Exploring Visual Explanations for CNNs

Jahyun Shin
University of Toronto
lucrece.shin@mail.utoronto.ca

Frank-Edward Nemeth
University of Toronto
frankedward.nemeth@mail.utoronto.ca

Abstract

In this work, we reproduce two saliency map visualization tools, SmoothGrad and GradCAM, and test them using sanity checks of randomizing data labels and trained weights. We demonstrate that both visualization tools pass the sanity checks and prove to be adequate for tasks that are sensitive to both the model and the data. For our experiments, we compare the quality of saliency maps using two different model architectures, AlexNet and VGG11. Finally, we show how the two visualization tools can help improve model generalization by detecting bias.

1 Introduction

Convolutional Neural Networks (CNNs) are prevalent in tasks in computer vision. Researchers have been investigating the workings of such architectures, in order to understand why a neural network predicts what it predicts. Tools to visualize how neurons interpret images highlight the regions of trained images that are considered most important for a network. Two such tools are GradCAM and SmoothGrad, which both create saliency maps to show regions of interest that a neural network is focusing on in an input image when predicting a particular class[1][2]. In this project, we implement these two visualization tools, as well as compare the resulting saliency maps using AlexNet [3] and VGG11 [4]. We also explore two sanity check methods: cascading weight randomizations and label permutations [5]. As an extension, we also investigate how each visualization tool can be used to detect bias, and find explanations for wrong predictions for improving model generalization.

2 Related Works

Numerous works have been looking at interpreting CNN models in order to explain why it predicts what it predicts [1]. This interpretability is very important in both debugging model performance and obtaining intuitions about the network that are translatable to humans. Most research in this area presents a concept called a "saliency map", which is an image of the same dimension with an input image that highlights the important area(s) of the original image that a trained model "looked at" while making a prediction. A saliency map highlights important pixels of the image whose change in intensities will lead to most impact on the prediction score of a target class[5]. One well-known example of such class-discriminative saliency map algorithm is Gradient-weighted Class Activation Mapping (GradCAM)[1], which computes the gradients of a target class flowing into the feature maps of the final convolutional layer of a given network. Saliency maps produced by GradCAM show versatile uses, such as debugging wrong predictions or detecting bias in the dataset. Another algorithm is SmoothGrad [2], which adds a Gaussian noise to the input image and averages over multiple saliency maps. Such methods facilitate in producing visually sharper saliency maps. For evaluating such visualization tools in terms of which aspects in machine learning pipeline (e.g. data annotation, model training, etc.) each one can explain, several sanity check procedures are presented in [5]. Methods such as layer-wise weight randomization and input label permutation are performed to see if such perturbations cause a detrimental effect to resulting saliency maps.

CSC2516 Neural Networks Course Project (2021), Toronto, Canada.

3 Methods

3.1 GradCAM[1]

Gradient-weighted Class Activation Mapping (Grad-CAM) is a class-discriminative localization mapping technique [1]. For a given input image, gradient of the prediction score for class c before $softmax(y_c)$ is computed with respect to the feature map activations A^k of the last convolutional layer, i.e.$\partial y_c(x)/\partial A^k$. The gradients are global average pooled over the width and height dimensions (indexed by i and j respectively) of the same convolutional layer to obtain the neuron importance weights α_k^c:

$$\alpha_k^c = \frac{1}{Z}\sum_i\sum_j \frac{\partial y_c(x)}{\partial A^k}$$

Finally, a weighted combination of forward activation maps is computed:

$$L^c_{Grad-CAM} = ReLU(\sum_k \alpha_k^c A^k)$$

This produces a coarse heatmap of the same dimension as the last convolutional feature map (14 by 14 in case of AlexNet [3] and VGG11 [4]), which can be interpolated to have the same dimensions as the input image. ReLU helps to only highlight pixels whose increased intensities also increase the predicted score for class of interest.1 Without ReLU, localization maps may also highlight other categories in the image. Figure 1 shows two GradCAM maps that explain two different classes of the same image.

