A Report on

CS671: Assignment 5

Convolutional Neural Networks(CNN)

Team Members(Group 04):

Ananya Angra(S22025)
Shilpa Chandra(S22004)
Pushkar Kumar(S22038)

Course Instructor

Dr. Dileep A. D.

IIT Mandi Academic Year 2022-2023

Contents

Chapter 1

Convolutional Neural Networks(CNN)

1.1 Convolution Operation

It is defined as the integral of the product of the two functions after one is reversed and shifted. The integral is evaluated for all values of shift, producing the convolution function. In Discrete convolution – Integral becomes summation. For a given input we re-estimate it as the weighted average of all the inputs around it. We have some weights assigned to the neighbor values and we take the weighted sum of the neighbor values to estimate the value of the current input/pixel. For 2D case there is a matrix of weights and it is referred to as the Kernel or Filter.

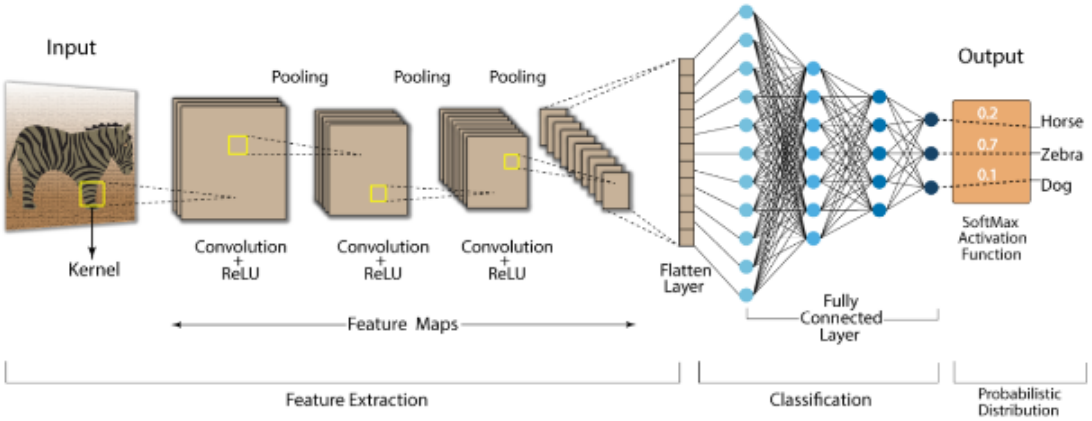


Figure 1.1: Convolutional Neural Network Architecture[1]

1.2 What is CNN

A Convolutional Neural Network (CNN) is a type of deep neural network that is commonly used in image and video recognition tasks. It is inspired by the structure and function of the visual cortex in the brain, which processes visual information by detecting patterns and features in an image. It consists of multiple layers, each of which performs a specific operation on the input data. The three main types of layers in a CNN are the convolutional layer, pooling layer, and fully connected layer. The convolutional layer is the core building block of a CNN. It applies a set of filters to the input image, each of which extracts a specific feature from the image. The filter is essentially a small matrix of weights that is convolved over the input image to produce an output feature map. The size of the filter is a hyperparameter that can be tuned to achieve different levels of abstraction in the extracted features. The pooling layer is used to downsample the output of the convolutional layer. It reduces the spatial size of the feature maps, while retaining the most important information. The most commonly used pooling operation is max pooling, which selects the maximum value in each sub-region of the feature map.

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$
$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$
$$D_2 = K$$

Figure 1.2: Relation between Input, Filter and Output[3]

- F = Size of filter
- S = Stride
- P = Pad
- W1 = Width of the input
- H1 = Height of the input

The fully connected layer is the final layer of the CNN, which performs the classification task. It takes the output of the previous layers and maps it

to a set of output classes using a set of weights and biases. The number of nodes in the fully connected layer corresponds to the number of output classes. During training, the CNN learns the optimal weights and biases for each layer using backpropagation.

1.3 How CNN is different from a regular FCNN

CNN is designed to process data with a grid-like topology such as images. It has convolutional layers that extract features from the input data, and pooling layers that reduce the dimensionality of the data. FCNN, on the other hand, stands for Fully Connected Neural Network and is designed to process data in a one-dimensional format such as a sequence of data points. It has fully connected layers that process the entire input at once. CNNs use pooling layers to reduce the dimensionality of the data, which helps to reduce overfitting and make the network more efficient. FCNNs do not have pooling layers and process the entire input at once, which can lead to overfitting if the data set is too large. To solve this problem, we use regularization in FCNN. CNNs process input data by using convolutional filters that scan the input data and extract local features. These local features are then combined to extract global features that are used to make a prediction. FCNNs process input data by using fully connected layers that process the entire input at once. They are often used for tasks such as speech recognition, where the input data is a sequence of audio samples.

1.4 Weight sharing

Weight sharing is a technique used in CNNs to reduce the number of parameters in the network and improve its ability to generalize to new data. In CNNs, weight sharing refers to the practice of using the same filter (set of weights) to process different regions of the input data. The idea behind weight sharing is that the same feature detectors can be used to detect the same pattern regardless of where it occurs in the input data. It is a trick way to connect each layer of convolution together obviously we aim to detect all segments of features by convolution layers in such a way that in layer one of convolution we detect the point of for example images are our input then in layer2 of convolution we discover the edges between these points after that we can discover partially the content of images in deep layers and so on.

1.5 Structure of CNN

It consists of 3 types of layers:

Convolutional Layer:

They are in charge of executing convolution operations. The Kernel/Filter performs the convolution operation. Until the complete image is scanned, the kernel makes horizontal and vertical adjustments dependent on the stride rate. The output obtained after performing convolution is passed through a non-linear activation function. The rectified linear unit (ReLU) is now the most commonly used non-linear activation function. f(x)=max(0,x)

Pooling Layer:

A pooling layer is a type of layer used in CNNs to reduce the spatial dimensions (width and height) of the input data while retaining its essential features. Pooling layers are typically placed after convolutional layers in a CNN and are used to reduce the computational cost of the network, prevent overfitting, and improve its ability to generalize to new data. It works by dividing the input data into a set of non-overlapping regions (called pooling windows) and then computing a summary statistic for each region. The most common types of pooling operations are max pooling and average pooling. Max Pooling is a pooling operation that calculates the maximum value for patches of a feature map, and uses it to create a downsampled feature map. It is usually used after a convolutional layer. It adds a small amount of translation invariance - meaning translating the image by a small amount does not significantly affect the values of most pooled outputs.

Fully Connected Layer:

The fully connected layer (FC) works with a flattened input, which means that each input is coupled to every neuron. After that, the flattened vector is sent via a few additional FC layers, where the mathematical functional operations are normally performed. The classification procedure gets started at this point. FC layers are frequently found near the end of CNN architectures if they are present.

