CS671 - Deep Learning and Applications

Assignment 3 : Visualizing Convolutional Neural Networks

ASSIGNMENT 3 REPORT

submitted by

Aditya Sarkar	Aaron Thomas Joseph	Srishti Ginjala
IIT Mandi	IIT Mandi	IIT Mandi
B19003	B19128	B19084

under the supervision of

Dr. Dileep AD

INDIAN INSTITUTE OF TECHNOLOGY, MANDI

May 2022

Abstract

In this programming assignment, we have improved our understanding of the convolution operation, convolutional neural networks and guided backpropagation visualizing them. The report starts by illustrating different output images obtained from convolution and correlation operations using a 3x3 filter which was initialised with Kaiming. Next two convolutional layers were developed with different number of filters and the feature maps obtained were observed. Using the above concepts, convolutional neural network was developed and training was performed on the images. We also visualized the patches in three images which are maximally activating the neurons. Next deep learning APIs were used to perform guided backpropagation to find the influence and observed the gradient images.
This report is made in LaTeX

Contents

Chapter 1

Convolution and Cross-Correlation Operation

1.1 Brief description of the dataset

We were given a subset of the Caltech-101 dataset consisting of three classes butterfly, kangaroo and sunflower. The Caltech-101 dataset consists of coloured images with varying sizes. The data consisted of three folders for training, test and validation.

1.2 Question 1

In the first question, we picked one image from each of the three classes and converted it into a single channel grayscale image as shown in Figure 1.1. Then we initialized a 3x3 convolutional filter using Kaiming initialization and traversed the convolution filter over all the pixels of the image with stride 1, padding 0 and obtained the final output as a feature map.

1.3 Convolution and Cross-Correlation

Cross-correlation and convolution are both operations applied to images. Cross-correlation means sliding a kernel (filter) across an image. Convolution means sliding a flipped kernel

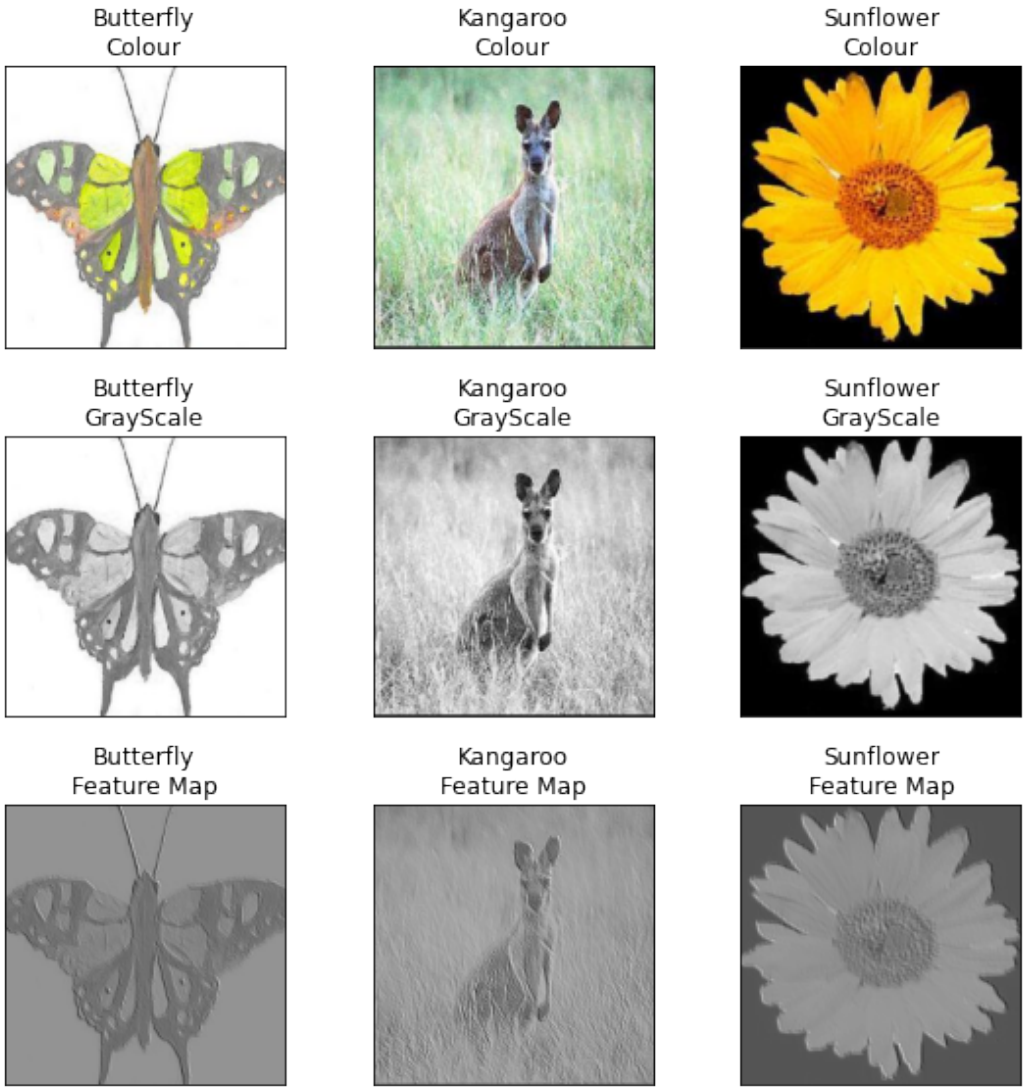

Fig. 1.1: Colour vs Grayscale vs Feature Map

across an image. Most inbuilt machine learning libraries for CNN are actually implemented using cross-correlation, but it doesn't change the results in actuality because if convolution were used instead, the same weight values would be learned in a flipped orientation.

1.4 Results and Observations

The convolutional filter was initialised by sampling a normal distribution with mean 0 and standard deviation $\sqrt{2/(224 * 224)}$. The values obtained for the 3x3 filter in one of the runs is shown in Figure 1.2.

The expected dimension of the feature map is given by the formula -

$$n^{'} = \frac{n + 2p - f}{s} + 1$$

where n is the dimension of the input vector, p is the padding, f is the filter size and s is the stride. Here in this question, on substituting **s=1, p=0, n=224 and f=3**, we expect the dimension of the feature map to be (224+0-3+1=222), which is also the obtained result. Hence we can conclude that we got the right result.

```
Convolutional Filter:
 [[ 0.00093  0.00477  0.00466]
 [-0.00691 -0.00194  0.00236]
 [ 0.00708 -0.01398  0.01168]]
```

Fig. 1.2: An instance of convolutional filter

1.4.1 Different initialization techniques

1.4.1.1 Constant value

If we initialize all the weights to the same value, then the activation values will be the same for the first hidden layer and hence during backpropagation all the weights in layer will get the same update and remain equal. These will repeat across all the layers and iterations and hence the symmetry never breaks during training. As a result, the model doesn't learn any features and performs poorly.So, this kind of strategy shouldn't be used.

1.4.1.2 Kaiming Distribution

We require the weights such that activation values will not blow up or shrink. For Rectified linear activation function we can use kaiming initialization where we draw the weights from zero mean normal distribution whose standard deviation is $\sqrt{2/(Nj)}$, where Nj is the number of incoming inconnections to a neuron in jth layer.

Chapter 2

Convolution layer

2.1 Brief description of the dataset

We were given a subset of the Caltech-101 dataset consisting of three classes butterfly, kangaroo and sunflower. The Caltech-101 dataset consists of coloured images with varying sizes. The data consisted of three folders for training, test and validation.

2.2 Observations

In this question, we have defined a class to implement the two convolutional layers having 32 and 64 3x3 filters. We have initialized convolutional filters using Kaiming initialization and passed images considered in the Question 1 in a feedforward manner to the two stacked layers. We have taken the stride to be 1 and padding 0.

The expected dimension of the feature map is given by the formula -

$$n^{'} = \frac{n + 2p - f}{s} + 1$$

where n is the dimension of the input vector, p is the padding, f is the filter size and s is the stride. Here in this question, on substituting **s=1, p=0, n=224 and f=3**, we expect the dimension of the feature map of the first layer to be (224+0-3+1=222), which is also the obtained result. Hence we can conclude that we got the right result. For the second hidden

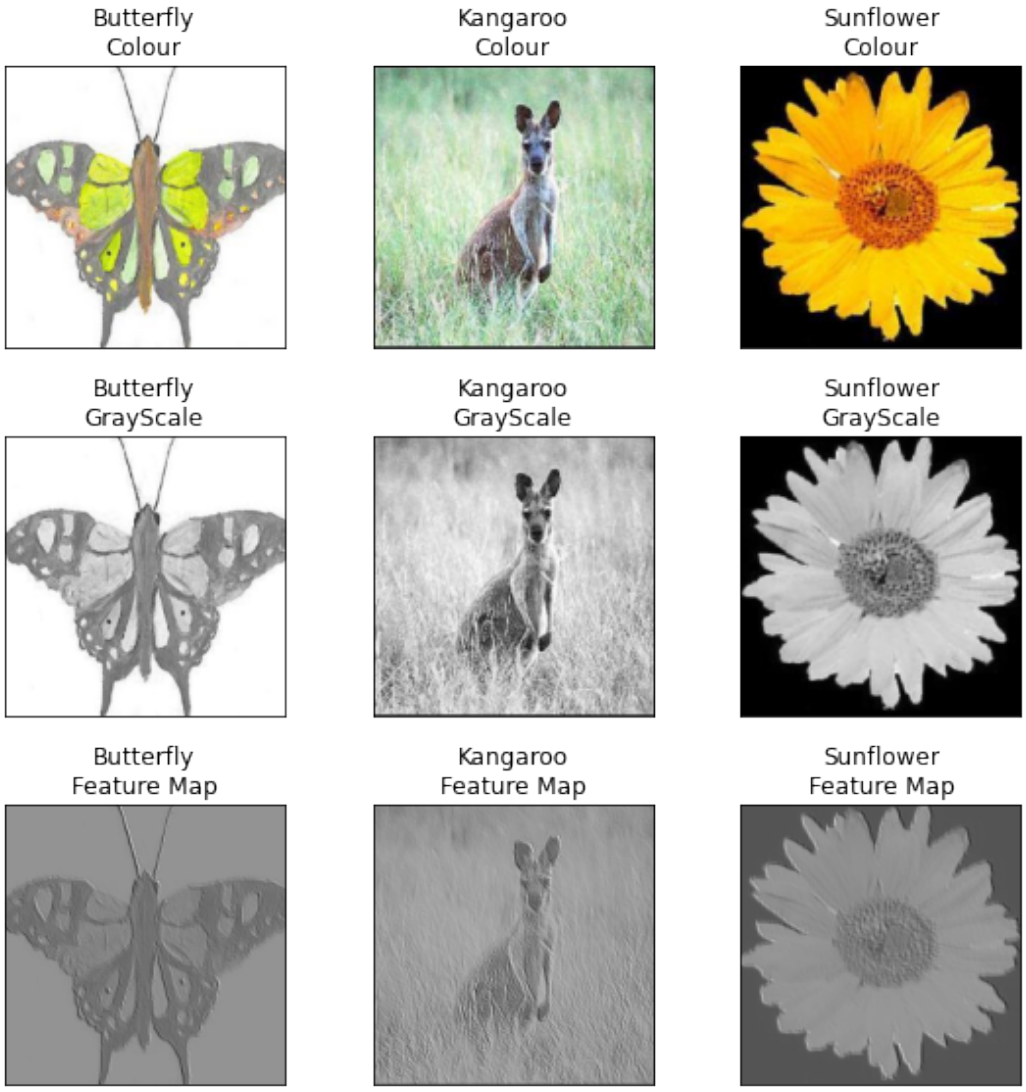

Fig. 2.1: Colour vs Grayscale

layer, n=222, we expect the dimension of the feature map to be (222+0-3+1=220), which is also the obtained result.

