Visualising Deep Convolutional Neural Networks

Tushar Gautam, Nitish Shingde, Soumyajit Saha

Due: 20 April 2022

Overview

Since their inception in early 1990s, Convolutional Neural Networks have excelled at tasks like hand-written digit recognition, face detection, etc. The idea has been around for around three decades now, but the factors like availability of larger datasets, powerful GPUs and better regularisation techniques have lead to a sudden rise in application of Convolutional Neural Networks. Despite such an enormous interest, to this day Neural Networks are considered to be much of a black box. From an engineering or mathematical point of view, getting an insight into the internal operation and behavior of these complex models, or how they achieve such good performance is still difficult. This is a big reason that still there's no justification as to why one architecture or activation functions out performs the other, and the practitioners or researchers have to resort to trial and error to get the best results possible.

Project Goals

In this project, we will study a visualization technique that reveals the input stimuli that excite individual feature maps at any layer in the model. It also allows us to observe the evolution of features during training and to diagnose potential problems with the model. The following are the goals of this project:

1. Visually inspect how well different parts (filters in hidden layers) are trained.
2. To find out which parts of the input affects prediction the most.
3. To analyse if a neural network can be fooled by *smartly* distorting a part of image.
4. Visualising how different filters evolve over training epochs.
5. To find out how useful transfer learning can be used effectively to train a neural network quickly.

Background and Related work

Visualising features is a great way to gain intuition about the Neural Network, but most of the work is restricted to the very first layer where projections to pixel space are easy to get. However, Erhan et al., 2009 performed the analysis for hidden layers by finding the optimal stimulus for each unit using gradient descent in image space. In this method, initialisation was done specifically and no information was given about unit's invariance.

Motivated by the short coming of the above mentioned work, (Le et al., 2010) (extending an idea by (Berkes & Wiskott, 2006)) showed how the Hessian of a given unit may be computed numerically around the optimal response, giving some insight into invariances. Here also there was a problem in deeper layers where the invariances are extremely complex. Hence the method used a simple quadratic and captured poor approximations.

Resources used for the project

The following provides the list of various resources used, however in the References section you can see details with proper citation.

1. Google colab was the biggest resource which provided GPUs for training.
2. All the code was written using Tensorflow library.
3. tf-keras-vis library to help visualize the CNN models.
4. Apart from the research papers (mentioned in the References section), we also used online articles with links as follows: Transfer Learning Basics, TF Fine Tuning article, Activation Maximization, Filter Visualisation.
5. You can find the code related to this project in this Github repository.

Project Description

Deep Neural Networks have shown extraordinary performance in field of visual recognition tasks, but there are far too easy ways to fool these models using imperceivable but carefully constructed nudge in the input. These are some adversarial examples that hamper the performance of the neural network in recognition of images."Adversarial Patch", a paper published at NIPS 2017 demonstrated how to generate a patch that can be placed any area within the field of view of the classifier and cause the classifier to output a targeted class. So, these attacks prove that the neural networks are still fragile and prone to adversarial attacks. In order to tackle this issue, we are trying to develop a visualisation strategy that would enable us to visualise how the focus of the feature maps are prioritised in some important layers of the neural network and how the layers are trained. This visualisation would give us an insight into the regions of a particular image that is being given more priority than others while training, even if those regions are not so relevant and could point out the flaw in the neural network layer that is making it more prone to adversarial patches. In most cases, the adversarial patches disrupts the classification process as in some layers irrelevant regions

are given more priority than important regions that actually contributes to classification process. Thus, we have decided to move forward with this visualisation strategy in Deep Convolutional Neural Networks in order to reveal what a particular feature map focuses on.

To visualise the feature activity in the hidden layers, a process of mapping these activities back to the input pixel space is used. This results in an input pattern that would excite the activation in the selected feature map. This mapping is done with a Deconvolutional Network, that is attached to each of the layers of a Convolutional Network providing a continuous path back to image pixels.

The methods of Visualizing a CNN model can be broadly categorized into two parts based on their internal workings:

1. **Activation based methods:** In these methods, we decipher the activations of the individual neurons or a group of neurons to get an intuition of what they are doing.

   (a) Maximal Activations: In a CNN, there several learned template matching filters that maximize their output when a similar template pattern is found in the input image. The idea is to generate an input image that maximizes the filter output activations, i.e. we compute and use that estimate to update the input. This helps us understand what sort of input patterns activate a particular filter. For example, there could be a nose filter that gets activated in the presence of nose within the input image.

   (b) Image occlusion: In image classification problem, we want to make sure the model is identifying the location of the object in the image, and not using the surrounding context instead. By occluding parts of input images, we will monitor how the output of classifier is affected. For example, we want to make sure the model identifies birds because of a bird in the image, and not because of tree leaves.

2. **Gradient based methods:** These methods tend to manipulate the gradients that are formed from a forward and backward pass while training a model.

   (a) Saliency Maps: We compute the gradient of output category with respect to input image. This should tell us how output category value changes with respect to a small change in input image pixels.

   (b) Class activation maps, or grad-CAM: Instead of using gradients with respect to output, grad-CAM uses penultimate (pre Dense layer) Conv layer output. The intuition is to use the nearest Conv layer to utilize spatial information that gets completely lost in Dense layers.

We would be using the pre-trained VGG16 model trained on ImageNet dataset for our project. VGG16 is 16 layer convolution neural network model used in the The ImageNet Large Scale Visual Recognition Challenge (ILSVRC), an annual computer vision competition. We'll perform all our tests on google colab.

After implementing the above methods we have the following answers:

1. The last Dense layer focuses on salient features of every Categories present in VGG16 model which are comes after fine tuning of filters from previous two Dense layers.

2. Evolution of complexity in the convolutional layers as we go deeper into the model. This also answers the question as to when do we need to add or stop adding more convolutional layers while weighing between improving the accuracy to keeping the model less computationally intensive.

3. The trained neural network was making predictions looking at the correct features which we can see from the heat maps.

4. Results of reverse engineering a random input image in the form of animation based on maximal activation of particular neurons from a layer.

5. Help distinguish whether the model is trying to classify/predict based on the correct features or not.

6. Transfer learning can speed up the training process massively and also result in superior accuracy.

Implementation details

Visualisation of Dense layers

The VGG16 CNN model has 3 Dense layers namely 'fc1' with 4096 neurons and 102764544 parameters, 'fc2' with 4096 neurons and 16781312 parameters, and 'predictions' with 1000 classes and 4097000 parameters. These Fully-Connected layers follow a stack of convolutional layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. We tried to visualise the features that were given the highest priority during training in these layers that contributed in the classification process in the last layer. For visualisation of the features we have selected 3 Categories from the VGG16 model, namely, 'Great White Shark', 'Bald Eagle' and 'Assault Rifle'. As in Figure 1, we have given a visual presentation of the 'fc1' layer. If we observe carefully, we cannot distinguish any relatable features of each of the Categories in these layer. In the next layer 'fc2' as given in Figure 2, we can see feather like features for the different orientations of 'Bald Eagle' category, but still no relatable features for rest 2 categories. But in the 'predictions' layer given in Figure 3, we can visualise the fins and tail like structures of different orientations of the 'Great White Shark' category, feathers and beak like structures of different orientations of the 'Bald Eagle' category; and magazine, grip and trigger like structures of different orientations of the 'Assault Rifle' category. Thus, at the last layer the crucial features are identified and these serve the main criterion for classification process. In the last layer the model predicts which class among 1000 categories does the input image maps to. We have used Activation Maximization and Categorical Score methods from tf_keras_vis.utils package in order to view the features identified during training of each layers. The Activation Maximization helps us to visualize what a class in your trained model looks like by inverting the process of training in a Neural Network model. It keep the weights and the desired output constant and modifies the input such that it maximizes certain neurons.

