

A Novel Approach to the ILSVRC Classification Competition

Parasmai Salunkhe

Thomas Jefferson High School for Science and Technology

Machine Learning

Dr. Yilmaz

Abstract

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) has served as a benchmark for evaluating the performance of deep learning models in image classification tasks. Despite significant advancements in model architectures and training techniques, there remains a continuous drive to enhance classification accuracy, reduce computational cost, and address generalization to diverse real-world datasets. This paper proposes several novel strategies to improve upon the current state-of-the-art performance in the ILSVRC. In this paper, I will go through the basics of object detection starting with the basic terminology of CNN and how we can extract more information using advanced techniques such as pooling, up/down sampling, and more techniques. In this paper I will also go through the different existing high performing models such as AlexNet, ResNet, GoogLeNet, and many other noble Cnn architectures. Our experimental results, conducted on the ILSVRC validation set, attempt to demonstrate substantial improvements in classification accuracy over existing models but eventually succumbed to the problem of exploding and vanishing gradients. Finally, we discuss perhaps more areas of improvement for future object detection and localization.

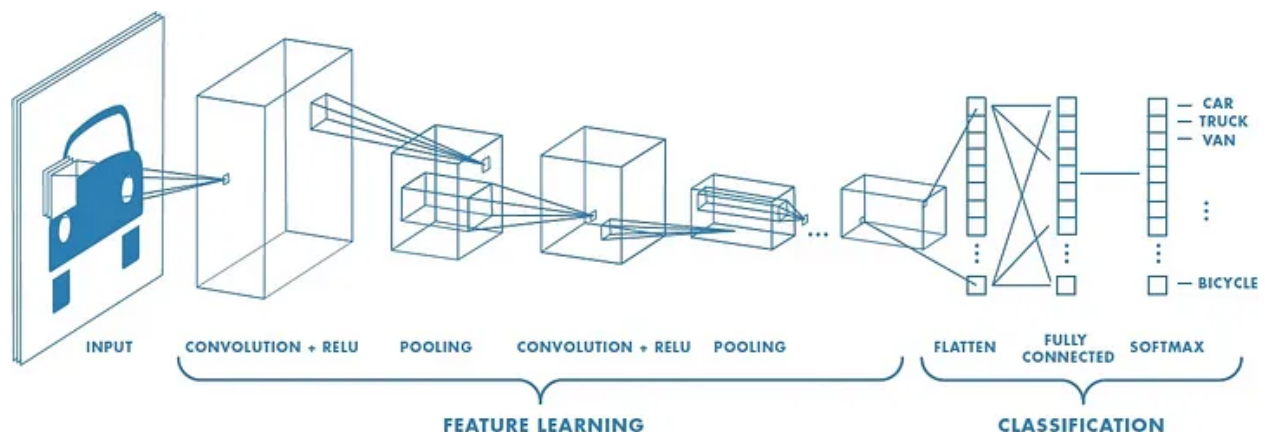
Introduction

Object detection and localization tasks are pretty simple tasks for humans (finding certain objects and where their locations are), but for machines both of these tasks become more complicated. As humans, finding and detecting objects is a pretty effortless task, the answer to this lies in the fact that this process lies outside of our consciousness with specialized visual, auditory, and other sensory modules in our brains. Once we find an object we want to look at you cannot choose not to see something (ex: a puppy, its cuteness, its color). One notable thing to mention is that as humans, we cannot explain how to “look” at something, it's just that our eyes and brain are pre trained to capture things and things about them as the human mind later develops. But for a machine, the task becomes simply daunting, unless you can somehow pick up patterns along the way.

In order to understand how the brain's visual cortex works, David H. Hubel and Torsten Wiesel performed an experiment on cats in 1958 and later in 1959 on monkeys that gave an important view into the structure of the visual cortex. One key finding from their research showed that many “many neurons in the visual cortex have a small local receptive field, meaning they react only to visual stimuli located in a limited region of the

