

CS671: Deep Learning and Applications  
Programming Assignment 5

RAUNAV GHOSH(M.Tech-CSP), T22104

PRASHANT DHANANJAY KULKARNI(M.Tech-CSE), T22058

SACHIN BAHULEYAN(M.Tech-CSE), T22060

# Contents

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Introduction</b>   | <b>2</b> |
| 1.1      | Problem statement . . . . .   | 2        |
| 1.2      | Methodology . . . . .   | 3        |
| 1.2.1    | Task 1: Develop a convolutional neural network where the input layer should take 224 x 224 3 channel image . . . . .                        | 3        |
| 1.2.2    | Task 2:Leverage Tensorflow Keras API, use VGG19 pretrained on ImageNet. Modify the classification layer of VGG19 to 5 output nodes. . . . . | 4        |
| <b>2</b> | <b>Results</b>  | <b>5</b> |
| 2.1      | Architecture 1 . . . . .  | 5        |
| 2.2      | Architecture 2 . . . . .  | 6        |
| 2.3      | Architecture 3 . . . . .  | 7        |
| 2.4      | Best Architecture: Architecture 1 . . . . .   | 8        |
| 2.5      | Using pretrained VGG19 to make CNN . . . . .  | 9        |
| 2.6      | Observations . . . . .  | 10       |

# Chapter 1

## Introduction

**Convolutional Neural Network** A Convolutional Neural Network (CNN) is a type of feed forward neural network that is commonly used in computer vision tasks, such as image classification, object detection, and segmentation. CNNs are designed to automatically learn and extract features from raw input data, such as images, by using convolutional filters or kernels that scan the input data and detect specific patterns or features. The convolution operation involves sliding a small window (the kernel) over the input data, multiplying the values within the window by the corresponding values in the kernel, and summing the results to produce a single output value. The output of the convolutional layer is then passed through a non-linear activation function, such as ReLU (Rectified Linear Unit), which introduces non-linearity into the model and helps to capture more complex patterns in the input data. CNNs typically consist of multiple convolutional layers, followed by pooling layers that downsample the feature maps to reduce the spatial dimensions and make the model more computationally efficient. The final output of the CNN is typically fed through one or more fully connected layers, which perform the classification or regression task based on the extracted features.

### 1.1 Problem statement

1. Develop a convolutional neural network where the input layer should take 224 x 224 3 channel image.
2. Leverage Tensorflow Keras API, use VGG19 pretrained on ImageNet. Modify the classification layer of VGG19 to 5 output nodes.

## 1.2 Methodology

For this assignment we received datasets of 5 classes which is summarised below,

| Class              | Train | Validation | Test |
|--------------------|-------|------------|------|
| <i>car side</i>    | 50    | 10         | 20   |
| <i>ewer</i>        | 50    | 10         | 20   |
| <i>grand piano</i> | 50    | 10         | 20   |
| <i>helicopter</i>  | 50    | 10         | 20   |
| <i>laptop</i>      | 50    | 10         | 20   |

Table 1.1: Dataset

### 1.2.1 Task 1: Develop a convolutional neural network where the input layer should take 224 x 224 3 channel image

In this task we designed three convolution neural networks with three different architecture and observed the output of these networks. Their architectures are given in Table 1.2, 1.3 and 1.4. All the convolutional layers were activated with *Rectified Linear* activation functions. All these networks were trained with *Adam optimizers* with a batch size set to 50 and learning rate of 0.001. The inputs were presented in a random fashion.

| Layer       | Output Shape |
|-------------|--------------|
| Input       | (224,224,3)  |
| Conv2D      | (54,54,8)    |
| Max Pooling | (26,26,8)    |
| Conv2D      | (22,22,16)   |
| Max Pooling | (10,10,16)   |
| Flatten     | 1600         |
| Dense       | 128          |
| Output      | 5            |

