# Image and Vision Computing

**s1950427** **s2450850**

## 1 Classifier Implementations

### 1.1 Classical Classifier: SVM with HoG features

#### 1.1.1 HoG features

Histogram of oriented Gradients ("HoG") is a classical computer vision feature descriptor that is broadly used for object detection purposes [3]. The HoG method divides an image into smaller, non-overlapping cells and calculates the direction and magnitude of the gradient of each pixel inside a cell. A histogram is then created for each cell by binning the gradient information into a number of bins. Each of these bins stores the sum of the magnitudes of the gradients and their corresponding directions in a specific range. In order to minimize the effect of shadows or high light concentrations, a number of cells are grouped into blocks. The results corresponding to each cell are then normalized using the data of all the cells inside a block.

To retrieve the HoG features for each image, the function **skimage.feature.hog()** was used. All images must have the same number of features and should be easily divisible into cells and blocks. Therefore, each image was converted to gray scale and resized to 112 x 112. Each cell was set to contain 16 x 16 pixels and each block 2 x 2 cells. The block - normalisation method used was "L2-Hys" which utilizes L2-norm and limits the maximum values to 0.2. Finally, the number of orientations is set to 30 such that each histogram contains 30 bins, each containing data for a gradient direction range of 6 degrees. In this way, we ended up with a feature descriptor of [30 bins per block * 12 blocks vertically * 12 blocks horizontally =] 4320 features.

#### 1.1.2 SVM classifier

We selected the Support Vector Machine classifier (SVM) due to its efficiency in high dimensional space and ability to handle multi-class classification. The **sklearn.svm.SVC()** method was used for our model with a grid search in order to tune hyper-parameters. This was done using the **sklearn..model_selection.GridSearchCV()** function. The optimal parameters were found to be kernel="rbf" with gamma=0.1 and regularization strength C=10. The model was trained using the train data set with 5-fold cross validation to validate the performance. The validation process used the average of the cross validation function of sklearn, **sklearn.model_selection.cross_val_score**(), with the "scoring" parameter set to the weighted F1 score in order to account for label imbalance. We saved the best version of the model to our machine and then used it to predict the classification labels of the test set. This resulted in an F1 score of 0.2230. Note that the test images are also resized to 112 x 112 and converted to gray-scale.

### 1.2 Deep Learning Classifier: ResNet-18

#### 1.2.1 Model Architecture

We consider ResNet-18 for the task of image classification using transfer learning. Pre-trained neural networks are able to classify images based on the class labels assigned to the training set.

The pre-trained ResNet-18 model is a residual network with a 72-layer architecture that includes 18 deep layers. The network architecture consists of five convolutional layers, a pooling layer, a fully connected layer and a softmax layer [2]. The convolutional layers in the network apply a filter that scans over the pre-processed images to create a feature map that is used in prediction. The first convolutional layer learns low level features such as edges and colour patches. Higher level features such as facial parts are learned in the deeper layers. The pooling layer reduces the spatial size of feature representations. Robustness to the exact spatial location of features is gained here. The output from the pooling layer is flattened and fed into the fully connected layer. Finally the softmax layer limits the output to the range [0,1].

The model aims to allow multiple convolutional layers to function efficiently whilst avoiding degradation of the output i.e. it aims to tackle the problem of vanishing gradient [1]. During backpropagation in training, a neural network's weights are updated proportionally to the partial derivative of the error function with respect to the current weight. In the presence of multiple layers, the gradient can become vanishingly small which leads to the saturation or degrading of the network performance.

### 1.2.2 Methods

The train data set was split into train and validation sets with 80% and 20% ratio, respectively. The validation data set provided was retained as a holdout set. All images are resized to the required input size of 224 x 224 x 3 where the depth of 3 represents RGB channels. Each channel must be normalised by subtracting the mean from each pixel and dividing by the standard deviation using the predefined values. To improve model generalisability, we pre-augment the images in the training set by applying random horizontal flips and random rotations. This synthesises new training examples by exploiting the idea that class labels should be invariant to some kinds of transformations. All images are finally converted into tensors and normalised as above.

To classify the images, we retrain a pre-trained ResNet-18 network by freezing the convolutional layers and updating the fully connected layers according to the given data set. In this case, the output features, originally 1000 in size, are adjusted to 8, corresponding to the 8 classes into which we wish to classify images. The stochastic gradient descent optimizer (SGD) is used during training due to its generalisation performance on image classification tasks. The model performance per epoch was tested on the validation set and measured using the weighted average of F1 scores. The validation set scores are used as an estimate of the model's generalisation performance on the test set. Initially, training was performed up to 30 epochs, but this was reduced to 25 because the model performance plateaued. We save the model weights with the highest validation accuracy.

To determine optimal hyper-parameters, we train the network on mini-batch sizes of 32, 64 and 128, adapting the learning rates accordingly. We find that a batch size of 64 with a learning rate of 0.0005 results in the highest overall performance. The saved model weights are used to predict the classification labels of the images in the test set resulting in an F1 score of 0.4954.

## 2 Algorithms used to perturb test images

Below are the descriptions of the algorithms used to perturb the test images. A function was created for every perturbation type and the code for the algorithms can be found in Appendix 4.1.

(a) **Gaussian Pixel Noise:** An array is defined with the same dimensions as each of the images and is filled with random normal values using **np.random.normal()**. These values represent "noise" and are defined using the specified standard deviations and a mean of 0. The noise array is added element-wise to an image array to create a noisy image array. Then, using a for-loop, the algorithm checks if any pixel value is more than 255 (white) or less than 0 (black) and sets those values to 255 and 0, respectively. The result of this is a noisy image array. Note that the length of the shape of an input image returns the image dimension i.e. 2-dimensional or 3-dimensional. This value is used to determine whether an image is in gray-scale (2-dimensional array) or "rgb" (3-dimensional array) such that it may be fed through the corresponding for-loop. If the input image is in "rgb" form, the for-loop iterates through all 3 layers of the image's pixel array.

(b) **Gaussian Blurring:** At the beginning of the algorithm, an array representing the mask $\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$ is created. We use the **signal.scipy.convolve2d()** function to convolve each image with this mask. The ’mode’ argument is set to ’same’ so that the output array has the same size as the input image. Using a for-loop, the image is convoluted the required number of times. As before, the length of the shape of the input image is used to determine whether the image is in "rgb" form (3-dimensional array) or otherwise such that the corresponding for-loop can be used. In the case of an "rgb" image, the mask must be used on all 3 layers. The convoluted image is then returned.

(c) **Image Contrast Change:** Each input image array is multiplied element-wise by a specified scaling factor. Then, using an if statement, we check if the required scaling factor is greater than 1 (contrast increase), or less than or equal to 1 (contrast decrease or remains the same). If the factor is greater than 1, a for-loop is used to check whether any of the new pixel values in the image array have a value that is greater than 255. If so, those pixel values are set to 255. The scaled image array is then returned. On the other hand, if the scaling factor is less than or equal to 1, a check is not performed. This is because the new pixel values will only decrease or remain constant and cannot become negative since we are multiplying by a non-negative number. As before, the length of the shape of the input image is used to determine if the image is in gray-scale (2-dimensional array) or in "rgb" form (3-dimensional array) such that the corresponding for-loop may be used to check pixel values.

(d) **Image Brightness Change**: Using an if statement, we check if the required change is positive (brightness increase), negative (brightness decrease), or 0. The change is then added element-wise to the input image array. If the change is positive, we use a for-loop to check if any of the new pixel values are greater than 255. In such a case, those values are set to 255 and the new image array is returned. Similarly, if the change is negative we check if any of the pixel values are less than 0, set those to zero and return the new image array. If the change is zero, we simply return the input image. As before, the length of the shape of the input image is used to determine whether the image is in gray-scale (2-dimensional array) or in "rgb" form (3-dimensional array) such that the corresponding for-loop may be used to check pixel values.

(e) **Occlusion:** First, we check the input image dimensions in order to determine the number of rows and columns in the array. The **random.randint()** function is then used to define a random integer from 0 to the number of rows - 1 - the required edge length of square. This is done to ensure that the square of pixels created is within the range of the image array. The defined random integer specifies the row on which the occlusion square will begin. The same process is repeated for columns such that we also define the column on which the occlusion square is to begin. The area to be occluded is then all pixels from the random row to the random row + the square edge length and all pixels from the random column to the random column + the square edge length. All pixel values within that area are set to 0 (black) in order to create an occlusion square. The image array is then returned.

(f) **Salt and Pepper Noise:** This algorithm uses the **skimage.util.random_noise()** function with the argument ’mode’ set to ’s&p’ and the argument ’amount’ set to the required strength of the salt and pepper noise. Since all pixel values are between 0 and 1, we multiply the pixel values by 255 and return the changed image array.

