Convolution-Neural-Network-based Single-molecule Classification of Circulating MicroRNA Mixtures

Gon Eyal Under the supervision of Dr. Yohai Bar Sinai (The research group includes: Amit Federbush, Jonathan Jeffet, Nadav Tenenboim, assisted by: Prof. Yuval Ebenstein)

August 29, 2023

Abstract

MicroRNA (miR) represents a category of small non-coding RNAs in the regulation of gene expression, and they are gaining recognition as potential indicators of diseases, such as cancer. These miRs can be found in the blood plasma via liquid biopsy analysis. However, extracting information from a single molecule microscopy measurement remains a major scientific challenge. This study presents a new technique to detect and classify multiplexed single-molecule of a selected panel of miRs. The study's procedure (pipeline) maximizes accuracy by encompassing data processing and a suitable classification technique, achieved through an exploration of various Machine Learning (ML) models. The proposed technique relies on a Convolution Neural Net (CNN)[1] Machine Learning-centered detection approach, implemented on convoluted labeled imagery data of miRs. This technique significantly improves classification accuracy in terms of recall and precision[2] compare to earlier methods, and has greater generalizability to a broader spectrum of miR types.

The classification efficacy of this method was tested on differentiating between 3 different miR markers in a mixture of colors triplet. The dataset comprised a total of 655, 141, and 611 labels for each corresponding miR type. In this differentiating test, the accuracy for previously unseen miRs resulted in recall rates of: 87%, 62%, 88%, accompanied by corresponding precision values of: 90%, 82%, 74% or each corresponding miR type. These achievements reveal the potential of our developed methodology as an automated accurate diagnostic tool for various diseases in their early stages.

Keywords: Circulating MicroRNA, Machine Learning, Convolution Neural Network, Single-molecule Classification, Spectral Imaging, Automated Diagnostics,

1 Introduction[3]

MicroRNA (miR) possess the ability to regulate gene expression and are emerging as powerful diseases indicators, notably in conditions like cancer. These biomolecules can be extracted from both tissue and plasma through a simple blood draw. This study builds upon 2 prior experiments in the field of sampling, detection, and classification of multiplexed single-molecule of a selected panel of miRs. The first experiment centered on developing spectral imaging schemes that encode color into a fixed spatial intensity distribution, which was introduced as continuously controlled spectral-resolution (CoCoS) microscopy. Its aim was sampling and distinguishing between circulating miRs. This was achieved by sampling the multi-color fluorescence molecules in gray-scale images [1]. The subsequent experiment served as a proof of concept for an automated classification of miR types and introduced miR Analysis by spectral Classification LEarning (miRACLE) pipeline. This primary pipeline combined principal component analysis (PCA) and support vector machine (SVM) machine learning methods [2]. The methodology developed in this current study is based on a more adaptive procedure, leveraging its neural net architecture. This approach significantly exceeds the prior method's ability to detect and classify miRs in terms of classification accuracy.

The samples utilized in this research are derived from the CoCoS system. Here is a characterization for this experimental framework sampling process:

[1] Machine Learning method that will be further explained in the *Results* chapter.

[2] Terms that will be further explained in the *Results* chapter.

[3] This section introduces briefly the concept of marker miRs via fluorescence molecules. For further read [1]

1. Fluorescently labeled DNA probes are designed to hybridize to the different expression signatures of circulating miR. Only positive miR-DNA hybrids are bound to a glass surface by an Anti-DNA-RNA hybrid S9.6 antibody [3] which solely captures DNA:RNA hybrids.

2. The reporter probes are imaged using an optimized spectral imaging scheme (CoCoS)[1] allowing the detection of the different color combinations simultaneously on CCD screen. Currently, this system processes images of 3 different fluorescent markers, while its final goal is to extend this capability to discerning up to 12 different color combinations

3. The output is a gray-scale image with different spatial dispersion for different color combinations. Each of these combinations should produce 2 blobs in the output image which are separated in different lengths by the frequencies of its colors (due to the dispersion of the prism system).

4. The pipeline introduced in the upcoming chapter aims to automatically classify the different colors combination within a sample, effectively equating to distinct miR types. The primary objective is to optimize recall and precision outcomes in this classification process while maintaining generalizability to a broader spectrum of miR types.

2 Methods

2.1 Data Processing

Image analysis methods have been developed to provide a quantitative assessment of microscopy data. "ImageJ" [4] is one software that can import data, run image processing functions and analytical tools that can be used to extract information from microscopy data. We use this software to manually classify the colors combination to produce our ground truth (GT). We, with the help of this software, can distinguish the different colors in the gray-scale image (as explained in the Introduction part). Consequently, the presumption arises that the task of miRs classification is presumably attainable by an automated computational pipeline. This software has many extensions, one of which is: "ThunderSTORM" [5] This plugin enables us to detect and localize single molecules with methods such as PALM (photo-activated localization microscopy) and STORM (stochastic optical reconstruction microscopy).

The markers are single-molecule fluorescent tags that have a very low signal, meaning that imaging at higher speeds is limited by the need for a good signal-to-noise ratio, with longer exposures needed to collect enough signal for a good image [6]. This is restricted by the weak and unstable molecular binding with these fluorescent markers.

The pipeline implemented in this research doesn't rely on any type of denoising technique [7]. Instead, a straightforward linear transformation was applied to the raw samples to normalize them. the ThunderSTORM blob detection algorithm was executed, which enabled us to crop the sample around each blob. These crops were manually classified into 4 categories: 3 for the 3 different miR types and 1 for noise class [4]. These 3 miR classes shall henceforth be referred to as red, green, and blue miRs. 3 different templates were produced from the labeled crops of the miRs, each representing a different class. These templates were calculated as the median of the pixels' value.

We explored the usage of 2 different techniques to process the normalized samples, simplifying the subsequent classification problem:

- Cross-correlation - our cross-correlation was between each of the obtained templates (noted as A_k) and the examined sample (noted as B)[5]:

$$(A \star B)[i,j,k] = \sum_m \sum_n A_k(i+m, j+n) B(m,n) \tag{1}$$

  Every cross-correlation result was then stored as a different RGB channel, such that the cross-correlation with the red miR template was assigned to the red channel, and equivalent allocations were applied to the green and blue templates (as demonstrated in Fig. 2). Our hypothesis anticipated that for a pair of blobs and an identical template, 3 vertically spaced blobs would emerge, with the central one exhibiting the supreme luminescence. This

[4] It's noteworthy that, since each miR is represented by a pair of vertically spaced blobs, manual labeling was done only on the upper blob in each pair (utilizing its typical stronger signal). Thus, any lower blobs within pairs were excluded from the dataset.

[5] Here, i and j represent the row and the column in the resulting image, while k represents the pixel's channel: red, green or blue.

luminescence difference stems from the multiplication of two sample blobs with their corresponding template counterparts, in contrast to the upper and lower blobs that result from the multiplication of only one sample blob with a template blob positioned contrarily. Similarly, when cross-correlating one miR type with another, the outcome consists of 4 vertically spaced blobs, each radiant value arising correspondingly to the multiplication of a sample blob with a template blob from the template. These hypotheses were later confirmed to be true.

- Cosine similarity - our cosine similarity methodology was based on the cross-correlation essence but we with normalization that considered the cross-correlation result between the sample and a matrix which includes exclusively ones:

$$S_C(A, B) = \frac{A \star B}{\sqrt{\mathbb{1} \star B^2}} \tag{2}$$

Such that $\mathbb{1}$ symbolize the matrix full of ones.

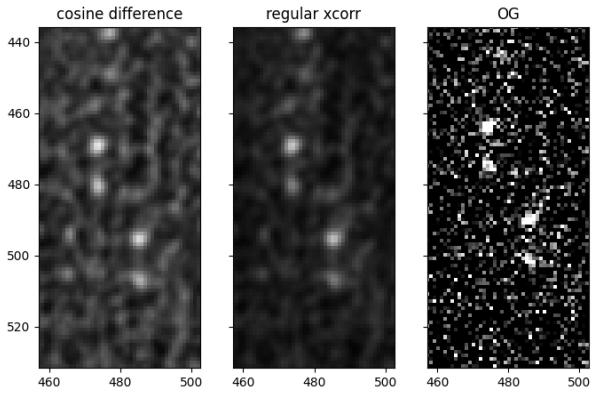


Figure 1: Comparison between the 2 techniques described above in a sample of red miRs with red miR template (from left to right): Cosine similarity, Cross-correlation, and the original sample.

As you can see in the comparison in Fig. 1, Even though the cosine similarity added a normalization to the calculation its result increases the background noise compared to the Cross-correlation technique.

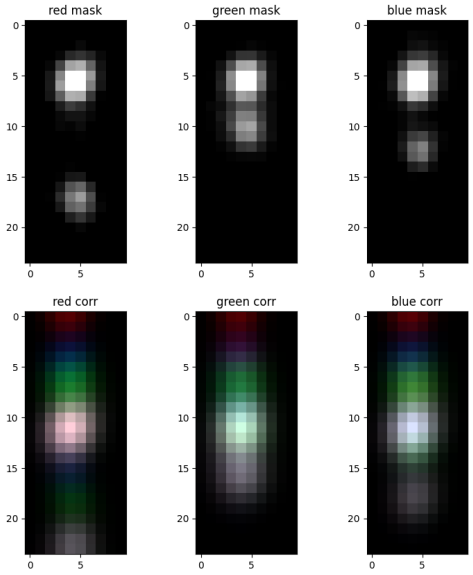


Figure 2: The 3 different templates of miR types and their RGB channels Cross-correlation result. A notable variation in color intensity becomes apparent upon visual inspection of these cross-correlation results. This difference in color signatures underscores the feasibility of employing a CNN to accurately classify these patterns. For additional RGB channels Cross-correlation results of real samples refer to Fig. 14 in the *Appendixes* chapter

2.2 Model Architecture

In the field of machine learning, the division of data into training, validation, and test sets significantly influences model generalization capabilities and performance. In the context of our study, the dataset comprised a total of 655 red, 141 green, and 611 blue miRs labels, alongside 2629 blobs labeled as noise class instances. Our approach involves partitioning a diverse dataset into three subsets: a training set comprising 60% of the data, a validation set encompassing 20%, and a test set constituting the remaining 20%. While adhering to these predetermined ratios, we seek to find the optimal balance model training, hyperparameter tuning, and learning rate tuning. Our preliminary results indicate that such a partitioning strategy promotes a well-robust model, yielding generalized insights into the test set from the training set.

The selection of the LeNet-5 Convolutional Neural Network (CNN) architecture for image classification is driven by its well-established capabilities in effectively and in simplicity extracting features from visual data for a long time. This structure was proposed by LeCun et al. in 1998. LeNet's architec-

ture, characterized by its convolutional, pooling, and fully connected layers, is adept at capturing intricate patterns within images [8]. By employing LeNet-5 CNNs, we aim to exploit their feature extraction, which makes them particularly suited for tasks involving spatial relationships and local patterns in images. Prior to model training, the input data underwent normalization to achieve a mean value of 0 and a standard deviation of 1, thus establishing a standardized foundation for the learning process.

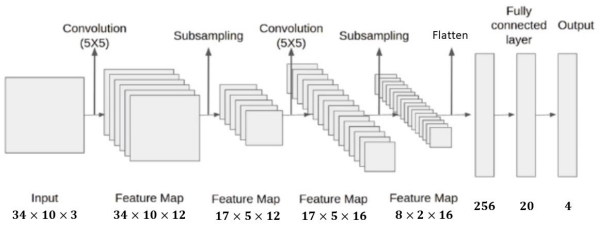


Figure 3: The chosen architecture of LeNet-5, a CNN with 5 layers with learnable parameters, here used for miRs classification. Each plane is a feature map, and each arrow is a mathematical transition between them. Here we have in total 10,952 parameters - all of them are trainable.

