Rutgers, The State University of New Jersey

Department of Computer Science

CS512: Data Structures and Algorithms

<u>**Project Report**</u>

Image Similarity Search

Prof. Antonio Miranda

Team Members: -

Mansi Rajesh Borole (mb2208) 218002227

Naga Vamsi Krishna Bulusu (nb847) 214005263

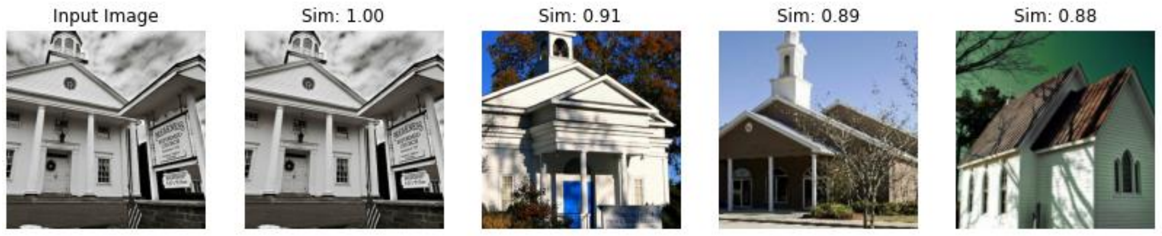
Anjali Menghwani (am3370)

Contents

1 Project Description

In this report we will be discussing our approach for implementing image similarity search using a pretrained Convolutional Neural Network (CNN) model and then using KNN classifier to find similar images.

We are using Imagenette dataset as the input to our model to extract the features' vector which will be used to find more similar images. This is essentially like the '*Google Reverse Image Search*'.



2 Data Description

ImageNetTE (Imagenette) is a small subset of the ImageNet dataset that was created by Jeremy Howard and Rachel Thomas. ImageNet is a large-scale dataset of images that is commonly used for image classification and object recognition tasks. Imagenette is a smaller version of ImageNet that is designed to be more accessible and easier to use for testing and debugging machine learning models.

Imagenette contains a total of 10 classes of images, with each class containing approximately 500 images. The classes in Imagenette are:

- Tench (a type of fish)
- English springer (a breed of dog)
- Cassette player
- Chain saw
- Church
- French horn
- Garbage truck
- Gas pump
- Golf ball
- Parasol

The images in Imagenette are a mix of high-quality professional photographs and lower-quality amateur photographs. The images are 256x256 pixels in size and are in JPEG format. The images are organized into a hierarchy of directories, with each class of images being stored in a separate directory.

Imagenette is often used as a benchmark dataset for evaluating the performance of machine learning models, especially for tasks related to image classification and object recognition. It is also useful for testing and debugging machine learning pipelines and for prototyping new models and approaches.

3 Implementation

3.1 Load Dataset and pre-processing

We load the Imagenette dataset from tensorflow_datasets. We are then resizing the images into desired size (Height and Width) and normalize the pixel values to be between 0 and 1 in the preprocess_image function.

Then we create batches of the training dataset in the get_image_batches function.

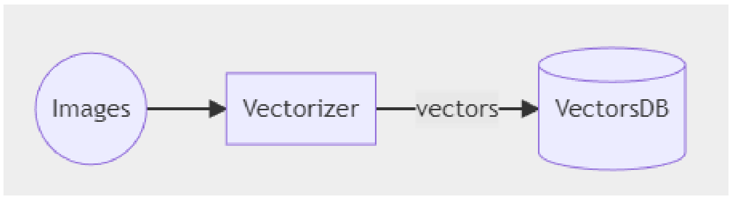
3.2 Feature extraction for training images using CNN

3.2.1 How are features extracted?

In the context of deep learning and computer vision, a convolution is a mathematical operation that combines two sets of data to produce a third set of data. In the case of image processing, the data sets are typically the input image and a set of weights, known as a kernel or filter.