1.6 Backpropagation Learning in CNN

- In CNN, Backpropagation Algorithm works by adjusting the weights of the connections between the neurons in the network in order to minimize the error. This is done by propagating the errors from the output layer back to the input layer.
- The algorithm works by calculating the error between the actual output and the desired output. The error is then propagated back through the network, and the weights of the connections between the neurons are adjusted accordingly.
- This process is repeated until the error is minimized and the network is able to accurately predict the desired output.

1.7 Transfer Learning

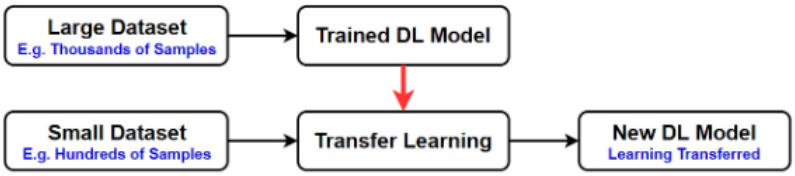


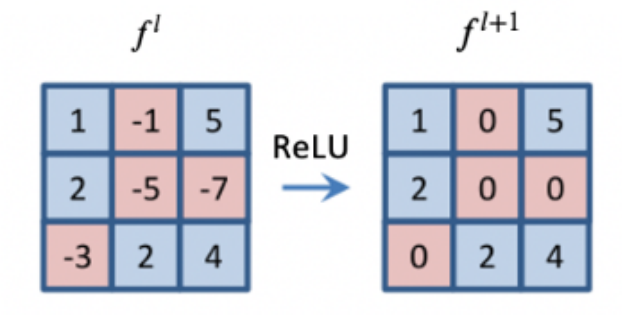
Figure 1.3: Steps for Transfer Learning[2]

- Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task.
- The basic idea behind transfer learning is to leverage the knowledge learned by a pre-trained model on a large dataset to improve the performance of a new model on a smaller dataset.
- In transfer learning, the pre-trained model is typically a CNN that has been trained on a large-scale dataset such as ImageNet, which contains millions of images belonging to thousands of categories. This pre-trained model has learned to extract relevant features from images that can be useful for other image classification tasks.

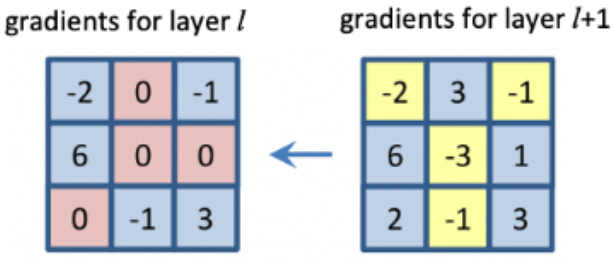
- To use transfer learning in a new CNN, one can take the pre-trained model and replace the final layer with new layers that are specific to the new classification task. These new layers are then trained using the smaller dataset, while the pre-trained layers are frozen and used as fixed feature extractors.

1.8 Guided Backpropagation

- Guided Backpropagation is a technique used to generate visual explanations of the decision-making process of a convolutional neural network (CNN) by highlighting the important regions of an input image that contributed to the network's output.
- The idea behind Guided Backpropagation is to generate a saliency map that highlights the regions of an input image that are most responsible for the network's output. This can be used to provide a visual explanation for the network's decision, making it easier to understand and interpret the model's predictions.
- The Guided Backpropagation algorithm is implemented by modifying the standard backpropagation algorithm. During the forward pass, only the positive activations are allowed to propagate forward, while the negative activations are set to zero.
- During the backward pass, only the positive gradients are allowed to propagate back, while the negative gradients are also set to zero. The resulting guided gradients are then used to generate a saliency map that highlights the regions of the input image that have the highest positive impact on the network's output.



(a) Fig: Forward Pass



(b) Fig: Backward Pass

Figure 1.4: Guided BackPropagation[4]

1.9 Grad-CAM

Grad-CAM (Gradient-weighted Class Activation Mapping) is a visualization technique that highlights the regions of an image that a convolutional neural network uses to make its prediction. It provides a visual explanation for the CNN's decision-making process, making it easier to understand and interpret the model's outputs. This technique uses the gradients of the output class score with respect to the feature maps of the last convolutional layer in the CNN. These gradients are then used to weight the activations of the feature maps, giving higher weights to regions that are more important for the prediction. The resulting weighted feature maps are then averaged spatially to obtain a heat map that highlights the regions of the input image that are most relevant for the CNN's prediction. It can be applied to any CNN-based image classification model, and it does not require any modifications to the architecture of the model. It is a post-hoc technique that can be used to visualize the decision-making process of the model without requiring any additional training or data. Overall, Grad-CAM is a powerful tool for visualizing and interpreting the outputs of CNNs, making it easier to understand and trust the decisions made by these models.

1.10 VGG19

VGG19 has 19 layers, including 16 convolutional layers and 3 fully connected layers. Each convolutional layer in VGG19 uses a 3x3 filter with a stride of 1, and is followed by a rectified linear unit (ReLU) activation function. The max-pooling layers use a 2x2 filter with a stride of 2, which reduces the spatial dimensions of the feature maps. The fully connected layers in VGG19 are used to combine the features extracted from the convolutional layers into a high-level representation of the input image, which is then used for classification. The output layer of the network uses a softmax activation function to produce a probability distribution over the possible classes. It has achieved high accuracy on image classification tasks such as the ImageNet dataset, which contains over 1 million images and 1000 different object categories. Its performance has been shown to be comparable to or even better than more complex models, making it a reliable choice for classification tasks. The pre-trained weights for VGG19 are also widely available online,

making it easy for researchers and developers to use and experiment with the architecture.

