Creating a New CNN Architecture for the ILSVRC Object Classification and Localization Challenge

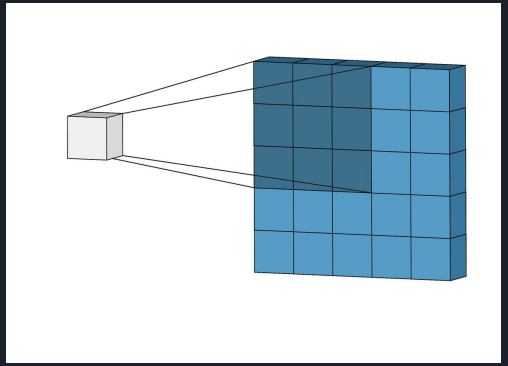
By Parasmai Salunkhe
Period 4, Dr. Yilmaz

Objective

- Preform object detection on the ImageNet dataset
- Create A CNN architecture that achieves a better Top 1 and Top 5 accuracy compared to the past ILSVRC competition winners from 2010-2017 (AlexNet, GoogLeNet, ResNet, SEnet)
- Metrics used: Top 1 accuracy: (prediction made by model), Top 5 accuracy (top 5 predictions made by model)

What is a Convolutional Layer?

- The input is typically a multidimensional array
- A filter with learnable weights.
- The filter slides over the input image, performing multiplication between the filter and the corresponding region of the input.
- This process is repeated across the entire input.
- Stride: Determines how much the filter moves across the input. A stride of 1 moves the filter one pixel at a time, while higher strides skip pixels.
- Padding: Adds borders to the input to control the size of the output.
- Same padding keeps the output size equal to the input size.
- Valid padding results in a smaller output
- activation function (commonly ReLU because it is know for a faster convergence in training)
- The result of the convolution is a feature map (or activation map) that highlights the presence of specific features in different regions of the input.

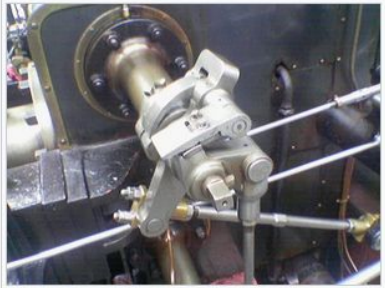


Example of a Famous CV Kernel

Sobel kernel

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A}$$

$$\mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$



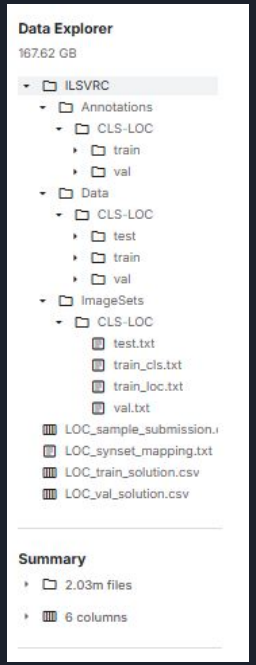
A color picture of an engine

The Sobel operator applied to that image

The ImageNet Dataset/ILSVRC Competition

- ImageNet is a dataset of over 15 million labeled high-resolution images belonging to roughly 22,000 categories.
- The images were collected from the web and labeled by human labelers using Amazon's Mechanical Turk crowdsourcing tool.
- ILSVRC (ImageNet Large Scale Visual Recognition Challenge) uses a subset of ImageNet with roughly 1000 images in each of 1000 categories. In all, there are roughly 1.2 million training images, 50,000 validation images, and 150,000 testing images.
- Goal Is to classify a 256 by 256 image into 1000 classes and localize detections (draw bounding boxes around detections in image)

The ImageNet Dataset/ILSVRC Competition (Continued)



Dataset Description

File descriptions

ILSVRC/ contains the image data and ground truth for the train and validation sets, and the image data for the test set.

- The image annotations are saved in XML files in PASCAL VOC format. Users can parse the annotations using the PASCAL Development Toolkit.
- Annotations are ordered by their synsets (for example, "Persian cat", "mountain bike", or "hot dog") as their `wnid`. These id's look like n00141669. Each image's name has direct correspondence with the annotation file name. For example, the bounding box for n02123394/n02123394_28.xml is n02123394_28.JPEG.
- You can download all the bounding boxes of a particular synset from http://www.image-net.org/api/download/imagenet.bbox.synset?wnid=[wnid]
- The training images are under the folders with the names of their synsets. The validation images are all in the same folder. The test images are also all in the same folder.
- ImageSet folder contains text files specifying lists of images for the main localization task.