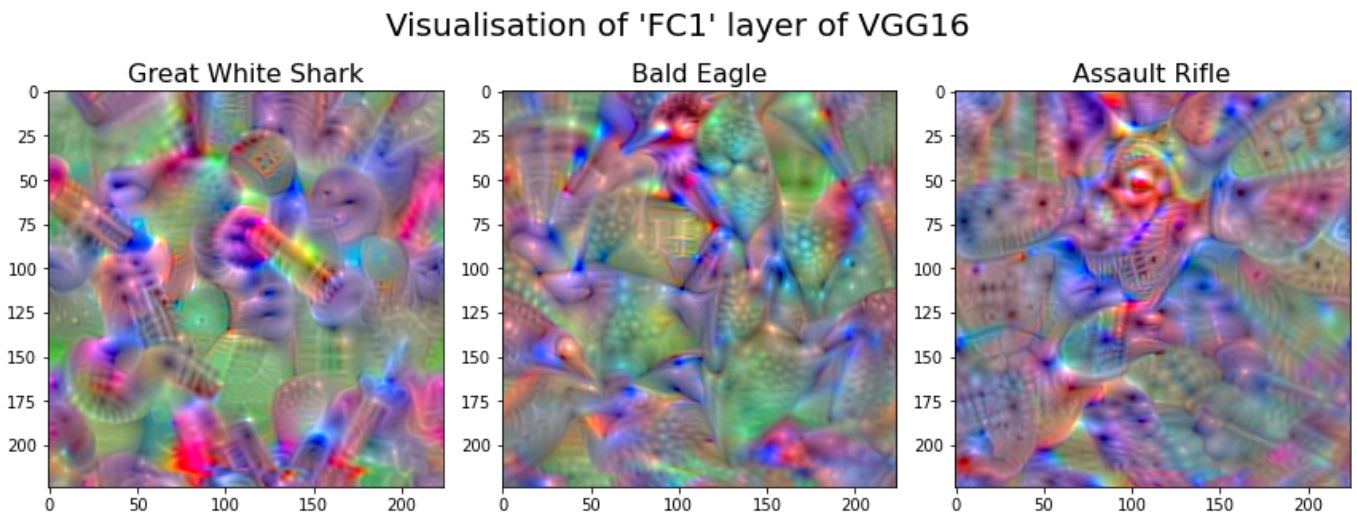


Figure 1: Visualisation of the features identified in the 'fc1' Dense layer in VGG16

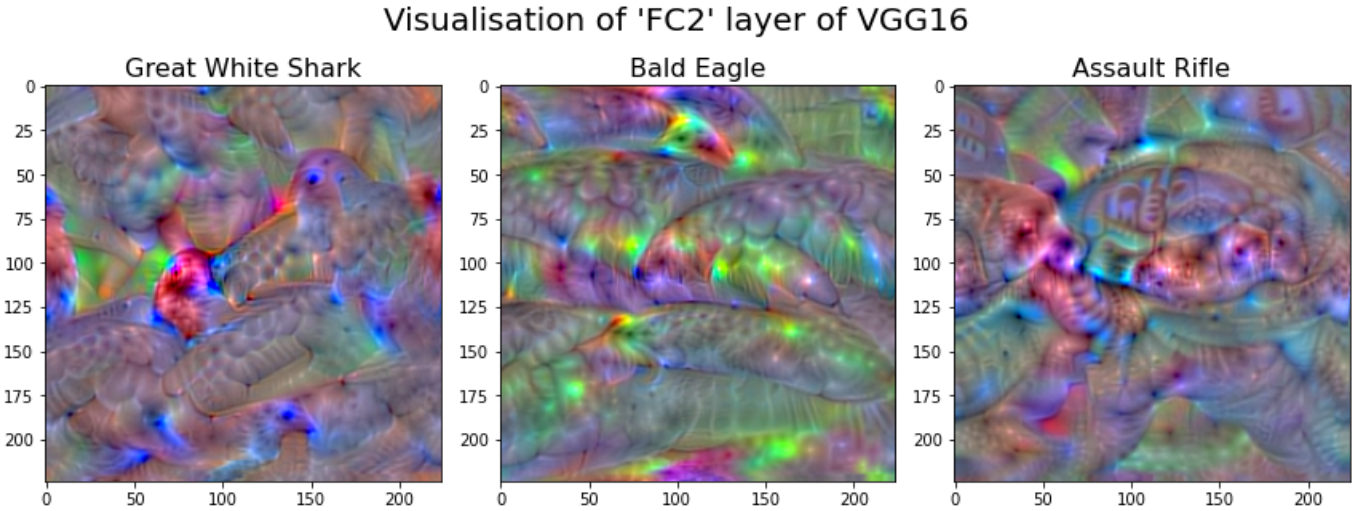


Figure 2: Visualisation of the features identified in the 'fc2' Dense layer in VGG16

