Deep Learning Practical Work 2-a

William KHIEU
Akli Mohamed Ait-Oumeziane

December 24, 2024

Contents

Section 1 - VGG16 Architecture

1. **Knowing that the fully-connected layers account for the majority of the parameters in a model, give an estimate on the number of parameters of VGG16**

Layer	Output Shape	Weights
`INPUT`	$224 \times 224 \times 3$	0
`CONV3-64`	$224 \times 224 \times 64$	$(3 \times 3 \times 3) \times 64 = 1{,}728$
`CONV3-64`	$224 \times 224 \times 64$	$(3 \times 3 \times 64) \times 64 = 36{,}864$
`POOL2`	$112 \times 112 \times 64$	0
`CONV3-128`	$112 \times 112 \times 128$	$(3 \times 3 \times 64) \times 128 = 73{,}728$
`CONV3-128`	$112 \times 112 \times 128$	$(3 \times 3 \times 128) \times 128 = 147{,}456$
`POOL2`	$56 \times 56 \times 128$	0
`CONV3-256`	$56 \times 56 \times 256$	$(3 \times 3 \times 128) \times 256 = 294{,}912$
`CONV3-256`	$56 \times 56 \times 256$	$(3 \times 3 \times 256) \times 256 = 589{,}824$
`CONV3-256`	$56 \times 56 \times 256$	$(3 \times 3 \times 256) \times 256 = 589{,}824$
`POOL2`	$28 \times 28 \times 256$	0
`CONV3-512`	$28 \times 28 \times 512$	$(3 \times 3 \times 256) \times 512 = 1{,}179{,}648$
`CONV3-512`	$28 \times 28 \times 512$	$(3 \times 3 \times 512) \times 512 = 2{,}359{,}296$
`CONV3-512`	$28 \times 28 \times 512$	$(3 \times 3 \times 512) \times 512 = 2{,}359{,}296$
`POOL2`	$14 \times 14 \times 512$	0
`CONV3-512`	$14 \times 14 \times 512$	$(3 \times 3 \times 512) \times 512 = 2{,}359{,}296$
`CONV3-512`	$14 \times 14 \times 512$	$(3 \times 3 \times 512) \times 512 = 2{,}359{,}296$
`CONV3-512`	$14 \times 14 \times 512$	$(3 \times 3 \times 512) \times 512 = 2{,}359{,}296$
`POOL2`	$7 \times 7 \times 512$	0
`FC`	$1 \times 1 \times 4096$	$7 \times 7 \times 512 \times 4096 = 102{,}760{,}448$
`FC`	$1 \times 1 \times 4096$	$4096 \times 4096 = 16{,}777{,}216$
`FC`	$1 \times 1 \times 1000$	$4096 \times 1000 = 4{,}096{,}000$
Total Parameters		≈ 138M

Table 1: Layer-wise dimensions and parameter counts for VGG16 (biases omitted).

2. **What is the output size of the last layer of VGG16? What does it correspond to?**

   The last fc layer has an output size of 1000. This corresponds to the number of classes in the ImageNet dataset. Each of these 1000 numbers represents the (pre-softmax) logit for one of the 1000 ImageNet classes.

3. **Apply the network on several images of your choice and comment on the results.**

   - **What is the role of the ImageNet normalization?**

     Normalizing images ensures that:

     – All input images have similar distributions (reducing variation in brightness, contrast, etc.).

  - The network sees data at test time that matches the statistical properties of what it saw during training.

  Thus it is necessary to normalize the input images during testing as well.

- **Why setting the model to eval mode?**

  Setting the model ‘model.eval()‘ places certain layers (such as dropout and batch normalization) into “inference” mode:

  - **Dropout** layers no longer randomly zero out neurons, ensuring deterministic behavior.
  - **Batch Normalization** layers use their running averages for mean and variance rather than computing them on the batch.