LOC_sample_submission.csv is the correct format of the submission file. It contains two columns:

- `ImageId`: the id of the test image, for example `ILSVRC2012_test_00000001`
- `PredictionString`: the prediction string should be a space delimited of 5 integers. For example, `1000 240 170 260 240` means it's label 1000, with a bounding box of coordinates (x_min, y_min, x_max, y_max). We accept up to 5 predictions. For example, if you submit `862 42 24 170 186 862 292 28 430 198 862 168 24 292 190 862 299 238 443 374 862 160 195 294 357 862 3 214 135 356` which contains 6 bounding boxes, we will only take the first 5 into consideration.

LOC_train_solution.csv and **LOC_val_solution.csv**: These information are available in **ILSVRC/** already, but we are providing them in `csv` format to be consistent with **LOC_sample_submission.csv**. Each file contains two columns:

- `ImageId`: the id of the train/val image, for example `n02017213_7894` or `ILSVRC2012_val_00048981`
- `PredictionString`: the prediction string is a space delimited of 5 integers. For example, `n01978287 240 170 260 240` means it's label n01978287, with a bounding box of coordinates (x_min, y_min, x_max, y_max). Repeated bounding boxes represent multiple boxes in the same image: `n04447861 248 177 417 332 n04447861 171 156 251 175 n04447861 24 133 115 254`

LOC_synset_mapping.txt: The mapping between the 1000 synset id and their descriptions. For example, Line 1 says `n01440764 tench, Tinca tinca` means this is class 1, has a synset id of `n01440764`, and it contains the fish tench.

Background/Related Research (AlexNet)

AlexNet (2012)

- Your typical CNN architecture
- Composed of Convolutions, Max Pooling layers, and Fully Connected layers
- Top - 1 accuracy: 62.5%
- Top - 5 accuracy: 84.7%

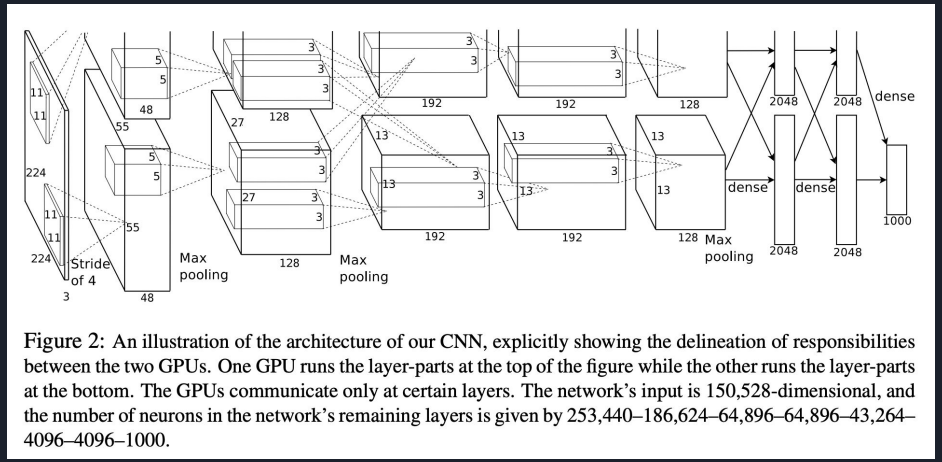
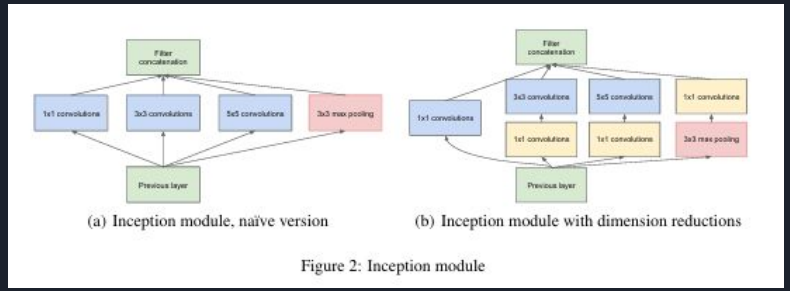