3.2 SmoothGrad [2]

SmoothGrad is another technique of visualizing the gradients of a neural network. For an input image x and its associated class c, let $S_c(x)$ be the predicted logit for class c. Then, a saliency map $M_c(x)$ can be obtained by differentiating $S_c(x)$ with respect to input x:

$$M_c(x) = \frac{\partial S_c(x)}{\partial x}$$

To improve this process, smoothing with additive random noise was introduced along with averaging over n samples:

$$\hat{M}_c(x) = \frac{1}{n}\sum_1^n M_c(x + N(0, \sigma^2))$$

where n corresponds to the number of samples and $N(0, \sigma^2)$ represents a zero-mean normal distribution of noise with standard deviation σ. $\hat{M}_c(x)$ represents the final SmoothGrad saliency map. Figure 2 shows the implementation of SmoothGrad for a given input image through AlexNet, and how the sensitivity map changes with n and σ. Using $n = 1$ and $\sigma = 0$ creates the sensitivity map without SmoothGrad. Generally, σ of 10-15% works best, and increasing n sharpens the map[2].

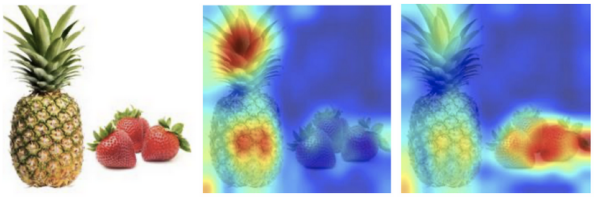

Figure 1: GradCAM saliency maps of two different classes

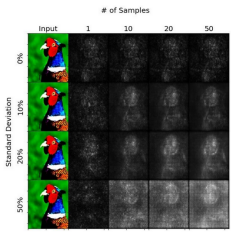

Figure 2: SmoothGrad saliency maps with varying n and sigma

4 Experiments and Discussion

Sanity checks check the sensitivity of the saliency maps to the model and the data [5]. Two different model architectures (AlexNet and VGG11) are used in comparison. We use Spearman Rank Correlation (SRC) as a quantitative metric [5]. Larger plots and test specifics are in Appendices A-D.

4.1 Sanity Check 1: Label Randomization

For label randomization sanity check, each network is trained with data with randomized labels in order to evaluate each visualization tool's relationship between input data and labels. If a saliency map of a model trained with randomized labels still preserves input features, it may indicate that the visualization may be overly dependent on the input image. For this sanity check, each model is overtrained such that it memorizes the training dataset and performs no better than random guessing during testing stage. MNIST dataset with only two hundred training samples was used to ensure that the networks would memorize the randomized data. Each image was upsampled from 28 by 28 to to 64 by 64 pixels to be compatible with AlexNet and VGG11 architectures. Both models were trained to 95% train accuracy [5]. Figure 3 displays the model outputs for a test image.

Results. SRC values were computed between the saliency maps of the true-label model and the randomized-label model. Figure 4 shows the average SRC values for the 100 test images, with error bars representing 95th-percentile ranges. SRC values of 0 are expected, indicating no correlation between the random maps and true maps. The resulting values for SmoothGrad were found to be quite high at around 0.8, while those for GradCAM averaged close to 0. These favourable results for GradCAM can be due to the fact that while gradients for SmoothGrad are computed with respect to the input image, those for GradCAM are computed with regards to the last convolutional layer that is further away from the input image. Consequently, GradCAM maps does not carry as much information from the input image as SmoothGrad. VGG11 shows more severe deterioration of maps compared to AlexNet in Figure 3, as well as having lower SRC scores for both GradCAM and SmoothGrad in Figure 4. This shows that the visualization tools may be more reliable in explaining certain architectures than others. This can be due to VGG11 having more convolutional layers (8) than AlexNet (5), which can escalate the discontinuity between the input and output gradients.

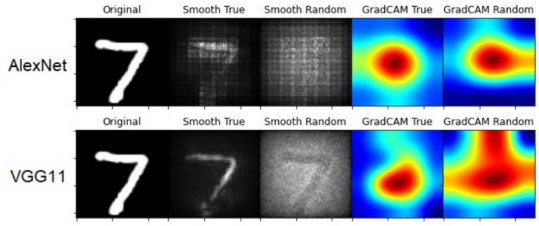

Figure 3: Model outputs after training with true and randomized labels (class = 7).

