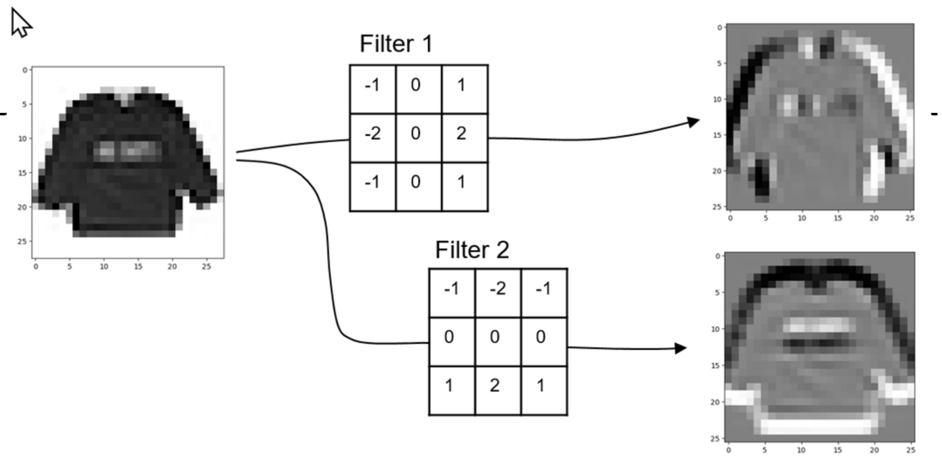
Deep Learning
Lecture 8

Data Augmentation
Pretrained Models and Transfer Learning

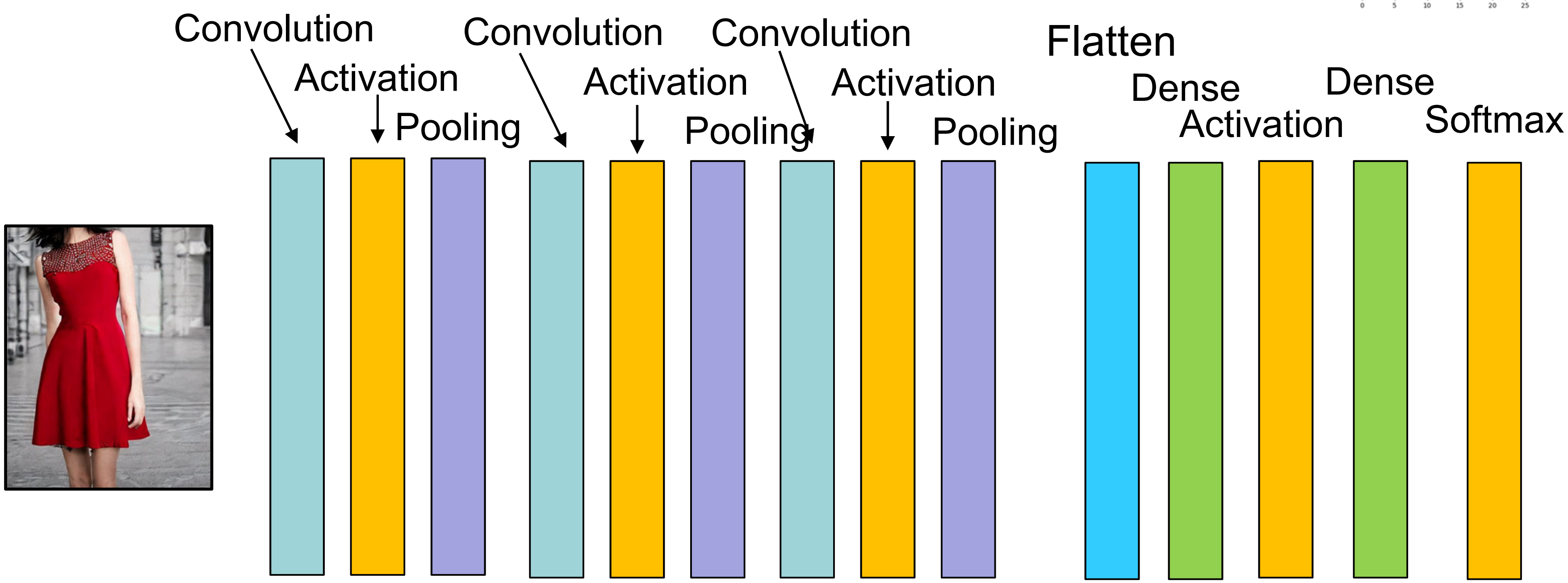
Prof. Dr. Rainer Herrler
Phone: 09721/ 940-8710
Email: rainer.herrler@fhws.de