The chosen loss function for all of the models' run in this study was categorical cross-entropy. This loss function holds significant utility in machine learning, especially for addressing multi-class classification problems. It measures the difference between the predicted probability distribution and the GT distribution. The lower the categorical cross-entropy, the better the model is at predicting the correct class. Its formula goes as follows:

$$H(y,p) = -\sum_{i} y_i \log(p_i) \tag{3}$$

where H is the entropy function, y is the GT distribution, p is the predicted probability distribution, and i is the index of the class [9]. An often-utilized activation function in multi-class classification scenarios is the softmax function[6] This activation was employed on the scores prior to the computation of the cross-entropy loss.

2.3 Tuning The Model

2.3.1 Hyperparameters Tuning

Effective hyperparameter tuning plays a key role in enhancing the performance of machine learning models. The process involves systematically adjusting the model's hyperparameters to find the configuration that yields the best results on validation data. By fine-tuning these hyperparameters, we aim to balance underfitting and overfitting, ultimately leading to improved generalization on unseen data (as illustrated in Fig. 4)[10][11]. Our study explored the impact on model convergence and predictive accuracy, in terms of recall and precision, of various hyperparameters: first, second, and third layer filter size, convolution kernel size, and max pooling size. Through rigorous experimentation, we were able to identify hyperparameters that enhance our machine learning algorithms.

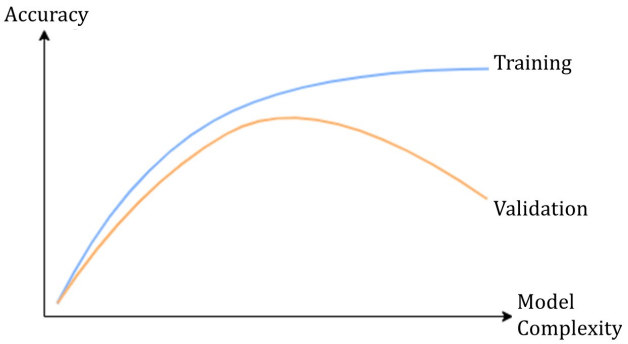


Figure 4: Expected behavior of model's accuracy over the training and validation sets as a function of the model complexity. The overfitting regime starts at the local maximum point of the validation set.

In our study, we systematically adjusted each hyperparameter of the model while keeping the remaining parameters fixed. Through this systematic process, we successfully pinpointed the apex of accuracy on the validation set, employing a methodology akin to coordinate descent [12]. An illustrative demonstration of this process for the initial layer's filter size is presented in Fig. 5. During this experiment, other parameters retained their values: filter2=16, filter3=30, convolution kernel size=(5,5), and max pooling size=(2,2). Aligning with our theoretical ex-

[6]The softmax activation function goes as follows:
$\sigma(\vec{z})_i = \frac{\exp z_i}{\sum_j \exp z_j}$ such that: $\vec{z}$ is the input vector, and σ_i is the softmax output for the i class.

[7]For additional hyperparameter adjusting graphs refer to *Appendixes* chapter

pectations, we observed the emergence of the overfitting phenomenon, prompting a decline in precision rates for both red and blue miRs, approximately at filter1=50 [7].

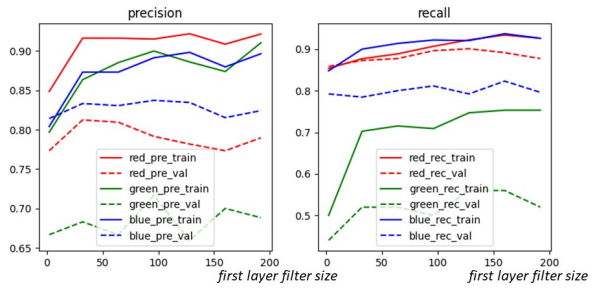


Figure 5: The model's accuracy assessed through precision and recall across both the training and validation sets in relation to the first layer filter size. As anticipated, the accuracy progressively improves until the model's complexity threshold is reached, initiating the overfitting regime - evident here as a decline in precision values for the red and blue miRs within the validation set.

2.3.2 Method of Optimizing & Learning Rate Tuning

We examined the model architecture described in Fig. 3, estimating its accuracy through the implementation of two distinct iterative optimization approaches: Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (AdaM). This comparative analysis was conducted in proximity to their respective nominal learning rates.

- SGD - This technique estimates the true gradient of a function (assuming appropriate smoothness properties) from a randomly selected subset of the data [8]. The weight vector $\overline{\theta}n+1$ of the current iteration is updated through the equation:

$$\overline{\theta}_{n+1} = \overline{\theta}_n - \eta \overline{\nabla}_\theta f_n \tag{4}$$

  where θ_n denotes the weight vector from the preceding iteration, f_n represents the loss function of the current iteration, and η represents the learning rate - influencing the magnitude of the iterative weight vector readjustments.

- AdaM - This technique is a stochastic gradient descent method rooted in the adaptive estimation of first-order and second-order moments. The weight vector of the current iteration is updated through the set of equations:

$$\overline{\theta}_{n+1} = \overline{\theta}_n - \eta \cdot \frac{\hat{m}_{n+1}}{\sqrt{\hat{v}_{n+1}} + \epsilon} \text{ such that:}$$
$$\hat{m}_{n+1} = \frac{\beta_1 \cdot \hat{m}_n + (1-\beta_1) \cdot \overline{\nabla}_\theta f_n}{1-\beta_1^n} \tag{5}$$
$$\hat{v}_{n+1} = \frac{\beta_2 \cdot \hat{v}_n + (1-\beta_2) \cdot (\overline{\nabla}_\theta f_n)^2}{1-\beta_2^n}$$

  In which $\hat{m}_{n+1}$ represents the bias-corrected first-moment estimate, and $\hat{v}_{n+1}$ represents the bias-corrected second raw moment estimate, both of the current iteration. Good default settings for nominal machine learning problem are $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ [13]. All operations on vectors are element-wise. With β_1^n and β_2^n we denote β_1 and β_2 to the power n. In summary, this optimization approach is computationally efficient, has little memory requirement, is invariant to diagonal rescaling of gradients, and showcases particular suitability within contexts typified by extensive data and parameter dimensions.

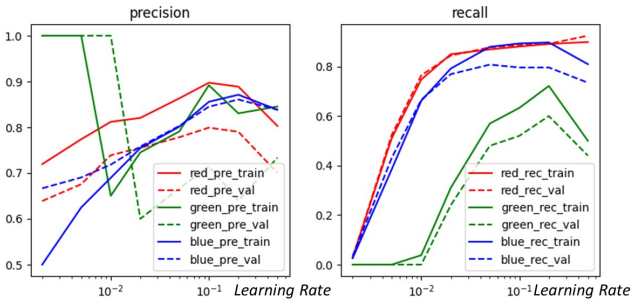


Figure 6: An example of SGD model's accuracy progression across the training and validation datasets in relation to the learning rate. The presented aligns with expected behavior: the accuracy exhibits enhancement for lower learning rates until a distinct optimal threshold, beyond which the optimal learning rate is surpassed.

3 Results

In this study, we examined the accuracy of our trained model through metrics encompassing precision, recall, and the utilization of the confusion matrix. The latter is a very popular metric in the realm of classification problem-solving. It can be applied

to binary classification as well as to multi-class classification problems, like in our case. The confusion matrix represents the counts derived from predicted and actual classes [14]. In this section, we used a notation in which "P" stands for precision which shows the proportion of correctly classified miRs within a specific class, relative to the total predicted instances associated with that class:

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

where TP stands for True Positive which indicates the number of positive miRs classified accurately, and FP shows False Positive value, i.e., the number of actual negative examples classified as positive.

Likewise, "R" stands for recall which shows the ratio of correctly classified miRs belonging to a specific class, with the total number of miRs truly belonging to the said class:

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

In which FN means a False Negative value which is the number of actual positive examples classified as negative [14].

In order to give a quantitative significance to the values within the cells of the heat map of the confusion matrix, and due to the marked disparity in class sizes that made any cell that is not noise class TP nearly invisible, a normalization scheme was employed. This normalization was executed with respect to the true labels, thus ensuring that each row's summation amounts to 1 [8]. The application of this normalization technique, while pivotal for enhancing the interpretability of the confusion matrix, has the potential to infect a degree of asymmetry into matrices that would otherwise be symmetrical. Closer inspection of the recall and precision values, however, exposes that within the conditions of this study the recall & precision rates are symmetrical:

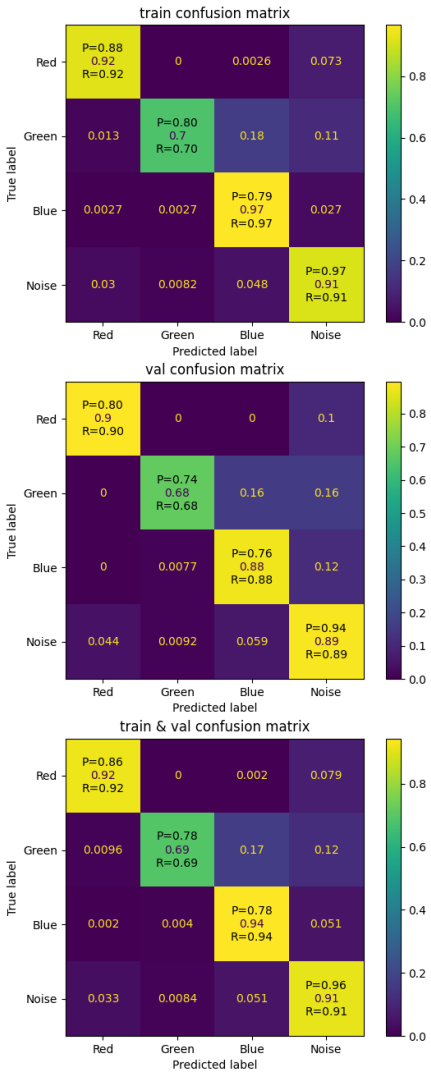


Figure 7: The confusion matrix resulting from training the model on the training set and generalizing its results to the validation set. This matrix was generated during a model run defined by: filter1=12, filter2=16, filter3=30, convolution kernel=(5,5), max pooling=(2,2), and AdaM optimizer with a learning rate of $2 \cdot 10^{-3}$.

In the totality of the experimental landscape, 10 distinct confusion matrices were generated from 10 different runs of the model with these hyperparameters. In each such run, the model's architecture and the partitioning into discrete training, validation, and test subsets were preserved. The sole variability across these iterative instances was the weights initialization of the model. The selection of the iteration number 10, was a deliberate choice driven by the intention to find a model with good generalization abilities over the validation set, while still avoiding overfitting in relation to this subset.