4. Visualize several activation maps obtained after the first convolutional layer. How can we interpret them?

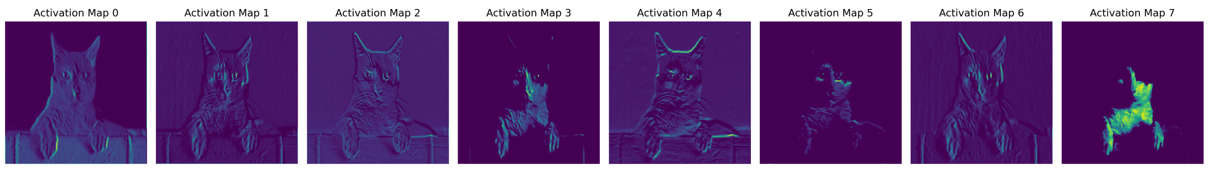

Figure 1: Some several activation maps obtained after the first convolutional layer.

Each activation map corresponds to the response of one filter (or “channel”) within the first convolutional layer. Because it is the very first layer of a deep network (like VGG16), these filters typically capture low-level features such as:

- **Edges and contours:** For example, horizontal or vertical lines.
- **Corners or simple shape outlines.**
- **Color or brightness gradients:** For instance, light vs. dark regions.

Section 2 - Transfer Learning with VGG16 on 15 Scene

2.1 Approach

5. Why not directly train VGG16 on 15 Scene?

The 15 Scene dataset is too small to effectively train a deep CNN from scratch without overfitting and large computational overhead:

- VGG16 has around 138 million parameters. Training such a large model typically requires hundreds of thousands (or millions) of labeled images.
- The 15 Scene dataset is relatively small (around 100 images per class). Training from scratch would lead to overfitting since there's not enough images for the network to learn robust filters without memorizing the training set.

6. How can pre-training on ImageNet help classification for 15 Scene?

ImageNet pre-training teaches VGG16 to recognize a wide variety of low-level and mid-level features (edges, textures, shapes). These features are reusable across many visual tasks. The first layers learn generic features (e.g., edges, corners, blobs) that are useful well beyond ImageNet's 1000 classes.

7. What limits can you see with feature extraction?

If the 15 Scene images are very different in style, color distribution, or content than the original ImageNet dataset, the network's early layers might not perfectly transfer. Without fine-tuning the network's weights on your new task, you can't adapt the learned filters to domain-specific nuances.

When you take the outputs of a specific layer (e.g., relu7), you are locked into the features that were learned for ImageNet.

2.2 Feature Extraction with VGG16

8. What is the impact of the layer at which the features are extracted?

- **Early layers (e.g., relu1):** Capture low-level features like edges, corners, and textures. These features are more generic and reusable across different tasks.
- **Later layers (e.g., relu7):** Capture more high-level features like object parts, textures, and patterns. These features are more task-specific and less reusable.

Since the target task is similar to ImageNet, using a later layer (like relu7) should yields more discriminative features.

9. The images from 15 Scene are black and white, but VGG16 requires RGB images. How can we get around this problem?

A simple fix is to replicate the single grayscale channel three times to create a 3-channel input:

$$\text{gray} \rightarrow \begin{bmatrix} \text{gray} \\ \text{gray} \\ \text{gray} \end{bmatrix}.$$

This maintains the same intensity in each of the R, G, and B channels and satisfies the expected (3-channel) input format.

2.3 Training SVM classifiers

10. Rather than training an independent classifier, is it possible to just use the neural network? Explain.

Yes, it is possible to use the neural network as a classifier by replacing (or fine-tuning) the last fully connected layer with a new one that has the desired number of output classes. However, there are practical reasons why training a separate SVM on extracted features is often simpler or more effective for smaller datasets:

- Fine-tuning a deep network with many parameters (with or without propogating the gradients to the rest of the network) on a small dataset (like 15 Scene) risks overfitting if you update too many layers.
- Using feature extraction + SVM leverages the robust features learned from ImageNet while only training a small number of SVM parameters.
- It's faster to train and implement an SVM on extracted features than to fine-tune a deep network.

