Multimedia Tools and Applications manuscript No.
(will be inserted by the editor)

Photo Quality Classification Using Deep Learning

Arash Golchubian · Oge Marques · Mehrdad Nojoumian

Received: date / Accepted: date

Abstract The detection of poor quality images for reasons such as focus, lighting, compression, and encoding is of great importance in the field of computer vision. The ability to quickly and automatically classify an image as poor quality creates opportunities for a multitude of applications such as digital cameras, phones, self-driving cars, and web search technologies. In this paper an end-to-end approach using Convolutional Neural Networks (CNN) is presented to classify images into six categories of bad lighting, Gaussian blur, motion blur, JPEG 2000, white-noise, and high quality reference images. A new dataset of images was produced and used to train and validate the model. Finally, the application of the developed model was evaluated using images from the German Traffic Sign Recognition Benchmark. The results show that the trained CNN can detect and correctly classify images into the aforementioned categories with high accuracy and the model can be easily re-calibrated for other applications with only a small sample of training images.

1 Introduction

The classification of unwanted images are of great importance in the field of computer vision. The ability to automatically detect undesirable images would enable many useful applications. Search engines would be able to automatically discard those images that are of poor quality; digital cameras and phone

Arash Golchubian · Oge Marques · Mehrdad Nojoumian
Florida Atlantic University
Department of Computer & Electrical Engineering and Computer Science
777 Glades Road
Boca Raton, FL 33431-0991
Tel.: +561-297-3855
Fax: +561-297-2800
E-mail: agolchub@fau.edu

camera software would be able to alert the user of a poor quality shot so that they may correct the mistake; autonomous driving technology would be able to ignore poorly shot frames to reduce the chances of a mistake. The types of problems that may exist within an image are from a wide range of issues related to improper photography techniques, such as bad lighting, or out of focus images, to encoding issues which causes unwanted artifacts such as what occurs in a JPEG-2000 image. There have been numerous studies and methods proposed for the detection of poor quality images. However, the use of deep learning for the purposes of image quality detection is limited within the literature.

Deep learning has shown great promise in solving complex tasks by using a black-box approach to the problem. With the advent of advanced deep learning models such as Convolutional Neural Networks (CNN), researchers have been able to produce remarkable results when it comes to the field of image recognition. Handwriting detection and simple object recognition have become almost common place within the computer vision field. These are tasks that only a short time ago were thought of as "cutting edge" yet today, these tasks can be performed with ease through the use of deep learning toolkits such as Google's TensorFlow and Microsoft's CNTK and other frameworks built on top of these packages such as Keras.

In this paper, a study is performed to assess the viability of using deep learning and CNNs in particular to classify images into six categories of bad lighting, Gaussian blur, motion blur, JPEG 2000, white-noise, and high quality reference images which are subjectively considered to be of good quality. To accomplish this task, a new dataset of images has been constructed from newly taken images, and datasets from two previous studies. The first of these datasets was used to asses the aesthetic quality of photos using machine learning techniques (Tang et al., 2013). The second dataset was used by Sheikh et al. (2005) to classify images based on different qualitative criteria. Not all images from these two datasets are used in this study since some of the images did not lend themselves to this particular task. In addition to these two datasets, additional images which were captured using a Sony Alpha 6000 camera which had been manually defocused to produce a Gaussian blur effect were included. Furthermore, the reference images were modified using software to produce a motion blur effect. Unlike previous works, we produce a general approach capable of determining the overall quality of images.

The rest of this paper is laid out as follows. In section 2 preliminary materials will be presented to introduce the types of image quality issues and their causes, and a short literature review to introduce previous works in this field, as well as a brief introduction to Convolutional Neural Networks, TensorFlow and Keras. Section 3 gives the specifics of our newly created dataset as well as methodologies we used to obtain the images. Section 4 will introduce the methodology of the proposed model. Section 5 will analyze the results of the experiments. Section 6 analyzes the application of the proposed model to self-driving cars. Finally, section 7 provides concluding remarks and future

steps. The code for the experiments is provided as a Jupyter notebook which is available for download.

2 Preliminary Materials

2.1 Image Quality Issues

There are many reasons for which a photograph can be considered as having poor quality. In this study, we have focused on five of these categories, Bad Lighting, Gaussian Blur, JPEG-2K, White Noise, and Motion Blur.