Convolutional neural networks (CNNs) are a type of deep learning model that are particularly well-suited for image classification and other tasks involving visual data. A CNN applies a series of convolutions to an input image to extract features from the image and build a feature map. The convolutions are performed by sliding the kernel over the image and computing the dot product of the entries in the kernel and the image at each position. The resulting feature map is then used by the rest of the network to make a prediction or perform some other task.

Convolutional layers are an important building block of CNNs and are typically stacked together to form the complete network. The use of convolutions allows CNNs to learn features and patterns in the input data directly from the data itself, rather than requiring manual feature engineering by the user. This makes them a powerful tool for tasks such as image classification, object detection, and segmentation.



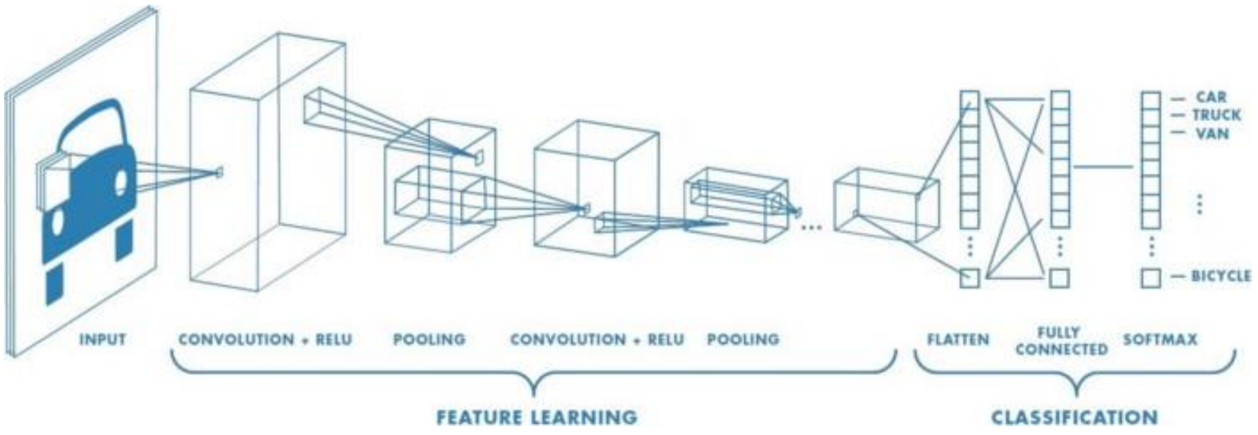
3.2.2 CNN model for Feature Extraction: InceptionResNetV2

InceptionResNetV2 is a convolutional neural network architecture that was introduced in the paper "*Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*" by Szegedy et al. It is an extension of the Inception architecture, which was introduced in the 2014 paper "*Going Deeper with Convolutions*" by Szegedy et al.

InceptionResNetV2 is a neural network architecture that combines the use of inception modules from the Inception architecture with residual connections from the ResNetV2 architecture. The use of both inception modules and residual connections allows the InceptionResNetV2 architecture to achieve improved performance on tasks such as image classification.

The Inception architecture is known for its use of "inception modules," which are blocks of layers that perform convolutions of different sizes on the input data and concatenate the resulting feature maps. The InceptionResNetV2 architecture builds on this by adding residual connections, which are additional connections that bypass one or more layers and allow the network to learn an identity function. This can help the network learn more efficiently and improve its performance on tasks such as image classification.

In summary, InceptionResNetV2 is a type of deep learning network that combines the use of inception modules and residual connections to achieve improved performance on tasks such as image classification. It has been widely used in various computer vision applications and has achieved state-of-the-art performance on several benchmarks.