Table 1.2: Convolutional network: Architecture 1

| Layer       | Output Shape |
|-------------|--------------|
| Input       | (224,224,3)  |
| Conv2D      | (54,54,8)    |
| Max Pooling | (26,26,8)    |
| Conv2D      | (22,22,16)   |
| Max Pooling | (10,10,16)   |
| Conv2D      | (8,8,32)     |
| Max pooling | (3,3,32)     |
| Flatten     | 288          |
| Dense       | 128          |
| Output      | 5            |

Table 1.3: Convolutional neural network: Architecture 2

| Layer       | Output Shape |
|-------------|--------------|
| Input       | (224,224,3)  |
| Conv2D      | (54,54,8)    |
| Max Pooling | (26,26,8)    |
| Conv2D      | (22,22,16)   |
| Max Pooling | (10,10,16)   |
| Conv2D      | (8,8,32)     |
| Conv2D      | (6,6,64)     |
| Max pooling | (2,2,64)     |
| Flatten     | 256          |
| Dense       | 128          |
| Dense       | 5            |

Table 1.4: Convolutional neural network: Architecture 3

### 1.2.2 Task 2:Leverage Tensorflow Keras API, use VGG19 pretrained on ImageNet. Modify the classification layer of VGG19 to 5 output nodes.

For this task we started with a pretrained VGG19 network and removed the Fully connected block from the network and added a Dense layer of 5 nodes to use the network classify 5 classes. The network is summarised in Table 1.5. The network was trained by first freezing the convolutional blocks and then training only the dense layer with *Adam optimizers* with a batch size set to 50 and learning rate of 0.001. The inputs were presented in a random fashion.

| Layer                      | Output Shape   |
|----------------------------|----------------|
| input 1 (InputLayer)       | (224, 224, 3)  |
| block1 conv1 (Conv2D)      | (224, 224, 64) |
| block1 conv2 (Conv2D)      | (224, 224, 64) |
| block1 pool (MaxPooling2D) | (112, 112, 64) |
| block2 conv1 (Conv2D)      | (112, 112, 12) |
| block2 conv2 (Conv2D)      | (112, 112, 12) |
| block2 pool (MaxPooling2D) | (56, 56, 128)  |
| block3 conv1 (Conv2D)      | (56, 56, 256)  |
| block3 conv2 (Conv2D)      | (56, 56, 256)  |
| block3 conv3 (Conv2D)      | (56, 56, 256)  |
| block3 conv4 (Conv2D)      | (56, 56, 256)  |
| block3 pool (MaxPooling2D) | (28, 28, 256)  |
| block4 conv1 (Conv2D)      | (28, 28, 512)  |
| block4 conv2 (Conv2D)      | (28, 28, 512)  |
| block4 conv3 (Conv2D)      | (28, 28, 512)  |
| block4 conv4 (Conv2D)      | (28, 28, 512)  |
| block4 pool (MaxPooling2D) | (14, 14, 512)  |
| block5 conv1 (Conv2D)      | (14, 14, 512)  |
| block5 conv2 (Conv2D)      | (14, 14, 512)  |
| block5 conv3 (Conv2D)      | (14, 14, 512)  |
| block5 conv4 (Conv2D)      | (14, 14, 512)  |
| block5 pool (MaxPooling2D) | (7, 7, 512)    |
| flatten (Flatten)          | (25088)        |
| dense (Dense)              | (5)            |

Table 1.5: Modified VGG19: Used for classifying 5 classes

# Chapter 2

## Results

### 2.1 Architecture 1

The metrics measured for architecture 1 are shown in Table 2.1. The corresponding confusion matrices are given in Table 2.10, 2.2 and 2.3.

| Data           | Loss   | Accuracy |
|----------------|--------|----------|
| Training set   | 0.0006 | 1.0000   |
| Validation set | 1.0972 | 0.8600   |
| Test set       | 1.2876 | 0.8000   |

