IMAGE AND VISION COMPUTING: MINI-PROJECT

s2450044 **s2446971**

November 25, 2022

ABSTRACT

A Classical model and a Deep Learning model are trained, optimised and tested in the classification of images of a data set containing Marvel superheroes. The classical model consists of concatenated representation transformations given by the SIFT detector and Bag of Words complemented by the use of the Support Vector Machines classifier. For the Deep Learning model ResNet18 with freezing is used. A comparison of the performance of these models in the task is given. At the end, an evaluation of their robustness to phenomena such as Gaussian blurring, Gaussian Noise, Salt and Pepper Noise, Contrast and Brightness variation and Occlusion is given. The models are then compared and their robustness or lack-there-of to the above perturbations examined.

1 INTRODUCTION

In this project we focus on the task of classification. As aforementioned, we implement two models by training them, validating different settings of them, and testing the best settings of them. Our Marvel data is obtained from [1] and contains superhero images spread among 8 classes: Black Widow, Captain America, Doctor Strange, Hulk, Ironman, Loki, Spider-man and Thanos.
The project is divided into several sections. In the first two sections, we describe the two models that are used in a hierarchical approach that describes the components of the models and, afterwards, how they are blended together and implemented. The following section discusses necessary data preparation in order to carry out our experiments. This is then followed by descriptions of the results obtained after training and validating our models with a range of parameters. At the end we explore the robustness of the best models of each approach (Classical and Deep Learning approaches) under several image transformations: Gaussian blurring, Gaussian Noise, Salt and Pepper Noise, Contrast and Brightness variation as well as Occlusion.

2 THE CLASSICAL APPROACH: SIFT DETECTOR, BoW AND SVM

In order to build our first classifier, we use the SIFT detector, BoW and the SVM classifier consecutively. During training, our classifier is built by going over the training images detecting their salient points with the SIFT detector and getting their descriptors, building a codebook by applying the k-Means Clustering Algorithm over the descriptors, computing the BoW representation of each of the training images with the codebook, fitting a SVM classifier to the BoW representations of the images to classify them into the classes of superheroes: Black Widow, Captain America, Doctor Strange, Hulk, Ironman, Loki, Spider-man and Thanos.

We now describe in detail each of the components of this classifier while describing how they are combined during training. The same process that we describe is done during testing with the exception that the codebook used for creating the BoW of each image is the one learned over the training process.

2.1 SCALE-INVARIANT FEATURE TRANSFORM (SIFT) DETECTOR

The SIFT detector is a detector of blob salient points which is scale-invariant in the sense that the salient points detected by it are consistently detected as we zoom in and out of the image at the location of the point. Due to the fact that images in our data set have zoomed versions in the same data set, we can expect this to occur for more superhero images that our classifier might encounter. That is the main reason why we choose the SIFT detector (for instance, over the

Harris Detector) for an initial representation of images in terms of their salient points, apart from the fact that the SIFT detector gives, robust, distinctive and compact descriptors, and has been shown to be efficient.

The SIFT detector is based on applying the Difference of Gaussians (DoG) filter which approximates the Laplacian of the Gaussian p.d.f. (LoG) filter, with different values of the standard deviation (effectively the variance) σ. This detects blobs at different scales as for smaller σ the DoG filter gives an image with high intensities at blobs that then are detected in a smaller scale (zoomed in) while, for a bigger σ, the detected salient points are detected in a bigger scale (zoomed out).

This SIFT detector then outputs the salient points that were detected after convolving our image with DoG filters with different sigmas. That is, the outputs are the blob salient points that are scale invariant.

In our implementation in the file "sift_bow_svm.py", the SIFT detector is applied when building the codebook for the BoW representation of images which is later explained in the function "build_codebook", and when creating the BoW representation for each image in the function "bag_of_word". In addition, we initially allow a maximum of 50 salient points per image (selected from the top salient points once ranked). In the next subsection we explain how the detected salient points are used to build a codebook.

Figure 1: Salient Points detected by the SIFT detector in an image of Iron Man from our training set. Figure made with cv2's ".drawkeypoints()" method.