The 3 filters and corresponding feature maps for inputs of each of the three classes in both layers are shown below.

2.2.1 First Convolution Layer

In the first convolutional layer, the neurons will try to capture very basic features like lines, edges, boundaries etc.

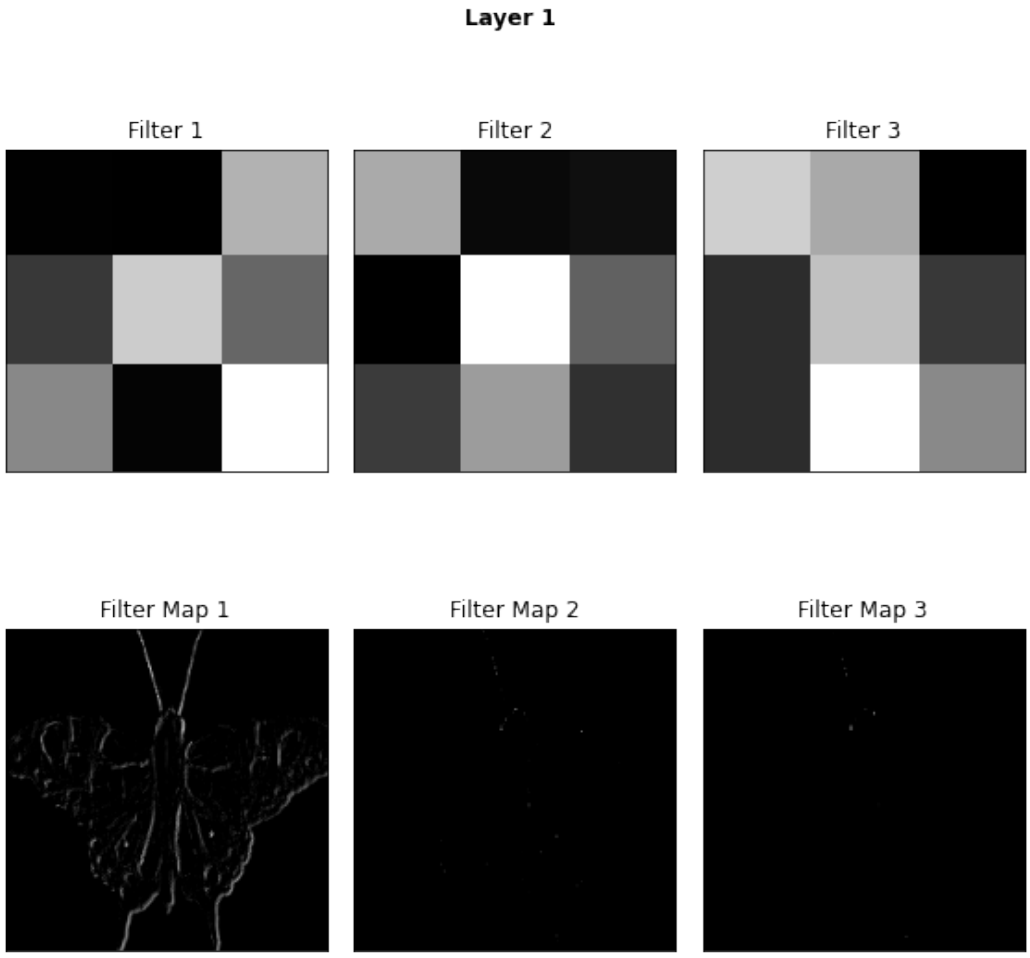

Fig. 2.2: Filter and feature map of butterfly in layer1

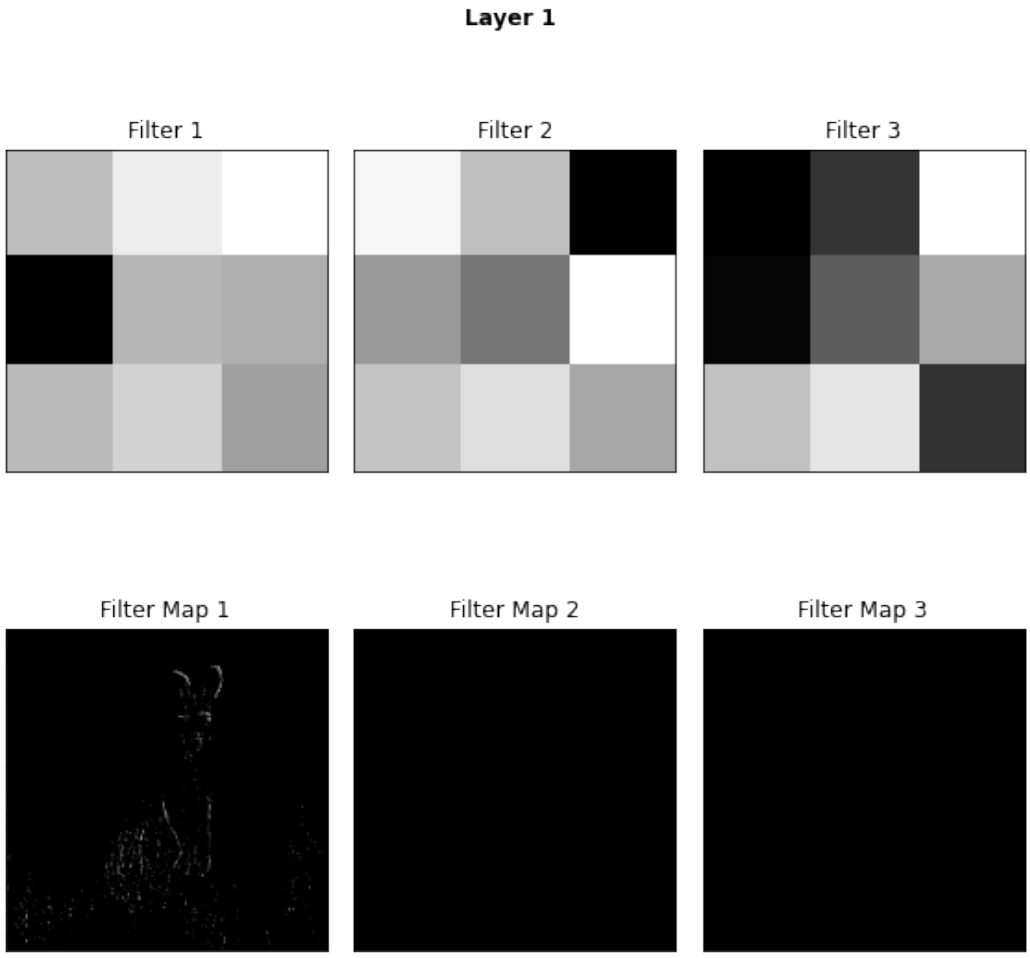

Fig. 2.3: Filter and feature map of kangaroo in layer1

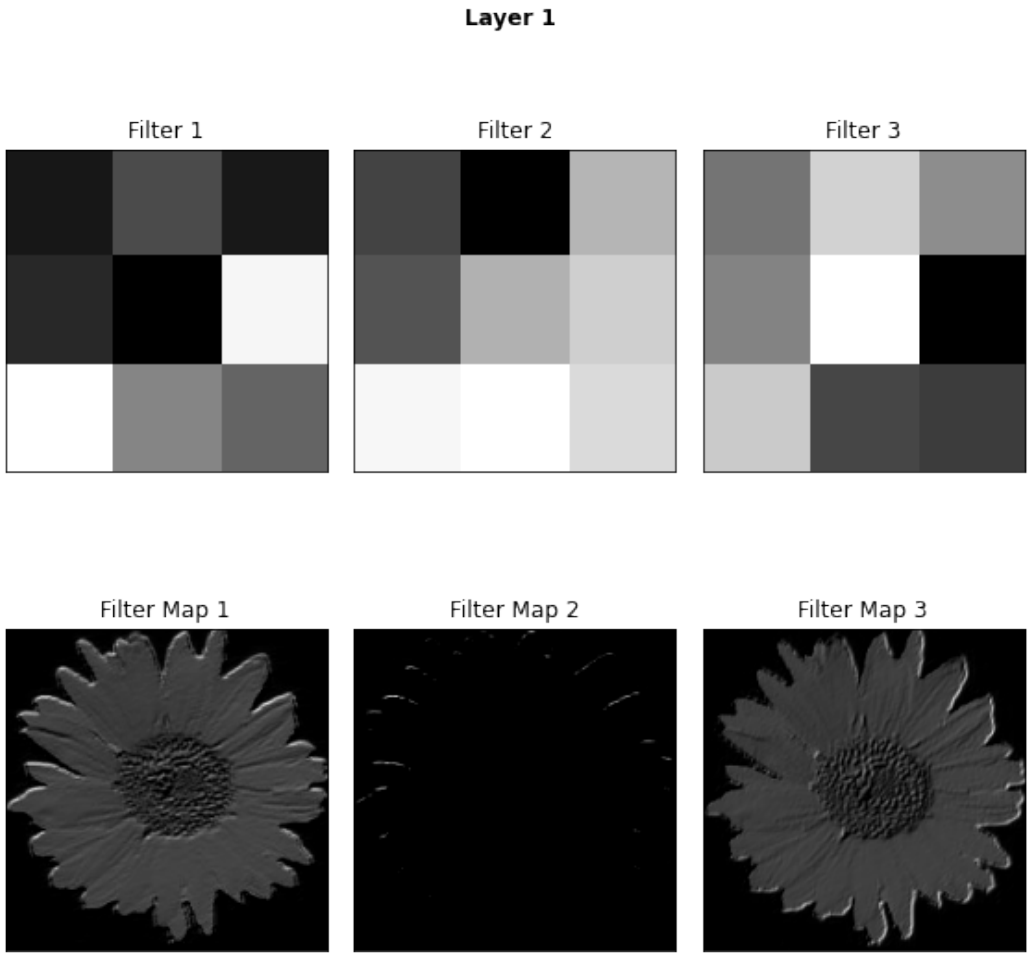

Fig. 2.4: Filter and feature map of sunflower in layer1

2.2.2 Second Convolution Layer

In the subsequent layers, more detailed and meaning patterns like shapes,internal features, faces are learned.