[8] It's easy to see from the normalization definition that these values are, in fact, equivalent to the recall rates

Among this collection, the matrix that exhibited the most promising generalization capabilities (depicted in Fig. 7) was selected for later evaluation on the test set:

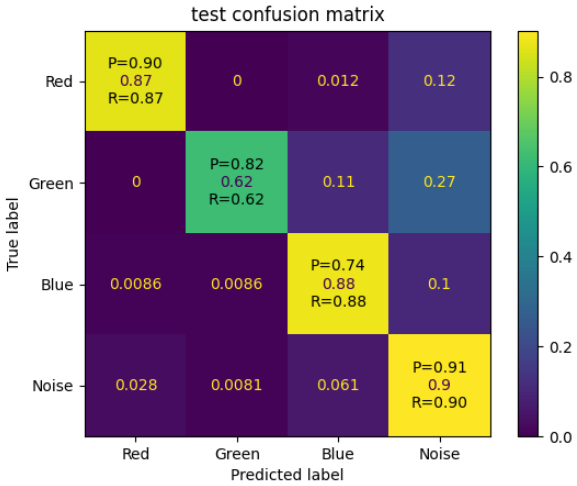


Figure 8: The confusion matrix resulting from training the model on the training set and generalizing its results to the test set. This particular confusion matrix was generated for the same model as in Fig. 7.

The Receiver Operating Characteristic (ROC) curve presents an alternative technique for measuring the accuracy of ML models. Represented graphically, it showcases the true positive rate (TPR) against the false positive rate (FPR) across various classification thresholds. An ideal model's ROC curve traverses from (0,0) to (1,1), implying the precise classification of both positive and negative instances. However, practicality dictates that no model achieves perfection, leading the ROC curve to consistently reside below this ideal line [15]. The ROC curve can be used to compare the performance of different ML models for the same problem. This visual representation holds value in assessing the effectiveness of our ML model in the complex realm of multi-class classification involving miR types:

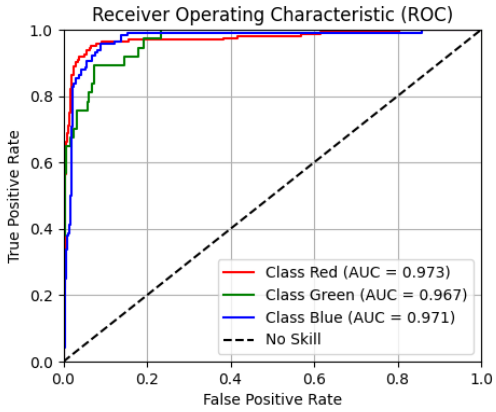


Figure 9: The ROC curved matrix resulting from training the model on the training set and generalizing its results to the test set. This particular ROC curve was generated for the same model as in Fig. 7.

All of the results showcased in this chapter validate the conjecture that this ML classification procedure is well-adjusted for the miR type classification problem within the realm of single-molecule spectroscopy. Moreover, minor adjustments, which will be discussed in the forthcoming *Conclusions* chapter, hold the potential to improve these favorable accuracy outcomes even further.

4 Discussion and Future Work

Anticipated progressions in the system's operational sampling capabilities, leading to decreasing noise signals, promise enhanced classification accuracy. This is achieved both by decreasing the number of noise blobs and by diminishing the background noise. The latter negatively impacted our results, both by directly introducing noise signal into the ML training process and indirectly by increasing the error rate in our GT (produced by manual miR type labeling). In conclusion, our comprehensive pipeline's resilience offers a promising foundation for future expansion, supporting the CoCoS system's goals.

To better comprehend and improve the practicality of the pipeline, two essential tasks remain:

- In this study, the model was trained exclusively on labeled samples representing mixtures of various miR types that we classified. Notably, we didn't train our ML model on miR samples that exclusively contained a single type of miR. This choice mirrors real-world medical scenarios, where samples could manifest as mixtures

of miR types. We initially thought that this would decrease the model accuracy due to further deviation of the cross-correlation result of each class. However, this could be addressed, in a prospective continuation of the study, by initiating the training process with a number of epochs on the alternate labeled database, followed by additional training epochs on the mixture database that we worked with. This approach will provide a larger GT dataset, presumably enhancing the accuracy even further. Additionally, the epochs partition will reduce susceptibility to overfitting on distinct noise signals between the datasets.

- An additional vital extension for the continuation of this study would involve testing the model's classification outcomes on medical samples. Under circumstances where system configurations remain relatively uniform (ensuring a consistent noise signal distribution), this could be achieved without necessitating the re-labeling of new blobs. This further extension would further verify the method developed in this study as highly automated, robust, and efficient for the diagnostic of various diseases such as cancer.

5 Conclusions

Our findings show the ability to accurately detect and classify miRs from 3 different categories. By leveraging the cross-correlation method to process the raw image data, we successfully harnessed the distinct luminous patterns of miR types to generate RGB channels that amplified our classification capabilities. The chosen LeNet-5 CNN architecture proved its robustness in this multi-class classification challenge, demonstrating its adeptness in extracting crucial features from visual data. Rigorous hyperparameter tuning, guided by precision and recall analyses, directs to optimal model performance, revealing its potential for extended application across a broader spectrum of miR types. These achievements reveal the potential of our developed methodology as an automated diagnostic tool for various diseases, notably cancer.

References

[1] Jonathan Jeffet, Ariel Ionescu, Yael Michaeli, Dmitry Torchinsky, Eran Perlson, Timothy D. Craggs, Yuval Ebenstein, "Multimodal single-molecule microscopy with continuously controlled spectral resolution (cocos),"

[2] Jonathan Jeffet, Sayan Mondal, Amit Federbush, Nadav Tenenboim, Miriam Neaman, Jasline Deek, Yuval Ebenstein, Bar-Sinai Yohai, "Machine-learning-based single-molecule quantification of circulating microrna mixtures,"

[3] S. J. Boguslawski, D. E. Smith, M. A. Michalak, K. E. Mickelson, C. O. Yehle, W. L. Patterson, and R. J. Carrico, "Characterization of monoclonal antibody to dna · rna and its application to immunodetection of hybrids," *Journal of Immunological Methods*, vol. 89, no. 1, pp. 123–130, 1986.

[4] S. Hartig, "Basic image analysis and manipulation in imagej," *Current protocols in molecular biology / edited by Frederick M. Ausubel ... [et al.]*, vol. Chapter 14, p. Unit14.15, 04 2013.

[5] M. Ovesny, P. Křížek, J. Borkovec, Z. Svindrych, and G. Hagen, "Thunderstorm: a comprehensive imagej plug-in for palm and storm data analysis and super-resolution imaging," *Bioinformatics (Oxford, England)*, vol. 30, 04 2014.

[6] Jonathan Jeffet, Prof. Yuval Ebenstein, "Teledyne's customer stories: Single molecule dna imaging," 2020. [Online; accessed 24-August-2023].

[7] A. Krull, T.-O. Buchholz, and F. Jug, "Noise2void - learning denoising from single noisy images," 2019.

[8] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[9] H. Song, M. Kim, D. Park, Y. Shin, and J.-G. Lee, "Learning from noisy labels with deep neural networks: A survey," 2022.

[10] Medium contributor Maciej Balawejder, "Overfitting in deep learning," 2022. [Online; accessed 24-August-2023].

[11] M. Bejani and M. Ghatee, "A systematic review on overfitting control in shallow and deep neural networks," *Artificial Intelligence Review*, vol. 54, pp. 1–48, 03 2021.

[12] Wikipedia contributors, "Coordinate descent — Wikipedia, the free encyclopedia," 2023. [Online; accessed 24-August-2023].

[13] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[14] D. K. Sharma, M. Chatterjee, G. Kaur, and S. Vavilala, "3 - deep learning applications for disease diagnosis," in *Deep Learning for Medical Applications with Unique Data* (D. Gupta, U. Kose, A. Khanna, and V. E. Balas, eds.), pp. 31–51, Academic Press, 2022.

[15] Foundational coursesCourse, "Classification: Roc curve and auc," 2022. [Online; accessed 24-August-2023].

6 Appendixes

6.1 Measurements Appendix

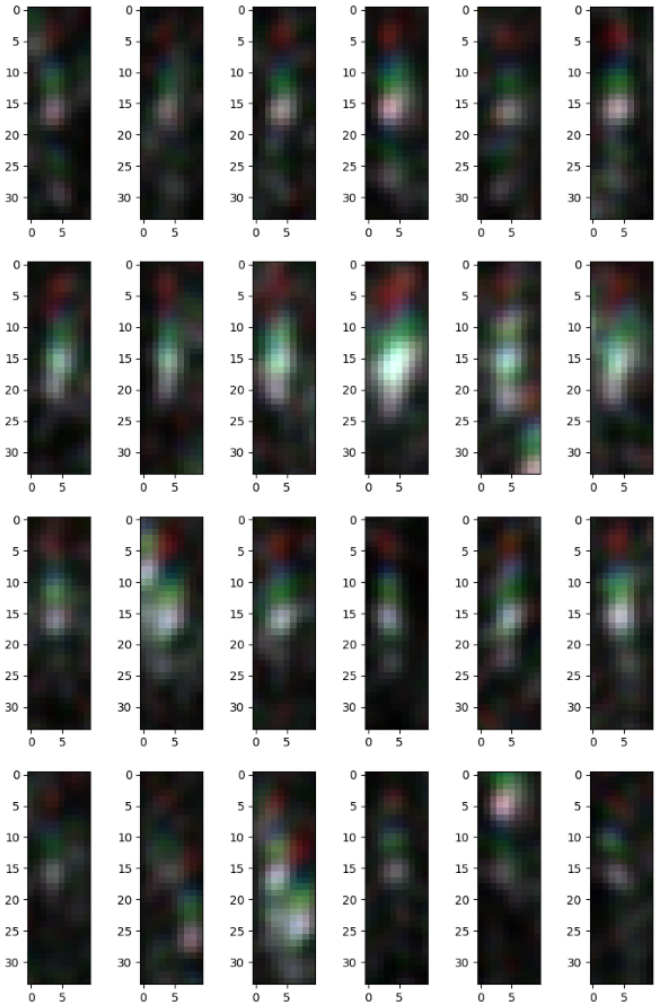


Figure 10: An example of the RGB channels Cross-correlation result of the 4 different classes (from top to button): red miRs, green miRs, blue miRs, and noise.

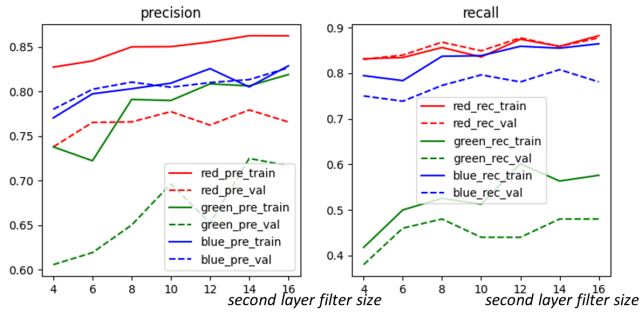


Figure 11: The model's precision and recall across both the training and validation sets in relation to the second layer filter size. As anticipated, the accuracy progressively improves until the model's complexity threshold is reached - evident here as a decline in precision values approximately at $filter2 = 16$.

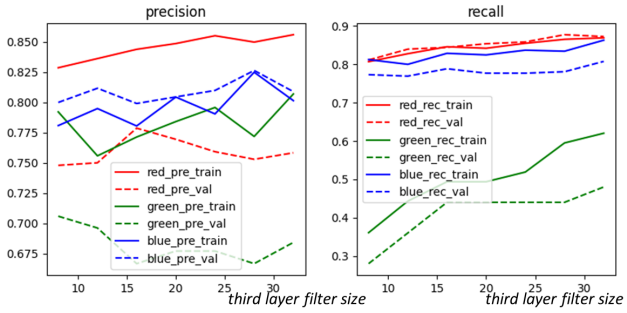


Figure 12: The model's precision and recall across both the training and validation sets in relation to the third layer filter size. As anticipated, the accuracy progressively improves until the model's complexity threshold is reached - evident here as a decline in precision values approximately at $filter3 = 30$.

