T.R.

ÜSKÜDAR UNIVERSITY

INSTITUTE OF SCIENCE

COMPUTER ENGINEERING PROGRAM

MASTER'S THESIS

HOGNET: TOWARD A LIGHTWEIGHT DEEP LEARNING MODEL WITH APPLICATIONS FOR VISUAL PLACE RECOGNITION

IMAD ODEH

Thesis Advisor

Doç. Dr. Abdul Hafiz ABDULHAFIZ

İSTANBUL - 2024

T.R.

ÜSKÜDAR UNIVERSITY

INSTITUTE OF SCIENCE

COMPUTER ENGINEERING PROGRAM

MASTER'S THESIS

HOGNET: TOWARD A LIGHTWEIGHT DEEP LEARNING MODEL WITH APPLICATIONS FOR VISUAL PLACE RECOGNITION

IMAD ODEH

Thesis Advisor

Doç. Dr. Abdul Hafiz ABDULHAFIZ

İSTANBUL - 2024

ABSTRACT

In this research, we present HOGNet, a lightweight deep-learning model built from scratch to compete with well-known pre-trained networks such as VGG16. HOGNet **prioritizes** a lightweight architecture for **efficient** computational performance while maintaining **competitive** accuracy. Firstly, we will show how we built HOGNet, including its architectural design and training stage. Additionally, we present experimental results demonstrating that HOGNet achieves significant results in the Visual Place Recognition (VPR) task. In general, pre-trained networks are widely used as backbones in VPR systems. In this research, we aimed to develop an efficient Convolutional Neural Network (CNN) model that could be used as the backbone or directly used in classification tasks. By leveraging the Histogram of Oriented Gradients (HOG), which is a good features descriptor, we developed our deep learning model. So, we call our network model as HOGNet. Our model gave good accuracy for VPR task on Nordland dataset with low computational time compared to VGG16 as reference. The accuracy of classifier affects clearly with the features of selected layer within same deep learning model.

Keywords: HOG, Transfer Learning, Visual Place Recognition.

THANKS TO

First, I want to express my gratitude to Doç. Dr. Abdul Hafiz ABDULHAFIZ for his supervision of my thesis, his creative ideas, and his guidance in keeping me on the right track. I also extend my respect to my professors at Üsküdar University, whose courses, particularly in 'Computer Vision' and 'Digital Image Processing,' were essential in laying the foundation for my thesis work. Additionally, the 'Research Methods and Scientific Ethics' course was invaluable for the writing process. I extend my gratitude to the Thesis Jury: Dr. Ahmet Şenol, who taught me three courses, including "Linux Essentials for Cybersecurity," which was useful for my work with Google Colab. I also thank Dr. Ali Karah bash for his valuable notes that significantly contributed to the extension of this thesis. I thank my parents for their unwavering support in every aspect of my life. Finally, I am deeply grateful to my wife, who helped me cope with all the pressures throughout this journey.

FORM OF DECLARATION

I hereby certify, that I obtained all the information and documents in this study within the framework of academic rules, presented all visual and written information and results in accordance with scientific ethical rules, did not falsify the data I used, referred to the sources I used in accordance with scientific norms, that my thesis was original except in the cases cited, produced by me and written in accordance with the Thesis Writing Guide of Master Degree Program of Computer Engineering in Graduate school of science in Üsküdar University.

01/07/2024

Imad Odeh

CONTENTS

INDEX of FIGURES

1. INTRODUCTION

Visual Place Recognition (VPR) is a process that aims to predict the place of a photo (it is called a query image); it depends on the similarity between the visual features of the query image and database (reference) images. VPR systems have an important role for robots during navigation (Abdul Hafez et al., 2022). Another application could be for Tourism applications. The VPR systems are distinguished from each other by the method of extracting the features and the distance function that calculates the similarity of features between query and reference images. There is a related concept called 'Image geo-localization' which can be defined as predicting the GPS coordinates of a query image using a geo-tagged image dataset (Aljuaidi et al., 2019).

There are a lot of VPR methods, which could be classified as follows:

- Handcrafted features VPR methods like: VLAD (Vector of Locally Aggregated Descriptors), BoW (Bag-of-Words) and CoHOG (Zaffar et al., 2020) which uses image entropy and HOG to extract and represent the regions of interest (ROI) in image (Abdul Hafez et al., 2022).
- Deep Learning-based VPR Methods like: NetVlad (Arandjelović et al., 2015), SqueezeNetVlad (Pal et al., 2023), Patch-NetVlad (Hausler et al., 2021), also NetVlad could be a part of whole system (Keetha et al., 2021), SFRS (Self-supervising Fine-grained Region Similarities) (Ge et al., 2020), CosPlace (Berton et al., 2022), MixVPR (Ali-bey et al., 2023) and EigenPlace (Berton et al., 2023). They depend on backbones that are actually pre-trained deep learning models to train and extract the best features. The same method could be used with different backbones. Eigen Places is VPR method represents state of the art (SOTA) on most datasets (Berton et al., 2023). The code is available in GitHub

For extracting features, almost all VPR systems depend on one of the pretrained CNNs, like VGG16 and ResNet50 (He et al., 2015). Examples of distance functions are Euclidean and Cos distances. There are many challenges for VPR systems due to changes in query image compared to reference images, like different viewpoints or illumination. There are also changeable things in the image, like cars and people. This research proposes HOGNet, a lightweight Convolutional Neural Network (CNN) (O'Shea & Nash, 2015) model designed for Visual Place Recognition (VPR) tasks and potentially applicable to other computer vision applications. We utilized the CIFAR-10 dataset to train a simple CNN (Gu et al., 2018) to predict handcrafted (HOG) features from images. HOG features have wide ranges of applications in image processing and computer vision fields. This resulted in a model that produces valuable features at various layers. The second stage is for discovering the best features in different layers in HOGNet. Getting good features is very important for getting good accuracy in classification or recognition tasks. The final stage is to use HOGNet in VPR task. We do fine-tuning on Nordland dataset (Olid et al., 2018), which contains different images for each place in different seasons: Summer, Fall, Winter and Spring. Our experiments demonstrated that HOGNet achieved good results with low computational time. HOGNet is a promising approach that begins with HOG features to build a pre-trained network. This network can be effectively used for transfer learning in various computer vision tasks. The contribution of this thesis is described as follows:

1. A novel deep learning model that transfers the knowledge from the HOG handcrafted features to the deep learning architecture.

2. Empirical analysis and comparison with the state-of-the-art models.

Computer vision concerns about DORI concept (Detection, Observation, Recognition and Identification):

1. **Detection** level: allows for reliable and easy determination of whether an object is present.
2. **Observation** level: gives characteristic details of an object.
3. **Recognition** level: determines with a high degree of certainty whether an object is similar to something that has been seen before.
4. **Identification** level: enables the identity of an object beyond a reasonable doubt.

Together, detection and recognition form a powerful combination that allows computer vision systems to perceive and interpret the visual world effectively, enabling a wide range of applications from simple monitoring to complex decision-making tasks. For example, for face emotions recognition, we should first detect the face then recognize the emotion in the face. Our thesis topic is considered in recognition level.

The model is tested and compared with efficient deep models like VGG16, it has shown superior performance in terms of time and space complexity. The proposal is promising and has several potential applications due to its lightweight and efficiency. The remainder of the thesis is organized as follows: The next section presents a literature review of the related works. Section III presents our proposed model and discusses the potential applications. Section IV presents our experimental analysis.

1.1 CIFAR-10 dataset

CIFAR-10 is abbreviation for Canadian Institute for Advanced Research 10. It is widely used in machine learning and computer vision applications. The dataset is divided into 10 mutually exclusive classes, each containing 6,000 images: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship and Truck. Dataset is splitted to 50,000 images as Training set, and 10,000 images for test set.

In developing HOGNet, we leveraged the CIFAR-10 dataset, despite its primary use for classification tasks. This choice was driven by the need to train a CNN for predicting HOG features, rather than the specific image categories within CIFAR-10. Ultimately, the content of the dataset is less important than its size for this training stage.

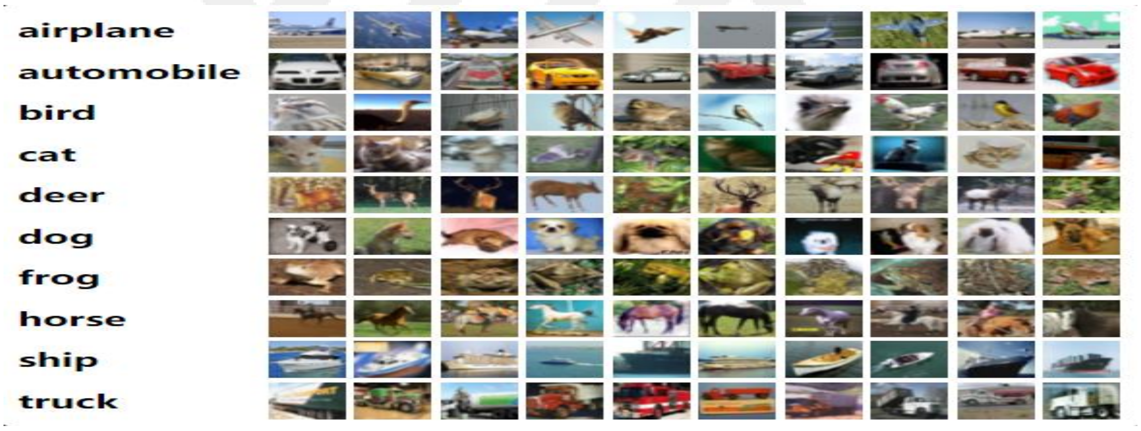

Figure 1: CIFAR-10 classes

1.2 Nordland dataset

It captures a train journey along the Nordland Railway in Norway. The dataset consists of images extracted from the train journey videos. It is widely used in the field of VPR. The journey is recorded in four different seasons: spring, summer, fall, and winter. We utilized Nordland dataset to test accuracy of HOGNet. The challenge in this dataset is to recognize the same place in different seasons.