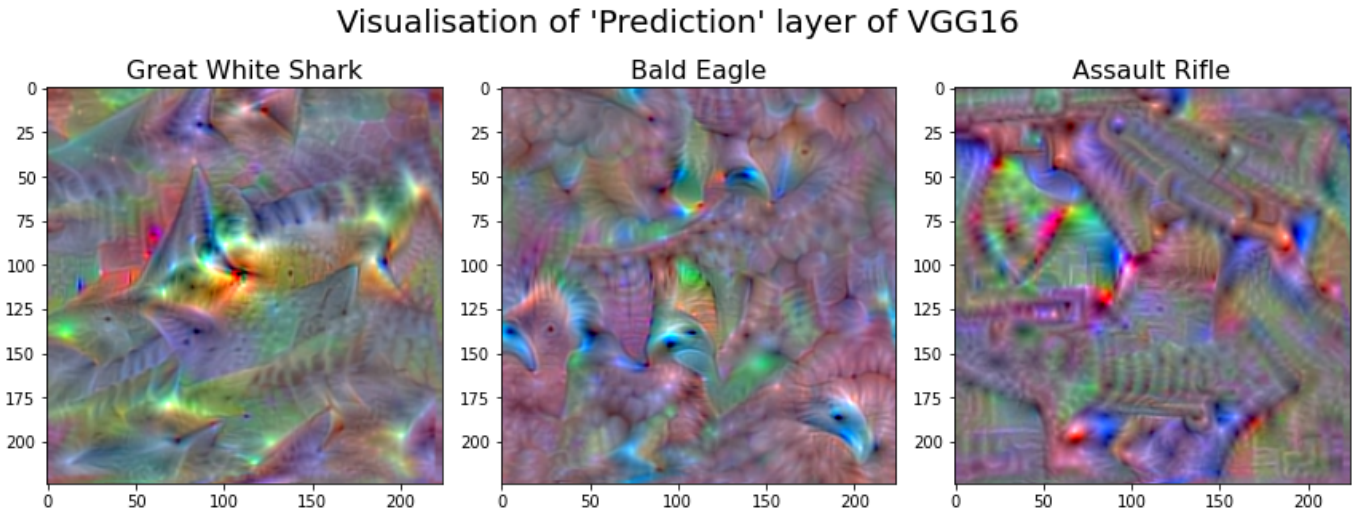
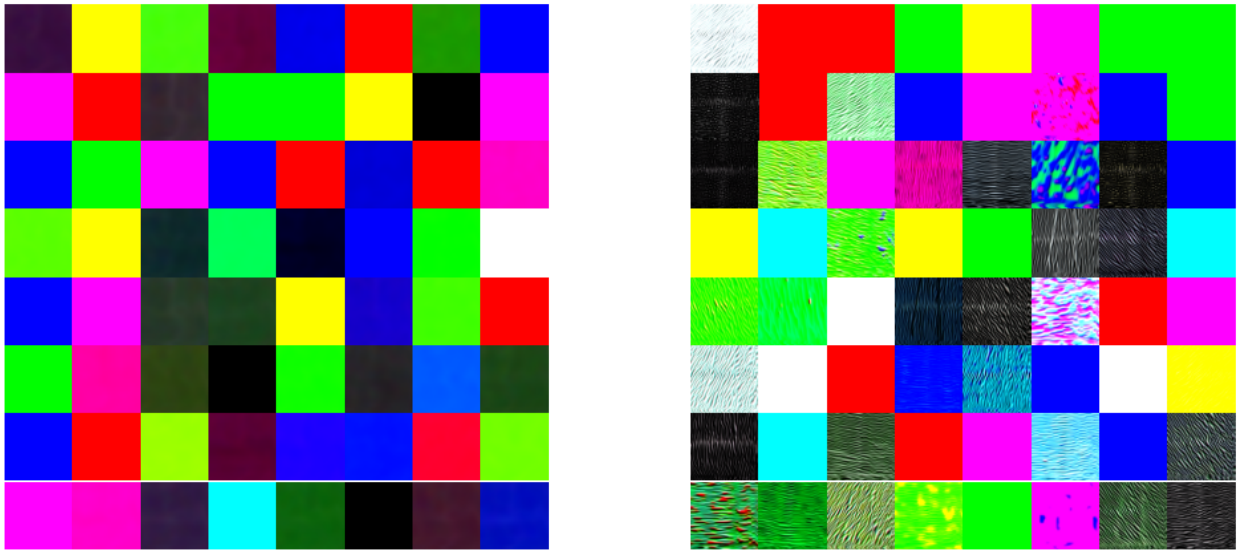


Figure 3: Visualisation of the features identified in the 'predictions' Dense layer in VGG16

Visualization of Conv2D layers



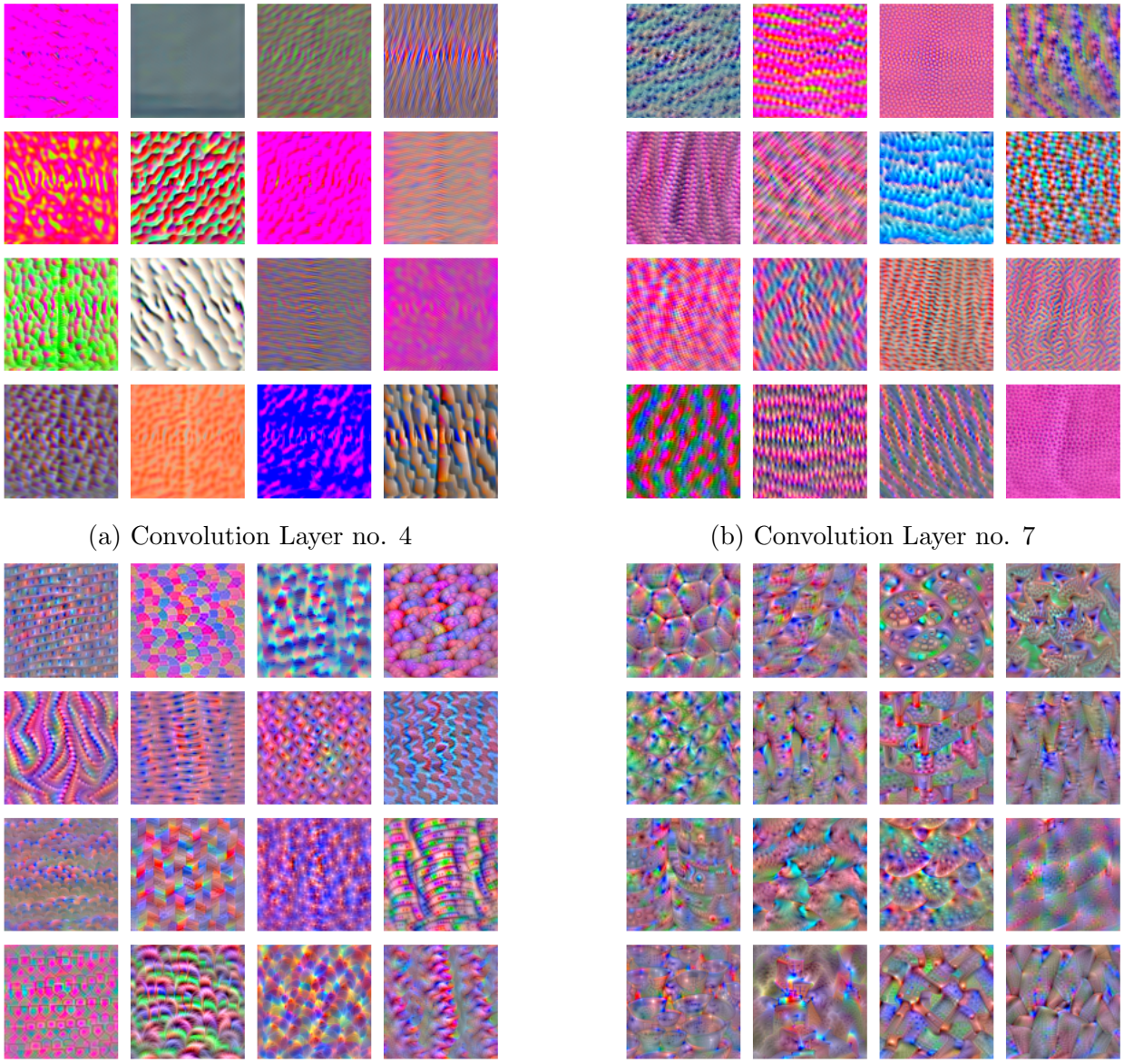
(a) Convolution Layer no. 1

(b) Convolution Layer no. 2

Figure 4: Convolution Layers

From figure 4, you can visualize all the 64 filters of the 1st two convolution layer each. In the 1st convolution layer we can observe that the filters are just smooth textures of different colors. Which means, it just tries to find which color patch is dominant when we pass the image to the model. While in the second convolution layer, we can see some strip patterns, i.e. apart from color it also tries to filter out some patterns. While these convolution layers are important, we can observe that they are quite simplistic, and therefore not good enough by themselves. Which is why we need to add more convolution layers to the model.

