What is Transfer learning?

Transfer learning is a method of utilizing a pre-built/pre-trained Neural Network (CNN in the case) to train on a slightly related data. If the data is similar, easier to utilize the knowledge.

This pre-trained model has weights that are more stable and optimal values, which has figured out most of the fundamental kernels/filters those that are used to form any object that the model has been trained on with maximum accuracy.

This method can be helpful when the data quantity to train on is trivial. Lessen the data, hard to train with traditional ML (from scratch) because this might lead to over fit as the learning is happening on very less data, and unseen data can be misclassified easily.

In such scenarios, utilizing the transfer learning technique can make predictions more reasonable because such utilization improves the Base Models' accuracy.

Another advantage with this method is that the learning can happen very quickly compared to custom user defined models as the kernel and parameters are pre-learnt and could take less epochs to readjust to new data. Hence the improved final performance on unseen data.

List out the transfer learning techniques?

Transfer learning methods can be categorized based on the type of knowledge needed, such as:

Inductive transfer:

- The source and target domains are the same, yet the source and target tasks are different from each other.
- The algorithms try to utilize the inductive biases of the source domain to help improve the target task.

Unsupervised transfer:

- Similar to inductive transfer, with a focus on unsupervised tasks in the target domain.
- The source and target domains are similar, but the tasks are different.
- Labeled data is unavailable in either of the domains.

Transductive transfer:

- There are similarities between the source and target tasks but the corresponding domains are different.
- The source domain has a lot of labeled data while the target domain has none.

Data Description:

Data Exploration: Image Classification

Images are classified into 10 classes, each class has set of images in a separate folder.

The Image data description:

➢ Airplanes	- 828 RGB images with resolution of 400x160 approx.
➢ Background_Google	- 759 RGB images with resolution of 500x300 approx.
➢ Bonsai	- 708 RGB images with resolution of 300x280 approx.
➢ Car_side	- 712 RGB images with resolution of 300x197 approx.
➢ Faces	- 730 RGB images with resolution of 500x340 approx.
➢ Faces_easy	- 731 RGB images with resolution of 280x320 approx.
➢ Grand Piano	- 689 RGB images with resolution of 300x280 approx.
➢ Leopards	- 775 RGB images with resolution of 192x128 approx.
➢ Motorbikes	- 798 RGB images with resolution of 260x140 approx.
➢ Watch	- 823 RGB images with resolution of 300x220 approx.

Train and Validation set details:

- 80% Training Data and 20% Validation Data
- 6046 images belonging to 10 classes in Train Data
- 1506 images belonging to 10 classes in Validation Data

Various Pre-Built Models available:

Keras Library has the following Pre-Built Models [1]:

- Xception,
- InceptionV3,
- InceptionResNetV2
- VGG16 /19,
- ResNet50 /101/152/50V2/101V2/152V2
- ResNeXt50 /101
- MobileNet/V2,
- DenseNet121/169/201
- NASNetMobile, NASNetLarge

Pytorch Library has the following Pre-Built Models [2]:

- Alexnet,
- googlenet,
- resnet18,
- squeezenet1_0,
- densenet161,
- shufflenet_v2_x1_0

GitHub has the following Pre-Built Model [3]:

