Journal- Gajaanan

OOD Generalization

Date 1	Description	Reference page
@July 21, 2023	Refer and explore about the research paper -OOD detection	LINe - Leveraging Important Neur
@July 24, 2023	Implement and test the LINe architecture - Pytorch	Untitled
@August 1, 2023	Learning algorithms - Meta learning, Ensemble learning	Meta learning for domain Generali
@August 10, 2023	Causal inference via style transfer	Untitled
@August 17, 2023	Causal invariant transformation for OOD generalization	Untitled
@August 24, 2023	SCONE- OOD detection and generalization	Untitled
@August 31, 2023	Feasibility report presentation - literature review	Untitled
@September 7, 2023	Invariant feature learning methods for OOD	Untitled
@September 14, 2023	Learned Image feature extractor and bert model	Test_FCN.ipynb - Colaboratory (g
@September 11, 2023	Learned Image feature extractor and bert model	Test_MobilenetV2.ipynb - Colabor
@September 21, 2023	Learned Image feature extractor and bert model	Test_BERT.ipynb - Colaboratory ((
@September 28, 2023	Learned Image feature extractor and bert model	Test_MobileNetV2_with_BERT.ipyr
@October 1, 2023	A sentence speaks a thousand images: domain generalization through distilling clip with language guidance (base reference for our methodology)	https://arxiv.org/pdf/2309.12530.p
	CLIP Notebooks (GitHub - openai/CLIP: CLIP (Contrastive Language-Image Pretraining), Predict the most relevant text snippet given an image)	Interacting_with_CLIP.ipynb - Cola
		Prompt_Engineering_for_ImageNe
@October 4, 2023	CLIP Image and text features to embedding mapping	https://www.kaggle.com/code/mc
	Clip model Architecture code model.py - https://github.com/openai/CLIP/blob/main/clip/model.py#L58 (Image encoder-Resnet50, Text encoder - distilbert-base-uncased)	OpenAI CLIP simple implementati
@October 5, 2023	**Natural language image search with a Dual Encoder**	https://keras.io/examples/vision/n
@October 11, 2023	Implement ResNet50 with distilbert (projection head (1,256))	Test ResNet50 with distilbert.ipyn

Date 1	Description	Reference page
@October 14, 2023	Bert Encoder Example(Text encoder - small_bert)	Copy of classify_text_with_bert.ipy
@October 16, 2023	Transformers - Explained	https://towardsdatascience.com/t
@October 18, 2023	Bert Models - Tensorflow	TensorFlow Hub (tfhub.dev)
@October 20, 2023	Image caption - Cross attention of image and text, attention map(Image encoder - MobileNetv3small)	Copy of image_captioning.ipynb -
@October 22, 2023	Transformers and multi head attention	https://colab.research.google.com
@October 25, 2023	Test inference for stage1_sample test images (gs)	Test inference_stage1.ipynb - Cola
@October 26, 2023	Test inference for stage1_sample test images (g8)	https://colab.research.google.com
	Test inference for stage1_sample trained dataset images (g8)	https://colab.research.google.com
@October 30, 2023	Modified stage1_rough (g8)	https://colab.research.google.com
@November 1, 2023	Modified Stage1 (g8)	https://colab.research.google.com
@November 28, 2023	Test inference of modified stage1 (model10,model 90) (g8)	https://colab.research.google.com
@December 4, 2023	Implement stage 1 with model and train code (include validation set)	Untitled
@December 6, 2023	new stage 1 sample training with modified loss function	Untitled
@December 10, 2023	new stage 1 training set image and text encoder non trainable	Untitled
@December 12, 2023	new stage 1 training set image and text encoder non trainable and with clip loss	Untitled
@December 16, 2023	new stage 1 code modified to train on GPU and server	Untitled
@December 18, 2023	new stage 1 code modified to train on GPU and server with masked loss function	Untitled
@January 1, 2024 → January 31, 2024	Implementation of Modified stage1, training , test inference with different variations	https://www.notion.so/Journal-Ga
@February 1, 2024 → February 29, 2024	Stage1 Evaluation and Feature visualization	https://www.notion.so/Journal-Ga
@March 1, 2024	Implementation of Sentence speak 1000 Images paper, train and evaluation on PAC dataset	• CLIP ViT-B/16 achieves 96.1%, 8 considering our reweigheted imag
@March 7, 2024	Train stage1 with multiple cross-attention layer, test inference (correct score alignment)	/home/diluksha/FYP_OOD/Gajaana
@March 11, 2024	Integrating Stage1 teacher into Sentence speak implementation, low accuracy on PACS dataset(25-35%)	Use model with output dimension
@March 18, 2024	Check and sample training of stage1 with clip encoders(on flickr30k), with RN50 as image encoder (1024 dim output embedding)	clip_stage1.pt, train loss around 3(
@March 20, 2024	Check test inference on trained CLIP_stage1 model on whole dataset, similarity matrix has the shape varied with word token size of each batch input,	Correct score matrix alignment
@March 22, 2024	Train CLIP_stage1 model with VIT as image encoder(output dim 512) on whole dataset	clip_stage1_ViT-B/16.pt
@March 25, 2024	GradCam feature visualization for Image encoder(Resnet50)	Resnet50 invariant feature visuali
@March 29, 2024	Gradcam feature visualization for stage1 with classifer (cifar10), 1-2% of classification loss	stage1_Evaluation_visualizer, mod
@April 1, 2024	Gradcam feature visualization for best stage1 with classifer(for cifar shifts). very lower classifcation loss	stage1_Evaluation_visualizer_cifar
@April 3, 2024	Stage1 with classifer training for classifcation with cifar10 dataset, comparatively higer loss and feature visualization also not good	stage1_classifer_feature_visualizat
@April 5, 2024	Gradcam feature visualization for best stage1 with classifer (imagenet),very lower classification loss similar as training	stage1_Evaluation_visualizer_imag

Date 1	Description	Reference page
	with cifar10 for best stage1	
@April 8, 2024	Train resnet50 classifier on cifar10 dataset(model: resnet50_classifier.pt) (same implementation as stage1 with classifer), check MSE between heat map images of resnet50 classifier.	Untitled
@April 9, 2024	MSE Comparision of heat maps	Untitled
@April 10, 2024	MSE Comparison heatmaps CIFAR10-C	Untitled
@April 11, 2024	Student model visualizations	Untitled
@April 12, 2024	Code implementation for generating Corrupted versions of CIFAR10 effects for sample input image.	Untitled

OOD Generalization

Our primary objective is to advance the understanding and techniques of out-of-distribution generalization in machine learning and develop a robust model to handle data distributions that deviate significantly from the training data.

Approach

In out of distribution generalization, learning invariant features makes the model more robust for different distribution shifts. In our approach we use text captions to highlight the invariant features to the model.

Model

The proposed model architecture consist of two stages:

Stage1, Teacher Network - It comprise of pre-trained image encoder and text encoder models with additional cross attention mechanism results in output image feature maps embedding which has the control from text captions as well.

Stage2, Student Network - Trained the simple image encoder model to extract the invariant features by following knowledge distillation process from stage1 teacher network. Current implementation on image classification task.

Datasets

- **Train dataset -** *Flikr30k, COCO-captions.*
- **Test dataset -** *MNIST-C, CIFAR-10-C, ImageNet-100-C, SVHN, Places365, LSUN-C, Textures, iNaturalist, PACS, VLCS, Office-Home, Terra Incognita, Digits-DG, NICO.*

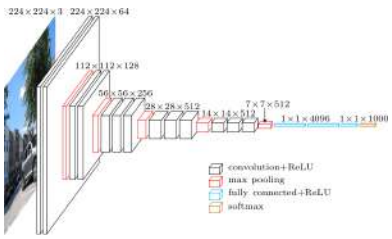
Experiment

Results

Application

Note1:

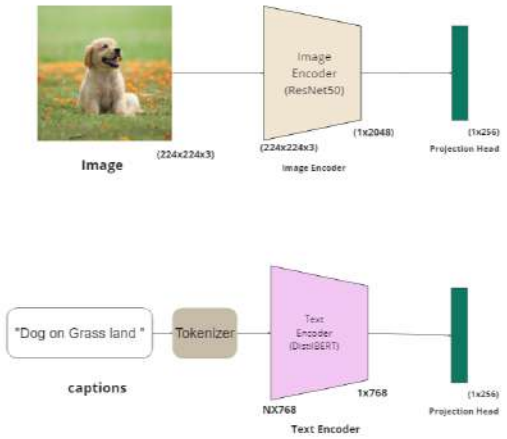
- Faster RCNN Architecture



- ResNet models Architecture

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

- Implemented ResNet50 with distilbert



- Pooling layer

Average/max Pooling	Adapti
torch.nn.AvgPool2d(*kernel_size*, *stride=None*, *padding=0*, *ceil_mode=False*, *count_include_pad=True*, *divisor_override=None*)	**torch.**
Essential to set the kernel size, stride and padding. Need to reconfigure when input size changed.	Output input s

- **MobileNets v3**

The following table describes the performance of MobileNets v3:

MACs stands for Multiply Adds

Classification Checkpoint	MACs(M)	Parameters(M)	Top1 Accuracy	Pixel1 CPU(ms)
mobilenet_v3_large_1.0_224	217	5.4	75.6	51.2
mobilenet_v3_large_0.75_224	155	4.0	73.3	39.8
mobilenet_v3_large_minimalistic_1.0_224	209	3.9	72.3	44.1
mobilenet_v3_small_1.0_224	66	2.9	68.1	15.8
mobilenet_v3_small_0.75_224	44	2.4	65.4	12.8
mobilenet_v3_small_minimalistic_1.0_224	65	2.0	61.9	12.2

resource: https://www.tensorflow.org/api_docs/python/tf/keras/applications/MobileNetV3Small

Note2

Multihead attention

Note: Multihead attention in tensorflow.

```
import tensorflow as tf

layer = tf.keras.layers.MultiHeadAttention(num_heads=2, key_dim=2)
target = tf.keras.Input(shape=[8, 16])
source = tf.keras.Input(shape=[4, 16])
output_tensor, weights = layer(target, source,
                               return_attention_scores=True)
print(output_tensor.shape)#output tensor has the shape of target(query) irrespective of num_heads, key_dim
print(weights.shape)
```

```
(None, 8, 16)
(None, 2, 8, 4)
```

```
[9] layer = tf.keras.layers.MultiHeadAttention(num_heads=2, key_dim=2)
    target = tf.keras.Input(shape=[8, 256])
    source = tf.keras.Input(shape=[4, 16])
    output_tensor, weights = layer(target, source,
                                   return_attention_scores=True)
    print(output_tensor.shape)
    print(weights.shape)
```

```
(None, 8, 256)
(None, 2, 8, 4)
```

Note3

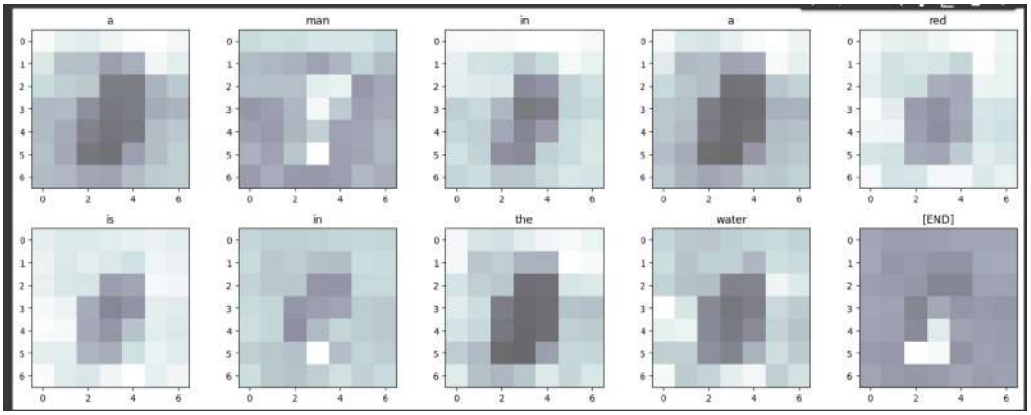
Cross attention

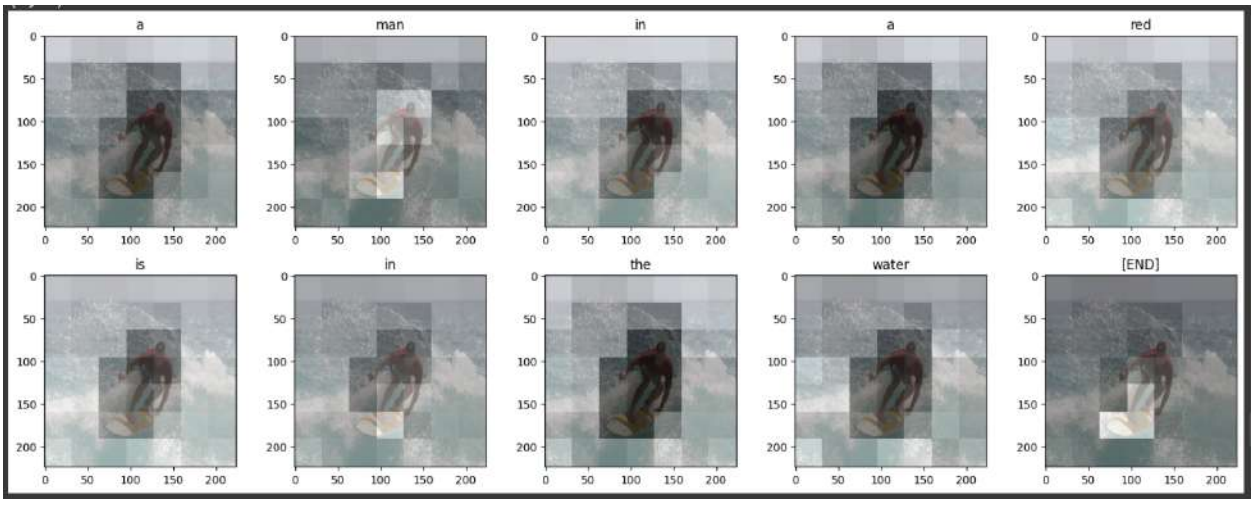
```
class CrossAttention(tf.keras.layers.Layer):
  def __init__(self,**kwargs):
    super().__init__()
    self.mha = tf.keras.layers.MultiHeadAttention(**kwargs)#(num_heads=num_heads,key_dim=units,dropout=dropout_rate)
    self.add = tf.keras.layers.Add()
    self.layernorm = tf.keras.layers.LayerNormalization()

  def call(self, x, y, **kwargs):
    print("cross attention")
    print("query_shape",x.shape)#query_shape (1, 1, 256)
    print("keyvalue_shape",y.shape)#key/value_shape (1, 49, 576)
    attn, attention_scores = self.mha(#output tensor, weights
             query=x, value=y, #key is same as value, key=y
             return_attention_scores=True)
    print("output tensor",attn.shape)#output tensor (1, 1, 256)
    print("attention scores",attention_scores.shape)#attention scores (1, 2, 1, 49)
    self.last_attention_scores = attention_scores

    x = self.add([x, attn])
    return self.layernorm(x)
```