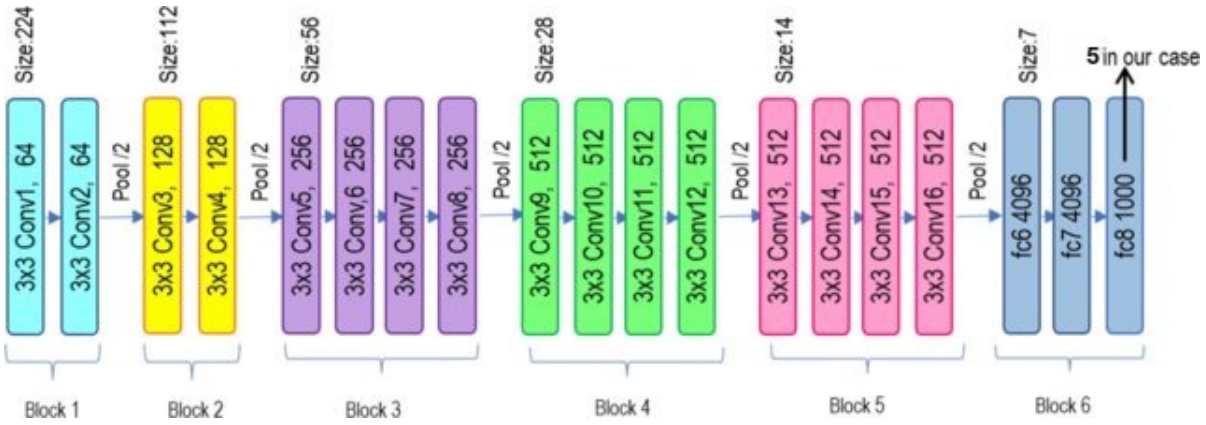


Figure 1.5: VGG19 Architecture[5]

Chapter 2

Results: Question 1

About the Dataset:

- **A subset of Caltech-101 dataset for the image classification is used.**
- **We are given 5 classes: Buddha , chandelier , hawksbill , ketch , menorah**
- **Note: In our comparisons sometimes we have referred to Buddha , chandelier , hawksbill , ketch , menorah as Class 1, Class2, Class3, Class4 and Class5 respectively.**

Given dataset is train-validation-test separated.

2.1 Description of Architectures

2.1.1 Architectures 1:

- **1st convolutional layer-** 8 - 11x11 filters , stride 4, padding 0. Followed by -
- **3x3 max pooling operation with stride 2.**
- **2nd convolutional layer-** with 16 5x5 filters, stride 1 , padding 0. Followed by -
- **3x3 max pooling operation with stride 2. Then flatten it. Then two fully connected layers are added.**
  - 128 hidden nodes with rectified linear activation function in the first hidden layer
  - 5 neurons with a softmax activation function in the second layer (output layer).

2.1.2 Architectures 2:

- **First 2 convolutional layers are same as Architecture 1.**
- **3rd convolutional layer-** 32 3x3 filters with stride 1 and padding 0 followed by 3x3 max pooling operation with stride 2. Then flatten it. **Then two fully connected layers are added.**
  - 128 hidden nodes with rectified linear activation function in the first hidden layer
  - 5 neurons with a softmax activation function in the second layer (output layer).

2.1.3 Architectures 3:

- **First 2 convolutional layers are same as Architecture 1.**
- **3rd convolutional layer-** 32 3x3 filters with with stride 1 and padding 0 and no max pooling operation.
- **4th convolutional layer-** 64 3x3 filters with stride 1 and padding 0 Followed by -
- **3x3 max pooling operation with stride 2.**
  Then two fully connected layers are added.
  - 128 hidden nodes with rectified linear activation function in the first hidden layer
  - 5 neurons with a softmax activation function in the second layer (output layer).

Architecture 1	
Layer Name	Output Shape
Convolutional layer 1	54x54x8
Max pooling layer 1	26x26x8
Convolutional layer 2	22x22x16
Max pooling layer 2	10x10x16
Flatten layer	1D tensor of size 1600
Architecture 2 (First two convolutional layers same as Architecture 1)	
Convolutional layer 3	10x10x32
Max pooling layer 3	4x4x32
Flatten layer	1D tensor of size 512
Architecture 3(First two convolutional layers same as Architecture 1)	
Convolutional layer 3	8x8x32
Convolutional layer 4	6x6x64
Max pooling layer 4	2x2x64
Flatten layer	1D tensor of size 256

Table 2.1: Size of the resulting feature maps at each layer of all Architectures

2.2 Results with transfer learning:

	Training Accuracy	Validation Accuracy
Architecture 1	0.7960000038146973	0.7400000095367432
Architecture 2	0.8900000166893005	0.7799999713897705
Architecture 3	0.8840000247955322	0.7599999904632568

Figure 2.1: Results after transfer learning

The best architecture based on validation accuracy is architecture 2.

		Predicted Label				
		Buddha	Chandelier	Hawksbill	Ketch	Menorah
True Label	Buddha	15	4	0	1	0
	Chandelier	3	15	0	2	0
	Hawksbill	3	1	15	0	1
	Ketch	4	2	0	14	0
	Menorah	0	1	2	1	16

Figure 2.2: The confusion matrix for best Architecture: Architecture 2 with transfer learning

Visualising Feature maps:

The following feature maps are based upon the best architecture based: which is architecture 2.

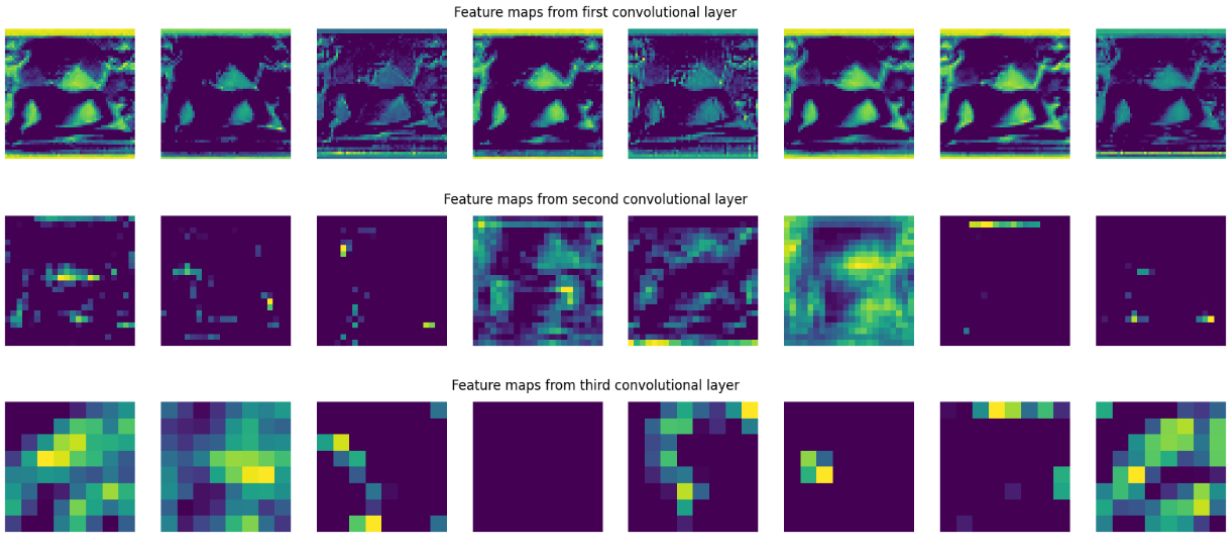


Figure 2.3: Feature maps from the three convoltuinal layers

In a convolutional neural network (CNN), the convolutional layers are responsible for learning features or patterns from the input data. Each convolutional layer typically learns a hierarchy of increasingly complex features by applying a set of filters (also known as kernels or weights) to the input image. As we start going deeper in the convolutional layers we see that:

- Fig 2.3, learns the low-level features such as edges, corners, and blobs.
- As we move deeper into the network, Fig 2.4, Fig 2.5 learn higher-level features such as shapes, textures, and object parts.