In figure 5, we tried to visualize random 16 filters from the 4th, 7th, 10th and the last convolution layers. In the previous figure, we observed single color and in some places 2 color dominated the filters. But here we start to observe a range of colors within a single filter itself. Also, as the layer gets deep, the complexity of pattern also starts to increase. Here we can see that how convolutions are use to detect edges and patterns like fish scales, eyes etc.



(a) Convolution Layer no. 4

(b) Convolution Layer no. 7

(c) Convolution Layer no. 10

(d) Convolution Layer no. 13

Figure 5: Convolution Layers

Attentions

Attention Images are the images which represents the contribution of each pixel in the image to the prediction. There are several ways to visualise this contribution and in this project we'll be focusing on two methods called *SmoothGrad* and *GradCam*. Each of these methods (in some sense) use gradients of the prediction with respect to the input pixels to analyse how the perturbations in the input affect the prediction. Using the same classes of **Great White Shark**, **Bald Eagle** and **Assault Rifle** (Shown in Figure 6, along with their probability distribution over top 5 Categories) let's see what part of images influence outputs the most.

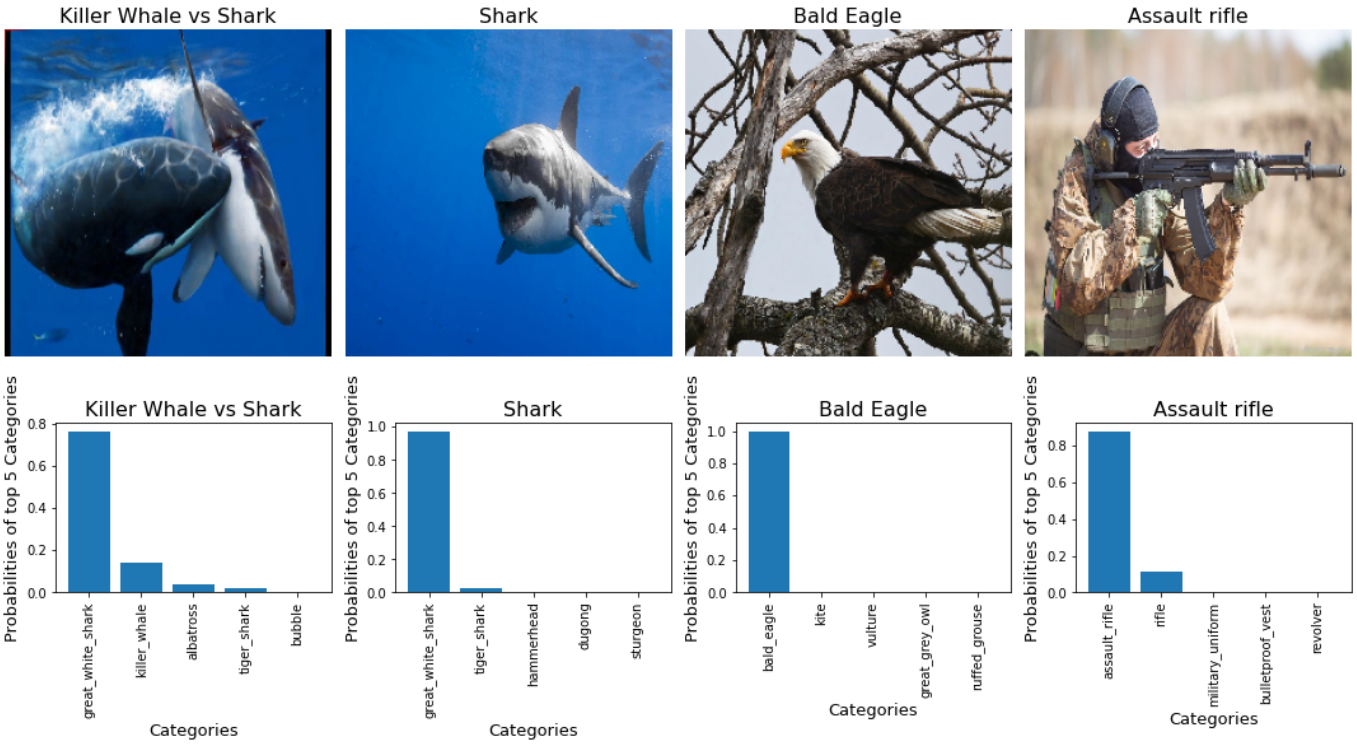


Figure 6: Images used for testing and their corresponding probability distribution over top 5 Categories

SmoothGrad

From Figure 7, we can see that in case of Shark, the model is looking at the fins and shape of the mouth of the shark the most. In case of Bald Eagle, the unique beak of the bird is influencing decision the most, and finally in assault rifle it was important to ensure that the model is infact looking at the gun and now something like the dress of the man (there are several instances recorded where model is predicting correctly but not looking at the features we want it to see).

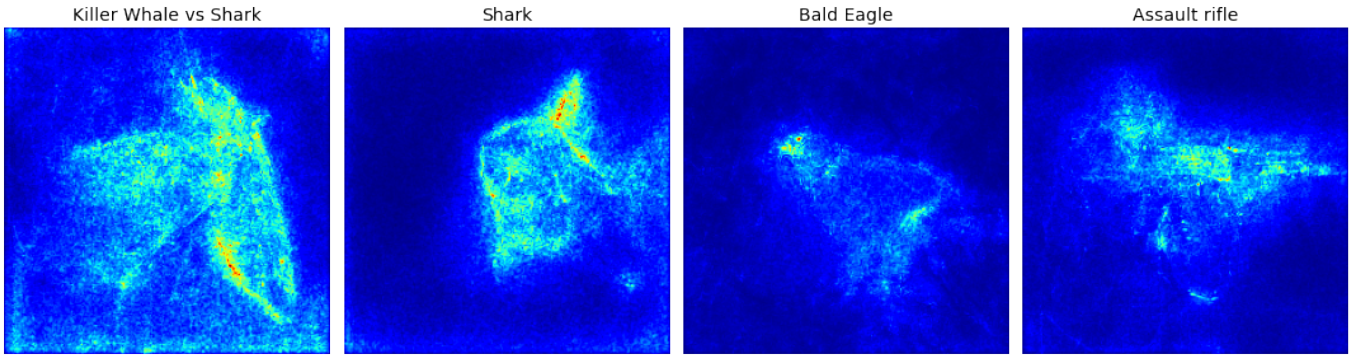


Figure 7: Results from SmoothGrad algorithm

GradCam

In the previous method, we could see the region affecting the predictions the most. However, it would be nice to see some kind of a heatmap over the image to visualise the same thing. Figure 8 shows the highlighted region over the image. These results align with the ones in the previous section. However it's interesting to see that model is confused in the 1st image where we can see blurry patterns on the whale.

