COMP9517 Project Report

Team:Genshin Launcher.exe

Moyu Ni
z5509650

Zijie Wang
z5392127

Tristan Fischer
z5310974

Kaize Niu
z5496085

Ziyang Chen
z5450036

I. Introduction

During the past dacade,The solar photovoltaic industry is developing at a very rapid rate and will play an increasingly important role in the world's electricity supply in the next few years.The statistics from the International Energy Agency (IEA) indicated that the installed PV capacity in 2019 surpassed 627 GW and the IEA’s latest 5-year forecast shows that total PV capacity will reach 1209 GW by 2024 (IEA, 2019). As the most important component of the entire solar photovoltaic equipment, improving the efficiency of solar panels has become a crucial issue, because this may directly affect the amount of power generated.Because of this, testing solar panels and ensuring their efficient operation has become an issue that must be discussed.Among many issues, a very critical one is to check the health of PV cells and predict their future health.

In recent years, EL(Electroluminescence Photovoltaic) has become a very common practice in the field of defect detection.

The ELPV dataset[1],fig.1 shows 4 images in it,consists of electroluminescence (EL) images of solar cells.These images are used to assess the quality of solar panels by recongnizing invisible problems such as cracks,fragment and other defect.This dataset contains 2,624 electroluminescence images showcasing both operational and faulty photovoltaic cells that exhibit various levels of deterioration, sourced from 44 distinct solar panels.There are two types of solar module,monocrystalline or polycrystalline.

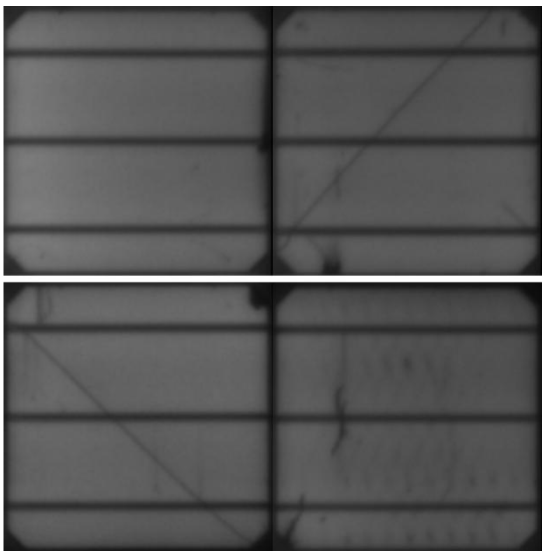


Fig .1

The project aims to categorize images of cells based on th e likelihood that they are flawed.In our experiments,we are pl anned to create and evaluate defect detection methods to asse ss the condition of photovoltaic cells using EL images of sola r panels and then predict the health of photovoltaic cells.

The rest of the paper is organized as follows: Section 2 in troduces related work on defect detaction and present our ide as to predict health of PV cells. Section 3 is the methods we used.The experimental results are given in Section 4 and disc ussed in Section 5 and we summarized our conclusion in Sec tion 6.

II. Literature

Classifing cell images according to their probability of d efectiveness is an active research topic due to the explosive development of the solar photovoltaic industry.A large amo unt of related work uses various methods to analyze cell ima ges to determine the possibility of cell defects, and improves the accuracy of prediction through various methods.The mo re mainstream models currently include the following, SGD model,Neural Network(CNN) model , etc.

Deep learning methods have been utilized in research for recongnizing defects increasingly in recent years and many schloars have focused on the automated defect detaction met hods for many years.Deitsch et al. [2] suggest two defect det ection methods,SVM and CNN.Through their experiments, t he results show that the efficiency of CNN ia relatively bette r which means CNN classifier can get higher accuracy.

In medical context,Kather et al. [3]evaluated multiple sta te-of-the-art CNNs including ResNet-50 for multi-class lung cancer prediction using transfer learning. They achieved hig h AUCs of 0.97-0.99 on four lung image datasets. Sun et al. [4] proposed a multi-scale CNN model incorporating both loc al and larger contextual information that gave a sensitivity o f 97.4% and specificity of 97.2% for patch-level lung cancer detection.

III. Methods

This part is divided into five parts. Since the five of us u sed different methods, we will introduce the methods we use drespectively below.Abbreviations and Acronyms

A. Tristan Resnet

Resnet (Residual Network) is a network architecture which utilises skip connections to make deep neural networks train faster and avoid other issues such as exploding gradient. More can be read on the offical paper. A variety of architectures were suggested in the original paper the one which I chose to adapt known as Resnet18 consists of 17 convolutional layers and 1 fully conected layer. It is one of the smallets networks suggested within this architecture and therefore is the most optimal to use for this project due to the small dataset, as I noted from initial testing that using larger networks such as Resnet50 resulted in overfitting.

The model is implemented with Pytorch.In the original Resnet18 architecture they use multiples of 64 filters within each resnet block all the way up to 512 filteres, as the original purpose for resnet was to be used in imagenet competitions it has alot of width so it can learn up to 10000 labels. In our case we have a smaller dataset and only 4 possible labels so I reduced the width of each layer down to a multiple of 2 and played around with the variable myself. If the width of the network was too high the model had a tendancy to overfit as seen in the following fig.2:

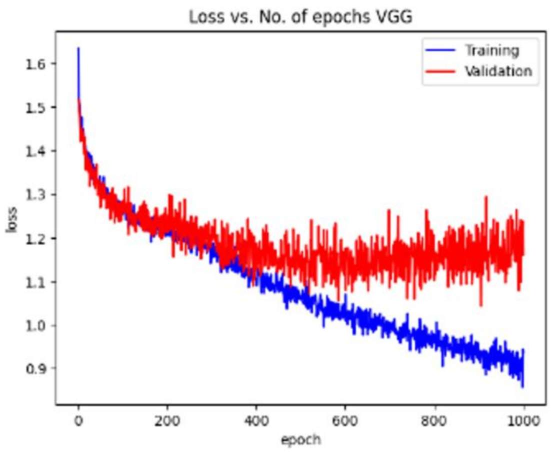


Fig.2

Overfitting occurs when the validation loss begins to increase whilest the training loss continues to decrease this is a sign that the model has begun learning the images, such as random noise rather than generalizing the results. a variety of methods can be used to overcome this.

For example reducing the model complexity as noted above doing some experimentations with this I found the best value for these filters was around multiples of 4 rather than 64 though I do believe more fine tuning in this case is possible.