Bad Lighting refers to images which have been shot without adequate light for the timing and aperture of the camera. This can cause images to look dull or dark.

Gaussian Blur is caused by an out of focus camera. This type of blur could be caused by faulty auto focus mechanism on a camera, a lens which is poorly constructed, or an image focused on the wrong subject (Brinded, 2011). Hsu and Chen (2008) describe the types of image quality and blur problems that may exist within an image. Figure 1 shows a sampling of images with Gaussian Blur.

JPEG-2K The JPEG-2K image format was introduced in the year 2000. While there are many advantages to this image compression format, one disadvantage is that the JPEG-2K format is much less content adaptive than the older JPEG format meaning that the image quality can be vastly different given the same bit-rate for different content. This makes it likely to end up with compression artifacts within the image.

White Noise is the exhibition of random white grains throughout an image. This noise is often caused by film grain, various sensors and circuits such as CCDs in digital cameras and detectors in a scanner, or it could be caused by the communications channel or signal quantization (Liu and Lin, 2013).

Motion Blur is caused by taking an image were the subject is not stationary in relation to the camera. This can be cause by either a shaking or moving camera, or it could be cause by a subject which is moving.

2.2 Current Literature

2.2.1 Learning Based Approaches

Darunga and Konik introduced a neural network based approach for the analysis of blurry regions within photographs in order to extract meta-data context



Fig. 1 Sample Gaussian Blur Images

from the images (Da Rugna and Konik, 2003). This method relies on passing segments of photographs to a multi-layer neural network or other machine learning based algorithm which can then analyze the blur present. This approach does not aim to determine whether or not an image is of high quality, it only analyzes the blur that is present in each region.

Liu et al. (2008) used a combinational method which used Local Power Specturm Slope, Gradient Histogram Span, and Maximum Saturation to first detect blurry images, then a Bayes classifier is trained to detect the type of blur by training on patches of images to make the process more efficient. Based upon similar ideas, Su et al. (2011); Gu et al. (2015) were able to obtain improved results. While this approach is able to determine the type of blur, it is not able to determine if an image is blurry on its own.

Yang et al. (2018) have proposed a method for focus quality measure using a deep learning approach for images taken from microscopes. They used a dataset of 384 in-focus Hoechst stain images of U2OS cells and used a synthetic de-focusing algorithm to produce out of focus images. This approach however does not look at the overall quality of an image, it focuses on determining if there is Gaussian blur present on particular segments of images.

Bianco et al. (2018) used a CNN to produce an image quality score by average-pooling the scores predicted on multiple sub-regions of an image. The authors used a CNN that was originally trained to discriminate 1,182 visual categories fine-tuned for category-based image quality assessment tasks. The CNN is used to extract features which were then sent to an SVR to predict

the quality score. The authors state that they were able to achieve a 0.91 Linear Correlation Coefficient with human subjective scores. This work does not attempt to make a categorization of images into poor quality categories.

Previous studies mostly focused on performing a binary classification for one image quality problem at a time. However, the methodology presented in this study is able to detect motion blur, Gaussian blur, poor lighting, white-noise, and JPEG-2000 compression errors in an image. To our knowledge at the time of this writing, there are no existing works which perform general classification of images based on image quality problem category.



Fig. 2 White Noise

2.2.2 Non-Learning Based Approaches

Non-learning based approaches for detecting blurry images have been used for many years. While these approaches produce fairly good results in detecting blurriness, they are not capable of categorizing an image into different poor image categories, and are not capable of determining if an image has other quality problems such as white noise or compression artifacts. We have however included the previous work using these non-learning based approaches for completeness.

Many developed blurred image detection methods are based on edge sharpness information. Marziliano et al. (2002) proposed a non-reference blur metric analyzing the spread of the edges in an image. Chung et al. (2004) proposed a non-parametric image blur image based on edge analysis obtained by combining standard deviation of the edge gradient magnitude profile and the value of edge gradient magnitude using a weighted average. Tong et al. (2004) developed a new blur detection scheme based on the edge type and sharpness analysis using Harr wavelet transform. In addition to detect the blurred images, this method was able to determine the extent of blurriness.

