<u>Final Report: Dog and Cat Image Classification Using Convolutional Neural Networks(CNN)</u>

Soleil St Louis, Calvin Tran, Muhammad Khan, Prahalad Muralidharan

Georgia State University

CSC 6850

Abstract— In our project, we built a Convolutional Neural Network to classify dogs and cats using a pre-trained model w/ VGG16 and a 5-layer CNN model. Then later visualize the features for each model using the Grad-Cam method and Saliency Map to generate them on an input image. This project aims to visualize how CNN learns to identify different features present in images to provide a deeper understanding of how the model works. It will also help to understand why the model might fail to classify some of the images correctly and hence fine-tune the model for better accuracy and precision.

To get a feature map(in the form of a heat map) a Grad-CAM method will be used. It takes the original image and place the heat map on top of it. Gradient-weighted Class Activation Mapping (Grad-CAM), uses the gradients of any target concept (for example: 'cat') flowing into the final convolutional layer to produce a coarse localization map highlighting the important regions in the image for predicting the concept. We will be classifying dogs & cats with a dataset from Kaggle. The dataset consists of two classes: cat and dog. The images of the cats and dogs are in various backgrounds, and different levels of brightness, and the animals are in different poses. To get a saliency map we used the saliency map explanation method that is used for interpreting the predictions of convolutional neural networks (CNNs). The saliency map of an input image specifies parts of it that contribute the most to the activity of a specific layer in the network, or the decision of the network as a whole. The saliency map computes the effect of each pixel on the final prediction then generates a map that describes the important pixel that has influenced the prediction.

I. Introduction

Convolutional Neural Network (CNN) is an algorithm taking an image as input then assigning weights and biases to all the aspects of an image and thus differentiates one from the other. Batches of photos with labels identifying the true nature of each image can be used to train neural networks. There may be a few tenths to hundreds of photos in a batch. The network prediction is compared to the corresponding existing label for each individual image, and the gap between the prediction and the reality is calculated for the entire batch. The network settings are then changed to reduce the distance, improving the network's capacity for prediction. Every batch continues to receive the same training. For our data, we used a dataset from Kaggle, 'Dogs and Cats'. The dataset consisted of a total of 10,000 images. With the training set being a balanced set of 8000 images and, the testing set being a balanced set of 2000 images. The 'Dogs and Cats' dataset is a standard computer vision dataset that involves classifying photos as either containing a dog or cat. With this dataset we will use a pretrained model with VGG16 to learn various distinctive features of cat and dog and differentiate images of cats and dogs.

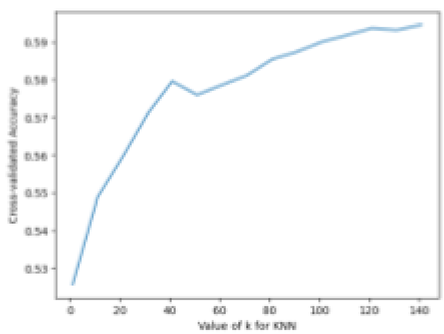
II. Methods/Experimental Results

For our first method, we used a KNN Model. The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. We pre-processed the data by changing the images into numpy arrays and creating 2 arrays one with only raw pixel data and another that extracts the color from the image to classify. The Raw

Pixel Data will contain 49152 pixels and the Feature Data will contain only 512 features. Then, split the data into Training(80%) and Testing(20%). We perform k-fold cross validation with 5 folds on the Training Set and calculate the average validation accuracy for each k value. This can be seen on Graph I and Graph 2 where the accuracy increased until k=70, then it started to level off. But it's important to know that the k was incremented by 20 starting from 1 to 141 to lower the time complexity.

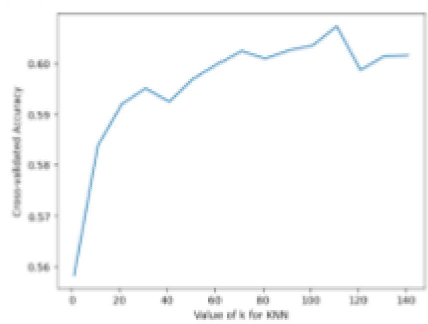
Graph I

Raw Pixel Data vs. K Values



Graph II

Extract Color vs. K Values



To evaluate the KNN model, it was observed that from k = 70+ is when it starts to converge and reaches the most optimal point and then begins to oscillate. It's hard to choose the "best" k, so to strike a balance between bias and variance, using square root of training size is good.