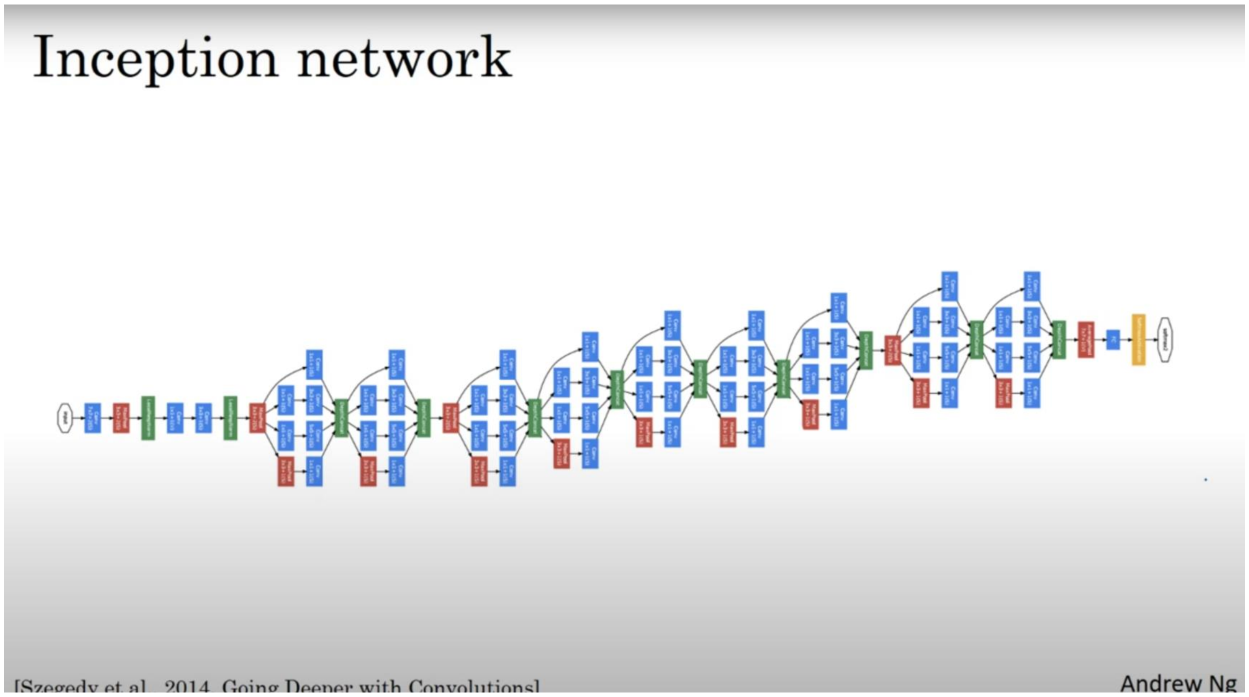


3.2.3 Residual Networks

Residual connections, also known as shortcut connections or skip connections, are additional connections in a neural network that bypass one or more layers and allow the network to learn an identity function. These connections are used in a type of neural network architecture called a residual network, which was introduced in the 2015 paper "Deep Residual Learning for Image Recognition" by He et al.

In a residual network, the bypass connections are added between layers in the network and allow the input data to be passed directly through to the output, in addition to the data that is transformed by the layers in the network. This allows the network to learn an identity function as a baseline and then build on top of it, rather than starting from scratch.

The use of residual connections can help the network learn more efficiently and improve its performance on tasks such as image classification. They also make it easier for the network to learn deep models, which are models with many layers, as they allow the gradients to flow more easily through the network during training. This can help the network learn more effectively and avoid the issue of vanishing gradients, which can occur when training deep models without residual connections.



3.3 Using KNN model to Fit the features into clusters

Once the feature vectors have been extracted, the KNN algorithm can be used to find the nearest neighbors to a given image by calculating the distance between the feature vectors of the images. The distance between feature vectors can be calculated using a variety of distance measures, such as Euclidean distance or cosine similarity. We are using cosine similarity.

To find the k nearest neighbors to a given image, the algorithm sorts the images by distance and selects the k images with the smallest distance. These k images are considered to be the most similar to the given image.