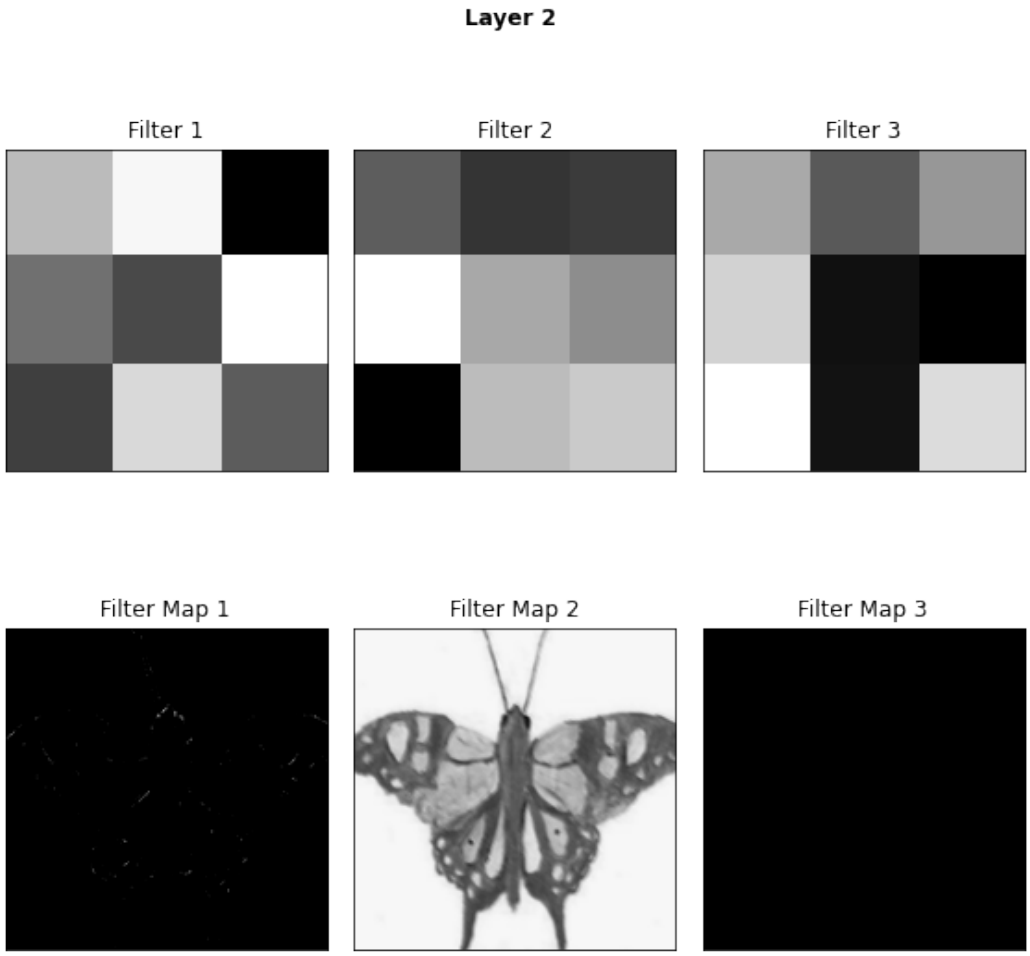

Fig. 2.5: Filter and feature map of butterfly in layer2

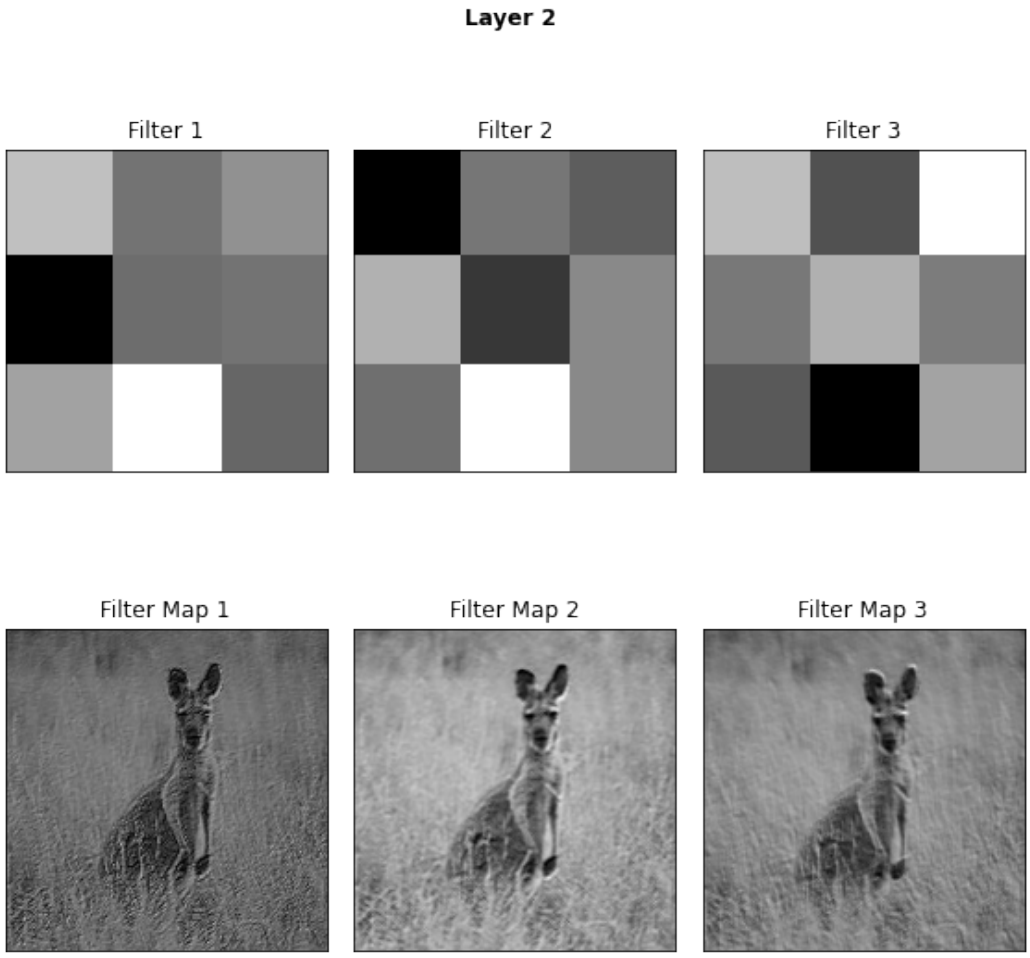

Fig. 2.6: Filter and feature map of kangaroo in layer2

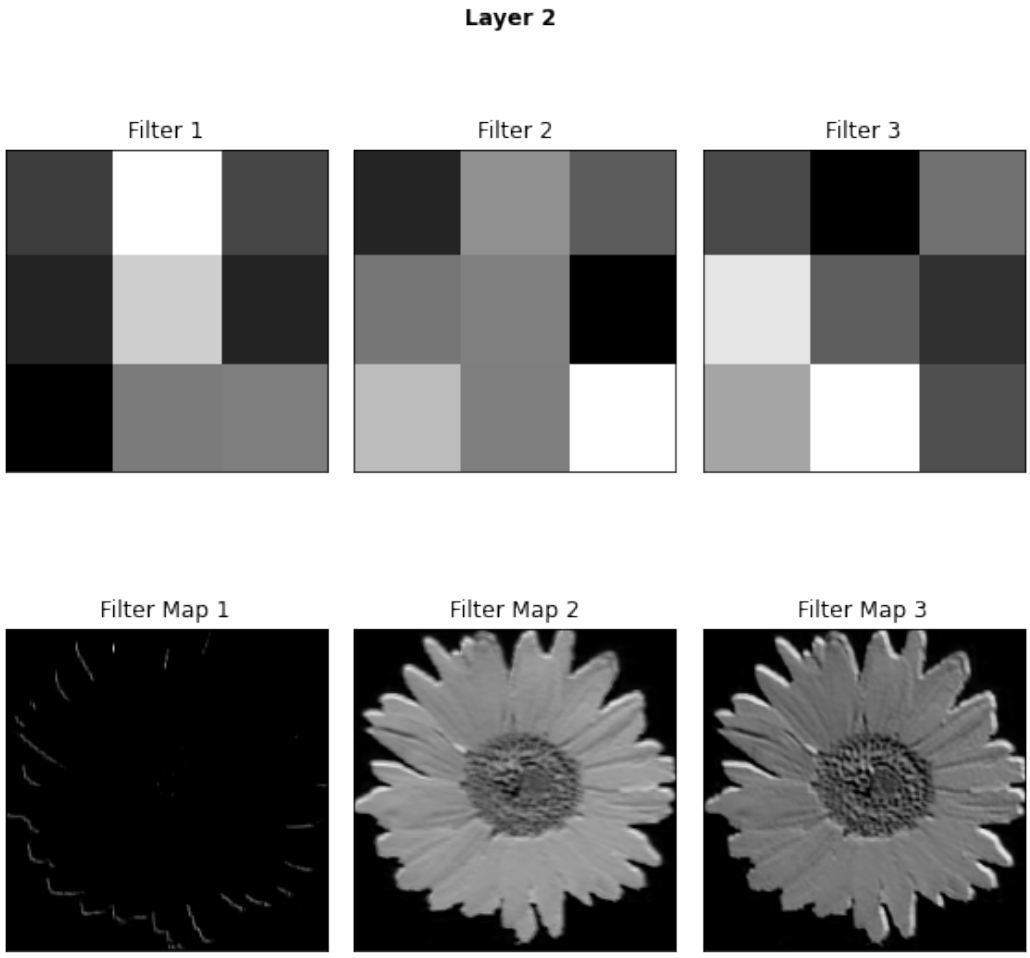

Fig. 2.7: Filter and feature map of sunflower in layer2

Chapter 3

Convolutional Neural Networks

3.1 Brief description of the dataset

We were given a subset of the Caltech-101 dataset consisting of three classes butterfly, kangaroo and sunflower. The Caltech-101 dataset consists of coloured images with varying sizes. The data consisted of three folders for training, test and validation.

