

# (Concepts of) Machine Learning

## Deep transfer learning with Keras on Kaggle

---

After this session, you should be able to:

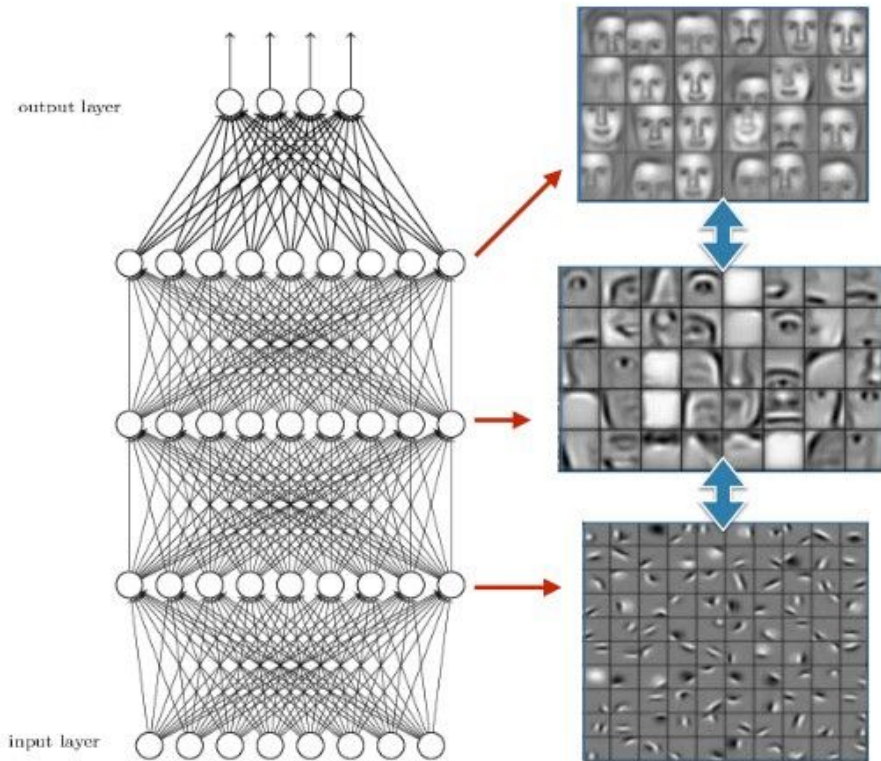
- Use kaggle.com, and get access to all datasets and competitions
  - Use keras to import pretrained models
  - Apply transfer learning techniques to reduce training time of deep nets
- 

### What is transfer learning?

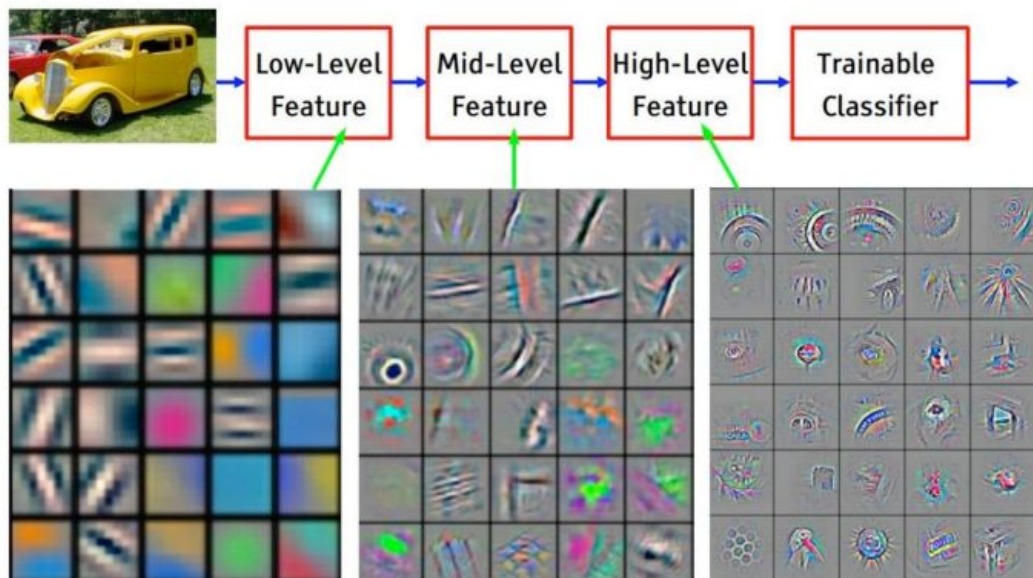
Transfer learning is a popular training technique used in deep learning; where models that have been trained for a task are reused as base/starting point for another model. To train an Image classifier that will achieve near or above human level accuracy on Image classification, we'll need massive amount of data, large compute power, and lots of time on our hands. This I'm sure most of us don't have. Knowing this would be a problem for people with little or no resources, some smart researchers built models, trained on large image datasets like ImageNet, COCO, Open Images, and decided to share their models to the general public for reuse. This means you should never have to train an Image classifier from scratch again, unless you have a very, very large dataset different from the ones above or you want to be an hero or thanos.