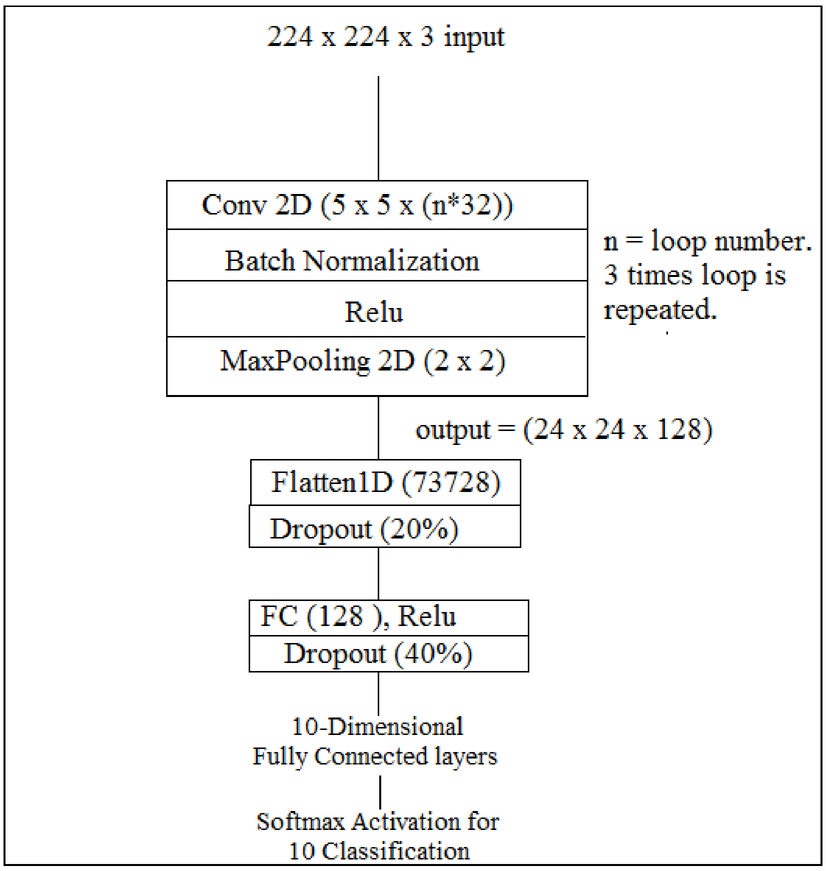
- FaceNet

Base Model, Xception, MobileNetV2, FaceNet are trained on the above data.

Base Model:

MODEL	INPUT	OUTPUT	DEPTH	PARAMETERS	SIZE
USER DEFINED	224 x 224 x 3	128 Dimensional Vectors	12	9,698,122	111 MB

Architecture:



Training Result:

Epoch 215/300: (Batch size = 128)

48/47 - 107s 2s/step - loss: 0.0959- acc: 0.9683 - val_loss: 0.0901- val_acc: 0.9741

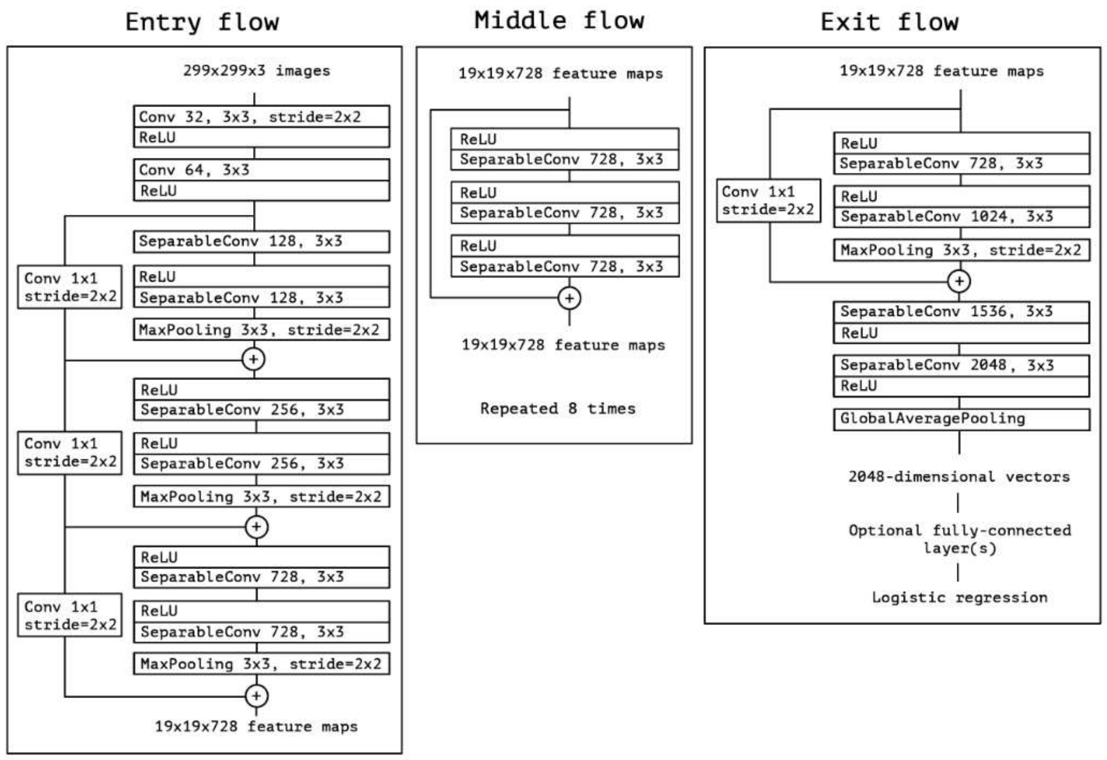
Total Time of Training: 33483 Sec. (9.3 hrs.) for 300 Epochs