2.2 BAG OF WORDS (BoW)

After the SIFT detector is applied to our training images, we obtain an initial representation of our images via the descriptors produced out of the salient points detected. For each image, given a particular salient point in the image, we compute its descriptor by taking the neighbouring pixels in a patch of 16×16 which we then divide into sixteen 4-by-4 cells, applying some selected gradient filters to calculate the gradients of the pixels in the cells, and then creating for each cell a histogram of gradients by allocating the gradient of each pixel (in the cell) into a bin which corresponds to an umbrella of directions. Typically, we use 8 bins for our histogram of gradients. Furthermore, for each cell's histogram of gradients we shift the bins so that the most frequent bin is the left most. This makes the descriptor orientation invariant. The ending descriptor of each salient point is the encoding of all cell histograms into a vector, which is then $16 * 8 = 128$-dimensional.

In our code, the descriptor is automatically obtained alongside the salient point detected by our SIFT detector when using the "detectAndCompute" method of the "cv2.SIFT_create" instance.

After we have the descriptors for all images in the training set, we apply the k-Means Algorithm with $k = 15$ number of centroids. The k-Means Algorithm is a clustering algorithm that, given a metric (in our implementation the Euclidean Metric) and a number of clusters k, constructs k clusters out of the data points (in our case the 128-dimensional descriptors of all images) minimising the aggregate intra-cluster distance. The algorithm starts by initializing k centroids. Then it iteratively does the following:

- it assigns all data points to the cluster with closest centroid from them.
- computes the average of the data points in each centroid and sets it as the replacement of the centroid.

until clusters do not change in terms of the data points that they contain. This step is done to create the codebook for the BoW that we describe afterwards.

Once the codebook is built, we create a Bag of Words (BoW) representation of each image as follows. We take the descriptors obtained from the image's salient points, and assigned them to the centroids from our codebook while counting the number of descriptors that each centroid receives for the image. Then, we represent the image by a $k = 15$-dimensional vector whose entries count the amount of descriptors that were assigned to a particular centroid (the one corresponding to that entry). After doing this for all training images, we have represented images via their BoW representation, and can continue towards fitting a Machine Learning Model for the classification of images into the available classes of superheroes.

In our implementation in the file "sift_bow_svm.py", the BoW representation is build using the the function "bag_of_words".

2.3 SUPPORT VECTOR MACHINES (SVMs)

Once we have our Bag of Words representation of all images in the training set, we standardise all the BoW vectors. It is over these standardised vectors that we train the SVM classifier for multi-class classification into the classes: Black Widow, Captain America, Doctor Strange, Hulk, Ironman, Loki, Spider-man and Thanos. Notice that as we standardised BoW vectors before training our classifier, in the validation and test sets we should standardise them as well with the same mean and standard deviation values before applying our classifier.

The SVM classifier used is based on the "one-versus-one" approach for multi-class classification as mentioned in the section of Multi-Class Classification in [2]. This approach is based on constructing several SVM classifiers. Each of them is used to discriminate between two classes, so that in total, if there are C classes, our SVM classifier first builds $\binom{C}{2} = \frac{C(C-1)}{2}$ separators which try to maximise the margin between the support vectors while correctly classifying as many data points in the two classes as possible. Then the classifier labels new data points by the class which was chosen more along the separators built [3].

Therefore, the basis of our classifier is the SVM Classifier for Binary Classification, which is applied per pairs of classes. More concretely, as our data is likely to not be separable since there might several common words for different superheroes, we use the version of the SVM Binary Classifier which softens the misclassification constrains. This version is based on the minimisation of

$$\frac{||\vec{w}||^2}{2} + C\sum_{i=1}^{N} \max\{0, 1 - y_i(\vec{W}\vec{x}_i + b)\}$$

with respect to the weight vector $\vec{w}$ and bias b, where $\{(\vec{x}_1, y_1), ..., (\vec{x}_N, y_N)\}$ are the data points (images) from the classes that our binary separator is trying to classify. The constant C is varied in our experiments to see how changing the penalization of misclassification alters our training and classification accuracy.

In addition in this project we used the kernel trick with linear, polynomial functions, in order to best fit our training data by having more complex decision boundaries.