In order to carry out the robustness exploration for both classifiers, we created a directory containing all of the perturbed images.

## 3 Robustness Exploration

We show the effect of a variety of perturbations on the performance of our classical classifier and neural network. The perturbations applied are as follows: (a) Gaussian pixel noise with standard deviations 0, 2, 4, 6, 8, 10, 12, 14 , 16, 18, (b) convolution of image with a Gaussian blurring mask of dimension 3x3, (c) image contrast increase by multiplying each pixel by 1.0, 1.01, 1.02, 1.03, 1.04, 1.05, 1.1, 1.15, (d) image contrast decrease by multiplying each pixel by 1.0, 0.95, 0.90, 0.85, 0.80, 0.60, 0.40, 0.30, 0.20, 0.10, (e) image brightness increase by adding 0, 5, 10, 15, 20, 25, 30, 35, 40, 45 to each pixel, (f) image brightness decrease by subtracting the aforementioned values from each pixel, (g) random occlusion of image by black pixel square of edge lengths 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, (h) salt and pepper noise of increasing strength with values 0.00, 0.02, 0.04, 0.06, 0.08, 0.10, 0.12, 0.14, 0.16, 0.18.

### 3.1 Gaussian Pixel Noise

Figure 1: Examples of different levels of Gaussian pixel noise

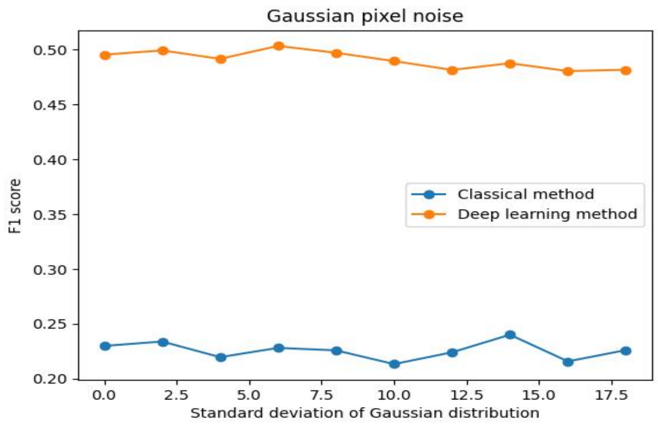

Figure 2: Performance of classical and neural network classifiers with respect to increasing levels of Gaussian pixel noise. The performance metric used is F1 score.

In figure 2, we observe that both classical and deep learning classifiers remain generally robust to the addition of pixel noise. We had expected that HOG features would result in a decrease in performance due to the adverse effects of increasing noise on edges. However, we find that the SVM with HOG features shows little variation in performance over the range of standard deviations with a small increase in performance at standard deviations of 12 and 14. As for ResNet-18, increased levels of noise can reduce the clarity of features in the image therefore a neural network that has not been trained on noisy images is expected to struggle. Although we observe a slight decrease in performance for standard deviations in the range 6-12, the overall performance of the neural network appears robust to Gaussian noise perturbations.

### 3.2 Gaussian Blurring

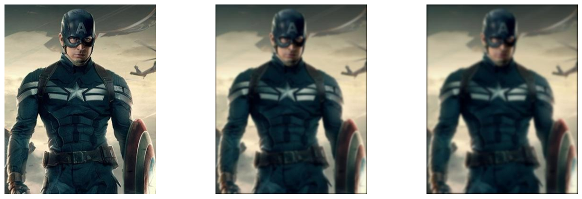
Figure 3: Examples of different levels of Gaussian blurring

Both the SVM and standard pre-trained network in figure 4 suffer degradation in performance when applied to increasingly blurred images. Both show a slightly more significant decrease following the second convolution with the Gaussian mask. The SVM with HOG features was trained on sharp images that included significant horizontal and vertical edges. The addition of blurring decreases the appearance of these edges thus making it difficult for the classical classifier to discern the relevant features it learned during training.

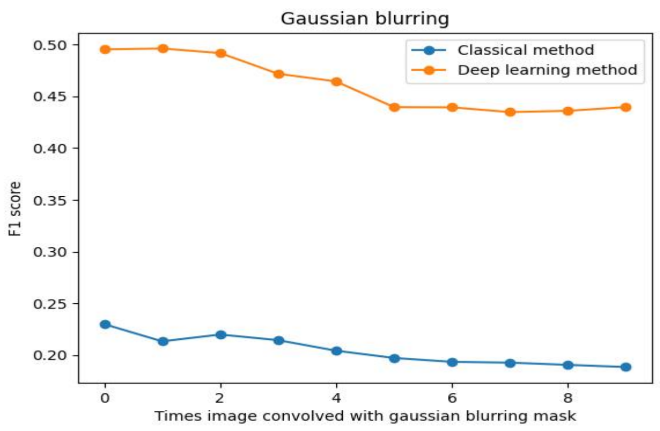

Figure 4: Performance of classical and neural network classifiers with respect to increasing levels of Gaussian blurring. The performance metric used is F1 score.

Increased blurring also results in a loss of information such as textures or facial features. The neural network may be looking for such features in order to make a prediction therefore, without their presence, it would struggle to classify an image successfully. It's reduction in performance could also be a result of having only trained the network on sharp images. This could be avoided if it were fine-tuned with a variety of blurry images in order to learn blur invariance.

### 3.3 Image Contrast Increase

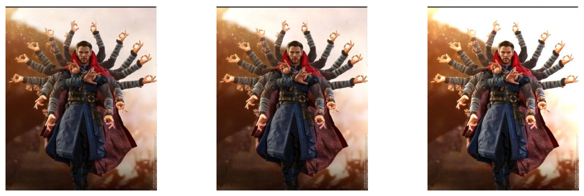

Figure 5: Examples of different levels of image contrast increase

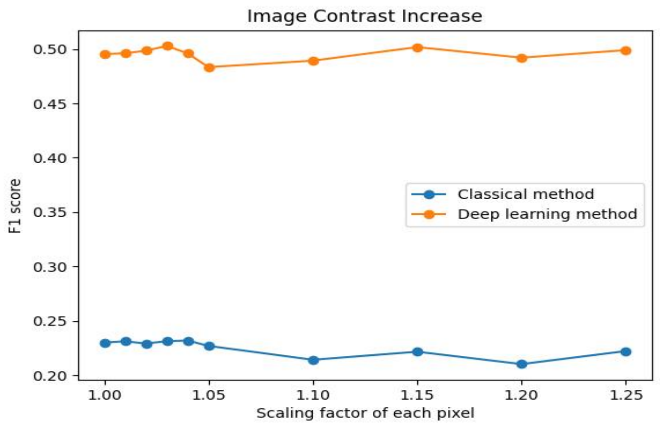

Figure 6: Performance of classical and neural network classifiers with respect to increasing image contrast. The performance metric used is F1 score.

The SVM and neural network remain fairly robust to an increase in image contrast as depicted in figure 6. The increase in contrast may have had little impact on the original content of the image such that no vital information about the features was lost. This means that each classifier would be able to use the information it learnt while training on the unperturbed images to make accurate predictions about those that were perturbed.

### 3.4 Image Contrast Decrease

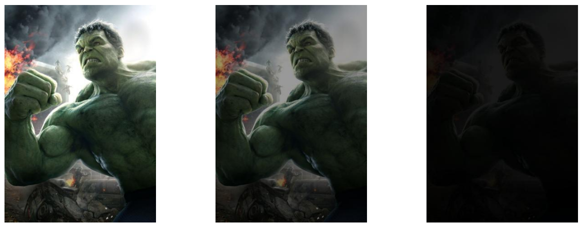

Figure 7: Examples of different levels of image contrast decrease

Note that in figure 8, the unperturbed performance of the classifiers are at a value of 1 which is the final value plotted in the figure.

We observe a significant degradation in the performance of ResNet-18 with decreasing contrast. It is with this type of perturbation that the neural network performance suffers the most. Similarly to blurring, the reduction of image contrast means that features become less distinct. A network that has not been trained to recognise low contrast images may struggle to discern the features it learnt while training on the original images. The SVM with HOG features on the other hand proves to be robust to such a perturbation up until the final scaling factor of 0.1. It is possible that HOG performs well even in the case of low contrast images because significant edges survive the transformation but, following a large decrease in contrast, a degradation in performance is to be expected. These results are unlike the result for image contrast increase where the model performances remained robust. We note that for a scaling factor of 0.1, the performance of both models decreases to an F1 score of 0.125. This is the baseline performance of a classifier that is predicting labels at random.