### Why does transfer learning work?

Well Transfer learning works for Image classification problems because Neural Networks learn in an increasingly complex way. i.e The deeper you go down the network the more image specific features are learnt. A neural network learns to detect objects in increasing level of complexity. Let's build some intuition to understand this better. In a neural network trying to detect faces, we notice that the network learns to detect edges in the first layer, some basic shapes in the second and complex features as it goes deeper.



So the idea here is that all Images have shapes and edges and we can only identify differences between them when we start extracting higher level features like-say nose in a face or tires in a car. Only then can we say, okay; this is a person, because it has a nose and this is an automobile because it has four tires.



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

The takeaway here is that the earlier layers of a neural network will always detect the same basic shapes and edges that are present in both the picture of a car and a person.

Now, taking this intuition to a problem of differentiating images of dogs from cats, it means we can use models that have been trained on huge dataset containing different types of animals. This works because these models have learnt already the basic shape and structure of animals and therefore all we need to do, is teach it (model) the high level features of our new images.

All I'm trying to say is that we need a network already trained on a large image dataset like ImageNet (contains about 1.4 million labeled images and 1000 different categories including animals and everyday objects). Since this model already knows how classify different animals, then we can use this existing knowledge to quickly train a new classifier to identify our specific classes (cats and dogs).

## **Deep neural network architectures and pretrained models**

Now that we have an understanding/intuition of what Transfer Learning is, let's talk about pretrained networks. There are different variants of pretrained networks each with its own architecture, speed, size, advantages and disadvantages. Keras comes prepackaged with many types of these pretrained models. Some of them are:

VGGNET : Introduced by Simonyan and Zisserman in their 2014 paper, Very Deep Convolutional Networks for Large Scale Image Recognition.

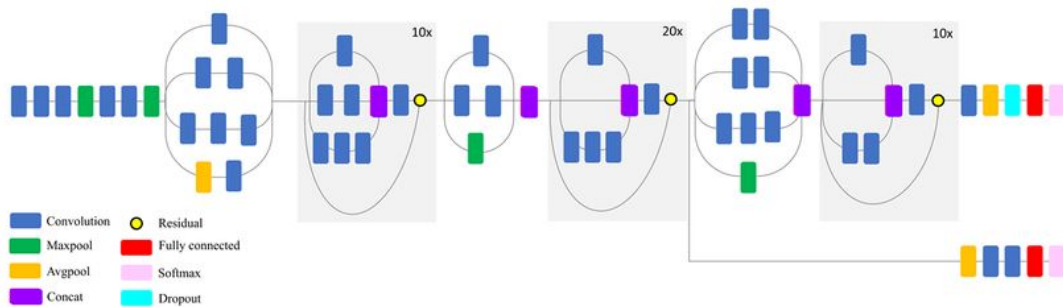
RESNET : First introduced by He et al. in their 2015 paper, Deep Residual Learning for Image Recognition

INCEPTION: The "Inception" micro-architecture was first introduced by Szegedy et al. in their 2014 paper, Going Deeper with Convolutions:

XCEPTION: Xception was proposed by François Chollet , the creator of the Keras library.

and many more. Detailed explanation of some of these architectures can be found [here](#).

We will be using the InceptionResNetV2 in this tutorial, feel free to try other models.