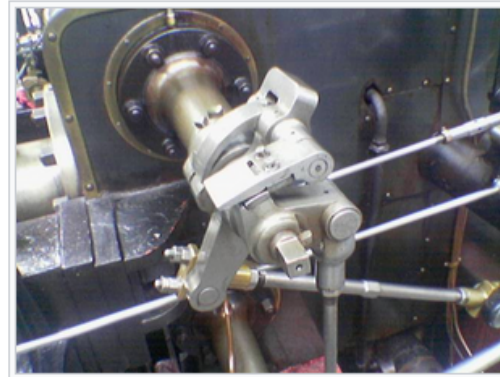
visual field”. Also the Authors show that some particular neurons reacted only to images of horizontal lines while others reacted to lines with different orientations. Other findings from these 2 experiments also indicated some neurons having a larger receptive field while others having smaller receptive fields leading to the idea that these neurons are based on the outputs of neighboring lower-level neurons. Based on all of the knowledge on how the Visual Cortex works in mammals, The closest thing that can represent the complexity of a real visual cortex is the Convolutional Neural Network where in a simple case like the MNIST dataset (classifying handwritten integers), a fully connected Deep neural network may be ideal, but a significant drawback when using large images with a Deep neural network is that as the number of pixels increases for a given image it starts to break down for larger images because of this massiver number of parameters. For example an 100 by 100 image has 10k pixels and if the first layer has 1000 neurons (which is severely restricting the amount of information transmitted to the next layer) and with the total amount of connections in the high millions. The Dnn would take up lots of space and time. This is where Cnn’s solve the problem by using partially connected layers and weight sharing.

One of the most important building blocks, a Convolutional layer is a specific type of deep learning model specifically designed for processing structured data (ex: images). They consist of layers that automatically learn spatial hierarchies of features from input data. They are also tuned to detect different patterns in images once all strung together. The basics of a Convolution layer lie in something called a kernel which is basically a square of n by n dimensions n being a natural integer which is then slid across an image capturing details along the way as show here:



Because it is possible to connect a large input layer to a much smaller layer by spacing out the receptive fields, this dramatically decreases the model's computational complexity. In a CNN the shift of the one receptive field in the next layer is called the stride (in this case 1) and the kernel which is a small matrix of weights which traverses

the whole image (in this case 3x3) and the kernels weights can be optimized for detecting different edges for example. An example of a well known kernel is the Sobel kernel can detect vertical or horizontal edges by emphasizing gradients in specific directions.



A color picture of an engine

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

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



The Sobel operator applied to that image

Nevertheless, the weights of a kernel are initially instantiated to be random, but as backpropagation and training progresses, the kernel's weights are fitted and adjusted to detect specific features.

Related work

From 2010 - 2017, there have been many groundbreaking architectures such as AlexNet and all the way to SEnets. AlexNet, developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, is a groundbreaking convolutional neural network (CNN) architecture that significantly advanced the field of computer vision. Introduced in 2012, AlexNet won the (ILSVRC) by a substantial margin, reducing the top-5 error rate from

26% to 15.3%. The architecture consists of eight layers, with five convolutional layers followed by three fully connected layers, and it was one of the first deep learning models to leverage the power of GPUs for training, which allowed for the processing of large datasets in a reasonable time frame. The architecture of AlexNet is characterized by its use of convolutional layers, ReLU (Rectified Linear Unit) activation functions, and max pooling layers.

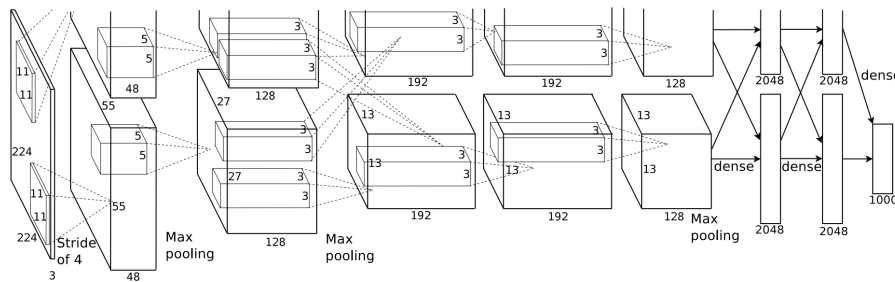


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input 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.