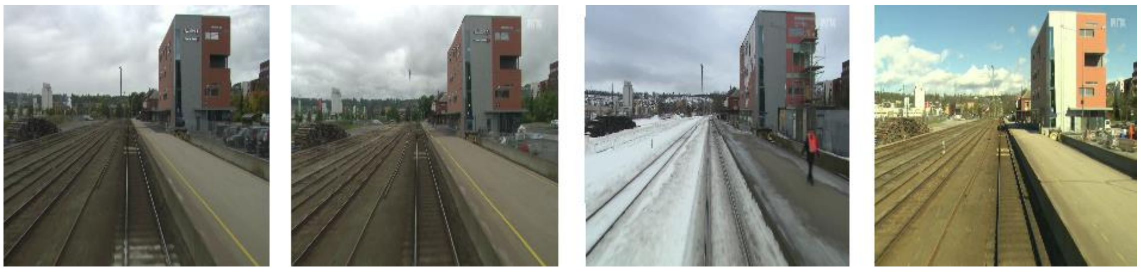
Figure 2: Same Place in different seasons

2. LITERATURE REVIEW

2.1. Conventional Neural Network (CNN)

The field of machine learning has taken a dramatic twist in recent times (O'Shea & Nash, 2015), with the rise of the Artificial Neural Network (ANN). These biologically inspired computational models can far exceed the performance of previous forms of artificial intelligence in common machine learning tasks. One of the most impressive forms of ANN architecture is that of the Convolutional Neural Network (CNN). CNNs are primarily used to solve difficult image-driven pattern recognition tasks and with their precise yet simple architecture, offers a simplified method of getting started with ANNs.

The main point is preprocessing images before training the ANN itself. The basic functionality of the example CNN above can be broken down into four key areas.

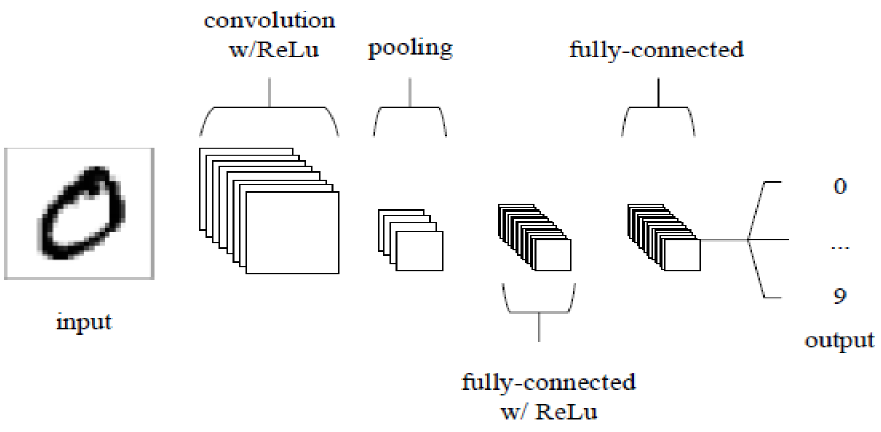

Figure 3: Simple CNN Architecture

1. As found in other forms of ANN, the **input layer** holds the raw pixel values of the image. Each pixel is represented as a numerical value, with color images typically having three channels (Red, Green, Blue).
2. The **convolutional layer** performs the core operation of the CNN. It applies convolution operations to the input by sliding a set of filters (kernels) over the input image. Each filter computes a scalar product between its weights and the local regions of the input, producing an activation map. The convolutional layer detects local features such as edges, textures, and patterns.
The **rectified linear unit** Introduces non-linearity into the network, allowing it to learn complex patterns. ReLU (Rectified Linear Unit) sets negative values to zero while

keeping positive values unchanged. Other activation functions like Leaky ReLU or tanh are also used.
3. The pooling layer performs down-sampling on the activation maps to reduce their spatial dimensions and the number of parameters, making the network more computationally efficient and less prone to overfitting. The most common types are:

- Max Pooling: Selects the maximum value from each patch of the feature map.
- Average Pooling: Computes the average value of each patch of the feature map.

4. The fully connected layers (Dense Layers): These layers are similar to those in standard Artificial Neural Networks (ANNs). Each neuron in a fully connected layer is connected to every neuron in the previous layer, allowing the network to combine features learned in previous layers to make predictions. ReLU activation functions are often used between fully connected layers to improve performance.
5. The Global Average Pooling Layers: These are special types of pooling layers applied globally across the entire spatial dimensions of the feature maps:

- Global Average Pooling (GAP): Reduces each feature map to a single value by averaging all the values in that feature map.
- Global Max Pooling (GMP): Reduces each feature map to a single value by taking the maximum value in that feature map6. The Global Max Pooling layer, very close to Global Average Pooling but instead of average it takes the maximum value from all the values in that feature map.

7. Batch Normalization Improves the training process of deep neural networks. It normalizes the activations of each layer, making them less sensitive to initialization values and allowing the network to learn faster.
8. The Dropout is a regularization technique used to prevent overfitting. During training, it randomly drops out (sets to zero) a certain percentage of neurons in the layer. This forces the network to learn redundant representations and makes it more robust.

2.2. Pre-trained Networks and Transfer Learning

Convolutional Neural Network (CNN) is a powerful deep learning model that is used in image recognition and classification tasks (Koyuncu & Koyuncu, 2019; NOGAY, 2018; Wang, 2017). The ability of CNNs to automatically learn features from raw data has led to advancements in these tasks. Now there are a lot of pre-trained CNN that have specific architectures of layers and are trained in huge datasets, like ImageNet dataset. Historically, pre-trained networks like VGG, ResNet(He et al., 2015), and MobileNet (Howard et al., 2017) were developed for different applications that focused on increasing accuracy or decreasing computational time, especially for limited resources devices. Pre-trained networks are widely used and are utilized and accessible in famous deep learning frameworks like PyTorch and Tensorflow. Transfer learning (**knowledge transfer**) in deep learning field, focus to using pre-trained networks for training in new datasets to benefit from their ability to get very good features (Pan & Yang, 2010). This reduces training time and improves model performance. It is often very difficult, and sometimes impossible, to collect a large dataset for training models from scratch.

Without a large dataset, achieving high accuracy is challenging. This is where the importance of transfer learning becomes evident, as it reduces effort and increases accuracy. In transfer learning, it could take a pre-trained network as it is or some layers. Also, it could add additional layers to adapt to the new task. Some layers can be frozen for training; in this case, they are considered as features extractors or include all layers in training. In general, using transfer learning techniques is better than training from scratch which may take a longer time for training and give less accuracy.

We can see the supported models in Pytorch in this link:

https://pytorch.org/vision/0.18/models.html

For Tensorflow, we can look at:

https://www.tensorflow.org/api_docs/python/tf/keras/applications

2.1.1 VGG16

One notable advancement is the VGGNet architecture by Simonyan and Zisserman (2015), which introduced deeper networks with smaller filter sizes. VGGNet demonstrated that increasing the depth of the network leads to improved accuracy in image classification tasks. Its architecture has inspired many other deep learning models and is still commonly used as a benchmark in research and practical applications.

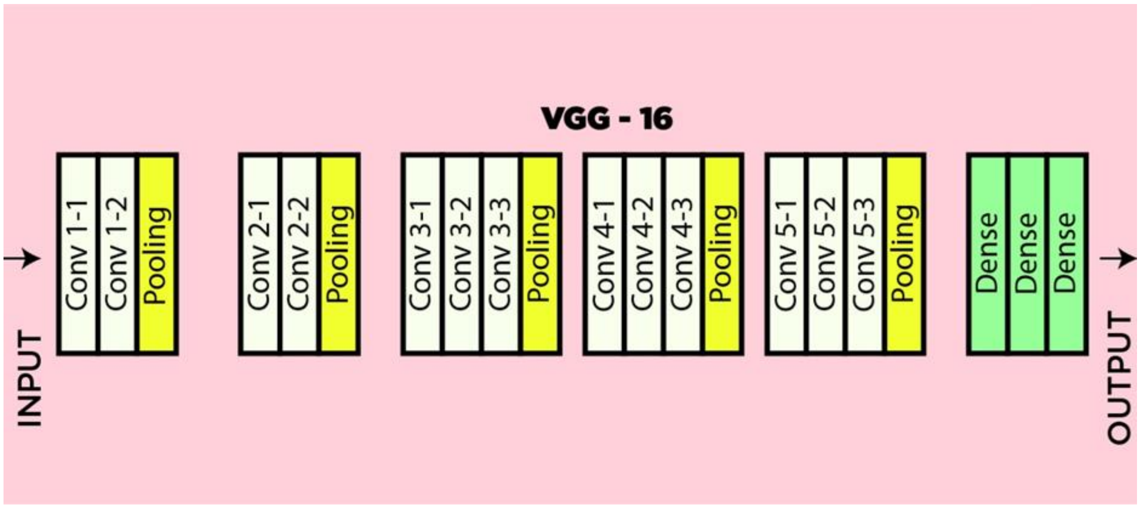

Figure 4: Architecture of VGG16

2.1.2 Resnet101

ResNet architecture proposed by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. (2016). ResNet addressed the issue of vanishing gradients in deep networks by introducing residual connections that enable the training of extremely deep models. ResNet achieved remarkable performance on various image recognition benchmarks and spurred the development of even deeper networks. Its architecture has significantly influenced the design of many subsequent deep learning models.

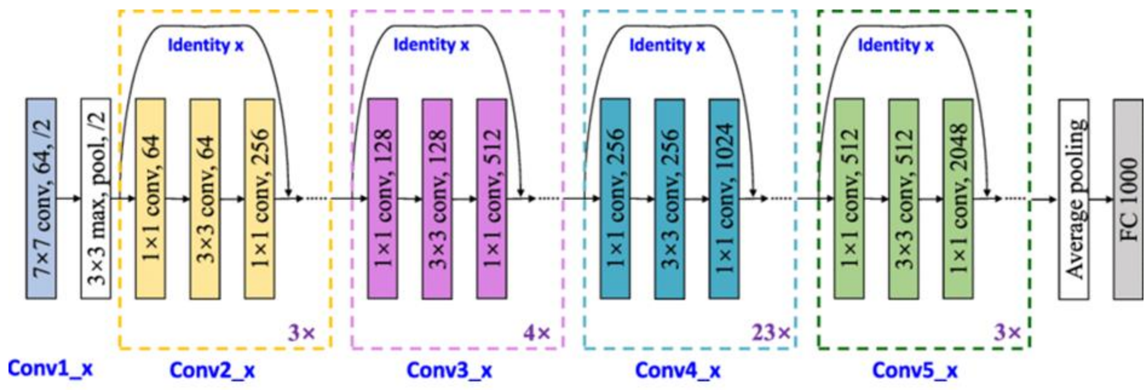

Figure 5: Architecture of Resnet101

2.1.3 DenseNet201

DenseNet (Huang et al., 2017) by Huang et al. (2017) emphasized densely connected layers, allowing information flow across different network depths and reducing the vanishing gradient problem. It is part of the DenseNet (Densely Connected Convolutional Networks) family.