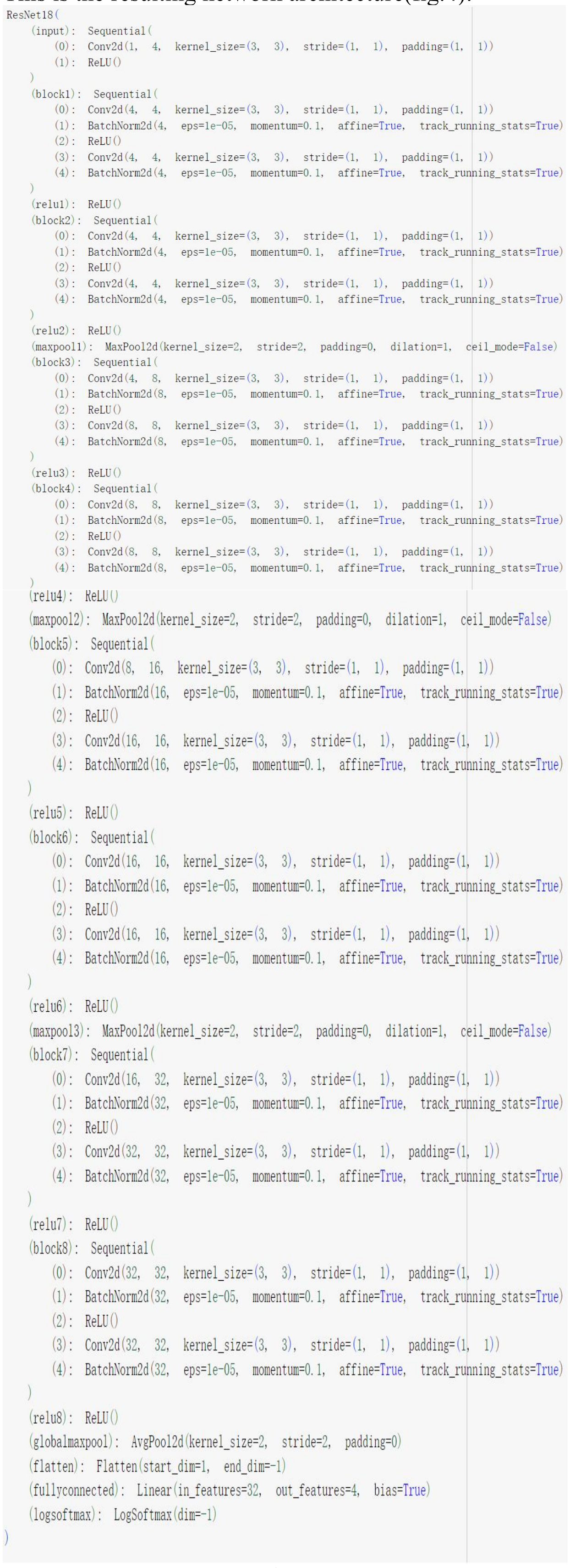
A couple of other methods to reduce the overfitting is increasing the dataset through methods such as image augmentation, this can include flipping the images, warping images, shifting the color ranges, this makes the network learn the features which would indicate a label rather than any specific noise patterns which could be used to label and image and therfore increase the generalisation of the network.

More recent discoveres such as batchnormalisation have also been implemented which help in training time and to reduce overfitting.

The last option to help in reducing the networks overfitting problem is to include L1 and L2 regularisation, in my case I opted to L2 regularisation in weight decay as, dropout works better on fully conected layers and larger networks and runs the risk of severly slowing training time in smaller networks due to dropped images in the cnn layers.

As for the optimisation algorithm I am using Adam as it is more forgiving then other methods such SGD and results in faster training times due to its adaptive momentum especially in smaller networks such as the one used.

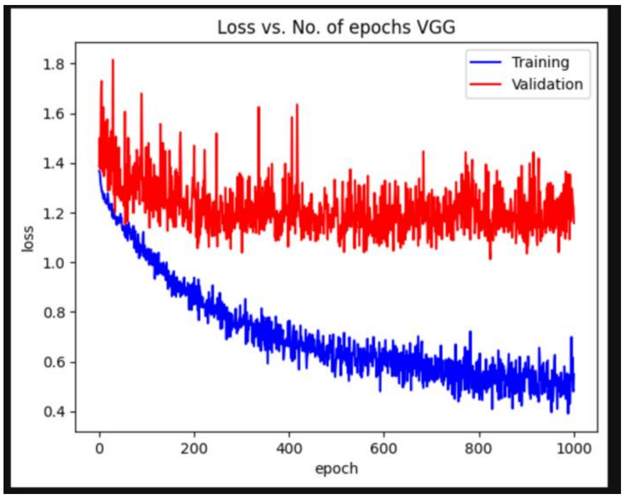
This is the resulting network architecture(fig.4):



```
ResNet18(
  (input): Sequential(
    (0): Conv2d(1, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
  )
  (block1): Sequential(
    (0): Conv2d(4, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(4, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (relu1): ReLU()
  (block2): Sequential(
    (0): Conv2d(4, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(4, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (relu2): ReLU()
  (maxpool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (block3): Sequential(
    (0): Conv2d(4, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (relu3): ReLU()
  (block4): Sequential(
    (0): Conv2d(8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (relu4): ReLU()
  (maxpool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (block5): Sequential(
    (0): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (relu5): ReLU()
  (block6): Sequential(
    (0): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (relu6): ReLU()
  (maxpool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (block7): Sequential(
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (relu7): ReLU()
  (block8): Sequential(
    (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (relu8): ReLU()
  (globalmaxpool): AvgPool2d(kernel_size=2, stride=2, padding=0)
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (fullyconnected): Linear(in_features=32, out_features=4, bias=True)
  (logsoftmax): LogSoftmax(dim=-1)
)
```

Fig.4

Results(fig.5 and fig.6):