Attention-map

```
attn_maps = [layer.last_attention_scores for layer in model.decoder_layers]
[map.shape for map in attn_maps]

attention_maps = tf.concat(attn_maps, axis=0)
attention_maps = einops.reduce(attention_maps,'batch heads sequence (height width) -> sequence height width',height=7, width=7,rec
einops.reduce(attention_maps, 'sequence height width -> sequence', reduction='sum')

def plot_attention_maps(image, str_tokens, attention_map):
    fig = plt.figure(figsize=(16, 9))
    len_result = len(str_tokens)
    titles = []
    for i in range(len_result):
      map = attention_map[i]
      grid_size = max(int(np.ceil(len_result/2)), 2)
      ax = fig.add_subplot(3, grid_size, i+1)
      titles.append(ax.set_title(str_tokens[i]))
      ax.imshow(map, cmap='bone', alpha=0.6,clim=[0.0, np.max(map)])
    plt.tight_layout()
plot_attention_maps(image/255, str_tokens, attention_maps)
```

bone





Note4

Implementation of modified stage1 with contrastive loss function (masked)

Model instance

```python
# Create an instance of the ImageTextMatchingModel
image_encoder = ImageEncoder("resnet50")
max_sequence_length = 30
text_encoder = TextEncoder("distilbert-base-uncased", max_sequence_length)
image_projection_head = ProjectionHead(2048)
text_projection_head = ProjectionHead(768)
embedding_dim = 256
cross_aatn = CrossAttentionLayer(embedding_dim, num_heads=4)

model = ImageTextMatchingModel(
    image_encoder=image_encoder,
    text_encoder=text_encoder,
    image_projection_head=image_projection_head,
    text_projection_head=text_projection_head,
    cross_attention_layer=cross_aatn
)
```

Trained on fliker30k dataset (5epochs, 40 minutes per epoch)

```
The number of images in the train set is : 127132
The number of images in the val set is : 31783
the size of the train dataloader 31783 batches of 4
the size of the test dataloader 7946 batches of 4
```

```
Epoch: 5
100%|██████████| 31783/31783 [38:31<00:00, 13.75it/s, lr=0.001, train_loss=0.00221]
100%|██████████| 7946/7946 [04:53<00:00, 27.10it/s, valid_loss=0.00221]
Saved Best Model!
```

Test inference

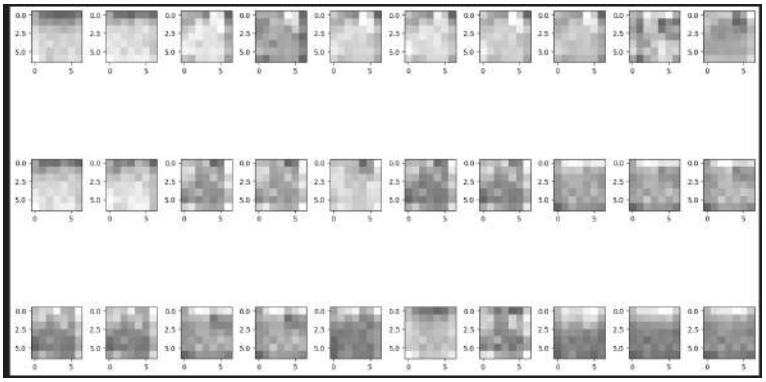
```python
# Create an instance of the model
model = ImageTextMatchingModel(
    image_encoder=image_encoder,
    text_encoder=text_encoder,
    image_projection_head=image_projection_head,
    text_projection_head=text_projection_head,
    cross_attention_layer=cross_aatn
)
image_captioning_model = model
model_path = '/home/diluksha/FYP_OOD/Gajaanan/Notebooks/best_copy.pt'

image_captioning_model.load_state_dict(torch.load(model_path, map_location='cpu'))
image_captioning_model.eval()
```

Test inference on trainset

score output

```
tensor([[ 0.9870,  0.2658, -0.0195,  0.4770],
        [ 0.2476,  0.9609,  0.2938, -0.1010],
        [-0.0092,  0.3301,  0.9852, -0.0855],
        [ 0.4359, -0.1055, -0.1333,  0.9914]])
```

Text and image embedding mapping



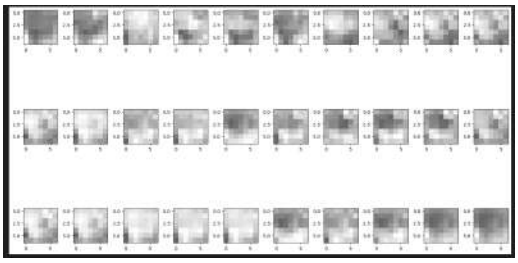
“Five people are sitting in a circle with instruments”

Test inference on sample test set

score output

```
tensor([[0.9831, 0.7531, 0.2524, 0.5252],
        [0.7291, 0.9884, 0.3684, 0.4629],
        [0.1944, 0.2851, 0.9801, 0.5737],
        [0.4502, 0.3890, 0.6357, 0.9874]], grad_fn=<CopySlices>)
```

Text and image embedding mapping



“Dog on a grass land”

Further move on stage1

Find ways to improving accuracy on trained model.- Prevent overfitting

reason for training, validation loss start with low values in initial epochs

Try different combinations of image and text encoder models for stage1.

select best feature extractor image encoder and more text feature extractor text encoder

image encoder: resnet50, resnet152

text encoder: distilbert, bert

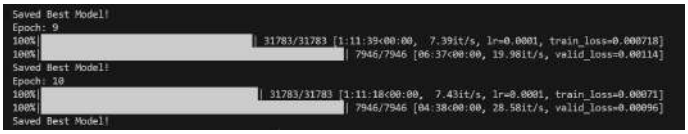
Training stage1 model with different image-caption datasets: COCO captions.

Use data augmentation for trainset

```
# Data augmentation for images in the training set
image_augmentation = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(20),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2),
])
```

Start with low train and validation loss values.

Trained for 10 epochs (1hr and 15 min per epoch)



```
Saved Best Model!
Epoch: 9
100%| | 31783/31783 [1:11:39<00:00,  7.39it/s, lr=0.0001, train_loss=0.000718]
100%| | 7946/7946 [06:37<00:00, 19.98it/s, valid_loss=0.00114]
Saved Best Model!
Epoch: 10
100%| | 31783/31783 [1:11:18<00:00,  7.43it/s, lr=0.0001, train_loss=0.000671]
100%| | 7946/7946 [04:38<00:00, 28.58it/s, valid_loss=0.00096]
Saved Best Model!
```

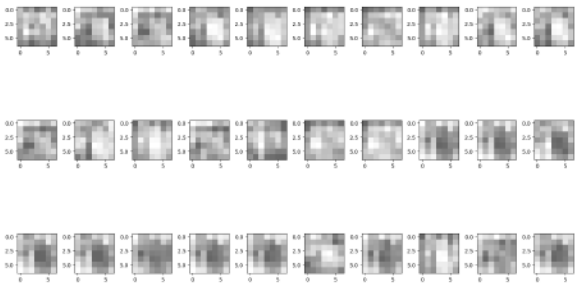
```
tensor([[ 0.9944,  0.2937,  0.1876,  0.1337],
        [ 0.2616,  0.9968,  0.1033, -0.3793],
        [ 0.1940,  0.1012,  0.9971, -0.5023],
        [ 0.1242, -0.3874, -0.4933,  0.9971]])
```

Trained set score

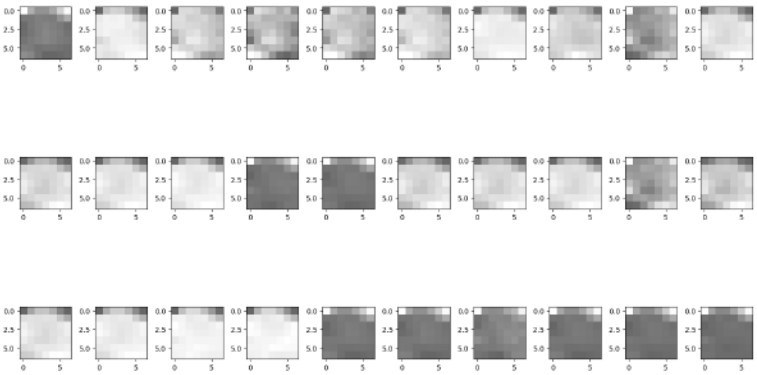
Train-set mapping

```
tensor([[0.9967, 0.4946, 0.3874, 0.5505],
        [0.4864, 0.9971, 0.6574, 0.6933],
        [0.3828, 0.6526, 0.9965, 0.7406],
        [0.5435, 0.6864, 0.7412, 0.9981]], grad_fn=<CopySlices>)
```

Test-set score



Test set mapping



Aeroplane fly in the sky

Text encoder with BiGRU Network

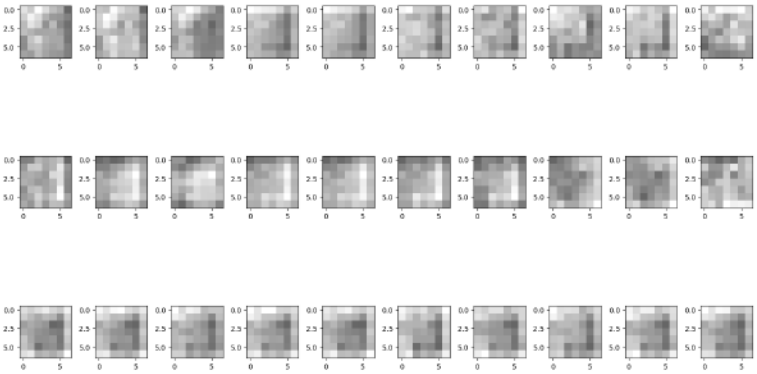
```python
class BiGRU(nn.Module):
    def __init__(self, input_size, hidden_size, bidirectional=True, batch_first=True):
        super(BiGRU, self).__init__()
        self.gru = nn.GRU(
            input_size=input_size,
            hidden_size=hidden_size,
            bidirectional=bidirectional,
            batch_first=batch_first
        )

    def forward(self, x):
        # Forward pass through the GRU
        text_embeddings, _ = self.gru(x)
        #print("bigru",text_embeddings.shape)
        return text_embeddings
```

Training details

Trained for 10 epoch(1hr and 30 minutes per epoch)

Image augmentation used

Inference results



This also results the image-text aligned embedding map: reason may be due to transfer learning approach weights corresponds to image and text encoders are not trained. **No, but with finetuning(ref: with bert text encoder: finetune training 1 epoch - best_new_50bert_finetune.pt) also I got embedding values for 30 tokens as well.**

Modified stage1

Initial model: Trained 2 epochs - best.pt

with masked loss: Trained 5 epochs - best_copy.pt

with image augmentation: Trained 10 epochs - best_aug1.pt

text encoder with bigru: trained 1 epoch - best_old.pt, trained 10 epochs - best_old_1.pt

with bert text encoder: transfer learning training 5 epochs - best_new_50bert.pt

with bert text encoder: finetune training 1 epoch - best_new_50bert_finetune.pt

Modified stage1 trained using MS-COCO captions dataset

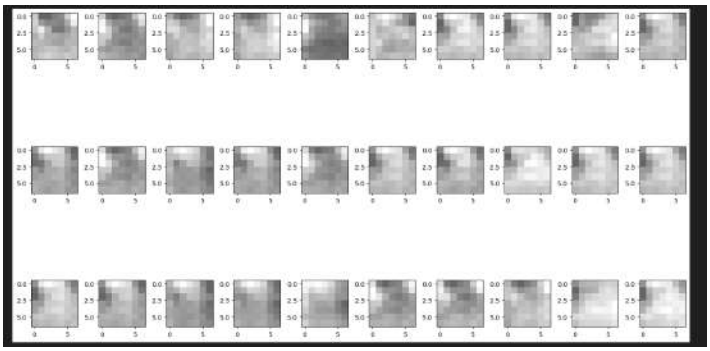
```
Epoch: 1
  0%|          | 0/82823 [00:00<?, ?it/s]
100%|██████████| 82823/82823 [1:33:14<00:00, 14.80it/s, lr=0.001, train_loss=0.00632]
100%|██████████| 20706/20706 [19:26<00:00, 17.74it/s, valid_loss=0.00642]
Saved Best Model!
Epoch: 2
 99%|█████████▉| 82391/82823 [1:55:42<00:35, 12.04it/s, lr=0.001, train_loss=0.00238]
```

Test inference

Test score output

```
tensor([[0.9807, 0.5878, 0.7408, 0.7174],
        [0.5561, 0.9831, 0.6690, 0.7581],
        [0.7690, 0.6653, 0.9848, 0.6764],
        [0.7280, 0.7743, 0.6532, 0.9861]], grad_fn=<CopySlices>)
```