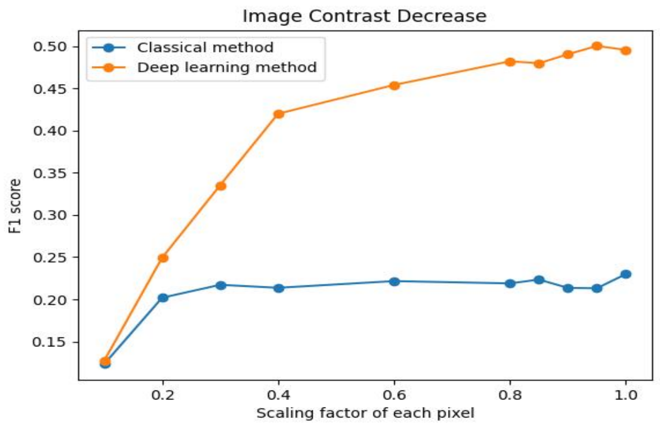

Figure 8: Performance of classical and neural network classifiers with respect to decreasing image contrast. The performance metric used is F1 score.

### 3.5 Image Brightness Increase

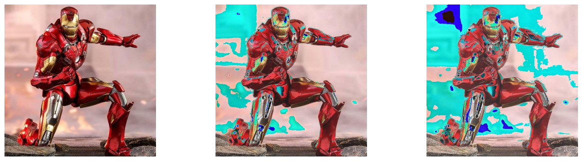
Figure 9: Examples of different levels of image brightness increase

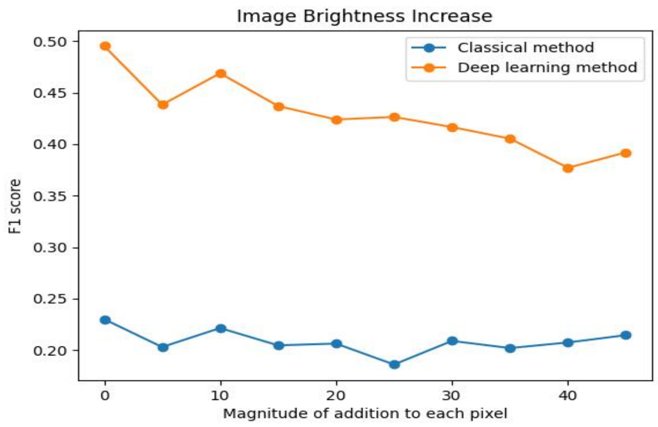

Figure 10: Performance of classical and neural network classifiers with respect to increasing image brightness. The performance metric used is F1 score.

In figure 10 we see that the SVM with HOG features generally performs well across the range of perturbations with its largest drop in performance for a value of 25 added to each pixel. Conversely, the neural network's performance gradually decreases with increasing image brightness. There is, however, an increase in F1 score for values of 10 and 45 added to each pixel. Increasing the image brightness affects the pixel intensities of each colour channel in an image meaning that the features may appear washed out. Increasing the diversity of brightness in the training images may help alleviate the reduction in performance.

### 3.6 Image Brightness Decrease

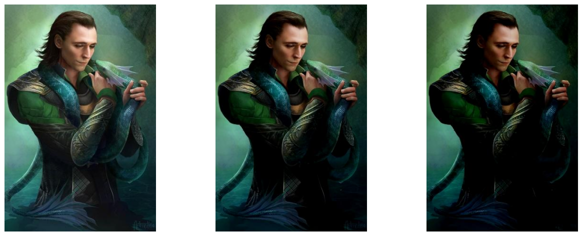
Figure 11: Examples of different levels of image brightness decrease

In figure 12 SVM with HOG features shows a small decrease in performance for values of magnitudes 0, 5, 10 and 30. Apart from this, the classifier shows only slight shifts in performance within the range of 0.2-0.25 over increasing levels of perturbation. The network's behaviour in the case of image brightness decrease is similar to that of image brightness increase i.e. there is a general downward trend in performance. We do however notice that an increase in brightness has a larger impact on performance than that of a decrease in brightness.

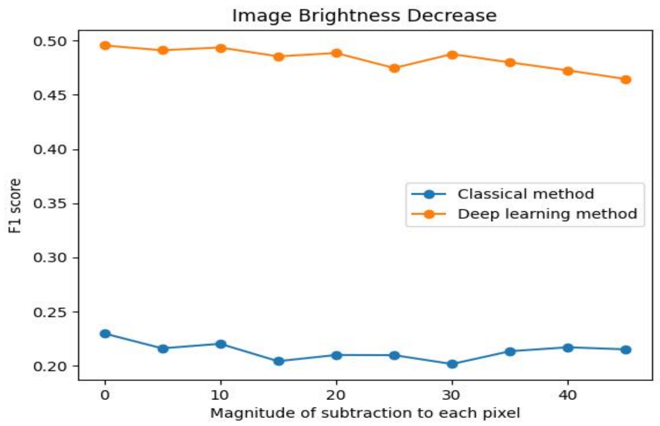

Figure 12: Performance of classical and neural network classifiers with respect to decreasing image brightness. The performance metric used is F1 score.

In the extreme case, a change in brightness can entirely change the appearance of an image: a black image is produced if the brightness is set to a low enough value. This image will no longer represent the original content and the classifiers, having trained on the original data, will not be able to accurately classify the perturbed image.

## 3.7 Occlusion

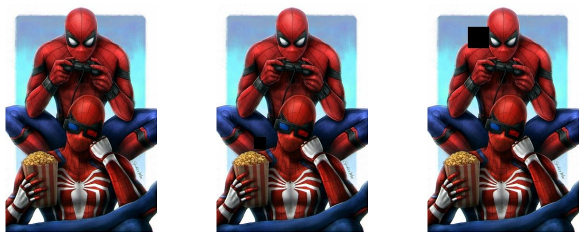

Figure 13: Examples of different levels of occlusion

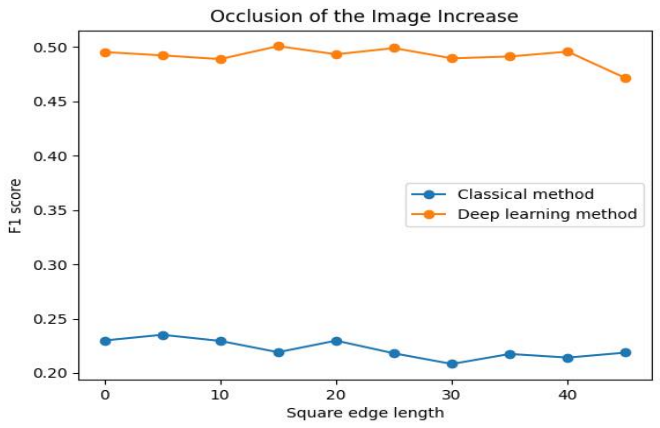

Figure 14: Performance of classical and neural network classifiers with respect to increasing image occlusion. The performance metric used is F1 score.

Like a number of other perturbation types, occlusion can result in a loss of information. Features once present in training images may no longer be present in the test images. In our case, occlusion seems to have little impact on the overall performance of both the SVM and the neural network. Although the size of the occlusion increases, the performances seems only to fluctuate about values of 0.50 and 0.23 for the neural network and SVM, respectively. The most significant decrease in the performance of the neural network occurs for a square of edge length 45. As for the SVM, the poorest performance is achieved for a square edge length of 30. We expect that the larger the occluded area, the more significant the impact on performance. However, since the square of black pixels is randomly placed within the image (which has dimensions 224 x 224), it is possible that features essential to the classification of each image were left unobscured in multiple cases.

### 3.8 Salt and Pepper Noise

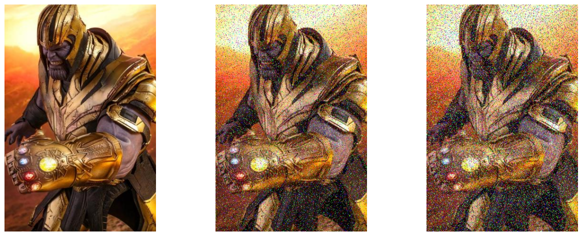

Figure 15: Examples of different levels of salt and pepper noise

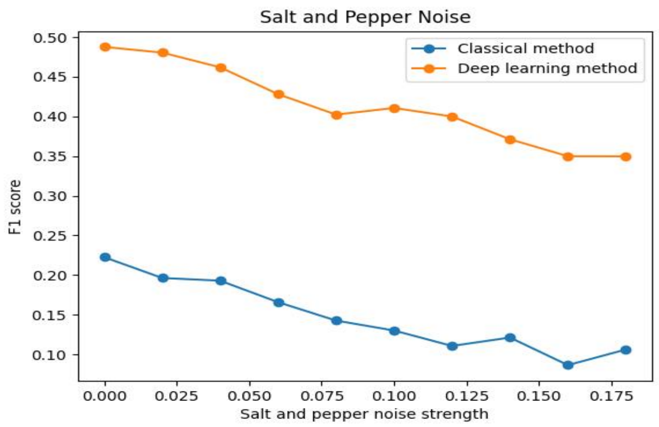

Figure 16: Performance of classical and neural network classifiers with respect to increasing salt and pepper noise. The performance metric used is F1 score.