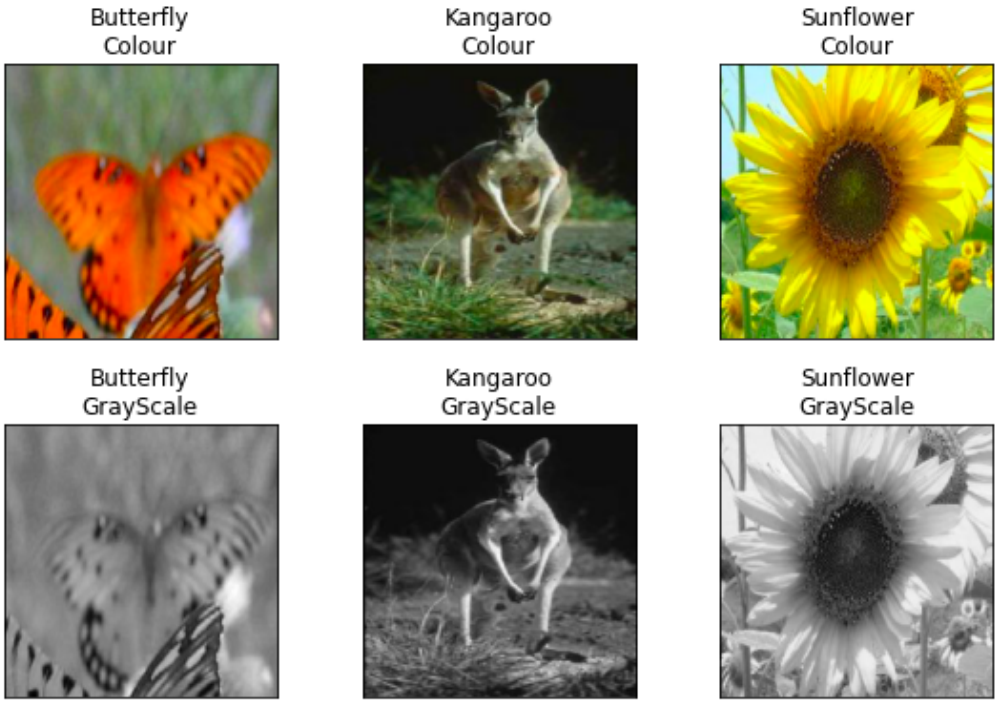

Fig. 3.1: Colour vs Grayscale

3.2 Observations

This section will explore the Convolution Neural Networks and the contents that the neurons are trying to learn while training.

3.2.1 Architecture

The CNN architecture has -

1. **Input layer** - It takes 224 x 224 x 3 channel images i.e. RGB images.
2. **Conv layers** - It has 2 Conv layers stacked after each other. Filter size is taken to be 32x32 and 64x64. Stride considered is 1 and 0 respectively.
3. **Maxpooling** - It down-samples the feature map. 2x2 max pooling operation with stride 1 is performed.
4. **Flatten** - It flattens the feature matrices so that FC layers can be applied. On flattening, spatial information is getting lost.
5. **FC layers** - It is applied with 128 nodes with rectified linear activation function. Then 3 node layer is applied for final classification.

The convolutional filters are initialized using Kaiming initialization and fully-connected layers with weights randomly sampled from a zero mean normal distribution with a standard deviation inversely proportional to the square root of the number of units. The weights are updated using a backpropagation algorithm with stochastic gradient descent optimizer.

3.2.2 Number of convolution layers

Different number of convolution layers has an impact on the classification but up to a particular threshold. Note that more the convolution layers you add, the more complex the model becomes. Adding more layers will extract more features, which after a certain point becomes unnecessary. After that, instead of extracting features, we will 'overfit' the model to the data. Overfitting can lead to serious problems. Consider automated driving, where the

model is supposed to identify pedestrian. If all the necessary features to classify a pedestrian is learned, then after having more layers, it will start detecting other unnecessary characteristics such as the trousers the pedestrian is wearing. If someone is wearing shorts, then the model will simply classify it as non-pedestrian and may run the car over it, which is dangerous Thus adding more layers beyond a certain threshold leads to finding unnecessary features in the data.

We have recent architectures on DenseNet and XceptionNet where the number of layers are very high. Practically speaking, it has been found that in many experiments, good features are being learnt using VggNet, ResNet or AlexNet. Having a network with more number of layers is actually not beneficial.

3.2.3 Different values of striding

Striding again plays an important role in downsampling its input. To observe the effect of striding, we fix a kernel and then cross-correlate it over the image. The image size is 224 x 224. For the report, we tried with two hand-crafted filters that are edge detection and Gaussian filters. We thus have two cases in this -

1. Low Striding - Low striding is when the stride is value is less than 5. We experimented with strides of 1 and 2.

2. High Striding - Low striding is when the stride is value is more than or equal to 5. We experimented with strides of 5 and 6. Benefit of having more striding is to reduce computations.

3.2.4 Different values of padding

When you convolve any filter over an image, we tend to lose pixels on the boundary of our image. This is also called the boundary problem. One solution is to reduce the filter size so that on using small kernels, for any given convolution, we will only lose a very few pixels, but at the end, this will get added up because we apply many successive stacked

convolutional layers along with max-pooling layers. Another solution is to add extra pixels around the boundary of the input image, thus increasing the effective size of the image. The size of padding is selected in a way that the filter is able to convolve completely over the image. However, one has the freedom to add more or padding depending on the task. To observe the effect of padding, we fix a kernel and then convolve it over the image. The image size is 150 x 150 (reduced the size from 224x224), filter size is 3 and stride is 4. For the report, we tried with two hand-crafted filters that are edge detection and Gaussian filters. We thus have two cases in this -

1. Low Padding - Low padding is when the pad value is less than the required number of padding to perform complete convolution. In our case, the required padding is . So we tested with .

2. Equal Padding - Low padding is when the pad value is equal to the required number of padding to perform complete convolution. In our case, the required padding is . So we tested with .

3. High Padding - Low striding is when the pad value is more than the required number of padding to perform complete convolution. So we experimented with pads of .

3.2.5 Different values of filters

Number of filters have the same impact as having a number of convolution layers. On increasing the number of filters, the model will tend to overfit and thus will try to learn the unnecessary features in the image. Note that the image contains a lot of information in the form of edges, luminosity, brightness etc. Our task is to learn only the discriminative features and not all the features. The more discriminative features it learns, the better our model is trained. Thus we have a trade-off between the number of convolutional layers and the number of filters in a convolutional layer. A good practitioner has to pick up these hyper parameters wisely depending on the problem he/she is trying to solve.

3.2.6 Different values of filter sizes

The filter can been seen as a matrix containing weights that the model must learn while training. The filter weights decides the feature of the image that it will detect and the strength of the activation decides the degree to which the feature is to be detected. Filter size is an important hyperparameter that one must select before training. High the filter size, more weights the model has learn, and vice-versa. To observe the effect of filter size, we fix a kernel and then cross-correlate it over the image. The image size is 224 x 224. For the report, we tried this with hand-crafted filter that is the Gaussian filter, for better understanding. Gaussian filter acts like an averaging filter. We, thus, will have two cases here -

1. Small filter size - Filter size less than or equal to 5 is considered small. So we considered size of 3 and 5. With small filter size, the image is somewhat sharp but the edges are washed out.

2. Large filter size - Filter size more than 5 is considered large. So we considered size of 21 and 35. On increasing the filter size, the image is getting more and more blurred.

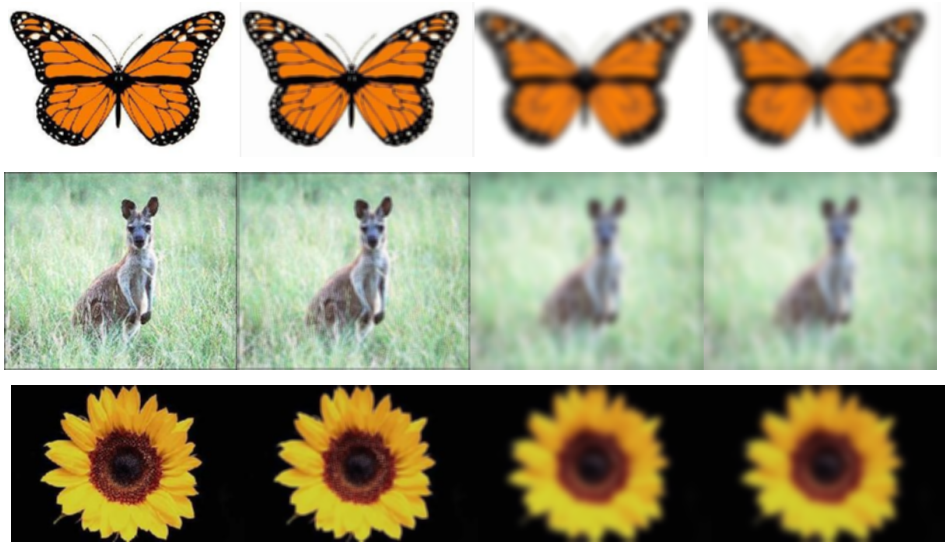

Fig. 3.2: Filter sizes in order - 3,5,21,35

3.2.7 Selection of pad, filter sizes and strides

In practice, we commonly use convolution kernels with odd height and width values, such as 1, 3, 5, or 7. This is a kind of a heuristic. Some say that it preserves spatial dimensionality while padding with the same number of rows on top and bottom, and the same number of columns on left and right. In our experiments, we didn't see much of a difference between keeping an odd or even number of pads, filter sizes and strides.

3.2.8 Training performance

The model was able to train well on the Caltech dataset. We can see the same from the confusion matrix and history plots.

Training Accuracy = 1.0.

Validation Accuracy = 0.8666666746139526.