In 2008, Tsomko et al. (2008) proposed a new measure based on computing the prediction residue of neighboring pixels in images and computing the variance to measure the blurriness without reference. Su et al. (2011) proposed an automatic image blurred detection and classification technique based on a new blur metric, Singular value feature, to detect the blurred region of an image. Also, they classified the type of blurred regions into defocus blur and motion blur analyzing the alpha channel information. Liu et al. (2008) developed a blur detection methods based on image patches, making region-wise training and classification in one image efficient. This method was also able to recognize the blur types for the detected regions using several blur features modeled by image color, gradient, and spectrum information.

Recently, Golestaneh and Karam (2017) developed a method to compute blur detection maps based on a novel High-frequency multi-scale Fusion and Sort Transform (HiFST) of gradient magnitudes.



Fig. 3 JPEG 2000 Compression Artifacts

3 Dataset

To train a model that could be generalized for a large number types of photos, there was a need for a diverse dataset containing images of different subjects and taken in various lighting conditions. Since such a dataset was not readily available, a new dataset was created from the combination of images from the dataset created by Tang et al. (2013) and a set of blurry images that were created specifically for this study. The blurry images were created using a Sony Alpha 6000 camera by manually defocussing the camera. The aperture, timing, and ISO were set automatically by the camera and vary for each image. The images were then resized prior to processing through python for performance reasons. The dataset contains a total of 125 out of focus images and 125 high quality images as classified in the prior publication (Tang et al. 2013). Motion blur images were produced by using the OpenCV Python library and performing a motion blur operation on the reference images.

The Reference images were taken from the images labeled as high quality by Tang et al. (2013) which were selected to ensure clarity and lack of blurry elements. Note that some of these images may contain out of focus or motion blurred sections, but those types of blurriness have been judged to be desirable. There are also images which are from the dataset by Sheikh et al. (2005) which where modified to produce the JPEG 2000 and white noise images.



Fig. 4 Clear Photo 2

4 Methodology

4.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are deep artificial neural networks (ANNs) applied primarily to classify images, cluster images by similarity, and perform object recognition within scenes. CNN consists of convolutional and sub-sampling layers followed by one or more fully connected layers. The architecture of CNN is designed to take advantage of the 2D structures of an input images. In addition, compared to fully connected networks, CNNs are easier to train and have fewer parameters. To train and test CNN model, each input image will pass through a series of convolution layers and pooling for feature learning. Finally, an activation function such as Softmax, Sigmoid, or ReLU is applied to classify an object. Along with other advanced machine learning algorithms, CNNs have become fundamental to the field of computer vision. In our approach, we build and train a CNN with 3 convolutional layers to classify images into the six predefined categories. We show that our approach is capable of achieving relatively high accuracy with just a limited dataset used for training, and furthermore, we analyze and show that for all six classes, we can achieve high specificity and relatively high sensitivity.

4.2 TensorFlow

Tensor flow is an open source package created using Python which is intended for use with deep learning. The package is able to be used not only for regular fully connected neural networks but also has extensions which allow it to be used for building complex networks such as CNNs and Recurrent Neural Networks. In order to increase the training performance of large networks, TensorFlow also has a variant which is able to use the massive multi-core capability of modern GPUs to accelerate the training process. We used an NVIDIA Quadro K2200 GPU for faster training.

4.3 Our Convolutional Neural Network

The network consists of three convolutional layers with a Softmax activation function followed by a fully connected layer with a sigmoid activation function and finally there is a 6 node output layer with a sigmoid activation function which outputs the 6 desired classes. The first convolutional layer accepts an array of 256×256 images and applies 32, 5×5 filters to produce 32, 252×252 feature maps. We start with the larger filter size since it has been shown by Ahmed et al. (2020) that a larger filter size in the upper layers produces the best results for image classification tasks. The feature maps are then normalized, a Softmax activation function is applied and a 2×2 Max Pooling is performed to obtain 32, 126×126 images. The second convolutional layer applies 64, 3×3 filters to produce 64, 124×124 feature maps As done with



Fig. 5 Dark/Bad Lighting

the first convolutional layer, the feature maps are then normalized, a Softmax activation function is applied and a 2×2 Max Pooling is performed to obtain 64, 62×62 images. This is run through a 3rd and final 3×3 convolutional layer with batch normalization and max pooling which produces 128 feature maps. The 128 feature maps are then flattened into an array of 73,856 parameters. A 20% dropout is performed before these parameters are passed into a fully connected layer (Dense in TensorFlow) with 64 nodes and a Relu activation function. Another dropout is performed with a drop percentage of 20%. This aggressive dropout is intended to keep the network from over-training on the training set and allowing for it to generalize better when applying to other types of images. Finally there is an output layer with 6 nodes with a sigmoid activation function to which all 64 nodes from the previous layer are connected.

