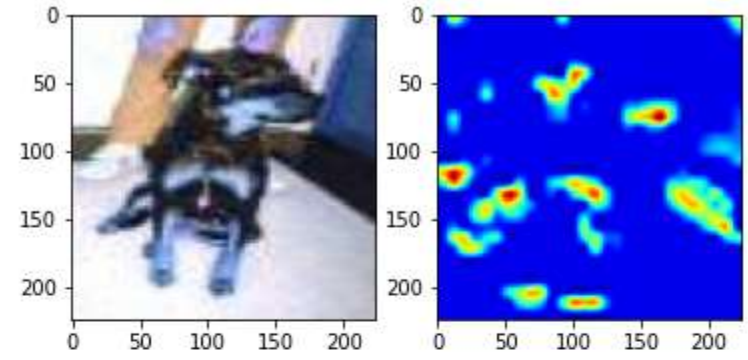
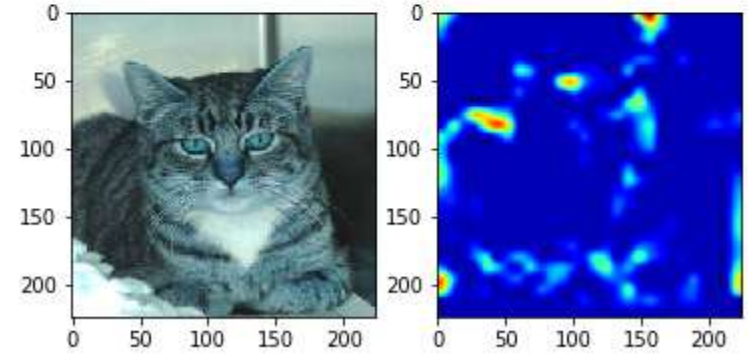
CSE 700: Master's Project

Applying Deep Learning Techniques and Understand the CNN with Grad-CAM Visualization techniques.



dog



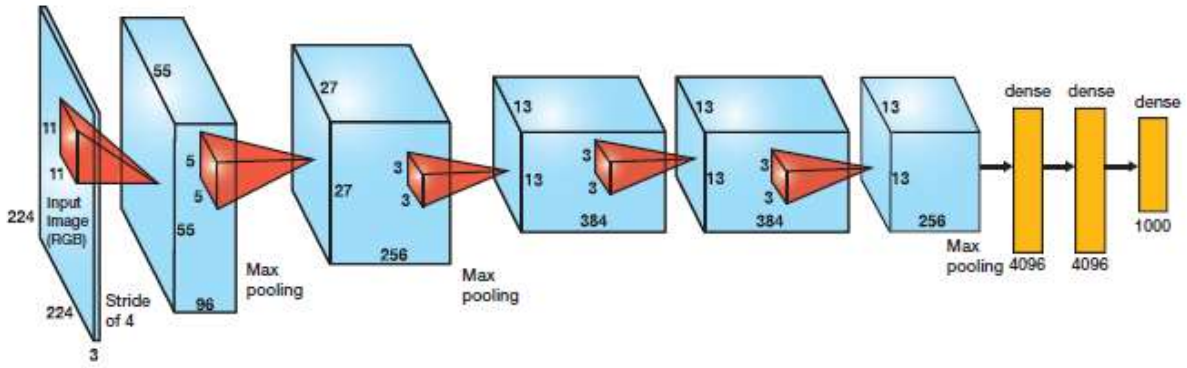
cat

Jay Bakshi
jaybaksh@buffalo.edu
UB Person Number: 50206954
Department of Computer Science & Engineering
SUNY University at Buffalo

AlexNet

The paper was presented by Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton, with University of Toronto – to win the ILSCVRC-2012 competition, successfully classifying the 1.2 million high-resolution images into 1000 different classes with the best top-1 and top-5 error rates compared to then state-of-art neural network architectures.

The architecture proposed an 8-layer structure, and was one of the first deep neural networks able to push ImageNet classification accuracy by a considerable amount in comparison to the previous traditional approaches. Basically, the 8-layer structure is composed of 5 convolutional layers followed by 3 of fully-connected layers also known as dense layers. An image representation can be seen in the Figure 1 below –



AlexNet used Rectified Linear Unit (ReLU) for the non-linear part instead of a Tanh or Sigmoid functions, which were the standard choice for previous traditional neural network approaches. ReLU is given by –

$$f(x) = \max(0, x)$$

The direct advantage of using ReLU is that it trains much faster in comparison, this is because the derivative of sigmoid becomes very small in the saturating region and causing the updates to the weight almost vanishing. This phenomenon is called as vanishing gradient problem. In AlexNet, ReLU layer is inserted after each and every convolutional and fully-connected layers(FC).