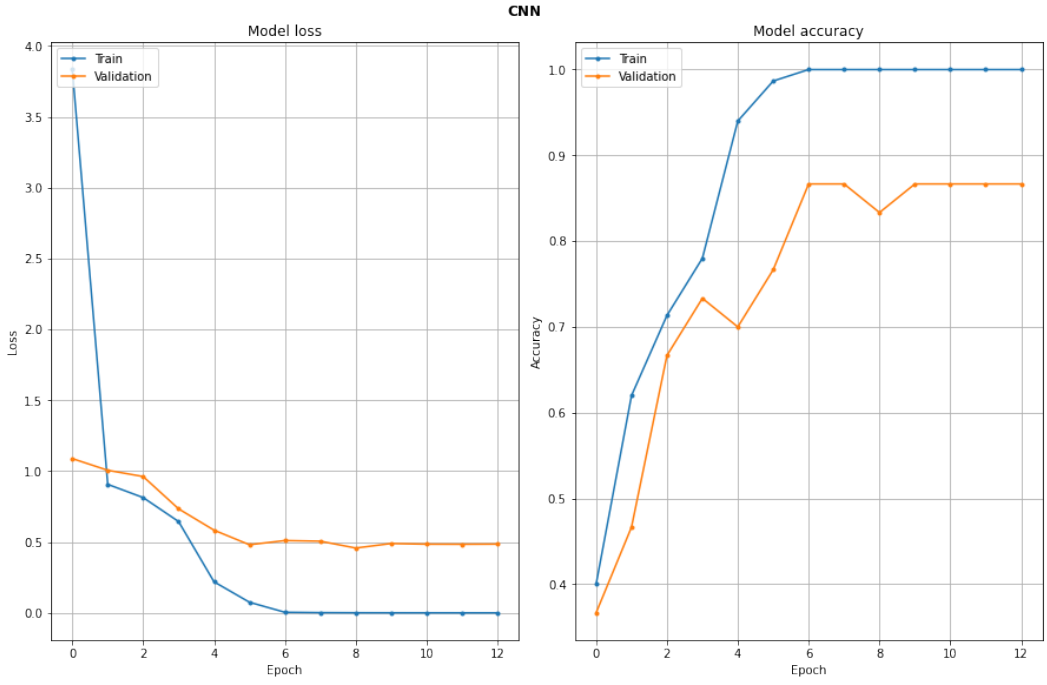

Fig. 3.3: Accuracy plots

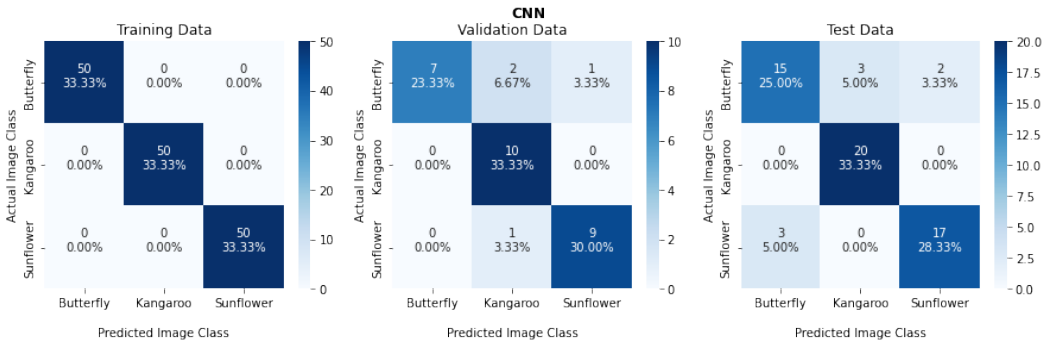

Fig. 3.4: Confusion matrix

3.2.9 Filters learned by CNN

The filters are drawn below that are learned by the CNN. In this question, the target of the CNN was learn the weights of the filters which can extract discriminative features so that it can perform proper classification. There is another concept related to spatial information which CNNs are good at learning. Spatial information means any information related to location. CNNs are good at learning it because of its 2D nature, but this information is lost when we flatten it before feeding it into fully connected layers.