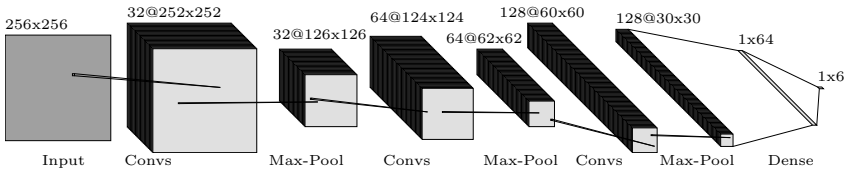


Fig. 6 Convolutional Neural Network Architecture

There are a total of 7,469,145 parameters in the network with 7,468,559 trainable parameters. Table 1 shows the layers and the parameters for each layer and Figure 6 shows the architecture of our CNN.

Table 1 Layers and Parameters in Proposed CNN

	Layer Type	**Output Shape**	**No. of Parameters**
1	conv2d_1 (Conv2D)	(None, 252, 252, 32)	2,432
2	batch_normalization_1	(None, 252, 252, 32)	128
3	activation_1 (Activation)	(None, 252, 252, 32)	0
4	max_pooling2d_1 (2×2)	(None, 126, 126, 32)	0
5	conv2d_2 (Conv2D)	(None, 124, 124, 64)	18,496
6	batch_normalization_2	(None, 124, 124, 64)	256
7	activation_2 (Activation)	(None, 124, 124, 64)	0
8	max_pooling2d_2 (2×2)	(None, 62, 62, 64)	0
9	conv2d_3 (Conv2D)	(None, 60, 60, 128)	73,856
10	batch_normalization_3	(None, 60, 60, 128)	512
11	activation_3 (Activation)	(None, 60, 60, 128)	0
12	max_pooling2d_3 (2×2)	(None, 30, 30, 128)	0
13	flatten_1 (Flatten)	(None, 115,200)	0
14	dropout_1 (Dropout)	(None, 115,200)	0
15	dense_1 (Dense)	(None, 64)	7,372,864
16	batch_normalization_4	(None, 64)	256
17	activation_4 (Activation)	(None, 64)	0
18	dropout_2 (Dropout)	(None, 64)	0
19	dense_2 (Dense)	(None, 6)	390
20	batch_normalization_4	(None, 6)	24
21	activation_5 (Activation)	(None, 6)	0

4.4 Our Implementation

Our program can be split into four steps. First the images are loaded and processed. Next, we split the data into three segments for training, validation, and testing. This is followed by training the network in a two step approach by using a fast learning rate for the first 50 epochs and then reducing the learning rate and using an early stopping approach to minimize the validation loss. Finally, The network is tested against the test dataset to determine the accuracy of the network. The rest of this section is dedicated to describing the implementation methodology of the aforementioned steps. All source code and images are available for download from the GitHub provided in the references of this publication (Golchubian et al., 2020).

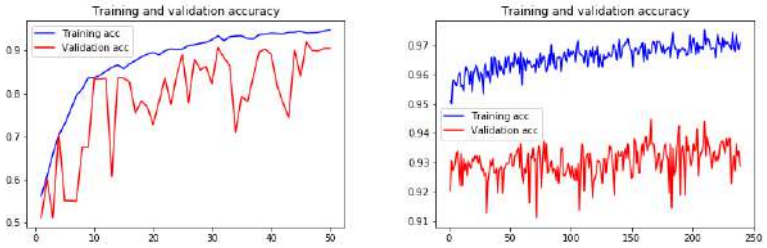
4.4.1 Loading Data

The images for our dataset are labeled by being placed in separate folders. In order to read the data we use Python to load the images from each directory

into a distinct array. We then process the images to produce an image that is 256×256 and that image is stored as a 3 channel image within a data frame. A labels array is created for the images from each directory with 0 indicated for the clear images and 1 for the out of focus images. All images and labels are concatenated together and returned as two arrays. Prior to training, the images are split into different sets and randomized to provide accurate results.