Xception Model:

MODEL	INPUT	OUTPUT	DEPTH	PARAMETERS	SIZE
XCEPTION	299 x 299 x 3	2048 Dimensional Vectors	126	22,910,480	88 MB

Architecture [4]:

Figure 5. The Xception architecture: the data first goes through the entry flow, then through the middle flow which is repeated eight times, and finally through the exit flow. Note that all Convolution and SeparableConvolution layers are followed by batch normalization [7] (not included in the diagram). All SeparableConvolution layers use a depth multiplier of 1 (no depth expansion).



Training Result:

Epoch 5/7:

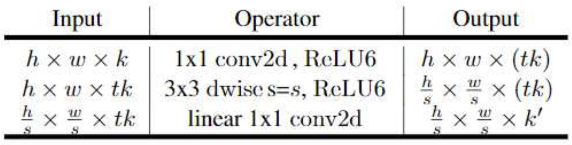
189/188 - 261s 1s/step - loss: 0.0071 - acc: 0.9987 - val_loss: 0.0281 - val_acc: 0.9934

Total Time of Training: 4039 Sec. (67.3 min) for 7 Epochs

MobileNetV2 Model:

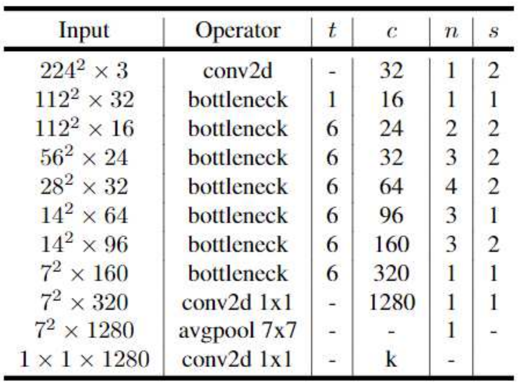
MODEL	INPUT	OUTPUT	DEPTH	PARAMETERS	SIZE
MOBILENETV2	224 x 224 x 3	1028 Dimensional Vectors	88	3,538,984	14 MB

Architecture [5]:



Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Table 1: *Bottleneck residual block* transforming from k to k' channels, with stride s, and expansion factor t.



Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Table 2: MobileNetV2 : Each line describes a sequence of 1 or more identical (modulo stride) layers, repeated n times. All layers in the same sequence have the same number c of output channels. The first layer of each sequence has a stride s and all others use stride 1. All spatial convolutions use 3×3 kernels. The expansion factor t is always applied to the input size as described in Table 1.

