# Session 8
# Discussion Forum 4
# Transfer Learning

## Organization of DISCUSION FORUMs

The discussion forum are exercises/challenges that will test practically the information shared in On-line classes. You must write a program and execute it in a notebook format. At the end of the week you must submit this notebook. The submission deadline date will be Saturday before the on-line class.

The notebook must contain the responses to some questions. To respond them, use a cell in the notebook with the answers.

During the week (Monday to Friday) you can post in the forum sharing your questions, your thoughts and your accomplishments to the group. I will try to answer all your posts trying to guide you to solve the exercise and any other question that you may raise.

## Learning Objectives

The objective of this Exercise is to use Transfer Learning to build a classifier for the Cats and Dogs dataset. The algorithm will be a binary classification of photos that will tell when a photo is of a cat and when a photo is of a dog. Using Transfer Learning you will apply architectures and training that have been used in the Imagenet Competition

## Files for this Practice

```
    08_CNN_Cats_and_Dogs-Transfer_Learning
```

## Some notes about Deep Learning environments

This Practice is to refresh how to create ANN to solve Deep Learning problems using KERAS, and to fine tune your environment to accomodate the Keras-Tensorflow packages and make sure they work together with yoour gymnasium ones.

There are two major tools for neural networks. Tensorflow-Keras and Pytorch. We will focus on KERAS in this course, however both tools are thriving and you can find many application in one or the other.

PyTorch is an open-source deep learning framework developed by Facebook's AI Research lab (FAIR). It has gained immense popularity in the machine learning community due to its dynamic computational graph, ease of use, and flexibility. PyTorch provides a comprehensive platform for building and training deep learning models, making it an excellent choice for both research and production. Keras has been developed by Google.

There are several resources for Deep Learning, I can recommend [Tra19], for Pytorch you can find a good introduction in [SAV20]. If you are interested in the best introduction to Neural Networks / Deep Learning look at this book, from François Chollet, A senior Google engineer and the author of Keras [Cho17], be sure you are reading the 2nd edition which is reviewed and more recent.

## The Flowers Dataset

The TensorFlow Flowers dataset is a well-known dataset provided by TensorFlow Datasets (TFDS), commonly used for multi-class image classification tasks. It consists of images belonging to five categories of flowers, making it ideal for learning and experimenting with computer vision models, especially in TensorFlow and Keras.

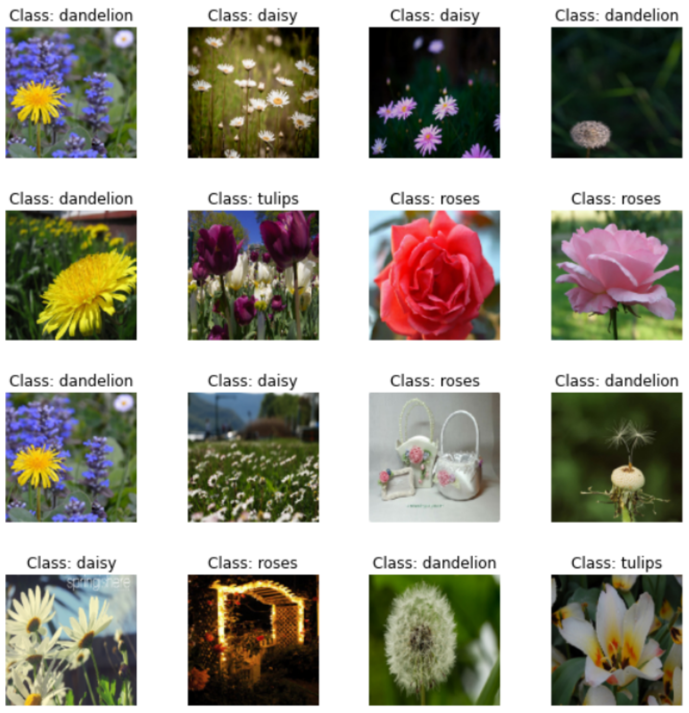


Figure 1: The Flowers dataset for multi-class classification

- Purpose: Image classification (multi-class)
- Number of Classes: (Daisy, Dandelion, Roses, Sunflowers, Tulips)
- Total Images: 3,670 Color images (RGB) Varies (resizing typically required during preprocessing)
- License: Creative Commons Attribution 4.0 International (CC BY 4.0)

## Background on the CATS and DOGS Dataset

The Cats and Dogs Dataset is a popular dataset commonly used in machine learning and computer vision tasks, particularly for image classification problems. It was featured in a Kaggle competition and serves as an excellent resource for beginners and experts to explore binary image classification and deep learning techniques.

originates from Microsoft Research and is part of their larger Microsoft Research Data Science Machine Learning Datasets. It was initially used for benchmarking computer vision tasks and is publicly available on platforms like Kaggle for educational and research purposes.