The InceptionResNetV2 is a recent architecture from the INCEPTION family. It works really well and is super fast for many reasons, but for the sake of brevity, we'll leave the details and stick to just using it in this post.

If you're interested in the details of how the INCEPTION model works then go [here](#).

## Kaggle.com

Kaggle is an online community of data scientists and machine learners, owned by Google. Kaggle allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges. Kaggle got its start by offering machine learning competitions and now also offers a public data platform, a cloud-based workbench for data science, and short form AI education.

In kaggle you have the concept of a Kernel which represents the environment where you will execute the code on kaggle's own infrastructure. This means that you will not run that code locally, instead you will run it on their servers. As you can imagine this means that you have to either use data which is made available by kaggle or upload your own. We will be using existing data as uploading new data is beyond the scope of this tutorial.

We will also work with Notebook kernels instead of Script as these are interactive and easier to debug on the fly, without the need of an IDE.

Please head over to <https://www.kaggle.com> and register.

## Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Use Keras if you need a deep learning library that:

Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).

Supports both convolutional networks and recurrent networks, as well as combinations of the two.

Runs seamlessly on CPU and GPU.

Read the documentation at <https://keras.io>.

## Notebooks on Kaggle.com

Login to <https://www.kaggle.com> and head to Kernels > New Kernel, where you should select **Notebook**.

Next you should click on **Settings** and enable the **GPU** and **Internet access** if they are not already enabled. If you get **popups**, please select **Accept** for all cases.

## Cats vs Dogs data

As a toy problem we will try to classify whether images contain either a dog or a cat. This is easy for humans, dogs, and cats. Your computer will find it a bit more difficult. This was an old Kaggle competition, if you want more information about it please visit: <https://www.kaggle.com/c/dogs-vs-cats>

We need to add the data to our workspace (kernel)

Click on **+ Add dataset** (top right corner), select **Competition data** and search for **Dogs vs. Cats Redux: Kernels Edition**. Finally press **ADD** and the data will be added to your kernel.

We can see three files, in the right hand side, under:

**Workspace > input > Dogs vs Cats:**

### **sample submission.csv**

This is a csv (comma separated value) file that is used for making submission after training your model and testing it on the test files given to you. Since this competition is over, we cannot make submissions, so we'll ignore this file.

### **test.zip**

This file contains the Images we're going to test our models on after training, to know if our model has learnt to differentiate dogs from cats.

### **train.zip**

This is the food for our model. It contains the data we're going to use to teach our model what a dog or a cat looks like.

Now to access our training Images, please create a new cell (press the **+ Code** button)


```
In[ ]: # This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

import os
print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.
```



and add type the following lines of code in the newly created cell, like in the image below:

```
In[]: # This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

import os
print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.
```

```
import cv2
import random

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline

tr_dir = '../input/train'
ts_dir = '../input/test'

tr_imgs = [f'{tr_dir}/{i}' for i in os.listdir(tr_dir)]
ts_imgs = [f'{ts_dir}/{i}' for i in os.listdir(ts_dir)]

def process_imgs(imgs, width=150, height=150):
```

```
import cv2
```

```
import random
```

```
import os
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import matplotlib.image as mpimg
```

```
%matplotlib inline
```

```
tr_dir = '../input/train'
```

```
ts_dir = '../input/test'
```

```
tr_imgs = [f'{tr_dir}/{i}' for i in os.listdir(tr_dir)]
```

```
ts_imgs = [f'{ts_dir}/{i}' for i in os.listdir(ts_dir)]
```

```
def process_imgs(imgs, width=150, height=150):
```

```
    x = []
```

```

y = []
for i in imgs:
    x.append(cv2.resize(cv2.imread(i, cv2.IMREAD_COLOR),
                        (width, height),
                        interpolation=cv2.INTER_CUBIC))

    label = 1 if 'dog' in i else 0
    y.append(label)

return np.array(x), np.array(y)

tr_x, tr_y = process_imgs(tr_imgs)

plt.figure(figsize=(20, 10))
cols = 5
for i in range(cols):
    plt.subplot(5 / cols+1, cols, i+1)
    plt.imshow(tr_x[i])

from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing.image import img_to_array, load_img

tr_data = ImageDataGenerator(rescale=1/255,
                              rotation_range=40,
                              width_shift_range=0.2,
                              height_shift_range=0.2,
                              shear_range=0.2,
                              zoom_range=0.2,
                              horizontal_flip=True)

tr_gen = tr_data.flow(tr_x, tr_y, batch_size=32)