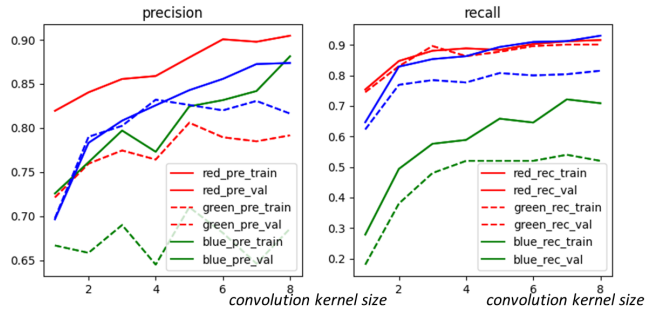


Figure 13: The model's precision and recall across both the training and validation sets in relation to the convolution kernel size. As anticipated, the accuracy progressively improves until the model's complexity threshold is reached - evident here as a decline in precision and recalls values approximately at $kernelsize = 5$.

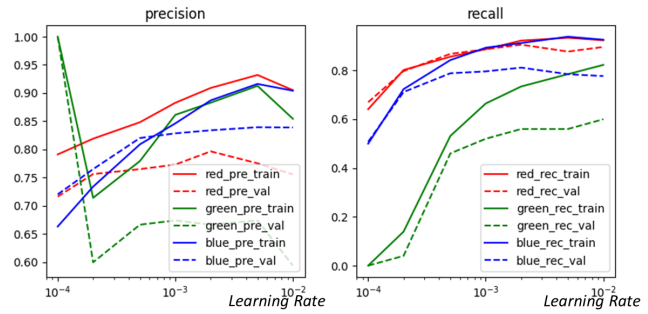


Figure 14: The model's precision and recall across both the training and validation sets in relation to the learning rate in SGD. As anticipated, the accuracy progressively improves until the model's complexity threshold is reached - evident here as a decline in recall and precision values approximately at $LearningRate = 2 \cdot 10^{-3}$.

6.2 Code Appendix

In [1]:
```
import numpy as np
import pandas as pd
import scipy.signal as sp
import matplotlib.pyplot as plt
from pathlib import Path
from importImages import read_tif_from_folder
from images2crops import blobs_to_crops
from skimage import feature
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from tensorflow import keras, config
%matplotlib notebook

np.random.seed(3)
```

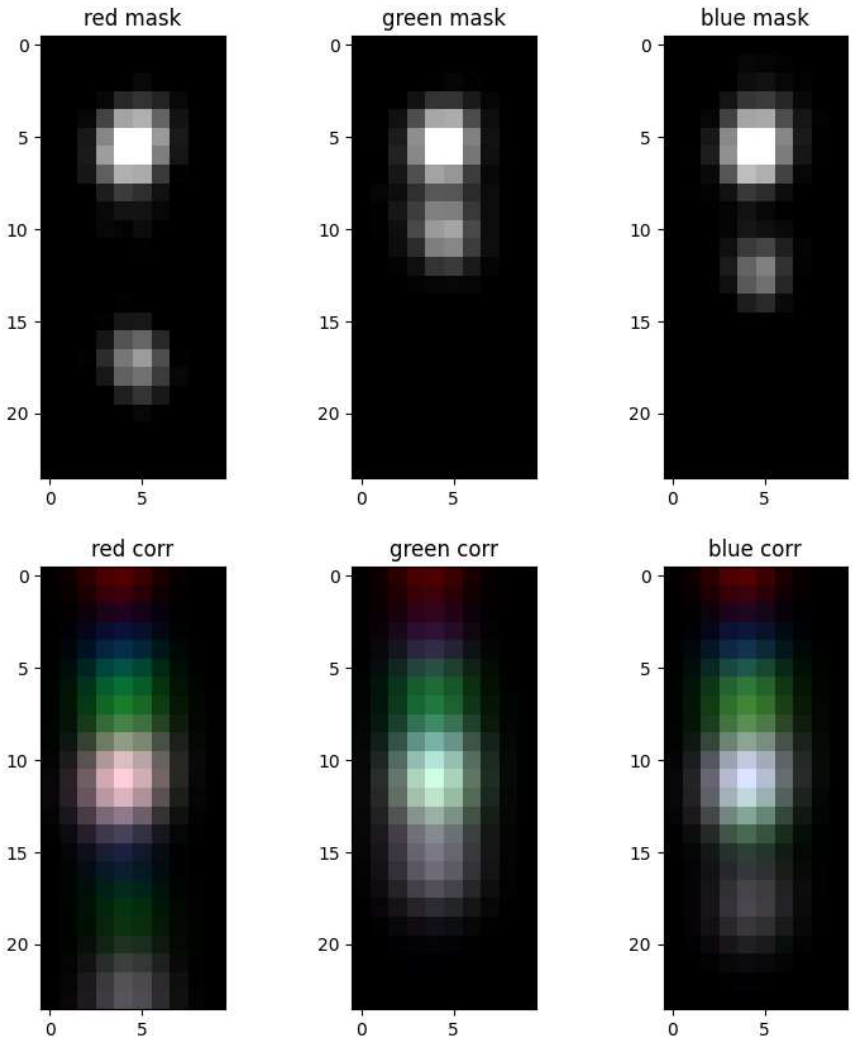
In [2]:
```
def template_matching(img, temp, smart_norm=True, method='LOG'):
    if smart_norm:
        img = img / np.linalg.norm(img)
        temp = temp / np.linalg.norm(temp)
    else:
        img = 255 * img / np.max(img)
        temp = 255 * temp / np.max(temp)

    corr = np.array(sp.correlate2d(img, temp, mode="same"))

    if method == 'LOG':
        blobs_loc = np.array(feature.blob_log(corr, min_sigma=1, max_sigma=7, th

    elif method == 'DOG':
        blobs_loc = np.array(feature.blob_dog(corr, threshold=0.002)) # output i

    return corr, blobs_loc
```

In [3]:
```
def template_matching_normed(img, temp):
    img = img / np.linalg.norm(img)
    temp = temp / np.linalg.norm(temp)

    corr = np.array(sp.correlate2d(img, temp, mode="same"))
    temp_ones = np.full_like(temp, 1)
    img_sqr = np.power(img, 2)
    corr_norm = np.array(sp.correlate2d(img_sqr, temp_ones, mode="same"))
    corr /= np.power(corr_norm, 0.5)

    blobs_loc = np.array(feature.blob_log(corr, min_sigma=1, max_sigma=7, thresh

    return corr, blobs_loc
```

In [4]:
```
def preprocessing (x, contrast_factor = 5):

    # BG_subtracting
    x = np.maximum(x - np.percentile(x, 70), 0)

    # strong dirty light subtracting
    x = np.minimum(x, np.median(x) + contrast_factor * x.std())

    return x
```

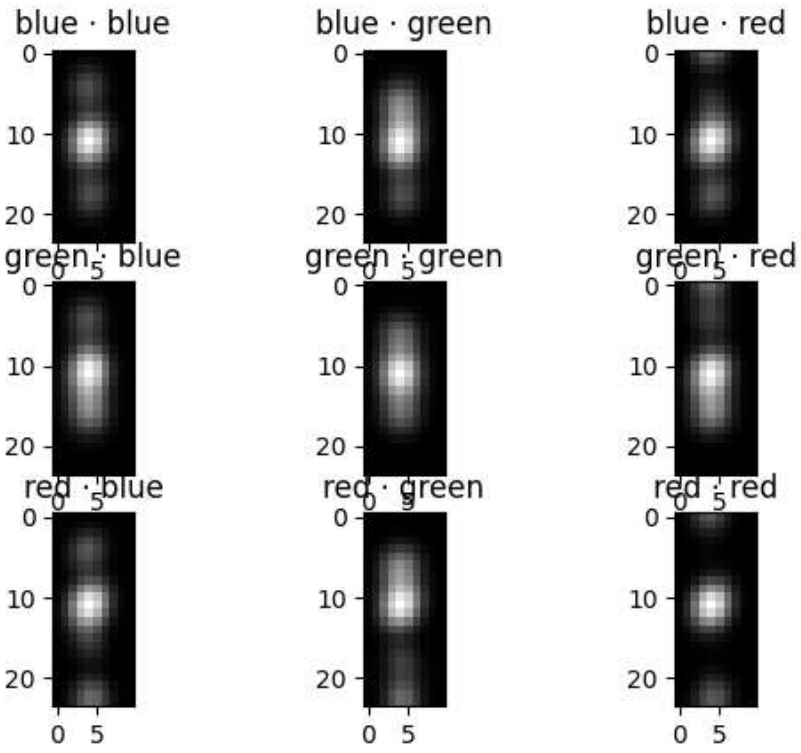
Templates w\ themselves

In [5]:
```
path = 'C:/Users/gongo/OneDrive - mail.tau.ac.il/Studies/Semester A/Research Prc
figure, axis = plt.subplots(2,3, figsize=(9,10))
temp_names = ['red_mir_mask.tif', 'green_mir_mask.tif', 'blue_mir_mask.tif']

for i, mask_name_1 in enumerate(temp_names):
    mat = np.zeros((3, 24, 10))
    mask1 = read_tif_from_folder(path + mask_name_1)
    mask1 = preprocessing(mask1)
    for j, mask_name_2 in enumerate(temp_names):
        mask2 = read_tif_from_folder(path + mask_name_2)
        mask2 = preprocessing(mask2)
        corr, blobs_loc = template_matching(mask1, mask2)
        mat[j] = corr
    axis[0,i].imshow(mask1, cmap='gray')
    axis[0,i].set_title(mask_name_1.split("_")[0] + ' mask')
    rgb_img = np.dstack([mat[j] for j in range(3)])
    axis[1,i].imshow(rgb_img)
    axis[1,i].set_title(mask_name_1.split("_")[0] + ' corr')

figure.savefig('templetes with themselfs rgb.png')
```



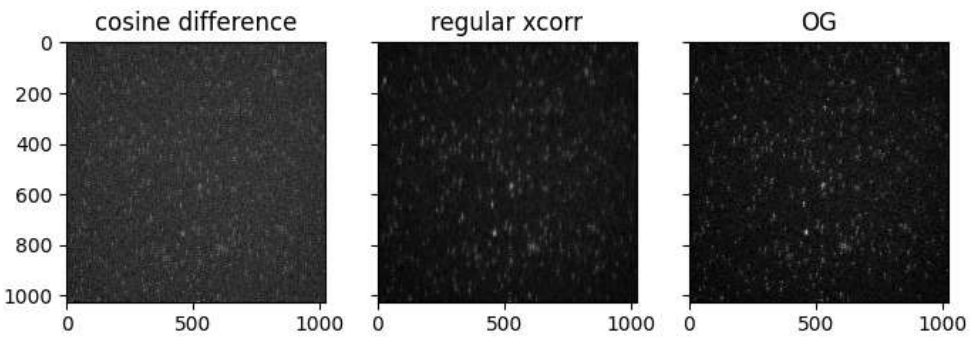
Templates w\ each other

In [6]:
```
path = 'C:/Users/gongo/OneDrive - mail.tau.ac.il/Studies/Semester A/Research Prc
figure, axis = plt.subplots(3,3)
temp_names = ['blue_mir_mask.tif', 'green_mir_mask.tif', 'red_mir_mask.tif']

for i, temp_name1 in enumerate(temp_names):
    for j, temp_name2 in enumerate(temp_names):
        temp1 = read_tif_from_folder(path + temp_name1)
        temp2 = read_tif_from_folder(path + temp_name2)
        temp1 = preprocessing(temp1)
        temp2 = preprocessing(temp2)
        corr, blobs_loc = template_matching(temp1, temp2)
        axis[i,j].imshow(corr, cmap='gray')
        axis[i,j].set_title(temp_name1.split("_")[0] + ' · ' + temp_name2.split(

figure.savefig('templetes with each other.png')
```