It is clear that both the SVM's and neural network's performance suffers with the addition of salt and pepper noise. This type of noise introduces sparse but intense disturbances that degrade image quality thus reducing overall classification performance.

### 3.9 Conclusions

It is evident that both classical and deep learning models are susceptible to image perturbations. Degradation of input image quality is consistently one of the prime reasons for poor performance. However, we find that our models prove to be generally robust under most instances of image perturbation. In particular, the classical model appears more robust than the deep learning model for perturbation types like image brightness increase or decrease and small image contrast decrease. Nevertheless, the deep learning model shows better robustness overall. Providing the neural network with further perturbed examples to train on could significantly boost robustness thus unlocking its true potential.

## Citations and References

## References

[1] Yashvi Chandola, Jitendra Virmani, H.S. Bhadauria, and Papendra Kumar. Chapter 4 - end-to-end pre-trained cnn-based computer-aided classification system design for chest radiographs. In Yashvi Chandola, Jitendra Virmani, H.S. Bhadauria, and Papendra Kumar, editors, *Deep Learning for Chest Radiographs*, Primers in Biomedical Imaging Devices and Systems, pages 117–140. Academic Press, 2021.

[2] Paolo Napoletano, Flavio Piccoli, and Raimondo Schettini. Anomaly detection in nanofibrous materials by cnn-based self-similarity. *Sensors (Basel, Switzerland)*, 18, 01 2018.

[3] Mrinal Tyagi. Hog (histogram of oriented gradients): An overview, 2021. `https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f/"`.

## 4 Appendix

Below you can find all code used:

### 4.1 utils.py

This file contains helper functions as well as the perturbation functions created.

```
import numpy as np
from PIL import Image
import os
import glob
import random
from scipy.signal import convolve2d
from skimage.util import random_noise


# Some functions defined below were taken from drill exercise 3 of the course.
def read_img(path, mono=False):
    if mono:
        return read_img_mono(path)
    img = Image.open(path)
    return np.asarray(img)


def read_img_mono(path):
    # The L flag converts it to 1 channel.
    img = Image.open(path).convert(mode="L")
    return np.asarray(img)


def resize_img(ndarray, size):
    # Parameter "size" is a 2-tuple (width, height).
    img = Image.fromarray(ndarray.clip(0, 255).astype(np.uint8))
    return np.asarray(img.resize(size))


def save_img(ndarray, path):
    Image.fromarray(ndarray.clip(0, 255).astype(np.uint8)).save(path)


# Used the number of images of the category with the least images, for both train and test
def get_image_paths(data_path, categories, num_train_per_cat=307, num_test_per_cat=54):
    '''
    This function returns lists containing the file path for each train
    and test image, as well as lists with the label of each train and
    test image.
    '''

    num_categories = len(categories)  # number of categories.

    train_image_paths = [None] * (num_categories * num_train_per_cat)
    test_image_paths = [None] * (num_categories * num_test_per_cat)

    train_labels = [None] * (num_categories * num_train_per_cat)
    test_labels = [None] * (num_categories * num_test_per_cat)

    for i, cat in enumerate(categories):
        images = glob.glob(os.path.join(data_path, 'train', cat, '*.jpg'))

        for j in range(num_train_per_cat):
            train_image_paths[i * num_train_per_cat + j] = images[j]
            train_labels[i * num_train_per_cat + j] = cat

        images = glob.glob(os.path.join(data_path, 'test', cat, '*.jpg'))
        for j in range(num_test_per_cat):
            test_image_paths[i * num_test_per_cat + j] = images[j]
            test_labels[i * num_test_per_cat + j] = cat
    return train_image_paths, test_image_paths, train_labels, test_labels


def get_test_image_paths(data_path, categories, num_test_per_cat=54):
    '''
    This function returns lists containing the file path for each test image,
    as well as lists with the label of each test image, for perturbation purposes.
    '''

    num_categories = len(categories)  # number of categories.

    test_image_paths = [None] * (num_categories * num_test_per_cat)

    test_labels = [None] * (num_categories * num_test_per_cat)

    for i, cat in enumerate(categories):
        images = glob.glob(os.path.join(data_path, cat, '*.jpg'))
        for j in range(num_test_per_cat):
            test_image_paths[i * num_test_per_cat + j] = images[j]
            test_labels[i * num_test_per_cat + j] = cat
    return test_image_paths, test_labels


# Helper functions for perturbations
def gaussian_pixel_noise(img, sd):
    shape = img.shape
    noise = np.random.normal(0, sd, shape)
    noised_image = img + noise
    if len(img.shape) == 2:
        grayscale = True
    else:
        grayscale = False

    if not grayscale:
        for i in range(shape[0]):
            for j in range(shape[1]):
                for k in range(shape[2]):
                    if noised_image[i][j][k] < 0:
                        noised_image[i][j][k] = 0
                    if noised_image[i][j][k] > 255:
                        noised_image[i][j][k] = 255
    else:
        for i in range(shape[0]):
            for j in range(shape[1]):
                if noised_image[i][j] < 0:
                    noised_image[i][j] = 0
                if noised_image[i][j] > 255:
                    noised_image[i][j] = 255
    return np.asarray(noised_image)


def gaussian_blurring(img, times):
    mask = [[1/16, 2/16, 1/16],
            [2/16, 4/16, 2/16],
            [1/16, 2/16, 1/16]]
    convolved_image = img.copy()

    if len(img.shape) == 2:
        grayscale = True
    else:
        grayscale = False

    if not grayscale:
        for i in range(times):
            convolved_image[:, :, 0] = convolve2d(convolved_image[:, :, 0], mask, mode='same')
            convolved_image[:, :, 1] = convolve2d(convolved_image[:, :, 1], mask, mode='same')
            convolved_image[:, :, 2] = convolve2d(convolved_image[:, :, 2], mask, mode='same')
    else:
        for i in range(times):
            convolved_image = convolve2d(convolved_image, mask, mode='same')

    return convolved_image


def image_contrast_change(img, scale):
    shape = img.shape
    changed_image = img * scale

    if len(img.shape) == 2:
        grayscale = True
    else:
        grayscale = False

    if not grayscale:
        if scale > 1:
            for i in range(shape[0]):
                for j in range(shape[1]):
                    for k in range(shape[2]):
                        if changed_image[i][j][k] > 255:
                            changed_image[i][j][k] = 255
    else:
        if scale > 1:
            for i in range(shape[0]):
                for j in range(shape[1]):
                    if changed_image[i][j] > 255:
                        changed_image[i][j] = 255
    return np.asarray(changed_image)


def image_brightness_change(img, change):

    if len(img.shape) == 2:
        grayscale = True
    else:
        grayscale = False

    if not grayscale:
        if change > 0:
            shape = img.shape
            changed_image = img + change
            for i in range(shape[0]):
                for j in range(shape[1]):
                    for k in range(shape[2]):
                        if changed_image[i][j][k] > 255:
                            changed_image[i][j][k] = 255
        elif change < 0:
            shape = img.shape
            changed_image = img + change
            for i in range(shape[0]):
                for j in range(shape[1]):
                    for k in range(shape[2]):
                        if changed_image[i][j][k] < 0:
                            changed_image[i][j][k] = 0
        else:
            changed_image = img
    else:
        if change > 0:
            shape = img.shape
            changed_image = img + change
            for i in range(shape[0]):
                for j in range(shape[1]):
                    if changed_image[i][j] > 255:
                        changed_image[i][j] = 255
        elif change < 0:
            shape = img.shape
            changed_image = img + change
            for i in range(shape[0]):
                for j in range(shape[1]):
                    if changed_image[i][j] < 0:
                        changed_image[i][j] = 0
        else:
            changed_image = img

    return np.asarray(changed_image)
```

```
def occlusion(img, length):
    shape = img.shape
    occluded_image = img.copy()
    random_row = random.randint(0, shape[0]-1-length)
    random_column = random.randint(0, shape[1]-1-length)
    occluded_image[random_row:random_row+length, random_column:random_column+length] = 0
    return np.asarray(occluded_image)


def salt_and_pepper_noise(img, strength):
    noised_img = random_noise(img, mode='s&p', amount=strength)
    return np.asarray(255*noised_img)
```