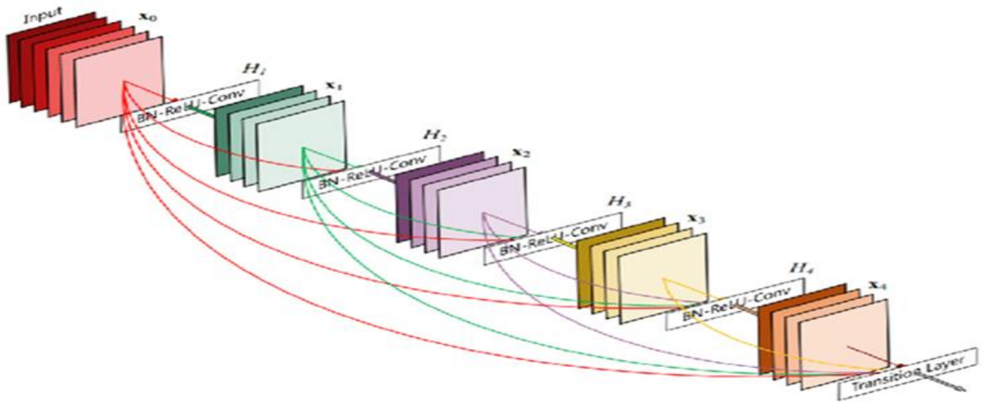

Figure 6: Architecture of DenseNet201

2.1.4 MobileNet

MobileNet (Howard et al., 2017), proposed by Howard et al. (2017), introduced efficient **Depthwise** separable convolutions to reduce model size and computational requirements while maintaining accuracy. MobileNet is an efficient model for mobile and embedded vision applications.

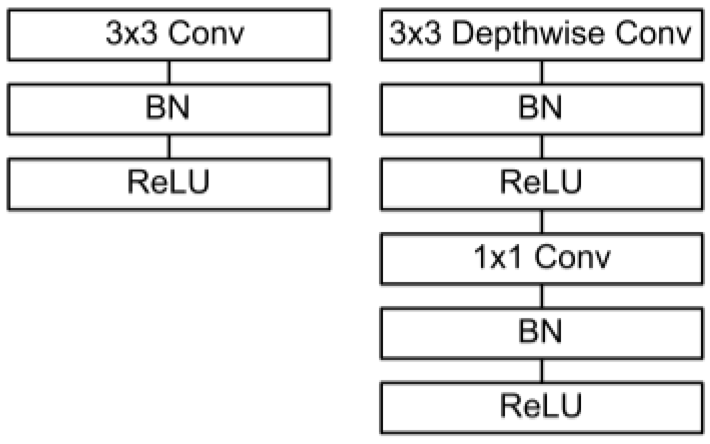

Figure 7: Comparing traditional conventional layer in left and Depthwise conventional layer in right.

2.2 Histogram of Oriented Gradients (HOG)

Histogram of Oriented Gradients, also known as HOG, is a feature descriptor like the Canny Edge Detector or SIFT (Scale Invariant and Feature Transform). It is used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in the localized portion of an image. The HOG descriptor focuses on the structure or shape of an object. It is better than any edge descriptor as it uses magnitude as well as angle of the gradient to compute the features. HOG involves dividing the image into small regions (cells) and calculating the distribution of oriented gradients within each cell. This distribution is then normalized to account for illumination variations. We can get different HOG features for the same image according to parameters that we choose. There are a lot of applications and tasks depending on HOG features like human detection (Dalal & Triggs, 2005) visual place recognition (Zaffar et al., 2020) or fault diagnosis (Xu et al., 2022). The main steps for computing HOG features from an image are:

- Compute the gradient magnitude and angle:

$$Gx(r, c) = I(r, c + 1) - I(r, c - 1) \quad (1)$$

$$Gy(r, c) = I(r - 1, c) - I(r + 1, c) \quad (2)$$

where r,c refer to rows and columns respectively

$$\text{Magnitude}(\mu) = \sqrt{Gx^2 + Gy^2} \quad (3)$$

$$\text{Angel}(\theta) = |\tan^{-1}(Gy / Gx)| \quad (4)$$

- Calculate Histogram of Gradient for cell
- Normalize the histogram for Blocks
- Collect all HOG Features

A famous library ‘skimage‘ that could be used to calculate hog directly.

```
from skimage.feature import hog
```

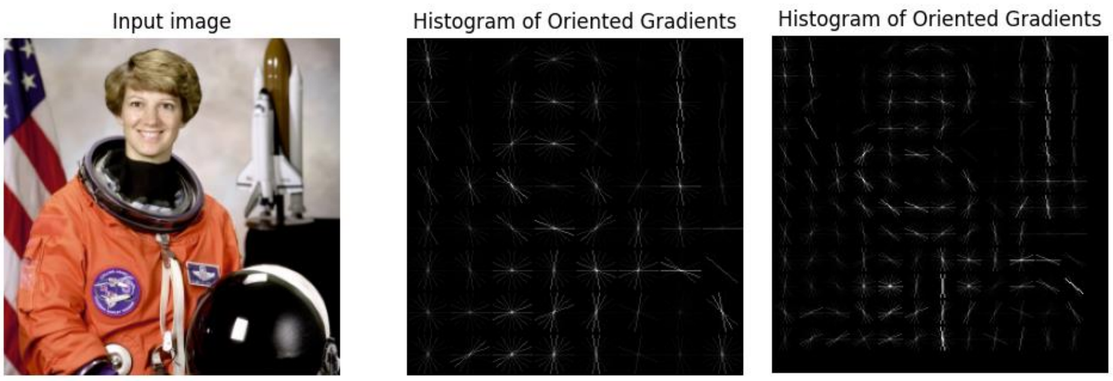
Input image
Histogram of Oriented Gradients
Histogram of Oriented Gradients

Figure 8: Different HOG features for same image if some parameters are changed like pixels per cell or cells per block

2.3 Methods based on Traditional features for VPR

There are many VPR methods that depend on handcrafted features like SIFT or HOG with other techniques like VLAD to find the closest image in reference (database) images from query image. One of these methods begins by extracting SIFT features from the image, followed by calculating VLAD, which represents the image as a single fixed-size vector using K-means clustering. The similarity grouping is then performed using distance measurements, such as the Euclidean distance (Aljuaidi et al., 2019).

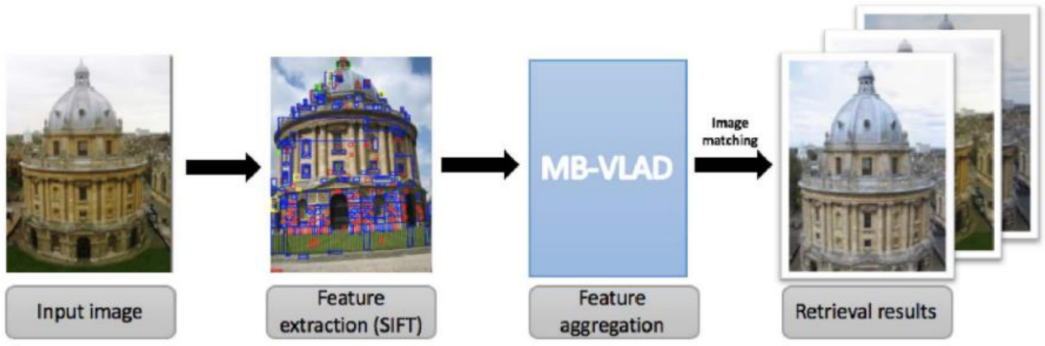

Figure 9: Example of VPR system that depends on SIFT (handcrafted features)

2.4 Deep learning methods to VPR

Deep learning-based VPR methods rely on pre-trained networks like VGG16 or ResNet50 to learn and extract features. Subsequently, techniques such as PCA or Eigen (Berton et al., 2023) decomposition is applied to distinguish or classify these features. All images representing the same location might be naively treated as a unique class by a classifier. However, these images might be taken by cameras pointing in different directions, thus capturing different scenes. Techniques like Eigen can reduce disruption or misleading information. Additionally, there are deep learning methods that employ self-supervised learning to improve VPR performance (Musallam & Lu, 2024.), (Ge et al., 2020). To see the importance of choosing a deep pre-trained network, we can use the following code to evaluate different VPR methods: The full code here. Of course, there are some packages that should be installed before, like faiss and einops.

Of course, there are some packages that should be installed before like faiss and einops.

For faiss:

```
pip install faiss-gpu
```
or
```
pip install faiss
```

For einops:
```
pip install einops
```

Let’s first remember Recall metric used to evaluate the performance of a classification model. Mathematically, recall is defined as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Recall@K is a metric used in ranking and recommendation systems to evaluate how well a model retrieves relevant items within the top K results. For example, Recall@5 concerns the top 5 predictions, if one of them is true that means it is considered as True Positive.

If we use VGG16 as backbone and Nordland as dataset (Summer as database images and Spring as query images), we will get these results:

- NetVlad method with descriptors dimension = 4096, we got these results of Recall values:

R@1: 49.3, R@5: 66.2, R@10: 73.1, R@20: 79.9

- SFRS method (Self-supervising Fine-grained Region Similarities) with descriptors dimension = 4096, we got these results of Recall values:

R@1: 57.1, R@5: 73.0, R@10: 79.6, R@20: 85.0

- CosPlace method with descriptors dimension = 512, we got these results of Recall values:

R@1: 85.2, R@5: 93.6, R@10: 96.0, R@20: 97.6

• EigenPlace method, with descriptors dimension = 512, we got these results of Recall values:

R@1: 86.2, R@5: 93.9, R@10: 96.1, R@20: 97.9

For two best results, if we replaced VGG16 with ResNet50, we would get these results:

• CosPlace method with descriptors dimension = 512, we got these results of Recall values:

R@1: 69.4, R@5: 83.1, R@10: 87.7, R@20: 91.3

• EigenPlace method, with descriptors dimension = 512, we got these results of Recall values:

R@1: 85.3, R@5: 94.0, R@10: 96.1, R@20: 97.4

So, the accuracy is affected by the choice of backbone for the same VPR method.

