

Project Report: EE 769

Classifying Droplet Distortions From Images

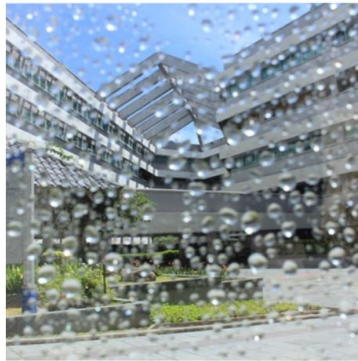
Sahar Nasser (194072001), Varsha Satish (193079005), Ruchita Korgaonkar (194076011)

Introduction:

During outdoor photography/videography, especially during monsoons, water droplets may stick to the camera thus distorting the captured images and/or videos. These videos and photographs are undesirable and take a lot of time for people to find and remove from their collection.

The objective is to detect whether the droplets in an image are distortion on the lens or are a part of the scene captured. Our dataset contains two classes, in which the image which has distortion (due to droplets on the lens) is considered as a positive example, while the image which has no droplets on the lens or droplets which are part of the scene is considered a negative example.

Positive examples



Negative examples



To tackle this problem, we use a combination of techniques namely Transfer Learning, Adversarial Learning, Label Smoothing, and Captum Insights. We try different backbones such as ResNet, DenseNet, Inception Net, and VGG Net.

The performance of our model on the testing dataset shows the ability of the model to handle various environmental variations, for instance, testing the model on indoor images which have drastically different illuminations from the ones of the outdoor training images.

In this report, we describe the datasets we prepared, shed light on the different techniques we incorporated, and conclude with the results and the future work.

Dataset:

We have downloaded datasets from various sources. The details of each dataset are provided on this google sheet:

https://docs.google.com/spreadsheets/d/1xoGfwbeM_fpKxx3zBTs1I8tYgGSzkEo1eLsD TX3ODKM/edit?ts=5e6f366b#gid=0

Dataset	Training		Validation		Size (Format= jpeg)
	Number of Distorted images	Number of non-Distorted images	Number of Distorted images	Number of non-Distorted images	
BinaryClassDataset_Resized	861	2863	282	45	350*350*3
NewDataset7_4	1244	1293	282	282	350*350*3
NewDataset12_4	1244	2430	282	226	350*350*3
FinalDataset1	270	1376	306	230	Various sizes

To access any of the datasets which we prepared please use the following link:
<https://mega.nz/C!wWBTSQxa>

Models:

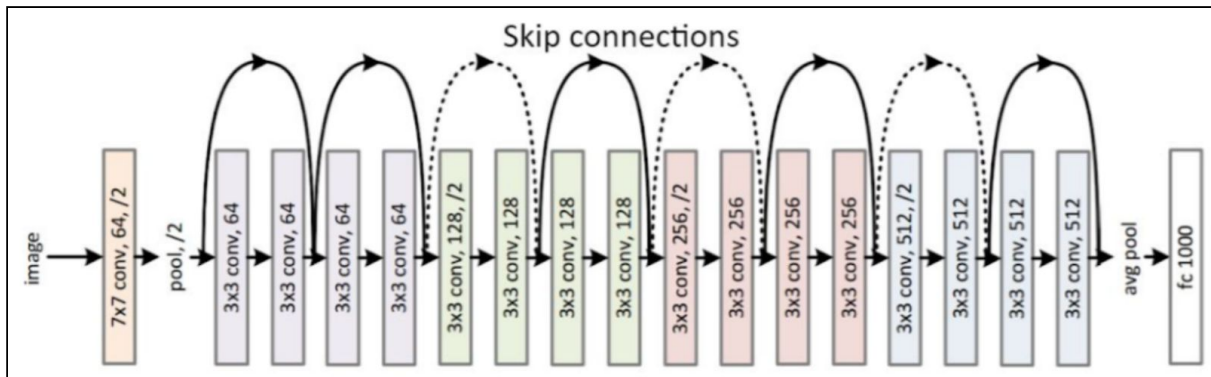
- 1) **VGG - 11** - The Convolutional neural networks developed for winning the ImageNet Challenge 2014 in localization and classification tasks are known as VGG nets. It takes input image of size $224 \times 224 \times 3$ (RGB image). It is built using - Convolution layers (3×3 size), Max pooling layers (2×2 size), Fully connected layers at end [4].



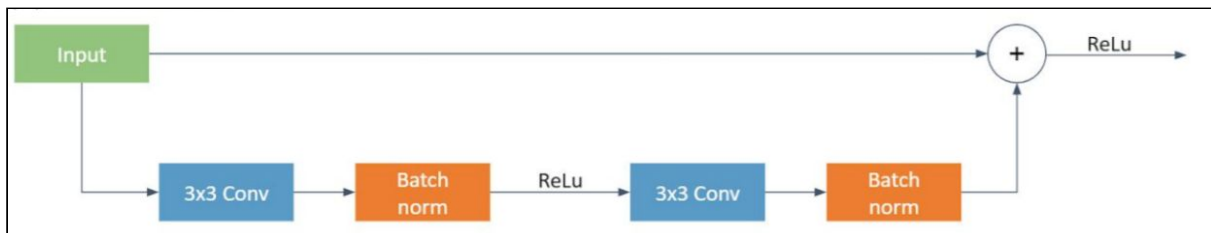
VGG -11 architecture

- 2) **Resnet - 18** - The main idea of ResNet is introducing “identity shortcut connection” that skips one or more layers. It is hypothesized that letting the stacked layers fit a residual mapping is easier than letting them directly fit the desired underlying mapping, which is carried out by residual modules in the network.

The ResNet comprises multiple modules that each perform a convolution, batch normalization, and nonlinear operation (rectified linear unit). The input (signal plus noise) is processed through 18 ResNet modules. These are connected in series, along with certain connections that simply pass along the module input (skip connections). The final stage is a fully connected layer that provides a classification decision. Dashed skip connections imply a resizing [5].