The dataset is derived from the Asirra (Animal Species Image Recognition for Restricting Access) project by Microsoft Research. This project aimed to explore CAPTCHA systems that use image classification to distinguish between cats and dogs.

- **Number of Samples:** The dataset contains over 25,000 training iumages
- **Image Dimensions:** Images are provided in standard formats, such as .jpg or .png. The resolution of the images varies, with most being medium to high resolution, but resizing is often required for model training. They have different sizes and resolutions and labeled as cat or dog
- **Classes:** Two classes cats and dogs, useful for binary classifications

## Example Images

Figure 2 shows examples of cats and dogs from the dataset.

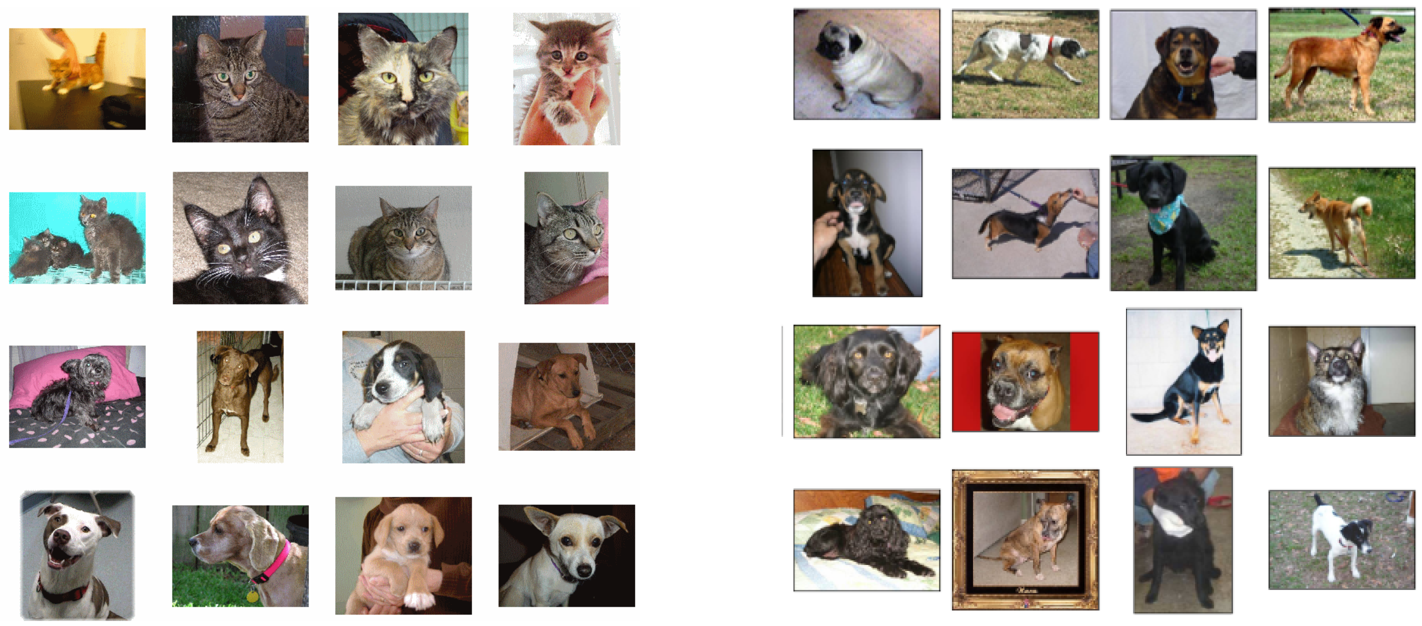


Figure 2: Images from the Cats and Dogs Dataset

## Background on Keras

Keras is a high-level neural network library that was first developed by François Chollet in 2015. It was initially designed to serve as a user-friendly interface for building deep learning models, sitting on top of lower-level machine learning frameworks like TensorFlow, Theano, or Microsoft Cognitive Toolkit (CNTK). Over time, it became tightly integrated into TensorFlow and is now its official high-level API.