3 THE DEEP LEARNING APPROACH: ResNet18.

3.1 Convolutional Neural Networks

From Deep Learning we chose to train and evaluate the ResNet18 model in the task of classifying the images in our marvel data set into our 8 Marvel character classes. The ResNet18 model is a deep Convolutional Neural Network (CNN) based on the research paper [**?**] which lead to a major shift in approach to how we train and use CNNs. Let us first begin with a description of traditional CNNs, these incorporated 3 fundamental ideas on top of traditional Neural Networks: 'local receptive fields', 'shared weights' and 'pooling'. Contrary to Neural Networks which traditionally connect every input pixel to every hidden neuron in the first layer, and continue using fully connected layers throughout, CNNs only make connections in small, localized regions of the input image ($(n \times n)$ regions), we call these regions local receptive fields. These receptive fields are then slid over the entire input image according to the stride length chosen. In this way we can think of individual neurons in deeper layers as each learning to analyze it's corresponding local receptive field. Furthermore we set exactly the same $(n \times n)$ shared weights for each of the neurons in that given layer, so that all the neurons on a certain layer look for the same feature. This combination of sharing weights and not having fully connected layers has the added benefit of drastically reducing the number of parameters our network has to learn, often by quite noticeable factors. This in turn allows us to add more channels in parallel known as feature maps for each layer, this allows different channels in a single layer to search for their own independent feature in the image at that level, in this way one layer can be looking for 'multiple features' at once, one for each channel. As such our layers are no longer a sequence of neurons but rather tensors, as is our input image. We can think of the input image as a

3Dimensional Tensor of shape (3,224,224) which can be thought of as (depth, height of image, width of image) where the depth are the 3 channels corresponding to RGB. Further layers also compose of tensors but the channels (depth dimension) here represent different features being learned, while the other dimensions represent the height and width of the outputs.

3.2 The benefits of ResNet

The problem with CNNs begins as we increase the depth of the networks, intuitively, as we add more layers our network performance should improve, however this is not always the case. Due to the Vanishing Gradient Problem, after a certain depth, CNNs become unable to learn the model. Here we can use ResNet, a solution which implements Batch Normalisation as well as Skip Connections, allowing us to build much deeper networks. The model we will be using 'freezing' on the covolutional layers, thus retaining some of the information learned from it's original training set (ImageNet) and learning through the fully connected layer. We tuned the 'momentum' and 'learning rate' parameters to further improve our F1 Score. Momentum is a variation of Stochastic Gradient Descent which allows us to speed up learning and move past local minima by using gradients from previous epochs as well as helping us avoid randomness in learning by being less vulnerable to learning noise in our data. Learning rate allows us to converge towards a minimum loss function by controlling the step size for each iteration of the algorithm, usually smaller learning rates are preferred so as to not overshoot however a learning rate too small may result in very slow learning or getting stuck altogether.

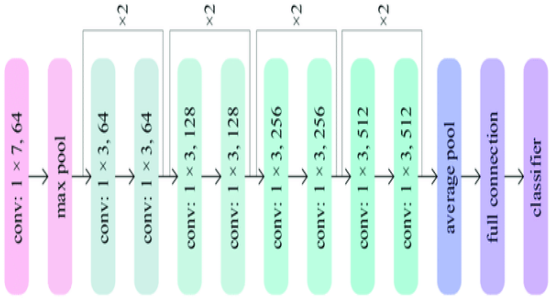

Figure 2: Structure of the layers of ResNet18 used, including 17 convolutional layers and one fully connected layer as well as Max Pooling and Average Pooling [**?**]

3.3 Max and Average Pooling

ResNet also implements Max Pooling straight after the initial activation function as well as Average Pooling right before the fully connected layer (right at the end of the model). Max pooling takes the maximum value of patches (whose size is determined by an $n \times n$ pool size) of a feature map and forwards these values to the next (downsampled) feature map, in a similar way Average pooling simply outputs the average value of all the values in the patch. Max Pooling is used initially for it's ability to extract features such as edges, which are useful for the neural network in learning more general features in deeper layers. Meanwhile, average pooling is much softer and is thus useful right before the fully connected layer. The primary benefit to these pooling layers is that, although they result in downsampling, they progressively lead to spatial invariance, allowing object recognition to be performed with a lesser degree of sensitivity towards specific image location.