2.4 Going further

11. For every improvement that you test, explain your reasoning and comment on the obtained results.

- **Change the layer at which the features are extracted** :

  We didn't use the earliest pooling layers (`pool1`, `pool2`, `pool3`) because their feature maps are extremely large in dimension (e.g., hundreds of thousands to over a million features per image), which can easily exceed available memory and **crash the Jupyter kernel**. For instance:

  – **pool1** shape: $64 \times 112 \times 112 = 802{,}816$ features

  – **pool2** shape: $128 \times 56 \times 56 = 401{,}408$ features

  – **pool3** shape: $256 \times 28 \times 28 = 200{,}704$ features

  These large feature vectors put high demands on RAM when we process a dataset of many images and train an SVM. Hence, we focused on **pool4** and **pool5** (and the fully connected layers **relu6** and **relu7**) to get a more manageable feature dimension.

  Importance of the Layer Depth:

  – Early layers (closer to `pool1`, `pool2`) tend to capture low-level features like edges, textures, and corners.

  – Deeper layers (like `pool4`, `pool5`) capture increasingly high-level or semantic concepts more relevant to classification tasks (e.g., object parts).

- Fully connected layers (`relu6`, `relu7`) are even more specialized to the ImageNet classes, which can sometimes help (or sometimes overfit) depending on how similar your dataset is to ImageNet.

Results:

Layer	Accuracy	Feature Dim	Feature Time	SVM Train Time
pool4	0.8797	100352	21.45 s	21.04 s
pool5	0.8988	25088	25.23 s	2.38 s
relu6	0.8737	4096	23.03 s	0.40 s
relu7	0.8760	4096	23.06 s	0.74 s

(a) **Accuracy**

- `pool5` gave the highest accuracy (0.8988).
- `relu6` and `relu7` were slightly lower (around 0.87–0.88).
- This suggests that features right after `pool5` in VGG16, which still retain some spatial structure, can be very effective.

(b) **Feature Dimension vs. Training Time**

- `pool4` has a much larger feature dimension (100k+) compared to `pool5` (25k) or `relu6`/`relu7` (4k).
- This leads to a longer SVM training time for `pool4` (21 s vs. $\sim$2 s for `pool5` and < 1 s for `relu6`/`relu7`).
- The accuracy for `pool4` is still decent, but the cost in memory and training time is higher.

(c) **Trade-Offs**

- If you want **best accuracy**, `pool5` appears strongest in this setup.
- If you want **fast classification** after feature extraction, `relu6` or `relu7` can be trained by an SVM in under a second, though with slightly lower accuracy.
- `pool4` is somewhat in-between, but training an SVM on a 100k-dimensional vector can be noticeably slower.

- **Try other available pre-trained networks** :

  Architectural Differences

  - **VGG16**:
    * A relatively simple chain of 2D convolutions (3×3 filters), ReLUs, and pooling layers.
    * Has about 138 million parameters.
    * Lacks shortcut/skip connections.
  - **ResNet50**:
    * Incorporates *residual (skip) connections*, allowing much deeper networks (50 layers for ResNet50) without vanishing gradients.
    * Around 25 million parameters—despite being deeper than VGG16, thanks to more efficient use of 1×1 convolutions and residual blocks.

Experiment Outcomes

- **Truncated ResNet50** (with SVM on extracted features) achieves $\sim$ 90.82% test accuracy.
- **Truncated VGG16** (with SVM on extracted features) achieves $\sim$ 88.58% test accuracy.

Interpretation

- The skip connections and deeper architecture in ResNet50 typically learn more robust, high-level representations.
- Even when truncated (removing the final fully connected layer and using a global average pooling output), the 2048-dimensional feature vectors are often *more linearly separable* for downstream tasks.

Small Performance Gap ($\sim$ 2% difference)

- Both VGG16 and ResNet50 are strong ImageNet-pretrained models. The fact that VGG16 obtains nearly 88.6% indicates it also provides rich features.
- The $\sim$ 2% difference is consistent with typical literature showing that ResNet variants generally outperform VGG on many tasks, especially when fine-tuned or used for feature extraction.

Trade-Offs

- ResNet50 might take slightly longer to run a forward pass on each batch due to its depth (more layers).
- VGG16 can be more memory-intensive in the fully connected layers, but has a more straightforward design.
- Ultimately, if maximum accuracy is the goal, ResNet50 is generally a better choice; if memory constraints or simpler implementation is critical, VGG16 may suffice.