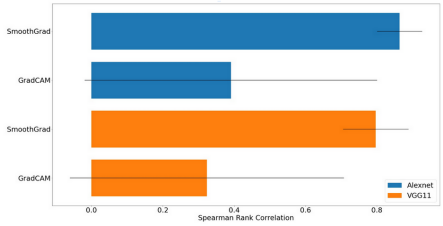

Figure 4: Average SRC values for each model and visualization tool

4.2 Sanity Check 2: Cascading Weight Randomization

In cascading weight randomization sanity check, the learned weights of the model are randomized with w_{random} as $N(0, 0.01^2)$ successively layer-by-layer, from top-most to bottom-most layer.

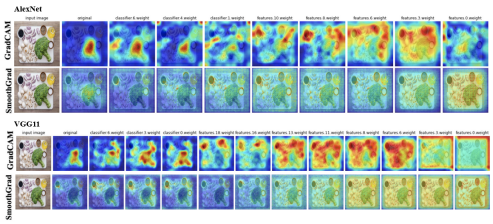

Figure 5: Cascading randomization on AlexNet and VGG11.

Results. Figure 5 shows saliency maps for cascading randomization on AlexNet and VGG11 pretrained on ImageNet for target class "broccoli". In the figure, progression from left to right indicates the randomization of weights up to that layer inclusive (with layer name shown on top). The last column corresponds to a network with completely randomized weights. With visual inspection, GradCAM maps for both models deteriorate slightly when the classifiers' weights are randomized, then deteriorate greatly as further convolutional layers are randomized. SmoothGrad maps for both

models show similar deterioration patterns that become severe from the convolutional layers. This shows that both methods are sensitive to model weights, and are viable in diagnosing the trained model. For SRC calculations, 20 test images from ImageNet dataset were used. It is expected that the saliency maps approach SRC score of 0 as each layer's weights are randomized successively, which is depicted in Figure 6. As layer randomization advances, SmoothGrad reaches a closer value to zero with VGG11 than with AlexNet. As mentioned before, this is likely due to VGG11 having more convolutional layers. Also, GradCAM deteriorate more (thus lower SRC score) than SmoothGrad, especially with AlexNet. Again, this may be because GradCAM's gradients are taken with respect to the last convolutional layer as opposed to the input image for the SmoothGrad.

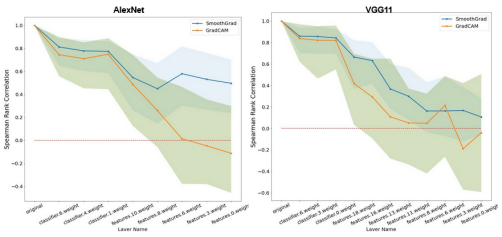

Figure 6: Average SRC values for cascading weight randomizations

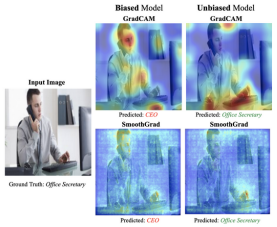

Figure 7: Saliency maps for biased and unbiased models

4.3 Detecting Bias in Dataset

When diagnosing poor generalization performance of a CNN model, if the saliency maps show that the model is paying attention to parts of the image that are not distinctive of the particular class, we can check if the original dataset is misleading. Existence of bias is a common source of such faults in datasets, which could lead to the model learning stereotypes of gender, race, or age.

Experiment. To test this, we performed a "office secretary" vs. "CEO" binary classification task using AlexNet, fined-tuned from ImageNet pre-trained weights. For training and validation, 150 to 200 images per class were scraped from Google with search keywords "office secretary" and "CEO". Distinctive features of the "office secretary" class include tools such as a telephone, pen, and keyboard. For test set, images of male office secretaries were used.