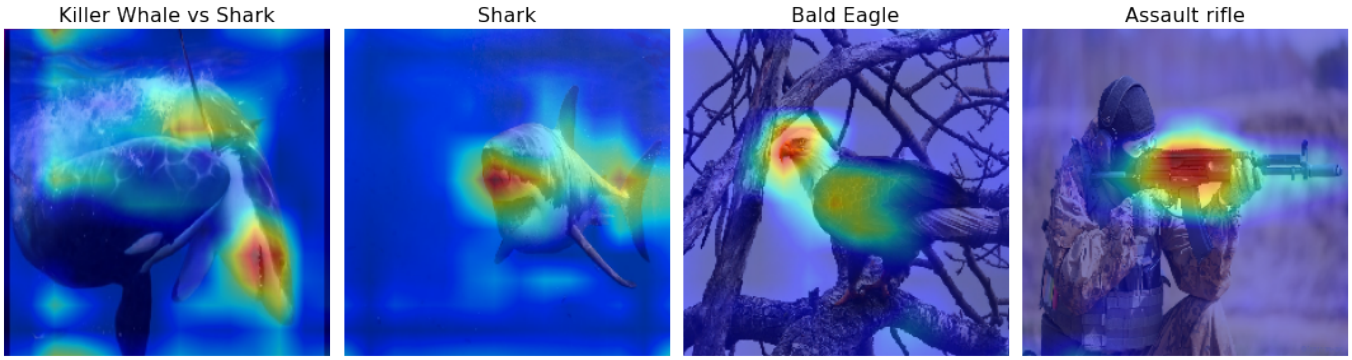


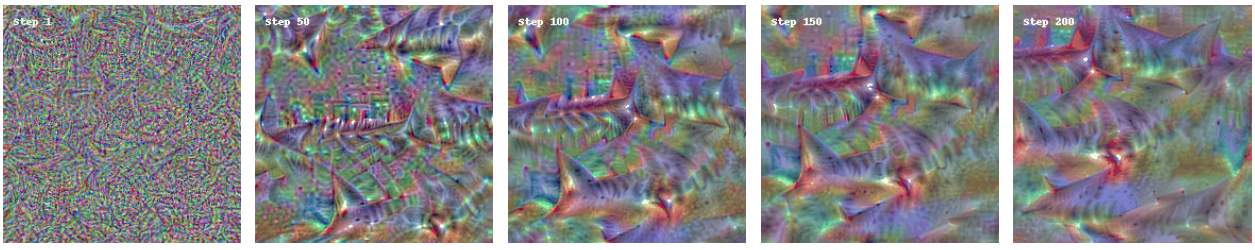
Figure 8: Results from GradCam algorithm

Animations

Sometimes a picture, or an animation in this case, is worth a thousand words. Here we tried to visualize the evolution of the last layer of the neural network by activating a particular neuron (category) at a time. Since it's not possible to add a video animation in latex or pdf, we included intermediate snapshots of the animation. You can access the actual animation on this Github page.

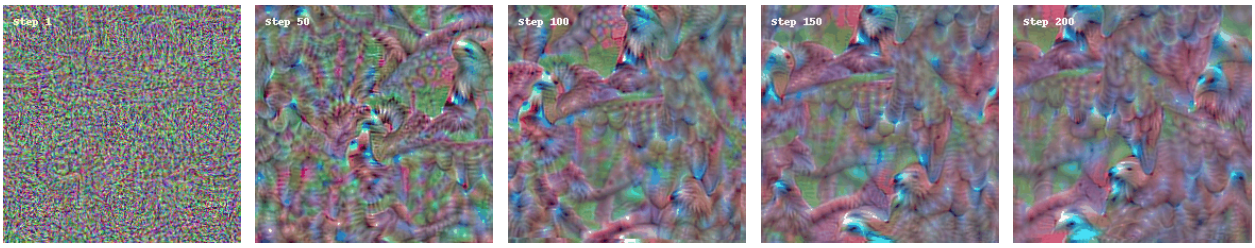
In figure 9, using maximal activation we are able to observe the gradual effects of training of the great white shark. As seen in epoch 1, we feed the model a random Gaussian-distributed RGB image. Based on what category we decide to activate, we start seeing the characteristics of the layer in the input image. Basically, we are trying to reverse engineer the input image based on the maximal activation of the category. Similarly, we also showcased intermediate steps for the bald eagle and assault rifle in figure 10 and 11 respectively.

Note: The dimension of the animations are a little small since VGG16 model accepts images of dimensions 224x224 px.



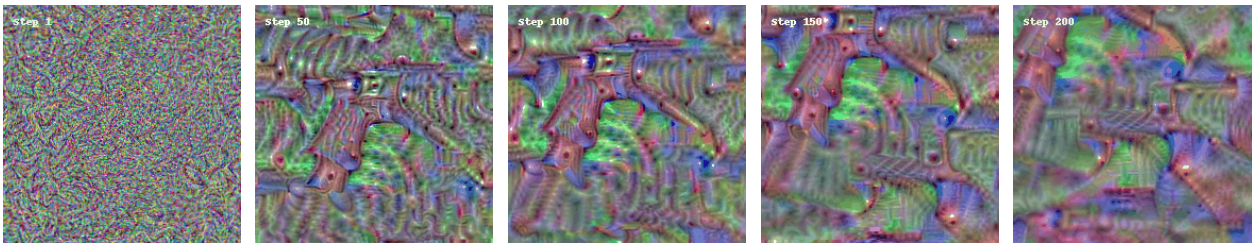
(a) Epoch 1 (b) Epoch 50 (c) Epoch 100 (d) Epoch 150 (e) Epoch 200

Figure 9: Great white shark animation stages



(a) Epoch 1 (b) Epoch 50 (c) Epoch 100 (d) Epoch 150 (e) Epoch 200

Figure 10: Bald eagle animation stages



(a) Epoch 1 (b) Epoch 50 (c) Epoch 100 (d) Epoch 150 (e) Epoch 200

Figure 11: Assault Rifle animation stages

Image Occlusion

In this section, we are going to discuss the **Image Occlusion** technique we have adopted, how these occlusions affect the prediction probabilities, and visualise the regions and heatmap of features that are affecting the predictions of these images into different categories in VGG16 Model.