Attention mapping



“Car on the road”

Modified stage1 training using bert text encoder

Transfer learning with flickr30k dataset (5 epochs, best_new_50bert.pt)

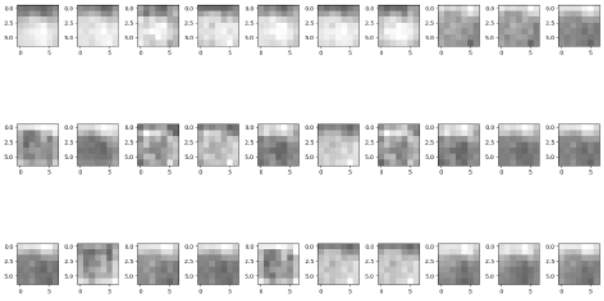
```
Total learnable parameters: 1117184
The number of images in the train set is : 127132
The number of images in the val set is : 31783
the size of the train dataloader 31783 batches of 4
the size of the test dataloader 7946 batches of 4
```

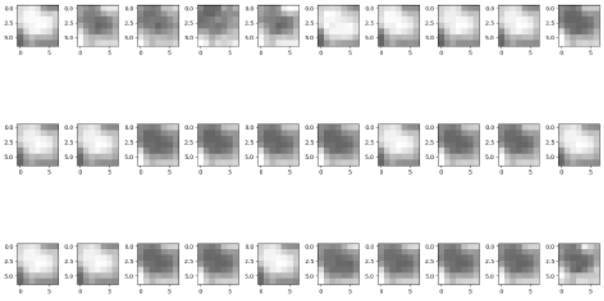
```
100%|| 7946/7946 [04:02<00:00, 32.71it/s, valid_loss=0.0019]
Epoch: 5
100%|| 31783/31783 [32:35<00:00, 16.25it/s, lr=0.001, train_loss=0.00142]
100%|| 7946/7946 [03:44<00:00, 35.46it/s, valid_loss=0.00167]
dilukshe@aberdeen:~/FYP_OOD/Gajaanan/Stage1(new)$
```

Score outputs

```
tensor([[0.9786, 0.0397, 0.3914, 0.2964],
        [0.0232, 0.9647, 0.0076, 0.4338],
        [0.4281, 0.0656, 0.9728, 0.4823],
        [0.2604, 0.4238, 0.4017, 0.9671]])
```

```
tensor([[ 0.9795,  0.5640,  0.2811,  0.2514],
        [ 0.6088,  0.9728,  0.1400,  0.0070],
        [ 0.2810, -0.0451,  0.9792,  0.3158],
        [ 0.1986, -0.1883,  0.3331,  0.9690]], grad_fn=<CopySlices>)
```





“Dog on grassland”

Finetune with flickr30k dataset (1 epoch, best_new_50bert_finetune.pt)

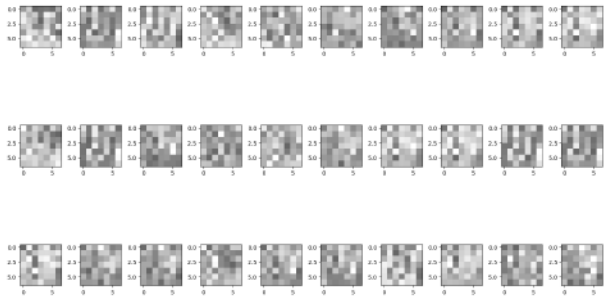
```
Total learnable parameters: 134107456
The number of images in the train set is : 127132
The number of images in the val set is : 31783
the size of the train dataloader 31783 batches of 4
the size of the test dataloader 7946 batches of 4
```

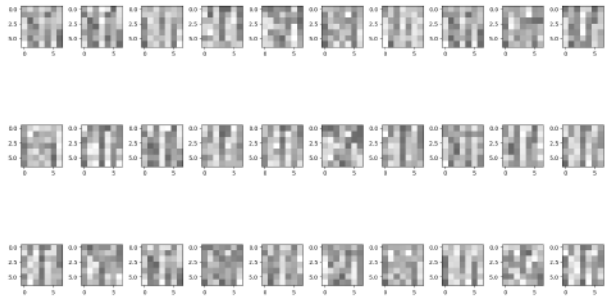
```
100%| | 31783/31783 [1:16:53<00:00,  6.89it/s, lr=0.001, train_loss=0.396]
100%| | 7946/7946 [02:50<00:00, 46.50it/s, valid_loss=4.8]
Saved Best Model!
```

Score outputs

```
tensor([[0.7265, 0.7265, 0.7265, 0.7265],
        [0.7265, 0.7265, 0.7265, 0.7265],
        [0.7265, 0.7265, 0.7265, 0.7265],
        [0.7265, 0.7265, 0.7265, 0.7265]])
```

```
tensor([[0.7265, 0.7265, 0.7265, 0.7265],
        [0.7265, 0.7265, 0.7265, 0.7265],
        [0.7265, 0.7265, 0.7265, 0.7265],
        [0.7265, 0.7265, 0.7265, 0.7265]], g
```





Stage1 Evaluation using image classification on cifar10 dataset

Use pretrained Resnet50 as base model, follows transfer learning and trained to 5 epochs

```
class ResNet50Model(nn.Module):
    def __init__(self, num_classes=10):
        super(ResNet50Model, self).__init__()
        resnet50 = models.resnet50(pretrained=True)
        self.base_model = nn.Sequential(*list(resnet50.children())[:-2])
        for param in self.base_model.parameters():
            param.requires_grad = False
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(2048*49, 512)
        self.batch_norm1 = nn.BatchNorm1d(512)
        self.dropout1 = nn.Dropout(0.5)
        self.fc2 = nn.Linear(512, 256)
        self.batch_norm2 = nn.BatchNorm1d(256)
        self.dropout2 = nn.Dropout(0.5)
        self.fc3 = nn.Linear(256, num_classes)
```

```
Epoch 1/5, Loss: 0.6601129254835951
Epoch 2/5, Loss: 0.43577855211846966
Epoch 3/5, Loss: 0.3416300677620541
Epoch 4/5, Loss: 0.2842717786960285
Epoch 5/5, Loss: 0.2375465763442199
Accuracy on the test set: 86.07%
```

```
    def forward(self, x):
        x = self.base_model(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.batch_norm1(x)
        x = nn.functional.relu(x)
        x = self.dropout1(x)
        x = self.fc2(x)
        x = self.batch_norm2(x)
        x = nn.functional.relu(x)
        x = self.dropout2(x)
        x = self.fc3(x)
        return x
```

Use pretrained stage1 model as base model, follows transfer learning and trained to 5 epochs

```
class ImageCaptioningResNetModel(nn.Module):
    def __init__(self, num_classes=10):
        super(ImageCaptioningResNetModel, self).__init__()
        self.image_captioning_model = image_captioning_model # Replac
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(256*49, 512)
        self.batch_norm1 = nn.BatchNorm1d(512)
        self.dropout1 = nn.Dropout(0.5)
        self.fc2 = nn.Linear(512, 256)
        self.batch_norm2 = nn.BatchNorm1d(256)
        self.dropout2 = nn.Dropout(0.5)
        self.fc3 = nn.Linear(256, num_classes)

    # Freeze the parameters of the image_captioning_model
        for param in self.image_captioning_model.parameters():
            param.requires_grad = False
```

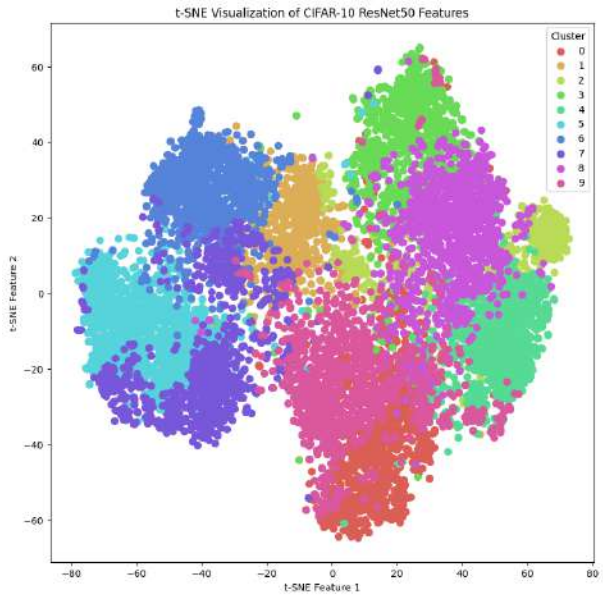
```
Epoch 1/5, Loss: 0.7234257697287202
Epoch 2/5, Loss: 0.6147956955755874
Epoch 3/5, Loss: 0.5740071919362992
Epoch 4/5, Loss: 0.5637095069368928
Epoch 5/5, Loss: 0.549053177734986
Accuracy on the test set: 100.0%
```

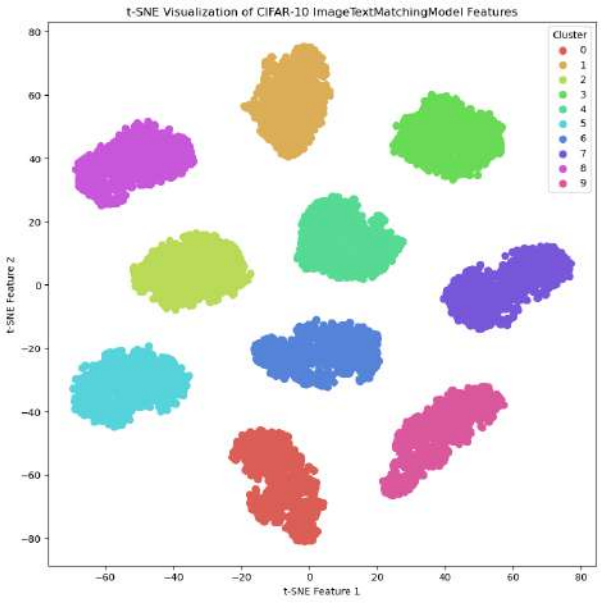
Training and test evaluation using CIFAR10 dataset

```python
def forward(self, images, texts):
    x = self.image_captioning_model(images, texts)
    x = self.flatten(x[1])
    x = self.fc1(x)
    x = self.batch_norm1(x)
    x = nn.functional.relu(x)
    x = self.dropout1(x)
    x = self.fc2(x)
    x = self.batch_norm2(x)
    x = nn.functional.relu(x)
    x = self.dropout2(x)
    x = self.fc3(x)
    return x
```

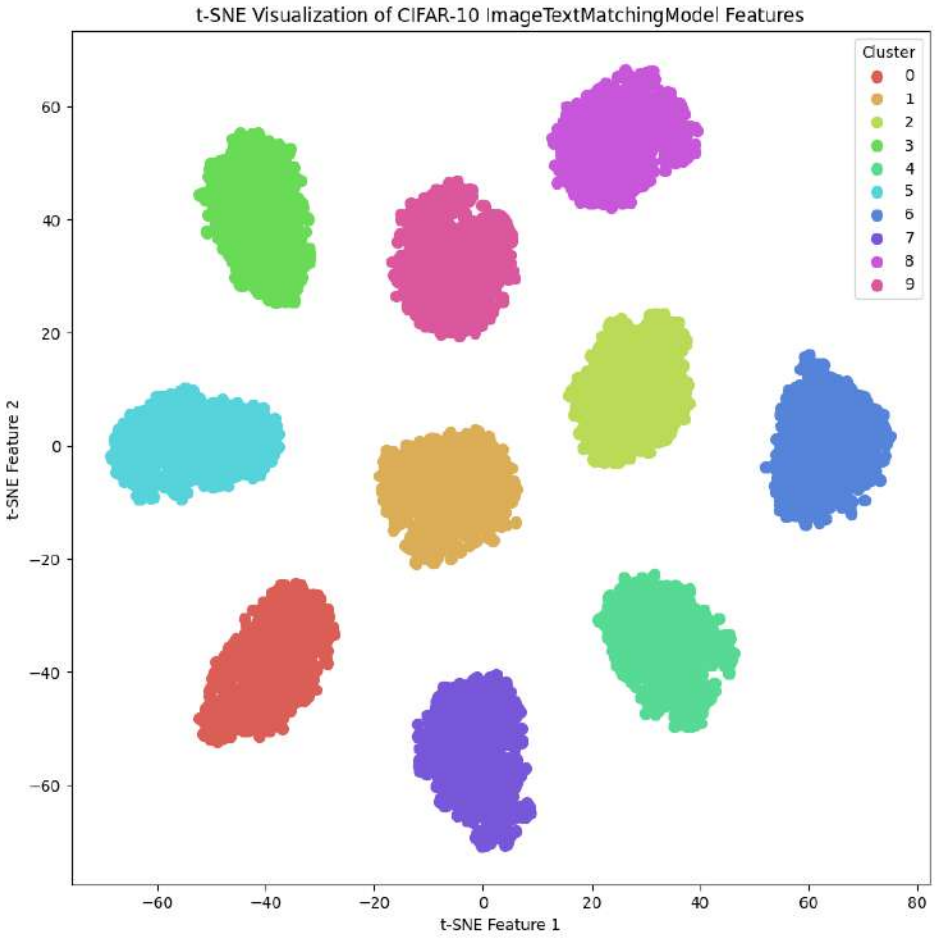
Training - CIFAR10 and Testing - CIFAR10C datasets

Feature Visualization by CIFAR10 classes





Feature separation for best_stage1 model ("/home/diluksha/FYP_OOD/Sajeepan/Models/degs_test.pt")



Stage1 visualization

In order to visualize the invariant features capturing in stage1 implementation, Grad CAM tool is used.