```



I will not walk you through all the code but what it does is get the images from the local folders we have just imported into the workspace, and process them into numpy arrays. We are also resizing all the images to 150x150, and creating Image Generators so we can use image augmentation. You can find more details on image generators and all the methods used on the Keras documentation page.

*Run the code by pressing the play button (on the left of the cell) or hold Shift and press Enter.*

## Import a pretrained model from Keras

Click on **+ Code** (below the last added cell), to add a new code cell and write the following lines of code:

```
from keras.applications import InceptionResNetV2

base=InceptionResNetV2(weights='imagenet',

                        include_top=False,

                        input_shape=(150, 150, 3))
```

We **import** the **InceptionResNetV2** model.

Next, we tell keras to download the model's pretrained weights and save it in the variable **base**.

To configure what we actually download, we pass in some important parameters such as:

**weights [imagenet]:** We tell keras to fetch InceptionReNetV2 that was trained on the imagenet dataset.

**include\_top [False]:** This tells Keras not to download the fully connected layers of the pretrained model. This is because the top layer (fully connected layers) does the final classification. I.e After the convolution layers extract basic features such as edges, blobs or lines from the input images, the fully connected layer then classifies them into categories.

Since all we need is 2 class (dogs and cats) classifier, we are going to remove the former and add our own.

**input\_shape [(150, 150, 3)]:** Here we specify our input dimension, this means we have a square image of 150 by 150 pixels and it has 3 color channels, RGB.

If you get any errors when running the code you have forgotten to activate the internet connection. This is needed as we have to download the pretrained weights.

*Run the code by pressing the play button (on the left of the cell) or hold Shift and press Enter.*

Next, we create our fully connected layers (classifier) which we add on-top of the model we downloaded. This is the classifier we are going to train. I.e after connecting the InceptionResNetV2 to our classifier, we will tell keras to train only our classifier and freeze the InceptionResNetV2 model.

Click on **+ Code** (below the last added cell), to add a new code cell and write the following lines of code:

```
from keras import layers, models

model = models.Sequential()

model.add(base)

model.add(layers.Flatten())

model.add(layers.Dense(256, activation='relu'))

model.add(layers.Dense(1, activation='sigmoid'))

base.trainable = False
```

Here we import layers and models from keras, and create a Sequential model.

We then add the pretrained base to our model and start adding new layers to tailor for the cats and dogs dataset we will use.

We flatten the output from the **base** because we want to pass it to our fully connected layer (classifier).

To keep things simple, I added a Dense network with an output of 256 (number not fixed) and used the popular ReLU activation.

We use a sigmoid for our final layer like we did last time.

The last layer has just 1 output. (Probability of classes)

Lastly we freeze our pretrained **base** so we can only train the new weights, which we have just added.

*Run the code by pressing the play button (on the left of the cell) or hold Shift and press Enter.*

Next we need to get an optimizer and compile the model before we fine-tune it.

Click on **+ Code** (below the last added cell), to add a new code cell and write the following lines of code:

```
from keras import optimizers

batch_size = 32

epochs = 20

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=2e-5),
              metrics=['acc'])

hist = model.fit_generator(tr_gen,
                          steps_per_epoch=tr_x.shape[0] // batch_size,
                          epochs=epochs)
```

We need to compile the model (this will build the tensorflow computational graph automatically) and lastly train it on the training data.

After you run this last cell you should see how your model is performing.

You can now try to test the performance on your test data. NB: You should look at the keras documentation and see how you can evaluate a model on the test data.

References:

<https://www.kaggle.com/kanncaa1/machine-learning-tutorial-for-beginners>

<https://towardsdatascience.com/image-detection-from-scratch-in-keras-f314872006c9>

<https://towardsdatascience.com/transfer-learning-and-image-classification-using-keras-on-kaggle-kernels-c76d3b030649>