- Sqrt(8000) ≈ 89.

It's better to keep it odd to remove ties. Then, we perform k=89 on our testing set to see how the model performs with the most "optimal k".

As seen in Data I and Data II below, the k-89 model performs slightly better than randomly guessing. This shows that there are some correlations between the pixels. Also, the Extract Color array performs slightly better by around 3%. However, this could be due to imbalance in the dataset with maybe a color in the dataset is more common with dogs for example.

Data I
Raw Pixel Score

```
k value: 89
Validation accuracy: 59.50%
```

Data II
Extract Color Score

```
k value: 89
Validation accuracy: 62.85%
```

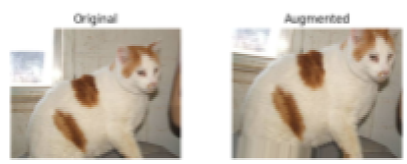
For our second method, we used a 5-layer CNN Model. We pre-processed the data by generating real-time batches of augmented images for training. We added rescaling, shearing, zoom, horizontal flip, width shifts, height shift, and rotation to the images. Below are some examples of the augmented images. Also, for validating and testing, there is no need to augment the images.

Image I

dog.3048.jpg

Image II

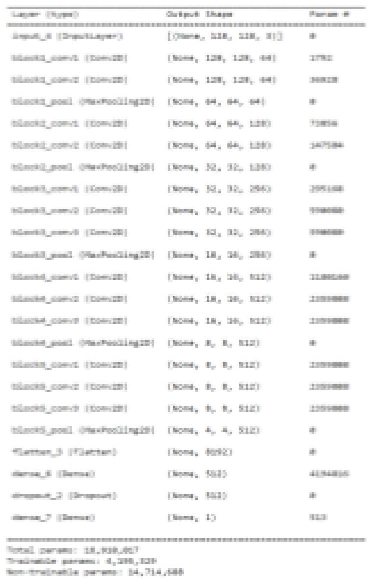
cat.3772.jpg



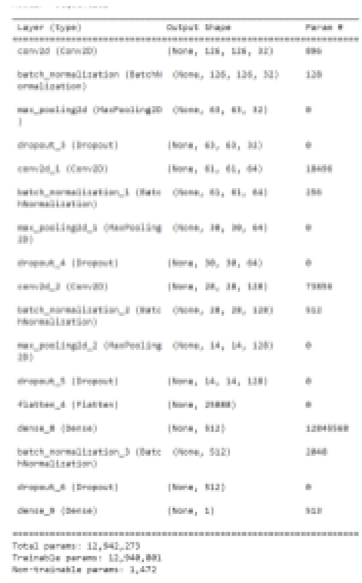
With our CNN model we used a pre-trained model with VGG16 by:

- Flatten the last layer into 1D.
- Create a dense layer with 128 nodes as weights to train on.
- Added dropout for regularization
- The last layer classified with sigmoid function to 2 classes

Model I
CNN Model w/ Pretrained VGG16 Model



Model II
CNN Modeh w/o Pretrained



To validate our model, we used 5-folds cross validation. Each fold will run 20 epochs on the model. Our loss function is binary_crossentropy. We are using Adam(Adaptive Moment Estimation) as the optimizer, an extension to stochastic gradient descent, where it does not have a fixed learning rate. We also included hyperparameter tuning on the dropout for each CNN model and performed cross-validation and ran 20 Epoch.

Table I

CNN Model w/ VGG-16 Cross-Validation Accuracy Hyper Parameterization Dropout Value

No Dropout	Validation Loss	Validation Accuracy
Fold 0	0.245606	0.8960
Fold 1	0.240518	0.8975
Fold 2	0.237667	0.9025
Fold 3	0.241967	0.9035
Fold 4	0.235723	0.9045
Average:	0.2403	0.9008

Dropout-0.1	Validation Loss	Validation Accuracy
Fold 0	0.237963	0.8970
Fold 1	0.247423	0.8980
Fold 2	0.250788	0.8990
Fold 3	0.239205	0.8975
Fold 4	0.241180	0.9000
Average:	0.2433	0.8983

Dropout-0.2	Validation Loss	Validation Accuracy
Fold 0	0.240569	0.9025
Fold 1	0.241613	0.8980
Fold 2	0.241613	0.8980
Fold 3	0.253671	0.8955
Fold 4	0.240958	0.8985
Average:	0.2444	0.8984