Training Result:

Epoch 45/50

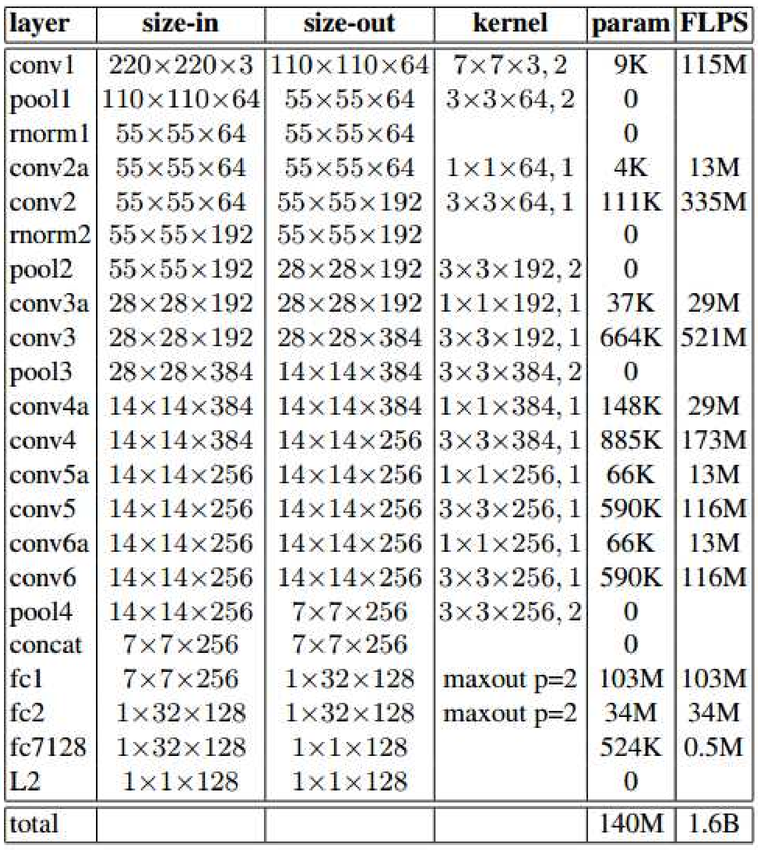
189/188 - 136s 719ms/step - loss: 0.0120 - acc: 0.9965 - val_loss: 0.0481 - val_acc: 0.9920

Total Time of Training: 8688 Sec. (144.8 min) for 50 Epochs

FaceNet Model:

MODEL	INPUT	OUTPUT	DEPTH	PARAMETERS	SIZE
FACENET	160 x 160 x 3 220 x 220 x 3	128 Dimensional Vectors	426	22,808,144	88 MB

Architecture [6]:



layer	size-in	size-out	kernel	param	FLPS
conv1	220×220×3	110×110×64	7×7×3, 2	9K	115M
pool1	110×110×64	55×55×64	3×3×64, 2	0	
rnorm1	55×55×64	55×55×64		0	
conv2a	55×55×64	55×55×64	1×1×64, 1	4K	13M
conv2	55×55×64	55×55×192	3×3×64, 1	111K	335M
rnorm2	55×55×192	55×55×192		0	
pool2	55×55×192	28×28×192	3×3×192, 2	0	
conv3a	28×28×192	28×28×192	1×1×192, 1	37K	29M
conv3	28×28×192	28×28×384	3×3×192, 1	664K	521M
pool3	28×28×384	14×14×384	3×3×384, 2	0	
conv4a	14×14×384	14×14×384	1×1×384, 1	148K	29M
conv4	14×14×384	14×14×256	3×3×384, 1	885K	173M
conv5a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv5	14×14×256	14×14×256	3×3×256, 1	590K	116M
conv6a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv6	14×14×256	14×14×256	3×3×256, 1	590K	116M
pool4	14×14×256	7×7×256	3×3×256, 2	0	
concat	7×7×256	7×7×256		0	
fc1	7×7×256	1×32×128	maxout p=2	103M	103M
fc2	1×32×128	1×32×128	maxout p=2	34M	34M
fc7128	1×32×128	1×1×128		524K	0.5M
L2	1×1×128	1×1×128		0	
total				140M	1.6B

Table 1. **NN1.** This table show the structure of our Zeiler&Fergus [22] based model with 1×1 convolutions inspired by [9]. The input and output sizes are described in $rows \times cols \times \#filters$. The kernel is specified as $rows \times cols, stride$ and the maxout [6] pooling size as $p = 2$.