Table 2.1: Architecture 1 metrics

|                    | Pred car side | Pred ewer | Pred grand piano | Pred helicopter | Pred laptop |
|--------------------|---------------|-----------|------------------|-----------------|-------------|
| Actual car side    | 50            | 0         | 0                | 0               | 0           |
| Actual ewer        | 0             | 50        | 0                | 0               | 0           |
| Actual grand piano | 0             | 0         | 50               | 0               | 0           |
| Actual helicopter  | 0             | 0         | 0                | 50              | 0           |
| Actual laptop      | 0             | 0         | 0                | 0               | 50          |

Table 2.2: Confusion matrix of Architecture 1 on Train data

|                    | Pred car side | Pred ewer | Pred grand piano | Pred helicopter | Pred laptop |
|--------------------|---------------|-----------|------------------|-----------------|-------------|
| Actual car side    | 10            | 0         | 0                | 0               | 0           |
| Actual ewer        | 0             | 8         | 1                | 0               | 1           |
| Actual grand piano | 2             | 0         | 7                | 1               | 0           |
| Actual helicopter  | 0             | 0         | 1                | 9               | 0           |
| Actual laptop      | 0             | 1         | 0                | 0               | 9           |

Table 2.3: Confusion matrix of Validation data

## 2.2 Architecture 2

The metrics measured for architecture 2 are shown in Table 2.4. The corresponding confusion matrices are given in Table 2.5 and 2.6.

| Data           | Loss   | Accuracy |
|----------------|--------|----------|
| Training set   | 0.2018 | 0.9320   |
| Validation set | 1.0138 | 0.7000   |
| Test set       | 1.0630 | 0.6700   |

Table 2.4: Architecture 2 metrics

|                    | Pred car side | Pred ewer | Pred grand piano | Pred helicopter | Pred laptop |
|--------------------|---------------|-----------|------------------|-----------------|-------------|
| Actual car side    | 48            | 1         | 0                | 1               | 0           |
| Actual ewer        | 0             | 50        | 0                | 0               | 0           |
| Actual grand piano | 0             | 0         | 50               | 0               | 0           |
| Actual helicopter  | 1             | 1         | 0                | 48              | 0           |
| Actual laptop      | 1             | 6         | 0                | 6               | 37          |

Table 2.5: Confusion matrix of Architecture 2 on Train data

|                    | Pred car side | Pred ewer | Pred grand piano | Pred helicopter | Pred laptop |
|--------------------|---------------|-----------|------------------|-----------------|-------------|
| Actual car side    | 10            | 0         | 0                | 0               | 0           |
| Actual ewer        | 0             | 9         | 1                | 0               | 0           |
| Actual grand piano | 2             | 1         | 6                | 0               | 1           |
| Actual helicopter  | 0             | 2         | 0                | 8               | 0           |
| Actual laptop      | 3             | 3         | 0                | 2               | 2           |

Table 2.6: Confusion matrix of Architecture 2 on Validation data

## 2.3 Architecture 3

The metrics measured for architecture 3 are shown in Table 2.7. The corresponding confusion matrices are given in Table 2.8 and 2.9.

| Data           | Loss   | Accuracy |
|----------------|--------|----------|
| Training set   | 0.0008 | 1.0000   |
| Validation set | 1.0095 | 0.8000   |
| Test set       | 1.1839 | 0.8000   |

Table 2.7: Architecture 3 metrics

|                    | Pred car side | Pred ewer | Pred grand piano | Pred helicopter | Pred laptop |
|--------------------|---------------|-----------|------------------|-----------------|-------------|
| Actual car side    | 50            | 0         | 0                | 0               | 0           |
| Actual ewer        | 0             | 50        | 0                | 0               | 0           |
| Actual grand piano | 0             | 0         | 50               | 0               | 0           |
| Actual helicopter  | 0             | 0         | 0                | 50              | 0           |
| Actual laptop      | 0             | 0         | 0                | 0               | 50          |

Table 2.8: Confusion matrix of Architecture 3 on Train data

|                    | Pred car side | Pred ewer | Pred grand piano | Pred helicopter | Pred laptop |
|--------------------|---------------|-----------|------------------|-----------------|-------------|
| Actual car side    | 10            | 0         | 0                | 0               | 0           |
| Actual ewer        | 0             | 9         | 0                | 1               | 0           |
| Actual grand piano | 1             | 1         | 6                | 1               | 1           |
| Actual helicopter  | 0             | 1         | 0                | 9               | 0           |
| Actual laptop      | 2             | 2         | 0                | 0               | 6           |