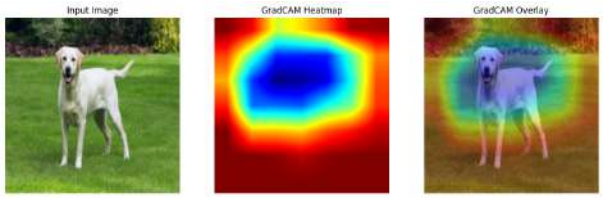
```
# Instantiate the ImageCaptioningResNetModel
image_captioning_resnet_model = ImageCaptioningResNetModel(num_classes=10)
model_path = '/home/diluksha/FYP_OOD/Gajaanan/Stage1(new)/stage1_classifier.pt'
image_captioning_resnet_model.load_state_dict(torch.load(model_path, map_location='cpu'))
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
image_captioning_resnet_model.to(device)
image_captioning_resnet_model.eval()
✓ 0.7s
```

Therefore, with our stage1 feature extractor, classifier network is added on top of it

- Initial model trained (5 epochs) with cifar10 dataset and checked for visualization.(stage1_classifier.pt)
- Further extend this to trained with ImageNet dataset and check the visualization.

Initial results with cifar10 trained models

Grad CAM Visualization

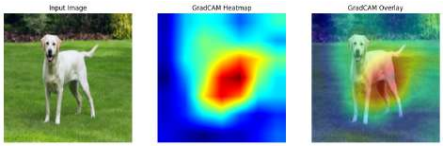


Classification with pretrained resnet50

Here, As an image encoder pretrained resnet50 model on *imagenet 1000* class is considered.

Last convolutional layer is considered for class activation map.

Predicted class: 208 - Labrador retriever



Classification with trained stage1 model

Here our stage 1 model considered. For grad CAM visualization classification task on CIFAR10 class trained and tested.

Attention output of cross attion layer is considered for class activation map.

Predicted class: 5 - dog

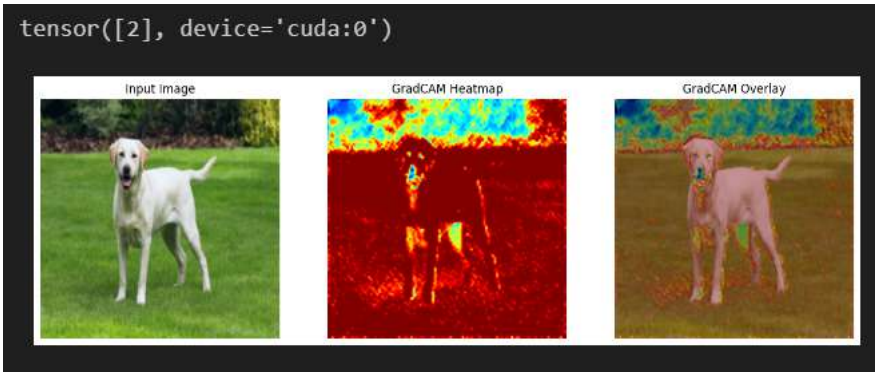
Resnet50_classifier trained on cifar10 dataset for 10 epochs

```python
class ResNet50Model(nn.Module):
    def __init__(self, num_classes=10):
        super(ResNet50Model, self).__init__()
        resnet50 = models.resnet50(pretrained=True)
        self.base_model = nn.Sequential(*list(resnet50.children())[:-2])
        for param in self.base_model.parameters():
            param.requires_grad = False
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(2048*49, 512)
        self.batch_norm1 = nn.BatchNorm1d(512)
        self.dropout1 = nn.Dropout(0.5)
        self.fc2 = nn.Linear(512, 256)
        self.batch_norm2 = nn.BatchNorm1d(256)
        self.dropout2 = nn.Dropout(0.5)
        self.fc3 = nn.Linear(256, num_classes)

    def forward(self, x):
        x = self.base_model(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.batch_norm1(x)
        x = nn.functional.relu(x)
        x = self.dropout1(x)
        x = self.fc2(x)
        x = self.batch_norm2(x)
        x = nn.functional.relu(x)
        x = self.dropout2(x)
        x = self.fc3(x)
        return x
```

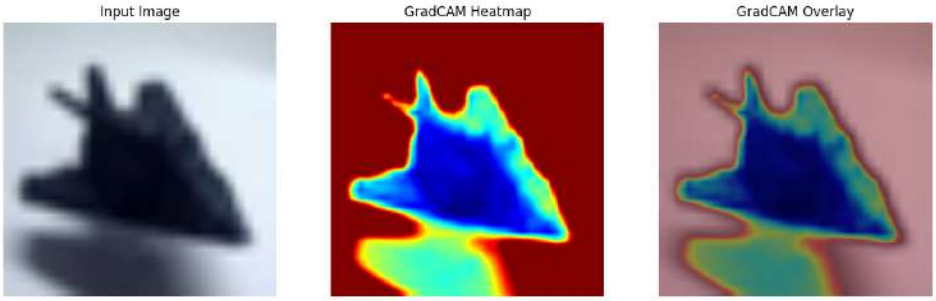
```
  warnings.warn(msg)
Epoch 1/10, Loss: 0.6655287366084126
Epoch 2/10, Loss: 0.43317510784053437
Epoch 3/10, Loss: 0.34253863330044404
Epoch 4/10, Loss: 0.2870911979033133
Epoch 5/10, Loss: 0.2384402237622939
Epoch 6/10, Loss: 0.20329765512910494
Epoch 7/10, Loss: 0.17784036166937378
Epoch 8/10, Loss: 0.15391119961362437
Epoch 9/10, Loss: 0.139360533243574
Epoch 10/10, Loss: 0.12771114635893413
Accuracy on the test set: 86.01%
```

Grad CAM visualization on test image (Wrong class prediction)

Sample test data



In distribution data (Cifar10 plane class image)

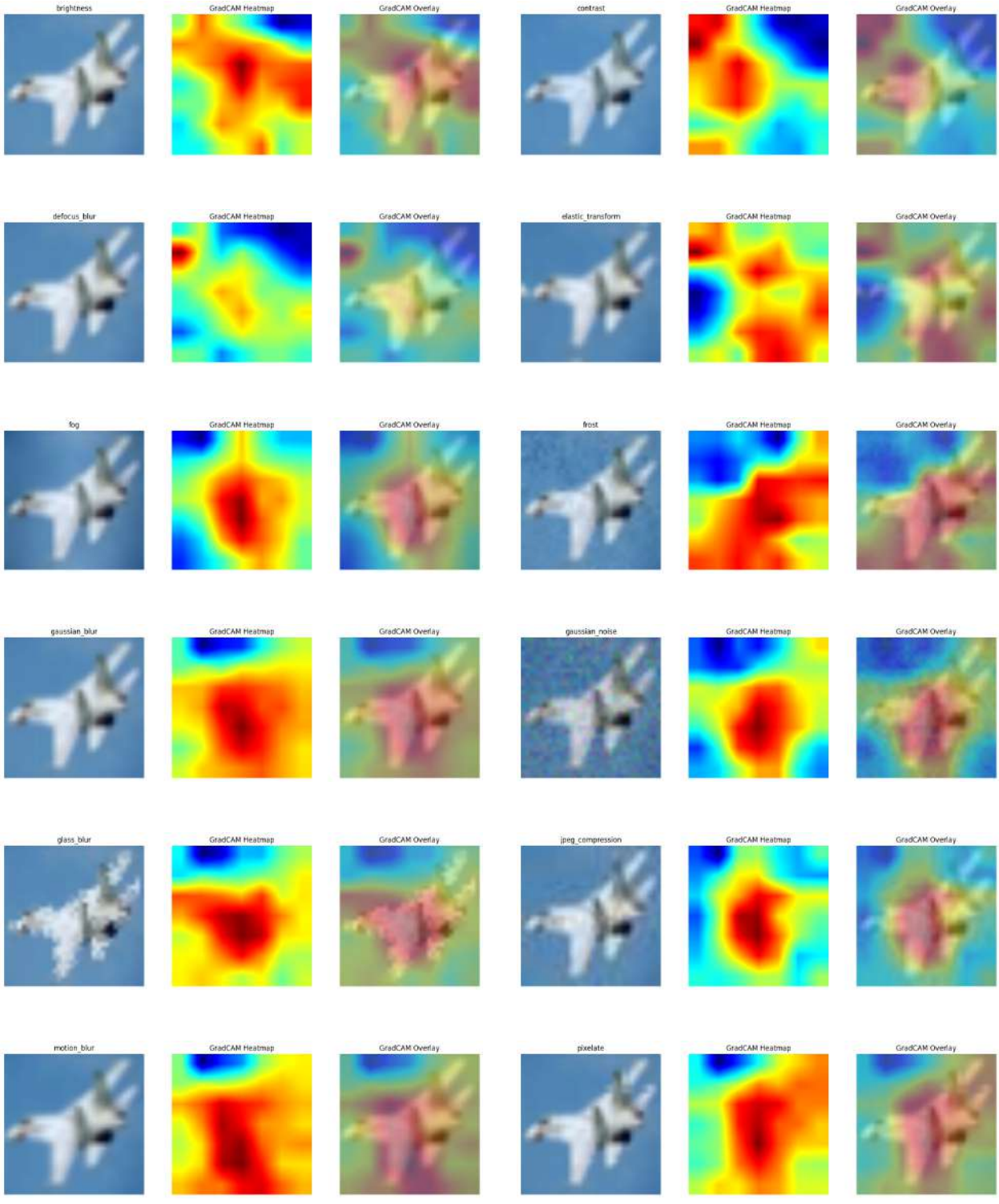


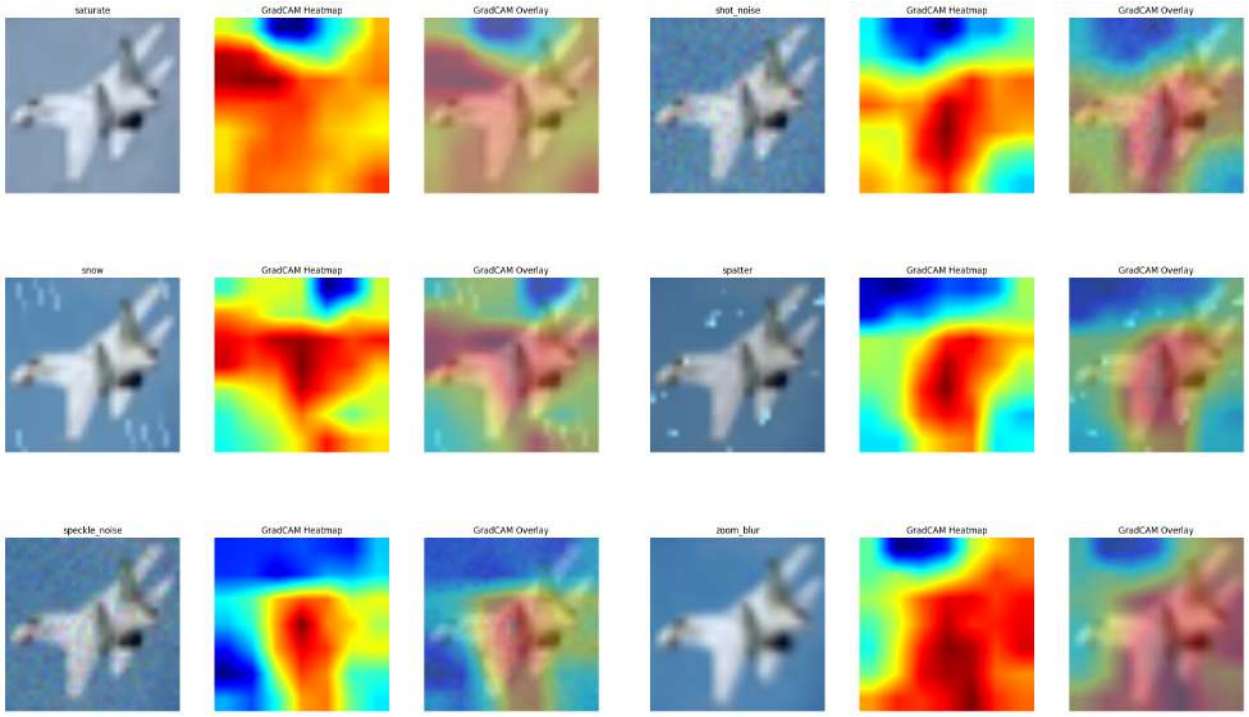
Feature visualization with different shifted CIFAR10-C datasets

There are 18 different corrupted versions of shifts are presents in the cifar10-c datasets.