This motivates our research to explore deep pre-trained networks.

the following as example of states above:

```
python3 path_to/main.py --method=sfrs --backbone=VGG16 --descriptors_dimension=4096 \

   --database_folder=path_to_database_set \

   --queries_folder=path_to_ queries_set
```

2.4.1 Siamese and triplet networks

When we talk about VPR methods, we should also talk about Siamese networks and triplet networks, which are a type of deep learning architecture designed for comparing similar or dissimilar inputs. They consist of two or more identical sub-networks that share weights. These sub-networks process two input images, and the resulting feature vectors are compared to determine similarity. We can also use pre-trained models in these networks. One of the Pros of Siamese neural network (SNN) is Giving a few images per class is sufficient for Siamese networks to recognize those images in the future with the aid of one-shot learning. But comparing with normal networks it needs more training time; Since SNNs involves learning from quadratic pairs (to see all information available) they're slower than the normal classification type of learning (point-wise learning) (Bromley et al., 1994). Triplet Networks consist of three identical sub-networks (sharing the same weights) that process three input images: an anchor, a positive example, and a negative example. It trains the model to ensure that the distance between the anchor and the positive example is less than the distance between the anchor and the negative example by a margin.

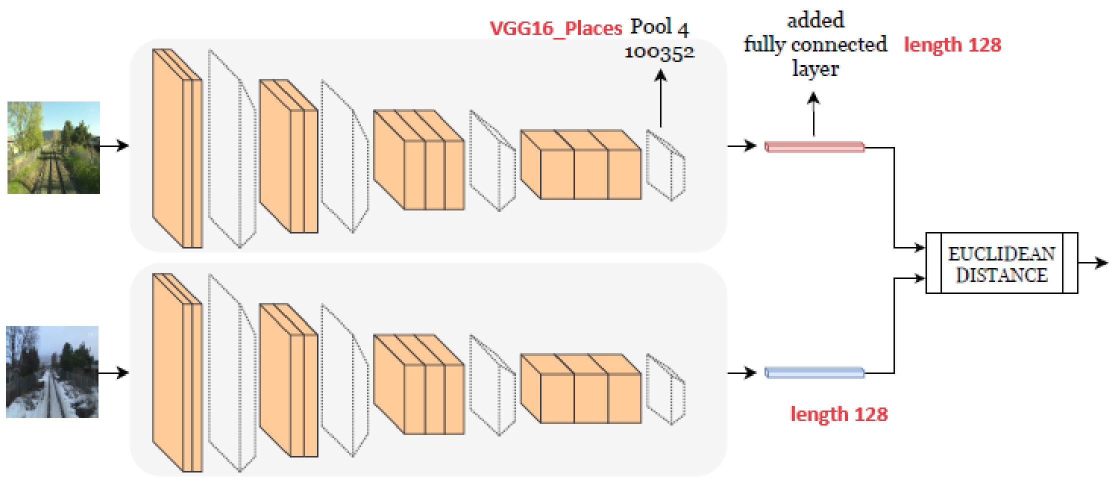

Figure 10: Example of Siamese network depends on VGG16 and uses Euclidean distance to compute similarity between images

3. MATERIALS & METHODOLOGY

Firstly, we did some experiments for transfer learning we did in Anaconda as package manager, Tensorflow as deep learning framework and Spyder as Integrated Development Environment (IDE). Second stage, we built our HOGNet model which is very essential in our research. We use Google Colab Pro to access high-performance computing capabilities. Third stage, we try to discover the best features in different layers in HOGNet model. Finally, we use HOGNet model to find his performance on VPR task. In all experiments, we retain only the **best** version of the model throughout all epochs in TensorFlow or PyTorch.

3.1 Transfer Learning

In the next experiments, we just tried to study the importance of transfer learning with pre-trained networks compared to training from scratch. The task is related to hand gestures classification. We chose 5 classes from two datasets. The first dataset contains images of hand signs of 1 to 6. All images are in jpg format. The images are taken against a clear background. Total Number of Images: 5080

https://www.kaggle.com/datasets/adeshdalvi41/hand-signs

For the second dataset, images are of project owner's family and friends and contain different hand gestures under 8 categories. The images are 207 x 207.

https://www.kaggle.com/datasets/ritikagiridhar/2000-hand-gestures

So, for each class (label) I used 116 images for the training set and 43 images for test set, all of them are collected from both two datasets. We tried to unify conditions to compare fairly between different types of CNN (from scratch, VGG16, ResNet101 and DenseNet201). So, in all cases, we compare results after 50 epochs.

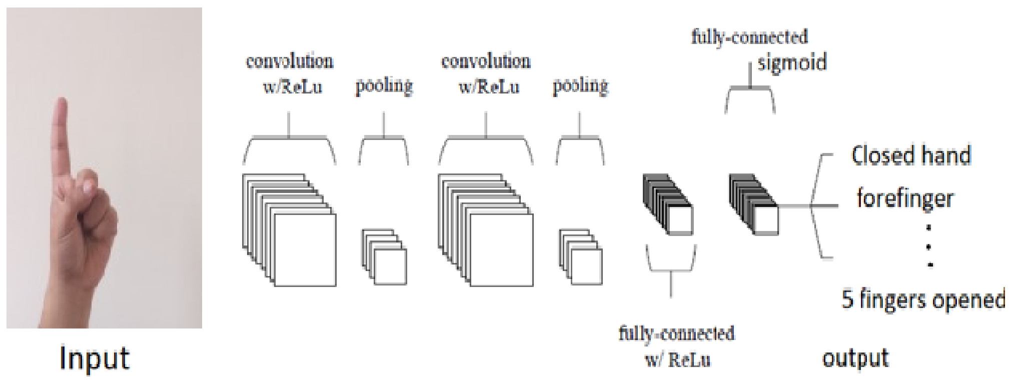

Figure 11: Training CNN from scratch

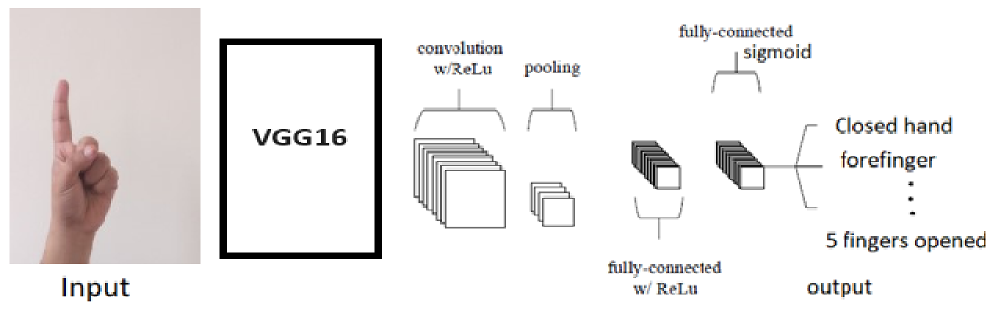

Figure 12: Training VGG16 (frozen layers) followed with additonal layers

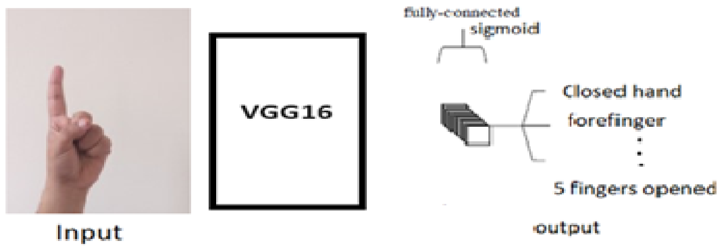

Figure 13: Training VGG16 (frozen layers) followed with classification (fully connected) layer

These tables summarize the results:

	CNN From scratch	VGG16 Additional layers	VGG16 + Train last layer	VGG16 + Train all layers
Validation Accuracy	0.4465	**0.6326**	0. 5907	0.2000

	ResNet101 + Additional layers	ResNet101 + Train last layer	ResNet101 + Train all layers
Validation Accuracy	**0.6419**	**0.6326**	0.5349

	DenseNet201 + Additional layers	DenseNet201 + Train last layer	DenseNet201 + Train all layers
Validation Accuracy	0.4605	0.4605	**0.6047**

As we can see in general, results gotten from transfer learning leveraging VGG16, ResNet101 or DenseNet201 are better than training from scratch. The best results were ResNet101 (frozen layers) followed with additional layers that trained to adapt the task.

3.2 The Proposed HOGNet Model

Google Colab is a free Jupyter notebook environment that you can access through web browser. It lets developers run Python code without needing to install anything on their own computer. This makes it a popular tool for machine learning, data science, and education purposes. Users can easily save and load files from their Google Drive, making it convenient to manage datasets and code. While the free tier offers substantial resources, Colab Pro and Pro+ provide enhanced features like more powerful GPUs, longer runtimes, and more memory for a monthly fee. Google Colab's runtime environment is based on Ubuntu Linux. So, maybe sometimes developers have to execute command line statements to install some packages or check existing resources.

Our experiments are done in Google Colab Pro to access high-performance computing capabilities with multiple GPUs. We used PyTorch as our deep learning framework, benefiting from its advantages in debugging internal instructions. The main idea for HOGNet is to train simple CNN on a large dataset to predict HOG vectors. This involves preparing our datasets such that the input consists of arrays of images and the output is the HOG vector for each image, framing the task as a regression problem. We selected the CIFAR-10 dataset, which contains 60,000 images of 32x32 pixels. In PyTorch there is already dataset **torchvision.datasets.CIFAR10**, we override it to make the target as HOG vector. As note, building HOGNet requires a large amount of RAM; otherwise, the code may crash during execution. We should mention also that small datasets are not enough to get good results according to our experiments.

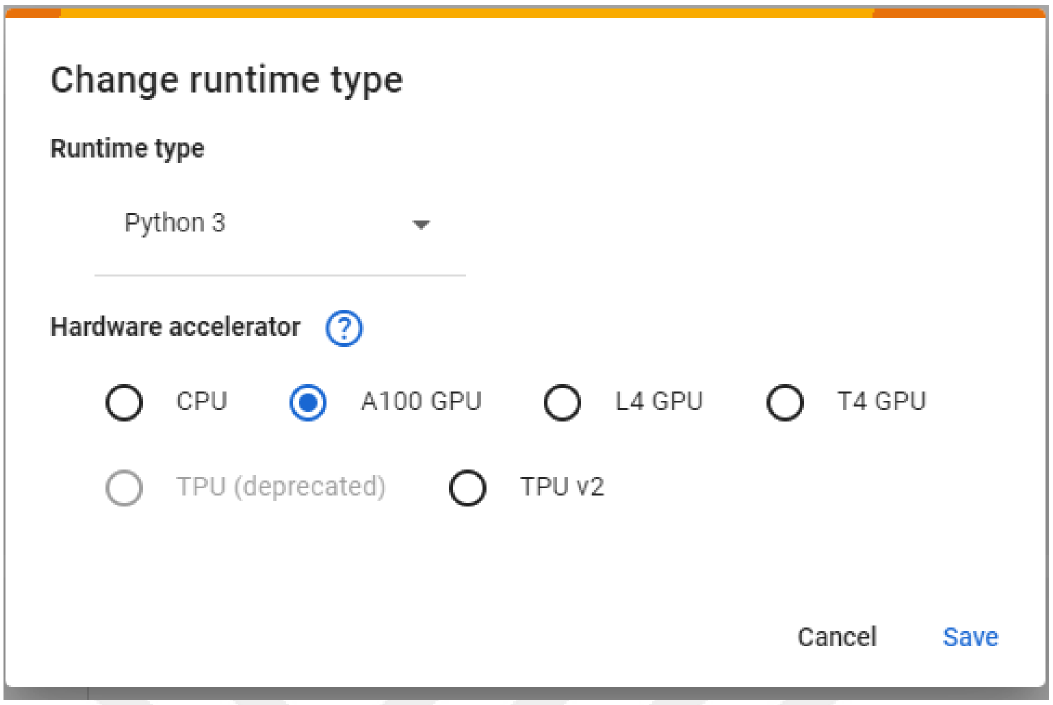

Figure 14: GPU types that appears in Google Colab Pro

The most powerful GPUs are the A100 and L4 GPUs. While any GPU is better than a CPU for deep learning tasks, the free tier of Google Colab doesn't always provide access to even the least powerful GPU. In such cases, Google Colab will automatically switch to using the CPU, making it very challenging to train on large datasets.