4 Data Preparation

The original data given consist of two sets labeled as train and validation which had an unbalanced number of images per each class, and which were small in size. In order to obtain a better model performance, we decide to augment the data by applying translations, rotations, flipping transformation, and scale variations (done with file DataAugmentation.ipynb using the ImageDataGenerator function from keras.preprocessing.Image). Furthermore, while doing this step we separate (in file datasplit.ipynb) our augmented data into two sets, train and validation, which have the same amount of images per class to avoid the problem of class imbalance. We relabel our original validation set as test set.
Having images of different sizes is potentially problematic in many tasks of this project. For instance, some of the models used such as ResNet18 requires inputs to have the same dimensions, or when performing an occlusion in our data with same size squares we might have same levels of occlusion being less significant due to the image size. We solve this problem via two approaches. The first one is to resize images while doing data augmentation (in the same

file for augmenting the data) and to resize the test data separately. The second is to resize the images created for the robustness exploration section while creating these sets (done with data_robustness.py). We resized our images to size 224×224. After preparing the data, we obtained three sets: train, validation and test set. They had the following amount of images per class respectively: 1049, 263, 50.

5 F1-SCORE: THE MEASURE OF EVALUATION

In this project we make use of Micro F1 Score as evaluation metric of the model. The reason for choosing Micro F1 score rather than Macro F1 score is based on the fact that Macro F1 score is typically used over Micro F1 score when having the problem of class imbalance in our data sets. However, as explained in the section of data preparation, we deal with the problem of class imbalances during the step of constructing the train, validation and test sets. Therefore, the use of Micro F1 score is justified and, henceforth, we refer to it as F1 score.

6 TRAINING AND VALIDATION OF MODELS

6.1 TRAINING AND VALIDATION OF THE MODEL OF THE CLASSICAL APPROACH

The baseline setting of our model of the classical approach is the following. The maximum number of detected salient points by the SIFT detector in each image is 50 (chosen after being sorted if there are more than 50 salient points), the number of words that our codebook has is 15, and the SVM classifier is set to have $C = 1$ and linear kernel. Initially, the number of images per class in our training data set is bigger than 300 (although they are close to 300) and the number of images per class in our test set is bigger than 50. Nevertheless, in order to avoid class imbalances we decide to use 300 images for each class during training. The performance of the baseline model in this training and validation set is:

Model	Train F1-score	Validation F1-score
Baseline	0.369	0.195

Table 1: Baseline Model for Classical approach performance.

Since both the training and validation F1 score are low we conclude that our initial model is underfitting the data. There are several possible reasons why this might happen: the collection of the data might have not been rigorous, the model chosen might not be appropriate, etc... We choose to study the following possibilities:

- The amount of data could be insufficient, so that our classifier is not able to achieve a good performance.
- The complexity of our model cannot capture the underlying structure of the data i.e. we need to have more complex decision boundaries.

We check these hypotheses in order. In order to check the first hypothesis, we start by seeing how our model would perform in a bigger data set. Therefore, we augment our training data by taking our original training set adding transformations of the original images, where the used transformations are: rotations, translations, scale variations and vertical and horizontal flippings. In order to validate different model settings used, we additionally separate twenty percent of the resulting training set in a validation set. In both the training and validation set, we apply downsampling in order to avoid class imbalances so that at last we have 1049 images per class in the training set, and 263 images per class in the validation set. The obtained sets are also used in the training and validation of different model setting of the ResNet18 Convolutional Neural Network.

Once this step is done,we see that indeed augmenting our data sets automatically improved the baseline model's performance:

Figure 3: An example of augmented data with Black Widow: Rotated (Left), Resized (Centre) and shifted (Right).

Model	Train F1-score	Validation F1-score
Baseline with Augmented data	0.403	0.292

Table 2: Baseline model's performance on Augmented Data.