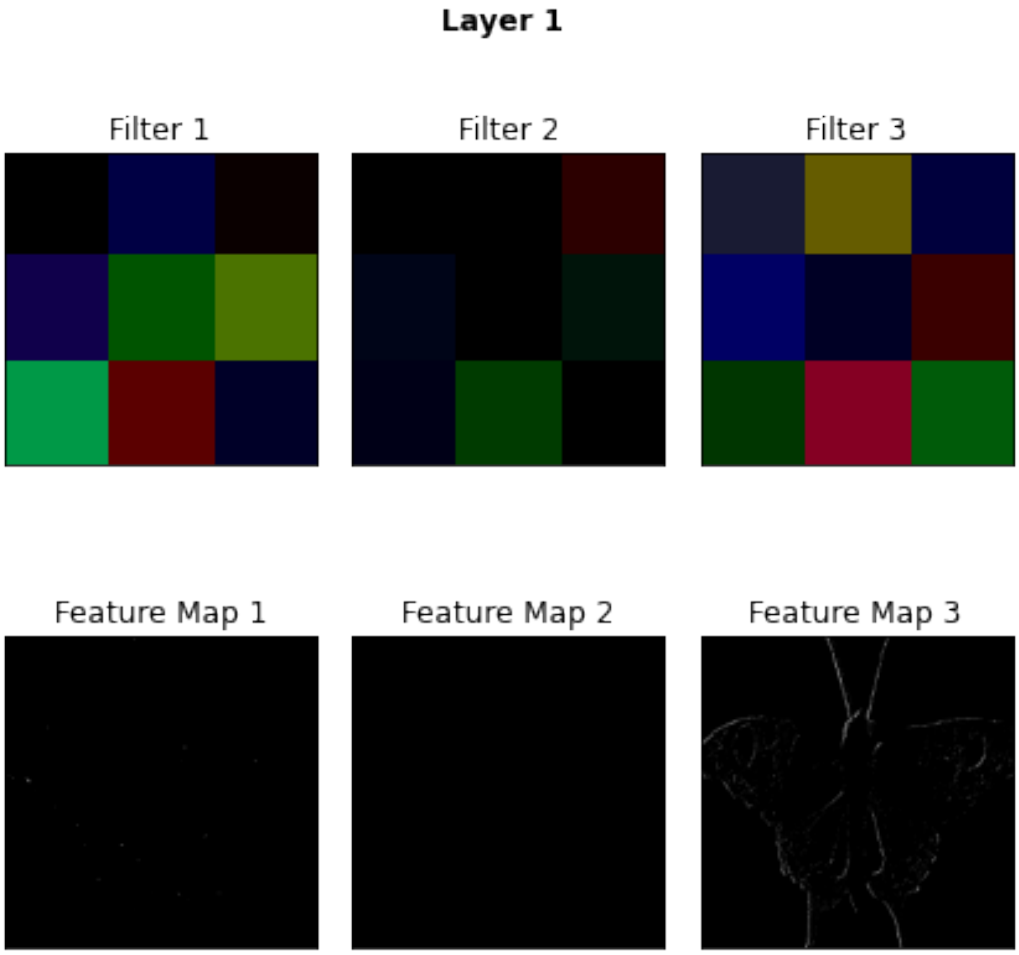

Fig. 3.5: filter and feature map of butterfly in layer1

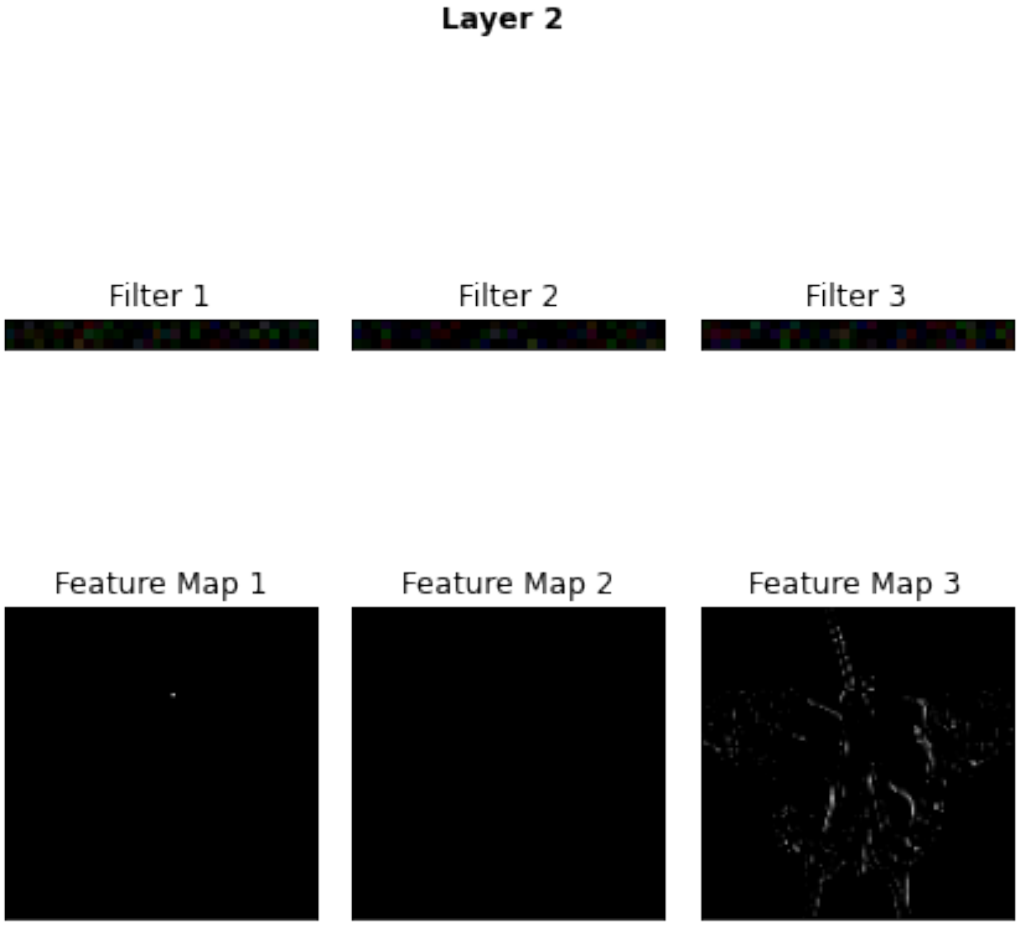

Fig. 3.6: filter and feature map of butterfly in layer2

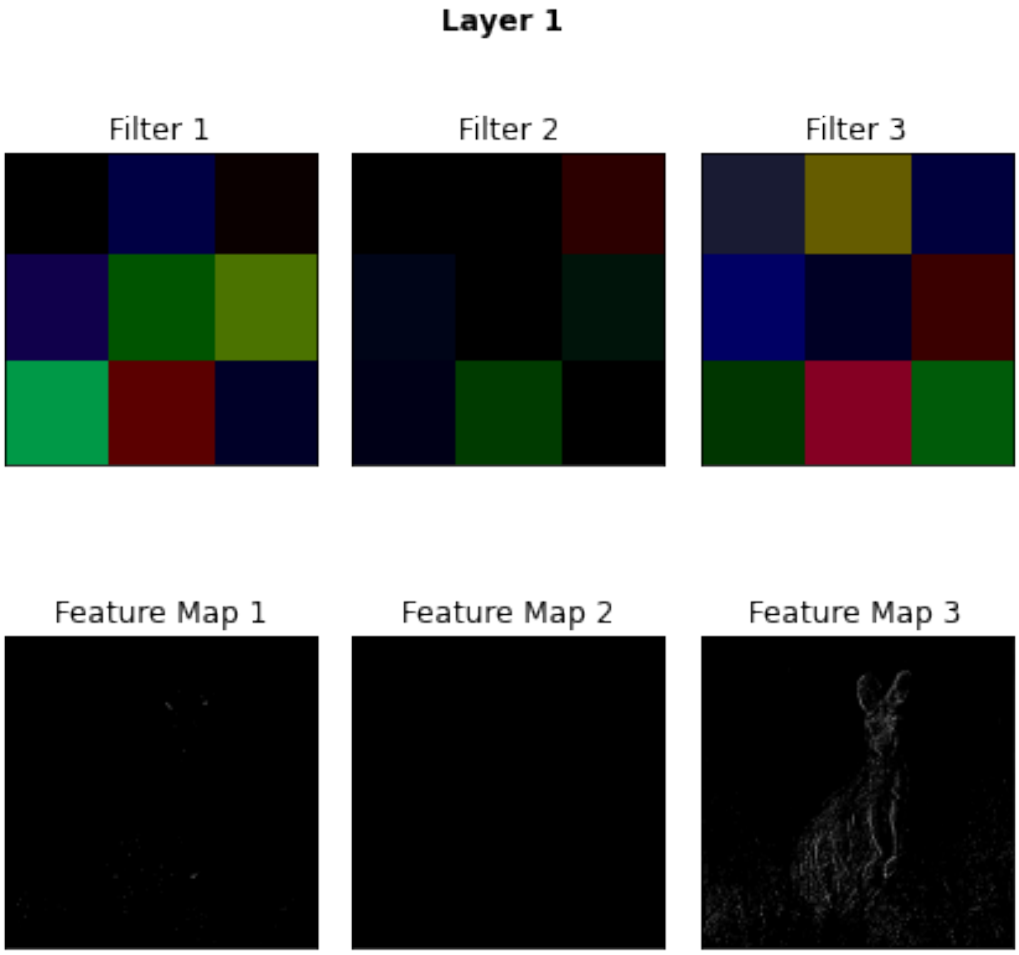

Fig. 3.7: filter and feature map of kangaroo in layer1

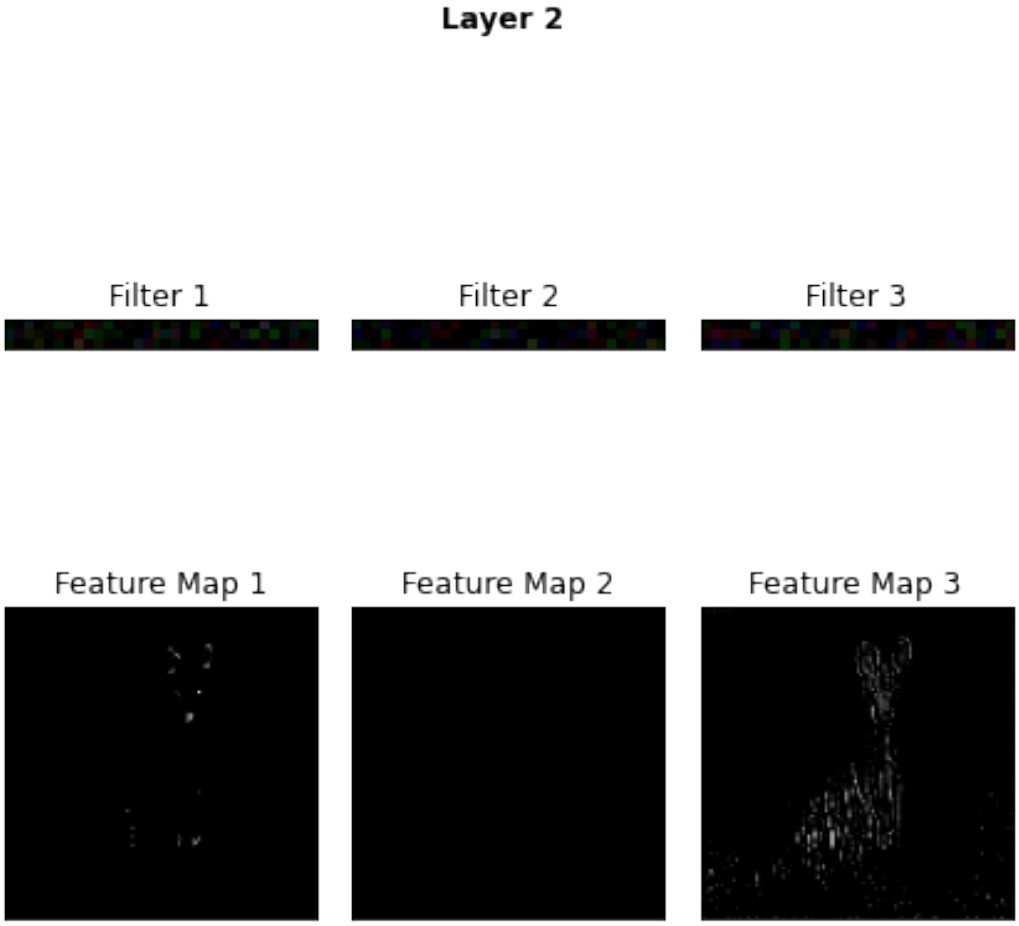

Fig. 3.8: filter and feature map of kangaroo in layer2

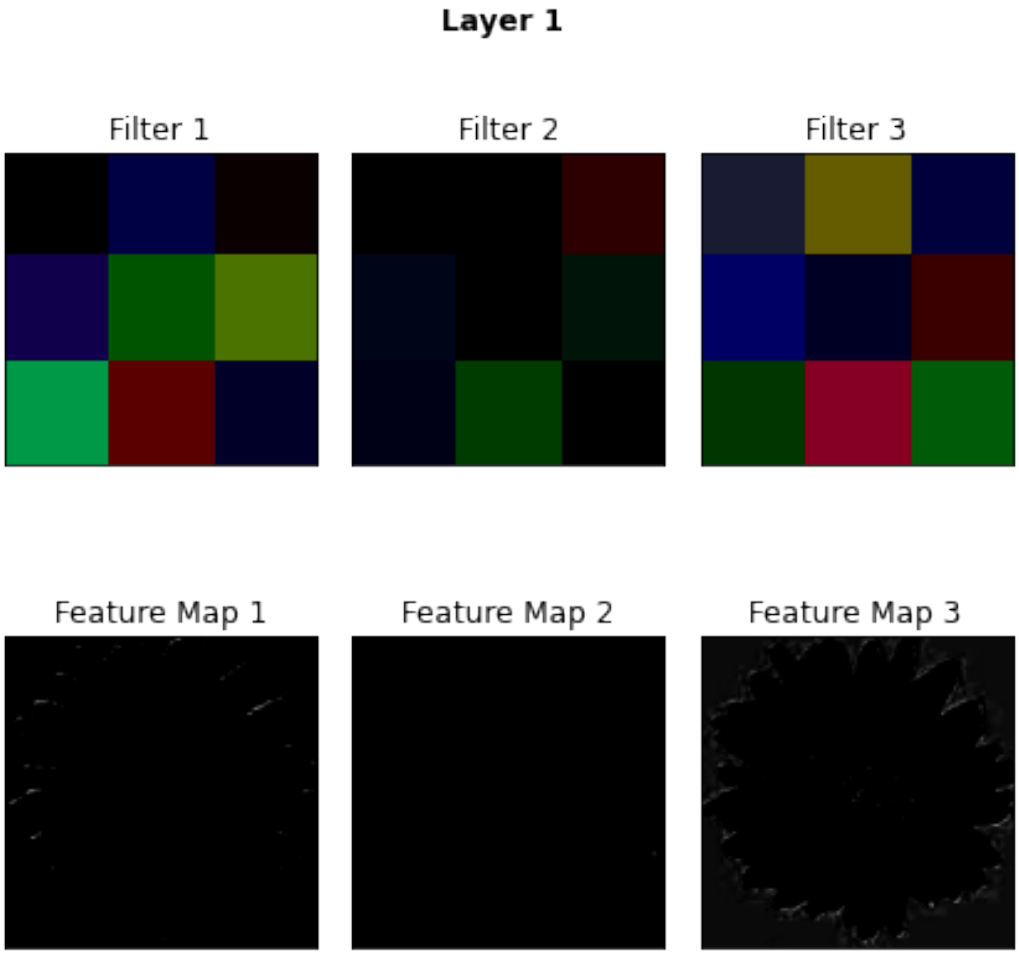

Fig. 3.9: filter and feature map of sunflower in layer1

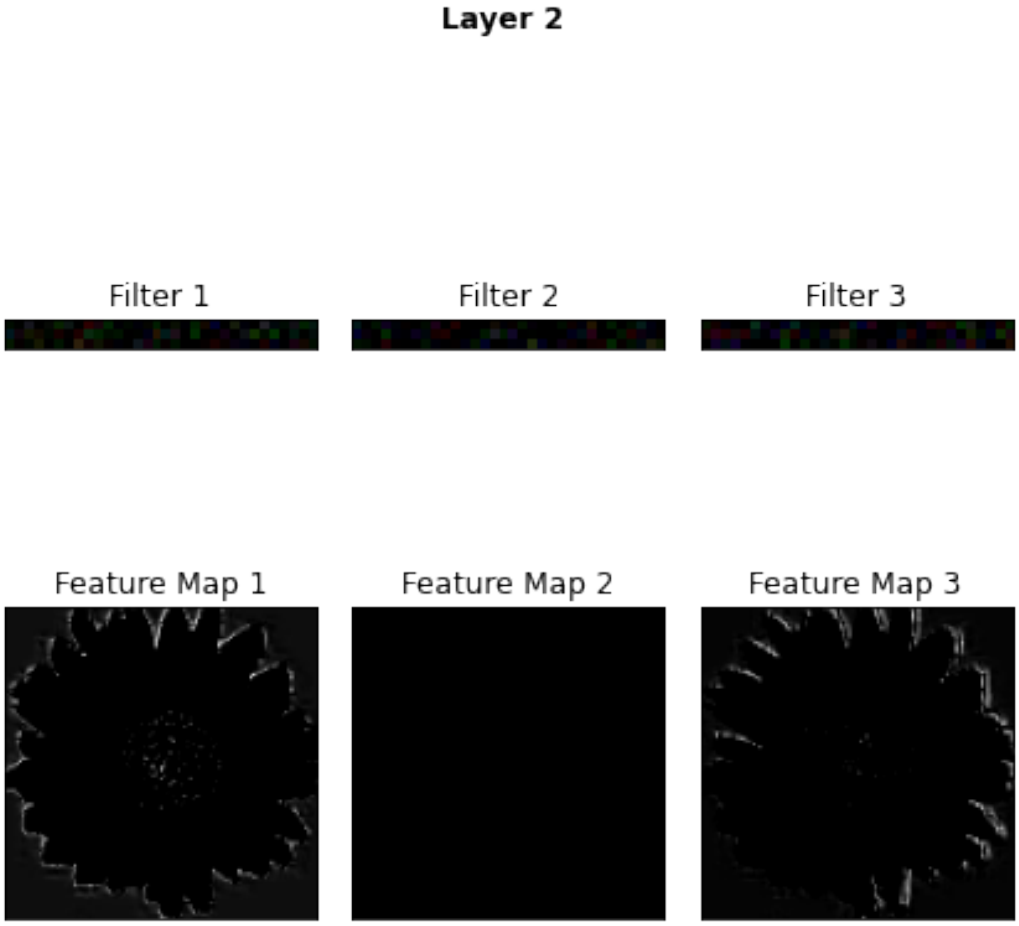

Fig. 3.10: filter and feature map of sunflower in layer2

3.2.10 Spatial invariance in CNNs

CNNs perform good in human-related tasks like object detection and face recognition. One of the reasons is because of the **spatial invariance property** of the convolution operation. CNNs are not sensitive to the position of any object in the picture. An invariant neuron is the one that maintains a high response to its feature despite certain transformations of its input. For example, a pedestrian selective neuron might respond strongly whenever a pedestrian-like patch is present in the image. If the network is not invariant, it will not respond if the image is tilted by a small angle, which again is not desirable.