With & Without Federbush's Normalization

In [7]:
```
imgs = read_tif_from_folder(path + 'red_mir_3_images.tif')
temp = read_tif_from_folder(path + 'red_mir_mask.tif')

for i, img in enumerate(imgs):

    if i == 0:
        continue

    img = preprocessing(img)
    temp = preprocessing(temp)
    figure, axis = plt.subplots(1,3, sharex=True, sharey=True, figsize=(8,5))
    axis[0].set_aspect('equal')
    axis[1].set_aspect('equal')
    corr1, blobs_loc1 = template_matching_normed(img, temp)
    corr2, blobs_loc2 = template_matching(img, temp)
    axis[0].imshow(corr1, cmap='gray')
    axis[0].set_title('cosine difference')
    axis[1].imshow(corr2, cmap='gray')
    axis[1].set_title('regular xcorr')
    axis[2].imshow(img, cmap='gray')
    axis[2].set_title('OG')

    plt.show()
    break
```



Labaled Mixed Data - going to RGB

In [8]:
```
data_folder = Path('C:/Users/gongo/OneDrive - mail.tau.ac.il/Studies/Semester A/
blobs_loc_by_TS_on_original_img_name = 'mixed miR for gon.csv'
imgs_names = 'validation_mix_NEW_1_1_1.tif'
imgs = read_tif_from_folder(data_folder / imgs_names)
blobs_loc_by_TS_on_original_img = pd.read_csv(data_folder / blobs_loc_by_TS_on_c

masks_folder = Path('C:/Users/gongo/OneDrive - mail.tau.ac.il/Studies/Semester A
masks_names = ['red_mir_mask.tif', 'green_mir_mask.tif', 'blue_mir_mask.tif']
```

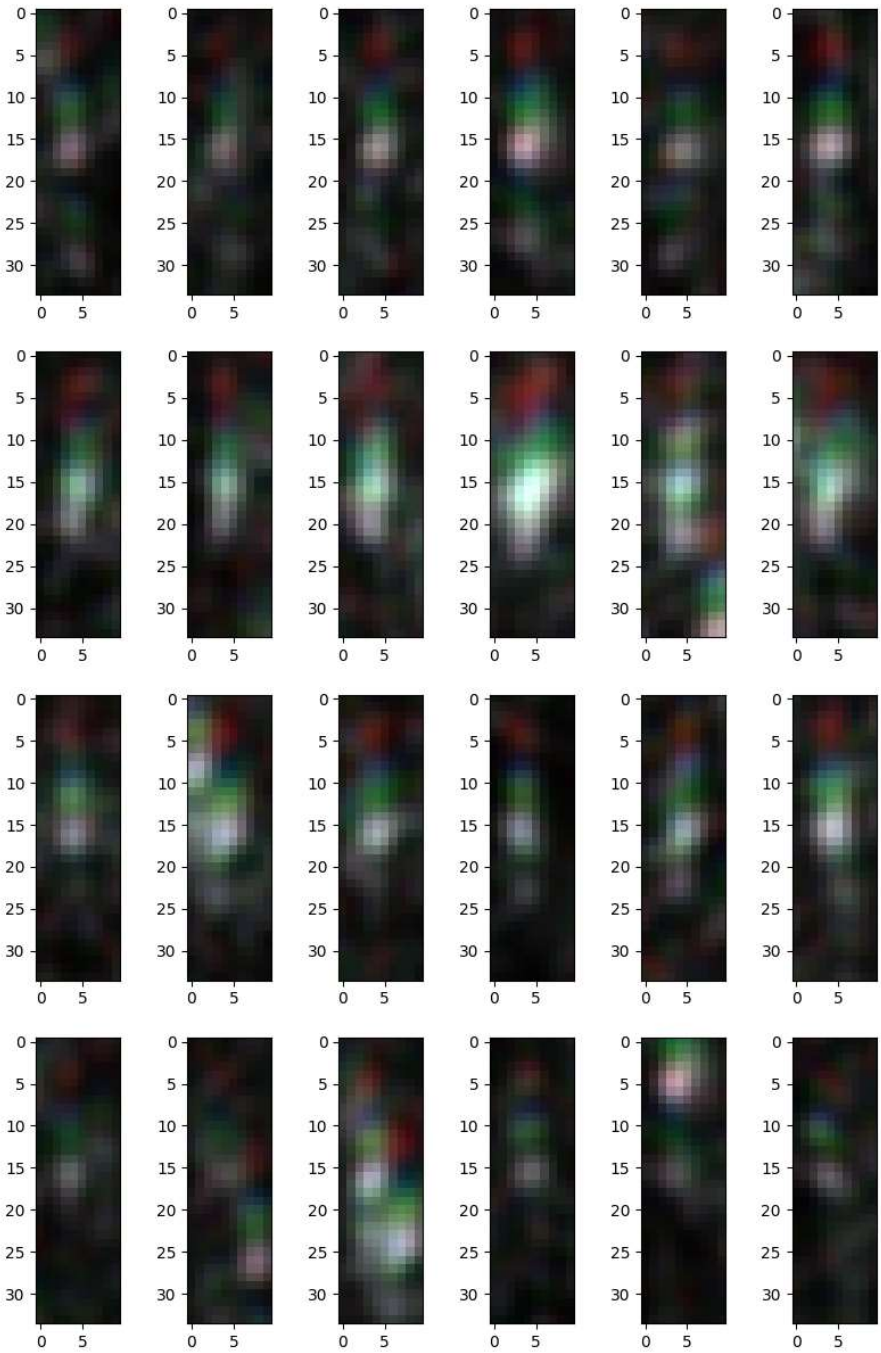
In [9]:
```
# Cross-correlation calculating between the samples and the templates
imgs_len = len(imgs)
rgb_images = np.zeros((imgs_len, 1024, 1024, 3))
for i, img in enumerate(imgs):
    img = preprocessing(img)
    mat = np.zeros((3, 1024, 1024))
    for j, masks_name in enumerate(masks_names):
        mask = read_tif_from_folder(masks_folder / masks_name)
        mask = preprocessing(mask)
        corr, blobs_loc = template_matching(img, mask)
        mat[j] = corr
    rgb_images[i] = np.dstack([mat[j] for j in range(3)])
    print(i, 'out of', imgs_len)


np.save('RGB_corr_validation_mix_NEW_1_1_1.npy', rgb_images)
```

In [10]:
```
# Cropping the images
imgs_rgb = np.load('RGB_corr_validation_mix_NEW_1_1_1.npy')
blobs_loc_by_TS_on_original_img = pd.read_csv(data_folder / blobs_loc_by_TS_on_c
crops, crops_rgb, success_mask = blobs_to_crops(imgs, imgs_rgb, blobs_loc_by_TS_

np.save('crops.npy', crops)
```

```
np.save('crops_rgb.npy', crops_rgb)
np.save('success_mask.npy', success_mask)
```

In [11]:
```
# Load the data
crops = np.load('crops.npy')
crops_rgb = np.load('crops_rgb.npy')
mask = np.load('success_mask.npy')
class_labels = ['Red', 'Green', 'Blue', 'Noise']

# Check for GPU and if exists  run the training on it.
physical_devices = config.list_physical_devices('GPU')
if len(physical_devices) > 0:
    config.experimental.set_memory_growth(physical_devices[0], True)

# Preprocess the data
cond = blobs_loc_by_TS_on_original_img['is_really_mir'] == 0
y = blobs_loc_by_TS_on_original_img['selected_mir_type'].copy()
y[cond] = 3 # Noise group
y = y[mask]
mask_manually_erase = y != -1 # Boolean indexing to remove -1 elements
y = y[mask_manually_erase] # Our GT
crops = crops[mask_manually_erase]
crops_rgb = crops_rgb[mask_manually_erase]
```

In [12]:
```
# Examples for different miR classes
figure, axis = plt.subplots(4,6, figsize=(10,15))
for i in range(4):
    indexes = np.where(y == i)[0][:6]
    for pos, j in enumerate(indexes):
        axis[i,pos].set_aspect('equal')
        axis[i,pos].imshow(crops_rgb[j] * 50)

plt.show()
```



In [13]:
```
# Presenting data about the model
model = keras.Sequential(
    [
        keras.layers.Conv2D(12, kernel_size=(5, 5), activation="relu", input_sha
        keras.layers.MaxPooling2D(pool_size=(2, 2)),
        keras.layers.Conv2D(16, kernel_size=(5, 5), activation="relu", padding='
        keras.layers.MaxPooling2D(pool_size=(2, 2)),
        keras.layers.Flatten(),
```

```
        keras.layers.Dense(20, activation="relu"),
        keras.layers.Dense(4, activation="softmax"),
    ]
)

# Print the summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 34, 10, 12)        912

 max_pooling2d (MaxPooling2D  (None, 17, 5, 12)        0
 )

 conv2d_1 (Conv2D)           (None, 17, 5, 16)         4816

 max_pooling2d_1 (MaxPooling  (None, 8, 2, 16)         0
 2D)

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 20)                5140

 dense_1 (Dense)             (None, 4)                 84

=================================================================
Total params: 10,952
Trainable params: 10,952
Non-trainable params: 0
_________________________________________________________________
```

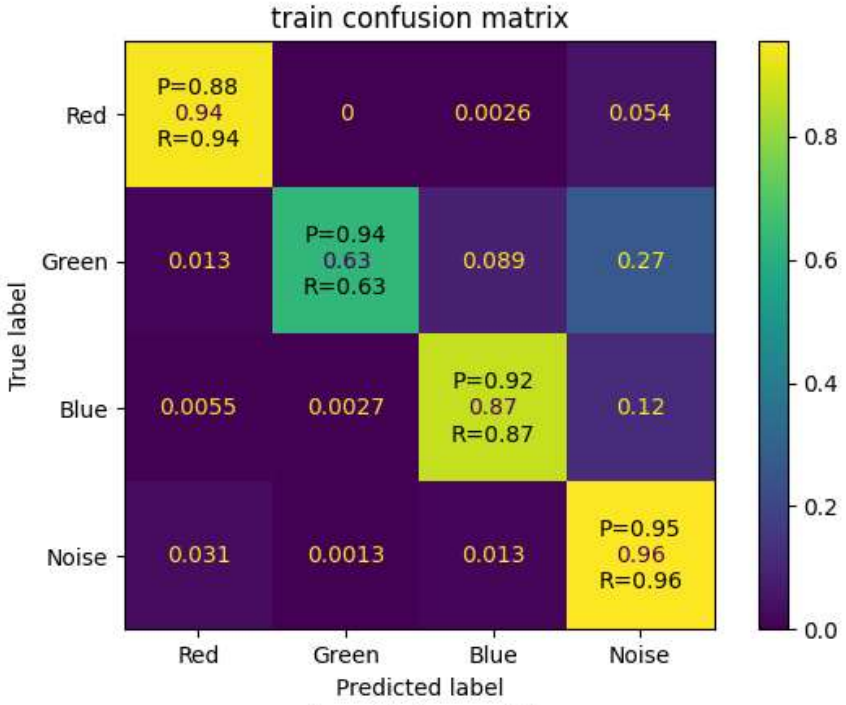
In [14]:
```
# Split the data into train, validation, and test sets
np.random.seed(3)
X_train, X_test, y_train, y_test = train_test_split(crops_rgb, y, test_size=0.2,
X_train, X_val, y_train, y_val = train_test_split(X_train,y_train, test_size=0.2

# Reshape the data
X_train = X_train.reshape(len(X_train), -1)
X_val = X_val.reshape(len(X_val), -1)
X_test = X_test.reshape(len(X_test), -1)

# Create a StandardScaler object
scaler = StandardScaler()

# Normalize the data
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

# Reshape the data back to the original shape
X_train = X_train.reshape(-1, 34, 10, 3)
X_val = X_val.reshape(-1, 34, 10, 3)
X_test = X_test.reshape(-1, 34, 10, 3)

# Convert labels to categorical
num_classes = len(np.unique(y))
y_train = keras.utils.to_categorical(y_train, num_classes)
```