Keras is known for its simplicity and modularity, making it ideal for beginners and experts alike. It supports building neural networks with layers, offering flexibility to design both simple and complex architectures. Key features include support for convolutional neural networks (CNNs), recurrent neural networks (RNNs), and hybrid models.

Keras provides built-in tools for data preprocessing, visualization, and evaluation. It also supports distributed training and is compatible with CPUs, GPUs, and TPUs. With extensive community support and a wealth of pre-trained models, Keras is a go-to library for deep learning tasks in academia and industry.

## 1 What is Transfer Learning

Transfer learning in deep learning is a technique where a model trained on one task is reused (partially or fully) for a different but related task. Instead of training a model from scratch, transfer learning leverages the knowledge learned by a pretrained model, often trained on a large dataset, and applies it to a new task with less data or computational resources.

Usually the pretrained models are trained professionally with large investments in data, computing and experimentation. Using a pretrained model is a secure way to reuse knowledge created by Companies and universities.

As the models become larger, pretrained models allow to reuse very large investments, for example the open-sourced Llama model allows us to use an investment made by the company META that will be impossible for small companies or universities to perform.

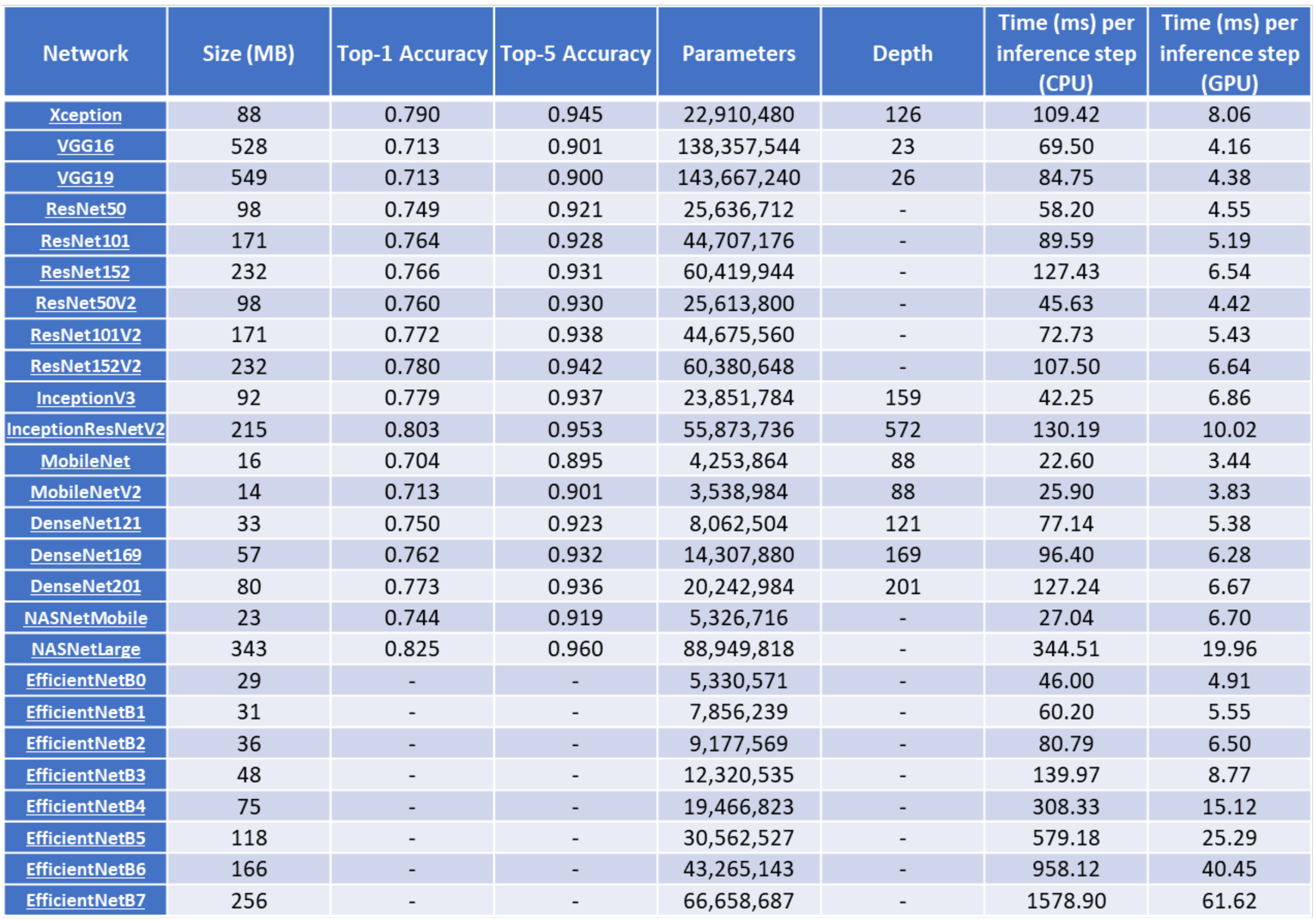
For images there are many pretrained models available, particularly in Keras you can find a whole list of convolutional based networks already trained in the Imagenet dataset (14 millions of images with more than 2000 labels). See 3