Grad-CAM visualizations(stage1_classifier, for all corrupted shifts predicted class: 0, plane) on CIFAR10C

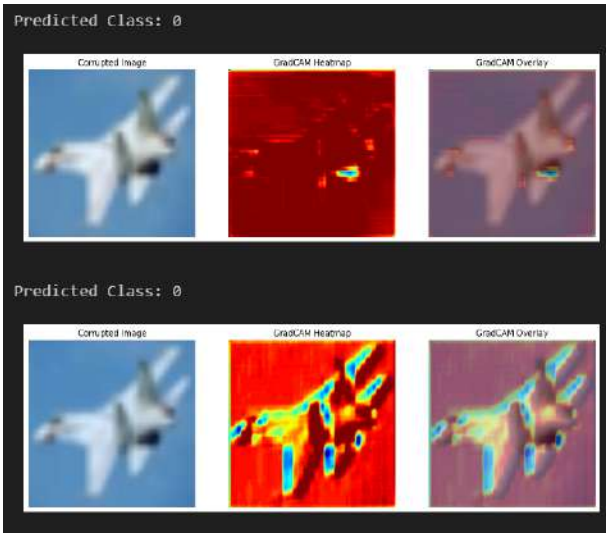
(stage1 model trained on flickr30k dataset, best_new_50bert.pt)

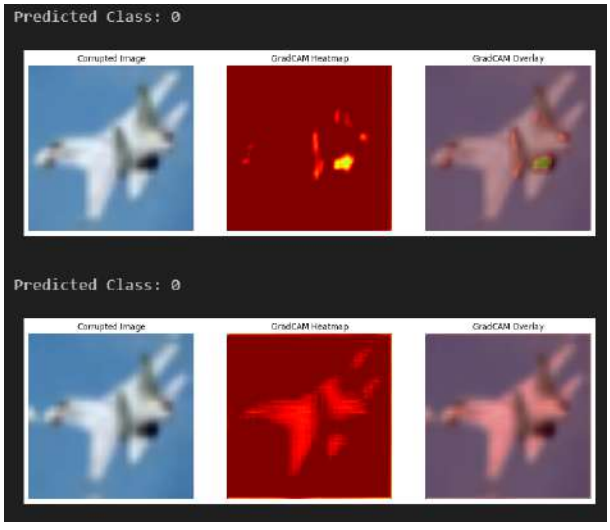

brightness
GradCAM Heatmap
GradCAM Overlay
contrast
GradCAM Heatmap
GradCAM Overlay
defocus_blur
GradCAM Heatmap
GradCAM Overlay
elastic_transform
GradCAM Heatmap
GradCAM Overlay
fog
GradCAM Heatmap
GradCAM Overlay
frost
GradCAM Heatmap
GradCAM Overlay
gaussian_blur
GradCAM Heatmap
GradCAM Overlay
gaussian_noise
GradCAM Heatmap
GradCAM Overlay
glass_blur
GradCAM Heatmap
GradCAM Overlay
jpeg_compression
GradCAM Heatmap
GradCAM Overlay
motion_blur
GradCAM Heatmap
GradCAM Overlay
pixelate
GradCAM Heatmap
GradCAM Overlay

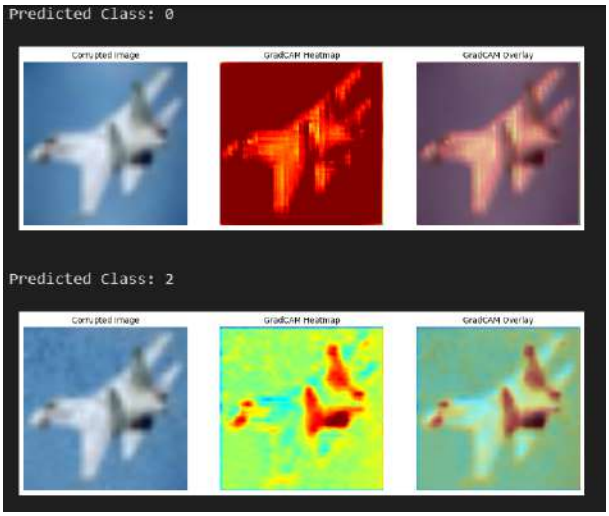


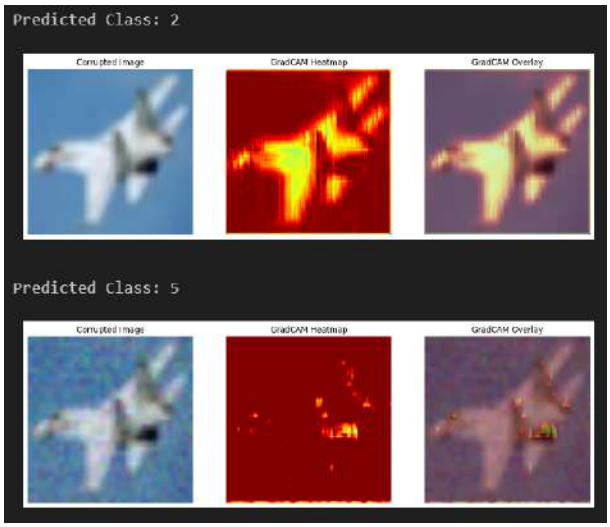
Grad-CAM visualizations(resnet50_classifier) on CIFAR10C

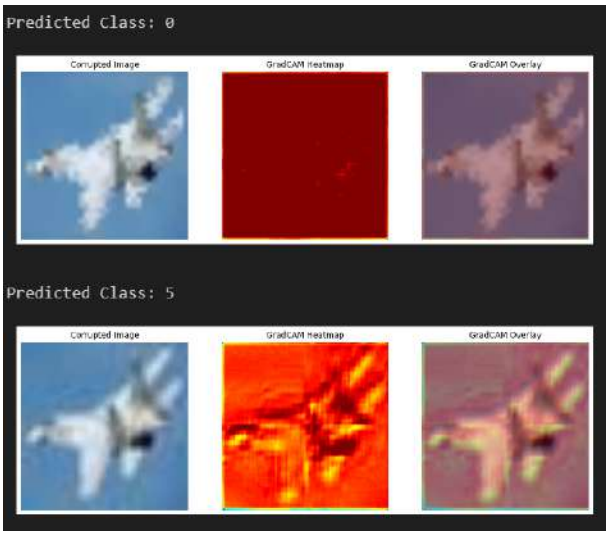
(pretrained resnet50 with classifier for cifar10)

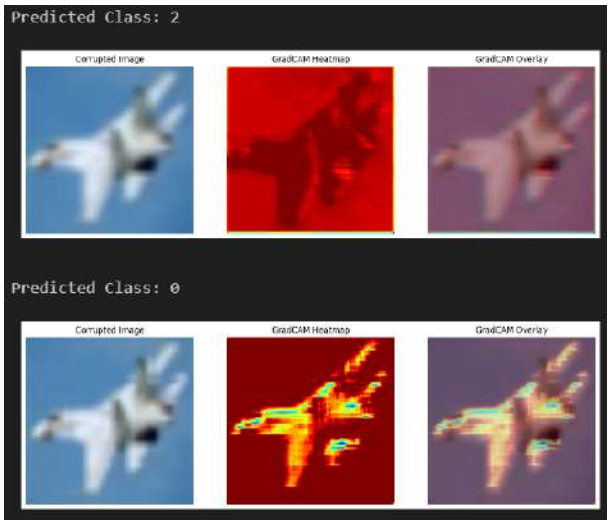


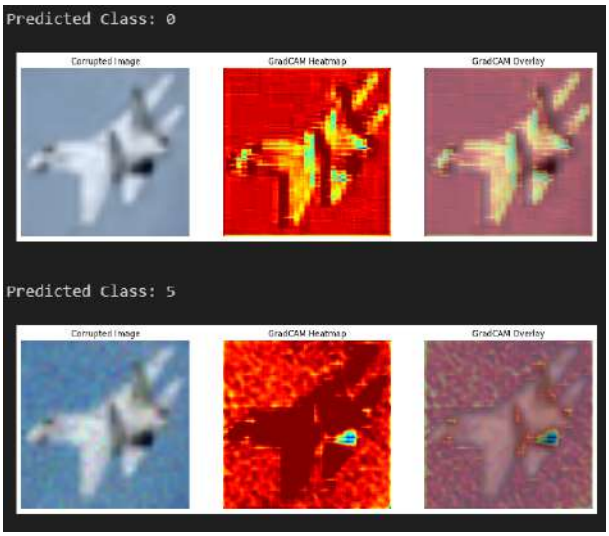


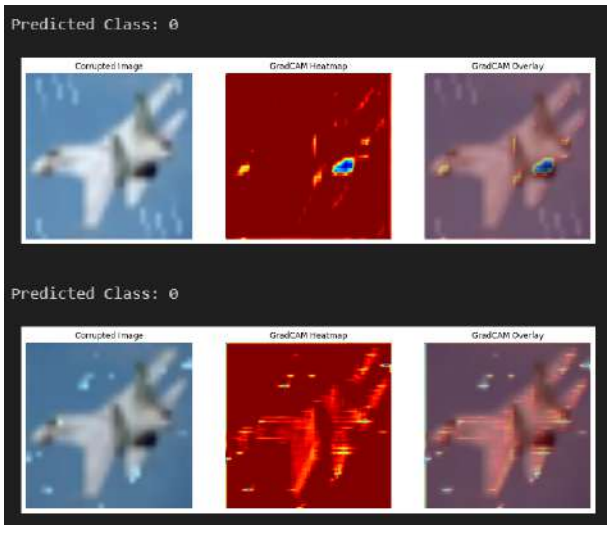

Predicted Class: 0
Corrupted Image
GradCAM Heatmap
GradCAM Overlay
Predicted Class: 2
Corrupted Image
GradCAM Heatmap
GradCAM Overlay

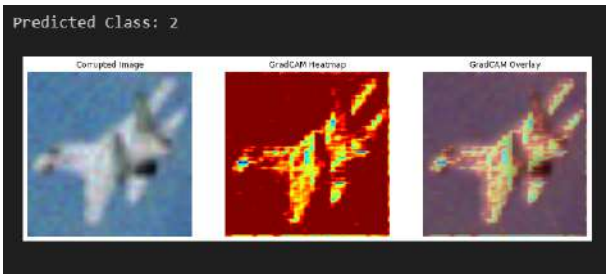

Predicted Class: 2
Corrupted Image
GradCAM Heatmap
GradCAM Overlay
Predicted Class: 5
Corrupted Image
GradCAM Heatmap
GradCAM Overlay


Predicted Class: 0
Corrupted Image
GradCAM Heatmap
GradCAM Overlay
Predicted Class: 5
Corrupted Image
GradCAM Heatmap
GradCAM Overlay


Predicted Class: 2
Corrupted Image
GradCAM Heatmap
GradCAM Overlay
Predicted Class: 0
Corrupted Image
GradCAM Heatmap
GradCAM Overlay


Predicted Class: 0
Corrupted Image
GradCAM Heatmap
GradCAM Overlay
Predicted Class: 5
Corrupted Image
GradCAM Heatmap
GradCAM Overlay

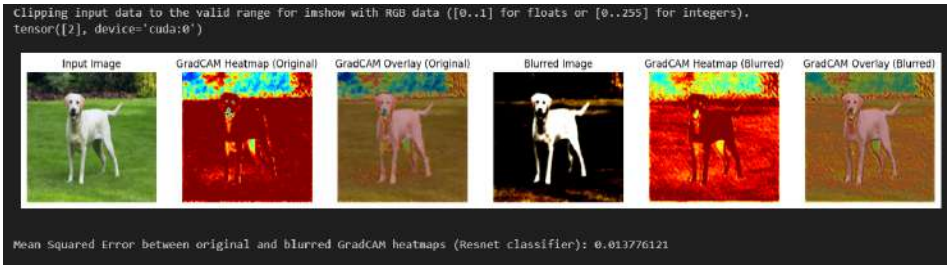

Predicted Class: 0
Corrupted Image
GradCAM Heatmap
GradCAM Overlay
Predicted Class: 0
Corrupted Image
GradCAM Heatmap
GradCAM Overlay

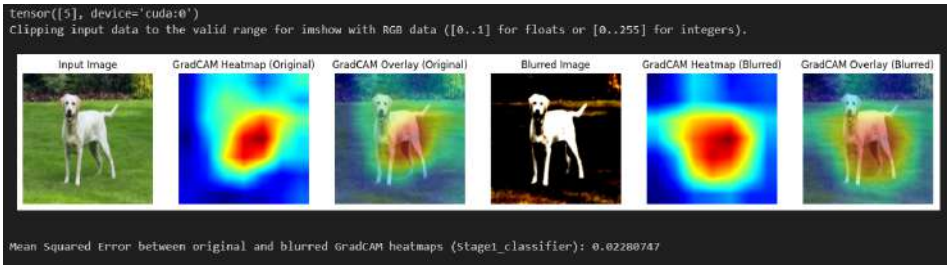


CIFAR10 classes: ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

MSE of resnet50 classifier and stage1 classifier for original and gaussian blurred images.

```
# Generate a blurred version of the input image
   blurred_image = gaussian_filter(input_image.squeeze().cpu().numpy(), sigma=1)
```





Resnet50 classifier	Stage1 classifier	Blurred Image (Sigma)
0.013776121	0.02280747	1
0.03096206	0.09212932	2
0.019949704	0.12074152	3
0.031366926	0.13294995	4
0.033826333	0.13743807	5s

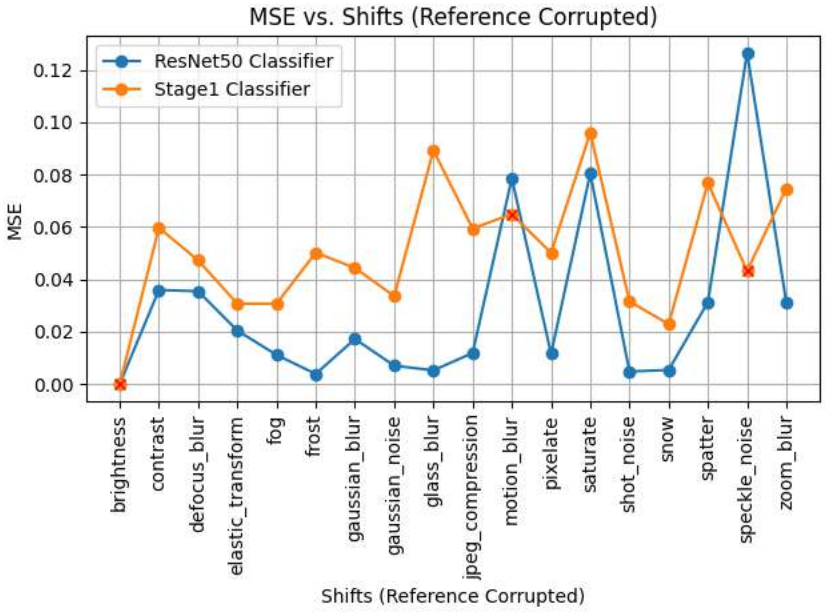
Resnet50 classifier

MSE values: [8.1713734e-16, 0.035943665, 0.035445806, 0.020446094, 0.011137732, 0.0038944623, 0.017241333, 0.0070375926, 0.005259537, 0.011881855, 0.07858172, 0.011560443, 0.080409676, 0.0048175235, 0.005394072, 0.031293496, 0.12653863, 0.03111785]

Note: Even for the reference image, grad cam visualization is not good, exact features are not correctly visualized.

