect enough image data. However, we don’t necessarily need 1000’s of images to build our own custom classifier, instead, we can use a pre-trained model.

Goal:

Can we easily say which one is Leo Messi and which one is Reza Parastesh (Messi’s lookalike)? I sometimes find it difficult to distinguish them.

Leo Messi

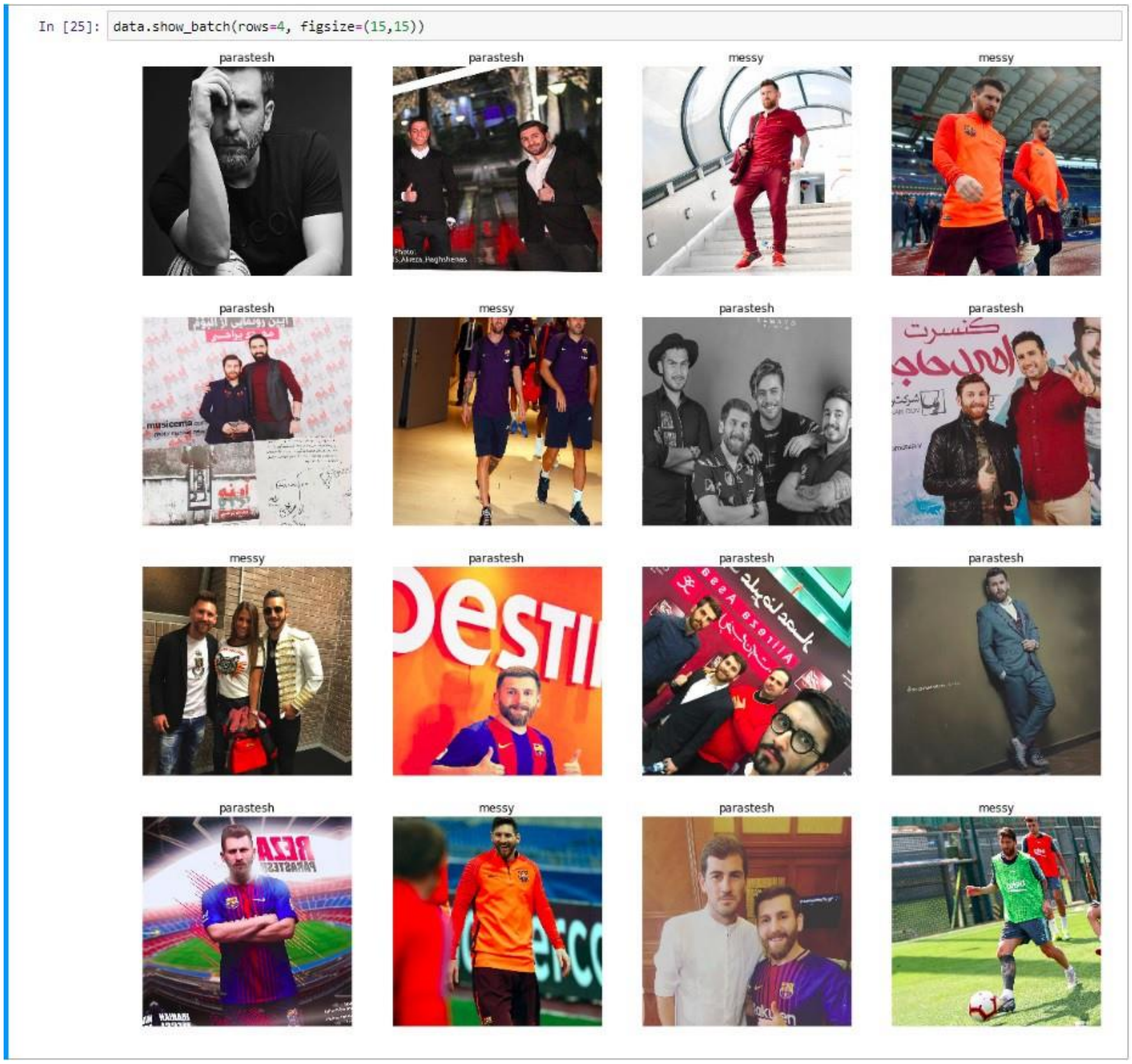
Reza Parastesh

The goal is to train a classifier on small amount of images which can distinguish Messi from it’s lookalike with high accuracy.