AlexNet architecture also uses Dropout layer, after each FC layer in the neural network. Dropout helps avoid overfitting because of 2 reasons the way Dropout works – (1) Dropout layer enables switching off different sets of neurons, representing different architectures which get trained in parallel. This means there is a structured model regularization occurs, which helps overfitting. (2) Since neurons being selected are at random, this layer also helps avoid developing co-adaptation between layers, making them extract meaningful features, independent of others.

Code

```
# MODEL ARCHITECTURE
model = Sequential()

# Layer 1
model.add(Conv2D(filters=96,kernel_size=(11,11),strides=4,padding='val
id',activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(3,3),strides=2))

# Layer 2
model.add(Conv2D(filters=256,kernel_size=(5,5),activation='relu',strid
es=1))
model.add(MaxPooling2D(pool_size=(3,3),strides=2))

# Layer 3
model.add(Conv2D(filters=384,kernel_size=(3,3),activation='relu'))
model.add(Dropout(0.25))

# Layer 4
model.add(Conv2D(filters=384,kernel_size=(3,3),activation='relu'))
model.add(Dropout(0.25))

# Layer 5
model.add(Conv2D(filters=256,kernel_size=(3,3),activation='relu'))
model.add(Dropout(0.25))
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

# Pool Flatten
model.add(Flatten())

# Layer 6
model.add(Dense(4096,activation='relu'))

# Layer 7
model.add(Dense(4096,activation='relu'))

# Layer 8
model.add(Dense(4096,activation='relu'))
model.add(Dropout(0.25))

# Final Layer
model.add(Dense(num_classes,activation='softmax'))
```

VGG16

This model is presented by Karen Simonyan and Andrew Zisserman working at Visual Geometry Group (VGG), University at Oxford. VGG Net is important and influential for Deep Learning Models since it proved that for Convolutional Neural Networks when have deep networks of layers give better results. The notion popularly is – "Keep it deep. Keep it simple."

The original network architecture presented was 19-layer configuration. However, for our implementation we have stuck to the 16-layer configuration model (seen highlighted in the Figure 2 below).

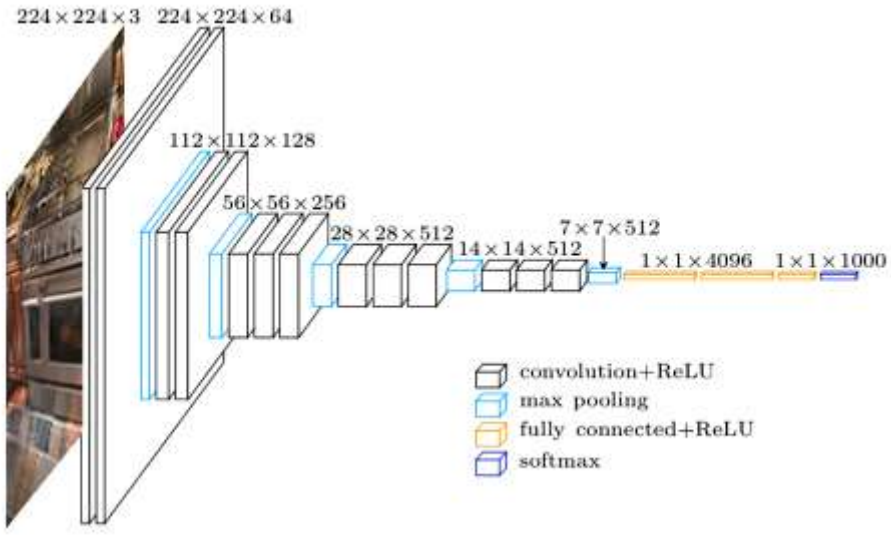
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB imag)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

The 6 different architecures of VGG Net. Configuration D produced the best results

The model is inspired by the success of AlexNet, the main difference being – using a combination of only 3x3 sized filters instead of 11x11 filters in AlexNet. The direct benefit of this change is that while the effective receptive field is similar in performance to larger filter, there is significant decrease in the number of parameters, thus the motto keeping it simple by increasing the depth by combination of

smaller filters. The number of filters doubles after each Maxpool layer, reinforcing the idea of reduction in spatial dimensions but increment in depth. ReLU layers have been used after each convolutional layer and trained with batch gradient descent.

The results secured 1st and 2nd places in localization and classification challenged for ILSVRC-2014.