Training Result:

Epoch 24/50

189/188 - 69s 365ms/step - loss: 0.0143 - acc: 0.9955 - val_loss: 0.0366 - val_acc: 0.9874

Total Time of Training: 7557 Sec. (125.95 min) for 50 Epochs

Models Comparison:

MODEL	PARAMETERS	DEPTH	SIZE (MB)	AVG. TIME / EPOCH (SEC)	EPOCH AT MAXIMUM VALIDATION ACCURACY	MAXIMUM VALIDATION ACCURACY
UserNet* (Basel Model)	9,698,122	12	111	107	215/300	97.4%
Xception	22,910,480	126	88	260.33	5/7	99.34%
Mobile Net V2	3,538,984	88	14	135.5	45/50	99.20%
Face Net	22,808,144	426	88	69.9	24/50	98.74%

*Model has trained on Batch size of 128, while rest has 32.

Advantages of Each of the Models:

Xception:

- Xception has only 126 layers
- Took only few epochs to train on the given data and reached nearly 99% accuracy on validation data.

Mobile Net V2:

- Mobile Net V2 has only 88 layers less than Xception Model.
- Mobile Net V2 has only 3mil parameters, nearly 7times less than Xception and FaceNet Model.
- Made it to nearly 99% accuracy on validation data set similar to Xception Model.
- Consumed only 14MB of memory to store the model.

FaceNet:

- FaceNet took nearly 3-4 times less time to train on the given data for each epoch.
- Took nearly 25 epochs to train on the given data and reached nearly 98% accuracy on validation data.

Disadvantages of Each of the Models:

Xception:

- On an average, Xception has 181,830 parameters/layer
- Took avg. of 260 sec to train each epoch.
- Model took 88MB of model memory to store its architecture.

Mobile Net V2:

- Took avg. of 135 sec to train each epoch.
- Took nearly 45 epochs for training on the given data to reach maximum validation accuracy

FaceNet:

- On an average, FaceNet has 53,541 parameters/layer
- Took nearly 25 epochs for training on the given data to reach maximum validation accuracy
- Model took 88MB of model memory to store its architecture.

Applications:

- Xception is Inception inspired model and has various applications to recognize wide set of images.
- MobileNetV2 is obviously built for portable devices like smart phones, surveillance devices to track objects and recognize them. Applications like the visual data to speech conversion has few of its utilities from this model architecture.
- FaceNet is mainly focused on detecting various faces and clustering the similar faces in applications like Face Clustering.

Reference:

[1] Keras Documentation, “Applications”.
[Online]. Available: https://keras.io/applications/#references_1
[2] Pytorch Documentation, “Torchvision.Models”.
[Online]. Available: https://pytorch.org/docs/stable/torchvision/models.html
[3] Hiroki Taniai, “keras-facenet”, Feb 23, 2018.
[Online]. Available: https://github.com/nyoki-mtl/keras-facenet
[4] François Chollet, “Xception: Deep Learning with Depth wise Separable Convolutions”, 4 Apr 2017.
[Online]. Available: https://arxiv.org/abs/1610.02357
[5] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen,
“MobileNetV2: Inverted Residuals and Linear Bottlenecks”, 21 Mar 2019.
[Online]. Available: https://arxiv.org/abs/1801.04381
[6] Florian Schroff, Dmitry Kalenichenko, James Philbin,
“FaceNet: A Unified Embedding for Face Recognition and Clustering”, 17 Jun 2015.
[Online]. Available: https://arxiv.org/abs/1503.03832