2.3 Results without transfer learning:

	Training Accuracy	Validation Accuracy
Architecture 1	0.8799999952316284	0.699999988079071
Architecture 2	0.8880000114440918	0.7200000286102295
Architecture 3	0.7080000042915344	0.6000000238418579

Figure 2.4: Results without transfer learning

The best architecture based on validation accuracy is architecture 2.

		Predicted Label				
		Buddha	Chandelier	Hawksbill	Ketch	Menorah
True Label	Buddha	11	8	1	0	0
	Chandelier	1	18	0	0	1
	Hawksbill	3	0	16	1	0
	Ketch	3	7	1	8	1
	Menorah	0	2	2	0	16

Figure 2.5: The confusion matrix for best Architecture: Architecture 2 with transfer learning

Visualising Feature maps:

The following feature maps are based upon the best architecture based: which is architecture 2.

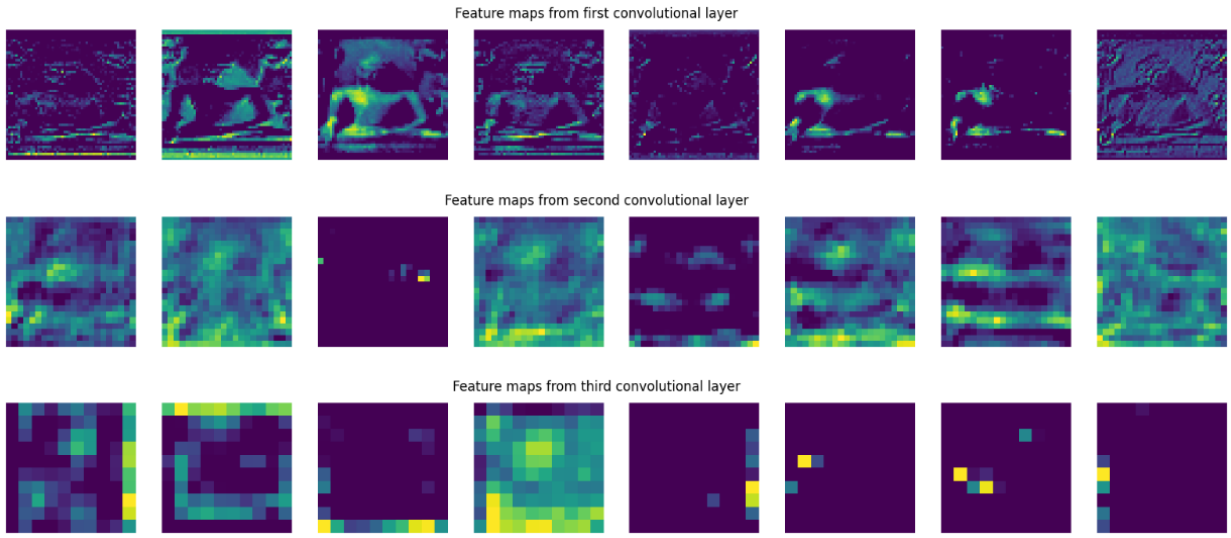


Figure 2.6: Feature maps from all three Convolutional Layer - when transfer learning is used

Same as results with transfer learning, as we start going deeper in the convolutional layers we see that:

- Fig 2.7, learns the low-level features such as edges, corners, and blobs.
- As we move deeper into the network, Fig 2.8, Fig 2.9 learn higher-level features such as shapes, textures, and object parts.

2.4 Visualize the patches in each of the images which maximally activate that neuron.

The following feature maps are based upon the best architecture based: which is architecture 2 with transfer learning. The patches are calculated after finding out the neuron in the last convolutional layer (for each image) that is maximally activated. And then tracing back to the patch in the image which causes these neurons to fire. This is done using back-calculating the feature area of the last convolutional layer responsible for firing that neuron, done using taking into account **strides,padding and kernel size**.

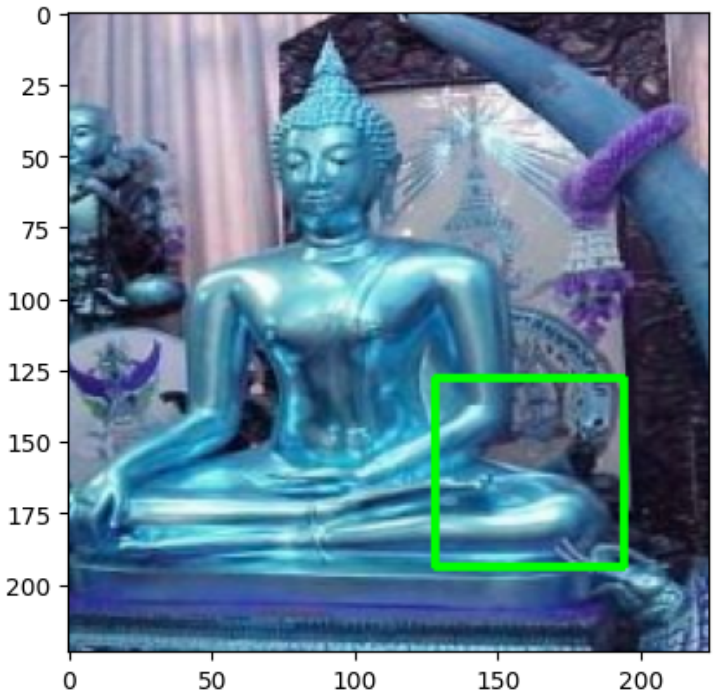


Figure 2.7: Class 'buddha': The neuron that is maximally activated in last layer is 73