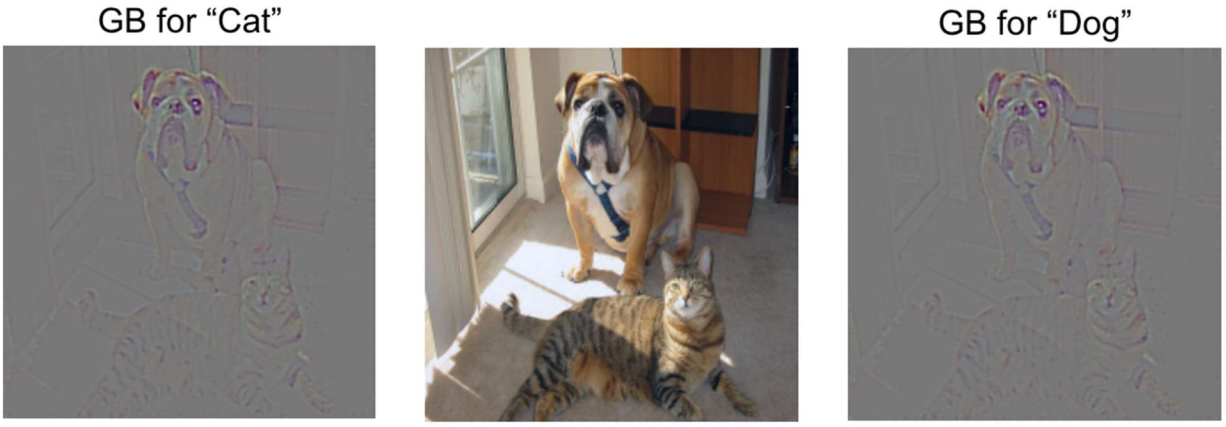
Code

```
# MODEL ARCHITECTURE
model = Sequential()

# Conv Block 1
model.add(Conv2D(64, (3, 3), input_shape=(224,224,1),
activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# Conv Block 2
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# Conv Block 3
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# Conv Block 4
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# Conv Block 5
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# FC layers
model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dense(4096, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

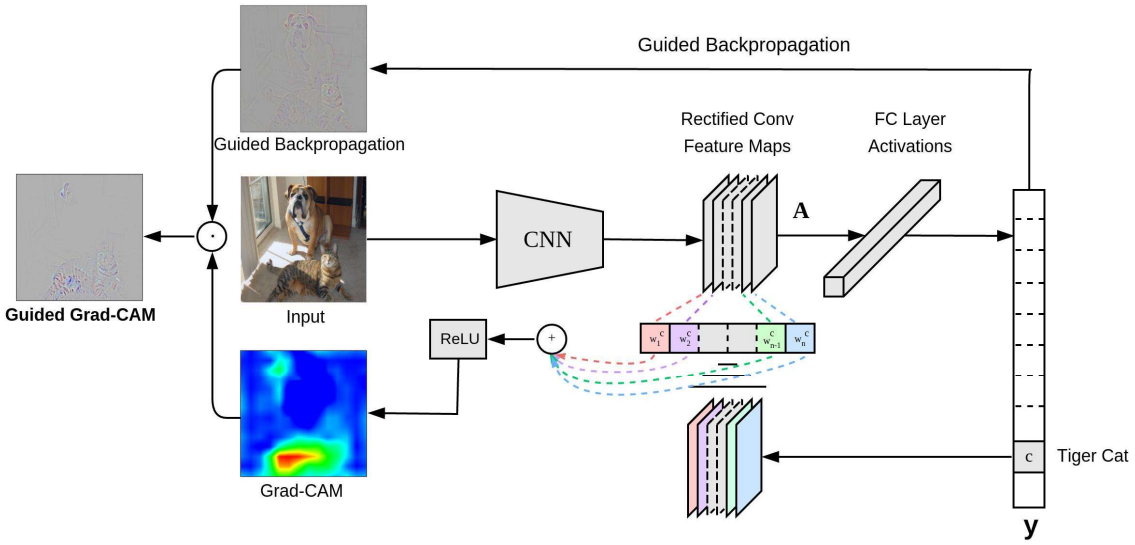
Grad-CAM

Convolutional Neural Networks and other Deep Learning techniques have proved invaluable in image classification and other tasks. However, it is equally well-know that when these techniques fail, they fail in a very disgraceful manner. The reason for failure is hard to understand, thanks to incoherent output of error without any warning or explanations. This lack of decomposability into intuitive and understandable components makes these CNN very hard to interpret and hence hindering the correction.

Traditional approaches for granting this interpretability to CNN is with Backpropagation, Deconvolution and Guided-Backpropagation techniques. Although, Deconv and Guided-Backpropagation have produced cleaner results, we still need better transparent interpretable systems.



It is a known fact that deeper the CNN, more of the higher-level visual concepts are captured. Class Activation Mapping (CAM) is a technique where in the network Convolutional layers are directly followed by a prediction layer. This helps us get the high-level semantics and the detailed spatial information. The generalized version of this technique is Grad-CAM (Gradient –weighted CAM), which can be applied to any CNN-based architecture. An even more advanced version of Grad-CAM is Guided Grad-CAM that in addition to being class-discriminative is also capable to showing visualization to predicting specific instance of the classes. E.g. ‘Tiger cat’ is an instance of class ‘cat’.


Guided Backpropagation
Guided Backpropagation
Rectified Conv Feature Maps
FC Layer Activations
A
CNN
Guided Grad-CAM
Input
ReLU
Grad-CAM
Tiger Cat
y


Grad-CAM for "Cat"
Grad-CAM for "Dog"


Guided Grad-CAM for "Cat"
Guided Grad-CAM for "Dog"

Code