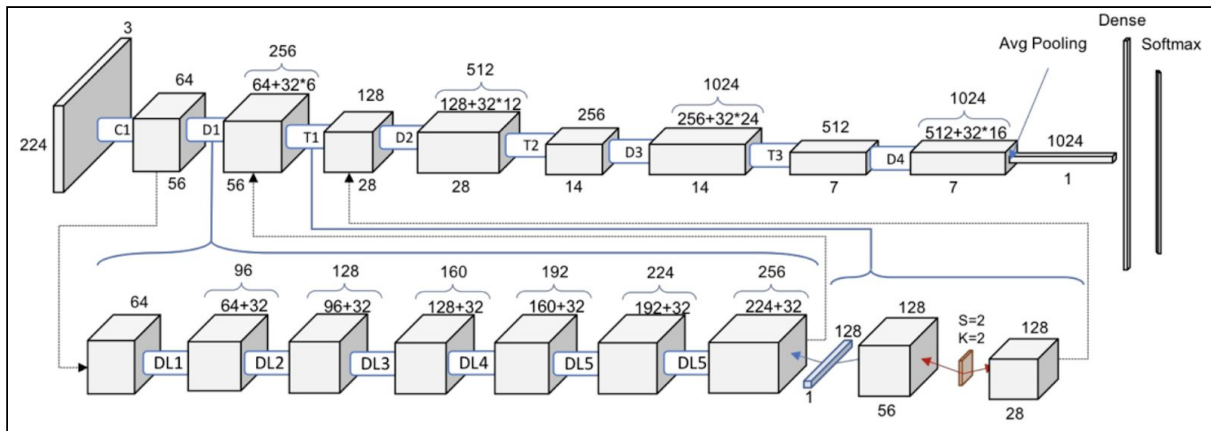
Resnet-18 architecture



Resnet module

(source - <https://arxiv.org/ftp/arxiv/papers/1911/1911.05055.pdf>)

3) **Densenet** - Densely Connected Convolutional Networks, DenseNets, help in increasing the depth of deep convolutional networks. They are similar to Resnets with a few modifications. DenseNets, unlike resnets, do not sum the output feature maps of the layer with the incoming feature maps but concatenate them. Like Resnets, DenseNets are divided into DenseBlocks, where the dimensions of the feature maps remain constant within a block, but the number of filters changes between them. These layers between them are called Transition Layers and take care of the downsampling applying a batch normalization, a 1x1 convolution and a 2x2 pooling layers [7].

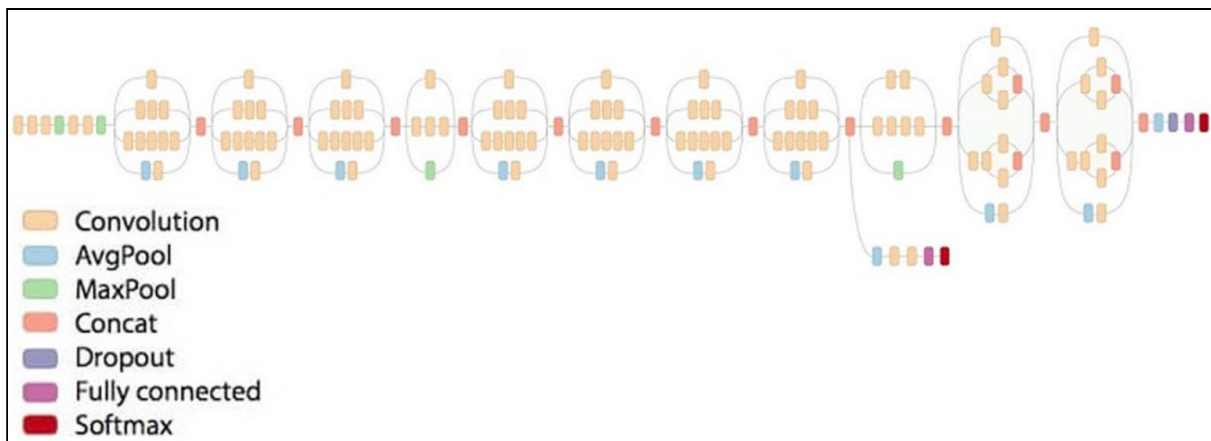


Dense Block and Transition Block. DLx: Dense Layer x

Densenet architecture

(source-<https://towardsdatascience.com/understanding-and-visualizing-densenets>)

- 4) **Inception V3** - Inception V3 by Google is the 3rd version in a series of Deep Learning Convolutional Architectures. Inception V3 was trained using a dataset of 1,000 classes from the original ImageNet dataset which was trained with over 1million training images. Inception V3 was trained for the ImageNet Large Visual Recognition Challenge where it was a first runner up [6].



Inception V3 architecture

(source - www.software.intel.com)

Loss Function:

Our problem is a binary classification problem. Cross-entropy is the default loss function to use for binary classification problems, where the target values are in the set $\{0, 1\}$.

Cross-entropy is a measure of the difference between two probability distributions for a given random variable or set of events. Mathematically, it is the preferred loss function under the inference framework of maximum likelihood. Cross-entropy will calculate a score that summarizes the average difference between the actual and predicted probability distributions for predicting class 1. The score is minimized and a perfect cross-entropy value is 0.

The cross-entropy loss function is the formula below

$$L_{cross-entropy}(\hat{y}, y) = - \sum y_i * \log(\hat{y}_i)$$

where, $y = y_1, y_2, \dots, y_n$ be a vector representing the distribution over the labels $1, \dots, n$, and let $\hat{y} = \hat{y}_1, \hat{y}_2, \dots, \hat{y}_n$ be the classifiers output \hat{y} .

Loss with Label Smoothing:

Label smoothing is used to make the model to train around mislabelled data and be robust. This prevents the model from being overconfident and poor generalisation.

Label smoothing changes the target vector by a small amount ϵ . Thus, instead of making the model to predict 1 for the right class, it is made to predict $(1-\epsilon)$ for the correct class and ϵ for all the others. So, the cross-entropy loss function with label smoothing is transformed into the formula below.

$$L_{cross-entropy \text{ with label smoothing}} = (1 - \epsilon) * ce(i) + \epsilon \sum \frac{ce(j)}{N}$$

where, $ce(x)$ denotes the standard cross-entropy loss of x (e.g. $-\log(p(x))$), i is the correct class and N is the number of classes ($N = 2$ in our problem).