Once observed that augmenting the data was beneficial, we set the training and validation sets resulting from the augmentation as our base data sets. Then, we check our second hypothesis, while seeking to find the model setting which best improves the validation F1 score:

Model Setting	Training F1-score	Validation F1-score
SVM(C=2, Kernel=poly, deg=2)	0.402	0.297
SVM(C=2, Kernel=poly, deg=4)	0.401	0.283
SVM(C=2, Kernel=poly, deg=6)	0.404	0.291
SVM(C=2, Kernel=poly, deg=8)	0.402	0.280

Table 3: Performance of Different Model Settings on the Augmented Data.

Among the studied models which in order of appearance in the last table have increasing complexity (attain more complex decision boundaries), we see that the simpler model which has $C = 2$, and a polynomial kernel of degree 2 obtains the best results. Furthermore, we see in the table that as complexity increases the validation F1-score decreases strictly. Hence, we propose a study of how training and validation performance varies as the maximum number of features and the codebook size used varies for the future. That is, we propose the study of how varying the parameters of the model which encode the representation of images, affects the performance of the classical approach on this task. Having realized that the best model found is the one with $C = 2$ and a polynomial kernel of degree 2, we test our model obtaining an F1-score in the Test set of 0.233.

6.2 TRAINING AND VALIDATION OF ResNet18

Our baseline model for ResNet 18 is implemented by slightly changing the code from [4]. With this code, the model received from our training process is the one corresponding to the epoch which gave the best overall F1 Score as we can see in the definition of the function "train_model()". As mentioned previously we will be conducting all our training, validation and testing on exactly the same image set as for our classical models, in order to allow us to accurate compare the results of the two.
The results for our ResNet18 model are shown in the table below:

Model	Train F1 Score	Val F1 Score
Baseline(Momentum = 0.9 , Learning Rate = 0.001)	0.6269	0.6327
ResNet18(Momentum = 0.91, Learning Rate = 0.001)	0.631	0.634
ResNet18(Momentum = 0.91 , Learning Rate = 0.00075)	0.6399	0.6417
ResNet18(Momentum = 0.91 , Learning Rate = 0.0001)	0.6174	0.6211
ResNet18(Momentum = 0.92 , Learning Rate = 0.0005)	0.6339	0.6388

Table 4: Baseline Model for Deep Learning approach performance.

We observe here the importance of momentum and learning rate. Specifically, it is apparent in the 3rd model how a much lower learning rate ($10\times$ smaller than that of our baseline model) can suffer from stagnation and become stuck in a local minima. We can also appreciate that increasing the momentum slightly has positive effects on learning and generally on the speed of training the models. Out of the models tested, the model with Momentum = 0.91 and Learning Rate slightly decreased to 0.00075 obtained the highest Validation F1 Score, as well as the highest overall Train F1 Score. In addition, it is important to note that the trained ResNet18 baseline model performs considerably better than even the best model from the classical approach (see table 1). Furthermore our final F1 Score on the test set was 0.595.

7 CLASSICAL VS DEEP LEARNING APPROACH

Overall we see that the Deep Learning Approach obtains generally a better performance with ResNet18 having always better training and validation F1-score than any of the classical models using a concatenation of SIFT detector, BoWs and SVMs. The decisive metric is given by the test set once we have chosen the best model per approach using the validation score. While the best classical model obtain a test F1-score of 0.233, the ResNet18 model obtained a value of 0.595. This is expected since heuristically the layers of ResNet18 learn a hierarchical representation of the classes in our sets which has more depth than the representation attained with the concatenation of the SIFT detector and BoWs: this would only correspond to one convolutional layer followed by one downsampling layer, while ReNet18 has 18 layers. Furthermore, the use of learnable filters throughout the image layers, and the use of downsampling layers makes learning location invariance.

8 ROBUSTNESS EXPLORATION

In this section we explore the robustness of the best classifiers of the classical and Deep Learning approach. We start by looking at some examples of images perturbed with the transformations used to test the robustness of the models:

Figure 4: An image of Spider Man with varying standard deviation for Gaussian Pixel Noise: 0, 8 and 18 respectively.

Figure 5: An image of Spider Man with varying times of applying a Gaussian Blurring filter: 0, 5 and 9 respectively.

Figure 6: An image of Spider Man with varying contrast by the factors 0.8, 1 and 1.2 respectively.