The first layer takes an input image of size 227x227 pixels and applies 96 convolutional filters of size 11x11 with a stride of 4, followed by a ReLU activation. Subsequent layers include additional convolutional layers with varying filter sizes and depths, max pooling layers to reduce spatial dimensions, and dropout layers to prevent overfitting. The final layers are fully connected, culminating in a softmax layer that outputs the probabilities for the 1,000 classes in the ImageNet dataset. Notably, AlexNet also introduced local response normalization (LRN) to enhance generalization. AlexNet's success can be attributed to several key innovations. The use of ReLU activation functions instead of traditional sigmoid or tanh functions allowed for faster training and mitigated the vanishing gradient problem. The architecture also employed data augmentation techniques, such as image translations and horizontal reflections, to artificially expand the training dataset and improve model robustness. Additionally, the implementation of dropout during training helped reduce overfitting.

VGGNet, developed by the Visual Geometry Group at the University of Oxford, is a convolutional neural network (CNN) architecture that gained prominence after its performance in the ILSVRC 2014 competition. The architecture is known for its simplicity and depth, consisting of 16 to 19 weight layers, depending on the variant used (VGG16 or VGG19). VGGNet achieved remarkable results in image classification tasks, significantly improving the state of the art at the time and demonstrating the effectiveness of deeper networks in learning complex features from images. The

VGGNet architecture is characterized by its use of small convolutional filters (3x3) stacked on top of each other, which allows the network to learn more complex features while maintaining a manageable number of parameters. The architecture typically starts with a series of convolutional layers followed by max pooling layers to reduce spatial dimensions. VGGNet employs a uniform architecture, where the number of filters increases as the depth of the network increases, starting from 64 filters in the first layer and going up to 512 filters in the deeper layers. The final layers consist of fully connected layers, culminating in a softmax layer for classification. This design emphasizes the importance of depth and the use of small filters to capture fine-grained details in images. The VGGNet architecture is characterized by its use of small convolutional filters (3x3) stacked on top of each other, which allows the network to learn more complex features while maintaining a manageable number of parameters. The architecture typically starts with a series of convolutional layers followed by max pooling layers to reduce spatial dimensions. VGGNet employs a uniform architecture, where the number of filters increases as the depth of the network increases, starting from 64 filters in the first layer and going up to 512 filters in the deeper layers. The final layers consist of fully connected layers, culminating in a softmax layer for classification. This design emphasizes the importance of depth and the use of small filters to capture fine-grained details in images. VGGNet was trained on the ImageNet dataset using stochastic gradient descent (SGD) with momentum and a learning rate schedule. VGGNet's performance on the ImageNet dataset was impressive, achieving a top-5 error rate of 7.3% for VGG16 and 7.1% for VGG19, which positioned it among the top-performing models of its time.

GoogLeNet, also known as Inception v1, is a convolutional neural network (CNN) architecture developed by researchers at Google, which gained significant attention after its success in the ILSVRC in 2014. The architecture is notable for its innovative use of the Inception module, which allows for the simultaneous extraction of features at multiple scales. GoogLeNet achieved a top-5 error rate of 6.67%, outperforming previous models while using fewer parameters, demonstrating that deeper networks can be both efficient and effective in learning complex representations from images.

The architecture of GoogleNet is characterized by its unique Inception modules, which consist of multiple convolutional filters of different sizes (1x1, 3x3, and 5x5) applied in parallel, along with pooling layers.

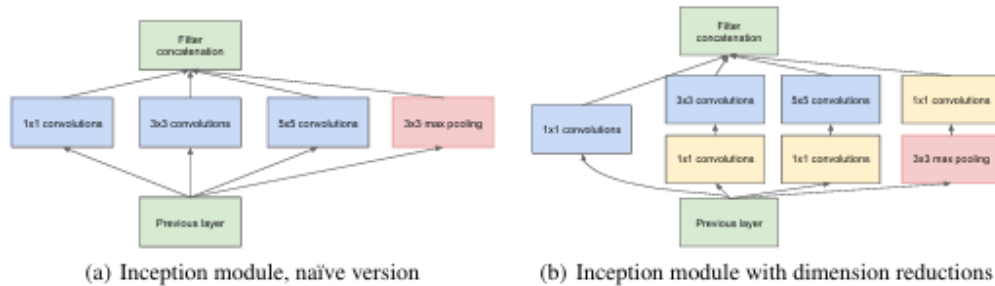


Figure 2: Inception module

This design enables the network to capture a wide range of features at various spatial resolutions. The use of 1x1 convolutions serves a dual purpose: it reduces the dimensionality of the input, thereby decreasing the computational cost, and it allows for the introduction of non-linearity. GoogleNet is significantly deeper than its predecessors, with 22 layers, but it maintains a relatively low number of parameters due to the efficient use of the Inception modules and the incorporation of global average pooling in the final layers, which replaces the fully connected layers typically found in earlier architectures.