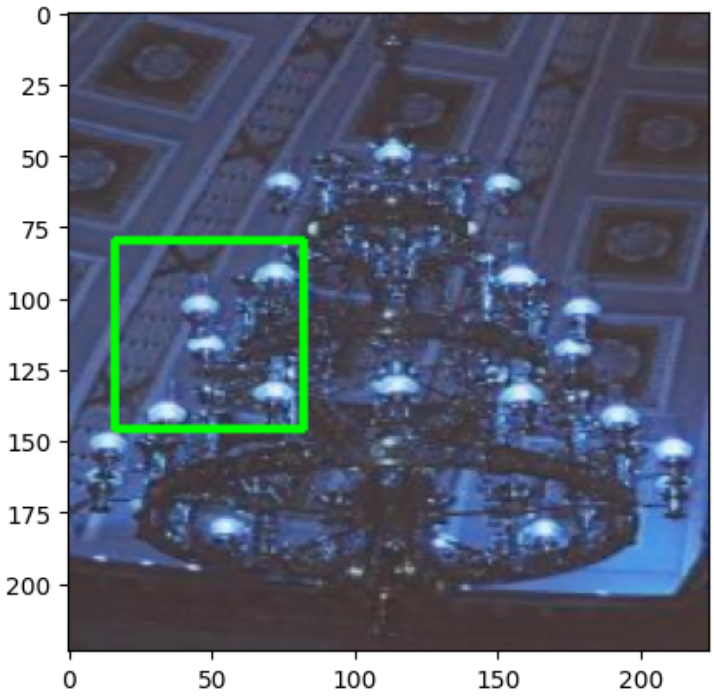


Figure 2.8: Class 'chandelier': The neuron that is maximally activated in last layer is 15

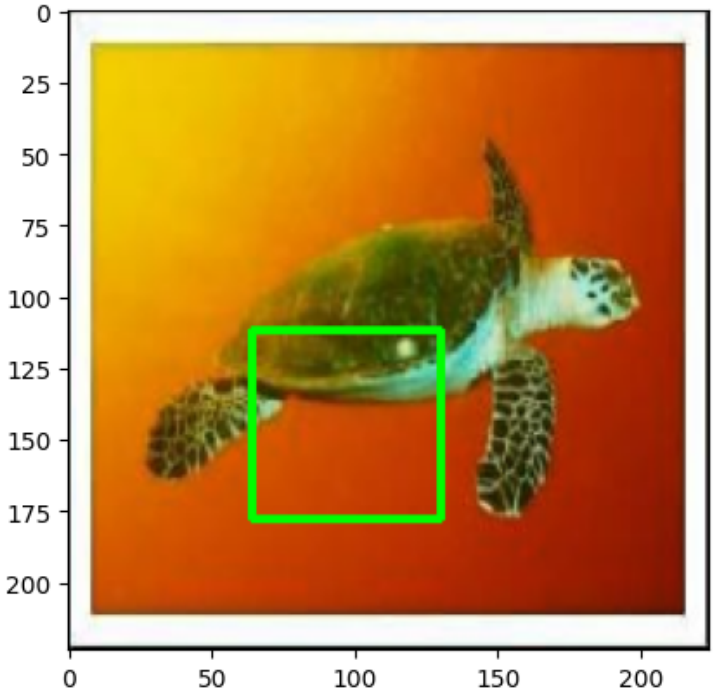


Figure 2.9: Class 'hawksbill': The neuron that is maximally activated in last layer is 47

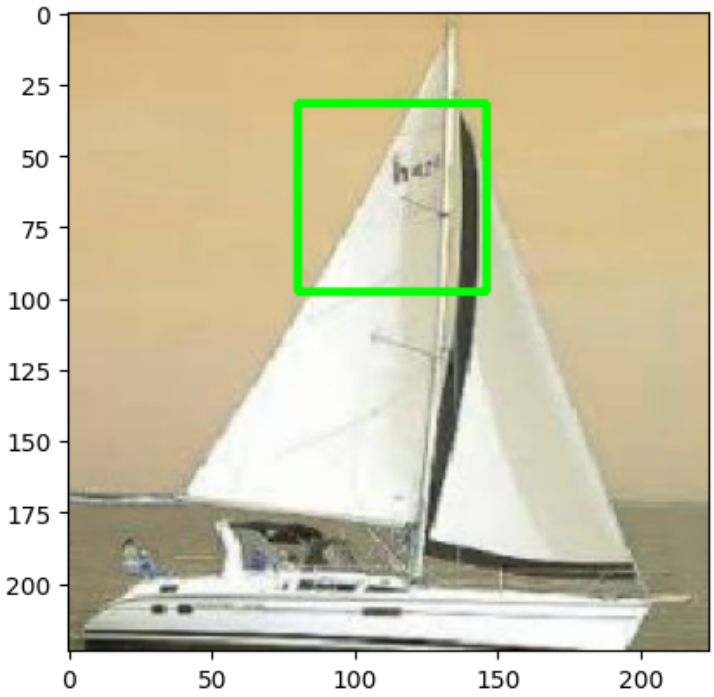


Figure 2.10: Class 'ketch': The neuron that is maximally activated in last layer is 52

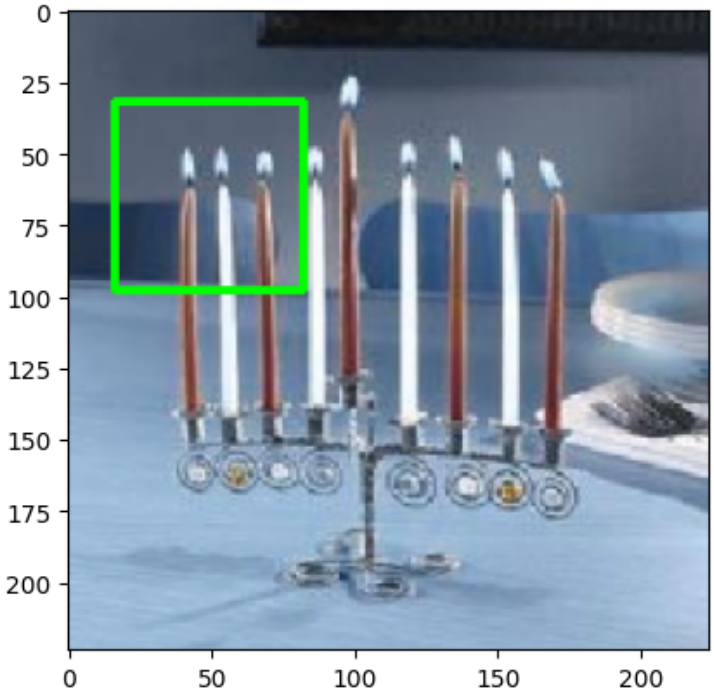


Figure 2.11: Class 'menorah': The neuron that is maximally activated in last layer is 12

Chapter 3

Results: Question 2

- We have used VGG19 model pretrained on ImageNet and also modified the classification layer of VGG19 to 5 output nodes to make a custom model for our 5 classes: Buddha , chandelier , hawksbill , ketch , menorah.
- Sometimes they are mentioned as: Class 1, Class2, Class3, Class4 and Class5 respectively in our report for simplicity.