Dropout-0.3	Validation Loss	Validation Accuracy
Fold 0	0.235808	0.9035
Fold 1	0.241638	0.8970
Fold 2	0.242490	0.8985
Fold 3	0.230783	0.9010
Fold 4	0.238538	0.9050
Average:	0.2379	0.9010

Dropout-0.4	Validation Loss	Validation Accuracy
Fold 0	0.241618	0.8960
Fold 1	0.243799	0.8965
Fold 2	0.241257	0.9000
Fold 3	0.237769	0.9045
Fold 4	0.239749	0.9040
Average:	0.2408	0.9001

Dropout-0.5	Validation Loss	Validation Accuracy
Fold 0	0.236241	0.8985
Fold 1	0.242323	0.8955
Fold 2	0.245269	0.8975
Fold 3	0.244413	0.8965
Fold 4	0.240657	0.9055
Average:	0.2418	0.8987

Table II

5-Layer CNN-Model Cross-Validation Accuracy Hyper Parameterization Dropout Value

No Dropout	Validation Loss	Validation Accuracy
Fold 0	0.356758	0.8575
Fold 1	0.328451	0.8670
Fold 2	0.355242	0.8650
Fold 3	0.473198	0.8190
Fold 4	0.397778	0.8470
Average:	0.3823	0.8511

Dropout-0.1	Validation Loss	Validation Accuracy
Fold 0	0.299898	0.8770
Fold 1	0.369503	0.8525
Fold 2	0.406691	0.8255
Fold 3	0.333443	0.8685
Fold 4	0.373018	0.8505
Average:	0.3565	0.8548

Dropout-0.2	Validation Loss	Validation Accuracy
Fold 0	0.458669	0.8270
Fold 1	0.343138	0.8650
Fold 2	0.555311	0.7965
Fold 3	0.398429	0.8495
Fold 4	0.434637	0.8185
Average:	0.4380	0.8313

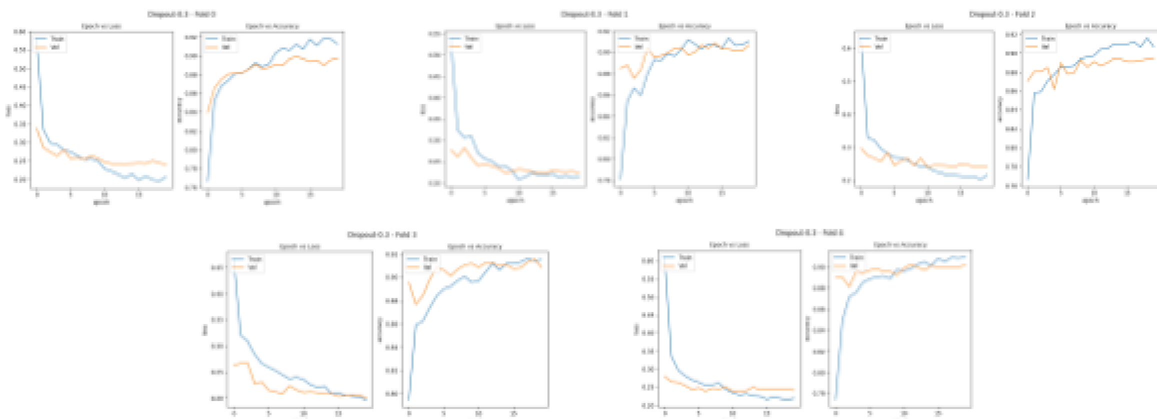
Dropout-0.3	Validation Loss	Validation Accuracy
Fold 0	0.562274	0.8065
Fold 1	0.576429	0.7995
Fold 2	0.468413	0.8080
Fold 3	0.624559	0.7700
Fold 4	0.569450	0.8120
Average:	0.5602	0.7992

Dropout-0.5	Validation Loss	Validation Accuracy
Fold 0	0.553542	0.7715
Fold 1	0.579628	0.7705
Fold 2	1.410269	0.6820
Fold 3	0.513357	0.7735
Fold 4	0.584390	0.7605
Average:	0.7283	0.7516

In Table I, the dropout has a little or no effect in the model whereas in Table II increasing the dropout value has a significant decrease in validation accuracy. Our prediction was that by adding regularization, it had a harder time finding the pattern in the data causing it to have lower validation accuracy. This can be seen in Graph IV where it started to overfit in the dropouts but lessen as the dropout value increases.A solution may be to increase the epoch to have more time to learn, but due to our time constraint, we weren’t able to increase the epoch. In the CNN model w/ the pretrained VGG-16 model, the model performs much better than the 5-Layer CNN model by 5% comparing the best average cross-validation accuracy. Dropout 0.3 performs the best for VGG16 Model and Dropout 0 performs the best for 5-Layer CNN model. It's also good to note that around dropout 0.3 for 5-Layer CNN model without VGG16, it significantly lowers the validation accuracy by 5%.