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Background/Related Research (GoogLeNet)

GoogLeNet (2014)

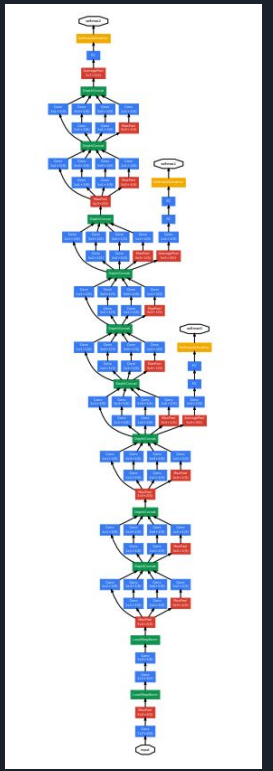
- Developed by Christian Szegedy
- Composed of 9 Inception modules
- Top -1 accuracy: 69.8%
- Top-5 accuracy: 89.6%



(a) Inception module, naïve version

(b) Inception module with dimension reductions

Figure 2: Inception module



Background/Related Research (ResNet)

ResNet (2014)

- Residual learning
- Goal - h(x)
- Input - f(x)
- Adding skip connection results in f(x) = h(x) - x
- Top-1 accuracy: 75.3%
- Top-5 accuracy: 92.2%

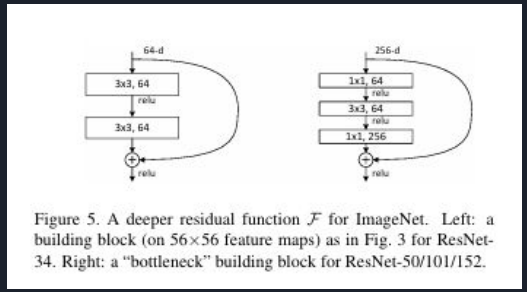


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

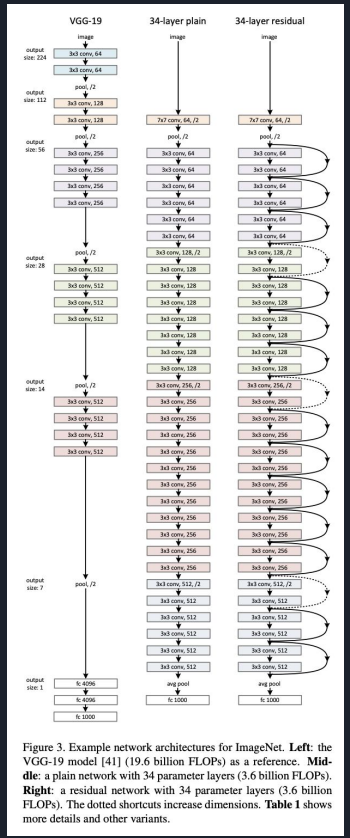


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

Background/Related Research (Xception)

- Top-1 accuracy - 79%
- Top-5 accuracy - 94.5%
- Uses Depth Wise Separable Convolutions (1x1 convolutions)
- separate spatial convolutions and depthwise convolutions, resulting in fewer parameters and faster computations.

Background/Related Research (SEnet)

SEnet (2017)

- Extends existing architectures (GoogLeNet and ResNet)
- Top-1 accuracy - 82.3%
- Top-5 accuracy - 95.0%
- Ads a SE block to every unit in the original architecture
- SE block calibrates which feature maps are more important whereas treated equally

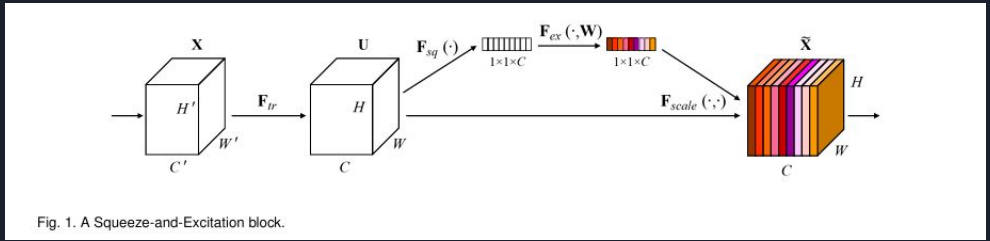


Fig. 1. A Squeeze-and-Excitation block.