Dataset:

I manually scraped Leo Messi’s official Instagram page (https://www.instagram.com/leomessi/) and Reza Parastesh's official Instagram page (https://www.instagram.com/rezaparastesh/) From each of them I extracted 50 photos, using 30 to train and 20 to validate. The dataset is available at this URL (https://www.dropbox.com/s/qwtr64qkcsij855/Messy-Parastesh.zip).

let’s look at the dataset:



```
In [25]: data.show_batch(rows=4, figsize=(15,15))
```

We can see that in most cases it’s very hard to distinguish them.

Transfer Learning:

Instead of training a model from scratch, we use a pre-trained model. A pre-trained model is a model created by someone else to solve a different problem. We modify such model to suit it to our needs. This technique is known as transfer learning. The pre-trained model that we will be using is the resnet34. This model has been trained on 1.2 million images and 1000 classes from the ImageNet database. The resnet34 is a version of the resnet models that won the 2015 ImageNet competition. It is a convolutional neural network (CNN), a type of neural network used extensively in computer vision.

Learning Rate in Transfer Learning:

discriminative fine tuning is a method where we set different learning rates to different layers in the network during training. This contrasts with how people normally configure the learning rate, which is to use the same rate throughout the network during training.

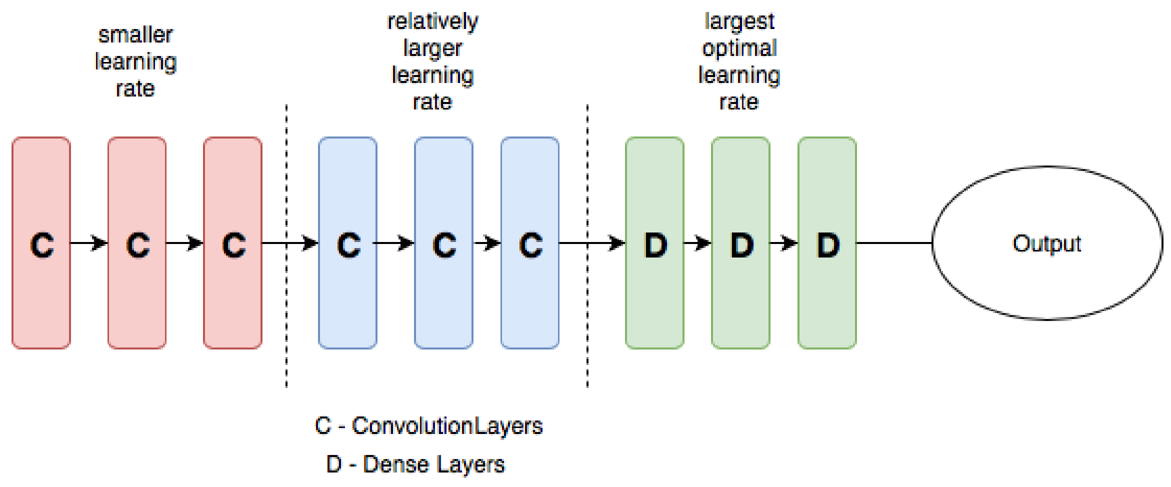
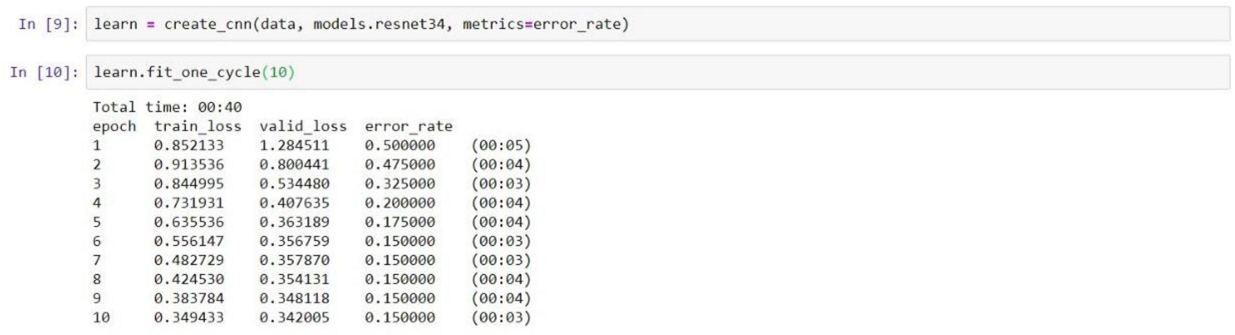


Image from: [https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10]

Our Model

First, we load the pretrained resnet34 then remove the last fully connected layer and add our own fully connected layers instead. Now we freeze the convolution layers, and train the model without using discriminative fine tuning.