## 4.2 **save**$_{p}erturbations.py$

This file is used to create a directory of perturbed images

```
from utils import get_image_paths, read_img, gaussian_blurring, \
    gaussian_pixel_noise, image_contrast_change, image_brightness_change, occlusion, salt_and_pep
    save_img

# Create a folder named "perturbations" in data folder
# Save all perturbed images in that folder in sub folders
# each sub folder named from A to H represents a type of perturbation in the order written in Ass
# Inside each sub folder there are sub sub folders from 0 to 9 representing the parameter used in
# in the order written in Assignment.
# Each of these contain the 8 superhero folders with the corresponding perturbed images.

if __name__ == '__main__':

    DATA_PATH = 'data'
    IMAGE_CATEGORIES = [
        'black widow', 'captain america', 'doctor strange', 'hulk',
        'ironman', 'loki', 'spider-man', 'thanos'
    ]

    train_image_paths, test_image_paths, train_labels, test_labels = \
        get_image_paths(DATA_PATH, IMAGE_CATEGORIES)

    # Gaussian pixel noise
    par_list = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
    index = 0
    for i in par_list:
        for img_path in test_image_paths:
            img = read_img(img_path)
            img = gaussian_pixel_noise(img, i)
            save_img(img, "data/perturbations/A/" + str(index) + img_path[9:])
        index += 1

    # Gaussian blurring
    par_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
    index = 0
    for i in par_list:
        for img_path in test_image_paths:
            img = read_img(img_path)
            img = gaussian_blurring(img, i)
            save_img(img, "data/perturbations/B/" + str(index) + img_path[9:])
        index += 1
```

```
# Image contrast increase
par_list = [1, 1.01, 1.02, 1.03, 1.04, 1.05, 1.10, 1.15, 1.20, 1.25]
index = 0
for i in par_list:
    for img_path in test_image_paths:
        img = read_img(img_path)
        img = image_contrast_change(img, i)
        save_img(img, "data/perturbations/C/" + str(index) + img_path[9:])
    index += 1

# Image contrast decrease
par_list = [1, 0.95, 0.90, 0.85, 0.80, 0.60, 0.40, 0.30, 0.20, 0.10]
index = 0
for i in par_list:
    for img_path in test_image_paths:
        img = read_img(img_path)
        img = image_contrast_change(img, i)
        save_img(img, "data/perturbations/D/" + str(index) + img_path[9:])
    index += 1

# Image brightness increase
par_list = [0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
index = 0
for i in par_list:
    for img_path in test_image_paths:
        img = read_img(img_path)
        img = image_brightness_change(img, i)
        save_img(img, "data/perturbations/E/" + str(index) + img_path[9:])
    index += 1

# Image brightness decrease
par_list = [0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
index = 0
for i in par_list:
    for img_path in test_image_paths:
        img = read_img(img_path)
        img = image_brightness_change(img, -i)
        save_img(img, "data/perturbations/F/" + str(index) + img_path[9:])
    index += 1

# Occlusion of the image increase plot
par_list = [0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
index = 0
for i in par_list:
    for img_path in test_image_paths:
        img = read_img(img_path)
        img = occlusion(img, i)
        save_img(img, "data/perturbations/G/" + str(index) + img_path[9:])
    index += 1

# Salt and pepper noise
par_list = [0, 0.02, 0.04, 0.06, 0.08, 0.10, 0.12, 0.14, 0.16, 0.18]
index = 0
for i in par_list:
    for img_path in test_image_paths:
        img = read_img(img_path)
        img = salt_and_pepper_noise(img, i)
        save_img(img, "data/perturbations/H/" + str(index) + img_path[9:])
        index += 1
```

### 4.3 svm$_{w}ith_{h}og.py$

This file is used to train and test the classical method model.

```
import os
import joblib
import numpy as np
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.svm import SVC
from utils import read_img, get_image_paths, resize_img
from skimage import feature


if __name__ == '__main__':
    # with help from https://debuggercafe.com/image-recognition-using-histogram-of-oriented-gradi
    DATA_PATH = 'data'
    IMAGE_CATEGORIES = [
        'black widow', 'captain america', 'doctor strange', 'hulk',
        'ironman', 'loki', 'spider-man', 'thanos'
    ]
    train_image_paths, test_image_paths, train_labels, test_labels = \
        get_image_paths(DATA_PATH, IMAGE_CATEGORIES)

    # If the model is saved do not retrain
    if os.path.exists('svm_hog.joblib'):
        print('Loading existing linear SVM model...')
        svm = joblib.load('svm_hog.joblib')
    else:
        # Train the model
        train_images = []
        for image_path in train_image_paths:
            # read the image as grayscale
            img = read_img(image_path, mono=True)
            # resize the image
            img = resize_img(img, (112, 112))
            # get the HOG descriptor for the image
            hog_desc = feature.hog(img, orientations=30, pixels_per_cell=(16, 16),
                                   cells_per_block=(2, 2), transform_sqrt=True, block_norm='L2-Hy
            # update the data
            train_images.append(hog_desc)
        print('Training on train images...')

        # #use grid search to tune the parameters.
        # with help from https://www.vebuso.com/2020/03/svm-hyperparameter-tuning-using-gridsearc
        # param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'po
        # grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=2)
        # grid.fit(train_images, train_labels)
        # print(grid.best_estimator_)
        svm = SVC(C=10, gamma=0.1)
        svm.fit(train_images, train_labels)
        joblib.dump(svm, 'svm_hog.joblib')
        print('Done')

        # 5-fold cross validation
        cross_val_scores = cross_val_score(svm, train_images, train_labels, cv=5, scoring='f1_wei
        print(np.mean(cross_val_scores))

    # Test the model using testing data
    test_images = []
    for image_path in test_image_paths:
        # read the image as grayscale
        img = read_img(image_path, mono=True)
        # resize the image
        img = resize_img(img, (112, 112))
        # get the HOG descriptor for the image
        hog_desc = feature.hog(img, orientations=30, pixels_per_cell=(16, 16),
                               cells_per_block=(2, 2), transform_sqrt=True, block_norm='L2-Hys')

        # update the data
        test_images.append(hog_desc)

    # predict the labels of the set of images
    test_predictions = svm.predict(test_images)
    # calculate accuracy using f1_score
    accuracy = f1_score(test_labels, test_predictions, average='weighted')
    print('Classification accuracy of SVM with HOG features:', accuracy)
```