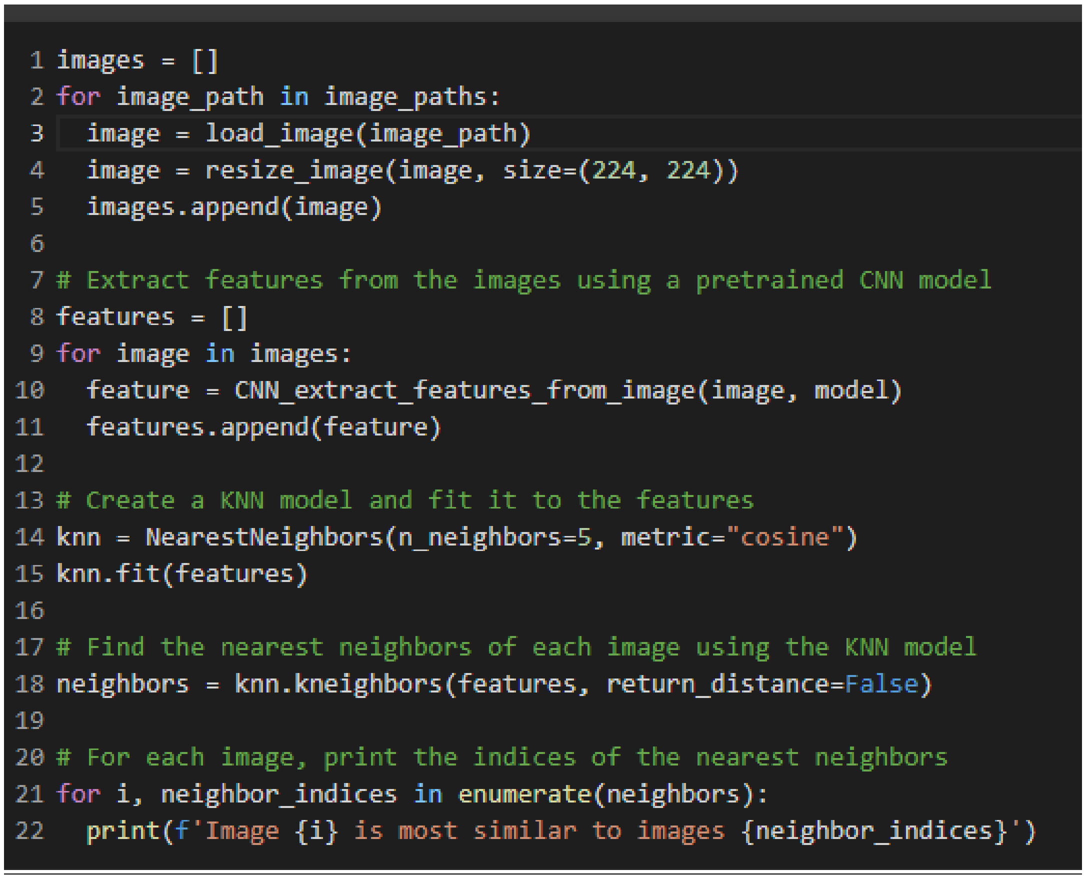
```
knn = NearestNeighbors(n_neighbors=5, metric="cosine")
knn.fit(features)
```

3.4 Testing: Finding similar images given test input

We generate a small batch of images from the testing dataset and then pick a few random input images from it. We then feed the test input to the KNN model and find the top k-similar images.

4 Pseudocode for the Implementation:

This pseudo code loads the images, extracts features from them using a pretrained CNN model, calculates the similarity between the features using cosine similarity, and then sorts the similarities in descending order and prints the top N most similar pairs. You can modify this code to suit your specific needs and use it as a starting point for implementing image similarity in your own project.