ResNet, or Residual Network, is a revolutionary convolutional neural network (CNN) architecture introduced by Kaiming He and his colleagues at Microsoft Research in 2015. It gained significant attention after winning the ILSVRC that year, achieving an impressive top-5 error rate of just 3.57%. The key innovation of ResNet lies in its use of residual connections, which allow gradients to flow more easily through the network during training. This effectively addresses the vanishing gradient problem that often hampers the training of very deep networks, demonstrating that it is feasible to train networks with hundreds or even thousands of layers without suffering from performance degradation. The architecture of ResNet is characterized by its unique residual blocks, which consist of two or three convolutional layers with skip connections that bypass one or more layers.

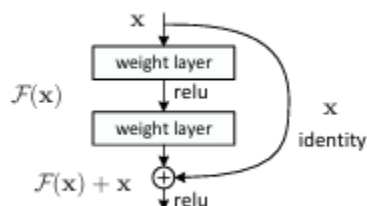


Figure 2. Residual learning: a building block.

These skip connections enable the network to learn the residual mapping instead of the original unreferenced mapping, allowing it to focus on learning the differences between the input and output. This design not only facilitates the training of deeper networks but also enhances generalization by allowing the model to learn identity mappings when necessary. The original ResNet architecture includes several variants, such as ResNet-50, ResNet-101, and ResNet-152, indicating the number of layers in each model. The use of batch normalization and ReLU activation functions further contributes to the robustness and efficiency of the architecture. The introduction of residual connections allows for the training of networks with hundreds of layers, which was previously thought to be impractical. This innovation has inspired a new wave of research into deep learning architectures, leading to the development of even deeper networks and variations of ResNet, such as ResNeXt and DenseNet, which build upon the principles of residual learning.

Squeeze-and-Excitation Networks (SENet) is a groundbreaking convolutional neural network architecture introduced by Jie Hu, Li Shen, and Gang Sun in 2017. SENet gained significant recognition after winning the ILSVRC in 2017, achieving a top-5 error rate of 2.251%. The key innovation of SENet lies in its introduction of the squeeze-and-excitation (SE) block, which enhances the representational power of the network by explicitly modeling the interdependencies between channels. This allows the network to focus on the most informative features, leading to improved performance across various tasks in computer vision.

The architecture of SENet is built upon existing convolutional neural network frameworks, such as ResNet, by integrating SE blocks into the architecture. Each SE block consists of two main operations: "squeeze" and "excitation." The squeeze operation aggregates feature maps across spatial dimensions using global average pooling, producing a channel descriptor that captures the global information of the feature maps. The excitation operation then applies a fully connected layer followed by a sigmoid activation function to generate channel-wise weights, which are used to scale the original feature maps.

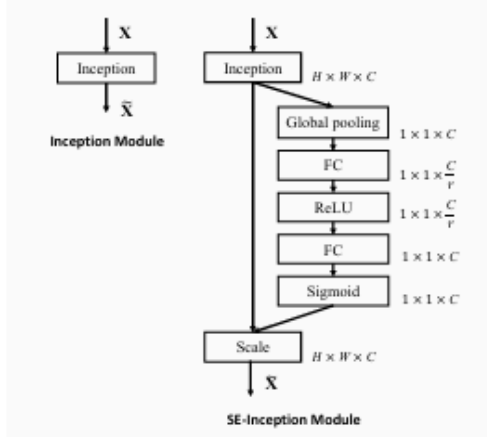


Fig. 2. The schema of the original Inception module (left) and the SE-Inception module (right).

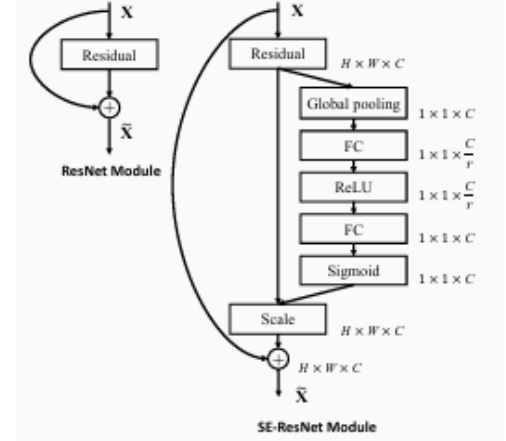



















Fig. 3. The schema of the original ResNet module (left) and the SE-ResNet module (right).