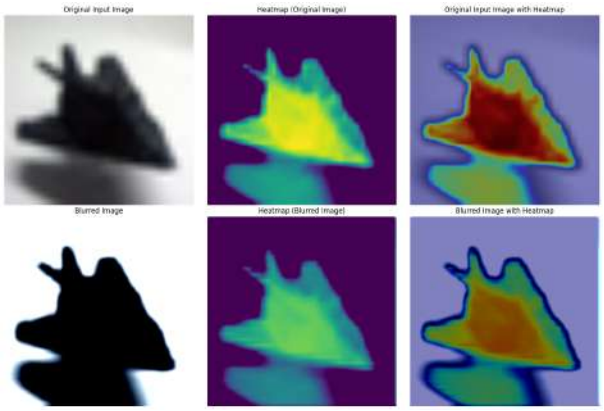
stage1 classifier

MSE values: [0.0, 0.05966551, 0.047272068, 0.030664518, 0.030741444, 0.050181653, 0.044350542, 0.03361135, 0.08924288, 0.059416275, 0.06493317, 0.050105814, 0.09593932, 0.031605944, 0.022880618, 0.07676983, 0.043204136, 0.074711956]



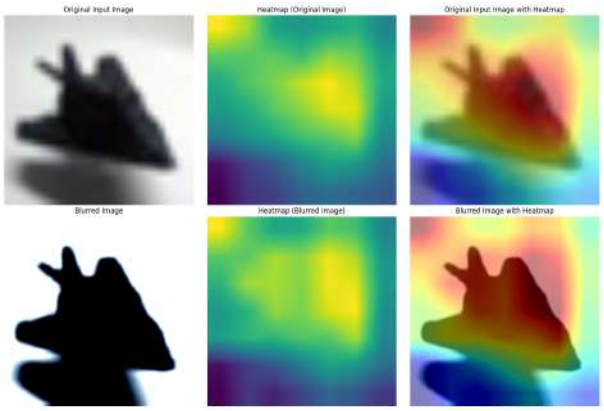
Note: By this MSE comparisons stage1 classifier has considerably lower values for the shifts of *motion_blur and speckle_noise*.

MSE of resnet50 classifier for original and gaussian blurred images.

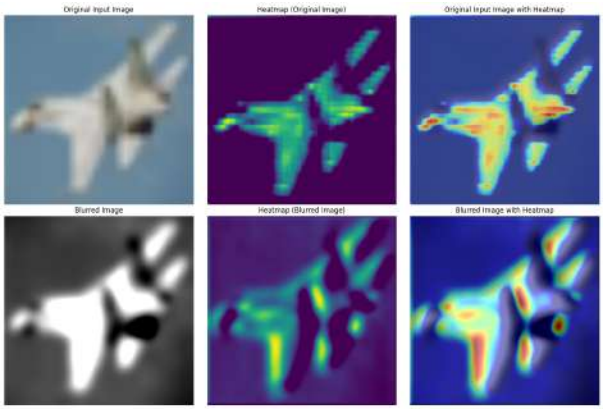


Mean Squared Error (MSE) between original and noise-added heatmaps: 0.009812849

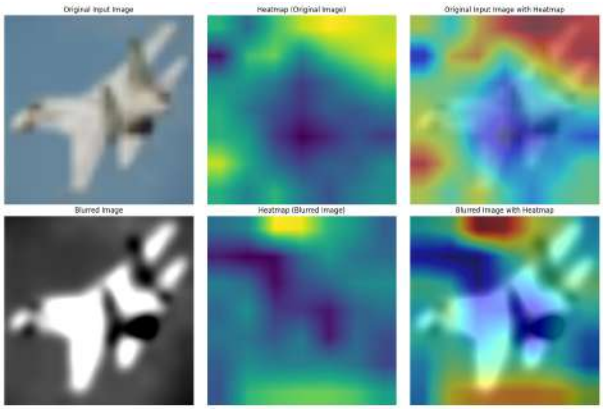
MSE of stage1 classifier for original images and gaussian blurred images.



Mean Squared Error (MSE) between original and noise-added heatmaps: 0.0036974384



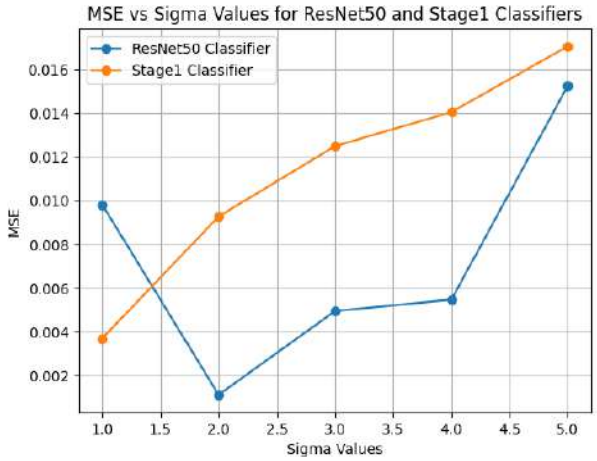
Mean Squared Error (MSE) between original and noise-added heatmaps: 0.06707073(sigma=5), 0.051272012(sigma=1)



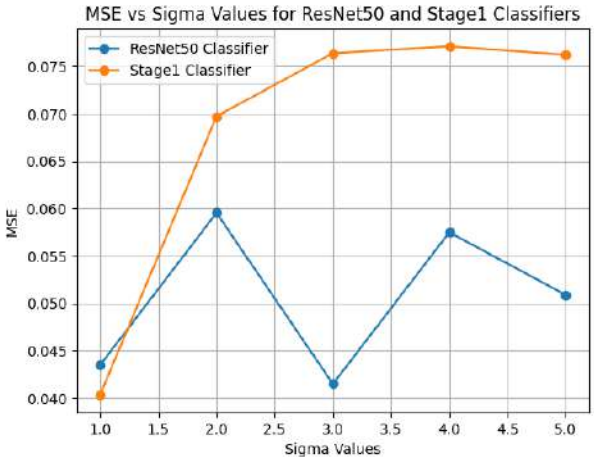
Mean Squared Error (MSE) between original and noise-added heatmaps: 0.06757125(sigma=5), 0.018898493(sigma=1)

MSE comparison table and sample plots

Resnet50 classifier (1 sample image)	Average of 10 sample image	Stage1 classifier 1 sample image)	Average of 10 sample image	Blurred Image (Sigma)
0.009812842	0.043467145	0.0036974384	0.04032632	1
0.001115551	0.059549164	0.009270788	0.06973738	2
0.0049486044	0.041469824	0.012488966	0.076413095	3
0.0054690046	0.057487838	0.014047238	0.077142	4
0.015253062	0.050880384	0.017057262	0.07624392	5



1 sample images



Average of 10 sample images

Note: By this MSE comparisons stage1 classifier has considerably lower values sigma =1

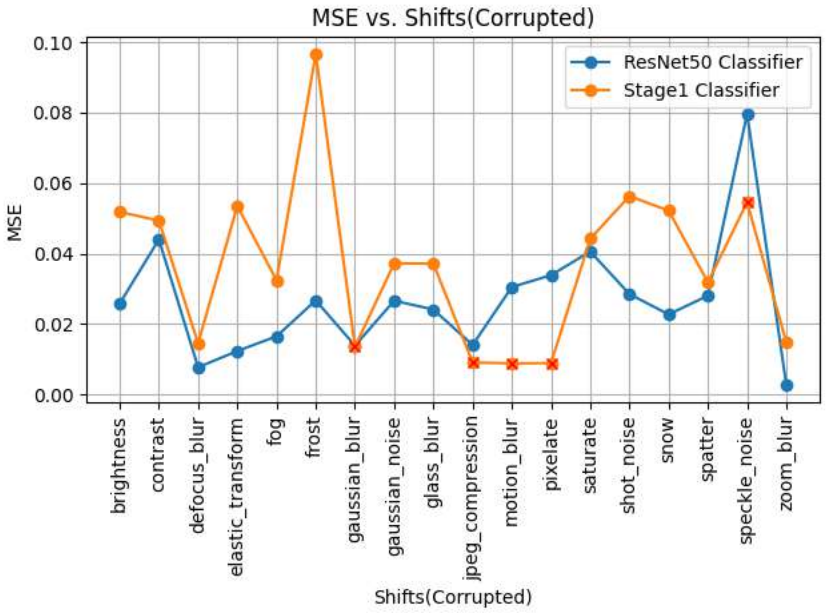
MSE comparison of CIFAR10 test data sample with corresponding corrupted shifts(18) in CIFAR10-C dataset.

Resnet50 classifier

MSE values: [0, 0.025807144, 0.04400859, 0.007730335, 0.012353547, 0.016524604, 0.026704894, 0.013869467, 0.026641123, 0.024115616, 0.01407222, 0.030497601, 0.03387049, 0.040579945, 0.028491583, 0.022739902, 0.028059667, 0.07960845, 0.0026819287]

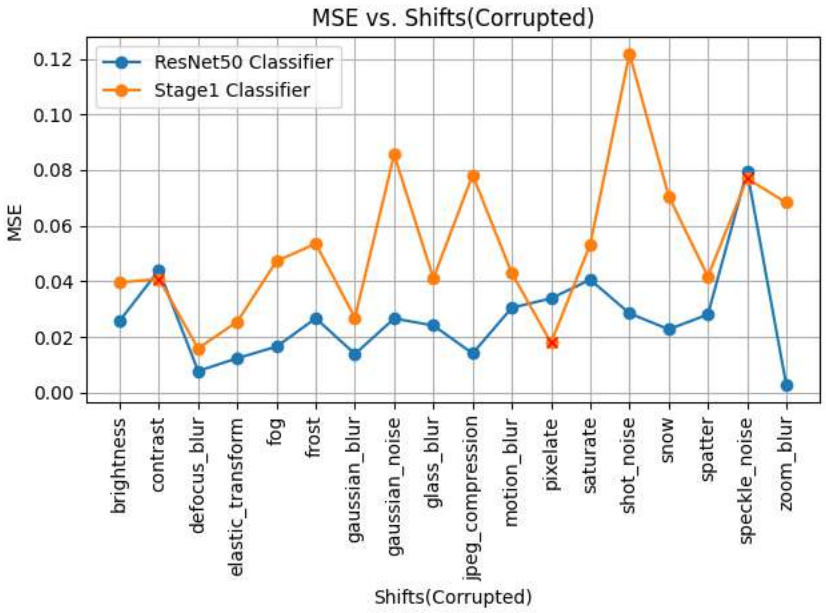
Stage1 Classifier

MSE values: [0, 0.051790092, 0.049356062, 0.01456873, 0.0534917, 0.03243517, 0.09670109, 0.013550751, 0.037234526, 0.037146006, 0.009112769, 0.008850927, 0.008933527, 0.044212297, 0.056300025, 0.052170303, 0.0318619, 0.05453119, 0.014804424]



Note: By this MSE comparisons stage1 classifier has considerably lower values for the shifts of ***motion_blur, gaussian_noise, jpeg_compression, pixelate and speckle_noise.***

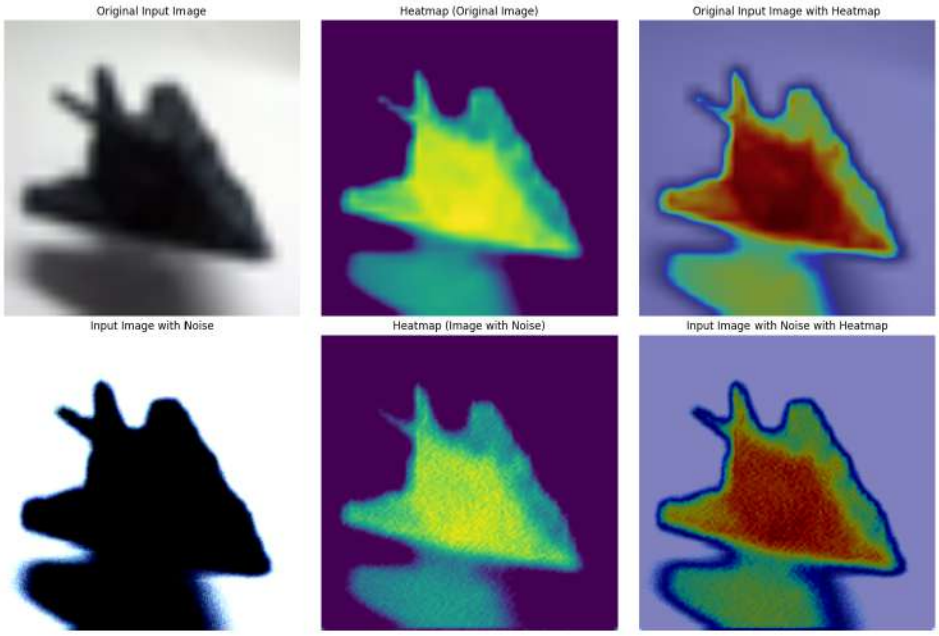
MSE plots considering best_stage1_classifier_tracher.pt model



Note: stage1_classifier.pt, best_stage1_classifier_tracher.pt considerable good visualization(but less compared with resnet50)

best_stage1_classifer.pt comparatively poor visualization.

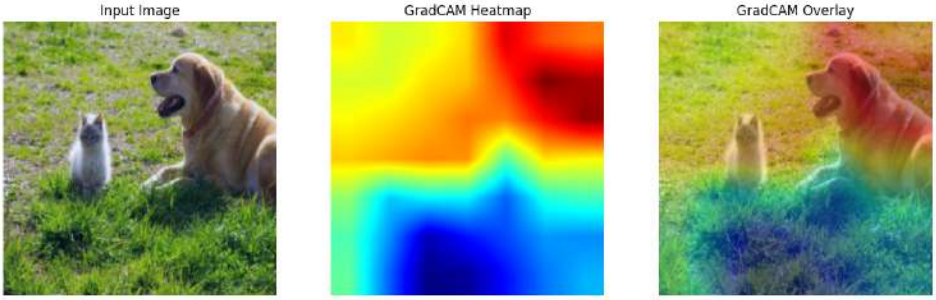
MSE of resnet50 classifier for original and gaussian noise images.