Table 2.9: Confusion matrix of Architecture 3 on Validation data



## 2.4 Best Architecture: Architecture 1

Architecture 1 was decided to be the best architecture among the three architecture. Table 2.11 shows the image that was passed through the network to observe the learned filters and the resultant feature maps obtained after each convolutional layer. Table 2.12 shows the patch from the network that activates a neuron maximally at the last convolutional layer.

|                           | Pred car side | Pred ewer | Pred grand piano | Pred helicopter | Pred laptop |
|---------------------------|---------------|-----------|------------------|-----------------|-------------|
| <b>Actual car side</b>    | 18            | 0         | 0                | 1               | 1           |
| <b>Actual ewer</b>        | 0             | 16        | 0                | 2               | 2           |
| <b>Actual grand piano</b> | 1             | 0         | 16               | 1               | 2           |
| <b>Actual helicopter</b>  | 0             | 0         | 2                | 15              | 3           |
| <b>Actual laptop</b>      | 0             | 4         | 0                | 1               | 15          |

Table 2.10: Confusion matrix of Architecture 1 on Test data

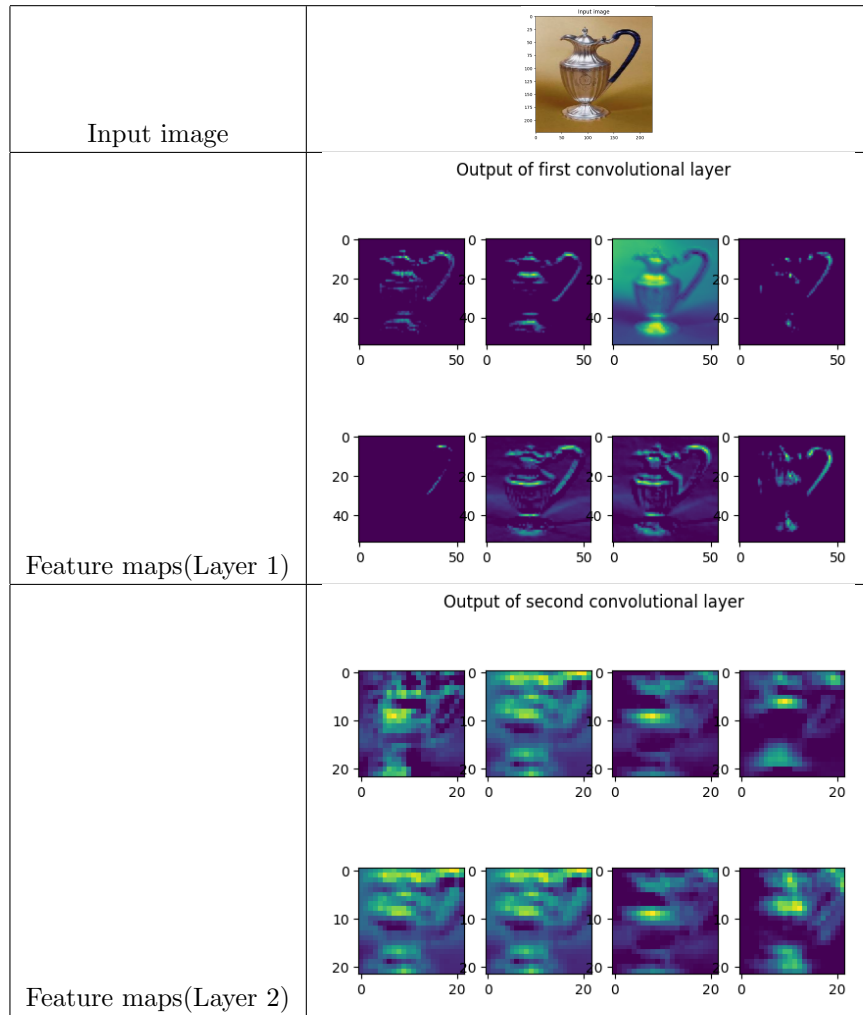


Table 2.11: Feature maps given by the best architecture(Architecture 3)



|                    |  |
|--------------------|--|
| Input images       |  |
| Patches activating |  |