Results. Although the trained model achieved a good validation accuracy, it only achieved 56% accuracy for the test set of male office secretaries. Saliency maps using this model are shown the second column of Figure 7, which show that the model has learned to pay attention to the face and hair of the person instead of tools like the phone and the keyboard, and misclassified the image as CEO. By inspecting the original dataset, 96% of the collected Google images of office secretaries were female, while 82% of images of CEOs were male. When the model was re-trained with a more balanced dataset in gender, test accuracy increased to 81%. As shown in Figure 7's third column, both saliency maps indicate the model now pays attention to the keyboard and typing gesture, while correctly classifying the image. This demonstrates the effective use of SmoothGrad and GradCAM maps to detect bias in the dataset, which has eliminated ethical biases and improved generalization.

5 Conclusion

In this project, we implemented two CNN-based neural network visualization tools, GradCAM and SmoothGrad, to test their ability to localize important regions of an input image for predicting a class. We observed that the saliency maps using both tools showed deterioration during sanity checks of randomizing trained weights layer-by-layer and retraining the model with random labels. This proved that both tools are sensitive to the model weights and labels and can be useful in diagnosing them. Further, varied results when using AlexNet versus VGG11 highlighted that the network-specific traits can affect the quality of the maps. Future work can build upon finding more applications wherein the two tools can be found useful in diagnosing and solving a problem about the model.

6 References

[1] Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Gradcam: Visual explanations from deep networks via gradient-based localization. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 618–626, 2017.

[2] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS*, 2012.

[4] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR*, 2015.

[5] Adebayo, J.; Gilmer, J.; Muelly, M.; Goodfellow, I.; Hardt, M.; and Kim, B. . Sanity checks for saliency maps. In *NeurIPS*, 9525–9536, 2018.

7 Coding References

Areas indicated in files

[1] Inkawhich, N. Finetuning Torchvision Models — PyTorch Tutorials 1.2.0 documentation. *Pytorch.org.*, 2021. Retrieved 19 April 2021, from https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html.

[2] Eli Schwartz, imagenet-sample-images, *GitHub repository*, 2019. Retrieved 19 April 2021, from https://github.com/EliSchwartz/imagenet-sample-images

Appendix A: Larger Images

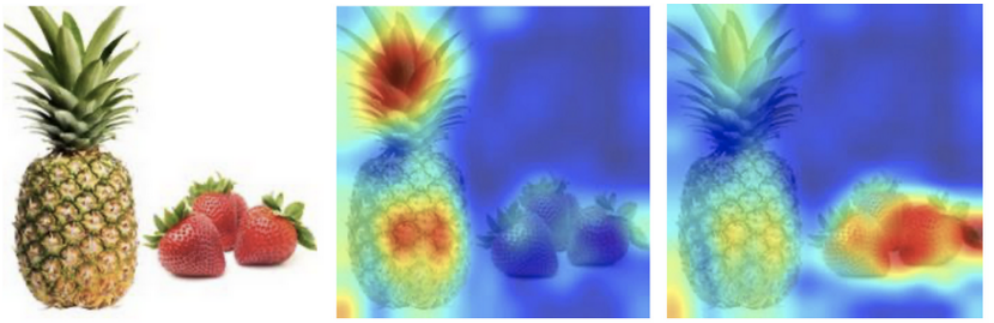

Figure A1: GradCAM implementation

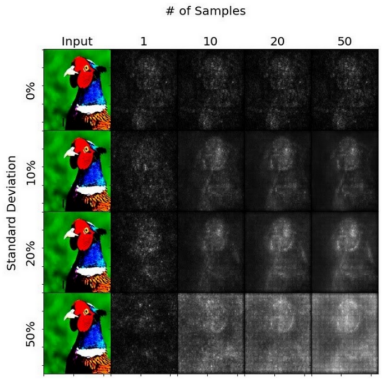

Figure A2: SmoothGrad Implementation

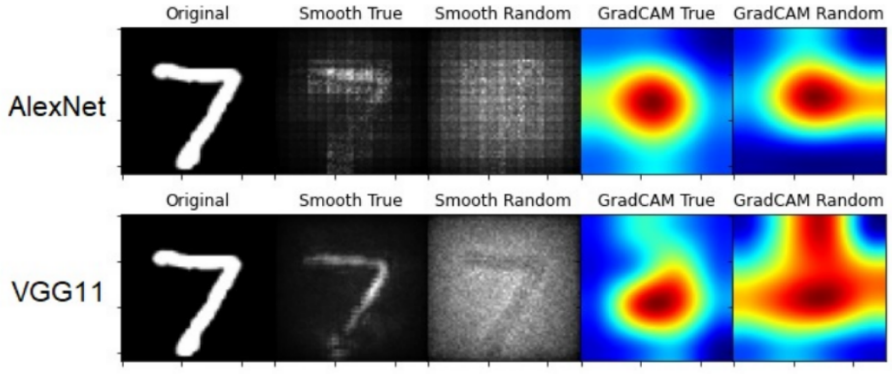

Figure A3: Model Outputs after training with True and Randomized Labels (class = 7)