3.2.11 Translational invariance and equivariance in CNNs

We will start with translational equivariance. Translational Equivariance is a very important property of the convolutional neural networks where the model is able to robustly detect the position of the object in the image without it to be present in a fixed order. It means that if there is any change in the input, we would expect a change in the output as well. In mathematics terms, translational equivariance is -

$$f(g(x)) = g(f(x))$$

The order in which the functions are applied does not change the final output. CNNs are able to achieve this because of the weights that are being shared. As a result, if there is any object in an image, then the neurons will be able to detect it irrespective of its position in the image. Thus CNN can detect if there is a pedestrian in an image, no matter where the pedestrian is present in that image.

Another concept related to this is translational invariance. Invariance to translation means that on translating the inputs, the CNN will manage to detect the class of the image. This is actually ensured by the max pooling operation. In max pooling operation, a filter is strided over the output of the convolutional layer and at a particular location, we tend to extract the max. Hence even if the inputs are changed slightly, it won't affect the values of most of the pooled outputs. This is again a very important property of CNN which helps in tasks like face recognition where we are not interested in the position of the eye

but in the eye itself. One can experiment these characteristics using data augmentation. On doing so, you will find that CNN will perform good in all possible orientations.

3.2.12 CNN Visualization : Patch detection

For each of the same 3 images, we found out 5 neurons in the last convolutional layer that are maximally activated. Maximally activated means the pixels in the feature maps is not zero. We then traced back to the patch in the image which causes these neurons to fire. You will observe that the patch is more near to the face of the animals or the portion of the image which contains the most discriminative features. This is happening because of properties of the translational invariance and equivariance. This is a very interesting feature of CNNs as it is able to find that patch where it can gain the most information.

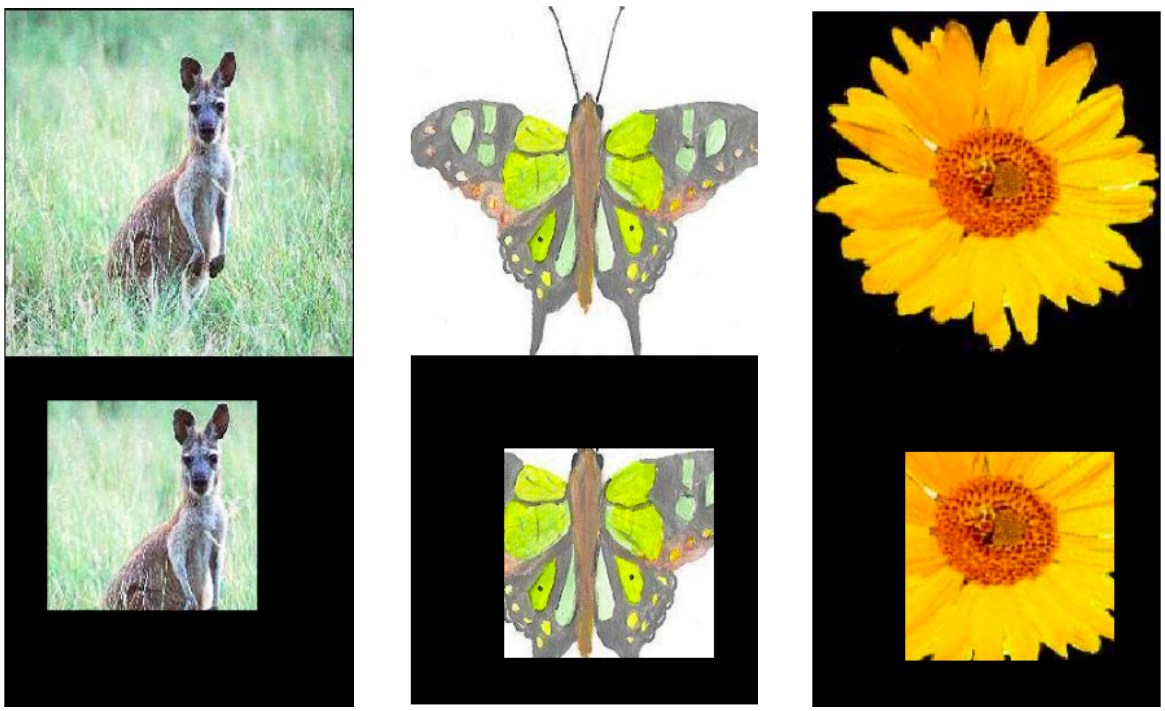

Fig. 3.11: Patch detection by CNNs

3.2.13 Why CNNs rule the computer vision field?

CNNs have three advantages over any standard neural networks -

1. Sparse connections - It makes it computational efficient to work with high dimensional data.

2. Weight sharing - It shares the same weights across the entire image, which causes reduced memory requirements and translational equivariance. It again helps in faster computations and better prediction.

3. Subsampling or pooling - most prominent pixels in terms of magnitude are passed on to the next layer and the rest are dropped. This helps in translational invariance.

Chapter 4

VGG 19 Visualization

4.1 Brief description of the dataset

We were given a subset of the Caltech-101 dataset consisting of three classes butterfly, kangaroo and sunflower. The Caltech-101 dataset consists of coloured images with varying sizes. The data consisted of three folders for training, test and validation.

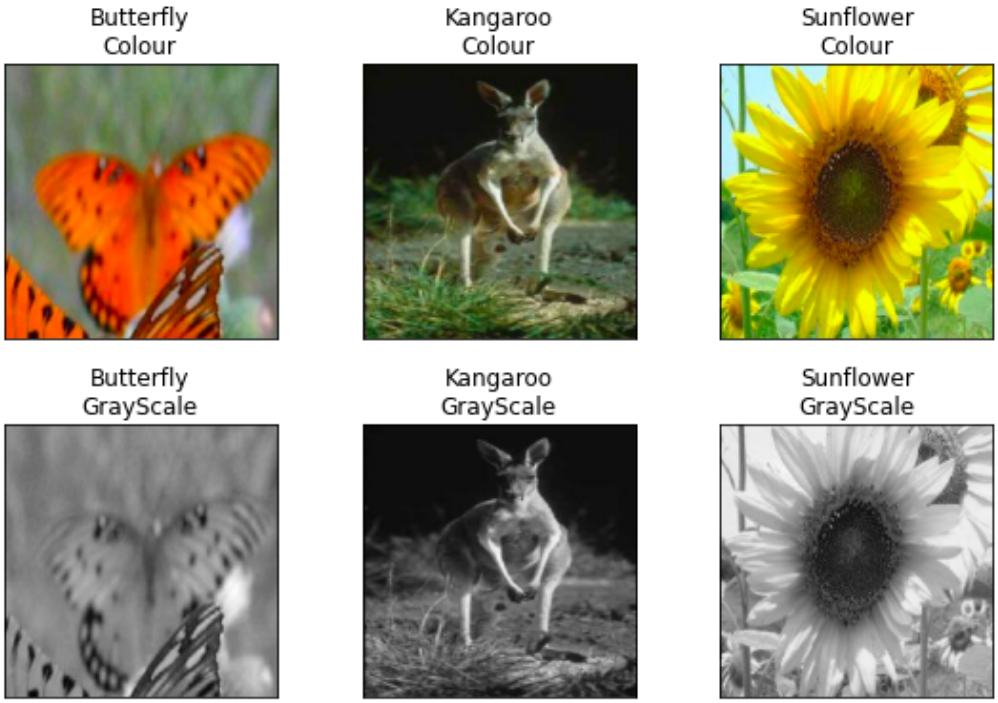

Fig. 4.1: Colour vs Grayscale

4.2 Observations

This section will explore the visualization of CNN and the learned filters. First, we have implemented transfer learning using VGG19 loaded with imagenet weights. Then we chopped off its prediction layer and added a new one which we trained on the Caltech dataset, while keeping the layers of VGG19 as freezed. Then to visualize its learning, we have picked up 5 neurons in the last convolutional layer that are maximally activated and found the patches. And lastly, we applied guided backpropagation to oberve the gradients image.

4.2.1 Model architecture

The architecture is -

1. **Input layer** - It takes 224 x 224 x 3 channel images i.e. RGB images.
2. **VGG19** - This is base model. Trained it on the imagenet datasets and loaded the best weights on the model. The weights are freezed in subsequent training.
3. **Dense layer** - This is the classification layer. Used for final prediction. This is the only trainable layer.

4.2.2 Performance

One can see from the confusion matrix and the loss vs epochs plots that the model has trained well without any overfitting.