Transfer Learning:

- We started with a pre-trained model on ImageNet, and we only updated the weights of the final layer.
- The steps of transfer learning in short [9]:
 1. Initialize the pre-trained model.
 2. Reshape the final layer(s) to have the same number of outputs as the number of classes in your custom dataset.
 3. Define which parameters you want to update during the training phase.
 4. Run the training step.
- We tried various backbones namely ResNet, VGG, DenseNet, and Inception3.
- As the final layer of a CNN model, which is often a fully connected (FC) layer, has the same number of nodes as the number of output classes in the dataset (In our case 1000 classes as all the models have been trained on ImageNet).
- We reshaped the last layer to have the same number of inputs as before, and to have the same number of outputs as the number of classes in our custom dataset.
- Models:
 1. ResNet: We used Resnet18 as a backbone. In this architecture the last layer is linear with input features = 512, and outputs = 1000. Thus we reinitialized this layer to become a linear layer with input features = 512 and two classes as outputs.
 2. VGG: We used VGG-11 with batch normalization as a backbone. The output comes from the 6th layer which is a linear with input features = 4096 , and outputs = 1000. Thus we reinitialized this layer to become a linear layer with input features = 4096 and two classes as outputs.
 3. DenseNet: We used DenseNet-121. In this architecture the last layer is linear with input features = 1024, and outputs = 1000. Thus we reinitialized this layer to become a linear layer with input features = 1024 and two classes as outputs.

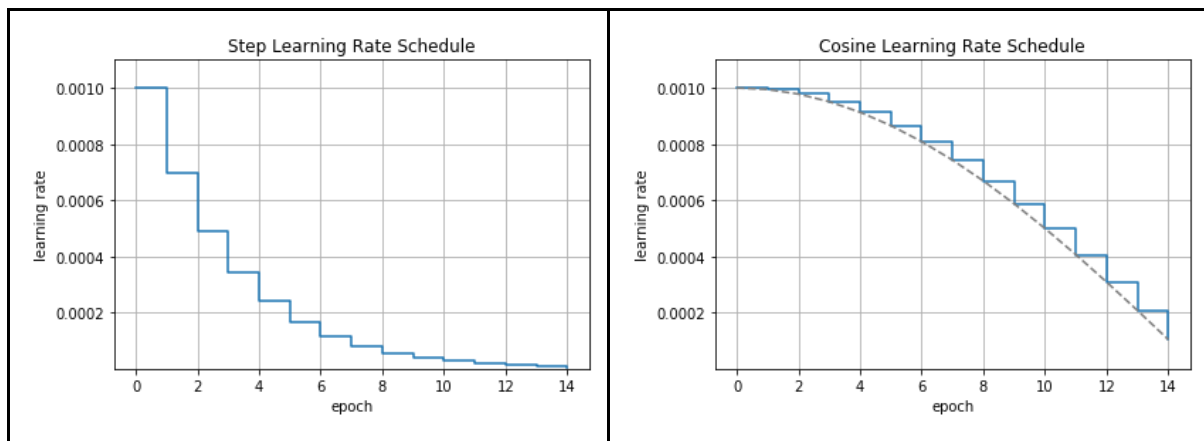
4. Inception V3:

This architecture has two output layers while training. The second output is known as the auxiliary output. The primary output is a linear layer at the end of the network.

The auxiliary output is a linear layer with 768 inputs and 1000 outputs. While the primary output layer is linear with 2048 inputs and 1000 outputs. We reshaped both layers into linear layers with the same input features as they were before and two classes output.

Learning Rate:

When training a model, it is useful to reduce learning rate as training progresses. This can be done using predefined schedulers. We tried constant learning rate, step learning rate and cosine learning rate for learning which has reduced oscillations in accuracy. Following figure shows variation in the learning rate for step based learning rate and cosine learning rate.[1,10]



Adversarial Learning:

- We implemented adversarial learning to make our classification method more robust to noisy data.
- We used one of the most popular attack methods namely the Fast Gradient Fast Attack (FGSM) to fool the classifier [8].
- The goal of this method is to add the least amount of perturbation to the input data to cause the desired misclassification.
- The attack adjusts the input data to maximize the loss based on the back propagated gradients.
- We trained our model on 50 epochs. We applied an attack at every 10 epochs with different values of noise (epsilon). $\epsilon \in [0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3]$
- We noticed that as epsilon increases, the model accuracy decreases.
- The function of the FGSM attack takes three inputs.
 1. The original clean image x.
 2. Epsilon which is a pixel-wise perturbation amount.

3. The gradient of the loss w.r.t the input image.

$$x_{noisy} = x + \varepsilon \times \text{sign}(\nabla_x J(\theta, x, y))$$

- Figure(1) shows a visualization of some successful adversarial examples during the validation phase with a confidence score of the new class.
- Figure(2) shows how the accuracy decreases as epsilon value increases, as larger value of epsilon means taking a larger step in maximizing the loss.