This adaptive recalibration of feature maps allows the network to emphasize important features while suppressing less informative ones, effectively enhancing the model's ability to learn discriminative representations. One of the significant contributions of SENet is its ability to improve the performance of existing architectures without significantly increasing the computational cost. By incorporating SE blocks, SENet can achieve state-of-the-art results on various benchmarks while maintaining a relatively low number of additional parameters.

Dataset and Features

The ImageNet Object Localization Challenge has been running from 2010 to 2017 and is currently hosted on Kaggle. As mentioned previously, the goal of my architecture is to classify a 256 by 256 pixel image into a class to see if we can perform as close to other notable architectures. The data for training our CNN architecture will be stored in the following directory called "train". In this folder, images for each of the classes labeled n01440764, n01443537, ... will be stored in a folder respective to their label and for interpreting these Id's a txt file called "LOC_sysnset_mapping" where each Id is corresponded to its actual name (ex: fish, dog, cat).

COMPETITIONS

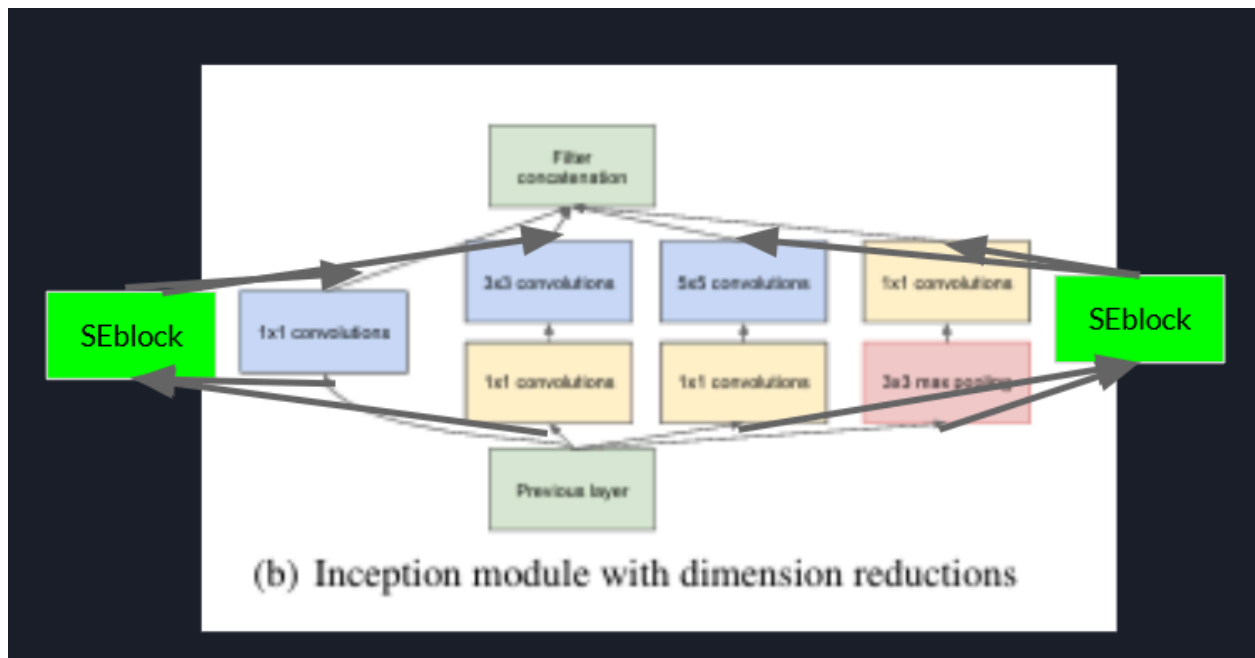
- ▼ dataset ImageNet Object Localization Challenge
 - ▼  ILSVRC
 - ▶  Annotations
 - ▼  Data
 - ▼  CLS-LOC
 - ▶  test
 -  train
 - ▶  val
 - ▼  ImageSets
 - ▼  CLS-LOC
 -  test.txt
 -  train_cls.txt
 -  train_loc.txt
 -  val.txt
 -  LOC_sample_submission.csv
 -  LOC_synset_mapping.txt
 -  LOC_train_solution.csv
 -  LOC_val_solution.csv