Figure 7: An image of Spider Man with varying brightness by the factors -20, 0 and 20 respectively.

Figure 8: An image of Spider Man with varying length of square used in the image occlusion: 0, 20 and 40 respectively.

In order to see how these perturbations are transformed, the file "data_robustness.py" can be seen (it is self explanatory, although it is important to see that the transformations are commented out in the function "transformation_applications"

Figure 9: An image of Spider Man with varying amounts of sp noise: 0, 0.02 and 0.14 respectively.

defined in the file.
Once we have seen how this perturbations look like and how to produce them, we can go on to test the robustness of our best classifiers on these. See Figure 10
It is noticeable that with respect to Gaussian Pixel Noise, ResNet18 is less robust than the classical model. Clearly, as the size of the standard deviation, from the Gaussian Distribution added pixel-wise to the original, increases, the difference between ReNet18's original performance and its performance on the perturbed set is bigger than that of the classical model with its original performance.
We observe here that neither model is strongly robust to Gaussian Blurring. In particular the ResNet model suffers a steepest decrease in performance as stronger blurring is applied, intuitively this makes sense as edges and features become exceedingly harder to identify.

We note that both models are particularly robust to occlusion. Both ResNet18 and the classical approach have the property of performing a hierarchical representation of images so that several features of images are learned to recognize each class. In the case of ResNet18 this is done by the amount of feature maps used, while for the classical approach having 20 SIFT descriptors of the image and a codebook achieves this. Therefore, both methods are able of distinguishing between classes (to some extent) even when some of the features that characterize a class (a superhero in this case) are occluded.
Both models are noticeably more robust to an increase rather than a decrease in contrast, however this may also be attributed to the factor of multiplication in both cases. When contrast is increased edges become easier to identify, and so shapes are generally more distinguishable overall. This is in contrast to when contrast is lowered, where pixel values become less distinguishable between themselves and thus both models struggle to accurately categorize the images. The classical approach does not appear particularly robust to either case however.

It can be seen that ResNet18 is more robust to increasing and decreasing brightness since we see that ResNet18's F1 score decreases from an original approximate value of 0.6 to a value of around 0.5, while the classical model decreases from an initial value of around 0.2 to around 0. Heuristically, this could be explain by noticing that blobs are less easily recognisable with decreasing and increasing brightness as can be observed in figure 7.
We see that both models decrease in performance as the amount of salt and pepper noise increases. Nevertheless, it is remarkable the lack of robustness of ResNet18 to this perturbation as it is the one in which its worsening is steepest. This can be explained by the fact that convolutional layers use filters which, although spatially invariant, are still susceptible to the global degrading of the image. This can be seen better when noticing that the global perturbations (such as Gaussian Pixel Noise, Gaussian Blurring, Salt and Pepper Noise,...) are the ones producing a steepest decrease of performance of ResNet18.

9 Conclusion

We have analysed and explored the implementations of two different learning approaches to an image classification task as well as explained in depth how they are applied and the benefits of each. Due to limited data size and quality our F1 Scores are noticebaly lower than they may have otherwise been in a clean dataset. Nonetheless they allowed us to show a concise representation of the impacts on performance of various perturbations.