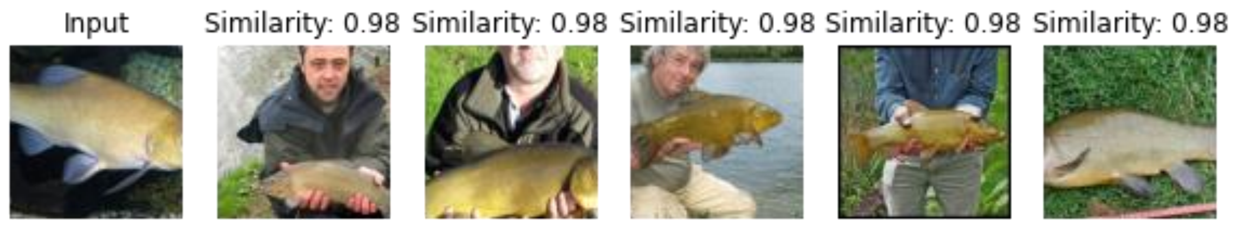


```
images = []
for image_path in image_paths:
  image = load_image(image_path)
  image = resize_image(image, size=(224, 224))
  images.append(image)

# Extract features from the images using a pretrained CNN model
features = []
for image in images:
  feature = CNN_extract_features_from_image(image, model)
  features.append(feature)

# Create a KNN model and fit it to the features
knn = NearestNeighbors(n_neighbors=5, metric="cosine")
knn.fit(features)

# Find the nearest neighbors of each image using the KNN model
neighbors = knn.kneighbors(features, return_distance=False)

# For each image, print the indices of the nearest neighbors
for i, neighbor_indices in enumerate(neighbors):
  print(f'Image {i} is most similar to images {neighbor_indices}')
```

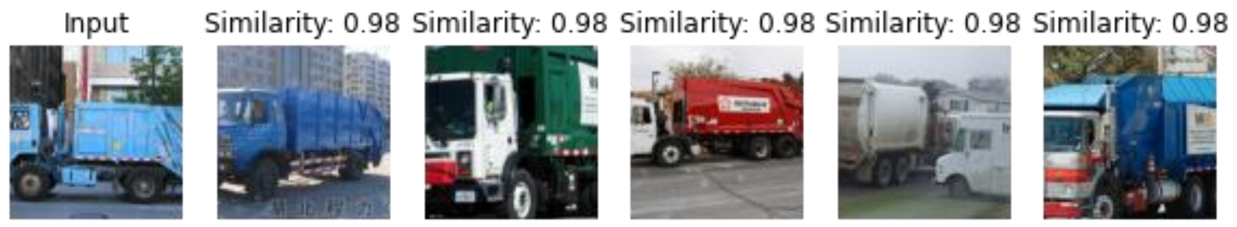
5 Outputs

5.1 Single image input

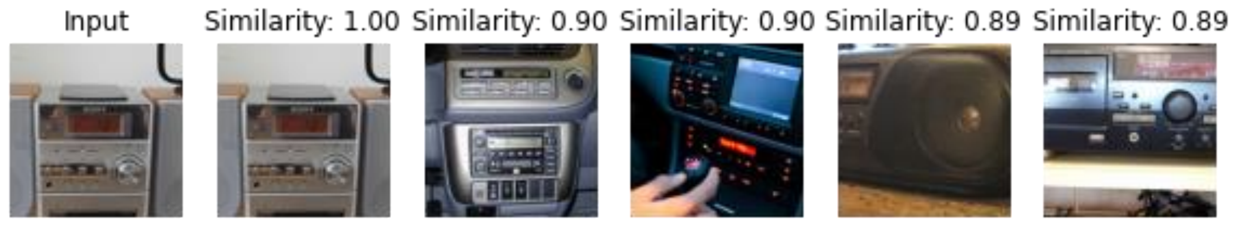
Test case 1:



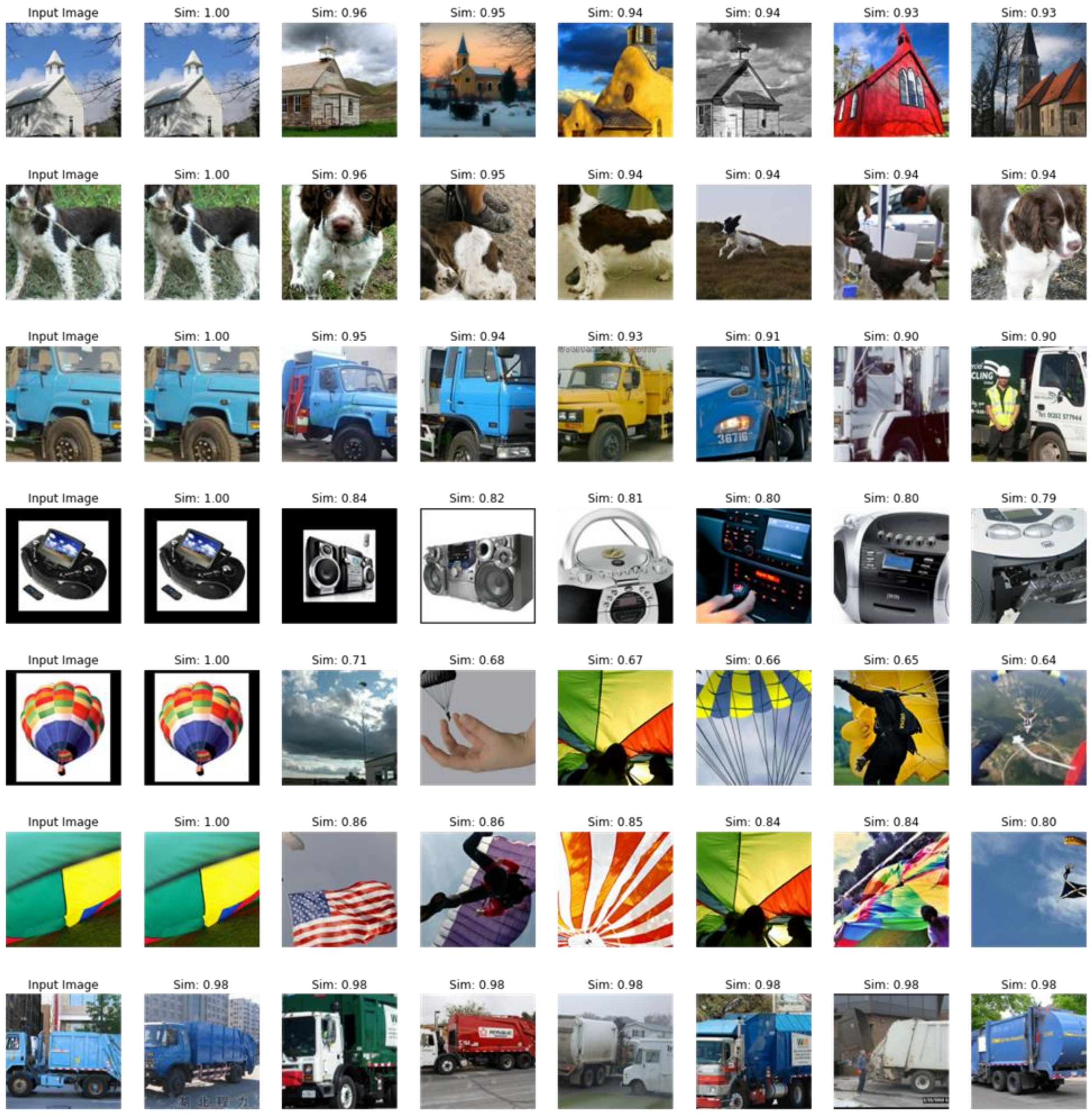
Test case 2:



Test case 3:



5.2 Multiple inputs



6 Complexity

6.1 CNN Model:

In general, the complexity of a CNN model is determined by the number of computations required to perform a forward pass through the network, which is a measure of the time and resources needed to make a prediction. The complexity of a CNN model can be affected by the number of layers, the number of filters in each layer, the size of the filters, the stride, and the padding.

Inception-ResNet-v2 is a relatively deep and complex model, with a total of 572 layers and approximately 55 million parameters. This makes it more computationally intensive than some other CNN models, but also allows it to achieve higher accuracy on image classification tasks.

6.2 KNN Clustering

The k-nearest neighbors (KNN) clustering algorithm to group similar images together, ahs a complexity of **O(n log n)**. Alternatively, we could use a pretrained model specifically designed for image similarity, such as a siamese network, which would have a different complexity profile.

7 Citations

[1] Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017). **Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning**. In AAAI Conference on Artificial Intelligence. https://doi.org/10.1609/aaai.v31i1.11231

[2] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). **Going deeper with convolutions.** In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9) https://doi.org/10.1109/CVPR.2015.7298594