3.2.1 HogNet_2Conv_2Pool_441

We call this model HogNet_2Conv_2Pool_441 because it predicts HOG vector with length 441 and its architecture consists basically of 2 Convolutional layers and 2 Pool layers, to distinguish with other HogNet_3Conv_3Pool_441 or HogNet_3Conv_3Pool_1764, which predicts HOG vector with length 1764. For HogNet_2Conv_2Pool_441 we made transformation for input images to resize them to 224x244 pixels. So, according to this input size and these parameters of hog function: **orientations= 9, pixels per cell=(32, 32), cells per block=(1, 1)**; the length of output vector will be 441. This is the architecture of HogNet 441:

```
conv1: Conv2d(3, 6, kernel size=(5, 5), stride=(1, 1))
pool1: MaxPool2d(kernel size=2, stride=2, padding=0, dilation=1, ceil mode=False)
conv2: Conv2d(6, 16, kernel size=(5, 5), stride=(1, 1))
pool2: MaxPool2d(kernel size=2, stride=2, padding=0, dilation=1, ceil mode=False)
flatten: Flatten(start dim=1, end dim=-1)
fc1: Linear(in features=44944, out features=4096, bias=True)
fc2: Linear(in features=4096, out features=441, bias=True)
```

In the first conventional layer, there are 6 Filters and kernel size is 5x5. In the second conventional layer, there are 16 Filters, and the kernel size is 5x5. The activation function is ReLU. It is applied after the two conventional layers and fully connected layer. No activation function applied after **the final** fully connected layer that gives us HOG vector.

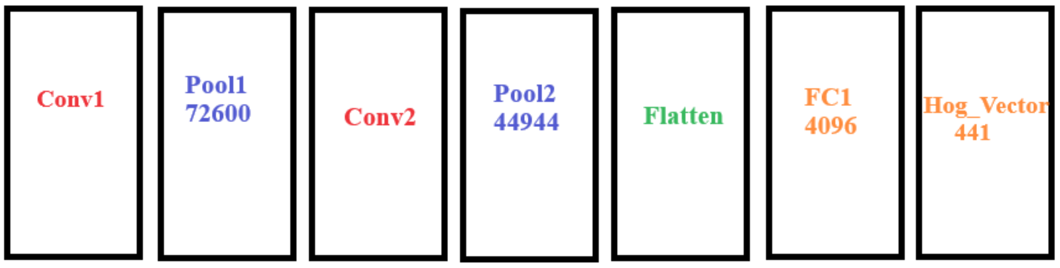

Figure 15: Architecture of HOGNet_2Conv_2Pool_ 441

We separated dataset: 50,000 as training set and 10,000 as validation test. After training model 15 epochs we got **validation loss: 0.0162**. We should pay attention that while training the model for additional epochs might result in a better validation loss, it could also negatively affect the performance of features for certain layers.

3.2.2 HOGNet_3Conv_3Pool_1764

For HogNet 1764 we made transformation for input images to resize them to 256x256 pixels. So, according to this input size and these parameters of hog function: orientations=9, pixels per cell=(32, 32), cells per block=(2, 2); the length of output vector will be 1764. This is the architecture of HogNet 1764:

conv1: Conv2d(3, 32, kernel size=(3, 3), stride=(1, 1))

pool1: MaxPool2d(kernel size=2, stride=2, padding=0, dilation=1, ceil mode=False)

conv2: Conv2d(32, 32, kernel size=(3, 3), stride=(1, 1))

pool2: MaxPool2d(kernel size=2, stride=2, padding=0, dilation=1, ceil mode=False)

conv3: Conv2d(32, 32, kernel size=(3, 3), stride=(1, 1))

pool3: MaxPool2d(kernel size=2, stride=2, padding=0, dilation=1, ceil mode=False)

flatten: Flatten(start dim=1, end dim=-1)

fc1: Linear(in features=28800, out features=4096, bias=True)

fc2: Linear(in features=4096, out features=4096, bias=True) hog vector: Linear(in features=4096, out features=1764, bias=True)

In all conventional neural networks, there are 32 Filters and kernel size is 3x3. The activation function is ReLU. It is applied after the conventional layers and fully connected layers. No activation function applied after **final** fully connected layer that give us HOG vector.

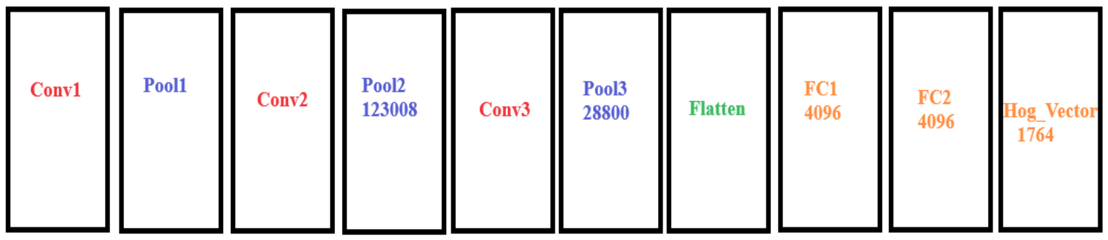

Figure 16: Architecture of HOGNet_3Conv_3Pool_1764

We separated the dataset into two sets: 50,000 as the training set and 10,000 as validation test. After training model 20 epochs, we got validation loss: 0.0042. The same note mentioned before for HogNet 441, we should pay attention that while training the model for additional epochs might result in a better validation loss, it could also negatively affect the performance of features for certain layers.

3.2.3 HOGNet_3Conv_3Pool_441

The same architecture of previous HOGNet_3Conv_3Pool_1764 except the output of predicted HOG vector's length is 441 instead of 1764. In all conventional neural networks, there are 32 Filters and kernel size is 3x3. The activation function is ReLU. It is applied after the conventional layers and fully connected layers. No activation function applied after **the final** fully connected layer that gives us HOG vector. We will discuss a little about the performance of layers in the next sections

4. RESULTS

4.1 Comparing the Performance of layers: VGG16, HOGNet_441 and HOGNet_1764

In this experiment, we employed a traditional Visual Place Recognition (VPR) method that involves extracting features from a query image and comparing them with features extracted from reference images (database images). We used a distance metric, such as Euclidean distance, to identify the closest match as the target place. Specifically, we used the Nordland dataset, which contains images of the same locations across different seasons (Summer, Fall, Winter, and Spring). Each season can be considered a partial dataset. For this experiment, we selected a query image from the Spring dataset and reference images from the Summer dataset. The Nordland dataset consists of a sequence of indexed images captured by a camera. We define a correct prediction as when the difference between the predicted image's index and the actual image's index is within 5 positions (the absolute value). This evaluation method relies on the assumption that the VPR system ranks images based on their similarity to the query, with the closest image (smallest distance) being ranked first. Additionally, we hypothesize that the dataset exhibits a sequential structure, where every 5 consecutive images represent the same place captured across different viewpoints or minor variations. These experiments were very important to discover and study the best features of layers in the same model. This is very useful when we want to use the model as features extractor in classification or recognition tasks.

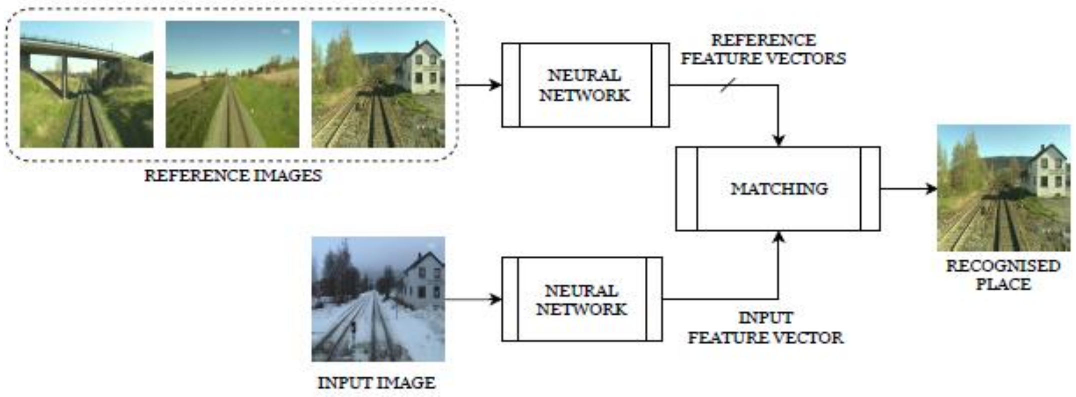

Figure 17: Clarifying how next experiments are done (Olid et al., 2018)

4.1.1 VGG16's Layers

We test the experiment for 4 layers in VGG16:

1. Pool4: length of output features is 100352.

2. Pool5: length of output features is 25088.

3. FC6: length of output features is 4096.

4. FC7: length of output features is 4096.

FC6 and FC7 are fully connected layers before the last layer (classification layer).