Pictures from Wikipedia / Pixabay
Some Pictures generated with Stable Diffusion

Recap CNNs



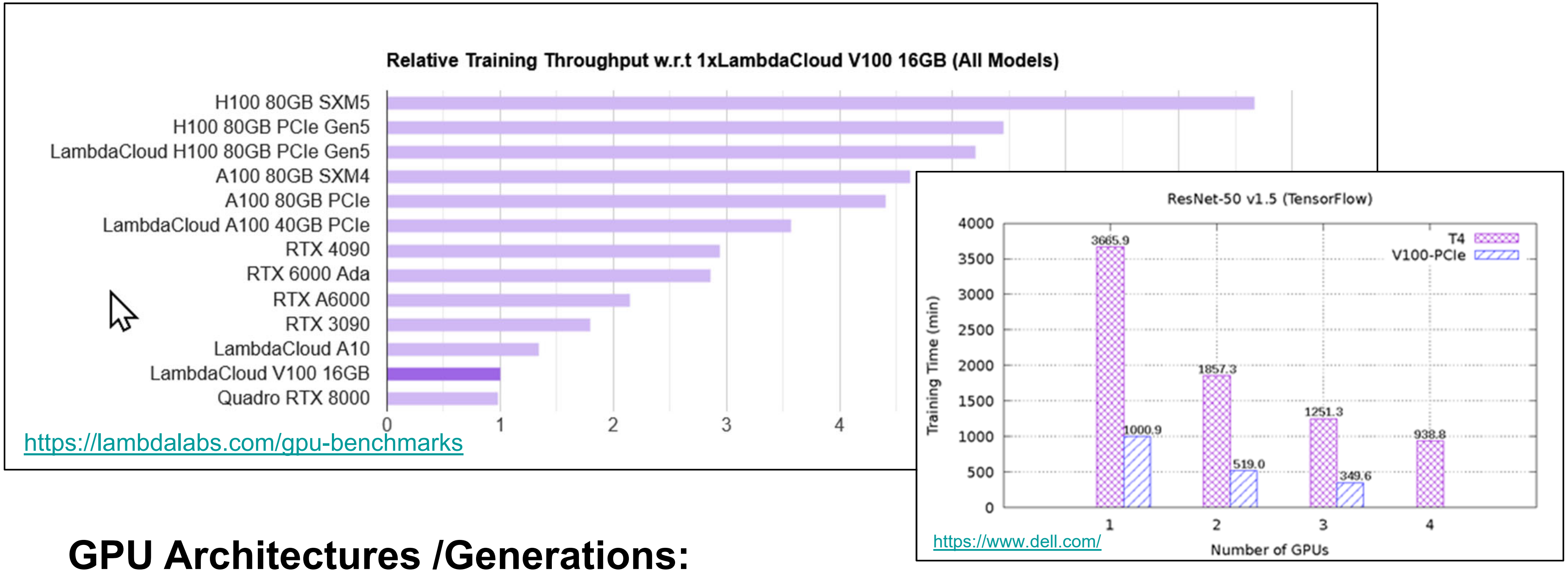
Typical Pattern:



Other layers like Normalization / Dropout etc can also be inserted

Question: How can we see what’s generated from the first layer?

GPU's for Deep Learning



GPU Architectures /Generations:

- Kepler, Pascal, Volta (2017), Turing (2018), Ampere (2020), Ada & Hopper (2022)

Use Case:

- Tegra (Embedded) / GeForce (Consumer)
- Quadro (Workstation) / Tesla (Datacenter)

e.g. my Notebook GTX 1650 mobile is a Turing Architecture ~6x slower than T4

Overview on current hosted services including GPU

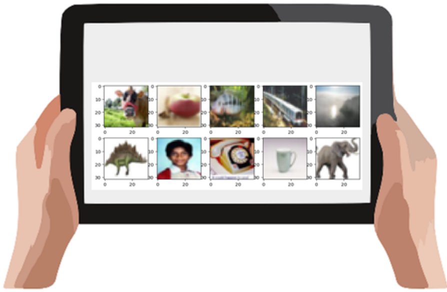
Free hosted Jupiter Notebooks

- Google Colab
  - https://colab.research.google.com/
  - Direct connection to google drive for persisting data (model weights, training data)
  - T4 GPU or comparable TPU
  - Upgradeable to Colab Pro for stonger GPUs
- Kaggle
  - https://www.kaggle.com/
  - Good for exercises with data provided by Kaggle
  - Provides P100 and T4 GPUs as well as TPU's
- Paperspace
  - https://www.paperspace.com/gpu-cloud
  - Free GPUs of different types, even very powerful ones
  - Very often out of resources