4.4.2 Data Splits and Training

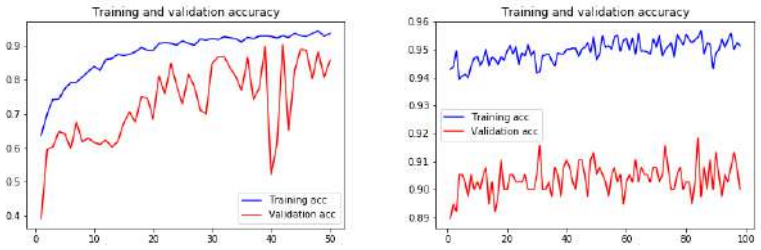
To train the network the dataset is split into a training and a test-validation set. The test-validation set is further split into two equal parts to obtain a test set and a validation set. There are two experiments which are performed in this study with different training, test, and validation ratios.



First 50 Epochs Early stopping refinement

Fig. 7 60/20/20 Training Accuracy History

The first experiment was performed using a ratio of 80/10/10, meaning 80% for training, 10% for validation and 10% for testing. The second experiment used a 60/20/20 ratio. The network is trained for 50 epochs for the 80/10/10 set and 50 epochs for the 60/20/20 split. Figures 7 and 8 show the training accuracy history of the network after these 50 epochs. Note that during these 50 epochs, we also perform data augmentation using Keras' ImageDataGenerator which we have configured to rotate, shift, shear, zoom and flip the images. The model is then further trained using early stopping with the "patience" set to 20 which allows for further training to continue until the validation loss is not improved for 20 epochs. If that condition is reached, then training will be stopped and the previously best network will be saved. During the second training phase, the learning rate is also reduced to 0.0001 which helps in finding the most optimal solution. Figures 7 and 8 show the accuracy trend of the training and validation accuracy for the first 50 epochs and the early stopping refinement epochs.



First 50 Epochs Early stopping refinement

Fig. 8 80/10/10 Training Accuracy History

4.4.3 Testing

All models were tested against the test set which was not used for the training or validation of the model. The results are analyzed by evaluating Type-I and Type-II errors and the accuracy of the predictions. To test the model, the test set was run through the trained model and the predictions were converted to an array of rounded integers.

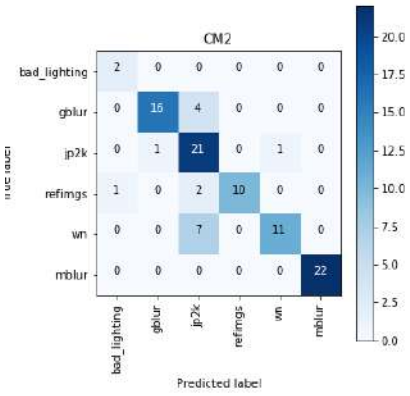


Fig. 9 80/10/10 Split

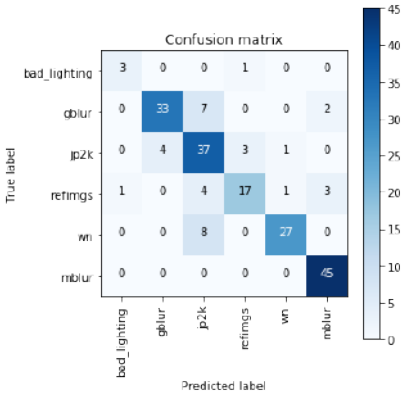


Fig. 10 60/20/20 Split

5 Results

The results show a ratio split of 80/10/10 produced better accuracy than the 60/20/20 split. This suggests that with additional data a higher accuracy can be achieved. The highest accuracy was obtained using the 80/10/10 split which

reached an accuracy of 81.6%. Because many of the images in the dataset were actually in the JPEG format, there is a higher than normal rate of images which are incorrectly classified as having JPEG 2000 compression errors. This may however be correctly classified since these images may have compression artifacts which we were not able to detect. The 60/20/20 split produced an accuracy of 77%.

Figure 9 shows the confusion matrix for the 80/10/10 split. We can observe that the most common mistake for the algorithm is to classify images as having JPEG 2000 compression errors. As mentioned earlier this is to be expected since we are using the JPEG format for many of the images in the data set. Figure 10 shows the confusion matrix for the 60/20/20 experiment and like the first 80/10/10 split the results show the most common mistake is a misclassification of items as JPEG 2000.