Table 2.12: Patches from images activating a neuron in the last layer of architecture 1

## 2.5 Using pretrained VGG19 to make CNN

|                     |        |
|---------------------|--------|
| Train Accuracy      | 1.0000 |
| Validation accuracy | 1.0000 |
| Test accuracy       | 1.0000 |

Table 2.13: Metrics of using pretrained model

|                    | Pred car side | Pred ewer | Pred grand piano | Pred helicopter | Pred laptop |
|--------------------|---------------|-----------|------------------|-----------------|-------------|
| Actual car side    | 20            | 0         | 0                | 0               | 0           |
| Actual ewer        | 0             | 20        | 0                | 0               | 0           |
| Actual grand piano | 0             | 0         | 20               | 0               | 0           |
| Actual helicopter  | 0             | 0         | 0                | 20              | 0           |
| Actual laptop      | 0             | 0         | 0                | 0               | 20          |

Table 2.14: Confusion matrix of pretrained VGG19 on Test data



Figure 2.1: GradCAM algorithm showing heatmaps of regions of images responsible for the network detecting that class



|                    |  |
|--------------------|--|
| Input images       |  |
| Patches activating |  |

Table 2.15: Patches from images activating a neuron in the last layer of VGG19


























| Class       | Guided backpropagation from neuron (10, 14, 512), (7, 14, 512), (6,14, 512), (2,14, 512), (13,14, 512) |   |   |  |   |
|-------------|--|---|---|--|---|
| Grand piano |                       |    |    |    |    |
| Car         |                       |    |    |    |    |
| Helicopter  |                       |    |    |    |    |
| Vase        |                      |   |   |   |   |
| Laptop      |                     |  |  |  |  |

Table 2.16: Guided backpropagation of VGG19 on class images

## 2.6 Observations

- **Edge detection:** One of the earliest layers in a CNN typically learns to detect edges and other low-level features such as corners and blobs. These features can be seen in the corresponding feature maps as regions of high activation where the edges or corners are present in the input image.
- **Increasing complexity:** As we move deeper into the network, the features become more complex and abstract. For example, in a CNN trained on image recognition, we might see feature maps that respond to certain textures, shapes, or even entire objects.
- **Spatial invariance:** Because of the pooling layers that are often used in CNNs, the feature maps become increasingly invariant to small translations and rotations of the input image. This means that the same feature map can be used to detect an object in different parts of the image, which is a useful property for image recognition.
- **Visualizing activation patterns:** By visualizing the feature maps, we can gain insights into what parts of the input image are most important for a particular classification decision. For example, we might see that a certain feature map activates strongly when a dog is present in the image, indicating that the network has learned to detect certain dog-like features.

- Importance of high-contrast regions: Guided backpropagation is a technique used to visualize which parts of an input image are most important for a particular output of a neural network. When visualizing guided backpropagation, we might observe that the regions of the image that have the highest contrast (i.e. the greatest difference in intensity between neighboring pixels) tend to have the highest activation values. This suggests that the network is paying particular attention to these high-contrast regions when making its classification decision.
- Localization of object features: Another observation we might make from guided backpropagation is that the technique can help us to localize object features within an image. By visualizing which parts of the input image contribute most strongly to the network's classification output, we can identify which regions of the image correspond to different parts of the object. By identifying these regions, we can gain insights into what features the network is using to make its classification decisions.
- Transfer learning can reduce the amount of training data required to achieve good performance on a new task. Pre-trained models have already learned useful features from a large dataset that can be transferred to the new task.
- The GradCAM algorithm can identify important regions of an input image for a CNN's classification decision by producing a heatmap using gradient information of the final layer.