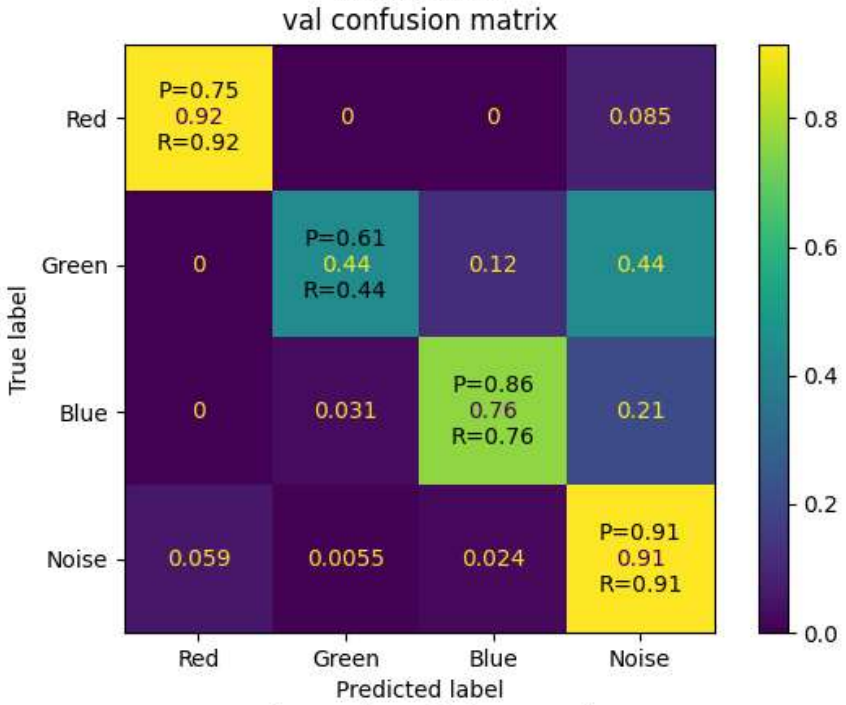
```
y_val = keras.utils.to_categorical(y_val, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# Define the LeNet model architecture
model = keras.Sequential(
    [
        keras.layers.Conv2D(12, kernel_size=(5, 5), activation="relu", input_sha
        keras.layers.MaxPooling2D(pool_size=(2, 2)),
        keras.layers.Conv2D(16, kernel_size=(5, 5), activation="relu", padding='
        keras.layers.MaxPooling2D(pool_size=(2, 2)),
        keras.layers.Flatten(),
        keras.layers.Dense(20, activation="relu"),
        keras.layers.Dense(num_classes, activation="softmax"),
    ]
)

# Compile the model
model.compile(loss="categorical_crossentropy", optimizer=keras.optimizers.Adam(l

# Train the model
model.fit(X_train, y_train, batch_size=128, epochs=10, validation_data=(X_val, y
```

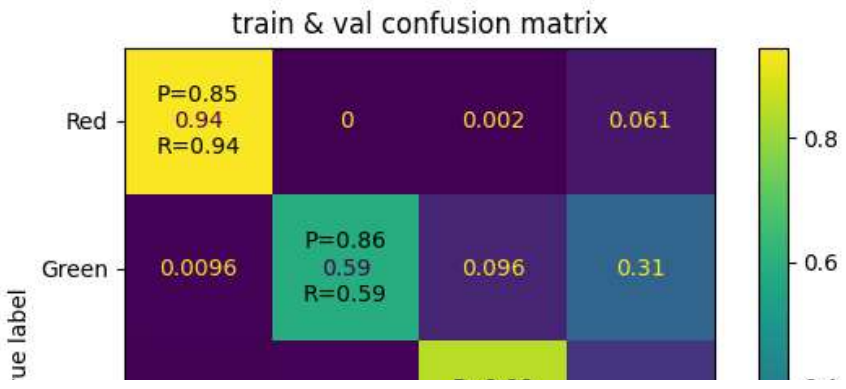
```
# Normalize Confusion Matricies after Yohai notes
# Evaluate the model
y_pred_train = model.predict(X_train)
y_pred_train = np.argmax(y_pred_train, axis=1)  # Convert predictions to class l
y_pred_val = model.predict(X_val)
y_pred_val = np.argmax(y_pred_val, axis=1)  # Convert predictions to class label
y_true_train = np.argmax(y_train, axis=1)
y_true_val = np.argmax(y_val, axis=1)

# Initilizing Confusion Matricies for each dataset
cm_train = metrics.confusion_matrix(y_true_train, y_pred_train).astype(float)
cm_val = metrics.confusion_matrix(y_true_val, y_pred_val).astype(float)
cm_train_val = metrics.confusion_matrix(np.concatenate((y_true_train,y_true_val)

cm_train_norm = np.zeros_like(cm_train)
cm_val_norm = np.zeros_like(cm_val)
cm_train_val_norm = np.zeros_like(cm_train_val)

for i in range(num_classes):
    sum_train = np.sum(cm_train[i])
    sum_val = np.sum(cm_val[i])
    sum_train_val = np.sum(cm_train_val[i])

    cm_train_norm[i] = cm_train[i] / sum_train
    cm_val_norm[i] = cm_val[i] / sum_val
    cm_train_val_norm[i] = cm_train_val[i] / sum_train_val

# Visualize the confusion matrix as a heatmap
fig, axis = plt.subplots(3, figsize=(7, 16))
metrics.ConfusionMatrixDisplay(confusion_matrix = cm_train_norm, display_labels
metrics.ConfusionMatrixDisplay(confusion_matrix = cm_val_norm, display_labels =
metrics.ConfusionMatrixDisplay(confusion_matrix = cm_train_val_norm, display_lab
axis[0].set_title('train confusion matrix')
axis[1].set_title('val confusion matrix')
```

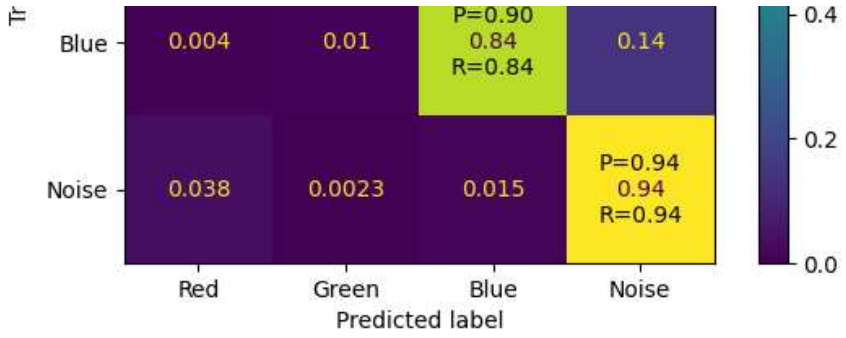
```
axis[2].set_title('train & val confusion matrix')

# Calculate precision and recall for each class
precision = lambda true_positives, false_positives: 1.0 if false_positives == 0
recall = lambda true_positives, false_negatives: 1.0 if false_negatives == 0 els

precision_train = np.zeros(num_classes)
precision_val = np.zeros(num_classes)
precision_train_val = np.zeros(num_classes)
recall_train = np.zeros(num_classes)
recall_val = np.zeros(num_classes)
recall_train_val = np.zeros(num_classes)

for i in range(num_classes):
    true_positives = cm_train[i, i]
    false_positives = sum(cm_train[:, i]) - true_positives
    false_negatives = sum(cm_train[i, :]) - true_positives
    precision_train[i] = precision(true_positives, false_positives)
    recall_train[i] = recall(true_positives, false_negatives)

    true_positives = cm_val[i, i]
    false_positives = sum(cm_val[:, i]) - true_positives
    false_negatives = sum(cm_val[i, :]) - true_positives
    precision_val[i] = precision(true_positives, false_positives)
    recall_val[i] = recall(true_positives, false_negatives)

    true_positives = cm_train_val[i, i]
    false_positives = sum(cm_train_val[:, i]) - true_positives
    false_negatives = sum(cm_train_val[i, :]) - true_positives
    precision_train_val[i] = precision(true_positives, false_positives)
    recall_train_val[i] = recall(true_positives, false_negatives)

    # Annotate the plot with precision and recall values
    axis[0].text(i, i, f'P={precision_train[i]:.2f}\n\nR={recall_train[i]:.2f}',
    axis[1].text(i, i, f'P={precision_val[i]:.2f}\n\nR={recall_val[i]:.2f}', ha=
    axis[2].text(i, i, f'P={precision_train_val[i]:.2f}\n\nR={recall_train_val[i

model.save("trained_model9.h5")
fig.savefig('cm_img9(12, 16, 20, (5, 5), (2, 2), 2e-3).png')
np.save('cm_data9(12, 16, 20, (5, 5), (2, 2), 2e-3).npy', [cm_train, cm_val, cm_
np.save('cm_sets9(12, 16, 20, (5, 5), (2, 2), 2e-3).npy', [X_train, X_val, X_tes


plt.show()
```

```
Epoch 1/10
19/19 - 3s - loss: 0.9019 - val_loss: 0.6186 - 3s/epoch - 157ms/step
Epoch 2/10
19/19 - 1s - loss: 0.5692 - val_loss: 0.4457 - 1s/epoch - 63ms/step
Epoch 3/10
19/19 - 1s - loss: 0.4103 - val_loss: 0.3950 - 1s/epoch - 66ms/step
Epoch 4/10
19/19 - 1s - loss: 0.3648 - val_loss: 0.3288 - 1s/epoch - 66ms/step
Epoch 5/10
19/19 - 1s - loss: 0.3303 - val_loss: 0.3266 - 1s/epoch - 63ms/step
Epoch 6/10
19/19 - 1s - loss: 0.3074 - val_loss: 0.3244 - 1s/epoch - 66ms/step
Epoch 7/10
19/19 - 1s - loss: 0.2813 - val_loss: 0.3296 - 1s/epoch - 66ms/step
Epoch 8/10
19/19 - 1s - loss: 0.2605 - val_loss: 0.3240 - 1s/epoch - 58ms/step
Epoch 9/10
19/19 - 1s - loss: 0.2451 - val_loss: 0.3149 - 1s/epoch - 53ms/step
Epoch 10/10
19/19 - 1s - loss: 0.2315 - val_loss: 0.3300 - 880ms/epoch - 46ms/step
76/76 [==============================] - 1s 12ms/step
26/26 [==============================] - 0s 9ms/step
```


train confusion matrix
P=0.88
0.94
R=0.94
0
0.0026
0.054
0.013
P=0.94
0.63
R=0.63
0.089
0.27
0.0055
0.0027
P=0.92
0.87
R=0.87
0.12
0.031
0.0013
0.013
P=0.95
0.96
R=0.96
Red
Green
Blue
Noise
True label
Predicted label
0.8
0.6
0.4
0.2
0.0


val confusion matrix
P=0.75
0.92
R=0.92
0
0
0.085
0
P=0.61
0.44
R=0.44
0.12
0.44
0
0.031
P=0.86
0.76
R=0.76
0.21
0.059
0.0055
0.024
P=0.91
0.91
R=0.91
Red
Green
Blue
Noise
True label
Predicted label
0.8
0.6
0.4
0.2
0.0