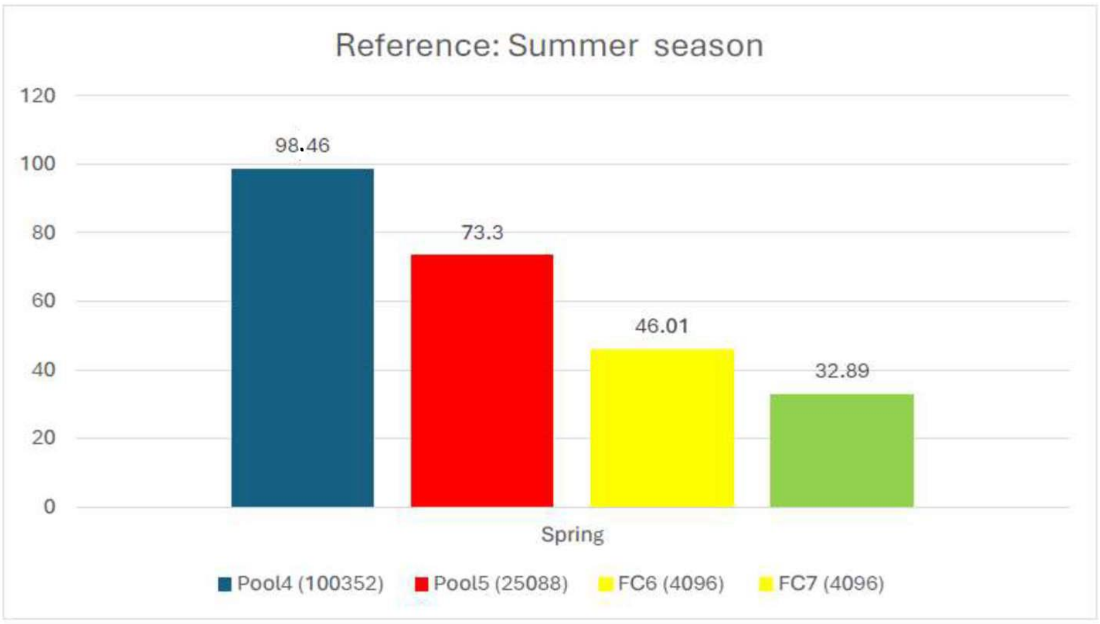

Figure 18: Performance (Accuracy) of 4 layers in VGG16

As we can see, the best features are related to is Pool2 layer (output length is 100352).

4.1.2 HOGNet_2Conv_2Pool_441's layers

We test the experiment for 2 layers in HOGNet_2Conv_2Pool_441:

1. Pool2: length of output features is 44944.

2. FC1: length of output features is 4096.

FC1 is fully connected layers before last layer that gives predicted HOG vector, which

Its length of output features is 441. The predicted HOG vector accuracy is 37.61%

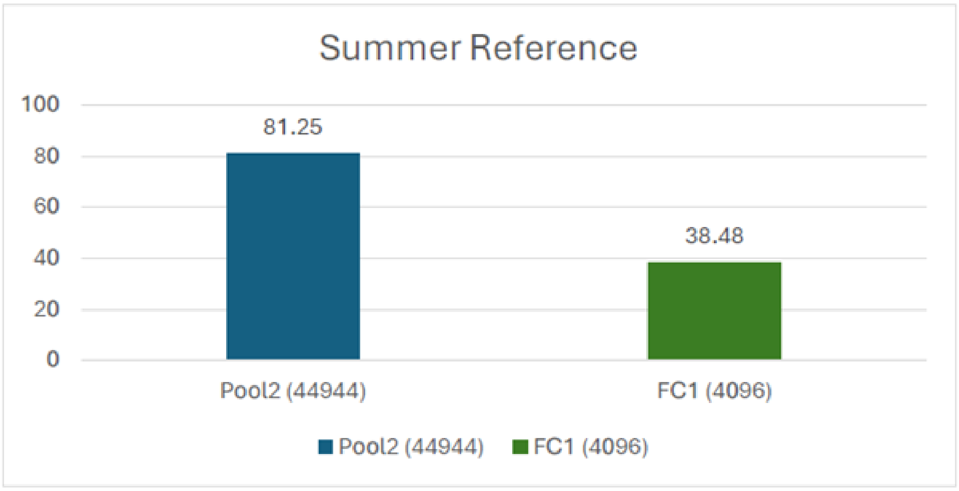

Figure 19: Performance (Accuracy) of 4 layers in HOGNet_2Conv_2Pool_441

4.1.3 HOGNet_3Conv_3Pool_1764's Layers

We test the experiment for 4 layers in HOGNet_3Conv_3Pool_1764:

1. Pool2: length of output features is 123008.

2. Pool5: length of output features is 28800.

3. FC1: length of output features is 4096.

4. FC2: length of output features is 4096.

FC6 and FC7 are fully connected layers before the last layer that gives predicted HOG vector, whose length of out output features is 1764. The predicted HOG vector accuracy is 61.45%.

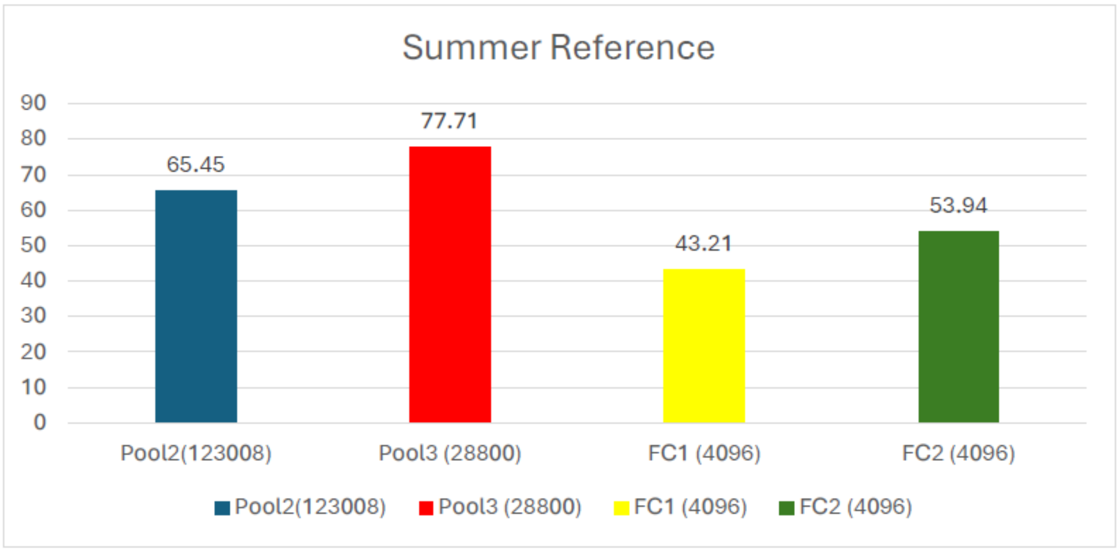

Figure 20: Performance (Accuracy) of 4 layers in HogNet_3Conv_3Pool _1764

As we can see, the best features are related to is Pool3 layer (output length is 28800).

4.1.4 HOGNet_3Conv_3Pool_441's Layers

1. Pool2: length of output features is 93312.
2. Pool5: length of output features is 21632.
3. FC1: length of output features is 4096.
4. FC2: length of output features is 4096.

FC6 and FC7 are fully connected layers before the last layer that gives predicted HOG vector, whose length of out output features is 1764. The predicted HOG vector accuracy is 61.45%

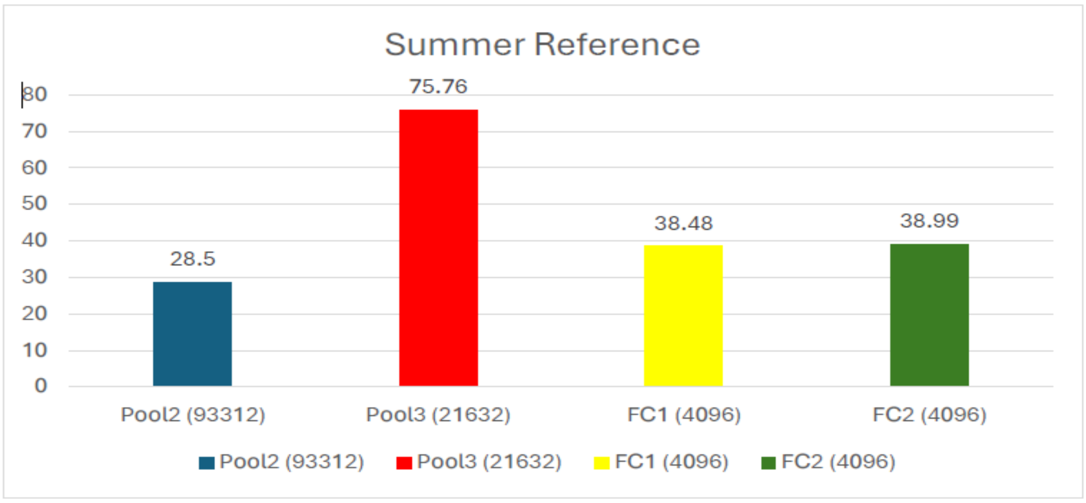

Figure 21: Performance (Accuracy) of 4 layers in HOGNet_3Conv_3Pool_441

As we can see, the best features are related to is Pool3 layer (output length is 21632).

The same model after training it extra 5 epochs starting from latest weights we got. The model has a bit better training loss but a bit worse for testing loss. So, the new model has become more **overfitting** for training set.

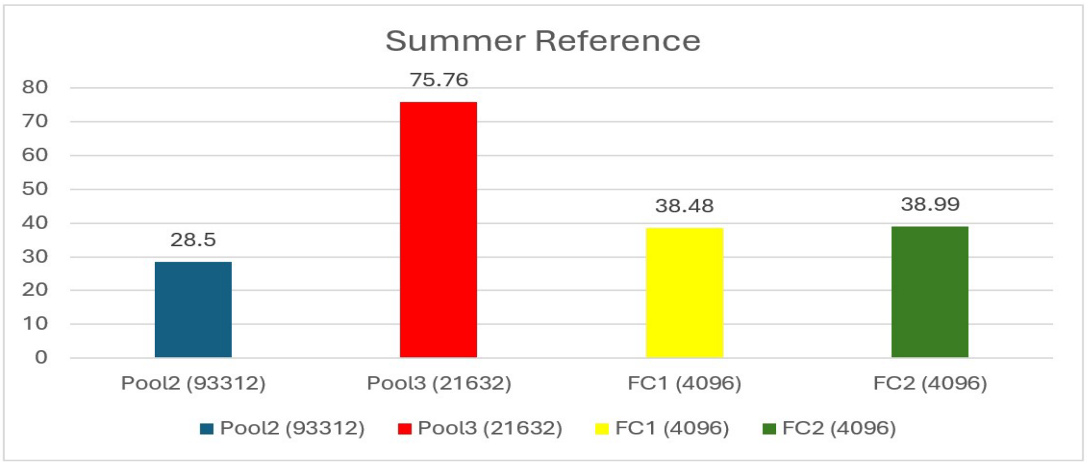

Figure 22: Performance (Accuracy) of 4 layers in HOGNet_3Conv_3Pool_441(second version)

We can see Pool2, FC1, FC2 are improved after re-training model 5 epochs but Pool3 became worse which should be more important for us because it gives the best results on HogNet model. This happened because of overfitting of second version of HOGNet_3Conv_3Pool_441.