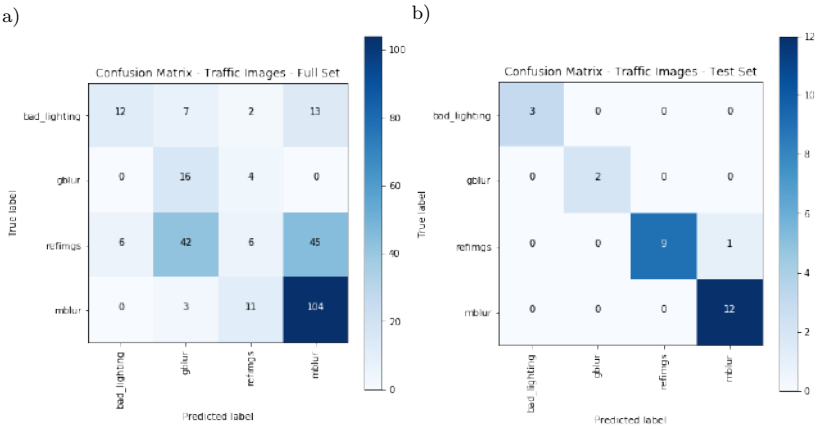


Fig. 11 Confusion Matrix - Traffic Images

5.1 Additional Analysis

Sensitivity, also known as the true positive rate, measures the proportion of the positive samples which are classified as that sample. For example, of the 20 images which had been labeled with Gaussian Blur, 16 were identified as having Gaussian Blur by the network. This produces a Sensitivity of 0.8, which is saying that 80% of images with Gaussian Blur were identified as having Gaussian Blur. In other words, we can identify images with Gaussian Blur 80% of the time.

Specificity, also known as the true negative rate, measures the proportion of the samples which do not have a certain attribute as not having that attribute.

For example, of the 78 samples labeled as something other than Gaussian Blur, 77 were identified as something other than Gaussian Blur and 1 was incorrectly classified as having Gaussian Blur. This produces a specificity of 0.99. This signifies that we can, with a very high accuracy, identify samples that do not have Gaussian Blur.

To get a better understanding of the performance of the classifier for each of the classes, we show the Sensitivity and Specificity for each of the six classes in Table 2. Sensitivity and Specificity are defined by equations 1 and 2, respectively. The results show the 80/10/10 split produces reasonably high sensitivity for all classes and very high specificity. In fact the Specificity of the classifier for the reference images category is 1.0. This suggests that the classifier can be trusted to a high degree when it classifies an image as a reference image.

$$Sensitivity = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{1}$$

$$Specificity = \frac{TrueNegatives}{TrueNegatives + FalsePositives} \tag{2}$$

Table 2 Classifier Performance by Class

Class	Sensitivity		Specificity	
	60/20/20	**80/10/10**	**60/20/20**	**80/10/10**
Bad lighting	0.75	1.0	0.99	0.99
Gaussian blur	0.79	0.8	0.98	0.99
JPEG-2K	0.82	0.91	0.88	0.83
Reference image	0.65	0.77	0.98	1.0
White noise	0.77	0.61	0.99	0.99
Motion blur	1.0	1.0	0.97	1.0

6 Application to Traffic Images

Misclassification of road images is mainly caused by quality issues such as blurriness or degradation (Aghdam and Heravi, 2017). An incorrectly classified object can lead to a misinterpretation of the scenes by the autonomous vehicle. This could cause undesirable driving behavior and could place the passengers and other drivers in danger. A self-driving system equipped with a low quality image detection mechanism, could reduce the likelihood of misclassifying an object captured within the video stream by removing poor quality images. To fulfill this requirement, the model proposed in this study can be utilized to detect poor quality images used for autonomous driving systems. In order to determine how well our network performs when applied to self-driving cars, we have selected images from the German Traffic Sign Recognition Benchmark (GTSRB) (Stallkamp et al., 2012). The selected images with the worst lighting and worst Gaussian blur have been manually labeled and the OpenCV library

on Python was used to introduce a random level of motion blur to the reference images to produce a dataset containing labeled images in four categories.