Payed Services

- Microsoft Azure
  - Free 100$ budget for students
  - Many Services provided –e.g. Azure Machine Learning Studio
  - Complex setup
  - Eg 90cent per h for Tesla K80
- Lambda Labs
  - https://lambdalabs.com/service/gpu-cloud/pricing
  - Newest types of GPU,
  - Good prices starting from 0.5$ per h
- Vast.ai
  - https://cloud.vast.ai/create/
  - Very Cheap options available,
  - From private or Datacenter
- Amazon, Jarvislabs…

Exercise



Try to create a CNN for CIFAR 100

- Short Link: https://bit.ly/3Oj1YeG
- Full Link: https://colab.research.google.com/drive/1x_PCB1JdyLZgG17nJ8ePBoO95RO22C7C?usp=sharing

- Difficulties
  - Many classes and few samples
  - Complex networks will overfit very fast
  - Regularization is needed!
- Realistic outcome:
  - Rates around >35 % accuracy should be achievable
  - More complex networks can reach 60% accuracy and more

DL_015_CIFAR-100-CNNs.ipynb

Further comments when training in the cloud

- Save the training results regularly
  - Typically your notebooks are persistently stored
  - Training data and trained weights are **not on a (very) permanent store**
  - Store training data somewere else (Github/Kaggle)
  - Connect Network Drive for output
  - Use Checkpoint-Callbacks to store data

```
from google.colab import drive
drive.mount('/content/gdrive')
```

- Prevent logout
  - Free services often stop the python kernel after a long period of inactivity or if they are short on resources
  - If you use the free service and you need a longer runtime while you are afk you can use browser plugins to simulate activity
  - In doubt switch to a payed plan

Data Augmentation
Extend your training data

Data Augmentation

- **Definition:** Data Augmentation creates more samples by slight modifications that usually don't have an effect on the predicted class
- Example Modifications for Images
  - Shifting, Skaling, Rotation
  - Adjusting Contrast, Saturation, Brightness
  - Adding noise
- Example Modification for Audio
  - Add Pitch
  - …
- Further Information:
  - https://www.datacamp.com/tutorial/complete-guide-data-augmentation

Data Augmentation with Tensorflow

- Option 1: Using special Layers in the model
  - RandomFlip
  - RandomRotation
  - With every epoch a randomly modified sample is being shown
  - We can create an extra Network for augmentation
  - These layers apply the random modification **just in training**
  - https://keras.io/api/layers/preprocessing_layers/image_augmentation/
- Option 2: Using the ImageDataGenerator
  - Here augmentation is not part of the model itself, instead it’s a sperate step
  - https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator
- Option 3: Use Keras-Function to read from Directory
  - Augmentation can be done while “streaming” data from the filesystem
  - `tf.keras.preprocessing.image_dataset_from_directory`
  - https://keras.io/api/preprocessing/image/