## 2 Developing a Deep Learning Classifier using a Transfer Learning

The basic notion of Transfer Lerning is using the original network trained (weights and architecture) and train only a few of the layers with the new data. In this way, if we have a pretrained network that has been trained in images and has very good capabilities, you can use it to classify images that are not in the original dataset (for instance like the cats and dogs dataset or the XRay images).

The technique to implement Transfer Learning consist of the implementation of the following steps.

1. Instantiate the standard architecture
2. Add new layers



| Network | Size (MB) | Top-1 Accuracy | Top-5 Accuracy | Parameters | Depth | Time (ms) per inference step (CPU) | Time (ms) per inference step (GPU) |
|---|---|---|---|---|---|---|---|
| Xception | 88 | 0.790 | 0.945 | 22,910,480 | 126 | 109.42 | 8.06 |
| VGG16 | 528 | 0.713 | 0.901 | 138,357,544 | 23 | 69.50 | 4.16 |
| VGG19 | 549 | 0.713 | 0.900 | 143,667,240 | 26 | 84.75 | 4.38 |
| ResNet50 | 98 | 0.749 | 0.921 | 25,636,712 | - | 58.20 | 4.55 |
| ResNet101 | 171 | 0.764 | 0.928 | 44,707,176 | - | 89.59 | 5.19 |
| ResNet152 | 232 | 0.766 | 0.931 | 60,419,944 | - | 127.43 | 6.54 |
| ResNet50V2 | 98 | 0.760 | 0.930 | 25,613,800 | - | 45.63 | 4.42 |
| ResNet101V2 | 171 | 0.772 | 0.938 | 44,675,560 | - | 72.73 | 5.43 |
| ResNet152V2 | 232 | 0.780 | 0.942 | 60,380,648 | - | 107.50 | 6.64 |
| InceptionV3 | 92 | 0.779 | 0.937 | 23,851,784 | 159 | 42.25 | 6.86 |
| InceptionResNetV2 | 215 | 0.803 | 0.953 | 55,873,736 | 572 | 130.19 | 10.02 |
| MobileNet | 16 | 0.704 | 0.895 | 4,253,864 | 88 | 22.60 | 3.44 |
| MobileNetV2 | 14 | 0.713 | 0.901 | 3,538,984 | 88 | 25.90 | 3.83 |
| DenseNet121 | 33 | 0.750 | 0.923 | 8,062,504 | 121 | 77.14 | 5.38 |
| DenseNet169 | 57 | 0.762 | 0.932 | 14,307,880 | 169 | 96.40 | 6.28 |
| DenseNet201 | 80 | 0.773 | 0.936 | 20,242,984 | 201 | 127.24 | 6.67 |
| NASNetMobile | 23 | 0.744 | 0.919 | 5,326,716 | - | 27.04 | 6.70 |
| NASNetLarge | 343 | 0.825 | 0.960 | 88,949,818 | - | 344.51 | 19.96 |
| EfficientNetB0 | 29 | - | - | 5,330,571 | - | 46.00 | 4.91 |
| EfficientNetB1 | 31 | - | - | 7,856,239 | - | 60.20 | 5.55 |
| EfficientNetB2 | 36 | - | - | 9,177,569 | - | 80.79 | 6.50 |
| EfficientNetB3 | 48 | - | - | 12,320,535 | - | 139.97 | 8.77 |
| EfficientNetB4 | 75 | - | - | 19,466,823 | - | 308.33 | 15.12 |
| EfficientNetB5 | 118 | - | - | 30,562,527 | - | 579.18 | 25.29 |
| EfficientNetB6 | 166 | - | - | 43,265,143 | - | 958.12 | 40.45 |
| EfficientNetB7 | 256 | - | - | 66,658,687 | - | 1578.90 | 61.62 |