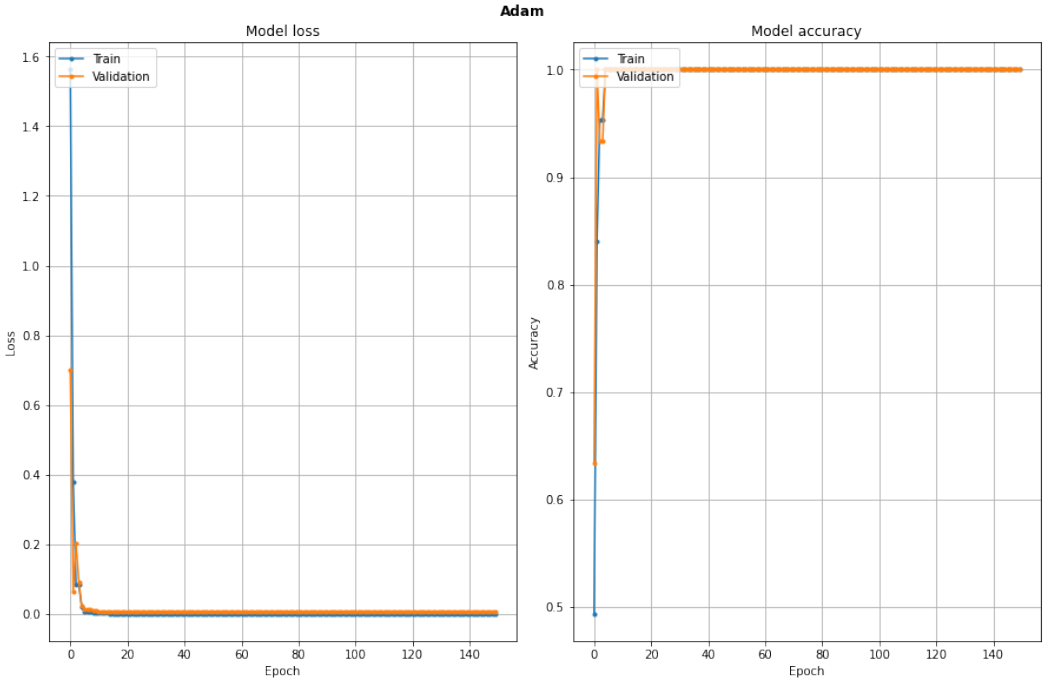

Fig. 4.2: History plots

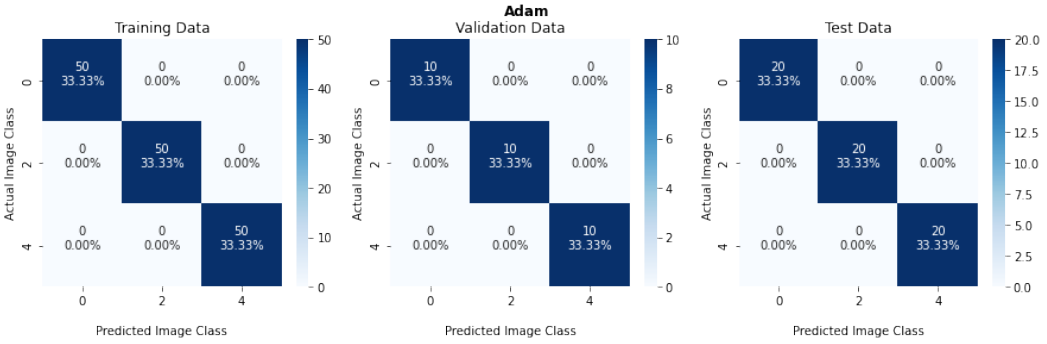

Fig. 4.3: Confusion Matrix

4.2.3 Patch detection

For each of the same 3 images, we found out 5 neurons in the last convolutional layer that are maximally activated. Maximally activated means the pixels in the feature maps is not zero. We then traced back to the patch in the image which causes these neurons to fire. You will observe that the patch is more near to the face of the animals or the portion of the image which contains the most discriminative features. The patches are plotted below.

Fig. 4.4: Patch detection by transfer learning

4.2.4 Guided Backpropagation

We also implemented Guided Backpropagation. In Guided backpropagation, the feature activation of intermediate layers is mapped back to the input image. While going back, we perform three operations for the following layers-

1. **Fully connected** layers - Simple $W^T X$ is performed where X is the feature map and W is the learned weights after a number of epochs.

2. **Maxpooling** layer - We perform upsampling instead of downsampling here.

3. **Convolution** layer - We perform deconvolution here. Deconvolution here is actually convolution with a different non-linear activation unit. It is guided ReLU, where we compress not only the negative values during forward propagation but the negative values that arise in backpropagation.

The neurons present in the feature map of the last convolutional layer can be classified into two parts -

1. Activated neurons - We picked up 5 activated neurons and performed the guided back-prop using the above operation. The plots are shown below. More activated the value is, the sharper images it forms after backprop.

2. Deactivated neurons - We picked up deactivated neurons and performed guided back-prop using the above operation. The plots we obtain are gray images without any edges. This is because any operation on a zero matrix will yield a zero matrix.

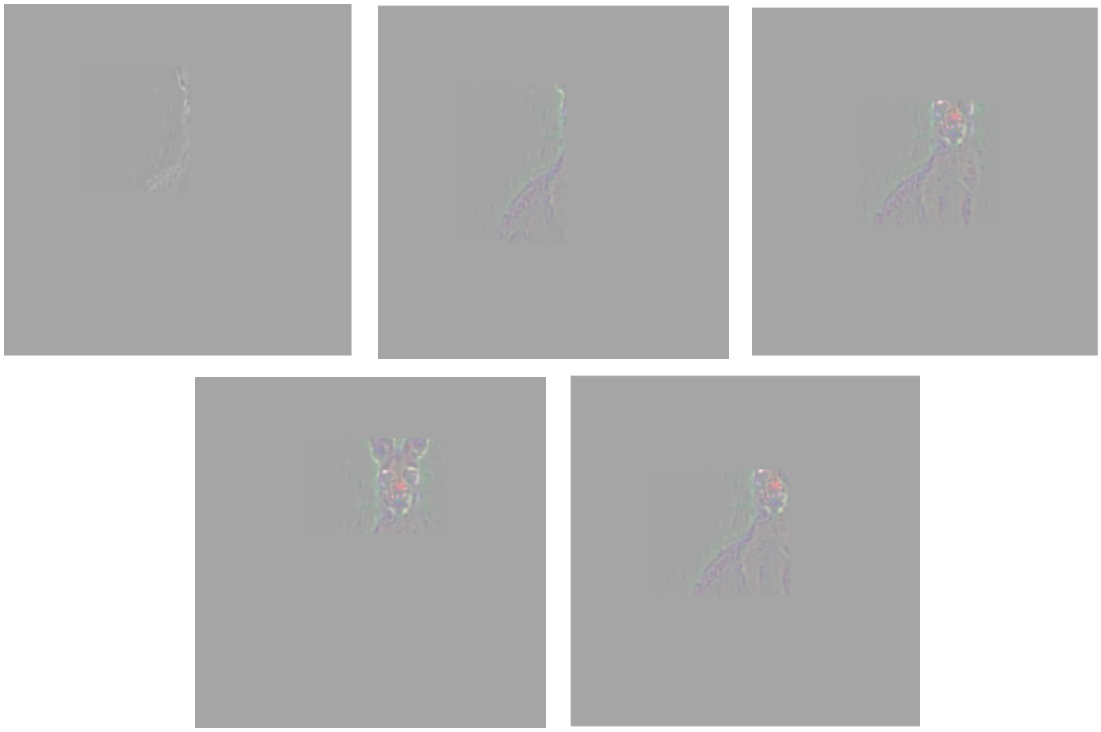

Fig. 4.5: Gradient images for Kangaroo

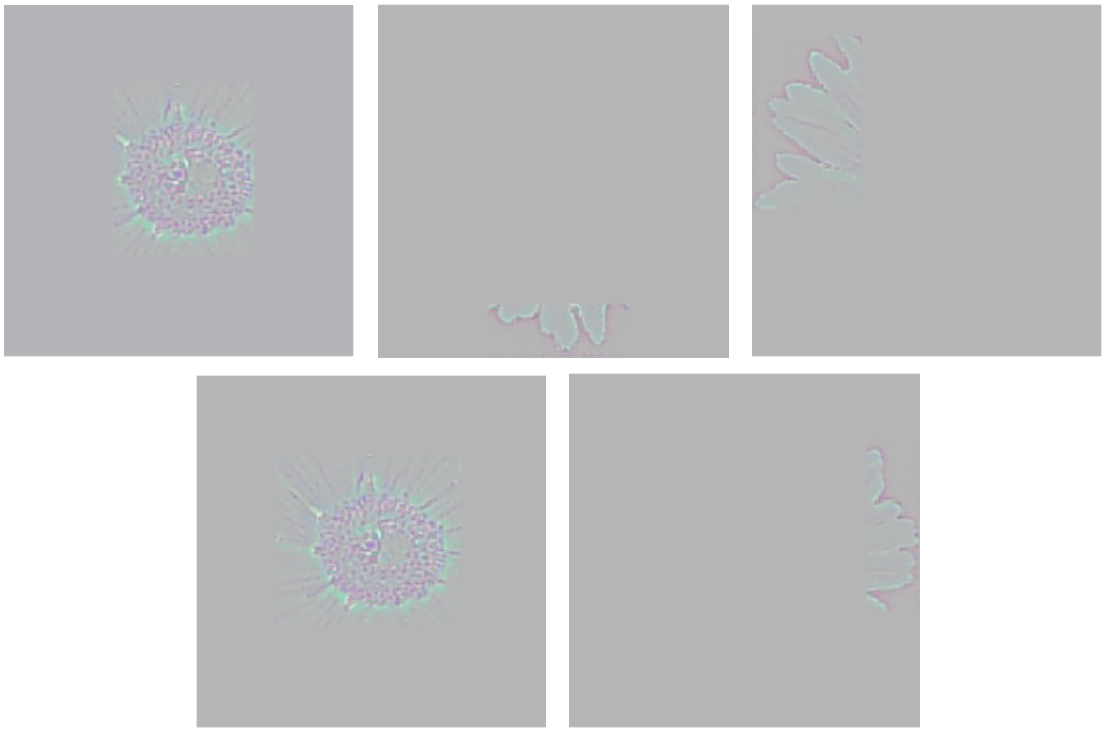

Fig. 4.6: Gradient images for Sunflower

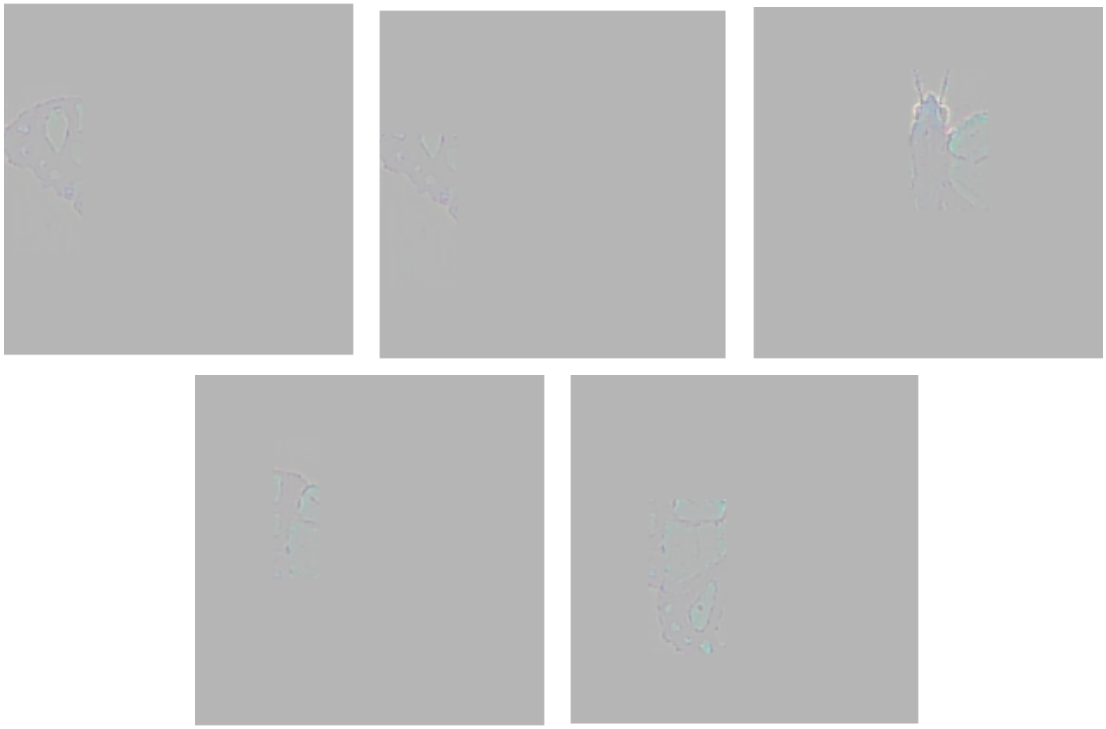

Fig. 4.7: Gradient images for Butterfly

References