### 4.4 ResNet-18.py

This file is used to train and test the deep learning method classifier.

```
import numpy as np
import matplotlib.pyplot as plt
import splitfolders

import torch
import torch.nn as nn
import torch.optim as optim

from torchvision import datasets, models, transforms
from torch.utils.data import DataLoader
from torchvision.models import ResNet18_Weights
from sklearn.metrics import f1_score

from tqdm import tqdm

if __name__ == '__main__':

    # Split train data into train/valid
    splitfolders.ratio("marvel/train", output="marvel/split",
                       seed=1337, ratio=(0.8, 0.2), group_prefix=None, move=False)

    train_transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.CenterCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(180),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])
    valid_transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.CenterCrop((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])
    test_transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.CenterCrop((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])

    # Load training and test sets
    train_ds = datasets.ImageFolder("data/split/train", transform=train_transform)
    valid_ds = datasets.ImageFolder("data/split/val", transform=valid_transform)
    test_ds = datasets.ImageFolder("data/test", transform=test_transform)

    print(f"Train data: Found {len(train_ds)} files belonging "
          f"to {len(train_ds.classes)} classes.")
    print(f"Validation data: Found {len(valid_ds)} files belonging "
          f"to {len(valid_ds.classes)} classes.")
    print(f"Test data: Found {len(test_ds)} files belonging "
          f"to {len(test_ds.classes)} classes.")

    # Enable GPU support if available and load data into iterable
    device = "cuda" if torch.cuda.is_available() else "cpu"
    kwargs = {"num_workers": 1, "pin_memory": True} if device == "cuda" else {}

    train_loader = DataLoader(
        train_ds,
        batch_size=64,
        shuffle=True,
        num_workers=8,
        **kwargs
    )
    valid_loader = DataLoader(
        valid_ds,
        batch_size=valid_ds.__len__(),
        shuffle=False,
        num_workers=8,
        **kwargs
    )
    test_loader = DataLoader(
        test_ds,
        batch_size=test_ds.__len__(),
        shuffle=False,
        num_workers=8,
        **kwargs
    )

    def view_image_batch():
        classes = {
            0: "Black Widow",
            1: "Captain America",
            2: "Doctor Strange",
            3: "Hulk",
            4: "Ironman",
            5: "Loki",
            6: "Spiderman",
            7: "Thanos"
        }

        rand_int = np.random.randint(2063)
        mean = [0.485, 0.456, 0.406]
        std = [0.229, 0.224, 0.225]

        label = train_ds[rand_int][1]
        image_class = classes[label]

        image = train_ds[rand_int][0].permute(1, 2, 0).numpy()
        image = image * std + mean

        plt.axis("off")
        plt.title(f"{image_class}")
        plt.imshow(image)
        plt.show()


    view_image_batch()


    def view_random_images():
        classes = {
            0: "Black Widow",
            1: "Captain America",
            2: "Doctor Strange",
            3: "Hulk",
            4: "Ironman",
            5: "Loki",
            6: "Spiderman",
            7: "Thanos"
        }

        figure = plt.figure(figsize=(8, 8))

        for i in range(1, 10):
            rand_int = np.random.randint(2063)
            mean = [0.485, 0.456, 0.406]
            std = [0.229, 0.224, 0.225]

            label = train_ds[rand_int][1]
            image_class = classes[label]

            image = train_ds[rand_int][0].permute(1, 2, 0).numpy()
            image = image * std + mean

            figure.add_subplot(3, 3, i)
            plt.axis("off")
            plt.title(f"{image_class}")
            plt.imshow(image)
        plt.show()


    view_random_images()

    # Define model parameters
    EPOCHS = 24
```

```
NUM_CLASSES = 8
learn_rate = 0.0005

# Load ResNet-18 with pre-trained weights
weights = ResNet18_Weights.DEFAULT
model = models.resnet18(weights=weights)

# Freeze all but fully connected layer
for name, param in model.named_parameters():
    if not name.startswith("fc"):
        param.requires_grad = False

# Adjust output to 8 classes
feature_number = model.fc.in_features
model.fc = nn.Linear(feature_number, NUM_CLASSES)

model = model.to(device)

# Use cross-entropy loss function and
# stochastic gradient decent
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(filter(
    lambda param: param.requires_grad,
    model.parameters()),
    learn_rate,
    momentum=0.9
)

loss_values_train = []
loss_values_valid = []
best_f1 = 0

for epoch in range(EPOCHS):
    """ Train """
    model.train()

    train_loss = 0.0
    train_corrects = 0
    f1_scores = []

    # Transfer data to GPU if available
    for inputs, labels in tqdm(train_loader):
        inputs = inputs.to(device)
        labels = labels.to(device)

        # Zero gradients
        optimizer.zero_grad()
        # Forward pass
        outputs = model(inputs)
        # Get predictions
        _, preds = torch.max(outputs, dim=1)
        # Find loss
        loss = criterion(outputs, labels)
        # Calculate gradients
        loss.backward()
        # Update weights
        optimizer.step()
        # Calculate loss
        train_loss += loss.item() * inputs.size(0)
```

```
            train_corrects += torch.sum(preds == labels.data)
            f1_scores.append(f1_score(labels.data, preds, average="weighted"))

        training_loss = train_loss / len(train_ds)
        loss_values_train.append(train_loss)
        accuracy = train_corrects / len(train_ds) * 100
        # Average of f1 score over batches
        f1 = np.mean(f1_scores)

        print(
            f"Epoch {epoch + 1} \n"
            f"-------- \n"
            f"Training Loss: {training_loss:.4f} \n"
            f"Accuracy: {accuracy:.4f}% \n"
            f"Average F1 Score: {f1} \n"
            f"---------------"
        )

        """ Validate """
        model.eval()

        with torch.no_grad():
            valid_loss = 0
            valid_corrects = 0

            # Transfer data to GPU if available
            for inputs, labels in valid_loader:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # Forward pass
                outputs = model(inputs)
                # Get predictions
                _, preds = torch.max(outputs, 1)
                # Find loss
                loss = criterion(outputs, labels)
                # Calculate loss
                valid_loss += loss.item() * inputs.size(0)
                valid_corrects += torch.sum(preds == labels.data)

        validation_loss = valid_loss / len(valid_ds)
        loss_values_valid.append(valid_loss)
        f1_val = f1_score(labels.data, preds, average="weighted")
        accuracy = valid_corrects / len(valid_ds) * 100

        if f1_val > best_f1:
            best_f1 = f1_val
            torch.save({
                "model_state_dict": model.state_dict(),
                "optimizer_state_dict": optimizer.state_dict()
            }, "best-model-weights")

        print(
            f"Validation Loss: {validation_loss:.4f} \n"
            f"Accuracy: {accuracy:.4f}% \n"
            f"F1 Score: {f1_val}"
        )

    plt.plot(loss_values_train)
    plt.show()

    plt.plot(loss_values_valid)
    plt.show()

    # Test Model
    # With help from https://debuggercafe.com/saving-and-loading-the-best-model-in-pytorch/

    # Load saved weights and params:
    best_params = torch.load("best-model-weights")

    # Load Resnet with best weights
    model = models.resnet18()

    # Freeze all but fully connected layer
    for name, param in model.named_parameters():
        if not name.startswith("fc"):
            param.requires_grad = False

    # Adjust output to 8 classes
    feature_number = model.fc.in_features
    model.fc = nn.Linear(feature_number, NUM_CLASSES)

    model.load_state_dict(best_params["model_state_dict"])
    model = model.to(device)

    """ Test """
    model.eval()

    with torch.no_grad():

        # Transfer data to GPU if available
        for inputs, labels in test_loader:
            inputs = inputs.to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model(inputs)
            # Get predictions
            _, preds = torch.max(outputs, 1)

    f1_test = f1_score(labels.data, preds, average="weighted")
    print("Testing...")
    print(f"F1 Score: {f1_test}")
```