We have created two sets of occluded images in which we have tried to occlude different important regions of the images with rectangular patches of grey color. In Figures 12, 13, 14, we have presented how the probability distribution of the predictions change from the original images, how the outputs of SmoothGrad and GradCam change after using Occluded Image set 1. It can be observed that the Probabilities have changed and the categories having the top 5 probabilities have also changed. In 12, the image where a Shark is being attacked by a Killer Whale, we have occluded the face of the Shark, hence, the probability of the image being of a Killer Whale increased, reducing the probability of being predicted as Shark. In Figure 13 (Output of SmoothGrad), we can observe that the face of the Shark was an important region in prediction as occluding that confused the model a bit as the focus was shifted towards the Killer Whale slightly, but the fin portion of the Shark still served as a significant region. Occluding the lower fins and jaws reduced the probability of being identified as a Shark more than 1%, and upper fins and tails were primary focus. When we occluded the wings of the Bald Eagle, the model could still identify the Bald Eagle with high probability as the beak that is an important feature is clearly visible. For Assault rifle, we occluded the middle portion having trigger, receiver and magazine. This occlusion lowered the prediction probability as thus we can infer that it was a significant region that

determines the prediction of Assault Rifle; so the model focused on the barrel and also on the man holding the rifle. The Figure 14 (Output of GradCam) shows the heatmap of the relevant regions of focus and also conveys the same explanation as mentioned above and makes the visualisation more informative.

In Figures 15, 16, 17, we have tried the experiment with another set of Occluded images where we occluded different regions than the ones discussed previously. In Figure 15, we occluded the middle portion of the body of the Shark, the end portion and tail of the Shark, head of the Bald Eagle, and Barrel and Muzzle of the Assault Rifle. Occluding the middle portion of the Shark Body increased the probability a bit as the face was now visible. The occlusion of the tail portion did not have much effect on prediction probability. The head and the beak of the Bald Eagle is an important feature that the model focuses on; this can be inferred from lowering of the prediction probability after occlusion of that region. The muzzle portion occlusion did not affect the probability from the original image. In Figure 16 (Output of SmoothGrad) we can see that the model could still focus on the body of the Shark capturing its relevant features even after occlusion. Occluding the head of the Bald Eagle made the model focus more on the feathers and bodily features of the Eagle. Thus, reducing the prediction probability. The barrel and muzzle occlusion did not hamper the prediction probability much as the middle portion of the Assault rifle was clearly visible which served as the main region of focus in the model. Figure 17(Output of GradCam), provides the heatmap representation of the focused features. It can be observed that the jaws of the shark is a primary region that makes the prediction probability high. This representation makes the visualisation more aligned to the aforementioned discussion.

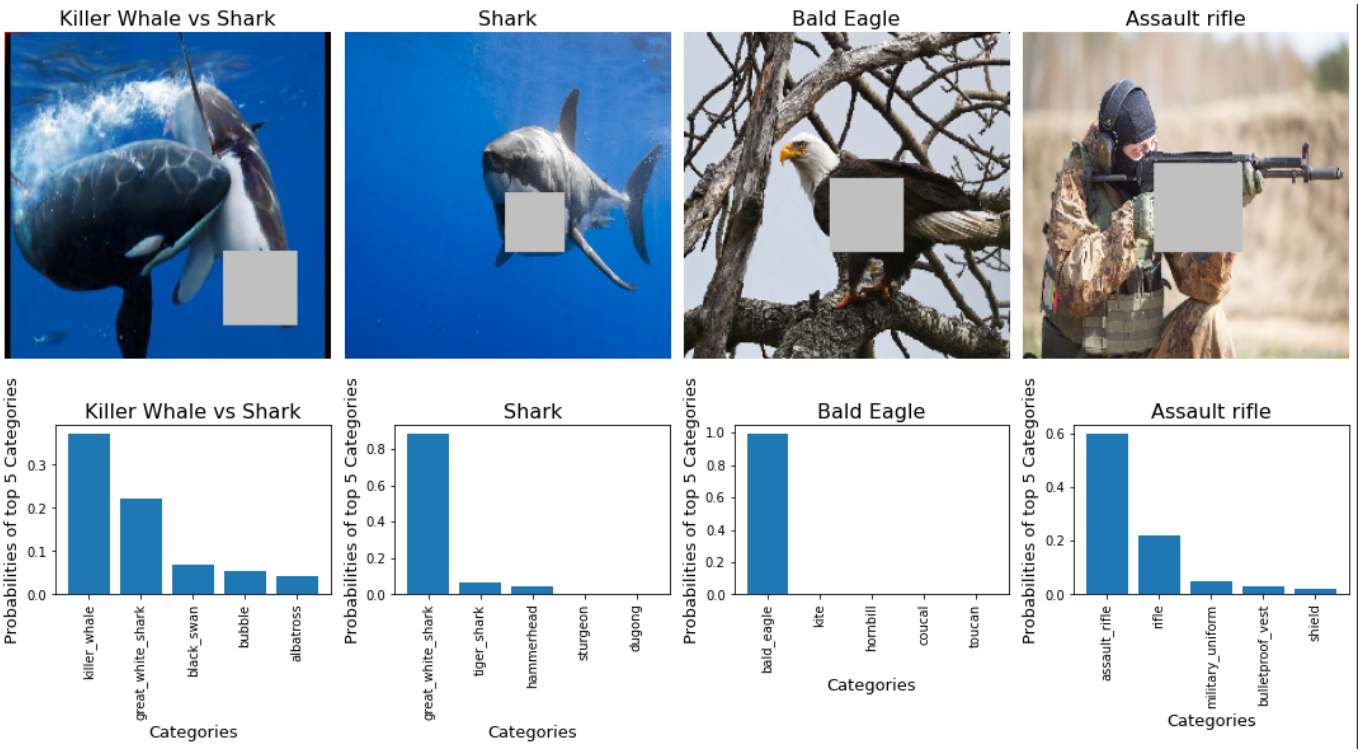


Figure 12: Occluded Image set 1 with its Probability Distribution over top 5 Categories