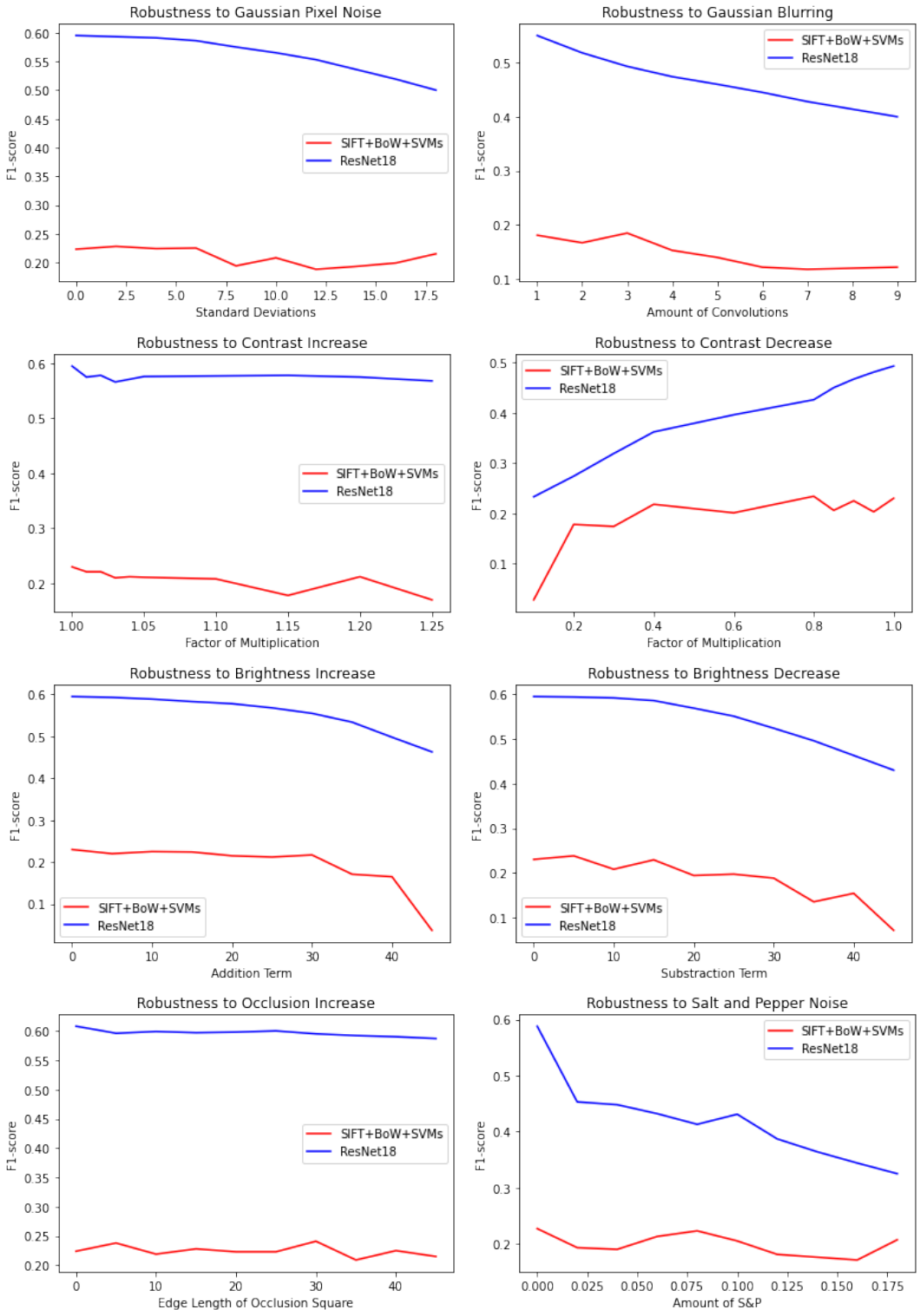

Figure 10: Robustness Exploration of the best Classical and Deep Learning models.

References

[1] `https://www.kaggle.com/datasets/hchen13/marvel-heroes` Last time checked: 17/11/2022.

[2] https://scikit-learn.org/stable/modules/svm.html#classification. Last time checked: 13/11/2022.

[3] https://courses.media.mit.edu/2006fall/mas622j/Projects/aisen-project/#:~:text=One-against-all%20%28OAA%29%20SVMs%20were%20first%20introduced%20by%20Vladimir,it%20and%20all%20other%20classes%27%20SVMs%20rejected%20it. Last time checked: 13/11/2022.

[4] https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html#sphx-glr-beginner-finetuning-torchvision-models-tutorial-py Last time checked: 19/11/2022.

[5] https://www.researchgate.net/publication/360470173_Multi-Classifier_Fusion_for_Open-Set_Specific_Emitter_Identification, Yurui Zhao. Last time checked: 25/11/2022.

[6] https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html#sphx-glr-beginner-finetuning-torchvision-models-tutorial-py. Last time checked: 25/11/2022.

[7] Code for SIFT, Bow, SVMs obtained from Lab3 from Image and Vision Computing although modified afterwards.