4.1.5 Geometric HOG and Predicted HOG

We compared the performance (accuracy) between Geometric (Normal) HOG and predicted HOG for different lengths.

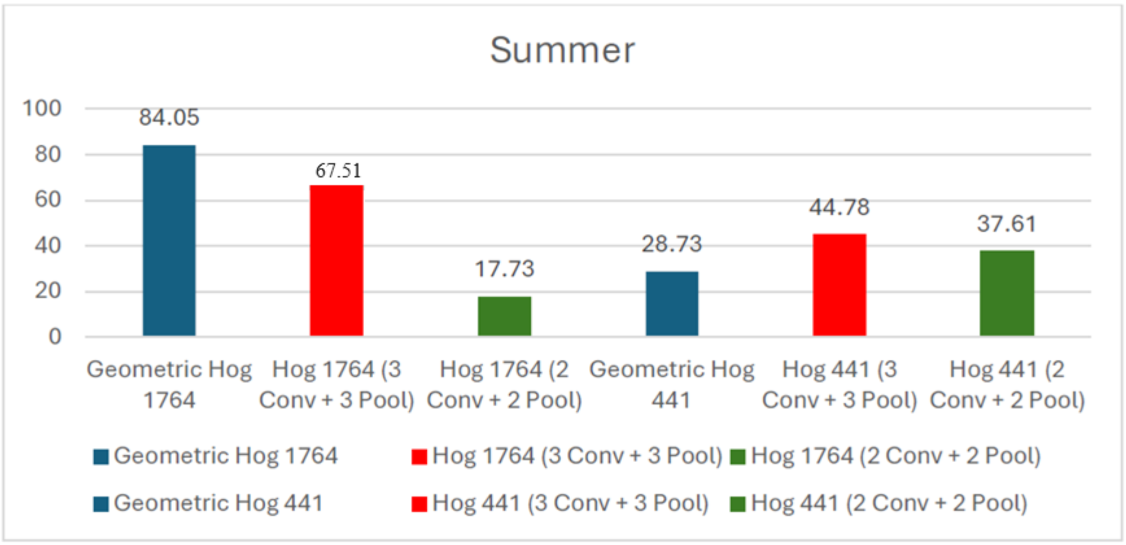

Figure 23: Performance between Geometric HOG and Predicted HOG.

The accuracy of a HOG vector with a length of 1764 is expected to be better than one with a length of 441 because it contains more detailed features. The predicted HOG vector with a length of 441 outperforms the original HOG vector. However, for the

HOG vector with a length of 1764, the performance was worse than the original. This may indicate that we need to further reduce the loss. Additionally, it is not necessary to always use features from the last layer; we can select features from the most effective layers based on our experiments.

4.1.6 Exhausted training for HOGNet_2Conv_2Pool_ 441 and HOGNet_3Conv_3Pool_ 1764

Exhaustive training refers to a comprehensive and thorough training process in machine learning where a model is trained extensively on a dataset. This involves using many iterations or epochs, often employing various techniques to ensure the model learns as much as possible from the data. The goal of exhaustive training is to optimize the model's performance by minimizing errors and improving accuracy. However, it requires significant computational resources and time. To ensure that we arrive the best version of two best two models: HOGNet_2Conv_2Pool_441 and HOGNet_3Conv_3Pool_1764 we trained both extra epochs starting from **latest** weights.

For **HogNet_3Conv_3Pool_1764** after training it extra epochs the validation loss became from 0.0042 to 0.0038:

Model (validation loss = 0.0042):

Accuracy of predicted Hog was 61.45%.

Accuracy of Pool3 was 77.71%.

Model (validation loss = 0.0038):

The accuracy of predicted Hog became 67.51 %.

But the accuracy of Pool3 became 75.48 %.

So, the output layer (predicted layer) became better, but the accuracy of best layer (Pool3) became a bit worse.

For **HogNet_2Conv_2Pool_441** after training it extra epochs the validation loss became from 0.0162 to 0.0159:

Model (validation loss = 0.0162):

The accuracy of predicted Hog was 37.61%.

Accuracy of Pool3 was 81.25%.

Model (validation loss = 0.0159):

The accuracy of predicted Hog became 30.2%.

But the accuracy of Pool3 became 81.19%.

So, both accuracy of the output layer (predicted layer) and best layer (Pool2) became a bit worse.

So, finally for both models: HogNet_2Conv_2Pool_441 and HogNet_3Conv_3Pool_1764 are Exhausted training: we can’t improve them more.

After conducting multiple experiments to optimize performance, we will discuss the two best models of HOGNet. From now onwards, we will call HOGNet_2Conv_2Pool_441 as **HOGNet_441** and HOGNet_3Conv_3Pool_ 1764 as **HOGNet_1764**.

4.2 HOGNet for VPR task

In this experiment, we prepare training set and validation (test) set from Nordland dataset as follows:

Summer, Fall and winter as training set (24000 images).

Spring as validation (test) set. (8000 images).

For each sequence of 5 images, we consider it as place (class), we have 1600 class.

I compare with same conditions these three backbones:

1) Features of Pool3 (28800) in HogNet_1764.

2) Features of Pool2 (44944) in HogNet_441.

3) Features of Pool4 (100352) in VGG16.

For accelerating data loading, we should add these parameters: num_workers and pin_memory, as this example:

```
DataLoader(val_dataset, batch_size=batch_size, shuffle=True,
num_workers=4, pin_memory=True)
```

num_workers: how many subprocesses to use for data loading. 0 means that the data will be loaded in the main process.

pin_memory: the data loader will copy Tensors into device/CUDA pinned memory before returning them.

To get the best model from all epochs, in **PyTorch** it is easy because, we can see internal statements of training but for TensorFlow, we should additional library like in the example below:

```
from keras.callbacks import ModelCheckpoint
checkpoint = ModelCheckpoint(filepath =
'MobileNet_Nordland_Classifier_10_images.hdf5',
monitor='val_accuracy', save_best_only=True, mode='max', verbose=1)

#monitor = 'val_accuracy' For regression tasks we should use
'val_loss'
#mode = max For accuarcy but it should equal 'min' for loss
```

These are results I have got these results after 50 epochs:

	HogNet_1764	HogNet_441	VGG16
Number of total parameters	**46,100,992**	**71,914,872**	**168,200,064**
Validation Accuracy	**65.86 %**	**67.7%**	**85.04 %**
Inference time	**2.7 ms**	**3.44 ms**	**124.17 ms**

In all models, we have stopped learning for features layers, just classification parameters between last feature layer (Pool3, Pool2 or Pool4) and fully connected layer of output classes are trained.

```
# For stop training for HOGNet layers
for param in hogNet.parameters():
    param.requires_grad = False
```

Then we repeated the same experiment but for each sequence of 10 images instead of 5, we consider it as place (class), we have 800 class. We added **ResNet101** and **MobileNet** to this experiment.

In Pytorch we can use this library:

```
from torchmetrics.classification import MulticlassRecall
```

We should mention an essential parameter called **average** that take one these values:

- micro: Sum statistics over all labels
- macro: Calculate statistics for each label and average them
- weighted: calculates statistics for each label and computes weighted average using their support
- "none" or None: calculates statistic for each label and applies no reduction

In our experiment, we used **micro** option.

We got these Recall values:

	HOGNet_441	VGG16	ResNet101	MobileNet
@Recall1	71.37	87.78	54.13	22.99
@Recall2	78.21	87.84	59.93	30.03
@Recall3	81.16	87.84	61.58	33.43
@Recall4	82.96	87.84	62.06	35.70
@Recall5	84.19	87.84	62.20	37.08
@Recall10	87.73	87.92	62.50	39.76
@Recall20	**90.76**	**88.00**	**62.70**	**40.69**

Figure 24: Recall values for HOGNet_441, VGG16, ResNet101 and MobileNet in VPR task.

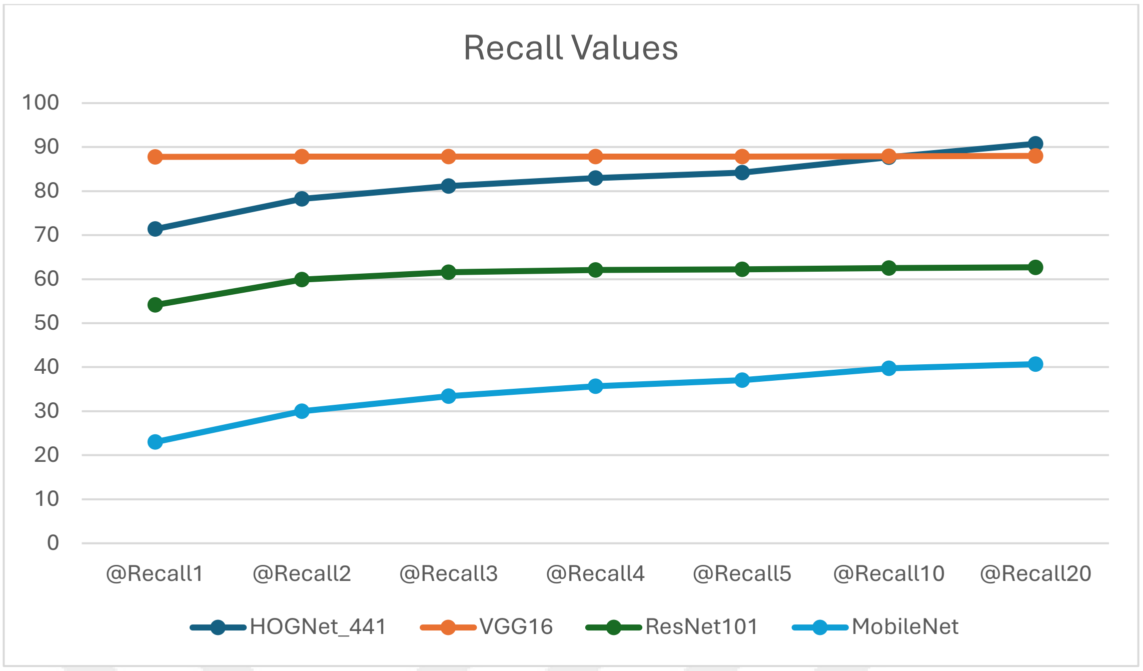

Figure 25: Recall Curve for HOGNet 441 and VGG16

As we can see, starting from Recall@10 both networks (HOGNet 441 and VGG16) are very closed as accuracy but of course HOGNet 441 need less computational time according to its lightweight architecture.