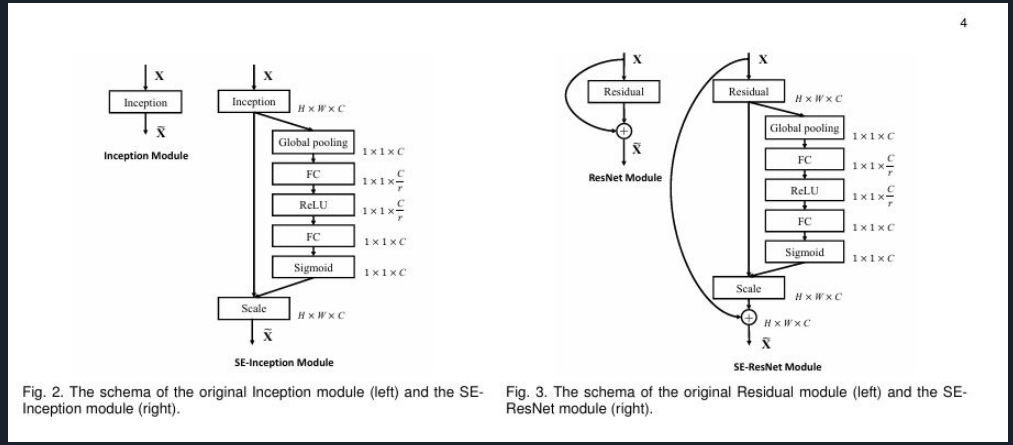
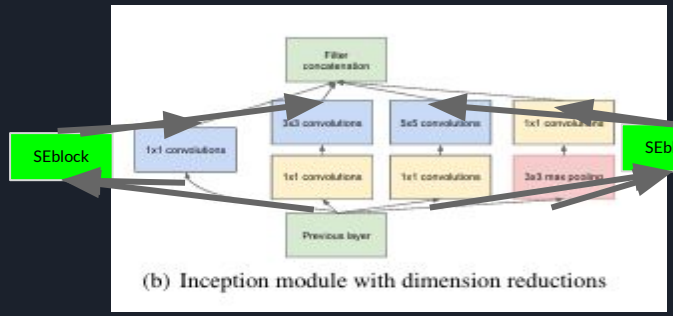


Fig. 2. The schema of the original Inception module (left) and the SE-Inception module (right).

Fig. 3. The schema of the original Residual module (left) and the SE-ResNet module (right).

Custom Model Architecture

- Use Pytorch Template of GoogLeNet (from Github)
- Add SEnet block to each path because of its feature calibration
- Keep original GoogLeNet Framework
- Increase from 6 million parameters to 13 million



(b) Inception module with dimension reductions

```
================================================================
Total params: 13,004,888
Trainable params: 13,004,888
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 94.26
Params size (MB): 49.61
Estimated Total Size (MB): 144.44
----------------------------------------------------------------
```

Preprocessing

```python
transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize images
    transforms.RandomHorizontalFlip(),  # Data augmentation
    transforms.ToTensor(),  # Convert images to PyTorch tensors
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]), #From ImageNet Website
])
```