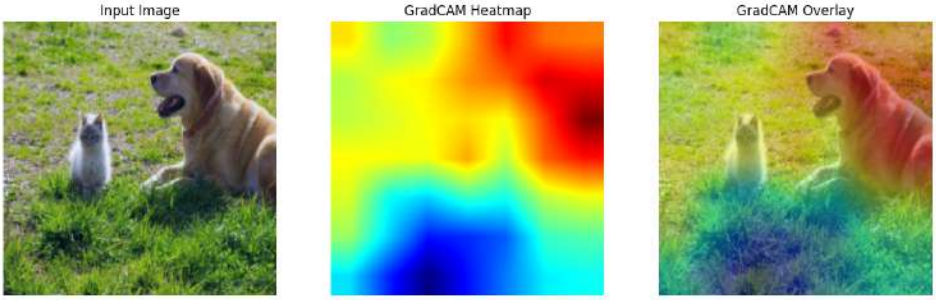
Mean Squared Error (MSE) between original and noise-added heatmaps: 0.0026332627

Issue!!!!

Considering “cats” as text input



Considering “dogs” as text input



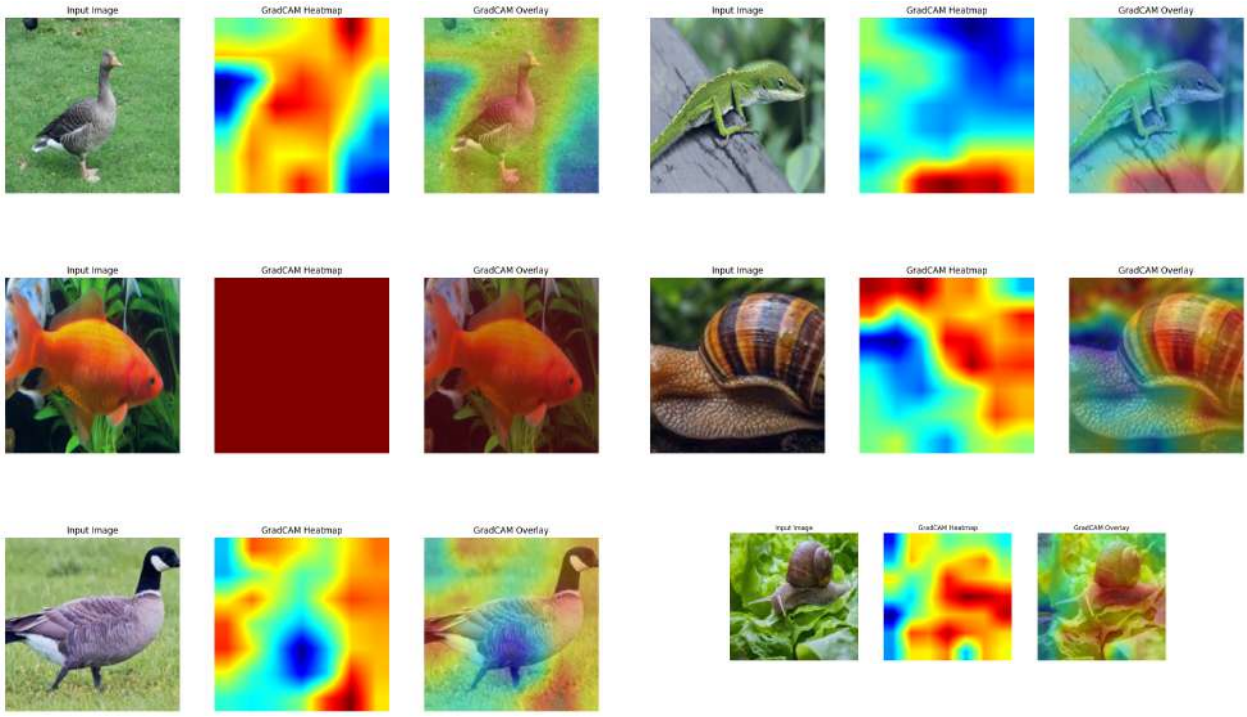
To Do

Need to integrate with best stage1 model for visualization

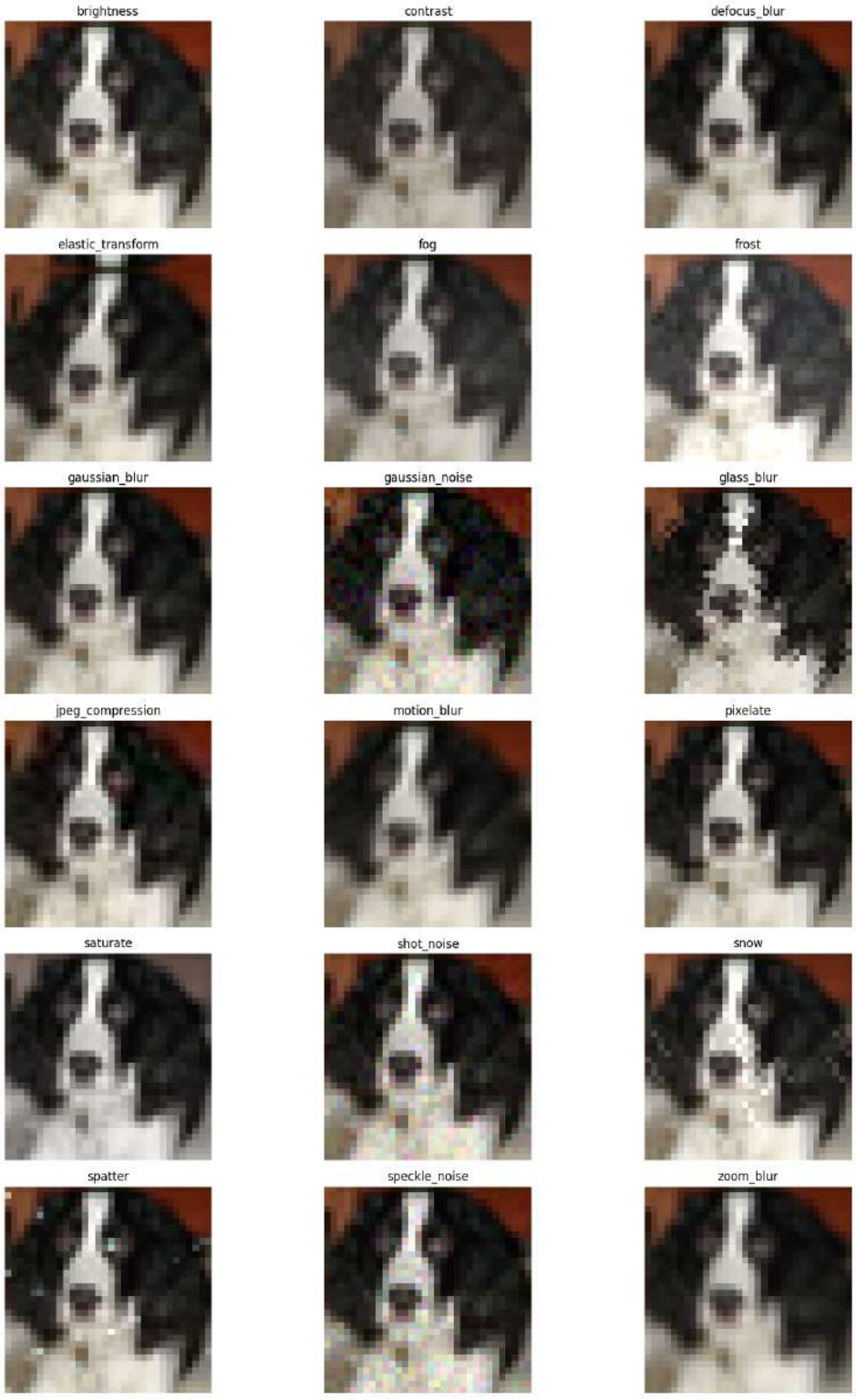
figure out issue on text name: cat no output, but cats presence of output visualization.

Grad CAM visualization with cifar10 trained stage1_classifier.

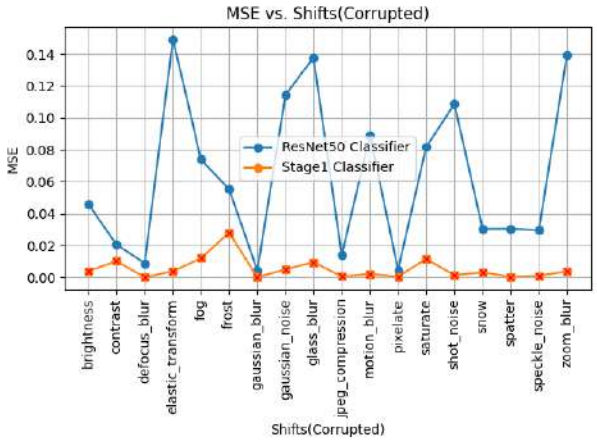
Grad CAM visualization with imagenet100 trained with best stage1_classifier.



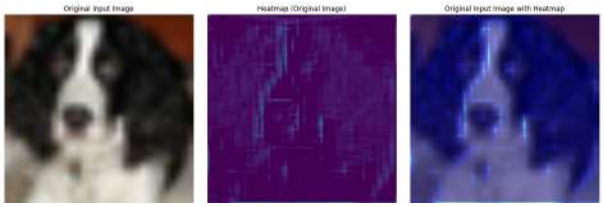
Important: Good results for specific class (Dog) - stage1_classifer.pt



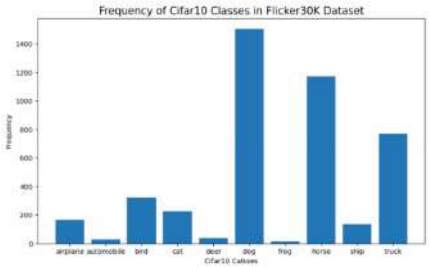
Reason: Training set contain specific class names in higher count.



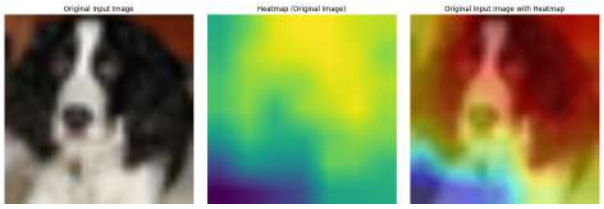
Original Image and heatmap



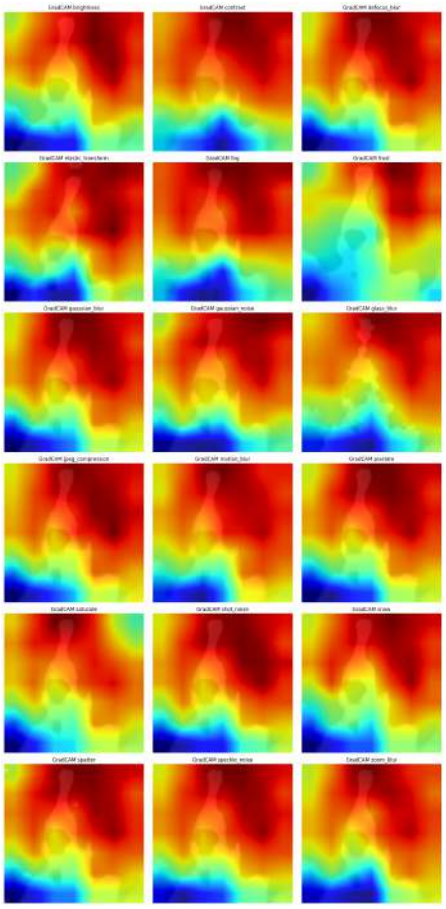
Heat maps of Corrupted versions.

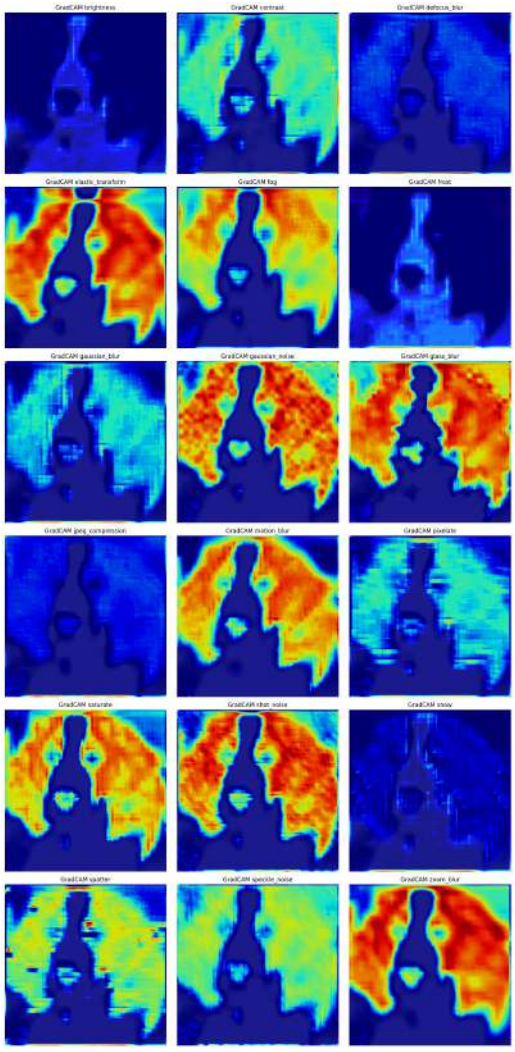


Original Image and heatmap



Heat maps of Corrupted versions.