## 4.5 plot$_{graphs.py}$

This file is used to plot the graphs for the exploration of robustness.

```
import joblib
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
from skimage import feature
from torchvision import datasets, models, transforms
from torch.utils.data import DataLoader
from sklearn.metrics import f1_score
from utils import read_img, resize_img, get_test_image_paths

if __name__ == '__main__':
    # Setup for classical method
    IMAGE_CATEGORIES = [
        'black widow', 'captain america', 'doctor strange', 'hulk',
        'ironman', 'loki', 'spider-man', 'thanos'
    ]

    # Load model
    classical_model = joblib.load('svm_hog.joblib')

    ############################
    # Setup for ResNet 18 method
    # Enable GPU support if available and load data into iterable
    device = "cuda" if torch.cuda.is_available() else "cpu"
    kwargs = {"num_workers": 1, "pin_memory": True} if device == "cuda" else {}

    # Transform used in training as well with no pre-augmentation
    test_transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.CenterCrop((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])

    # Load saved weights and params:
    best_params = torch.load("best-model-weights")

    # Load Resnet with best weights
    model = models.resnet18()

    # Freeze all but fully connected layer
    for name, param in model.named_parameters():
        if not name.startswith("fc"):
            param.requires_grad = False

    NUM_CLASSES = 8

    # Adjust output to 8 classes
    feature_number = model.fc.in_features
    model.fc = nn.Linear(feature_number, NUM_CLASSES)

    model.load_state_dict(best_params["model_state_dict"])
    model = model.to(device)

    ##################
    # Plot the graphs
    # classical_model is the model using hog features
    # model is the model using Resnet18

    # In each for-loop "i" refers to the index of the parameter used in each perturbation
    # cm refers to classical method
    # nn refers to neural network method (RESNET 18)

    # Gaussian pixel noise plot (use sub folder "A")
    gaussian_pixel_noise_list = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
    cm_gaussian_pixel_noise_accuracy_list = []
```

```
nn_gaussian_pixel_noise_accuracy_list = []
for i in range(10):
    DATA_PATH = "data/perturbations/A/" + str(i)

    # Classical method
    test_image_paths, test_labels = \
        get_test_image_paths(DATA_PATH, IMAGE_CATEGORIES)

    perturbed_test_images = []
    for img_path in test_image_paths:
        img = read_img(img_path, mono=True)
        img = resize_img(img, (112, 112))
        # get the HOG descriptor for the image
        hog_desc = feature.hog(img, orientations=30, pixels_per_cell=(16, 16),
                               cells_per_block=(2, 2), transform_sqrt=True, block_norm='L2-Hy

        # update the data
        perturbed_test_images.append(hog_desc)

    perturbed_test_predictions = classical_model.predict(perturbed_test_images)
    accuracy = f1_score(test_labels, perturbed_test_predictions, average='weighted')
    cm_gaussian_pixel_noise_accuracy_list.append(accuracy)

    # Res-Net 18 method
    """ Test """
    model.eval()
    # test sets
    test_ds = datasets.ImageFolder(DATA_PATH, transform=test_transform)

    test_loader = DataLoader(
        test_ds,
        batch_size=test_ds.__len__(),
        shuffle=False,
        num_workers=8,
        **kwargs
    )

    with torch.no_grad():
        # Transfer data to GPU if available
        for inputs, labels in test_loader:
            inputs = inputs.to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model(inputs)
            # Get predictions
            _, preds = torch.max(outputs, 1)

    f1_test = f1_score(labels.data, preds, average="weighted")
    nn_gaussian_pixel_noise_accuracy_list.append(f1_test)
    print(f1_test)

plt.plot(gaussian_pixel_noise_list, cm_gaussian_pixel_noise_accuracy_list, marker="o", label=
plt.plot(gaussian_pixel_noise_list, nn_gaussian_pixel_noise_accuracy_list, marker="o", label=
plt.legend()
plt.xlabel("Standard deviation of Gaussian distribution")
plt.ylabel("F1 score")
plt.title("Gaussian pixel noise")
plt.savefig("gaussian_pixel_noise.jpg")
plt.show()

# Gaussian blurring plot (use sub folder "B")
gaussian_blurring_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
cm_gaussian_blurring_accuracy_list = []
nn_gaussian_blurring_accuracy_list = []
for i in range(10):
    DATA_PATH = "data/perturbations/B/" + str(i)

    # Classical method
    test_image_paths, test_labels = \
        get_test_image_paths(DATA_PATH, IMAGE_CATEGORIES)

    perturbed_test_images = []
    for img_path in test_image_paths:
        img = read_img(img_path, mono=True)
        img = resize_img(img, (112, 112))
        # get the HOG descriptor for the image
        hog_desc = feature.hog(img, orientations=30, pixels_per_cell=(16, 16),
                               cells_per_block=(2, 2), transform_sqrt=True, block_norm='L2-Hy

        # update the data
        perturbed_test_images.append(hog_desc)

    perturbed_test_predictions = classical_model.predict(perturbed_test_images)
    accuracy = f1_score(test_labels, perturbed_test_predictions, average='weighted')
    cm_gaussian_blurring_accuracy_list.append(accuracy)
    # Res-Net 18 method
    """ Test """
    model.eval()
    # test sets
    test_ds = datasets.ImageFolder(DATA_PATH, transform=test_transform)

    test_loader = DataLoader(
        test_ds,
        batch_size=test_ds.__len__(),
        shuffle=False,
        num_workers=8,
        **kwargs
    )

    with torch.no_grad():
        # Transfer data to GPU if available
        for inputs, labels in test_loader:
            inputs = inputs.to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model(inputs)
            # Get predictions
            _, preds = torch.max(outputs, 1)

    f1_test = f1_score(labels.data, preds, average="weighted")
    nn_gaussian_blurring_accuracy_list.append(f1_test)
    print(f1_test)

plt.plot(gaussian_blurring_list, cm_gaussian_blurring_accuracy_list, marker="o", label="Class
plt.plot(gaussian_blurring_list, nn_gaussian_blurring_accuracy_list, marker="o", label="Deep
plt.legend()
```

```python
plt.xlabel("Times image convolved with gaussian blurring mask")
plt.ylabel("F1 score")
plt.title("Gaussian blurring")
plt.savefig("gaussian_blurring.jpg")
plt.show()

# Image contrast increase plot (use sub folder "C")
image_contrast_increase_list = [1, 1.01, 1.02, 1.03, 1.04, 1.05, 1.10, 1.15, 1.20, 1.25]
cm_image_contrast_increase_accuracy_list = []
nn_image_contrast_increase_accuracy_list = []
for i in range(10):
    DATA_PATH = "data/perturbations/C/" + str(i)

    # Classical method
    test_image_paths, test_labels = \
        get_test_image_paths(DATA_PATH, IMAGE_CATEGORIES)

    perturbed_test_images = []
    for img_path in test_image_paths:
        img = read_img(img_path, mono=True)
        img = resize_img(img, (112, 112))
        # get the HOG descriptor for the image
        hog_desc = feature.hog(img, orientations=30, pixels_per_cell=(16, 16),
                               cells_per_block=(2, 2), transform_sqrt=True, block_norm='L2-Hy

        # update the data
        perturbed_test_images.append(hog_desc)

    perturbed_test_predictions = classical_model.predict(perturbed_test_images)
    accuracy = f1_score(test_labels, perturbed_test_predictions, average='weighted')
    cm_image_contrast_increase_accuracy_list.append(accuracy)

    # Res-Net 18 method
    """ Test """
    model.eval()
    # test sets
    test_ds = datasets.ImageFolder(DATA_PATH, transform=test_transform)

    test_loader = DataLoader(
        test_ds,
        batch_size=test_ds.__len__(),
        shuffle=False,
        num_workers=8,
        **kwargs
    )

    with torch.no_grad():
        # Transfer data to GPU if available
        for inputs, labels in test_loader:
            inputs = inputs.to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model(inputs)
            # Get predictions
            _, preds = torch.max(outputs, 1)

    f1_test = f1_score(labels.data, preds, average="weighted")
    nn_image_contrast_increase_accuracy_list.append(f1_test)
    print(f1_test)

plt.plot(image_contrast_increase_list, cm_image_contrast_increase_accuracy_list, marker="o",
         label="Classical method")
plt.plot(image_contrast_increase_list, nn_image_contrast_increase_accuracy_list, marker="o",
         label="Deep learning method")
plt.legend()
plt.xlabel("Scaling factor of each pixel")
plt.ylabel("F1 score")
plt.title("Image Contrast Increase")
plt.savefig("image_contrast_increase.jpg")
plt.show()

# Image contrast decrease plot (use sub folder "D")
image_contrast_decrease_list = [1, 0.95, 0.90, 0.85, 0.80, 0.60, 0.40, 0.30, 0.20, 0.10]
cm_image_contrast_decrease_accuracy_list = []
nn_image_contrast_decrease_accuracy_list = []
for i in range(10):
    DATA_PATH = "data/perturbations/D/" + str(i)

    # Classical method
    test_image_paths, test_labels = \
        get_test_image_paths(DATA_PATH, IMAGE_CATEGORIES)

    perturbed_test_images = []
    for img_path in test_image_paths:
        img = read_img(img_path, mono=True)
        img = resize_img(img, (112, 112))
        # get the HOG descriptor for the image
        hog_desc = feature.hog(img, orientations=30, pixels_per_cell=(16, 16),
                               cells_per_block=(2, 2), transform_sqrt=True, block_norm='L2-Hy

        # update the data
        perturbed_test_images.append(hog_desc)

    perturbed_test_predictions = classical_model.predict(perturbed_test_images)
    accuracy = f1_score(test_labels, perturbed_test_predictions, average='weighted')
    cm_image_contrast_decrease_accuracy_list.append(accuracy)

    # Res-Net 18 method
    """ Test """
    model.eval()
    # test sets
    test_ds = datasets.ImageFolder(DATA_PATH, transform=test_transform)

    test_loader = DataLoader(
        test_ds,
        batch_size=test_ds.__len__(),
        shuffle=False,
        num_workers=8,