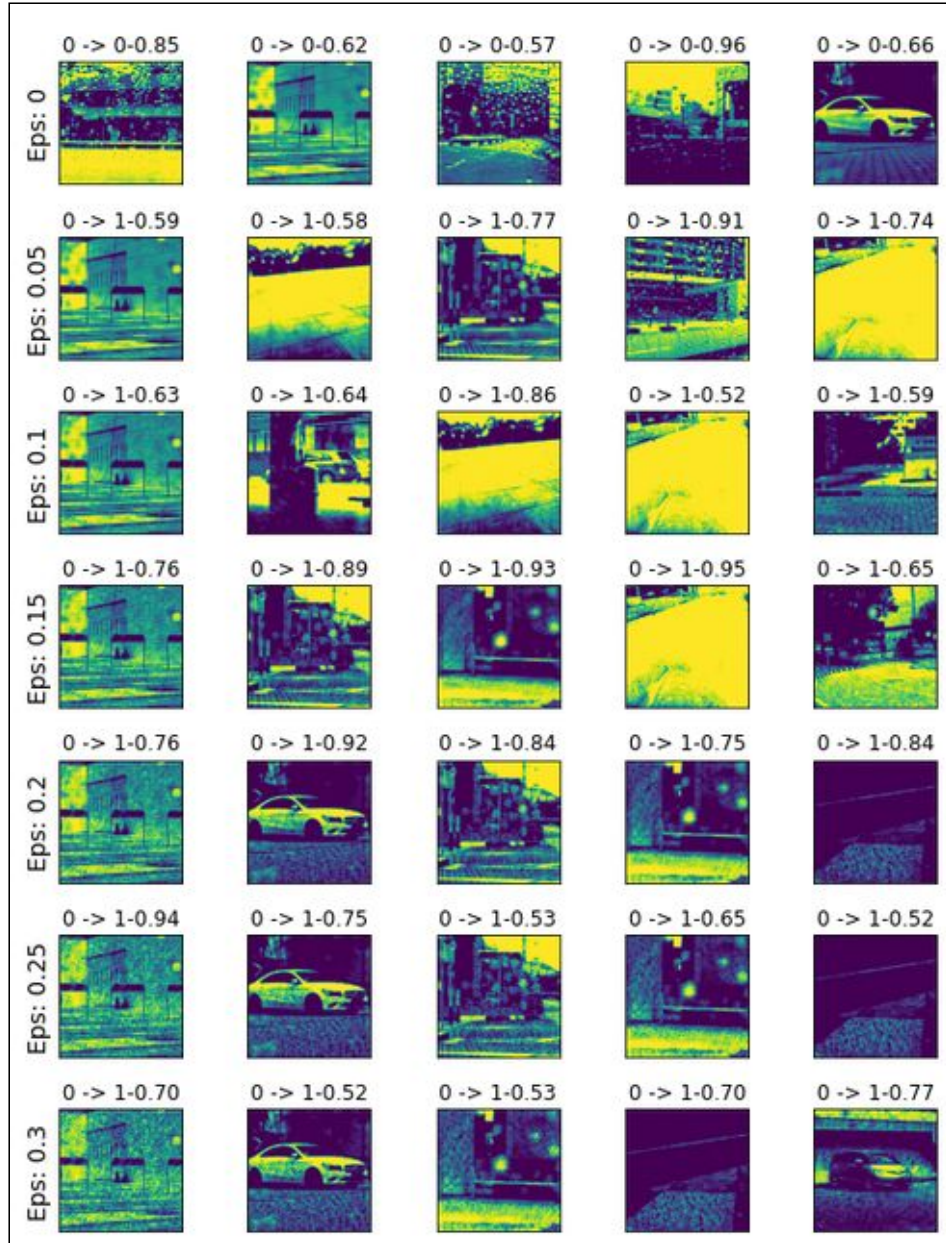


Fig1. A visualization of some adversarial examples during the validation phase. We printed the original class, the new class with a confidence score of the new class on top of every image in this figure.

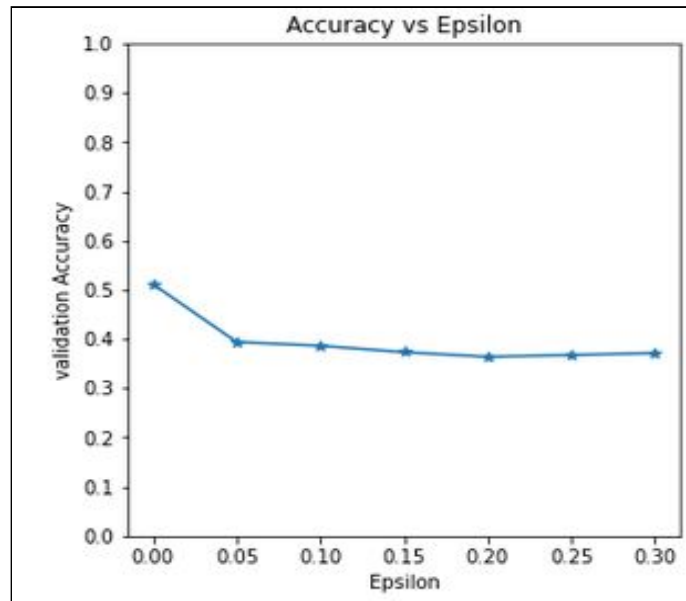






Fig 2. The validation accuracy against epsilon.

Experiments:

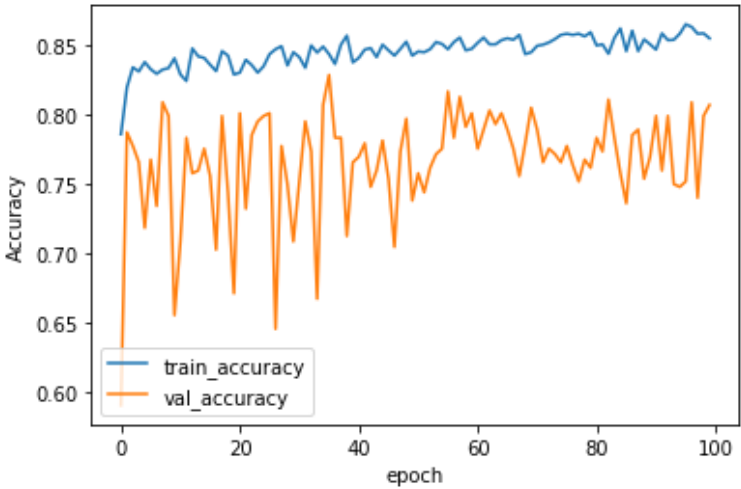
We performed many experiments to explore the effects of various techniques on the model's performance. The details of each experiment are provided in the google sheet, following is the link for google sheet: https://docs.google.com/spreadsheets/d/1pr-kNZQy1fSSIV9EW6EQfBvC0KD-UpkonV_AiA5fpr8/edit?usp=sharing





1)