Data Augmentation Notebook

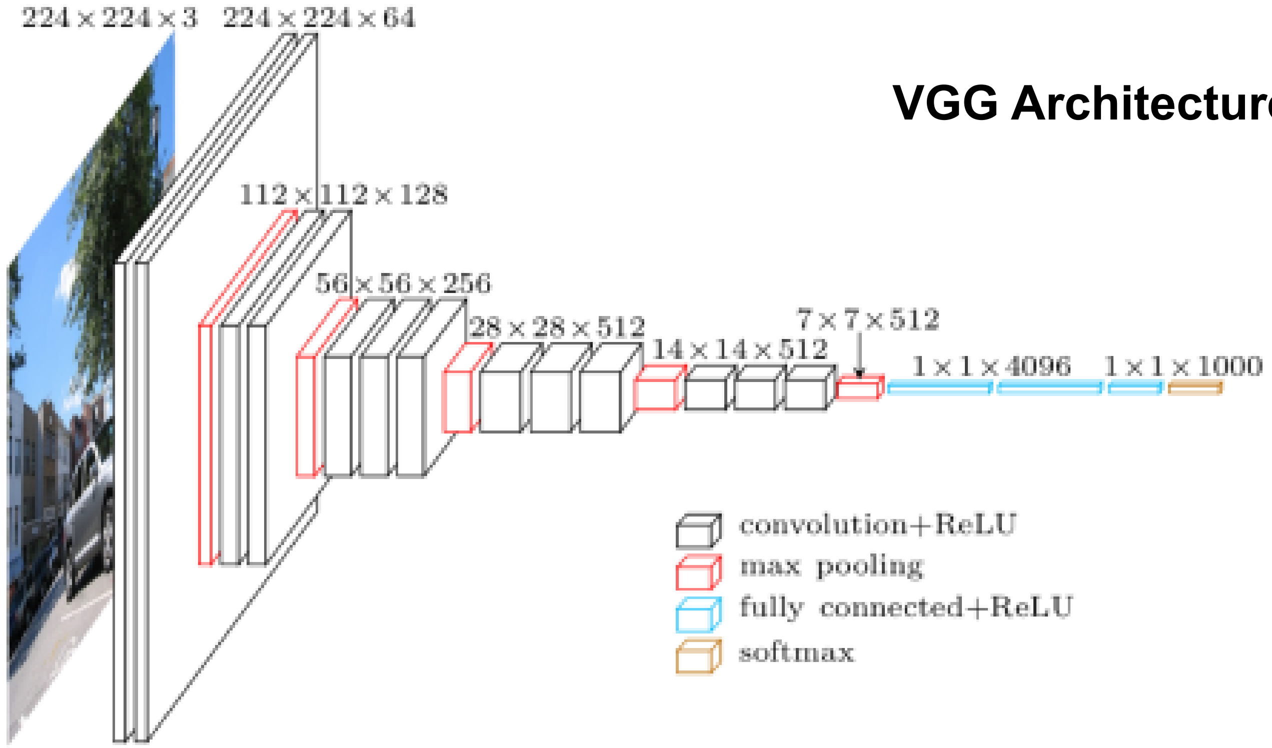
[DL_014_DataAugmentation.ipynb]

What did we learn from the last Notebook

- Data Augmentation helps to fight overfitting
  - **accuracy: 0.9400** - val_accuracy: **0.2145 (without augmentation)**
  - **accuracy: 0.3980** - val_accuracy: **0.3218 (with augmentation)**
- Networks can be nested
  - Use network as a layer to compose more complex models
  - Take care that input and output layers fit together and that input_size for first layer is given
- Streaming images with Iterators
  - Images for training don’t have to be in memory, but can be loaded step by step
  - Creation of Iterators for streaming
    - `ImageDataGenerator.flow(..`
    - `tf.keras.utils.image_dataset_from_directory()`
- We have written a own training-Loop for several epochs
  - Most of the times this is not necessary but it gives you additional flexibility

CNN-Architectures and pretrained models

VGG Architecture



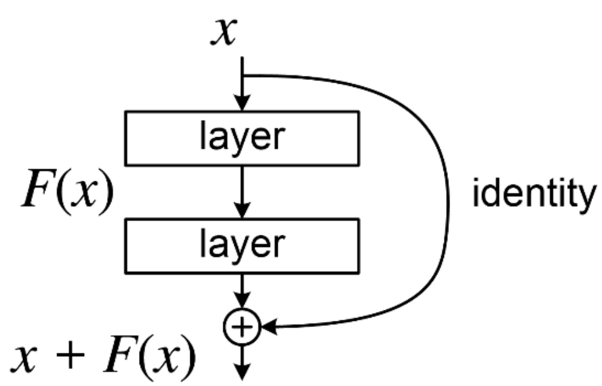
Pretrained Models for Classification in Keras

- Architectures
  - VGG16 - 16 Layers
  - VGG19 – 19 Layers
  - InceptionV9
  - ResNet (50 to 100Layers)
    Residual Connections
  - EfficientNetB0
- Pretrained Models
  - Weights available for different image training sets
  - Imagenet, COCO etc