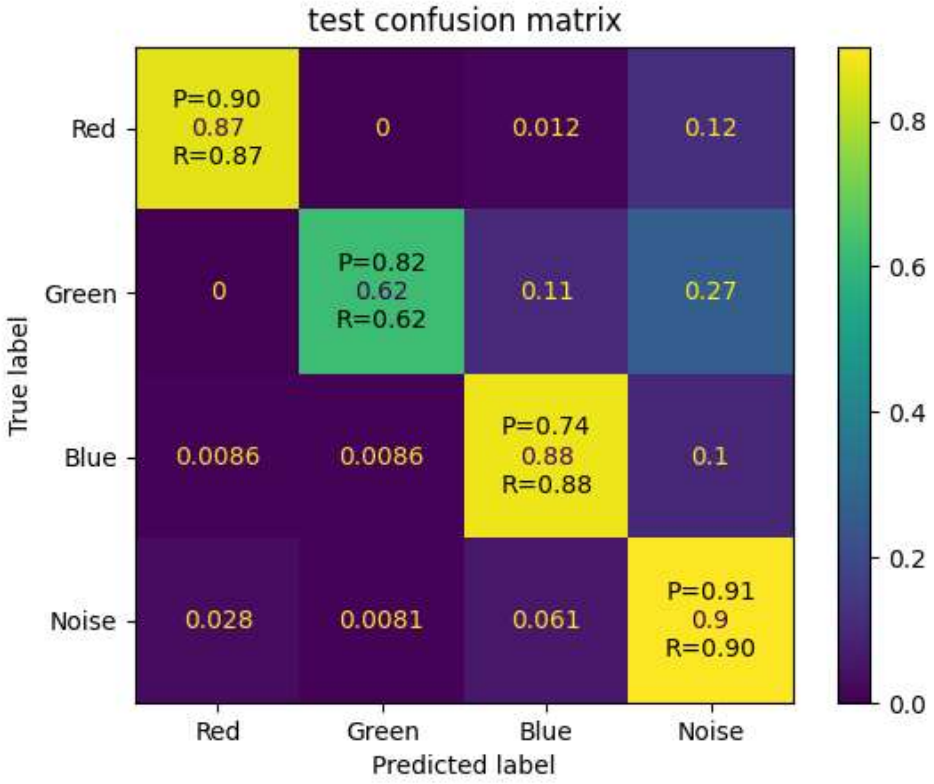

train & val confusion matrix
P=0.85
0.94
R=0.94
0
0.002
0.061
0.0096
P=0.86
0.59
R=0.59
0.096
0.31
Red
Green
ue label
0.8
0.6



In [15]:
```
# Test set
model = keras.models.load_model("trained_model2.h5")
X_train, X_val, X_test, y_train, y_val, y_test = np.load('cm_sets2(12, 16, 20, (


# Evaluate the model
num_classes = len(y_test[0])
y_pred_test = model.predict(X_test)
y_pred_test = np.argmax(y_pred_test, axis=1)  # Convert predictions to class lab
y_true_test = np.argmax(y_test, axis=1)

# Initilizing Confusion Matricies for each dataset
cm = metrics.confusion_matrix(y_true_test, y_pred_test).astype(float)
cm_norm = np.zeros_like(cm)

for i in range(num_classes):
    sum_test = np.sum(cm[i])
    cm_norm[i] = cm[i] / sum_test

# Visualize the confusion matrix as a heatmap
metrics.ConfusionMatrixDisplay(confusion_matrix = cm_norm, display_labels = clas
plt.title('test confusion matrix')

# Calculate precision and recall for each class
precision = lambda true_positives, false_positives: 1.0 if false_positives == 0
recall = lambda true_positives, false_negatives: 1.0 if false_negatives == 0 els

precision_test = np.zeros(num_classes)
recall_test = np.zeros(num_classes)

for i in range(num_classes):
    true_positives = cm[i, i]
    false_positives = sum(cm[:, i]) - true_positives
    false_negatives = sum(cm[i, :]) - true_positives
    precision_test[i] = precision(true_positives, false_positives)
    recall_test[i] = recall(true_positives, false_negatives)

    # Annotate the plot with precision and recall values
    plt.text(i, i, f'P={precision_test[i]:.2f}\n\nR={recall_test[i]:.2f}', ha='c

plt.savefig('cm_test2(12, 16, 20, (5, 5), (2, 2), 2e-3).png')
np.save('cm_test_data2(12, 16, 20, (5, 5), (2, 2), 2e-3).npy', [cm])
```

```
plt.show()
```

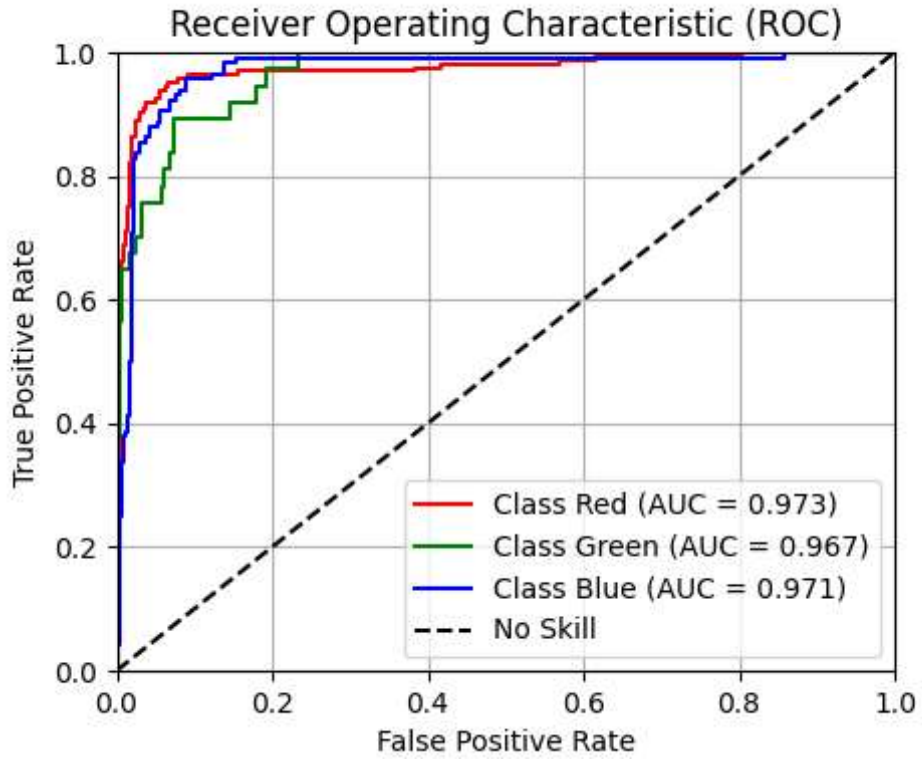
```
26/26 [==============================] - 0s 8ms/step
```



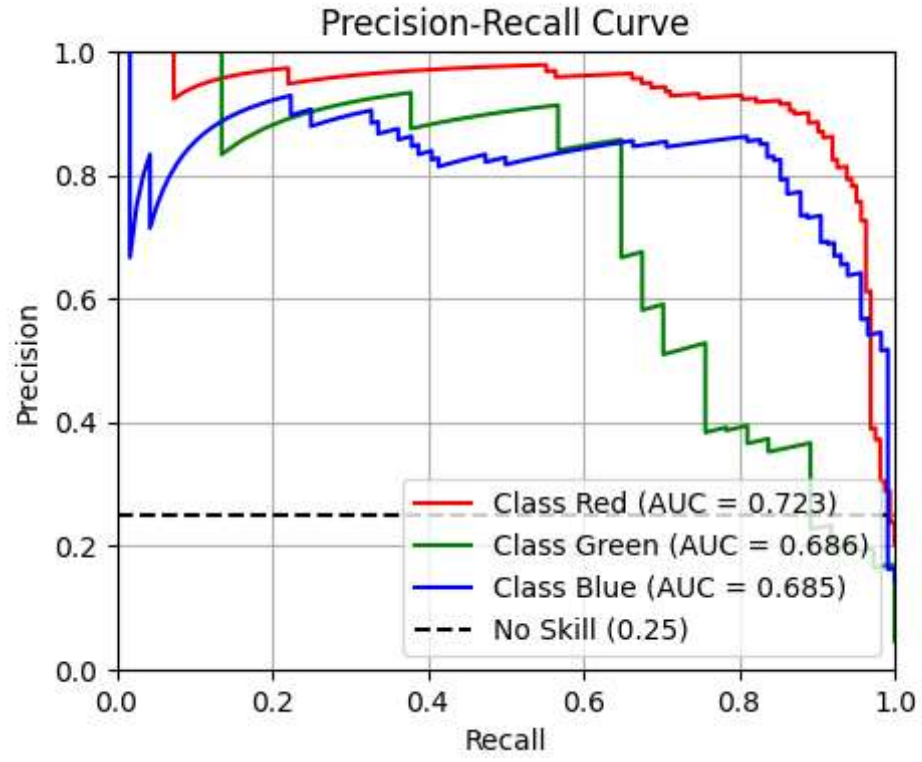
In [16]:
```
# Compute ROC curve and AUC for each class
model = keras.models.load_model("trained_model2.h5")
X_train, X_val, X_test, y_train, y_val, y_test = np.load('cm_sets2(12, 16, 20, (

y_pred_prob = model.predict(X_test)

fpr = dict()
tpr = dict()
roc_auc = dict()

num_classes = len(y_test[0])
for i in range(num_classes-1):
    fpr[i], tpr[i], _ = metrics.roc_curve(y_test[:, i], y_pred_prob[:, i])
    roc_auc[i] = metrics.auc(fpr[i], tpr[i])

# Plot ROC curves for each class
plt.figure(figsize=(5, 4))
plt.plot(fpr[0], tpr[0], '-r', label='Class {} (AUC = {:.3f})'.format(class_labe
plt.plot(fpr[1], tpr[1], '-g', label='Class {} (AUC = {:.3f})'.format(class_labe
plt.plot(fpr[2], tpr[2], '-b', label='Class {} (AUC = {:.3f})'.format(class_labe

plt.plot([0, 1], [0, 1], 'k--', label='No Skill')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.grid(True)
```

```
plt.savefig('ROC2(12, 16, 20, (5, 5), (2, 2), 2e-3).png')
np.save('ROC_data2(12, 16, 20, (5, 5), (2, 2), 2e-3).npy', [fpr, tpr])

plt.show()


# Compute Precision-Recall curve and AUC for each class
y_scores = model.predict(X_test)
print(y_test.shape)

precision = dict()
recall = dict()
pr_auc = dict()

for i in range(num_classes-1):
    precision[i], recall[i], _ = metrics.precision_recall_curve(y_test[:, i], y_
    sorted_indices = np.argsort(precision[i])
    sorted_precision = precision[i][sorted_indices]
    sorted_recall = recall[i][sorted_indices]
    pr_auc[i] = metrics.auc(sorted_precision, sorted_recall)

# Plot the Precision-Recall curve for each class
plt.figure(figsize=(5, 4))
plt.plot(recall[0], precision[0], '-r', label='Class {} (AUC = {:.3f})'.format(c
plt.plot(recall[1], precision[1], '-g', label='Class {} (AUC = {:.3f})'.format(c
plt.plot(recall[2], precision[2], '-b', label='Class {} (AUC = {:.3f})'.format(c

plt.plot([0, 1], [1/4, 1/4], 'k--', label='No Skill (0.25)')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="lower right")
plt.grid(True)

plt.savefig('PRC2(12, 16, 20, (5, 5), (2, 2), 2e-3).png')
np.save('PRC_data2(12, 16, 20, (5, 5), (2, 2), 2e-3).npy', [recall, precision])

plt.show()
```

```
26/26 [==============================] - 0s 9ms/step
```