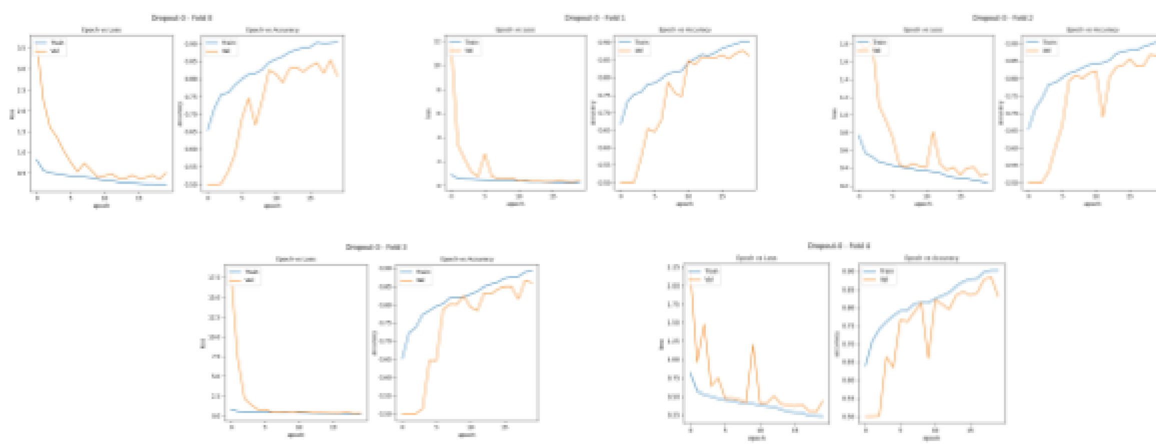
Graph III

VGG16 Model Dropout 0.3 Loss and Accuracy vs. Epoch



Graph IV

5-Layer CNN Model w/o Pre-trained Dropout 0 Loss and Accuracy vs. Epoch



In Graph III, VGG16 Model Dropout 0.3, the best model was in fold 4 and in Graph IV, 5- Layer CNN Model, the best model was in fold 1. In both models, it was clear to see an overfitting issue where the training accuracy continues to increase where the validation accuracy stays leveled.

For our third method we used the Grad-CAM(Gradient-weighted Class Activation Mapping) and Grad-CAM++ technique. Grad-CAM is a popular technique for creating a class-specific heat-map based off of a particular input image, a trained CNN, and a chosen class of interest. It visualizes where a convolutional neural network model is looking. It's closely related to CAM. The Grad-CAM technique utilizes the gradients of the classification score with respect to the final convolutional feature map, to identify the parts of an input image that most impact the classification score.

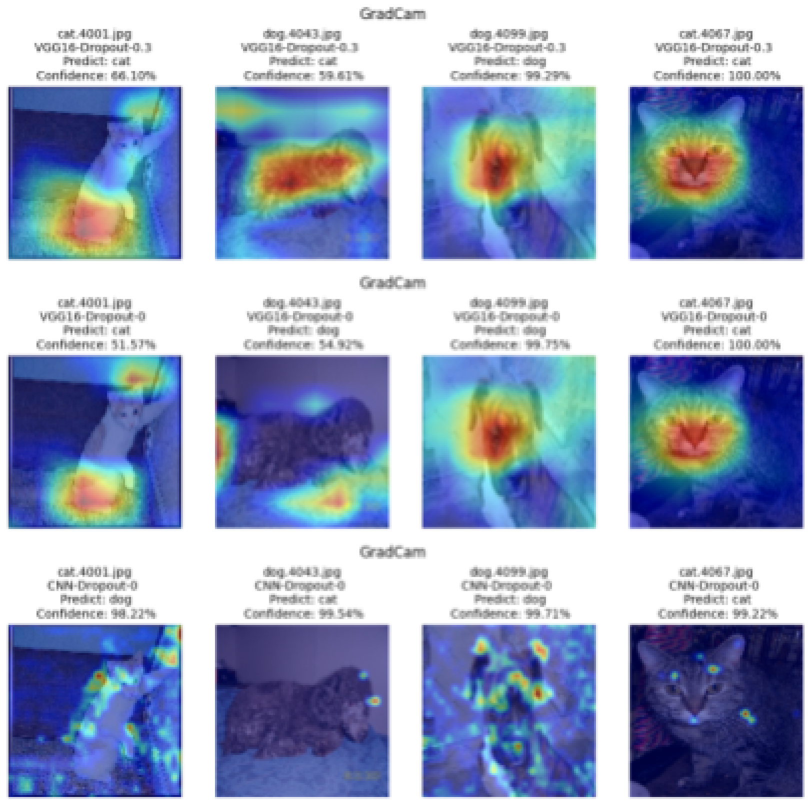
Classification score being the ratio of number of correct predictions to the total number of input samples. The heat-map should display the most accurate visual explanation of the object being classified by the model.