We should consider both ResNet101 and MobileNet, using the features from the last layer in the feature extraction part of the model. However, to ensure we are using the best layer, additional experiments, like those conducted in **Section** 4.1, are needed.

	HOGNet_441	VGG16	ResNet101	MobileNet
Total parameters	35,958,872	87,917,664	122,940,576	43,370,464

Figure 26: Total parameters of each model

HOGNet_441 model has the least total parameters which means the least computational time regardless of type of processor that is used. HOGNet is more resource-efficient compared to MobileNet, which is widely used in mobile and resource-limited devices, while still delivering excellent accuracy. The feature length in VGG16 is 100,352, whereas HOGNet reduces this to 44,944, which is less than half.

```
resnet_model = ResNet101(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))

mobilenet_model = MobileNet(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))
```

```
# weights='imagenet' To use model with weights that are gotten
after training on imageNet dataset

# 'include_top=False' To exclude the classification part of model
and get just features extracting part of model
```

4.3 Siamese Network using VGG16

In this experiment, we just tried to discover the performance of Siamese networks. The Siamese model consists of two identical networks, each one of them has same architecture as following: pre-trained VGG16 that uses weights of ImageNet. I excluded the output layer and stopped (froze) the VGG16 for training; then I added two dense layers the first with 512 parameters and the second with 192. At the end I added GlobalAveragePooling2D layer. The training will be in the last three layers. So, the final Siamese model will be as follows:

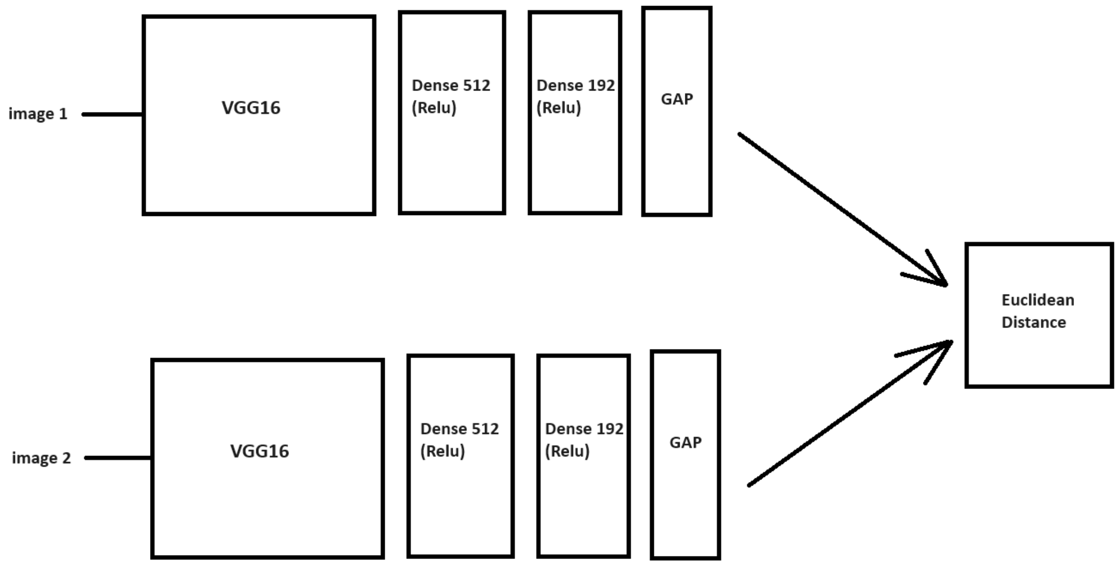

Figure 27: Siamese network using VGG16 and Global Average Pooling layer

I have prepared pairs of images with the same place, I consider that each 6 consecutive images belong to same place which means 3 pairs. The dataset is from India in Hyderabad. The experiment is done on a local PC via Anaconda. The process of training is very slow and even the PC has collapsed. We had to choose just a sample with 202 images (101 pairs). After 50 epochs I got a loss = 0.04.

For testing the Siamese network, I requested the model to find the closest image from reference images (the same used in training model). The query image is outside of reference images. The smallest distance from all distances that model give us is considered as the correct image.

We compared Siamese network with the previous method that depends on VGG16 without any training. VGG16 is used to extract features then depending on smallest Euclidian distance between query image and reference images. This method gives us a closer image from the Siamese image. In the Siamese network, the query image that its index is 202 and the closest image its index is 196. But in VGG16 the closest image its index is 198 which is closer to query image. This may yield negative results due to variations in viewpoint or illumination. Additional experiments are necessary to fully evaluate the performance of the Siamese network. Our current work provides an initial understanding of its capabilities.

5. CONCLUSIONS

Leveraging HOG features, we developed HOGNet, a robust pre-trained network suitable for image recognition and classification tasks, such as Visual Place Recognition (VPR). HOGNet is particularly advantageous for devices with limited computational resources or for real-time applications. We compared performance with well-known pre-trained model VGG16.We did a lot of experiments to find the best layers that gave good features. Our results showed that HOGNet achieves recall values very close to VGG16, specifically at @5, @10, and @20. HOGNet achieves much better than ResNet101 and MobileNet models. The limitation of HOGNet is for applications that prioritize very high accuracy at first prediction. As a promising direction for future research, we recommend exploring HOGNet as the backbone network for deep learning-based VPR methods. We recommend benefitting from HOGNet as Transfer learning application in different images classification tasks. In this thesis, the importance of transfer learning is demonstrated in comparison to learning from scratch.

REFERENCES

Abdul Hafez, A. H., Tello, A., & Alqaraleh, S. (2022). COLCONF: Collaborative ConvNet Features-based Robust Visual Place Recognition for Varying Environments. *Arabian Journal for Science and Engineering*, *47*(2), 2381–2395. https://doi.org/10.1007/s13369-021-06148-8

Ali-bey, A., Chaib-draa, B., & Giguère, P. (2023). *MixVPR: Feature Mixing for Visual Place Recognition*. http://arxiv.org/abs/2303.02190

Aljuaidi, R., Su, J., & Dahyot, R. (2019). *Mini-Batch VLAD for Visual Place Retrieval.*

Arandjelović, R., Gronat, P., Torii, A., Pajdla, T., & Sivic, J. (2015). *NetVLAD: CNN architecture for weakly supervised place recognition*. http://arxiv.org/abs/1511.07247

Berton, G., Masone, C., & Caputo, B. (2022). *Rethinking Visual Geo-localization for Large-Scale Applications*. http://arxiv.org/abs/2204.02287

Berton, G., Trivigno, G., Caputo, B., & Masone, C. (2023). *EigenPlaces: Training Viewpoint Robust Models for Visual Place Recognition*. http://arxiv.org/abs/2308.10832

Bromley, J., Guyon, I., Lecun, Y., Sickinger, E., Shah, R., Bell, A., & Holmdel, L. (n.d.). *Signature Verification using a "Siamese" Time Delay Neural Network*.

Dalal, N., & Triggs, B. (2005). *Histograms of Oriented Gradients for Human Detection*. http://lear.inrialpes.fr

Ge, Y., Wang, H., Zhu, F., Zhao, R., & Li, H. (2020). *Self-supervising Fine-grained Region Similarities for Large-scale Image Localization*. http://arxiv.org/abs/2006.03926

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., & Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern Recognition*, *77*, 354–377. https://doi.org/10.1016/j.patcog.2017.10.013

Hausler, S., Garg, S., Xu, M., Milford, M., & Fischer, T. (2021). *Patch-NetVLAD: Multi-Scale Fusion of Locally-Global Descriptors for Place Recognition*. http://arxiv.org/abs/2103.01486

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition*. http://arxiv.org/abs/1512.03385

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. http://arxiv.org/abs/1704.04861

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, *2017-January*, 2261–2269. https://doi.org/10.1109/CVPR.2017.243

Keetha, N. V., Milford, M., & Garg, S. (2021). *A Hierarchical Dual Model of Environment- and Place-Specific Utility for Visual Place Recognition*. http://arxiv.org/abs/2107.02440

Koyuncu, B., & Koyuncu, H. (2019). Handwritten Character Recognition by using Convolutional Deep Neural Network; Review. In *INTERNATIONAL JOURNAL of ENGINEERING TECHNOLOGIES-IJET Koyuncu and Koyuncu* (Vol. 5, Issue 1).

Musallam, M. A., & Lu, V. G. (n.d.). *Self-Supervised Learning for Place Representation Generalization across Appearance Changes VincentGaudillì ere*. https://www.lmo.space

NOGAY, H. S. (2018). Classification of Different Cancer Types by Deep Convolutional Neural Networks. *Balkan Journal of Electrical and Computer Engineering*, *6*, 56–59. https://doi.org/10.17694/bajece.410250

Olid, D., Fácil, J. M., & Civera, J. (n.d.). *Single-View Place Recognition under Seasonal Changes*. http://webdiis.unizar.es/

O'Shea, K., & Nash, R. (2015). *An Introduction to Convolutional Neural Networks*. http://arxiv.org/abs/1511.08458

Pal, C., Verma, P., Rohit, H., Gyaneshwar, D., Channappayya, S. S., & Acharyya, A. (2023). SqueezeNetVLAD: High-speed power and memory efficient GPS less accurate network model for visual place recognition on the edge. *21st IEEE Interregional NEWCAS Conference, NEWCAS 2023 - Proceedings*. https://doi.org/10.1109/NEWCAS57931.2023.10198114

Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. In *IEEE Transactions on Knowledge and Data Engineering* (Vol. 22, Issue 10, pp. 1345–1359). https://doi.org/10.1109/TKDE.2009.191

Wang, H. (2017). Detection of Humans in Video Streams Using Convolutional Neural Networks. In *DEGREE PROJECT COMPUTER SCIENCE AND ENGINEERING*.

Xu, P., Huang, L., & Song, Y. (2022). An optimal method based on HOG-SVM for fault detection. *Multimedia Tools and Applications*, *81*(5), 6995–7010. https://doi.org/10.1007/s11042-022-12020-0

Zaffar, M., Ehsan, S., Milford, M., & McDonald-Maier, K. (2020). CoHOG: A light-weight, compute-efficient, and training-free visual place recognition technique for changing environments. *IEEE Robotics and Automation Letters*, *5*(2), 1835–1842. https://doi.org/10.1109/LRA.2020.2969917