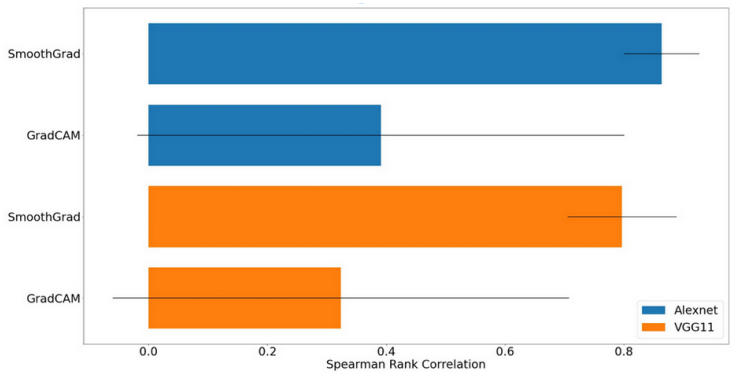

Figure A4: Average SRC Values for each Model and Visualization Tool

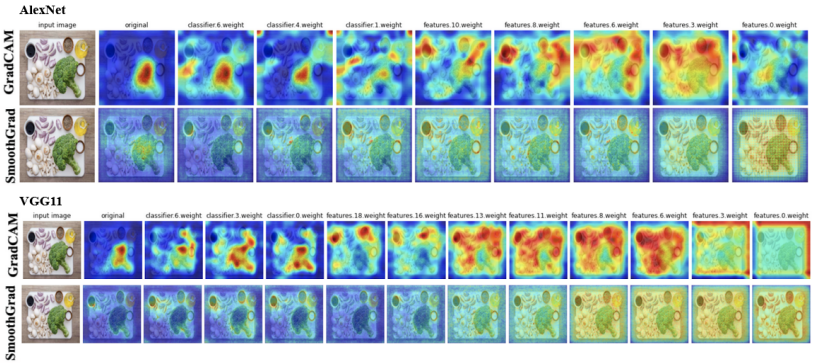

Figure A5: Cascading Randomization on AlexNet and VGG11

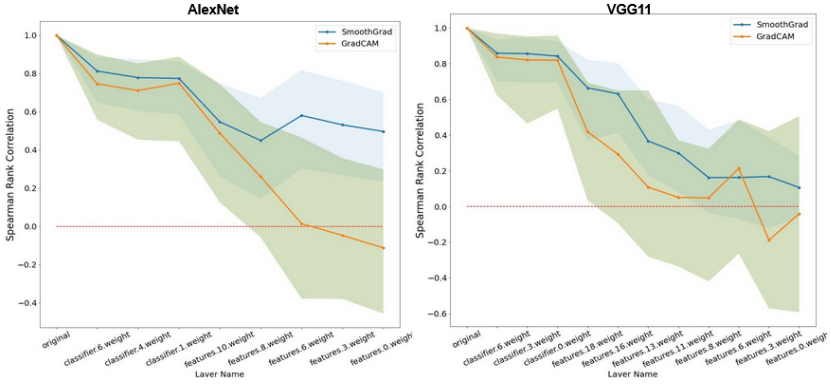

Figure A6: Average SRC values for Cascading Weight Randomizations

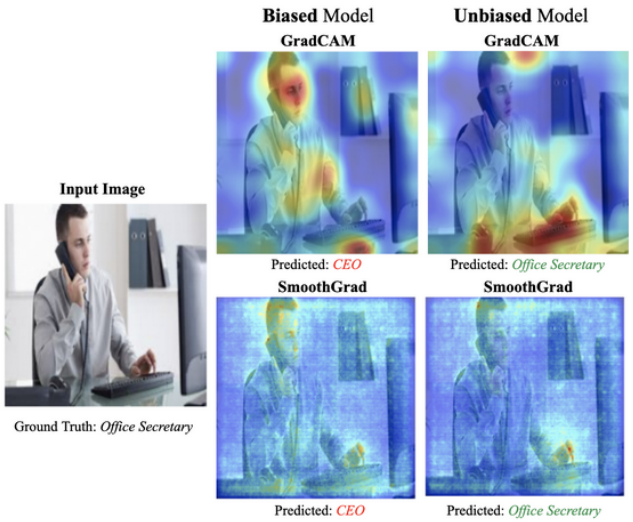

Figure A7: Bias Detection in Neural Network

Appendix B: Label Randomization Test Details

B.1 Random Labels Training Parameters

```
num_classes = 10 # classes in mnist
batch_size_train = 16
batch_size_test = 16
num_epochs = 200

learning_rate = 0.001
momentum = 0.9
log_interval = 10
random_seed = 1
torch.manual_seed(random_seed)
device = torch.device("cuda")
```

Note values were adjusted to ensure models would train to 95% accuracy in a reasonable time. Epochs were set to 200, but training would stop early (once accuracy achieved)

B.2 SmoothGrad Parameters

$\sigma = 20$, n= 20
These values were chosen to balance computation costs with descent image quality

B.3 AlexNet Model Description

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 128, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(128, 384, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(384, 480, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(480, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=8192, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=8192, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=10, bias=True)
  )
)
```

Note, layers had inputs scaled upwards to increase learning parameters. AlexNet in the random labels test was not converging on test accuracy (was not able to memoize the data)

B.4 VGG11 Model Description

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (11): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (12): ReLU(inplace=True)
    (13): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (14): ReLU(inplace=True)
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (16): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): ReLU(inplace=True)
    (18): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (19): ReLU(inplace=True)