Training, Validation and Testing accuracy for our dataset on our custom VGG19 model

Training Accuracy	Validation Accuracy	Test Accuracy
0.9879999756813049	0.9800000190734863	0.9800000190734863

Figure 3.1: Accuracy's on custom VGG19 model

Confusion Matrix on custom VGG19 model

		Predicted Label				
		Buddha	Chandelier	Hawksbill	Ketch	Menorah
True Label	Buddha	18	1	0	1	0
	Chandelier	0	20	0	0	0
	Hawksbill	0	0	20	0	0
	Ketch	0	0	0	20	0
	Menorah	0	0	0	0	20

Figure 3.2: Confusion Matrix on custom VGG19 model

Comparison with best architecture from 1b:

Architecture	Training Accuracy	Validation Accuracy	Test Accuracy
Architecture2(With transfer learning) - Q1)b	0.8900000166893005	0.7799999713897705	0.75
Custom VGG19 model	0.9879999756813049	0.9800000190734863	0.9800000190734863

Figure 3.3: Comparison with best architecture from 1b

- It can be observed from the above that training, validation and test accuracy obtained using VGG19 is much more as compared to the best architecture from 1b.
- This is because VGG19 is pre-trained using a large dataset and a model that has been pretrained on a large dataset can generalize well to new, unseen data. Also the model has learned general features and patterns that are applicable to a wide range of data.

- Also because VGG19 contains 19 layers for training.

Pretraining a model can reduce the time required to train it on a specific task. This is because the model has already learned some of the features and patterns required for the target task during pretraining.

3.1 Visualize the patches in each of the images which maximally activate that neuron.

The following feature maps are based upon our custom VGG19 model. The patches are calculated after finding out the neuron in the last convolutional layer (for each image) that is maximally activated. And then tracing back to the patch in the image which causes these neurons to fire. This is done using back-calculating the feature area of the last convolutional layer responsible for firing that neuron, done using taking into account strides,padding and kernel size.

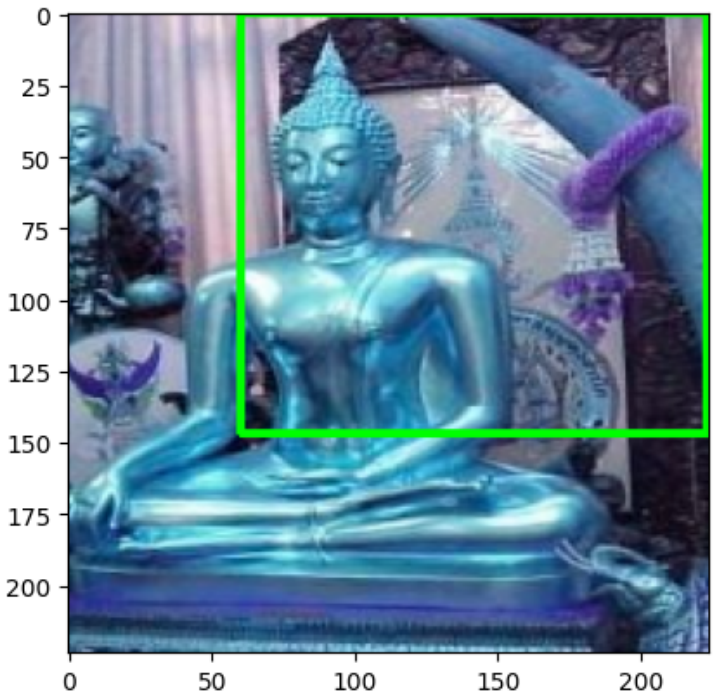


Figure 3.4: Class 'buddha': The neuron that is maximally activated in last layer is 129

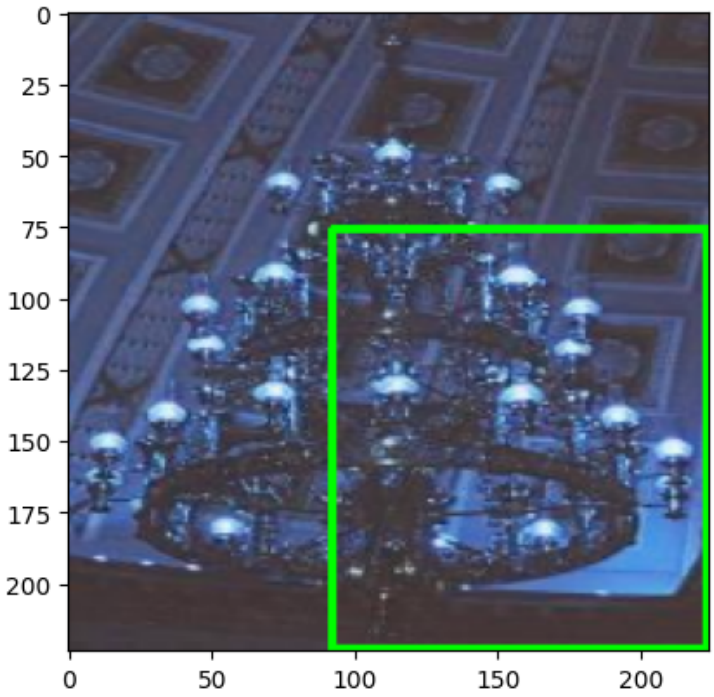


Figure 3.5: Class 'chandelier': The neuron that is maximally activated in last layer is 164

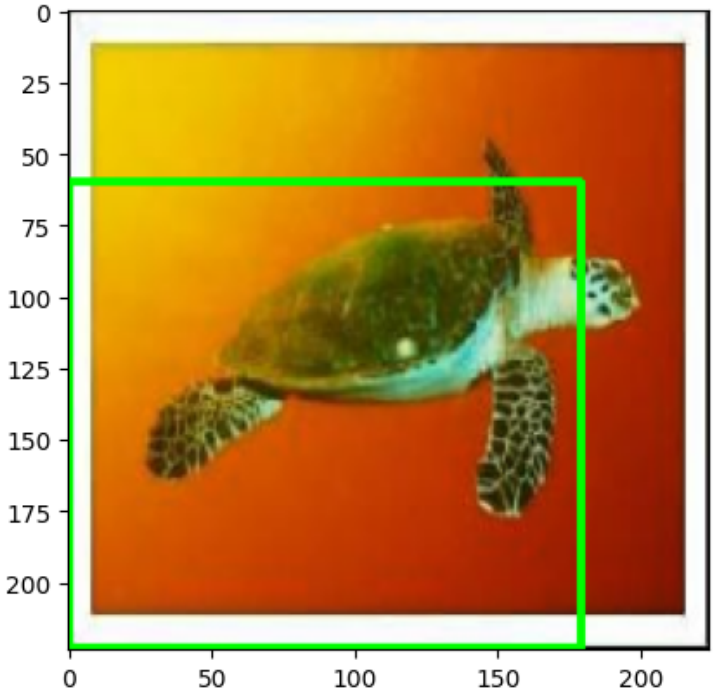


Figure 3.6: Class 'hawksbill': The neuron that is maximally activated in last layer is 79