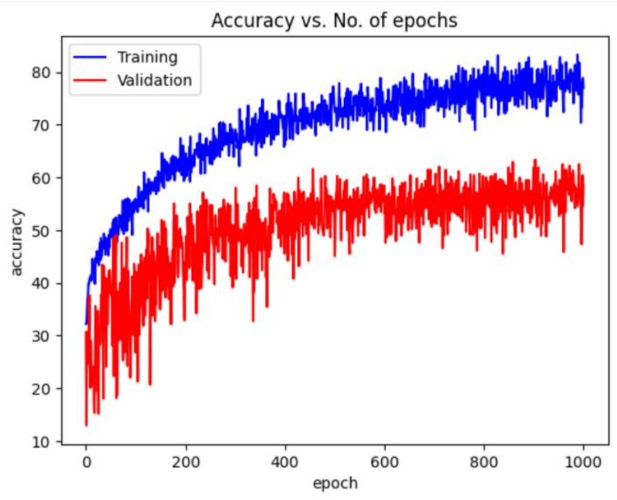


Fig.6

B. *Moyu Self-designed CNN*

CNN model is a very typical method for image analysis.[6] gives us ways to implement CNN models by using pytorch, keras, tensorflow.A CNN is composed of an input layer, an output layer, and many hidden layers in between.These layers carry out operations that change the data to learn features that are unique to the data. Convolution, ReLU or activation, and pooling are among the most commonly used layers. The benefits of convolutional neural networks are as follows:

CNN has very good feature extraction capabilities. Compared with other traditional machine learning methods, due to the existence of convolutional layers, there is no need to separately design how to perform feature extraction. This is very effective for completing image analysis tasks. This can greatly save working time and improve work efficiency.

CNN is very adaptable. CNN can be applied to images of different sizes. And I am able to adapt to different task requirements through different designs of convolutional layers, such as adjusting the size of the convolution kernel, increasing or decreasing the number of various layers, adjusting the depth of the neural network, etc.

Spatial invariance. CNN can capture the spatial correlation of images through the convolution layer and achieve relative position invariance through the pooling layer, which enhances the robustness of the model. Therefore, a series of methods such as rotation and scaling can be applied in CNN to achieve image enhancement and improve the accuracy of available image analysis data.

In CNN, image enhancement is a commonly used technique to expand the training set by transforming the original image to generate new, modified images, which can enhance the diversity of the data. It can also help the model learn more robust features. There are many methods of image enhancement. In my CNN experiment, there are two methods used, noise reduction and histogram equalization.

Image reduction is a common method to improve image quality. There are many ways to achieve this, such as smoothing filters, median filters, bilateral filters, etc. In this experiment, I use the Gaussian filter, which is a type of smoothing filter. The reason for choosing it is that it uses Gaussian distribution as a weight to perform a weighted average of surrounding pixels, which can preserve edges more naturally.

Histogram equalization is also an image enhancement method that effectively stretches the histogram of the image to improve the contrast of the image when the photo is covered by the background,but in my experiment I don't think it works well.

During training, the code implements common data augmentation and normalization techniques for image classification. The transforms are randomized during training to increase variance. Test images are standardized without augmentation. This preprocessing helps the model generalize better.

The model is implemented with keras and tensorflow.

The following is the structure of this CNN model: Convolutional layer (Conv2D): There are 4 in total, namely: The first convolutional layer has 16 filters and the convolution kernel size is 3x3. The second convolutional layer has 32 filters and the convolution kernel size is 3x3. The third convolutional layer has 64 filters and the convolution kernel size is 3x3. The fourth convolutional layer has 128 filters and the convolution kernel size is 3x3. Maximum pooling layer (MaxPooling2D): There are 4 in total, each with a 2x2 pooling window. Flattening layer (Flatten): There is 1, which is used to flatten the multi-dimensional output of the convolutional layer into one dimension so that it can be processed by the fully connected layer. Fully connected layer (Dense): There are 2, namely: The first fully connected layer has 256 neurons and uses the ReLU activation function. The number of neurons in the second fully connected layer is equal to the number of categories (num_classes), and the softmax activation function is used for multi-classification. Dropout layer (Dropout): There is 1, set to a dropout rate of 0.5, used to reduce overfitting.

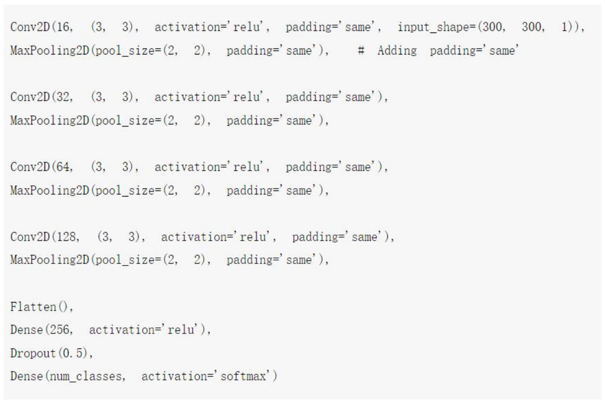
In the model, I use the Adam optimizer is because Mehta, Smit, Chirag Paunwala, and Bhaumik Vaidya[7]show that it is able to increase accuaracy for CNN model.he Adam optimizer is a gradient descent optimization algorithm used in deep learning applications. Its features are very suitable for my CNN models. Its advantages include:

Adaptive learning rate:The Adam optimizer will calculate its own learning rate for each parameter. The higher the learning rate is set, the larger the step size is; the lower the learning rate is set, the smaller the step size is.

Momentum: By taking into account past gradients, it can help speed up the learning process, especially on deep networks and complex datasets.

I use the given ELPV dataset as the dataset, set 25% of the images as the test set and 75% of the images as the training set, and then conduct experiments using the CNN model without image enhancement.

This is the resulted network architecture(fig.7):



```
Conv2D(16, (3, 3), activation='relu', padding='same', input_shape=(300, 300, 1)),
MaxPooling2D(pool_size=(2, 2), padding='same'),   # Adding padding='same'

Conv2D(32, (3, 3), activation='relu', padding='same'),
MaxPooling2D(pool_size=(2, 2), padding='same'),

Conv2D(64, (3, 3), activation='relu', padding='same'),
MaxPooling2D(pool_size=(2, 2), padding='same'),

Conv2D(128, (3, 3), activation='relu', padding='same'),
MaxPooling2D(pool_size=(2, 2), padding='same'),

Flatten(),
Dense(256, activation='relu'),
Dropout(0.5),
Dense(num_classes, activation='softmax')
```

Fig.7

Here are some of the confusions and problems I encountered while training the model. According to the requirements of the project, I first divided the results into four classes, but in the process I found that the performance of the third class is obviously not good and the precision is only 0.32, so I tried to differentiate the results into two classes instead of four, which gave a relatively large improvement in the results, with an accuracy of about 90%. However, since the project required us to divide into four categories, I continued to work in four categories. After that I added preprocessing methods to my model, which included de-noising, Histogram equalization two methods. After adding these two methods, after debugging, the performance of the results improved, from 68% accuracy to 71%, which proved that my preprocessing methods achieved some results. Secondly I also used dropout layer, this regularization technique significantly helped me to reduce the overfitting and enhance the generalization ability of the model, which also improved the accuracy to some extent.