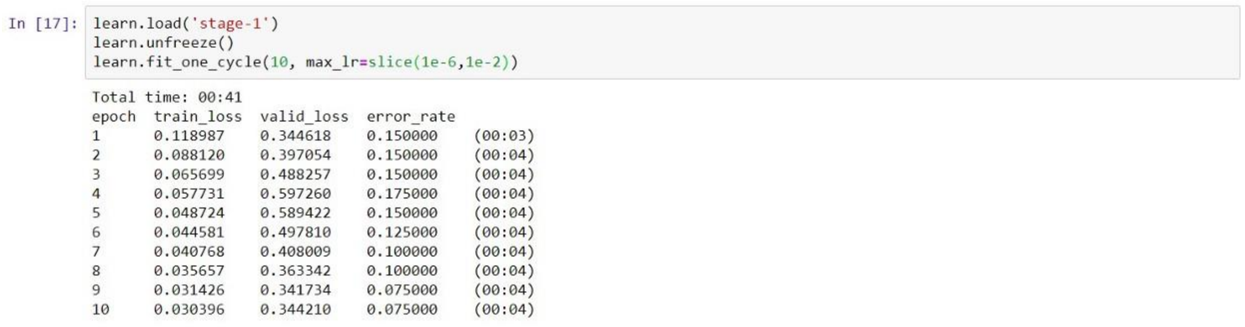


```
In [9]:  learn = create_cnn(data, models.resnet34, metrics=error_rate)

In [10]: learn.fit_one_cycle(10)

         Total time: 00:40
         epoch  train_loss  valid_loss  error_rate
         1      0.852133    1.284511    0.500000    (00:05)
         2      0.913536    0.800441    0.475000    (00:04)
         3      0.844995    0.534480    0.325000    (00:03)
         4      0.731931    0.407635    0.200000    (00:04)
         5      0.635536    0.363189    0.175000    (00:04)
         6      0.556147    0.356759    0.150000    (00:03)
         7      0.482729    0.357870    0.150000    (00:03)
         8      0.424530    0.354131    0.150000    (00:04)
         9      0.383784    0.348118    0.150000    (00:04)
         10     0.349433    0.342005    0.150000    (00:03)
```

we trained a resnet34 model which has 85% accuracy. Now lets unfreeze pre-trained layers and try to train the whole model again, using different learning rates for different layers:



```
In [17]: learn.load('stage-1')
         learn.unfreeze()
         learn.fit_one_cycle(10, max_lr=slice(1e-6,1e-2))

         Total time: 00:41
         epoch  train_loss  valid_loss  error_rate
         1      0.118987    0.344618    0.150000    (00:03)
         2      0.088120    0.397054    0.150000    (00:04)
         3      0.065699    0.488257    0.150000    (00:04)
         4      0.057731    0.597260    0.175000    (00:04)
         5      0.048724    0.589422    0.150000    (00:04)
         6      0.044581    0.497810    0.125000    (00:04)
         7      0.040768    0.408009    0.100000    (00:04)
         8      0.035657    0.363342    0.100000    (00:04)
         9      0.031426    0.341734    0.075000    (00:04)
         10     0.030396    0.344210    0.075000    (00:04)
```

we reached 92.5 % accuracy. We had 40 images for validation, this accuracy means our model has failed to correctly classify only 3 images.

Software:

“The fastai library simplifies training fast and accurate neural nets using modern best practices. It's based on research in to deep learning best practices undertaken at fast.ai, including "out of the box" support for vision, text, tabular, and collab (collaborative filtering) models.” From fastai official documentation [https://docs.fast.ai/]

Hardware:

We need access to a computer with an NVIDIA GPU, so, we used Google Cloud Platform (GCP) to use PyTorch 1.0.0 and fastai 1.0.2. The base platform we used is called n1-highmem-8, and costs $0.12 per hour. Attaching a P4 GPU costs $0.26 per hour so both together amount to $0.38 per hour.