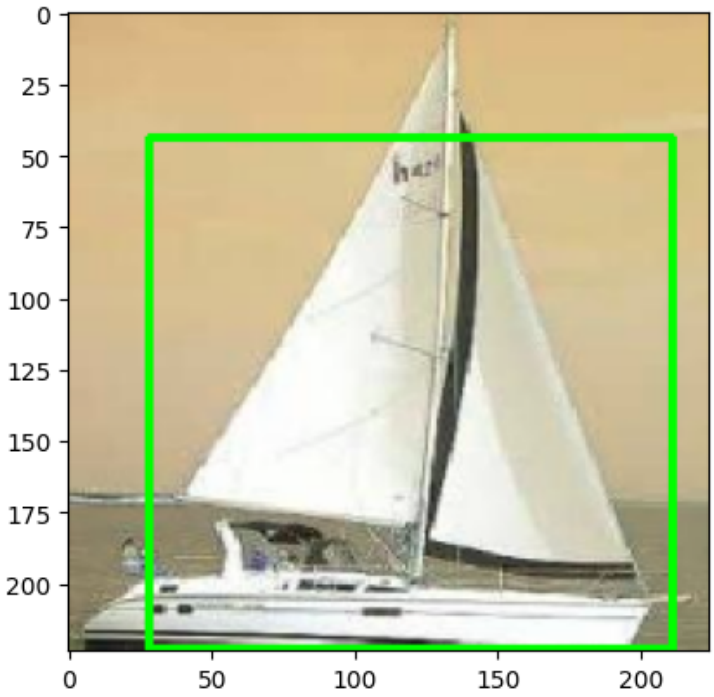


Figure 3.7: Class 'ketch': The neuron that is maximally activated in last layer is 106

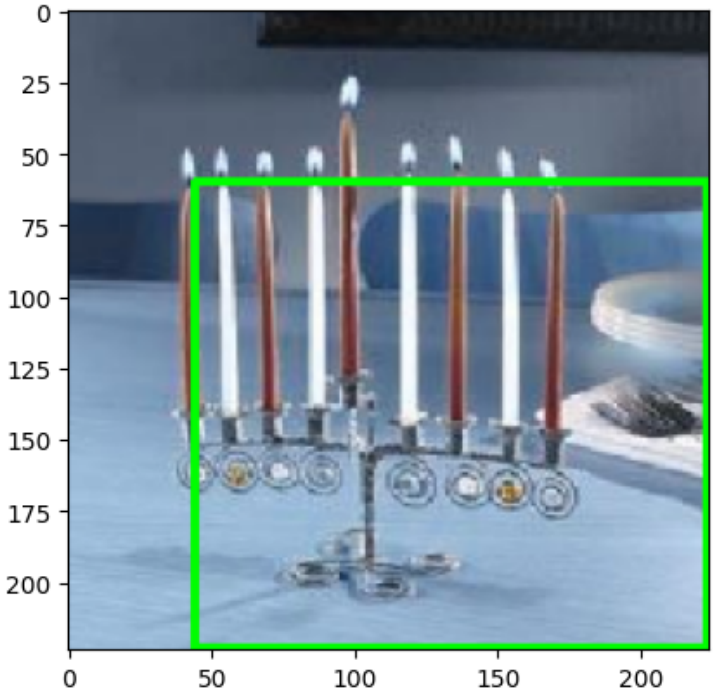


Figure 3.8: Class 'menorah': The neuron that is maximally activated in last layer is 121

3.2 Guided-backpropagation algorithm

Influence of input pixels on any of the 5 neurons in the last convolutional layer of our custom VGG19 model. Input image to the model is given in Fig 3.9

Neuron number	Resultant gradient images
0	
1	
6	
8	
11	

Table 3.1: Influence of input pixels on custom VGG19 model

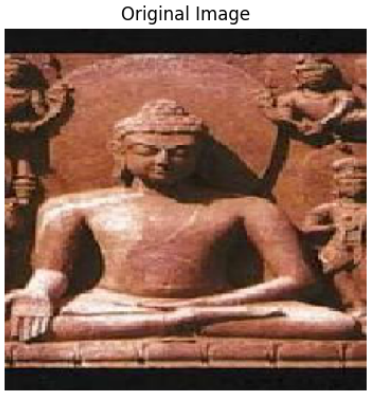


Figure 3.9: Input image to the custom VGG19 model

- Each neuron in the last layer is learning some different portions of the input image respectively(seen in Table 3.1).
- Guided backpropagation adds an additional rule to the backpropagation algorithm that only allows the positive gradients (gradients with a positive value) to backpropagate through the network while ignoring the negative gradients (gradients with a negative value). This ensures that the final visualization highlights only the regions of the input image that contribute positively to the output, while suppressing the regions that do not.
- In other words, guided backpropagation learns to identify the parts of an input image that are most relevant to the output of the neural network.

3.3 Visualizing Localization map using GradCam

Localization map (heat map) highlighting the important regions in the image for predicting its class and other classes