Model: Resnet-18	Dataset: NewDataset12_4																																																
Epochs = 15, Step Learning Rate, Optimizer = Adam, Weighted Cross Entropy loss, Time required on CPU: 34 minutes 45 seconds	<table><caption>Accuracy Data (Estimated from Graph)</caption><thead><tr><th>epoch</th><th>train_accuracy</th><th>val_accuracy</th></tr></thead><tbody><tr><td>0</td><td>0.78</td><td>0.64</td></tr><tr><td>1</td><td>0.82</td><td>0.52</td></tr><tr><td>2</td><td>0.81</td><td>0.62</td></tr><tr><td>3</td><td>0.84</td><td>0.79</td></tr><tr><td>4</td><td>0.84</td><td>0.70</td></tr><tr><td>5</td><td>0.83</td><td>0.63</td></tr><tr><td>6</td><td>0.84</td><td>0.77</td></tr><tr><td>7</td><td>0.84</td><td>0.74</td></tr><tr><td>8</td><td>0.84</td><td>0.79</td></tr><tr><td>9</td><td>0.84</td><td>0.79</td></tr><tr><td>10</td><td>0.85</td><td>0.82</td></tr><tr><td>11</td><td>0.85</td><td>0.68</td></tr><tr><td>12</td><td>0.84</td><td>0.76</td></tr><tr><td>13</td><td>0.85</td><td>0.74</td></tr><tr><td>14</td><td>0.85</td><td>0.79</td></tr></tbody></table>	epoch	train_accuracy	val_accuracy	0	0.78	0.64	1	0.82	0.52	2	0.81	0.62	3	0.84	0.79	4	0.84	0.70	5	0.83	0.63	6	0.84	0.77	7	0.84	0.74	8	0.84	0.79	9	0.84	0.79	10	0.85	0.82	11	0.85	0.68	12	0.84	0.76	13	0.85	0.74	14	0.85	0.79
epoch	train_accuracy	val_accuracy																																															
0	0.78	0.64																																															
1	0.82	0.52																																															
2	0.81	0.62																																															
3	0.84	0.79																																															
4	0.84	0.70																																															
5	0.83	0.63																																															
6	0.84	0.77																																															
7	0.84	0.74																																															
8	0.84	0.79																																															
9	0.84	0.79																																															
10	0.85	0.82																																															
11	0.85	0.68																																															
12	0.84	0.76																																															
13	0.85	0.74																																															
14	0.85	0.79																																															
Best Validation Accuracy at Epoch = 10 Training Accuracy: 0.8443 Validation Accuracy: 0.8130																																																	

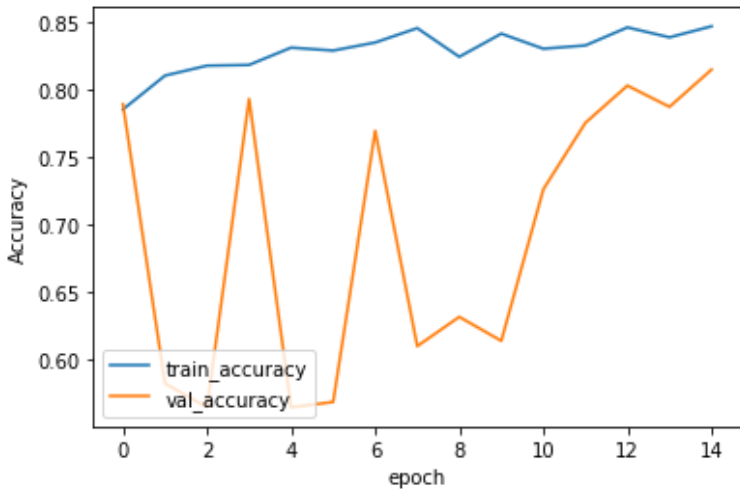
<u>Best Examples</u>	<u>Worst Examples</u>
<u>Predicted Class: Distortion, prob = 0.99</u>	<u>Misclassified: no_Distortion, prob = 0.98</u>
	
<u>Predicted Class: no_Distortion, prob = 0.99</u>	<u>Misclassified: Distortion, prob = 0.97</u>
	





2)

Model: Resnet-18	Dataset: NewDataset12_4
Epochs = 100, Cosine Learning Rate, Optimizer = SGD, Weighted Cross Entropy loss, Time required on CPU: 215 minutes 42 seconds	
Best Validation Accuracy at Epoch = 35 Training Accuracy: 0.8438 Validation Accuracy: 0.8287	

<u>Best Examples</u>	<u>Worst Examples</u>
<u>Predicted Class: Distortion, prob = 0.99</u>	<u>Misclassified: no_Distortion, prob = 0.98</u>
	
<u>Predicted Class: no_Distortion, prob = 0.99</u>	<u>Misclassified: Distortion, prob = 0.97</u>
	





3)

Model: Resnet-18	Dataset: NewDataset12_4																																																
Epochs = 15, Step Learning Rate, Optimizer = SGD, Weighted Cross Entropy loss, Time required on CPU: 33 minutes 20 seconds	 <table><caption>Approximate Accuracy Data from Graph</caption><thead><tr><th>epoch</th><th>train_accuracy</th><th>val_accuracy</th></tr></thead><tbody><tr><td>0</td><td>0.79</td><td>0.79</td></tr><tr><td>1</td><td>0.81</td><td>0.60</td></tr><tr><td>2</td><td>0.82</td><td>0.60</td></tr><tr><td>3</td><td>0.82</td><td>0.79</td></tr><tr><td>4</td><td>0.83</td><td>0.60</td></tr><tr><td>5</td><td>0.83</td><td>0.60</td></tr><tr><td>6</td><td>0.84</td><td>0.77</td></tr><tr><td>7</td><td>0.84</td><td>0.61</td></tr><tr><td>8</td><td>0.83</td><td>0.63</td></tr><tr><td>9</td><td>0.84</td><td>0.61</td></tr><tr><td>10</td><td>0.83</td><td>0.73</td></tr><tr><td>11</td><td>0.84</td><td>0.78</td></tr><tr><td>12</td><td>0.85</td><td>0.80</td></tr><tr><td>13</td><td>0.84</td><td>0.79</td></tr><tr><td>14</td><td>0.85</td><td>0.82</td></tr></tbody></table>	epoch	train_accuracy	val_accuracy	0	0.79	0.79	1	0.81	0.60	2	0.82	0.60	3	0.82	0.79	4	0.83	0.60	5	0.83	0.60	6	0.84	0.77	7	0.84	0.61	8	0.83	0.63	9	0.84	0.61	10	0.83	0.73	11	0.84	0.78	12	0.85	0.80	13	0.84	0.79	14	0.85	0.82
epoch	train_accuracy	val_accuracy																																															
0	0.79	0.79																																															
1	0.81	0.60																																															
2	0.82	0.60																																															
3	0.82	0.79																																															
4	0.83	0.60																																															
5	0.83	0.60																																															
6	0.84	0.77																																															
7	0.84	0.61																																															
8	0.83	0.63																																															
9	0.84	0.61																																															
10	0.83	0.73																																															
11	0.84	0.78																																															
12	0.85	0.80																																															
13	0.84	0.79																																															
14	0.85	0.82																																															
Best Validation Accuracy at Epoch = 14 Training Accuracy: 0.8470 Validation Accuracy: 0.8150																																																	