Because the dataset is hosted on Kaggle and Kaggle allows up to 30 h's of GPU processing per week, I will work with the datasets on Kaggle

Methods

Because SEnet has the capability of feature recalibration where because of the neural network block the best feature maps become more important, I decided that incorporating SEnet in each individual Inception module by adding a SEnet block to every path inside of it unlike the module which passes an Inception module through a Squeeze and excitation network. So for example taking the Inception module with

reduction I will add a Squeeze and excitation module to the 4 paths



between the filter concatenation and Previous layers. In order to do so I used an implementation of GoogleLeNet and the SEnet from Github and on kaggle we started training. For the Data preprocessing we do the following:

```
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]),
])
```

We Resize to 224 by 224 pixels and then we do RandomHorizontal flip. Then we also convert to a Tensor and normalize the Image. Once we defined our transformation we

then load our training and validation datasets using pytorch:

```
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')
        # print(Allfolders)
        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, pin_memory=True)

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, pin_memory=True)
```

We create an ImageCSVLoader class that reads the dataset provided a csv file of each file and its prediction and the directory the dataset is located in, then we use a data loader to create batches of 256 data instances with their labels. Once we are done loading our dataset then it is time to create the model. Just like previously mentioned we will introduce SE-net Squeeze and Excitation blocks into our GoogLeNet architecture by modifying the Inception block itself by using templates from [maciejbalawejder/Deep-Learning-Collection](#) already implemented in github. Once we have done that in pytorch we start the training phase.

Experiments/Results/Discussion

Once we have done the data preprocessing and created our custom model, it is time to create a summary of our model, which results in the following:

```
=====
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
-----
```

Compared to the original GoogLeNet model, I have approximately 6 million more trainable parameters and in order to combat the problem of vanishing and exploding gradients I have also incorporated the original 2 Auxiliary outputs that output the gradients in the middle of the CNN, and in order to finish the creation of the model I am using the following:

```
criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.SGD(model.parameters(), lr=0.1, momentum=0.9, weight_decay=1e-4)
```

A criterion that calculates the CrossEntropyLoss, an Stochastic gradient descent optimizer with an initial learning rate of 0.1, momentum of 0.9 and weight decay of 0.0001 which was also used by GoogLeNet during training. I decided that because of the constraints of the number of hours you can use a Kaggle notebook with a GPU P100 accelerator I will train the model for 10 epochs. Unfortunately due to Kaggles Timeouts I trained the model for 3 epochs before it timed out. Then I later trained the model again for another 3 epochs resulting in 6 epochs being trained out of the 10 epochs. The Results showed that I have been dealing with the vanishing and exploding gradients problem because my loss function has been fluctuating around 11 with no improvement on the validation set from 0.1% accuracy. If given more GPU power, more epochs, and no timeouts the model would have converged to a better solution.

Conclusion/Future Work

Although the training is incomplete given the constraints on the Kaggle notebook for training with a GPU accelerator and the inactivity time outs, The model's true potential seems to be capable of reaching GoogLeNet's top 1 and top 5 accuracy. Something that might have worked better was having a physical GPU available to train on given the fact that Online GPUs have time constraints. Although I could have used TJ's Jupyterhub clusters to train the model on, The problem with Jupyterhub was the fact that all of the GPU's provided had a CUDA version that was outdated hence not being able to support CUDA acceleration. If given the chance with unlimited space and GPU, This project would have turned out better.

Contributions

Everything Solo (you should consider giving me 1 point for extra credit 🙄)

References/Bibliography

[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-ed0ea3a35ff>

<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.

[maciejbalawejder/Deep-Learning-Collection](#)

[ImageNet Winning CNN Architectures \(ILSVRC\) | Kaggle](#)

Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. (* = equal contribution) **ImageNet Large Scale Visual Recognition Challenge**. *arXiv:1409.0575*, 2014. [paper](#) | [bibtex](#)
[\[1409.0575\] ImageNet Large Scale Visual Recognition Challenge](#)

[ImageNet Benchmark \(Image Classification\) | Papers With Code](#)