    (20): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=10, bias=True)
  )
)
```

B.5 MNIST Rescaling and Normalization

```
preprocessing = transforms.Compose([
    transforms.Resize(size = (64,64)),
    transforms.ToTensor(),
    transforms.Lambda(lambda x: x.repeat(3,1,1)),
    torchvision.transforms.Normalize((0.1307,), (0.3081,))
    ])
```

Images upscaled to work with the prebuilt architectures

B.6 AlexNet Train Results

B.6.1 True Labels

```
Epoch 25/199
----------
train Loss: 0.1276 Acc: 0.9620
Training complete in 4m 28s
Best val Acc: 0.924100
```

B.6.2 Test Labels

```
Epoch 203/249
----------
train Loss: 0.1536 Acc: 0.9500
Training complete in 36m 8s
Best val Acc: 0.140600
```

B.7 VGG11 Train Results

B.7.1 True Labels

```
Epoch 9/199
----------
train Loss: 0.1166 Acc: 0.9630
Training complete in 2m 1s
Best val Acc: 0.940100
```

B.7.2 Test Labels

```
Epoch 68/199
----------
train Loss: 0.1588 Acc: 0.9550
Training complete in 14m 57s
Best val Acc: 0.145400
```

Appendix C: Cascading Weight Randomization Test Details

C.1 SmoothGrad Parameters

$\sigma = 20$, n= 20
These values were chosen to balance computation costs with descent image quality

C.2 ImageNet Rescaling and Normalization

```
preprocessing = transforms.Compose([
    transforms.Scale(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    torchvision.transforms.transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])
```

Appendix D: Spearman Rank correlation Formula

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$

d_i represents the distance between the ranks of the observations

n represents the number of observations

Additional Explanation: Consider two saliency maps A and B. The rank for each map would be computed, A_rank and B_rank, by assigning each pixel an integer value based on its relative size with the other pixels (ie, order all values from min to max, and reassign their value as their placement to obtain the ranks). Then subtract the map values as the distance. In this case n is the number of pixels is a saliency map.
Averages would compute an SRC value for each pair of saliency maps (random and true map pairs) and then take the mean of all SRC values.