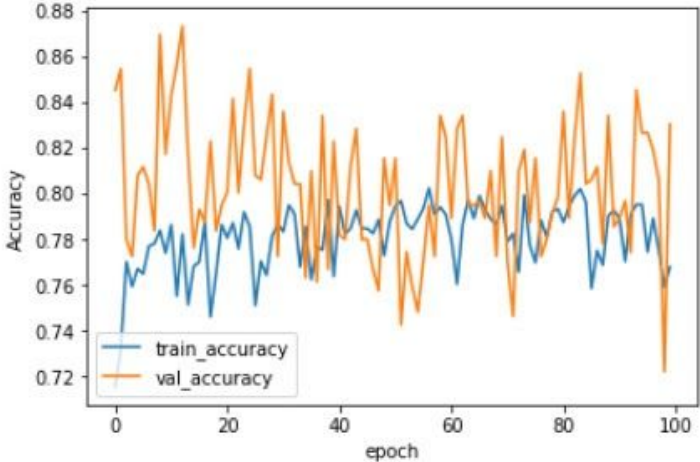
<u>Best Examples</u>	<u>Worst Examples</u>
<u>Predicted Class: Distortion, prob = 0.99</u>	<u>Misclassified: no_Distortion, prob = 0.99</u>
	
<u>Predicted Class: no_Distortion, prob = 0.99</u>	<u>Misclassified: Distortion, prob = 0.99</u>
	





5)

Model: DenseNet	Dataset: FinalDataSet1 (Images with Various Size)
Epochs = 50, Cosine Annealing Learning Rate, Optimizer = Adam, Weighted Cross Entropy loss, Adversarial Learning , Time required on CPU: 266 minutes 55 seconds	
Best Validation Accuracy at Epoch = 16 Training Accuracy: 0.79 Validation Accuracy: 0.69	

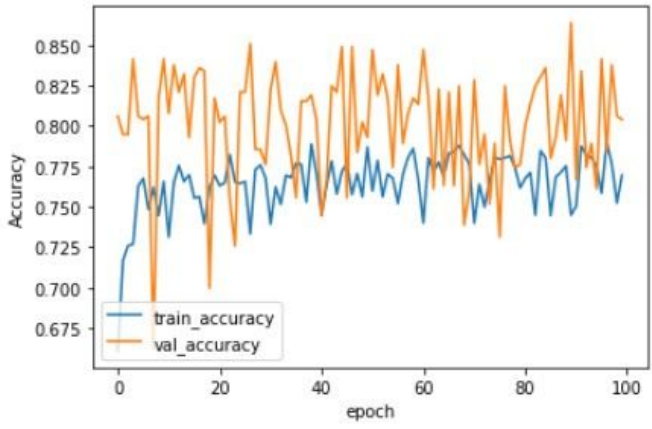
<u>Best Examples</u>	<u>Worst Examples</u>
<u>Predicted Class: Distortion, prob = 0.99</u>	<u>Misclassified: no_Distortion, prob = 0.99</u>
	
<u>Predicted Class: no_Distortion, prob = 0.99</u>	<u>Misclassified: Distortion, prob = 0.88</u>
	


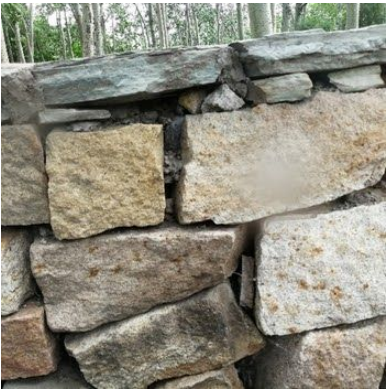
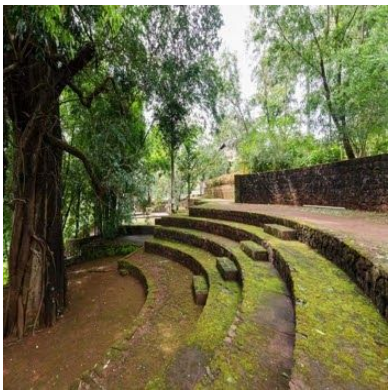
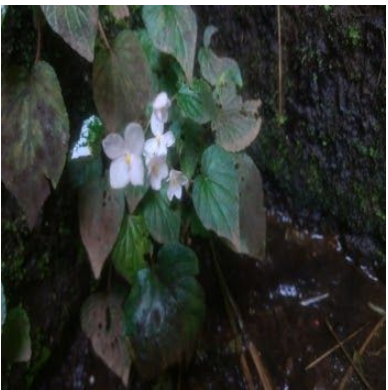
6)

Model: DenseNet	Dataset: FinalDataSet1 (Images with Various Size)
Epochs = 100, Cosine Annealing Learning Rate, Optimizer = Adam, Weighted Cross Entropy loss, Time required on CPU: 163 minutes 47 seconds	
Best Validation Accuracy at Epoch = 12 Training Accuracy: 0.78 Validation Accuracy: 0.873	

<u>Best Examples</u>	<u>Worst Examples</u>
<u>Predicted Class: Distortion, prob = 0.99</u>	<u>Misclassified: no_Distortion, prob = 0.99</u>
	
<u>Predicted Class: no_Distortion, prob = 0.99</u>	<u>Misclassified: Distortion, prob = 0.77</u>
	