Python

Preparing Datasets

- Solutions for Validation and Training in CSV file
- For each item apply transformation
- Create Data Loader that loads data into batches

```python
import torch
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms

class ImageCSVLoader(Dataset):
    def __init__(self, csv_file, img_dir, transform=None):
        self.data = pd.read_csv(csv_file)
        self.img_dir = img_dir
        self.transform = transform
        Allfolders =  os.listdir("/kaggle/input/imagenet-object-localization-challenge/ILSVRC/Data/CLS-LOC/train")
        self.class_map = {label: idx for idx, label in enumerate(Allfolders)}

    def __len__(self):
        return len(self.data)  # Total number of images

    def __getitem__(self, idx):
        img_name = os.path.join(self.img_dir, self.data.iloc[idx, 0])
        newLabel = img_name.replace(self.img_dir, '')
        if "train" in self.img_dir:
            image = Image.open(self.img_dir + newLabel.split("_")[0] + "/" + newLabel + ".JPEG").convert("RGB")
        else:
            image = Image.open(img_name + ".JPEG").convert("RGB")  # Load image
        label = str(self.data.loc[self.data["ImageId"] == newLabel, "PredictionString"].values[0].split()[0])
        if self.transform:
            image = self.transform(image)
        return image, torch.tensor(self.class_map[label], dtype=torch.long)

val_dataset = ImageCSVLoader(csv_file="/kaggle/input/imagenet-object-localization-challenge/LOC_val_solution.csv",
                             img_dir='/kaggle/input/imagenet-object-localization-challenge/ILSVRC/Data/CLS-LOC/val/', transform=transform)
val_dataloader = DataLoader(val_dataset, batch_size=256, shuffle=True, num_workers=4, pin_memory=True, prefetch_factor=2)

train_dataset = ImageCSVLoader(csv_file="/kaggle/input/imagenet-object-localization-challenge/LOC_train_solution.csv",
                               img_dir='/kaggle/input/imagenet-object-localization-challenge/ILSVRC/Data/CLS-LOC/train/', transform=transform)
train_dataloader = DataLoader(train_dataset, batch_size=256, shuffle=True, num_workers=4, pin_memory=True, prefetch_factor=2)

```

Training Environment/Hyperparameters

- Using Kaggle Environment
- GPU Nvidia P100 Accelerator
- Loss function: nn.CrossEntropyLoss()
- GoogLeNet: 2 Auxiliary outputs + 1 final output
- New Loss function: 2(0.3 * Aux) + 0. 4 * Final
- Hyperparameters tuning used from GoogLeNet paper
- Optimizer: optim.SGD(model.parameters(), lr=0.1, momentum=0.9, weight_decay=1e-4) Used by GoogLeNet
- Train for 10 epochs

Results

- After 6 epochs (Kaggle Timeout after 12 hours twice)

  Epoch 3/10: Train Loss = 11.0243, Val Loss = 11.0786, Val Accuracy = 0.10%

- Epoch 6/10: Train Loss = 10.935, Val Loss = 11.0585, Val Accuracy = 0.11%
- Succumbed to the Vanishing/Exploding Gradients phenomenon and a Kaggle Notebook Timeout twice

The Vanishing/Exploding Gradients Problem

- Gradients get smaller and smaller
- Gradient Descent update leaves the lower layers virtually unchanged
- Backpropagating complete nonsense for many epochs
- Alternatively Gradients can explode causing error to go to “nan”
- Solutions to this:
  - Reduce Learning rate (using scheduler after x number of epochs)
  - Decrease number of parameters while Increase depth of model
  - Add skip connections (ResNet)
  - Different weight Initialization (Glorot and He)

Limitations

- Succumbed to the Exploding/Vanishing Gradients problem
- Number of Parameters (13 million) hence requiring lots of GPU
- Amount GPU/Computing Processing available
- JupyterHub very slow + (Cuda drivers outdated)
- Kaggle Timeouts

Conclusion

- More GPU required to train
- Using SEnet Blocks require more parameters and time
- General trend of all these models is that the models become deeper and deeper but at the same time reducing the number of parameters used
- If properly trained and hyperparameters tuned properly, accuracy would be around that of GoogLeNet

Sources

1409.4842 Googlenet

https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf AlexNet

https://arxiv.org/pdf/1512.03385 ResNet

https://arxiv.org/pdf/1709.01507 SEnet

https://medium.com/analytics-vidhya/how-to-train-a-neural-network-classifier-on-imagenet-using-tensorflow-2-ede0ea3a35ff

https://medium.com/analytics-vidhya/talented-mr-1x1-comprehensive-look-at-1x1-convolution-in-deep-learning-f6b355825578

https://paperswithcode.com/method/googlenet

https://www.geeksforgeeks.org/understanding-googlenet-model-cnn-architecture/

https://paperswithcode.com/sota/image-classification-on-imagenet?metric=Top%205%20Accuracy

https://pmc.ncbi.nlm.nih.gov/articles/PMC1357023/pdf/jphysiol01301-0020.pdf

https://pmc.ncbi.nlm.nih.gov/articles/PMC1363130/pdf/jphysiol01298-0128.pdf

Géron, Aurélien. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 2nd ed., O'Reilly Media, 2019.