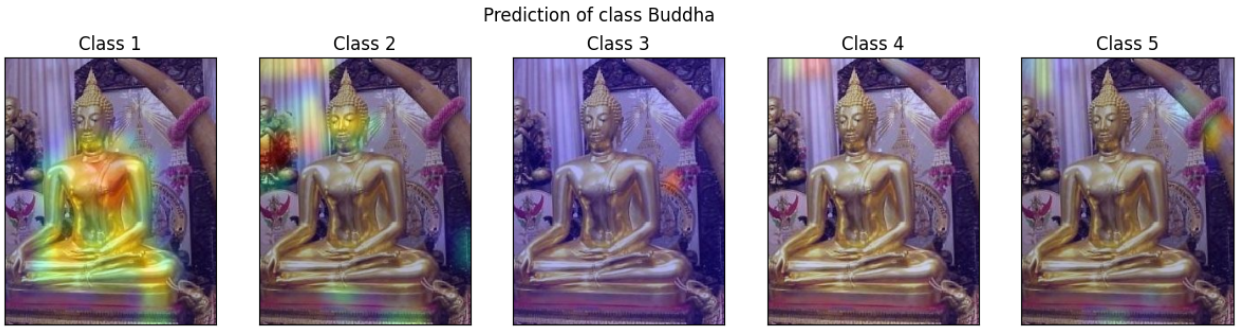


Figure 3.10: Heat maps of all classes when image belongs to class = 'buddha'/1

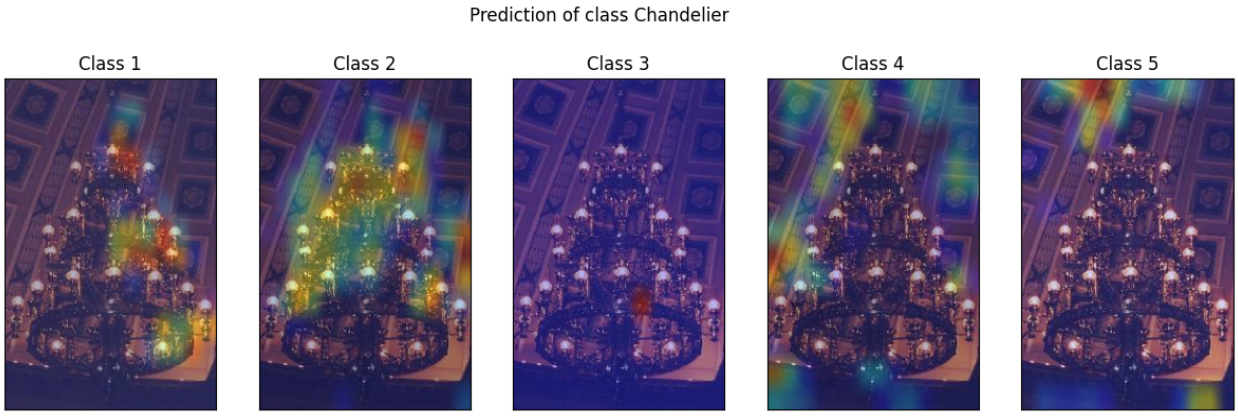


Figure 3.11: Heat maps of all classes when image belongs to class = 'chandelier'/2

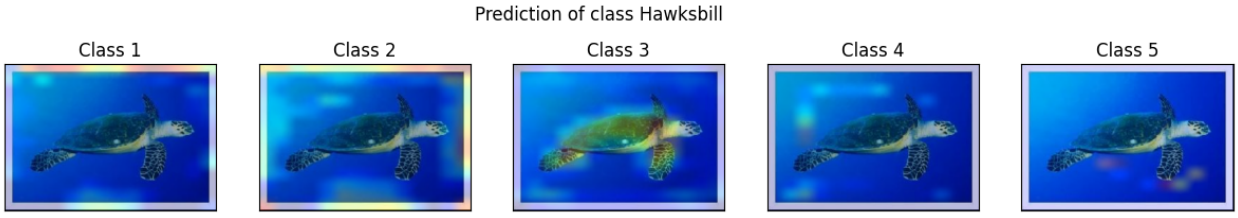


Figure 3.12: Heat maps of all classes when image belongs to class = 'hawksbill'/3

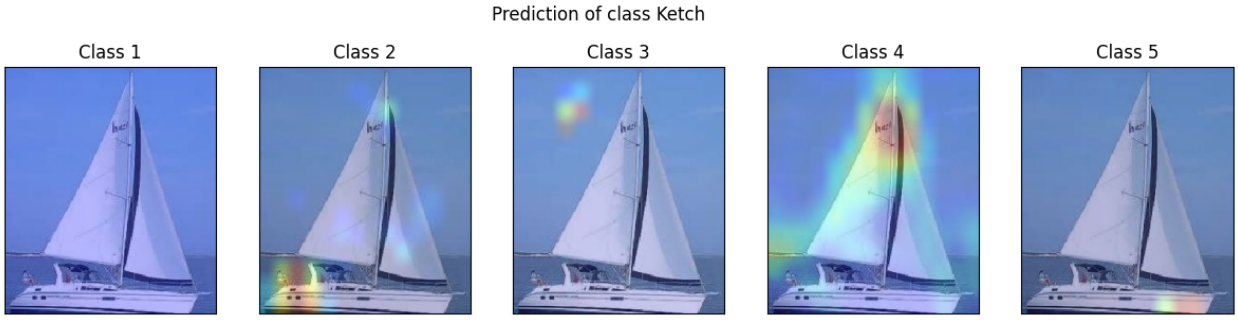


Figure 3.13: Heat maps of all classes when image belongs to class = 'ketch'/4

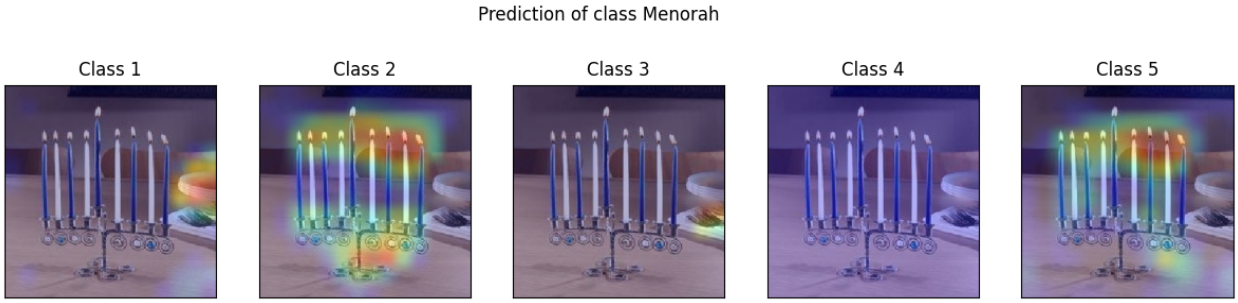


Figure 3.14: Heat maps of all classes when image belongs to class = 'menorah'/5

As we can see:

- In Fig 3.10, the heat map localizes mostly on 'buddha' correctly. But it also localizes some portion on 'chandelier' as the curtains in the image behind buddha kind of act as candles in 'chandelier'.
- In Fig 3.11 the heat map localizes mostly on 'chandelier' correctly. But it also localizes some portion on 'buddha' as texture matches the bun of 'buddha'.
- In Fig 3.12,3.13 the heat map localizes mostly on 'hawksbill' and 'ketch' classes correctly.
- In Fig 3.14, the heat map localizes both on 'menorah' and 'chandelier' mostly equally. Mostly because both have similar shape and patterns. So it's difficult to distinguish between the two.

References

[1] https://i0.wp.com/developersbreach.com/wp-content/uploads/2020/08/cnn_banner.pngfit=1200

[2] alibabacloud.com

[3] Lecture notes of CS671 Feb 2023-24 by Dr.Dileep A D

[4] https://leslietj.github.io/2020/07/22/Deep-Learning-Guided-BackPropagation/

[5] Yang, Jie Zhao, Junhong Lu, Lu Pan, Tingting Jubair, Sidra. (2020). A New Improved Learning Algorithm for Convolutional Neural Networks. Processes. 8. 295. 10.3390/pr8030295.