C. Kaize SVM

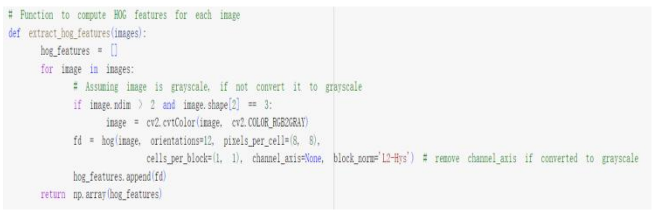
Support Vector Machine (SVM) is a versatile and powerful supervised learning algorithm, In classification tasks, the main goal of SVM is to find a hyperplane in an N-dimensional space that distinctly classifies the data points.

The reason I choose SVM is because it is particularly adept at finding optimal decision boundaries in high-dimensional spaces, even with relatively few data points. This capability is crucial for image classification tasks, as image data typically gets transformed into high-dimensional feature vectors. A key characteristic of SVMs is their generalization capability, meaning they can avoid overfitting and perform well on unseen data. Ensuring that a model generalizes well to new data is very important for tasks like EL image classification. And also it is suitable for Small to Medium Datasets.

In the beginning, I only used SVM for training, but the results were not very satisfactory. So I used HOG for feature extraction. HOG works by dividing the image into small regions and computing the distribution (histograms) of gradient directions (or edge orientations) within these regions. This approach is useful for capturing shape and texture information, which is essential when analyzing PV cell images for defects. once HOG features are computed, it will form a robust and fixed-size feature vector representation for each image. SVM classifiers work exceptionally well with such high-dimensional data. It's less prone to overfitting.

After this change, the accuracy has been significantly improved, but it can only remain between 60% and 70%. I think it may be caused by the size of the cell in which gradient histograms are computed in HOG being too large. Larger cells may miss small features. So I adjusted the size and orientation parameters. This time the accuracy can be maintained at around 75%.

I also tried to augment my dataset by applying transformations like rotation, flipping, scaling, or adding noise., but it had no obvious effect and sometimes made the results worse. There may be many reasons for this phenomenon. I have not yet discovered it. This is also something I still need to learn.



```
# Function to compute HOG features for each image
def extract_hog_features(images):
    hog_features = []
    for image in images:
        # Assuming image is grayscale, if not convert it to grayscale
        if image.ndim > 2 and image.shape[2] == 3:
            image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
        fd = hog(image, orientations=12, pixels_per_cell=(8, 8),
                      cells_per_block=(1, 1), channel_axis=None, block_norm='L2-Hys') # remove channel_axis if converted to grayscale
        hog_features.append(fd)
    return np.array(hog_features)
```

Fig.8

D. Ziyang BEiT with fine-tuning

- BET

BEit is a pre-trained model developed by Microsoft. It was training on ImageNet-21k (14 million images, 21,841 classes) at resolution 224x224, and them fine-tuned on ImageNet 2012 (1 million images, 1,000 classes) at resolution 224x224. Using pre-trained models can greatly reduce the time needed for training, and they can achieve very good results.

- Fine-tuning

Fine-tuning is a method in machine learning. By using the model has already been trained on a large dataset and has learned basic patterns and features, and then provide additional training on relevant data to perform well on a specific task. This process does not start from zero, but uses the knowledge the pre-trained model already has. The purpose of fine-tuning is to make the model better adapt to the specific task by learning features and patterns unique to the task. During the fine-tuning process, some parameters of the model are adjusted. This helps the model learn the new task more effectively while retaining the knowledge it already has. Successful fine-tuning can improve the model's performance on specific tasks while maintaining a good ability to generalize to new situations. I used the images from the project as a training set for fine-tuning. By doing this additional fine-tuning, I achieved very good results.

- Potential improvement of BEiT with fine-tuning

Use a Larger Model: The BEiT model currently in use has the smallest number of parameters due to hardware limitations. This means it has limited knowledge and learning capability. Using a larger model with more parameters, like the latest ChatGPT-4 Turbo, can provide access to a broader range of pre-existing knowledge and greater potential for improvement.

Use LoRA: Instead of training the entire model, consider training a LoRA (Low-Rank Adaptation) model. LoRA significantly reduces the training time required and enhances the ability to perform specific classification tasks. However, the current parameter size of the model in use is not large enough to necessitate the use of LoRA, so it's not essential in this case.

E. *Zijie Alexnet*

- Model Configuration

AlexNet contained eight layers; the first five were convolutional layers, some of them followed by max-pooling layers, and the last three were fully connected layers. The network, except the last layer, is split into two copies, each run on one GPU.The entire structure can be written as Fig.9

$$(CNN \to RN \to MP)^2 \to (CNN^3 \to MP) \to (FC \to DO)^2 \to \text{Linear} \to \text{softmax}$$

Fig.9

Where:

CNN = convolutional layer (with ReLU activation)

RN = local response normalization

MP = maxpooling

FC = fully connected layer (with ReLU activation)

Linear = fully connected layer (without activation)

DO = dropout It used the non-saturating ReLU activation function, which showed improved training performance over tanh and sigmoid. To simplify my code, I implement AlexNet model with library torchvision which provides many pre-trained computer vision models. And in my implementation, I didn't load any pretrained parameters and the final fully connected layer has been modified and replaced with a linear layer with an output latitude of 4 followed by a softmax layer because we are doing a four labels classification task.

- Training Configuration.

During training, I trained my model on a P100 Nvidia GPU on kaggle platform. And I trained my model for 60 epochs with Adam optimizer. My objective function is CrossEntropyLoss, this is a classical classification objective function. After some attempts, I chose the learning rate of 0.001 and weight_decay0.001 to make my model achieve better performance as much as possible.

In order to improve the performance of the model, I also normalized the data in the process of data preprocessing to constrain the numerical range of the model and improve the robustness of the model. And to deal with the serious imbalance problem in the data set, I used a simple resampling technique.

After training 60 Epochs I got a training loss of 0.08 and evaluated this model as my final model for my paper presentation.

IV. Experimental Results

A. *Evaluation Metrics*

Accuracy,recall,precision and F1 score are the four metrics commonly used to evaluate good models.For these four types of data, the larger the value obtained, the better the model results are, indicating that this model is more suitable for the analysis of ELPV datasets than other models.