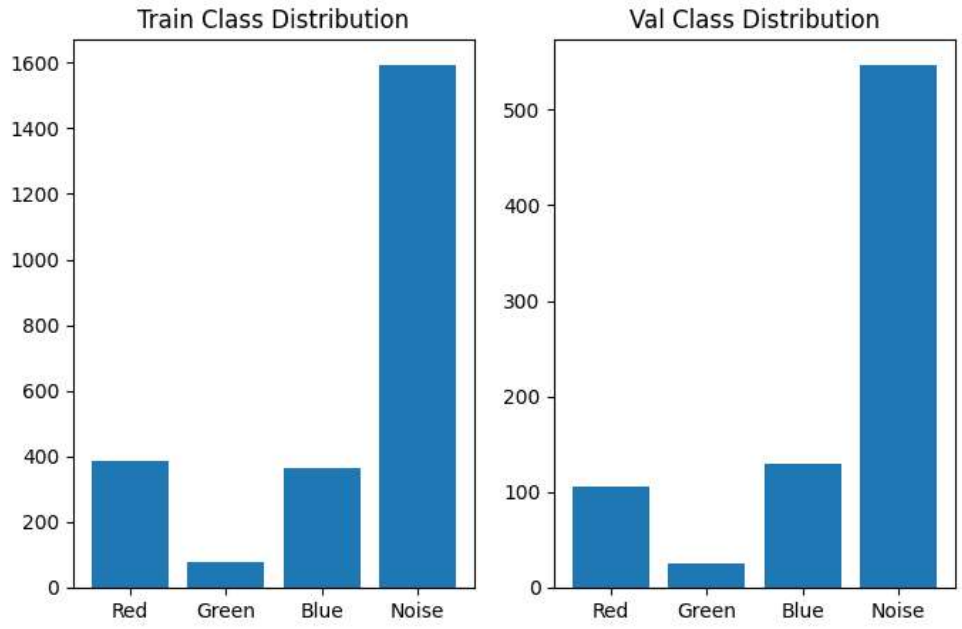

```
26/26 [==============================] - 0s 8ms/step
(808, 4)
```



In [17]:
```
# Count the occurrences of each class
class_counts_train = np.sum(y_train, axis=0)
class_counts_val = np.sum(y_val, axis=0)

# Get the predicted labels for the validation set
y_pred_train = model.predict(X_train)
y_pred_train = np.argmax(y_pred_train, axis=1)  # Convert predictions to class l
y_pred_val = model.predict(X_val)
y_pred_val = np.argmax(y_pred_val, axis=1)  # Convert predictions to class label
```

```
# Plot the bar graph
figure, axis = plt.subplots(1,2, figsize=(8,5))
axis[0].bar(class_labels, class_counts_train)
axis[0].set_label('Class')
axis[0].set_label('Count')
axis[0].set_title('Train Class Distribution')
axis[0].set_xticks(np.arange(len(class_labels)), class_labels)

axis[1].bar(class_labels, class_counts_val)
axis[1].set_label('Class')
axis[1].set_label('Count')
axis[1].set_title('Val Class Distribution')
axis[1].set_xticks(np.arange(len(class_labels)), class_labels)

plt.show()

figure.savefig('occurrences of each class')
```

```
76/76 [==============================] - 1s 8ms/step
26/26 [==============================] - 0s 6ms/step
```



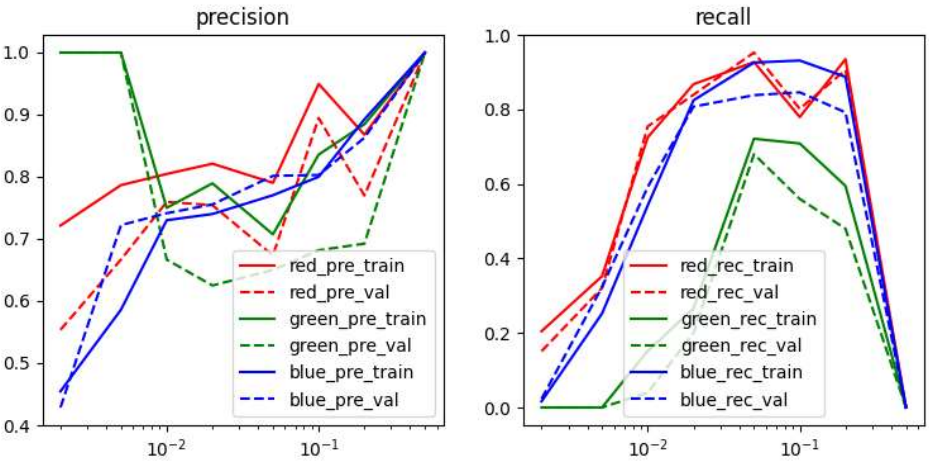
In [18]:
```
def lenet5_same(filter_size1, filter_size2, filter_size3, kernel_size, pool_size

    # Split the data into train, validation, and test sets
    np.random.seed(3)
    X_train, X_test, y_train, y_test = train_test_split(crops_rgb, y, test_size=
    X_train, X_val, y_train, y_val = train_test_split(X_train,y_train, test_size

    # Reshape the data
    X_train = X_train.reshape(len(X_train), -1)
    X_val = X_val.reshape(len(X_val), -1)
    X_test = X_test.reshape(len(X_test), -1)

    # Create a StandardScaler object
    scaler = StandardScaler()

    # Normalize the data
    X_train = scaler.fit_transform(X_train)
```

```
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

# Reshape the data back to the original shape
X_train = X_train.reshape(-1, 34, 10, 3)
X_val = X_val.reshape(-1, 34, 10, 3)
X_test = X_test.reshape(-1, 34, 10, 3)

# Convert labels to categorical
num_classes = len(np.unique(y))
y_train = keras.utils.to_categorical(y_train, num_classes)
y_val = keras.utils.to_categorical(y_val, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# Define the LeNet model architecture
model = keras.Sequential(
    [
        keras.layers.Conv2D(filter_size1, kernel_size=kernel_size, activatic
        keras.layers.MaxPooling2D(pool_size=pool_size, padding='same'),
        keras.layers.Conv2D(filter_size2, kernel_size=kernel_size, activatic
        keras.layers.MaxPooling2D(pool_size=pool_size, padding='same'),
        keras.layers.Flatten(),
        keras.layers.Dense(filter_size3, activation="relu"),
        keras.layers.Dense(num_classes, activation="softmax"),
    ]
)


# Compile the model
model.compile(loss="categorical_crossentropy", optimizer=keras.optimizers.SG

# Train the model
model.fit(X_train, y_train, batch_size=128, epochs=10, validation_data=(X_va

# Evaluate the model
y_pred_train = model.predict(X_train)
y_pred_train = np.argmax(y_pred_train, axis=1)  # Convert predictions to cla
y_pred_val = model.predict(X_val)
y_pred_val = np.argmax(y_pred_val, axis=1)  # Convert predictions to class l
y_true_train = np.argmax(y_train, axis=1)
y_true_val = np.argmax(y_val, axis=1)

# Calculate precision and recall for each class
cm_train = metrics.confusion_matrix(y_true_train, y_pred_train)
cm_val = metrics.confusion_matrix(y_true_val, y_pred_val)

precision_train = np.zeros(num_classes)
recall_train = np.zeros(num_classes)
precision_val = np.zeros(num_classes)
recall_val = np.zeros(num_classes)

for i in range(num_classes):
    true_positives = cm_train[i, i]
    false_positives = sum(cm_train[:, i]) - true_positives
    false_negatives = sum(cm_train[i, :]) - true_positives
    precision_train[i] = 1.0 if false_positives == 0 else true_positives / (
    recall_train[i] = 1.0 if false_negatives == 0 else true_positives / (tru

    true_positives = cm_val[i, i]
    false_positives = sum(cm_val[:, i]) - true_positives
```

```
        false_negatives = sum(cm_val[i, :]) - true_positives
        precision_val[i] =  1.0 if false_positives == 0 else true_positives / (t
        recall_val[i] = 1.0 if false_negatives == 0 else true_positives / (true_

    return precision_train, recall_train, precision_val, recall_val
```

In [19]:

```
num_classes = len(np.unique(y))
xs = []

lst = [2e-3, 5e-3, 1e-2, 2e-2, 5e-2, 1e-1, 2e-1, 5e-1]
ys = np.zeros((4, len(lst), num_classes))
filter_size1, filter_size2, filter_size3, kernel_size, pool_size, lr = 12, 16, 2
for ind, i in enumerate(lst):
    pre_train, rec_train, pre_val, rec_val = [], [], [], []
    for j in range(30):
        precision_train, recall_train, precision_val, recall_val = lenet5_same(f

        pre_train.append(precision_train)
        rec_train.append(recall_train)
        pre_val.append(precision_val)
        rec_val.append(recall_val)

    xs.append(i)
    ys[0][ind] = np.median(pre_train, axis=0)
    ys[1][ind] = np.median(rec_train, axis=0)
    ys[2][ind] = np.median(pre_val, axis=0)
    ys[3][ind] = np.median(rec_val, axis=0)

    print(ind, 'out of', len(lst))

# Represent The Data in Graphs
fig, ax = plt.subplots(1, 2, figsize=(9, 4))
ax[0].plot(xs, ys[0,:,0], '-r', xs, ys[2,:,0], '--r',
          xs, ys[0,:,1], '-g', xs, ys[2,:,1], '--g',
          xs, ys[0,:,2], '-b', xs, ys[2,:,2], '--b')
ax[0].set_title('precision')
ax[0].set_xscale('log')
ax[0].legend(['red_pre_train', 'red_pre_val',
              'green_pre_train', 'green_pre_val',
              'blue_pre_train', 'blue_pre_val'], loc='best')

ax[1].plot(xs, ys[1,:,0], '-r', xs, ys[3,:,0], '--r',
          xs, ys[1,:,1], '-g', xs, ys[3,:,1], '--g',
          xs, ys[1,:,2], '-b', xs, ys[3,:,2], '--b')
ax[1].set_title('recall')
ax[1].set_xscale('log')
ax[1].legend(['red_rec_train', 'red_rec_val',
              'green_rec_train', 'green_rec_val',
              'blue_rec_train', 'blue_rec_val'], loc='best')

fig.savefig('lr_SGD2_img(12, 16, 20, (5, 5), (2, 2), 1e-3).png')
np.save('lr_SGD2_x(12, 16, 20, (5, 5), (2, 2), 1e-3).npy', xs)
np.save('lr_SGD2_y(12, 16, 20, (5, 5), (2, 2), 1e-3).npy', ys)
```


precision
1.0
0.9
0.8
0.7
0.6
0.5
0.4
10^{-2}
10^{-1}
red_pre_train
red_pre_val
green_pre_train
green_pre_val
blue_pre_train
blue_pre_val
recall
1.0
0.8
0.6
0.4
0.2
0.0
10^{-2}
10^{-1}
red_rec_train
red_rec_val
green_rec_train
green_rec_val
blue_rec_train
blue_rec_val