Image III

Image Prediction for VGG16-Model and 5-Layer CNN Model

Image IV

GradCAM Output



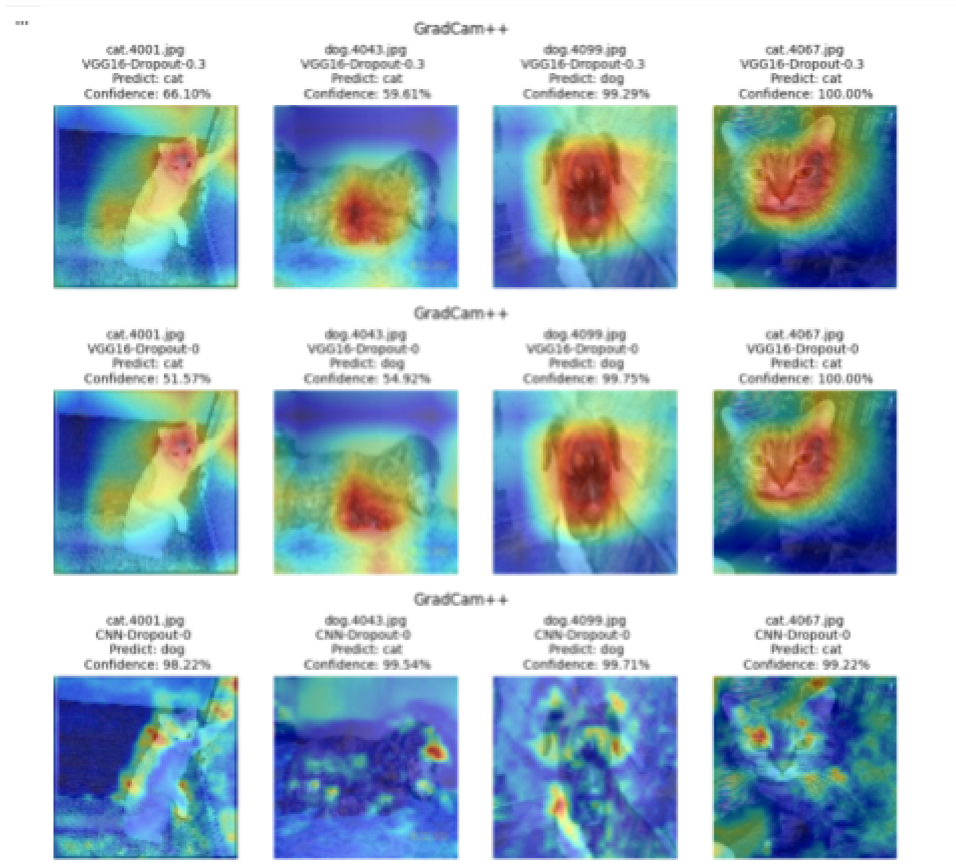
In Image IV, we see that the upper face of dog.4043, dog.409, cat.4067 has the greatest impact on the classification.

Problems that we found with using Grad-Cam was that, although Grad-CAM is supposed to be able to explain what regions of an image a model used for prediction, it turns out that Grad-CAM is not guaranteed to do this. In brief, because of the gradient averaging step, Grad-CAM's heat maps do not reflect the model's computations and can highlight irrelevant areas that weren't used for prediction. For example in Image IV, it shows that the lower body of cat4001.jpg has the greatest impact on the classification score which is irrelevant.