Accuracy is a metric that approximates the performance of the model across all categories. It is calculated by using the following formula:

Accuracy = TP + TN / TP + TN + FP + FN

where respectively TP and TN represent the number of true positives and true negatives,FP and FN represent the number of false positives and false positives.

Recall is a metric that the radio between true positives(TF) and the sum of false negatives(FN) and TF.It's calculated using the following formula:

Recall = TP / TP + FN

Precision is a metric that the radio between true positives(TF) and the sum of false positives(FN) and TF.It's calculated using the following formula:

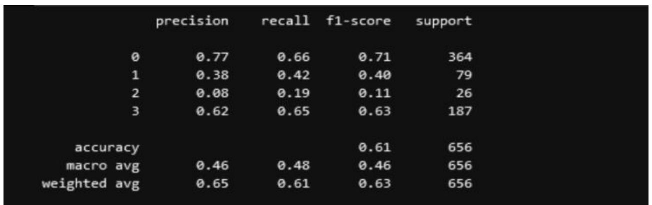
Precision = TP / TP + FP

The F1 score is based on recall and precision,which takes the harmonic means of racall and precision.It's calculated using the following formula:

F1 score = 2 * precision * recall / precision + recall

B. *The methods' results*

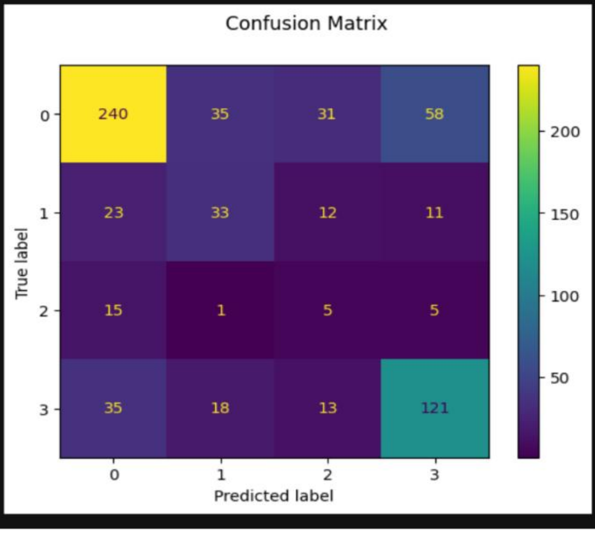
- Resnet 18 Results(Fig.10)



```
              precision    recall  f1-score   support

           0       0.77      0.66      0.71       364
           1       0.38      0.42      0.40        79
           2       0.08      0.19      0.11        26
           3       0.62      0.65      0.63       187

    accuracy                           0.61       656
   macro avg       0.46      0.48      0.46       656
weighted avg       0.65      0.61      0.63       656
```



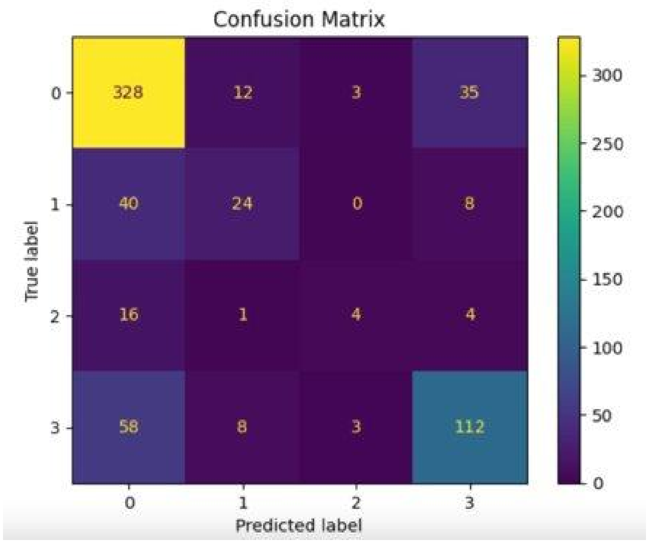
- Self-designed CNN Model

The final results(Fig.11 and Fig.12 and Fig.13):

```
Test accuracy: 0.7134146094322205
              precision    recall  f1-score   support

           0       0.74      0.87      0.80       378
           1       0.53      0.33      0.41        72
           2       0.40      0.16      0.23        25
           3       0.70      0.62      0.66       181

    accuracy                           0.71       656
   macro avg       0.59      0.49      0.52       656
weighted avg       0.70      0.71      0.70       656
```



.Fig.11

```
Classification report for Mono:
              precision    recall  f1-score   support

           0       0.73      0.92      0.81       139
           1       0.75      0.26      0.39        34
           2       0.38      0.21      0.27        14
           3       0.81      0.71      0.76        76

    accuracy                           0.74       263
   macro avg       0.66      0.53      0.56       263
weighted avg       0.73      0.74      0.71       263
```

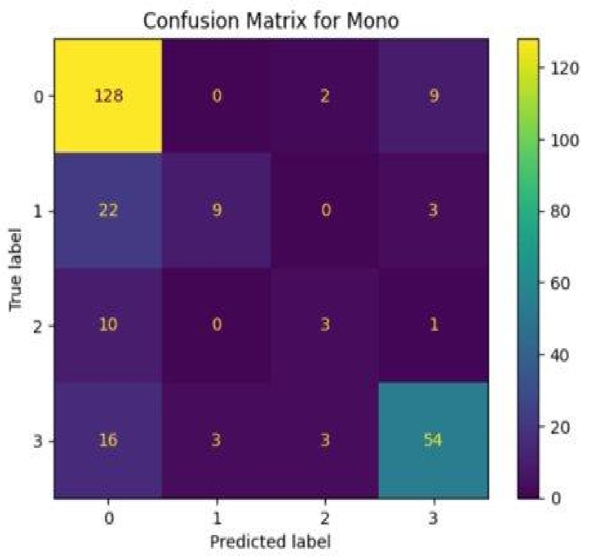


Fig.12

```
Classification report for Poly:
              precision    recall  f1-score   support

           0       0.75      0.84      0.79       239
           1       0.45      0.39      0.42        38
           2       0.50      0.09      0.15        11
           3       0.63      0.55      0.59       105

    accuracy                           0.70       393
   macro avg       0.58      0.47      0.49       393
weighted avg       0.68      0.70      0.68       393
```