6.1 Initial Results

Applying the trained network, without modification, resulted in an accuracy of 50%. The majority of incorrect classifications are those reference images which were either classified as motion blur or Gaussian blur. Figure 11a shows the confusion matrix for the full traffic dataset. We visually observed the reference images which were incorrectly classified to determine the cause of the misclassification. Figure 12 shows the reference images which were classified as bad lighting, Gaussian blur, and motion blur. Upon inspection, it is noticeable that the images which were classified into one of these 3 categories are actually of poor quality and it could be argued that the majority of these images were classified correctly. This suggests that while the model may correctly have detected some degree of the attributes associated with poor image quality, the sensitivity of our classifier to these attributes is too high; hence, to make our model useful for the purposes of reducing error rates in the traffic sign classification, we must re-calibrate our model to account for this variability. This can be accomplished utilizing a technique known as transfer learning.

6.2 Transfer Learning

The transfer learning process begins with loading the developed CNN model which was previously trained to classify images into the six categories of bad lighting, Gaussian blur, motion blur, JPEG 2000, white-noise, and high quality reference images. According to previous studies, misclassification of road images is caused by poor quality images that are either blurry or have very poor lighting (Aghdam and Heravi, 2017). Therefor, for this task only the categories of bad lighting, Gaussian blur, motion blur and reference images are considered. First, all layers are set as not trainable. Then the last 3 layers were removed and a dense layer with 4 neurons was added. This allows for the model to be trained more quickly since only 260 parameters need to be learned. The dataset is then split into three segments of 80% for training, 10% for validation, and 10% for test. The network was trained for 50 epochs and additional training was then performed utilizing early stopping which stopped at 289 epochs. Each epoch completed within one second, and the total training task completed in 5 minutes. This retraining process accomplishes two objectives, 1) changing the number of classes from 6 to 4, and 2) allowing for the network to tune it's sensitivity for the given task. The resulting network was able to achieve a 96% accuracy on the test dataset. This shows that while the exact calibration of the model is dependent on the given task, the feature space learned through the original training process can be applied to different sets of images without loosing accuracy. The confusion matrix for the retrained network is shown in Figure 11b.

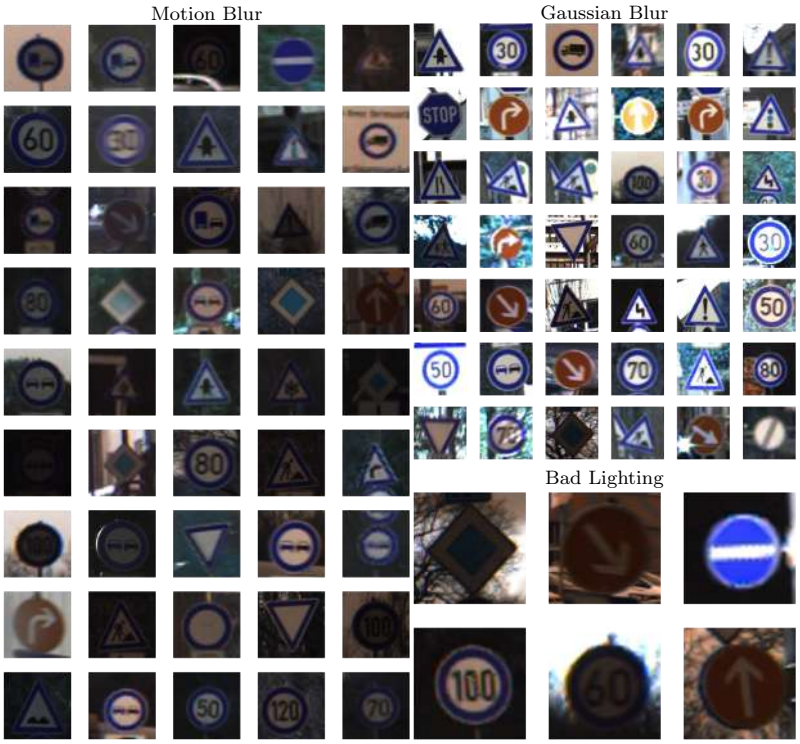


Fig. 12 Incorrectly Classified Reference Images

7 Conclusions and Future Directions

With the rapid advancement of machine learning and the increasing growth of autonomous technologies, it has become even more important that machine learning algorithms be capable of detecting when the results of some action are not desirable. In this paper, results show that our proposed CNN is capable of detecting poor quality images with high accuracy. A new dataset was created to train the developed CNN model to detect images which were manually labeled into the six categories of bad lighting, Gaussian blur, motion blur, JPEG 2000, white-noise, and high quality reference images. Finally the application of the developed model was tested to detect images which may be unsuitable to the sign classification task required by autonomous vehicles. We showed that by simply calibrating the model for a given task through transfer learning we can achieve very high accuracy for the given classification task.