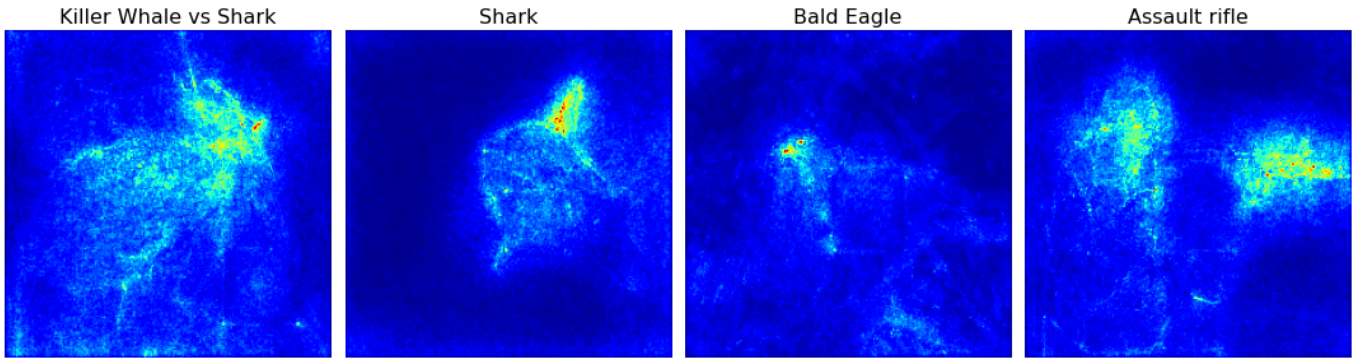


Figure 13: Results from SmoothGrad algorithm on the Occluded Image set 1

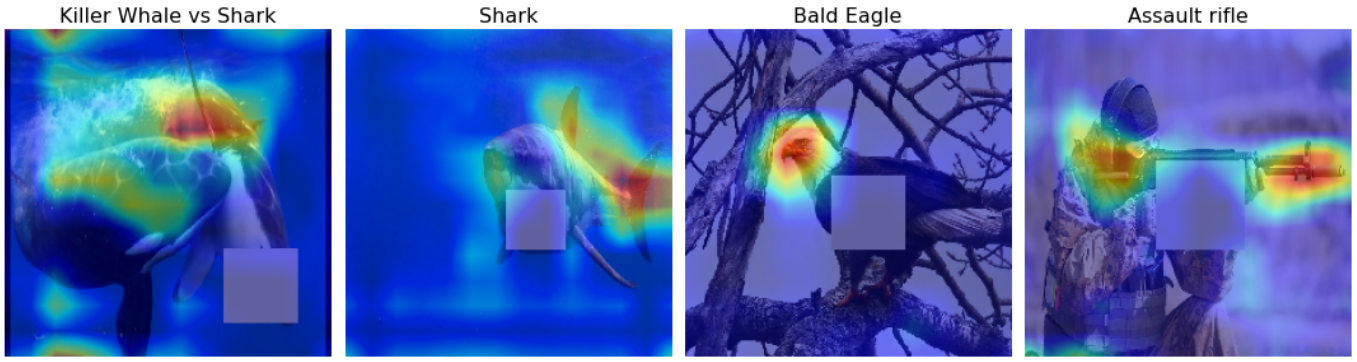


Figure 14: Results from GradCam algorithm on the Occluded Image set 1

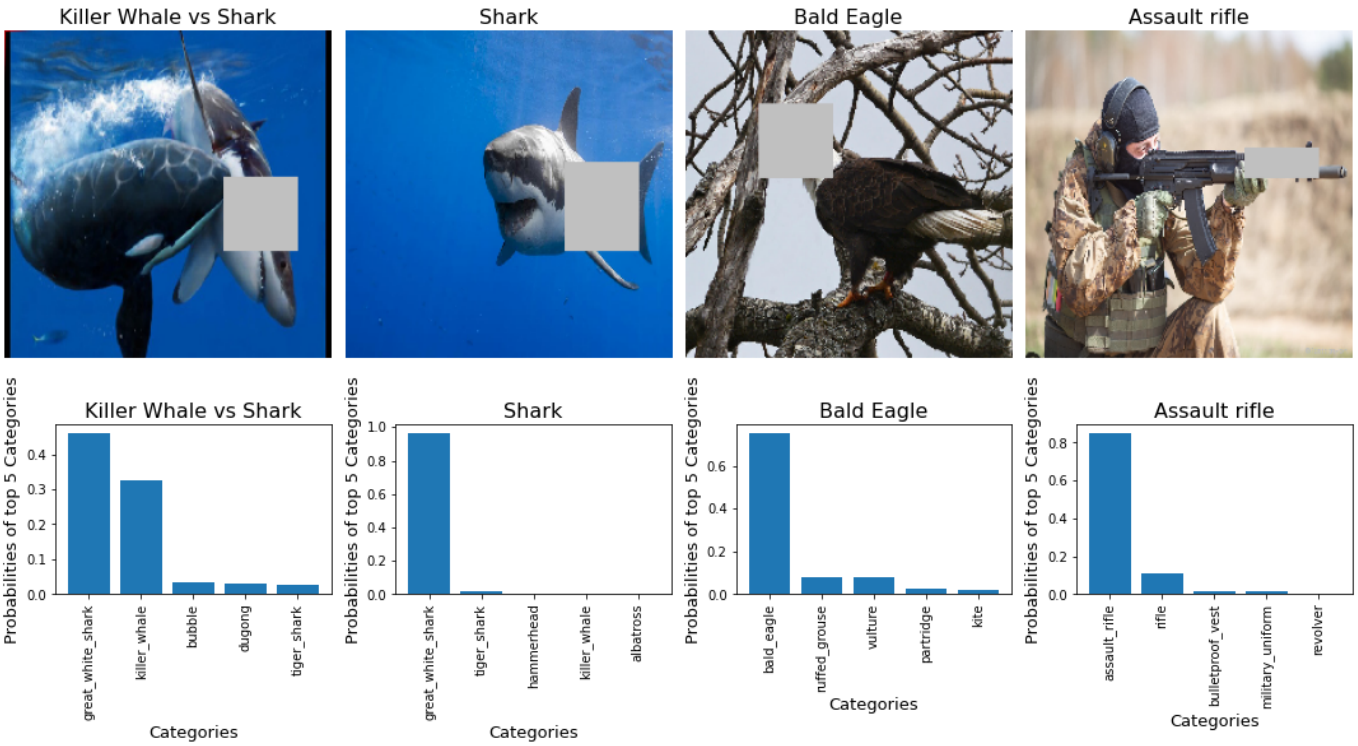


Figure 15: Occluded Image set 2 with its Probability Distribution over top 5 Categories