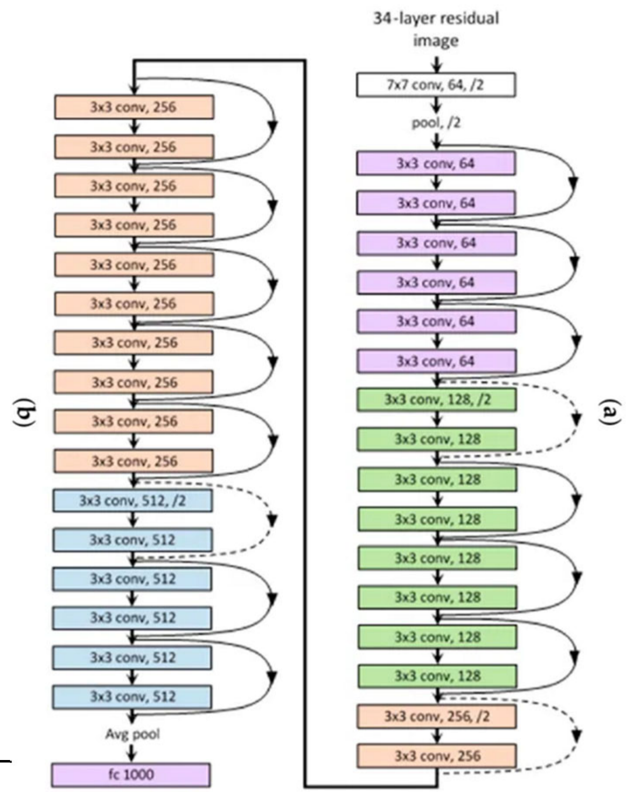
https://pub.towardsai.net/from-vggnet-to-efficientnet-key-milestones-in-the-evolution-of-cnn-design-d778aa1e1bed

ResNet Architecture (2015)

- ResNet is the abbreviation of Residual Neural Network
- Authors: Kaiming He et al (formerly Microsoft now Facebook)
- Design:
  – Introduction of so called “Skip-Connections” aka “residual connections” to form Residual Blocks
  – A couple of these blocks are stacked
- Effect:
  – Identity function allows the upstream gradient to better flow backwards (no vanishing gradients)
  – Networks can be deeper



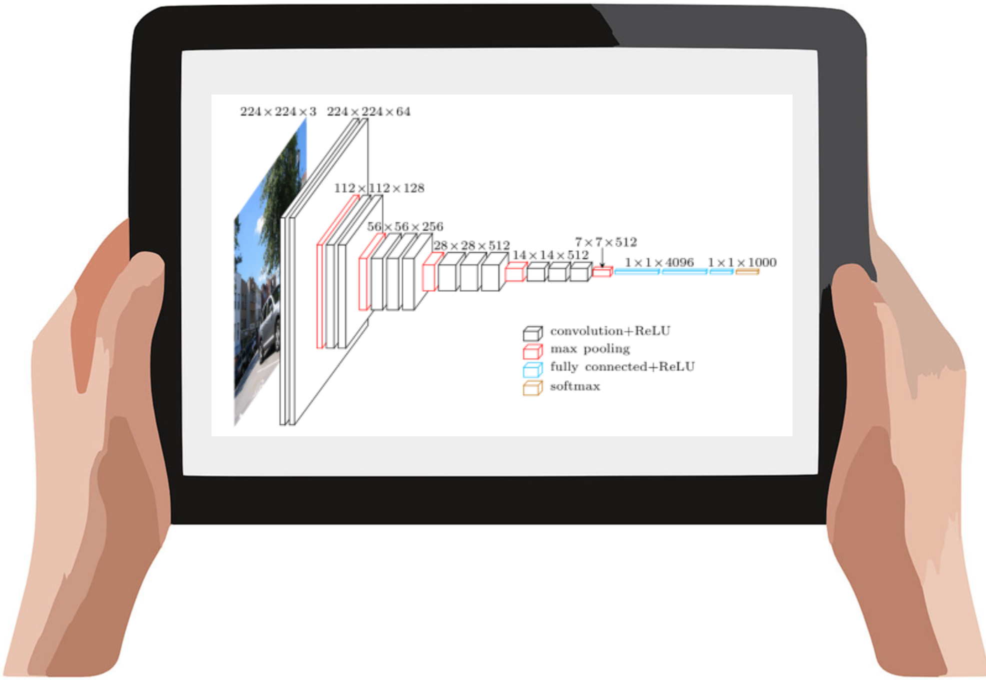
Picture from Wikipedia



Picture from original Paper of Kaiming He

Applying pretrained models (Trained on ImageNet)

DL_014_VGG16.ipynb
DL_015_VGG19.ipynb
DL_016_InceptionV3.ipynb
DL_017_EfficientNetB0.ipynb