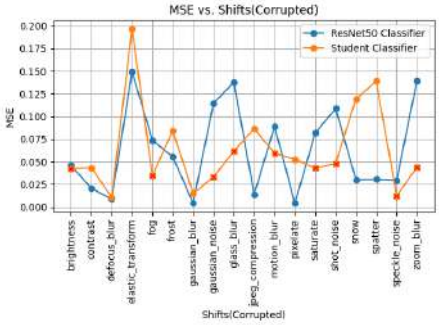


Correct Predicted Class: 5 (1/18)

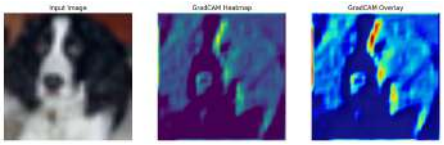
Grad CAM Visualization with student model

Correct Predicted Class: 5 (18/18)

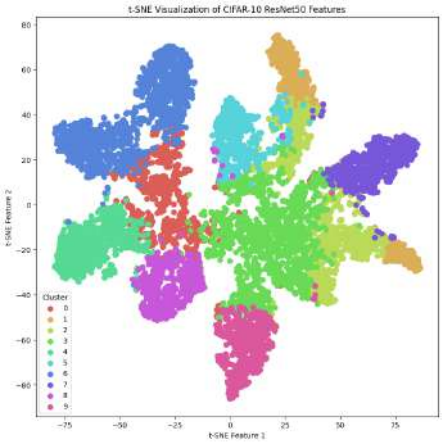
Student Classifier MSE comparisons.

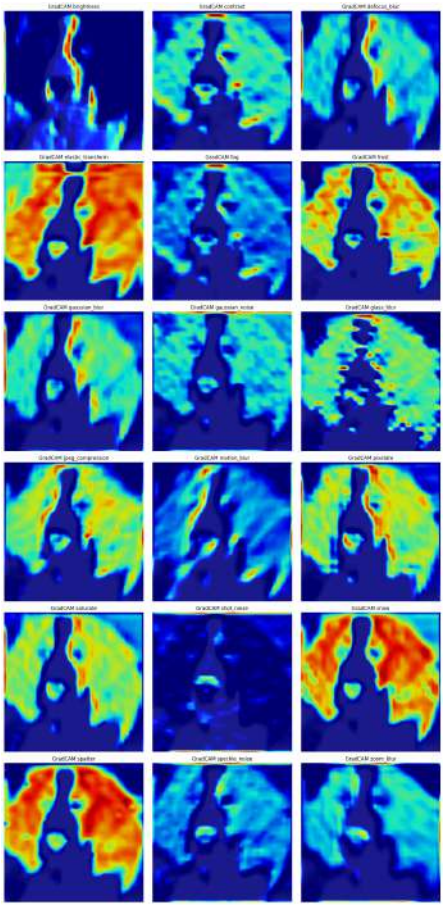


Grad CAM visualization of original input image via student classifier.



Feature separation by cifar0 class

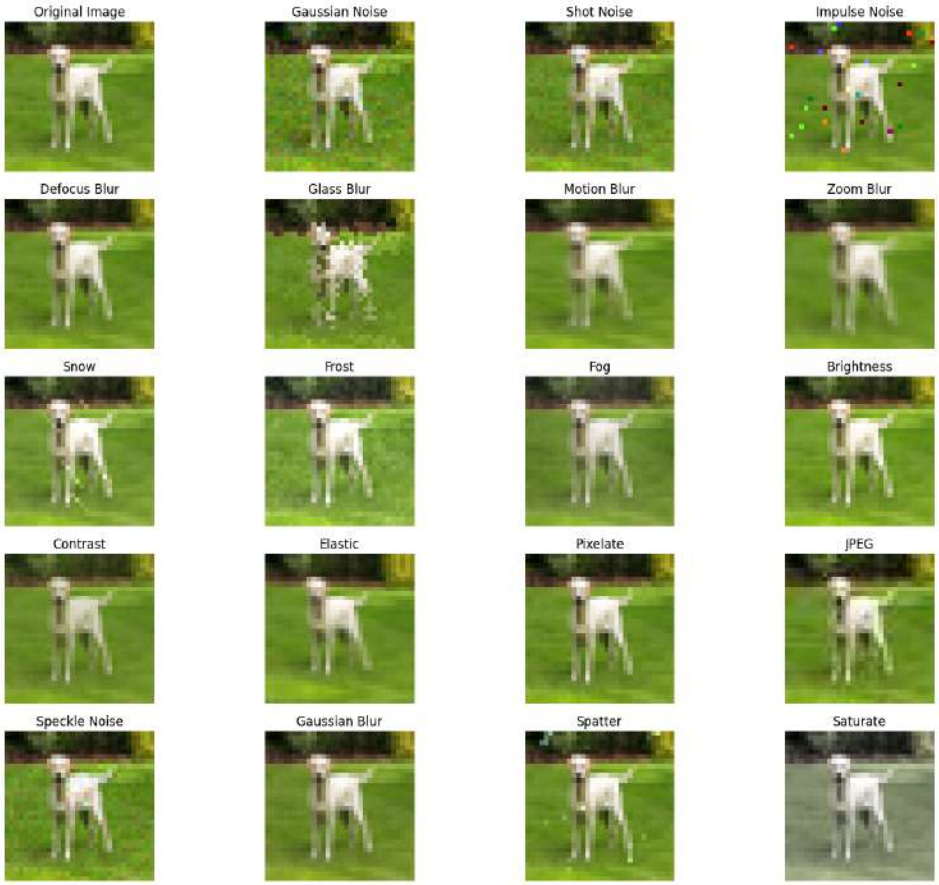




Correct class prediction : 5 (18/18)

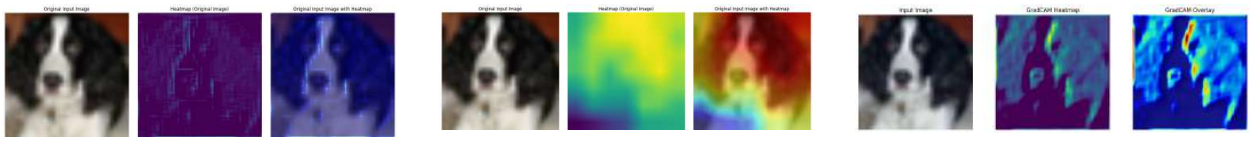
Student classifier test image visualizations (Correct prediction)

Code implementation for generating Corrupted versions of CIFAR10 effects for sample input image.

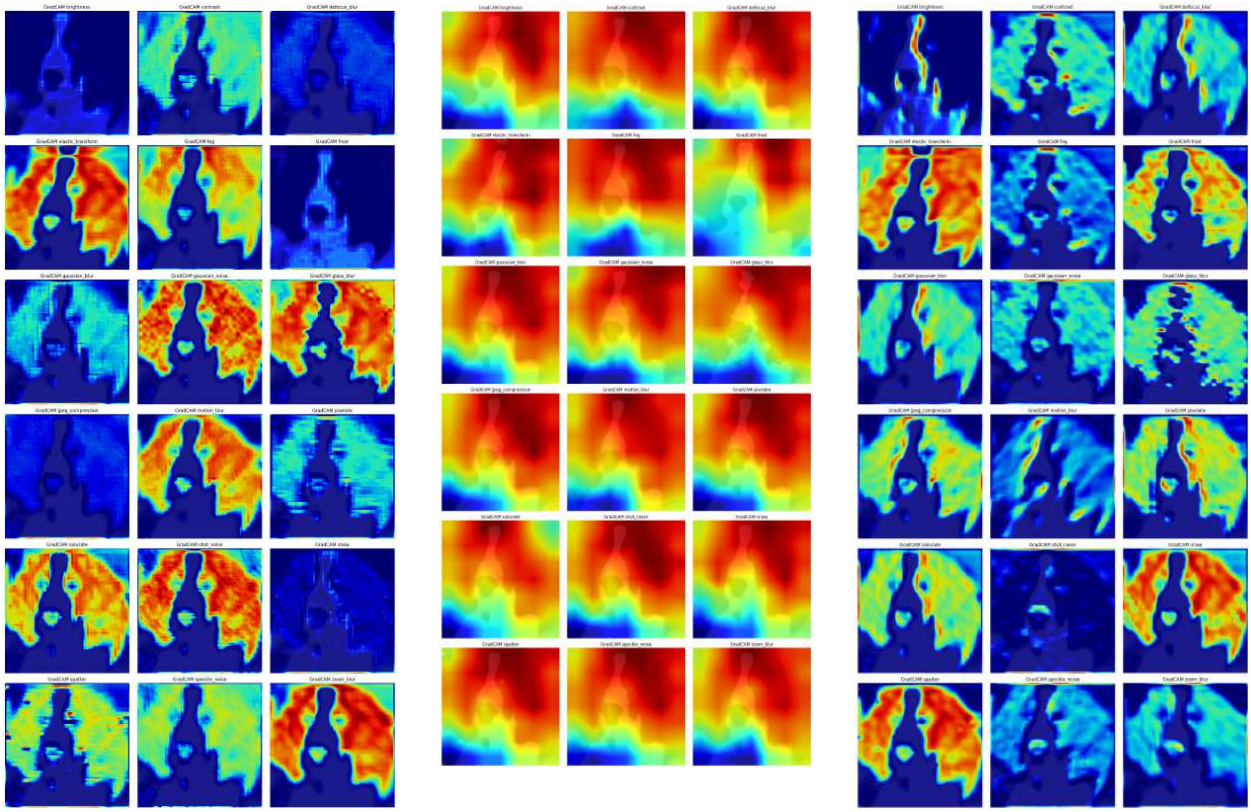


Final Comparisons Resnet50, Teacher model, Student model

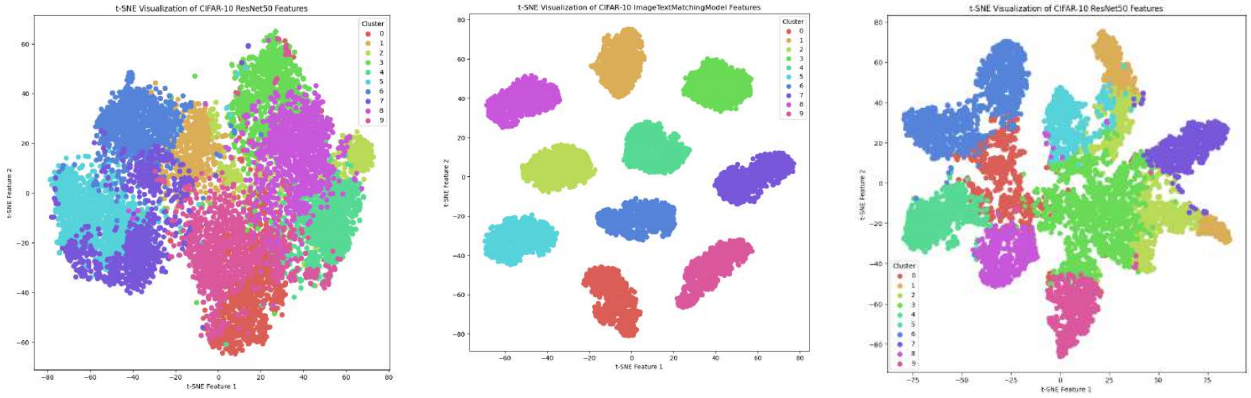
Original Input Image



Corrupted image Grad CAM Visualization

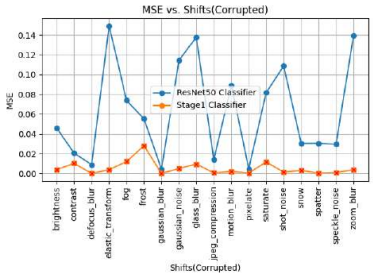


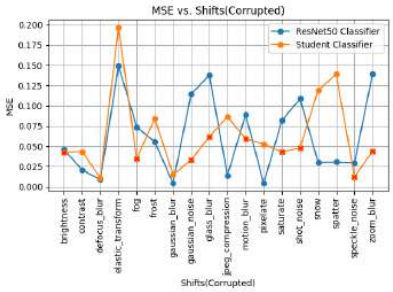
Feature Separation by CIFAR10 Classes



Heatmaps MSE Comparisons

Reference to Resnet50





Evaluation for OOD Generalization

Table 1. Commonly used image datasets for OOD generalization. Shift type denotes the type of distributional shifts, and the mixed type in image type means that there are both real and unreal images.

Image Data	**ImageNet-Variant** [100, 97, 99]	**Colored MNIST** [11]	**MNIST-R** [80]	**Waterbirds** [221]	**Camelyon17** [15]	**VLCS** [66]	**PACS** [143]
# Domains	-	3	6	2	5	4	4
# Categories	-	2	10	2	2	5	7
# Examples	-	-	6k	4.8k	450k	2.8k	9.99k
Shift Type	Adversarial Policy	Color	Angle	Background	Hospital	Data Source	Style
Image Type	Mixed Type	Digits	Digits	Birds	Tissue Slides	Real Objects	Mixed Type
Image Data	**Office-Home** [252]	**DomainNet** [199]	**iWildCam** [16]	**FMoW** [40]	**PovertyMap** [288]	**NICO** [94]	**NICO++** [299]
# Domains	4	6	323	16 × 5	23 × 2	188	810
# Categories	65	345	182	62	Real Value	19	80
# Examples	15.5k	570k	200k	500k	20k	25k	230k
Shift Type	Style	Style	Location	Time, Location	Country, Urban/Rural	Background, Attribute, Action, View and Co-occurring Object	
Image Type	Mixed Type	Mixed Type	Real Animals	Satellite	Satellite	Real Objects	

Future works:

- Try with finetuning approach for stage1
- Stage1 Evaluation using different OOD dataset
- Visualizing intermediate features
- Try with transfer learning for stage2 classification
- Improve stage2 accuracy by modify the classifier
- Approach for handling semantic shifts

Doubt: how we can validate that our methodology works for capturing invariant features specially for covariate shifts.(Also need to ensure that invariant feature extractions works correctly for semantic shifts).