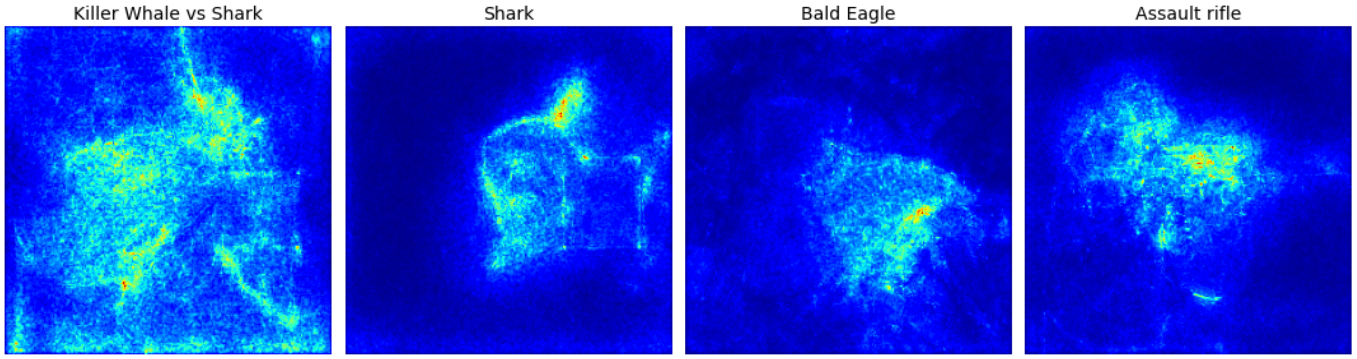


Figure 16: Results from SmoothGrad algorithm on the Occluded Image set 2

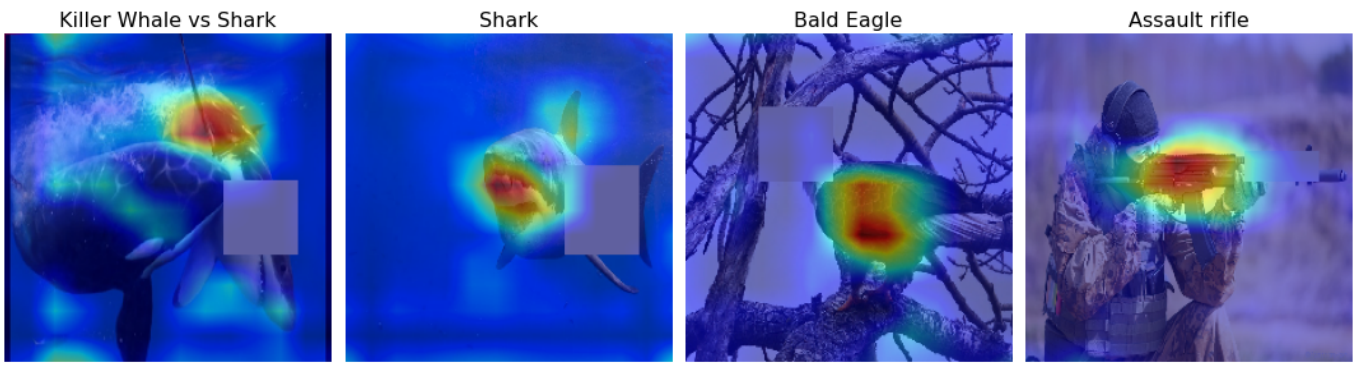


Figure 17: Results from GradCam algorithm on the Occluded Image set 2

Transfer Learning

Transfer learning (TL) is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. In this work, we used a Neural Network pretrained on Imagenet dataset and used transfer learning to tune it's parameters to make predictions on Tensorflow's Flower Dataset. From the visualisations in previous sections, we have shown that the early convolution layers only learn to detect broad features like edges, colours, etc, while the complex features are detected by deeper layers. Hence, in this test:

1. We took a pre-trained VGG16 Neural Network and removed all the fully connected layers at the last.
2. We then added a new set of fully connected layers on top of these convolutional layers.
3. Fixed the convolutional layers such that they do not train. By doing this we effectively reduced the number of parameters that we want to train.
4. Used flowers dataset to train the weights in fully connected layers.
5. Made predictions using this transfer learned model.

We have also contrasted the "normal" way of training, i.e. initialising the Neural Network with random weights/filters with the transfer learning. In the plot below, we have shown the training and validation loss curves for both cases together. Figure 18 shows the accuracy for both cases, where

1. Blue and Orange curve shows the results for transfer learning. Train Accuracy is the accuracy on training dataset while val accuracy is the accuracy on validation dataset.
2. Green and Red curve shows the results for normal training. Train Accuracy is the accuracy on training dataset while val accuracy is the accuracy on validation dataset.

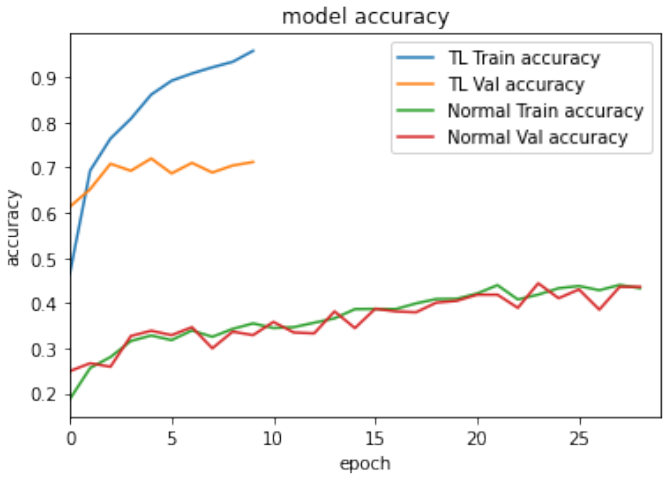


Figure 18: Training curve for Transfer Learning vs normal training.

From Figure 18, we can clearly see the advantage of using Transfer Learning where the training concluded in less than 10 epochs. Whereas for normal training, it took over 25 epochs and still the final accuracy was way below the transfer learned model. Apart from this, the accuracy on test dataset was very different as well where Transfer Learning got accuracy of around 89% while normal training struggled with 48% (which is worse than a random guess!).

Conclusion

In this section, we have discussed:

1. The questions answered in this project.
2. Success of the project.
3. Areas of improvement.

We used the pre-trained VGG16 model from ImageNet for our project. VGG16 is 16 layer convolution neural network model used in the The ImageNet Large Scale Visual Recognition Challenge (ILSVRC), an annual computer vision competition.

1. **Visualising learned filters:** In a deep neural network, it's commonly known that shallow layers learn simple features like edges while the deeper layers learn complex features like eyes. The very first task was be to demonstrate this and explain how the filters vary from layer to layer. We have shown this in section named Visualisation of Dense layers and Visualization of Conv2D layers.
2. **Maximal Activations:** In this part, we showed the areas of the input image that contribute most towards the prediction. You can find the details in section Attentions.

3. **Image Occlusion:** To verify that the heat-maps generated in the "Maximal Activations" section are correct, we altered the parts of the input image and observed how the predictions vary. Details in section Image Occlusion.

4. **Filter Animations:** Visualisation of the evolution of filters in several layers of a neural network was really interesting. We animated the features that triggers the activation of several filters over the time of training. Details in section Animations.

5. **Transfer Learning:** The last task was to show how a neural network can be retrained to predict a different dataset quickly. Details in section named Transfer Learning.

We were not able to reconstruct the patterns for the convolutional filters by feeding real or valid images to the model. For example, if we provide a dog picture to the model, edge filters trained to detect things like a dog's snout, ears, tail, etc., would get activated the most. Based on the maximal neuron activations, we would have a set of neurons, which we could then reverse engineer their pattern by using maximal activation techniques. That way, we would have been able to quantify the purpose of some of the filters. In this project, though, we did manage to reverse engineer the patterns, but it was done so by randomly selecting the neurons from the convolution layer.

References

1. Zeiler, M., Taylor, G., and Fergus, R. Adaptive deconvolutional networks for mid and high level feature learning. In ICCV, 2011.

2. Brown, T.B., Mané, D., Roy, A., Abadi, M. and Gilmer, J., 2017. Adversarial patch. arXiv preprint arXiv:1712.09665.

3. Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. arXiv preprint arXiv:1312.6034

4. Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. (2020). Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. International Journal of Computer Vision, 128(2), 336–359.

5. Wang, H., Wang, Z., Mardziel, P., Du, M., Yang, F., Hu, X., Zhang, Z., and Ding, S. (n.d.). Score-CAM: Score-Weighted Visual Explanations for Convolutional Neural Networks.

6. Chattopadhyay, A., Sarkar, A., Howlader, P., and Balasubramanian, V. (n.d.). Grad-CAM++: Improved Visual Explanations for Deep Convolutional Networks.