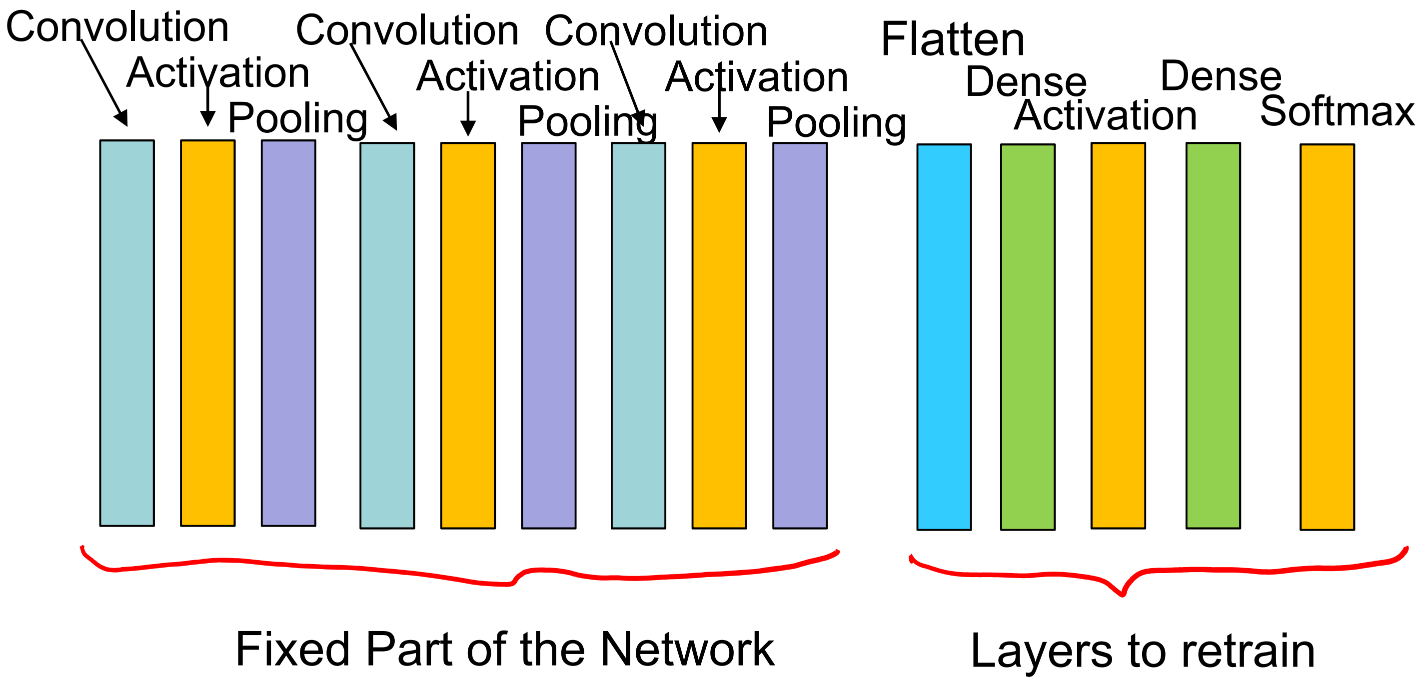
For these CNN-Architectures
we can get weights for
different image datasets

All classes of the image dataset used for training can be predicted!

Transfer learning

- NN models can just predict what they are trained for
  - Same kind of input format (pixel resolution and channels)
  - Same prediction classes (e.g 100 different classes)
- For example a NN trained on Imagenet
  - can distinguish between cats and dogs
  - cannot distinguish between different breeds of dogs
- Luckily we can reuse models trained on different tasks and extend them to new use cases
  - Early layers are frozen and weights later layers are recreated (reinitialized)
  - Then they are trained with new samples
- Transfer learning is applicable even when…
  - …we don't have enough data for the new task to train
  - …we don't have the time and resource to train from scratch

Transfer Learning Idea



Model is already capable of:

- Detecting Edges and Corners
- Textures & Patterns
- Object Parts

Transfer Learning Idea

- Different approaches possible
  - Just retrain the last Fully Connected Layer
  - Retrain multiple layers from the end (all Dense layers)
  - You can also unlock earlier Convolutional Layers and adjust weights here (e.g Fine Tune / improve results with new data)

- General Rules
  - The more layers you retain the more data and epochs are required
  - You typically shouldn't unlock certain types of layers like BatchNormalization Layers
    (Reference: https://pub.towardsai.net/batchnorm-for-transfer-learning-df17d2897db6)

Reusing Pretrained Modells (on Imagenet) for own tasks

DL_024_Transfer Learning-EfficientNet.ipynb

https://bit.ly/42P1jGm