```
from keras.layers import GlobalAveragePooling2D
# get layers and add average pooling layer
vgg_model.layers.pop()
x = vgg_model.layers[-1].output
x = GlobalAveragePooling2D()(x)

from keras.layers import Dense
#output layer
prediction_layer = Dense(2, activation='softmax')(x)

from keras import Model
model = Model(inputs=vgg_model.input, outputs=prediction_layer)

# freeze pre-trained model area's layers
for layer in vgg_model.layers:
    layer.trainable = False

from keras.optimizers import RMSprop
from keras.losses import categorical_crossentropy
# update the weight that are added
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
model.fit(training_images, training_onehot_labels)

# choose the layers which are updated by training
layer_num = len(model.layers)
for layer in model.layers[:15]:
    layer.trainable = False
for layer in model.layers[15:]:
    layer.trainable = True

from keras.optimizers import SGD
# training
model.compile(optimizer=SGD(lr=0.001, momentum=0.9),
              loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(training_images, training_onehot_labels,
          batch_size=32, epochs=5, shuffle=True, validation_split=0.3)

from vis.visualization import visualize_cam
def compare_original_heatmap(i):
    if (int(training_onehot_labels[i][0])) == 1:
        heat_map = visualize_cam(model, 15, 0, training_images[i])
    else:
        heat_map = visualize_cam(model, 15, 1, training_images[i])
    plt.subplot(1,2,1)
```

```
plt.imshow(training_images[i])
plt.subplot(1,2,2)
plt.imshow(heat_map)
plt.show()
```

Results

Technique	No. of Epochs	Training Accuracy / Loss	Validation Accuracy / Loss
AlexNet	15	0.9712 / 0.0746	0.8489 / 0.4327
VGG16	15	0.9891 / 0.0331	0.8751 / 0.6345
Grad-CAM over VGG16	5	0.9883 / 0.0364	0.9764 / 0.0654

References

Dataset

1. https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data

Technical Paper

2. https://arxiv.org/pdf/1610.02391.pdf
3. https://arxiv.org/pdf/1506.06579.pdf
4. https://arxiv.org/pdf/1312.6034.pdf
5. https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Online Courses

6. http://course.fast.ai/
7. https://www.udemy.com/python-for-data-science-and-machine-learning-bootcamp/learn/v4/content
8. http://euler.stat.yale.edu/~tba3/stat665/lectures/lec18/notebook18.html
9. http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture12.pdf
10. https://youtu.be/6wcs6szJWMY
11. https://www.youtube.com/watch?v=yX8KuPZCAMo
12. https://youtu.be/VpUKOLtqBSA
13. https://youtu.be/epS9UVRuoOE

Code Implementation References and Examples

14. https://ramprs.github.io/2017/01/21/Grad-CAM-Making-Off-the-Shelf-Deep-Models-Transparent-through-Visual-Explanations.html
15. http://gradcam.cloudcv.org/
16. https://github.com/Hvass-Labs/TensorFlow-Tutorials
17. https://raghakot.github.io/keras-vis/
18. https://github.com/ramprs/grad-cam/
19. https://github.com/jacobgil/keras-grad-cam
20. https://keras.io/applications/
21. http://agnesmustar.com/
22. https://marubon-ds.blogspot.com/2018/03/object-detection-by-cam-with-keras.html
23. https://stackoverflow.com/
24. https://hackernoon.com/learning-keras-by-implementing-vgg16-from-scratch-d036733f2d5
25. http://cv-tricks.com/tensorflow-tutorial/understanding-alexnet-resnet-squeezenetand-running-on-tensorflow/
26. https://pythonprogramming.net/convolutional-neural-network-kats-vs-dogs-machine-learning-tutorial/
27. https://github.com/duggalrahul/AlexNet-Experiments-Keras
28. https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
29. https://sushscience.wordpress.com/2016/12/04/understanding-alexnet/
30. https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html
31. VGG16 layer size illustration https://www.abtosoftware.com/blog/kitchen-furniture-appliances-recognition-in-photos