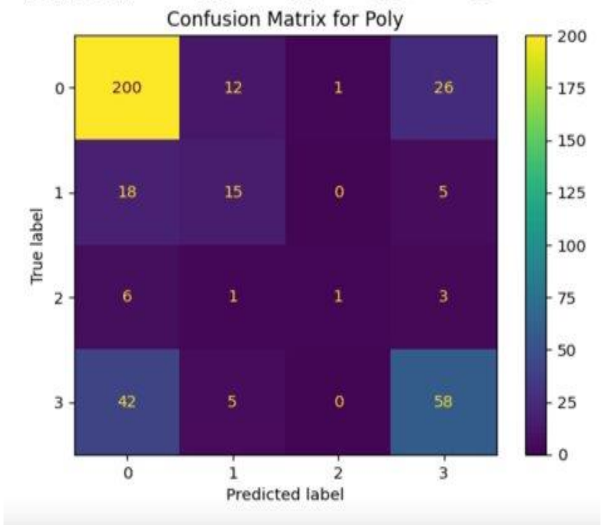


Fig.13

- SVM method

The final result are(Fig 14 and Fig 15 and Fig 16):

```
              precision    recall  f1-score   support

           0       0.75      0.91      0.82       377
           1       0.58      0.31      0.40        72
           2       0.33      0.05      0.09        20
           3       0.78      0.66      0.71       187

    accuracy                           0.75       656
   macro avg       0.61      0.48      0.51       656
weighted avg       0.73      0.75      0.72       656

Accuracy: 74.54%
```

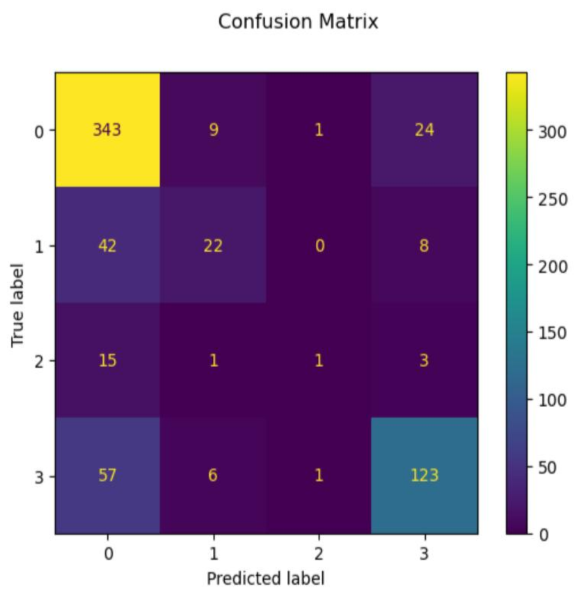
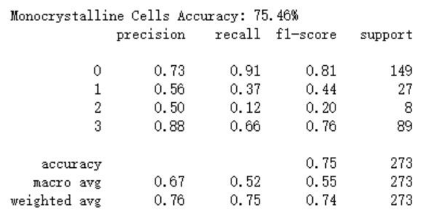


Fig 14



```
Monocrystalline Cells Accuracy: 75.46%
              precision    recall  f1-score   support

           0       0.73      0.91      0.81       149
           1       0.56      0.37      0.44        27
           2       0.50      0.12      0.20         8
           3       0.88      0.66      0.76        89

    accuracy                           0.75       273
   macro avg       0.67      0.52      0.55       273
weighted avg       0.76      0.75      0.74       273
```

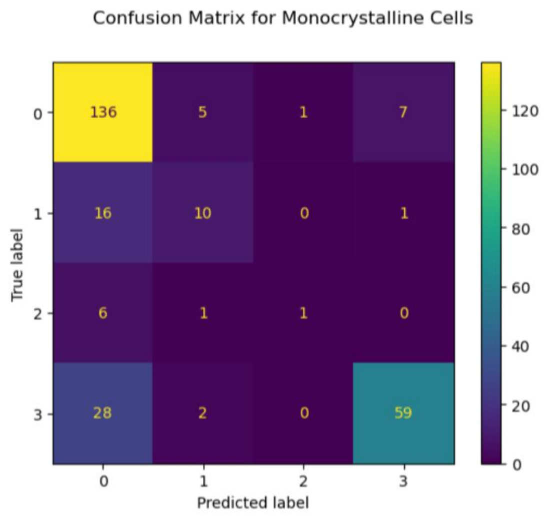
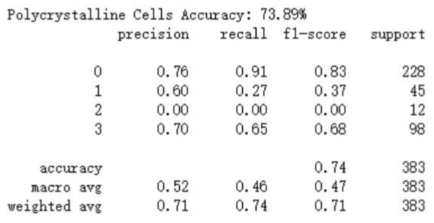


Fig.15



```
Polycrystalline Cells Accuracy: 73.89%
              precision    recall  f1-score   support

           0       0.76      0.91      0.83       228
           1       0.60      0.27      0.37        45
           2       0.00      0.00      0.00        12
           3       0.70      0.65      0.68        98

    accuracy                           0.74       383
   macro avg       0.52      0.46      0.47       383
weighted avg       0.71      0.74      0.71       383
```

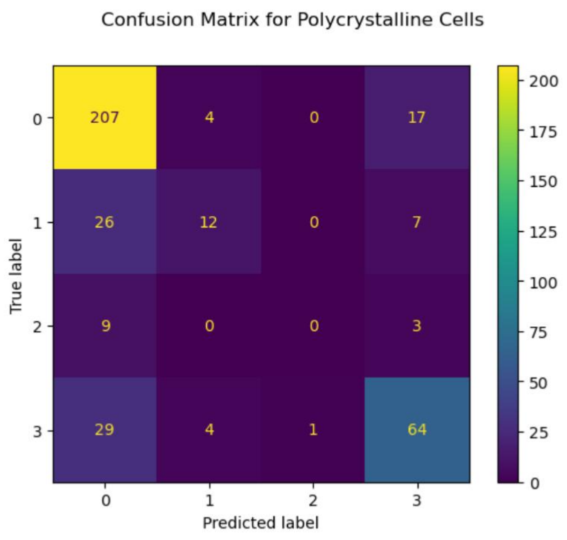
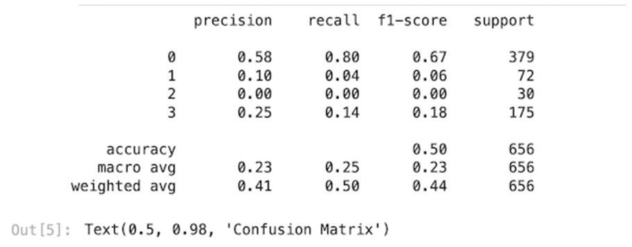


Fig.16

The accuracy rate of monocrystalline is higher than that o f polycrystalline possibly because polycrystalline cells have more variation within the class compared to monocrystalline cells, this could lead to lower classification accuracy. And als o may because HOG might be more effective for capturing th e characteristics of monocrystalline cells.