This study was focused on developing a method to detect poor quality images. There are several avenues for future studies to continue upon this work. In a future study we will perform sensitivity analysis to determine how differ-

ences in camera hardware affect the accuracy of the model.

References

Aghdam HH, Heravi EJ (2017) Guide to convolutional neural networks. New York, NY: Springer doi 10:225–226

Ahmed WS, et al. (2020) The impact of filter size and number of filters on classification accuracy in cnn. In: 2020 International Conference on Computer Science and Software Engineering (CSASE), IEEE, pp 88–93

Bianco S, Celona L, Napoletano P, Schettini R (2018) On the use of deep learning for blind image quality assessment. Signal, Image and Video Processing 12(2):355–362

Brinded M (2011) Computer vision methods for detection of blurry photographs. PhD thesis, University of Leeds, School of Computing Studies

Chung YC, Wang JM, Bailey RR, Chen SW, Chang SL (2004) A non-parametric blur measure based on edge analysis for image processing applications. In: Cybernetics and Intelligent Systems, 2004 IEEE Conference on, IEEE, vol 1, pp 356–360

Da Rugna J, Konik H (2003) Automatic blur detection for meta-data extraction in content-based retrieval context. In: Internet Imaging V, International Society for Optics and Photonics, vol 5304, pp 285–295

Golchubian A, Marquez O, Nojoumian M (2020) Photo Quality Classification Using Deep Learning - Dataset and Programming. URL `https://github.com/agolchub/Photo_Quality_Classification`

Golestaneh SA, Karam LJ (2017) Spatially-varying blur detection based on multiscale fused and sorted transform coefficients of gradient magnitudes. In: CVPR, pp 596–605

Gu K, Zhai G, Lin W, Yang X, Zhang W (2015) No-reference image sharpness assessment in autoregressive parameter space. IEEE Transactions on Image Processing 24(10):3218–3231

Hsu P, Chen BY (2008) Blurred image detection and classification. In: International Conference on Multimedia Modeling, Springer, pp 277–286

Liu R, Li Z, Jia J (2008) Image partial blur detection and classification. In: Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, IEEE, pp 1–8

Liu W, Lin W (2013) Additive white gaussian noise level estimation in svd domain for images. IEEE Transactions on Image processing 22(3):872–883

Marziliano P, Dufaux F, Winkler S, Ebrahimi T (2002) A no-reference perceptual blur metric. In: Image processing. 2002. Proceedings. 2002 international conference on, IEEE, vol 3, pp III–III

Sheikh HR, Wang Z, Cormack L, Bovik AC (2005) Live image quality assessment database release 2 (2005)

Stallkamp J, Schlipsing M, Salmen J, Igel C (2012) Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. Neural networks 32:323–332

Su B, Lu S, Tan CL (2011) Blurred image region detection and classification. In: Proceedings of the 19th ACM international conference on Multimedia, ACM, pp 1397–1400

Tang X, Luo W, Wang X (2013) Content-based photo quality assessment. IEEE Transactions on Multimedia 15(8):1930–1943

Tong H, Li M, Zhang H, Zhang C (2004) Blur detection for digital images using wavelet transform. In: Multimedia and Expo, 2004. ICME'04. 2004 IEEE International Conference on, IEEE, vol 1, pp 17–20

Tsomko E, Kim HJ, Paik J, Yeo IK (2008) Efficient method of detecting blurry images. Journal of Ubiquitous Convergence Technology 2(1):pp–27

Yang SJ, Berndl M, Ando DM, Barch M, Narayanaswamy A, Christiansen E, Hoyer S, Roat C, Hung J, Rueden CT, et al. (2018) Assessing microscope image focus quality with deep learning. BMC bioinformatics 19(1):77