Figure 3: Available pretrained networks in Keras

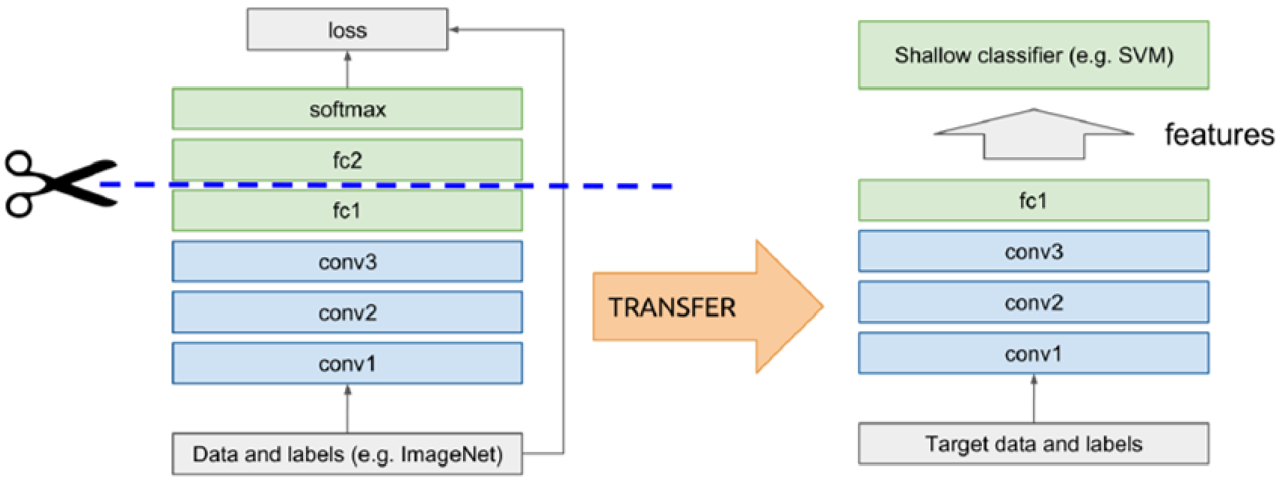


Figure 4: How Transfer Learning works

3. Freeze or unfreeze layers for training
4. Train the network (unfrozen layers)

You can see the details in the example cited at the top of this document. But now we will review the most important steps to instantiate and train a pre-trained architecture.

### Instantiate the pre-trained network

In this first step we instantiate the pre-trained network. See at the parameters that we are loading the weights from the imagenet training. See that we are loading the whole network but removing the top. this means that the 3 fully connected layers will be removed. We see as well that the whole network is frozen (trainable=False)

```
from keras.applications.vgg19 import VGG19
base_model = VGG19(
    weights='imagenet',
    input_shape=(img_height, img_width, 3),
    include_top=False)

base_model.trainable = False
```

### Add new layers

Now we add new layers to the structure, in this case we add a dense and the output layout.

```
inputs = Input(shape=(img_height, img_width, 3))

x = base_model(inputs)

# New layers
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.1)(x)

#Output layer
output = Dense(1, activation='sigmoid')(x)

model = Model(inputs=[inputs], outputs=output)
model.summary()
```

### Compile and then train

The network is compiled (added optimizer, metrics, loss function, and then we can train the new model.

```
model.compile(loss='binary_crossentropy',
            optimizer = Adam(learning_rate=0.001), metrics=['acc'])
```

# 3 EfficientNet and VGG19

I recommend you to use transfer learning with two different architectures (you can try more if you want). The first one is VGG19, a classical CNN architecture that is widely used and well known.

This architecture has the classical structure of CNN blocks and Dense layers at the end. You must remove the dense layers and train the classifier with your new layer.

Note: GlobalAveragePooling2D() performs a flatten on the CNN features, so that you don't need Flatten layer, just use AVg pooling. You can try using one or the other.

## 3.1 VGG19

VGG19 is a deep convolutional neural network (CNN) developed by the Visual Geometry Group (VGG) at Oxford, introduced in the 2014 ImageNet competition. It consists of 19 weight layers, including 16 convolutional layers and 3 fully connected layers.

The architecture follows a simple yet effective design using only 3x3 convolutional filters with a stride of 1, followed by 2x2 max pooling layers to reduce spatial dimensions. ReLU activations are used after each convolutional layer, and the network is divided into five blocks, where deeper layers extract more complex features.