- BEiT with fine-tuning result

The performance of the BEiT model with fine-tuning wasn't as good as expected when compared to other methods. Its accuracy could only maintain around 50%, and its F1 score was even worse. I think this is related to the training data used in the original model. BEiT was trained with a vast array of different types of object images, but what it learned from them is not very helpful for the specific task of classifying solar panels. For highly specialized classification tasks, fine-tuning doesn't perform well.



```
              precision    recall  f1-score   support

           0       0.58      0.80      0.67       379
           1       0.10      0.04      0.06        72
           2       0.00      0.00      0.00        30
           3       0.25      0.14      0.18       175

    accuracy                           0.50       656
   macro avg       0.23      0.25      0.23       656
weighted avg       0.41      0.50      0.44       656

Out[5]: Text(0.5, 0.98, 'Confusion Matrix')
```

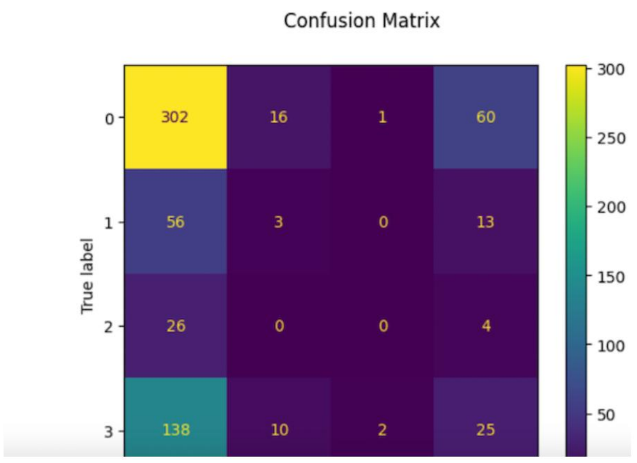
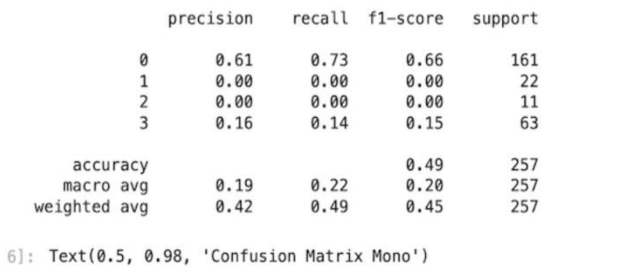


Fig.17



```
              precision    recall  f1-score   support

           0       0.61      0.73      0.66       161
           1       0.00      0.00      0.00        22
           2       0.00      0.00      0.00        11
           3       0.16      0.14      0.15        63

    accuracy                           0.49       257
   macro avg       0.19      0.22      0.20       257
weighted avg       0.42      0.49      0.45       257

6]: Text(0.5, 0.98, 'Confusion Matrix Mono')
```

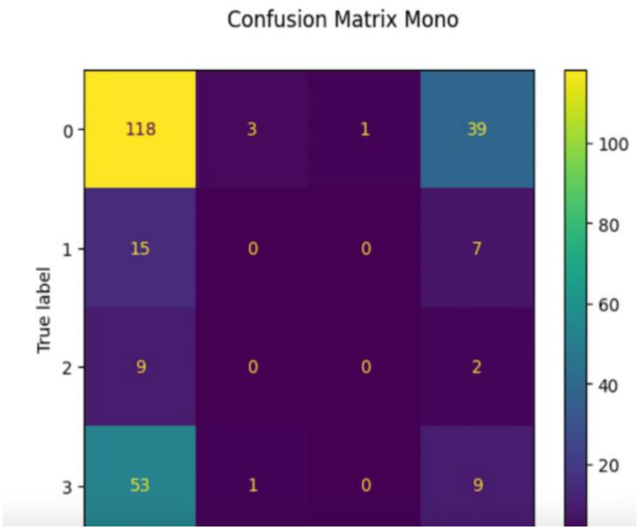


Fig.18

```
              precision    recall  f1-score   support

           0       0.56      0.84      0.68       218
           1       0.12      0.06      0.08        50
           2       0.00      0.00      0.00        19
           3       0.36      0.14      0.20       112

    accuracy                           0.51       399
   macro avg       0.26      0.26      0.24       399
weighted avg       0.42      0.51      0.44       399

: Text(0.5, 0.98, 'Confusion Matrix Poly')
```

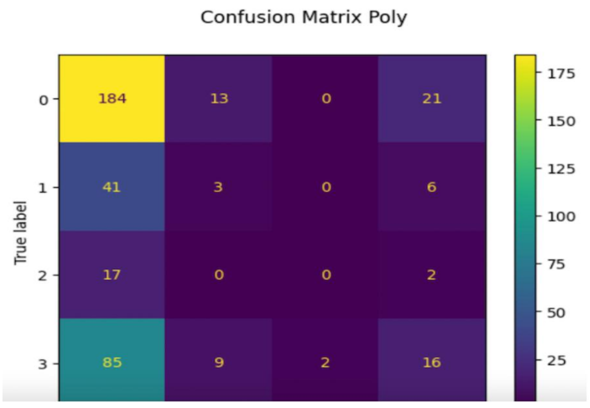


Fig.19

- AlexNet Result

Compare with other methods mentioned in our report. AlexNet achieved a not very good performance, but we can still find some interesting things from the evaluation results. Since the defect probability is 0.33 and the data sample of 0.66 is very, very small, our model hardly attempts to predict these two categories. As a result, the macro f1 score is only 0.46, but the weighted f1 score is 0.67. However, we can see that for the data with real labels of 0.00 and 1.00, AlexNet predicted that F1 Sore was still very good, reaching 0.79 and 0.64 respectively. Maybe we need a better technology to handle data imbalance problem and improve AlexNet's performance.