        **kwargs
    )

    with torch.no_grad():
        # Transfer data to GPU if available
        for inputs, labels in test_loader:
            inputs = inputs.to(device)
            labels = labels.to(device)
            # Forward pass
            outputs = model(inputs)
            # Get predictions
            _, preds = torch.max(outputs, 1)

    f1_test = f1_score(labels.data, preds, average="weighted")
    nn_image_contrast_decrease_accuracy_list.append(f1_test)
    print(f1_test)

plt.plot(image_contrast_decrease_list, cm_image_contrast_decrease_accuracy_list, marker="o",
         label="Classical method")
plt.plot(image_contrast_decrease_list, nn_image_contrast_decrease_accuracy_list, marker="o",
         label="Deep learning method")
plt.legend()
plt.xlabel("Scaling factor of each pixel")
plt.ylabel("F1 score")
plt.title("Image Contrast Decrease")
plt.savefig("image_contrast_decrease.jpg")
plt.show()

# Image brightness increase plot (use sub folder "E")
image_brightness_increase_list = [0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
cm_image_brightness_increase_accuracy_list = []
nn_image_brightness_increase_accuracy_list = []
for i in range(10):
    DATA_PATH = "data/perturbations/E/" + str(i)

    # Classical method
    test_image_paths, test_labels = \
        get_test_image_paths(DATA_PATH, IMAGE_CATEGORIES)

    perturbed_test_images = []
    for img_path in test_image_paths:
        img = read_img(img_path, mono=True)
        img = resize_img(img, (112, 112))
        # get the HOG descriptor for the image
        hog_desc = feature.hog(img, orientations=30, pixels_per_cell=(16, 16),
                               cells_per_block=(2, 2), transform_sqrt=True, block_norm='L2-Hy

        # update the data
        perturbed_test_images.append(hog_desc)

    perturbed_test_predictions = classical_model.predict(perturbed_test_images)
    accuracy = f1_score(test_labels, perturbed_test_predictions, average='weighted')
    cm_image_brightness_increase_accuracy_list.append(accuracy)

    # Res-Net 18 method
    """ Test """
    model.eval()
    # test sets
    test_ds = datasets.ImageFolder(DATA_PATH, transform=test_transform)

    test_loader = DataLoader(
        test_ds,
        batch_size=test_ds.__len__(),
        shuffle=False,
        num_workers=8,
        **kwargs
    )
    with torch.no_grad():
        # Transfer data to GPU if available
        for inputs, labels in test_loader:
            inputs = inputs.to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model(inputs)
            # Get predictions
            _, preds = torch.max(outputs, 1)

    f1_test = f1_score(labels.data, preds, average="weighted")
    nn_image_brightness_increase_accuracy_list.append(f1_test)
    print(f1_test)

plt.plot(image_brightness_increase_list, cm_image_brightness_increase_accuracy_list, marker='
         label="Classical method")
plt.plot(image_brightness_increase_list, nn_image_brightness_increase_accuracy_list, marker='
         label="Deep learning method")
plt.legend()
plt.xlabel("Magnitude of addition to each pixel")
plt.ylabel("F1 score")
plt.title("Image Brightness Increase")
plt.savefig("image_brightness_increase.jpg")
plt.show()

# Image brightness decrease plot (use sub folder "F")
image_brightness_decrease_list = [0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
cm_image_brightness_decrease_accuracy_list = []
nn_image_brightness_decrease_accuracy_list = []
for i in range(10):
    DATA_PATH = "data/perturbations/F/" + str(i)

    # Classical method
    test_image_paths, test_labels = \
        get_test_image_paths(DATA_PATH, IMAGE_CATEGORIES)

    perturbed_test_images = []
    for img_path in test_image_paths:
        img = read_img(img_path, mono=True)
        img = resize_img(img, (112, 112))
        # get the HOG descriptor for the image
        hog_desc = feature.hog(img, orientations=30, pixels_per_cell=(16, 16),
                               cells_per_block=(2, 2), transform_sqrt=True, block_norm='L2-Hy

        # update the data
        perturbed_test_images.append(hog_desc)

    perturbed_test_predictions = classical_model.predict(perturbed_test_images)
    accuracy = f1_score(test_labels, perturbed_test_predictions, average='weighted')
    cm_image_brightness_decrease_accuracy_list.append(accuracy)

    # Res-Net 18 method
    """ Test """
    model.eval()
    # test sets
    test_ds = datasets.ImageFolder(DATA_PATH, transform=test_transform)
    test_loader = DataLoader(
        test_ds,
        batch_size=test_ds.__len__(),
        shuffle=False,
        num_workers=8,
        **kwargs
    )

    with torch.no_grad():
        # Transfer data to GPU if available
        for inputs, labels in test_loader:
            inputs = inputs.to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model(inputs)
            # Get predictions
            _, preds = torch.max(outputs, 1)

    f1_test = f1_score(labels.data, preds, average="weighted")
    nn_image_brightness_decrease_accuracy_list.append(f1_test)
    print(f1_test)

plt.plot(image_brightness_decrease_list, cm_image_brightness_decrease_accuracy_list, marker='
         label="Classical method")
plt.plot(image_brightness_decrease_list, nn_image_brightness_decrease_accuracy_list, marker='
         label="Deep learning method")
plt.legend()
plt.xlabel("Magnitude of subtraction to each pixel")
plt.ylabel("F1 score")
plt.title("Image Brightness Decrease")
plt.savefig("image_brightness_decrease.jpg")
plt.show()

# Occlusion of the image increase plot (use sub folder "G")
occlusion_increase_list = [0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
cm_occlusion_increase_accuracy_list = []
nn_occlusion_increase_accuracy_list = []
for i in range(10):
    DATA_PATH = "data/perturbations/G/" + str(i)

    # Classical method
    test_image_paths, test_labels = \
        get_test_image_paths(DATA_PATH, IMAGE_CATEGORIES)

    perturbed_test_images = []
    for img_path in test_image_paths:
        img = read_img(img_path, mono=True)
        img = resize_img(img, (112, 112))
        # get the HOG descriptor for the image
        hog_desc = feature.hog(img, orientations=30, pixels_per_cell=(16, 16),
                               cells_per_block=(2, 2), transform_sqrt=True, block_norm='L2-Hy

        # update the data
        perturbed_test_images.append(hog_desc)

    perturbed_test_predictions = classical_model.predict(perturbed_test_images)
    accuracy = f1_score(test_labels, perturbed_test_predictions, average='weighted')
    cm_occlusion_increase_accuracy_list.append(accuracy)
    # Res-Net 18 method
    """ Test """
    model.eval()
    # test sets
    test_ds = datasets.ImageFolder(DATA_PATH, transform=test_transform)

    test_loader = DataLoader(
        test_ds,
        batch_size=test_ds.__len__(),
        shuffle=False,
        num_workers=8,
        **kwargs
    )

    with torch.no_grad():
        # Transfer data to GPU if available
        for inputs, labels in test_loader:
            inputs = inputs.to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model(inputs)
            # Get predictions
            _, preds = torch.max(outputs, 1)

    f1_test = f1_score(labels.data, preds, average="weighted")
    nn_occlusion_increase_accuracy_list.append(f1_test)
    print(f1_test)

plt.plot(occlusion_increase_list, cm_occlusion_increase_accuracy_list, marker="o", label="Cla
plt.plot(occlusion_increase_list, nn_occlusion_increase_accuracy_list, marker="o", label="Dee
plt.legend()
plt.xlabel("Square edge length")
plt.ylabel("F1 score")
plt.title("Occlusion of the Image Increase")
plt.savefig("occlusion.jpg")
plt.show()

# Salt and pepper noise plot (use sub folder "H")
salt_and_pepper_noise_list = [0, 0.02, 0.04, 0.06, 0.08, 0.10, 0.12, 0.14, 0.16, 0.18]
cm_salt_and_pepper_noise_accuracy_list = []
nn_salt_and_pepper_noise_accuracy_list = []
for i in range(10):
    DATA_PATH = "data/perturbations/H/" + str(i)

    # Classical method
    test_image_paths, test_labels = \
        get_test_image_paths(DATA_PATH, IMAGE_CATEGORIES)

    perturbed_test_images = []
    for img_path in test_image_paths:
        img = read_img(img_path, mono=True)
        img = resize_img(img, (112, 112))
        # get the HOG descriptor for the image
        hog_desc = feature.hog(img, orientations=30, pixels_per_cell=(16, 16),
                               cells_per_block=(2, 2), transform_sqrt=True, block_norm='L2-Hy

        # update the data
```

```
            perturbed_test_images.append(hog_desc)

        perturbed_test_predictions = classical_model.predict(perturbed_test_images)
        accuracy = f1_score(test_labels, perturbed_test_predictions, average='weighted')
        cm_salt_and_pepper_noise_accuracy_list.append(accuracy)

        # Res-Net 18 method
        """ Test """
        model.eval()
        # test sets
        test_ds = datasets.ImageFolder(DATA_PATH, transform=test_transform)

        test_loader = DataLoader(
            test_ds,
            batch_size=test_ds.__len__(),
            shuffle=False,
            num_workers=8,
            **kwargs
        )

        with torch.no_grad():
            # Transfer data to GPU if available
            for inputs, labels in test_loader:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # Forward pass
                outputs = model(inputs)
                # Get predictions
                _, preds = torch.max(outputs, 1)

        f1_test = f1_score(labels.data, preds, average="weighted")
        nn_salt_and_pepper_noise_accuracy_list.append(f1_test)
        print(f1_test)

    plt.plot(salt_and_pepper_noise_list, cm_salt_and_pepper_noise_accuracy_list, marker="o",
             label="Classical method")
    plt.plot(salt_and_pepper_noise_list, nn_salt_and_pepper_noise_accuracy_list, marker="o",
             label="Deep learning method")
    plt.legend()
    plt.xlabel("Salt and pepper noise strength")
    plt.ylabel("F1 score")
    plt.title("Salt and Pepper Noise")
    plt.savefig("salt_and_pepper_noise.jpg")
    plt.show()
```