The final layers include fully connected layers and a Softmax classifier for multi-class predictions. VGG19 has over 143 million parameters, making it computationally expensive and memory-intensive. Compared to its predecessor, VGG16, it has additional convolutional layers for improved feature extraction but lacks residual connections, making it prone to vanishing gradients.

While newer architectures like ResNet have surpassed it in efficiency, VGG19 remains widely used for image classification, object detection, and neural style transfer, especially in transfer learning applications with pretrained weights. Despite its limitations, it serves as a benchmark model in deep learning research

To instantiate it use the following

```
from keras.applications.vgg19 import VGG19
base_model = VGG19(
    weights='imagenet',
    input_shape=(img_height, img_width, 3),
    include_top=False)

base_model.trainable = False
```

VGG19 is a good architecture but has been improved with subsequent ones. However is easy to understand. You can try to unfreeze some layers and retrain those layers, but do it when the basic architecture (only last layers unfreezed) works.

To train only the last 5 layers do as follows:

```
# Freeze all layers except the last 10
for layer in base_model.layers[:-10]: # Freeze all but last 10 layers
    layer.trainable = False
```

Use a very small learning rate if you are training a transfer model architecture.

Make it to work, but don't spend too much time optimizing it as you need good GPU

## 3.2 EfficientNet

EfficientNet, first introduced in Tan and Le, 2019 is among the most efficient models (i.e. requiring least FLOPS for inference) that reaches State-of-the-Art accuracy on both imagenet and common image classification transfer learning tasks.

The smallest base model is similar to MnasNet, which reached near-SOTA with a significantly smaller model. By introducing a heuristic way to scale the model, EfficientNet provides a family of models (B0 to B7) that represents a good combination of efficiency and accuracy on a variety of scales. Such a scaling heuristics (compound-scaling, details see Tan and Le, 2019) allows the efficiency-oriented base model (B0) to surpass models at every scale, while avoiding extensive grid-search of hyperparameters.

Efficientnet is possibly the most used Transfer learning architecture nowadays due to its effectiveness. However it has an architecture with only one classifier (Dense) layer at the end. This makes using this architecture a bit different.

I recommend you to use this approach

```
efnModel = EfficientNetB7(weights = 'imagenet', nput_shape = (img_height, img_width, 3),
    include_top = False)

model.add(efnModel)
model.add(layers.GlobalAveragePooling2D())
model.add(layers.Dense(2,activation='softmax'))
```

One final note. When building a binary classifier you can opt for two approaches

- Using one final Dense Layer with one neuron (in this case the class is defined by the closest to 0 or 1 by the resulting number.
- Using a final Dense layer with 2 neurons. The biggest number is the final result (np.argmax(final neuron)).

Remember when using one Neuron the final activation is SIGMOID and when using two neurons the final activation is SOFTMAX.

## 4 What should be done in the Assignment

You can use many pre-trained architectures. My advice is to start with VGG16 or VGG19. VGG19 is an architecture of 19 Convolutional layers that ended first in the IMAGENET Challenge of 2014 [SZ15]. Then, when you have reached the results with VGG19, try EfficientNet.

Use the Notebook as starting point. Try to optimize the network and to run it, either in your environment or in COLAB.

Bear in mind that these networks are quite large. You'll need to use the COLAB GPU to have them trained in an effective time.

Perform 2 experiments

1. Use a Transfer learning approach with the cats and dogs dataset. See if you can beat your numbers in Assignment 3
2. Use Transfer learning to do a Multi-class classifier with the Flowers dataset. How high can you get?

Don't submit 2 notebooks. Submit only one with the flowers dataset but add in the last cell the accuracy obtained in the Cats and dogs with your transfer learning experience

## 5 Questions to Answer

1. What is the precision that you obtained?
2. What are the implications of using Transfer Learning regarding training (is it faster, slower,...)?

## 6 Deliverables and submission

Submit the following files with content

- A Notebook with the Transfer Learning program on the Flowers dataset with the answers to the questions

## References

[SZ15] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV]. URL: https://arxiv.org/abs/1409.1556.

[Cho17] François Chollet. *Deep Learning with Python*. 2nd ed. Manning, Nov. 2017. ISBN: 9781617294433.

[Tra19] Andrew Trask. *Grokking Deep Learning*. 1st. USA: Manning Publications Co., 2019. ISBN: 1617293709.

[SAV20] Eli Stevens, Luca Antiga, and Thomas Viehmann. *Deep Learning with Pytorch*. Manning, 2020. ISBN: 9781633438859.