For All Test Images(Fig.20):

```
              precision    recall  f1-score   support

           0       0.73      0.86      0.79       375
           1       0.63      0.31      0.41        78
           2       0.00      0.00      0.00        27
           3       0.64      0.64      0.64       176

    accuracy                           0.70       656
   macro avg       0.50      0.45      0.46       656
weighted avg       0.67      0.70      0.67       656
```

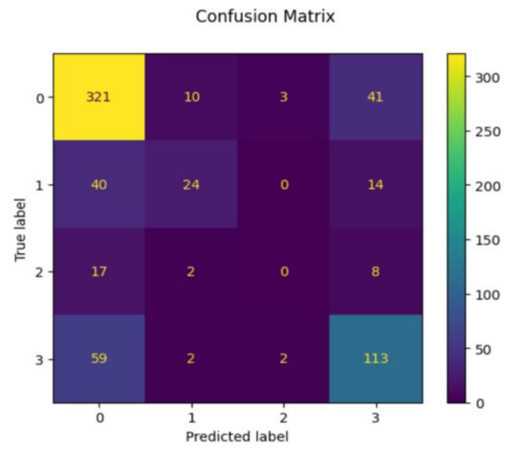


Fig.20

For mono(Fig.21):

```
              precision    recall  f1-score   support

           0       0.82      0.87      0.85       139
           1       0.00      0.00      0.00         0
           2       0.00      0.00      0.00         0
           3       0.85      0.66      0.74        83

    accuracy                           0.79       222
   macro avg       0.42      0.38      0.40       222
weighted avg       0.83      0.79      0.81       222
```

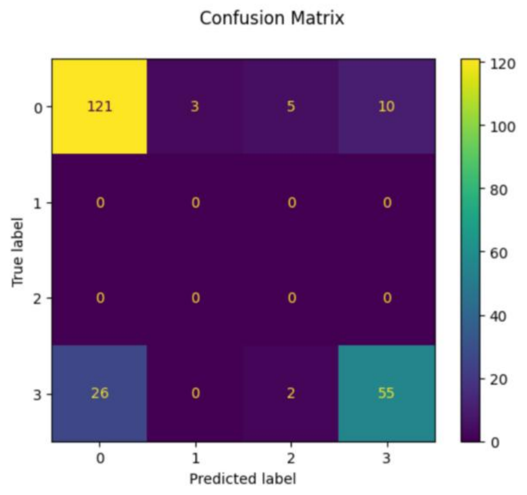


Fig.21

For Poly(Fig.22):

```
              precision    recall  f1-score   support

           0       0.86      0.85      0.86       236
           1       0.00      0.00      0.00         0
           2       0.00      0.00      0.00         0
           3       0.66      0.62      0.64        93

    accuracy                           0.79       329
   macro avg       0.38      0.37      0.37       329
weighted avg       0.80      0.79      0.79       329
```

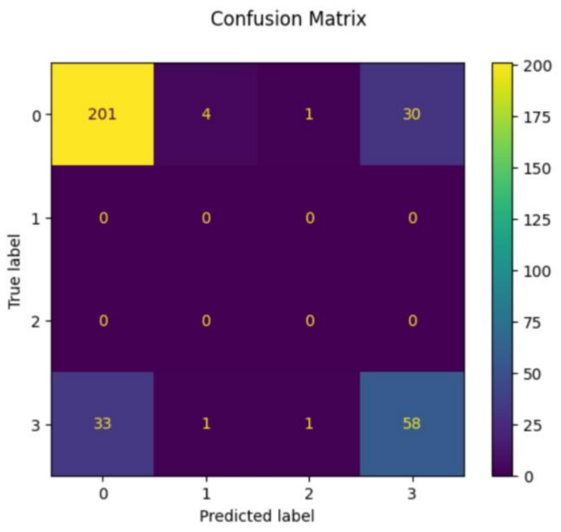


Fig.22

V. DISCUSSION

Based on our results, we can compare and find that the SVM model has relatively the best overall results, with his accuracy of 74.54%, and the CNN architecture with four layers of preprocessing has an accuracy of 71.54%. the accuracy of Alexnet and Resnet are at 70% and 61%, respectively. the BEiT with fine-tuning has an accuracy of 50%. We believe that there may be several reasons for this result: First, theoretically, resnet is better suited for this task from an architectural aspect and deserves to get better data, but since resnet is more difficult to train compared to the other models, it does not give the best results. In subsequent work, we will try to use pre-trained resnet models thus improving the accuracy. Secondly we can find that the pre-processed CNN model is not bad. This is due to the fact that CNN models have excellent feature extraction capabilities and also local sensing mechanisms. Adding the right preprocessing method, the efficiency of the CNN model will be excellent. The accuracy of BEiT with fine-tuning is only 50%, which means that this method is not suitable for this type of problem.

VI. Conclusion

In this paper,we proposed five methods on how to classify cell images according to their probability of defectiveness, and among these five methods we can find that, in general, the SVM model is better compared to other models. Through these five methods, we can detect the damage of solar panels relatively efficiently, which makes the maintenance and upkeep of solar panels more efficient, and improves its working efficiency to a certain extent, which improves the working efficiency of the whole solar photovoltaic industry. Our work also has some limitations, first of all, we did not make an attempt on more in-depth algorithms.We believe that more optimized CNN models such as pre-trained resnet models may be more suitable for this project. Secondly, the efficiency problem of solar panels may be affected by a number of factors rather than just its damage condition. In future work, firstly, the damage detection algorithm can be optimized, and secondly, the various types of damage can be refined, so that more professional engineers can solve the problem more easily and quickly.We still have a long way to go in this field.

References

[1] ELPV Dataset. A Benchmark for Visual Identification of Defective Solar Cells in Electro-luminescence Imagery. https://github.com/zae-bayern/elpv-dataset.

[2] Deitsch, S.; Buerhop-Lutz, C.; Sovetkin, E.; Steland, A.; Maier, A.; Gallwitz, F.; Riess, C. Segmentation of photovoltaic module cells in uncalibrated electroluminescence images. Mach. Vision. Appl. 2021, 32, 84.

[3] Kather et al. "Deep learning can detect microscopic lung cancer in CT scans." Nature Medicine.

[4] Sun et al. "Multiple instance learning convolutional neural networks for lung cancer diagnosis." IEEE.

[5] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[6] Géron, Aurélien. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow. " O'Reilly Media, Inc.", 2022.

[7] Mehta, Smit, Chirag Paunwala, and Bhaumik Vaidya. "CNN based traffic sign classification using Adam optimizer." 2019 international conference on intelligent computing and control systems (ICCS). IEEE, 2019.

[8] Hangbo Bao, Li Dong, Songhao Piao, Furu Wei. BEiT: BERT Pre-Training of Image Transformers. https://arxiv.org/abs/2106.08254

[9] Microsoft/beit-base-patch16-224 https://huggingface.co/microsoft/beit-base-patch16-224