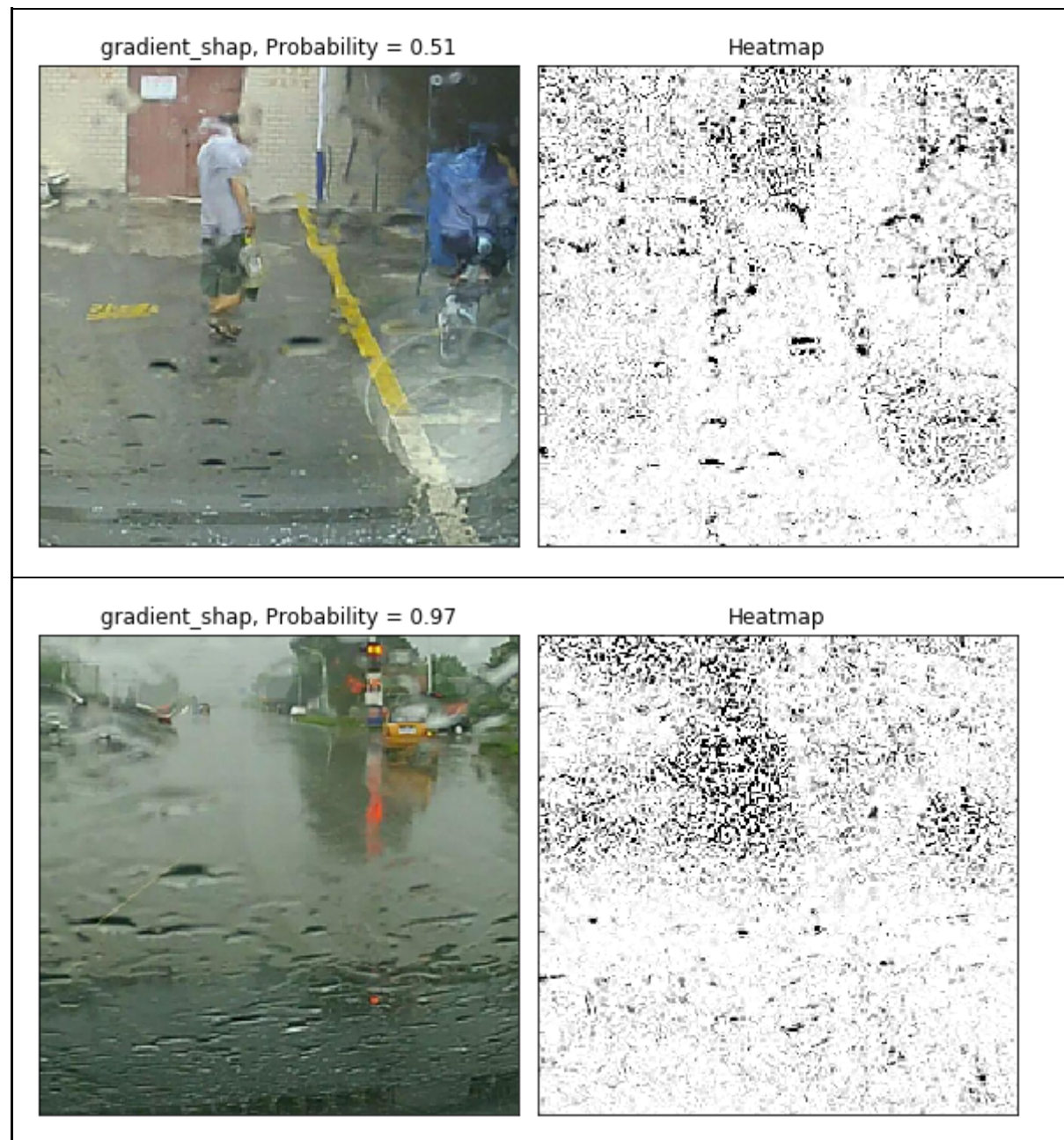
7)

Model: Resnet-18	Dataset: FinalDataSet1 (Images with Various Size)
Epochs = 100, Cosine Learning Rate, Optimizer = Adam, Weighted Cross Entropy loss, Time required on GPU: 61 minutes 3 seconds	
Best Validation Accuracy at Epoch = 92 Training Accuracy: 0.7821 Validation Accuracy: 0.86	

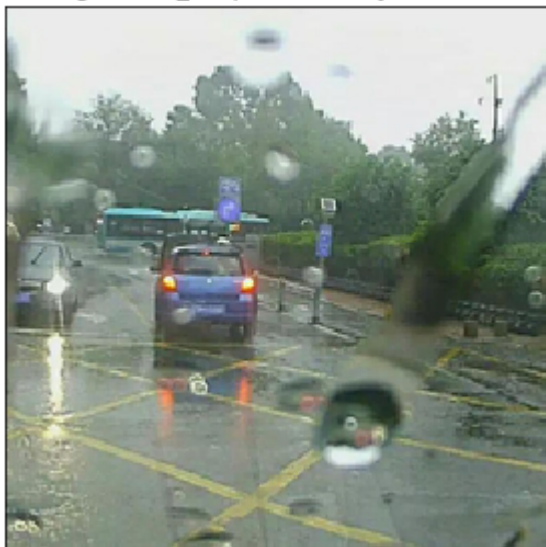
<u>Best Examples</u>	<u>Worst Examples</u>
<u>Predicted Class: Distortion, prob = 0.99</u>	<u>Misclassified: no_Distortion, prob = 0.99</u>
	
<u>Predicted Class: no_Distortion, prob = 0.99</u>	<u>Misclassified: Distortion, prob = 0.99</u>
	

Interpretation of Model:

We used python library - 'Captum' to get insights on our trained model. It shows the contribution of input at a particular position to prediction (depicted by the black/greyish markings in the heatmap). We have seen that the features detected by model are rotation or shift invariant. Following figure shows those insights and it's probability of detection with class - droplet distorted image [11][12].