To fix this problem, we used Grad-CAM++, built on Grad-CAM, because it provides better visual explanations of CNN model predictions, in terms of better object localization as well as explaining occurrences of multiple object instances in a single image. This can be seen in Image V:

Image V

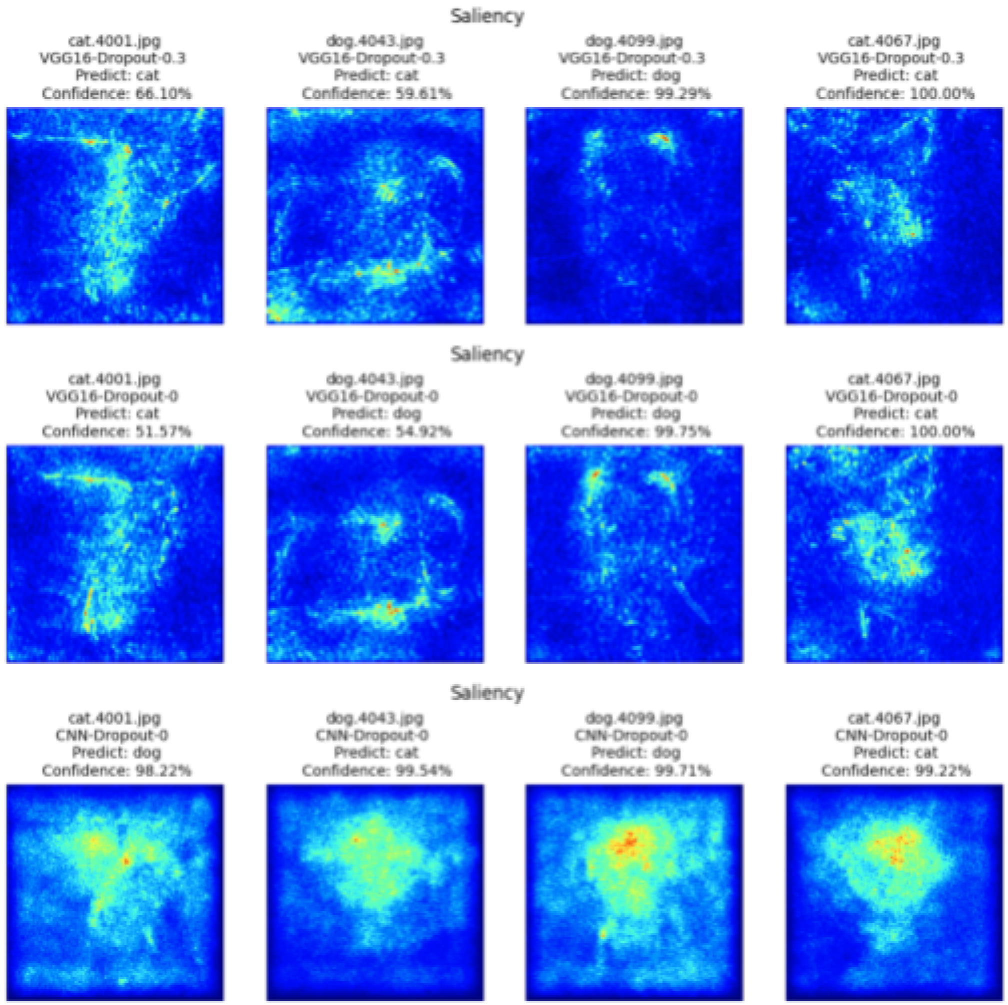
GradCAM++ Output



For our fourth method we used the Saliency Map. Saliency map is an explanation method used for interpreting the predictions of convolutional neural networks (CNNs). The saliency map of an input image specifies parts of it that contribute the most to the activity of a specific layer in the network, or the decision of the network as a whole. It also computes the effect of each pixel on the final prediction then generates a map that describes the important pixels that have influenced the prediction.

Image VI

Saliency Map Output



For example, according to Image VI, prediction of the cat and dog is highly influenced by the pixel around their faces.

III. Conclusion

In conclusion, we've been able to successfully develop a CNN model for image classification. With the help of the cats and dogs dataset, we've explored how a model differentiates classes. We've used Grad-CAM++ to zero in on areas of bias and weakness in the model. By generating heat maps that highlight the most important regions of an image for a given classification decision, Grad-CAM and Grad-CAM++ can provide insight into how your model is making its predictions. Only a couple problems were found with using the CNN Model. For the 5-Layer we found it has a low training and validation accuracy score meaning that the model is not complex enough. Since in this model, we are not using transfer learning, the model is using a small dataset to train on therefore causing a low accuracy. Another problem was that sometimes on each fold the model is overfitting.