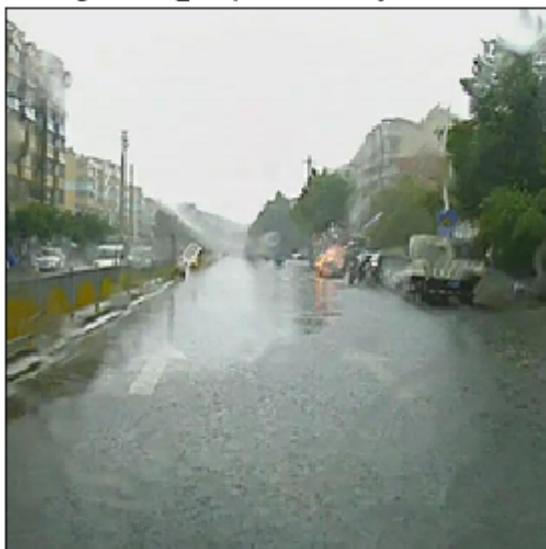
gradient_shap, Probability = 0.83



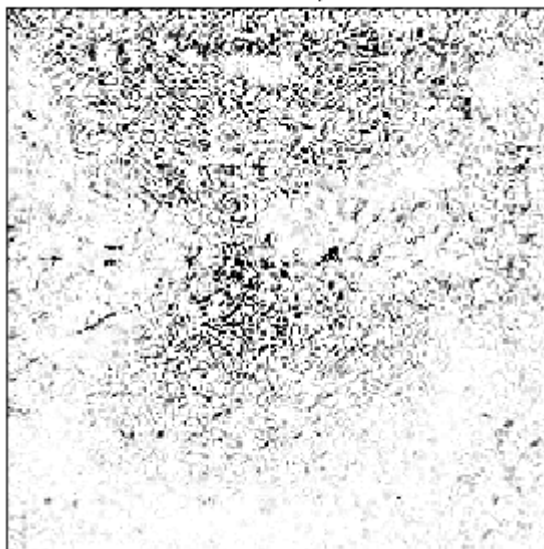
Heatmap



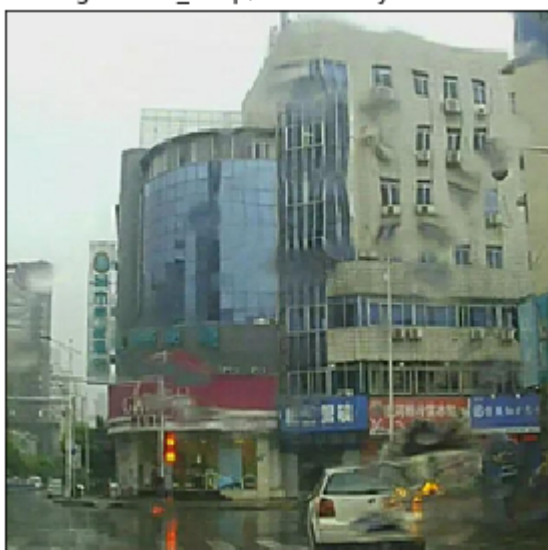
gradient_shap, Probability = 0.97



Heatmap



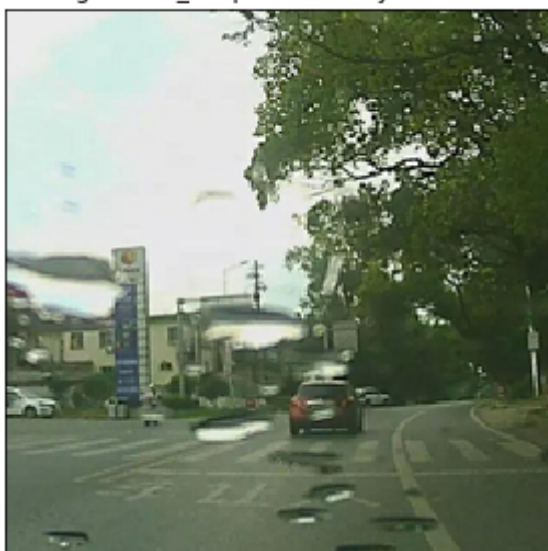
gradient_shap, Probability = 0.98



Heatmap



gradient_shap, Probability = 0.82



Heatmap



Results:

The table documents the summary of results of trained models, tested on a dataset of industry.

Model	Dataset	Details	Training Accuracy	Validation Accuracy	Testing Accuracy
Resnet-18	NewDataset12_4	Step Learning Rate, Optimizer = Adam, Weighted Cross Entropy loss	0.8443	0.813	0.5649
Resnet-18	NewDataset12_4	Cosine Learning Rate, Optimizer = SGD, Weighted Cross Entropy loss	0.8438	0.8287	0.5698
Resnet-18	NewDataset12_4	Step Learning Rate, Optimizer = SGD, Weighted Cross Entropy loss	0.847	0.8149	0.6059
VGG	FinalDataSet1	Cosine Annealing Learning Rate, Optimizer = Adam, Weighted Cross Entropy loss, Adversarial Learning	0.767	0.733	0.6289
DenseNet	FinalDataSet1	Cosine Annealing Learning Rate, Optimizer = Adam, Weighted Cross Entropy loss, Adversarial Learning	0.7945	0.6903	0.67
DenseNet	FinalDataSet1	Cosine Annealing Learning Rate, Optimizer = Adam, Weighted Cross Entropy loss	0.78	0.873	0.67
Resnet-18	FinalDataSet1	Cosine Annealing Learning Rate, Optimizer = Adam, Weighted Cross Entropy loss	0.7821	0.86	0.6946

Future Work:

Our network can now distinguish clean images from distorted images. Further, network can be trained to tackle the ill-posed problem of de-raining the images, which comes with a lot of challenges.

References:

- [1] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, Mu Li - “Bag of Tricks for Image Classification with Convolutional Neural Networks”
- [2] Rui Qian, Robby T. Tan, Wenhan Yang, Jiajun Su, and Jiaying Liu - “Attentive Generative Adversarial Network for Raindrop Removal from A Single Image” (CVPR'2018)
- [3] He Zhang, Vishwanath Sindagi, Vishal M. Patel - “Image De-raining Using a Conditional Generative Adversarial Network”
- [4] Karen Simonyan, Andrew Zisserman - “Very Deep Convolutional Networks for Large-scale Image Recognition”
- [5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun - “Deep Residual Learning for Image Recognition”
- [6] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens - “Rethinking the Inception Architecture for Computer Vision”
- [7] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger - “Densely Connected Convolutional Networks”
- [8] J. Goodfellow, Jonathon Shlens, Christian Szegedy - “Explaining and Harnessing Adversarial Examples”
- [9] <https://cs231n.github.io/transfer-learning/>
- [10] <https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10>
- [11] <https://captum.ai/api/insights.html>
- [12] <https://gilberttanner.com/blog/interpreting-pytorch-models-with-captum>