Projektarbeit Kurs „Maschinelles Lernen“

Erkennung von Malaria-inzierten Zellen

ANDRES
OSCAR

Inhalt

1. Ausgangslage
2. Vorgehensweise
3. Daten Pre-Processing
4. Datenanalyse
5. Ergebnisse
6. Empfehlungen

Ausgangslage

- Mikroskopische Bilder von Malaria-infizierten Zellen
- Klassifizierung der Zellen
- Ziel = die Zelle sind richtig klassifiziert
- Analyse der Daten/Bildern sowie Untersuchung der relevanten Modellen
- Handlungsempfehlung zur Verbesserung der Vorhersage

Vorgehensweise

1. Pre-Processing der Daten
2. Diskussion in der Gruppe hinsichtlich möglicher Probleme und Arbeitsteilung
3. Datenexploration
4. Vorbereitung der Daten bzw. Feature Reduktion Verfahren (PCA)
5. Datasplit nach Test und Trainingsdaten
6. Klassifikation mit SVM und Logistische Regression
7. (Convolutional) Neuronal Network
8. Diskussion und Verdichtung der Ergebnisse
9. Ableiten von Handlungsempfehlung

Pre-Processing der Bildern

```python
y = []
X = []

images_pos = []
images_neg = []

# Parasitized cells
for f in glob.glob("./cell_images/Parasitized/*.png"):
    img = cv2.imread(f)
    img_padded = img_prep(img)
    img_resized = cv2.resize(img_padded,(128,128))
    img_gray = cv2.cvtColor(img_resized, cv2.COLOR_BGR2GRAY) / 255
    X.append(img_gray)
    images_pos.append(img)
    y.append(1)

# Uninfected cells
for f in glob.glob("./cell_images/Uninfected/*.png"):
    img = cv2.imread(f)
    img_padded = img_prep(img)
    img_resized = cv2.resize(img_padded,(128,128))
    img_gray = cv2.cvtColor(img_resized, cv2.COLOR_BGR2GRAY) / 255
    X.append(img_gray)
    images_neg.append(img)
    y.append(0)
```

```python
np.shape(images_pos)
```

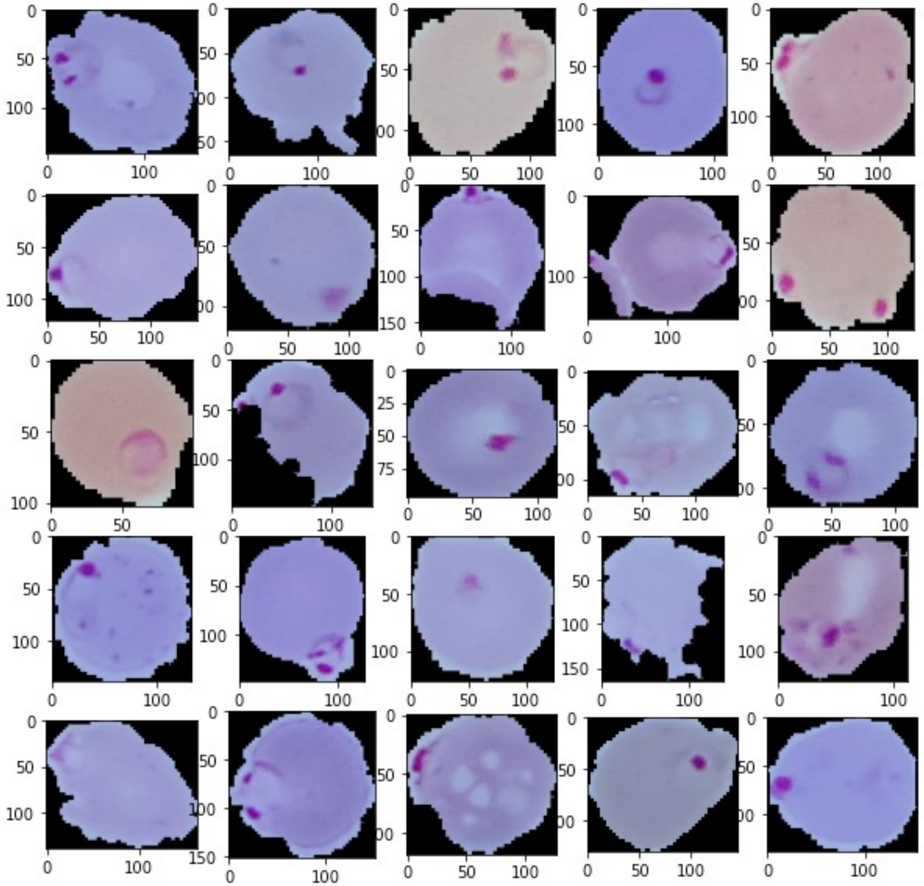
```
(13779,)
```

```python
np.shape(images_neg)
```

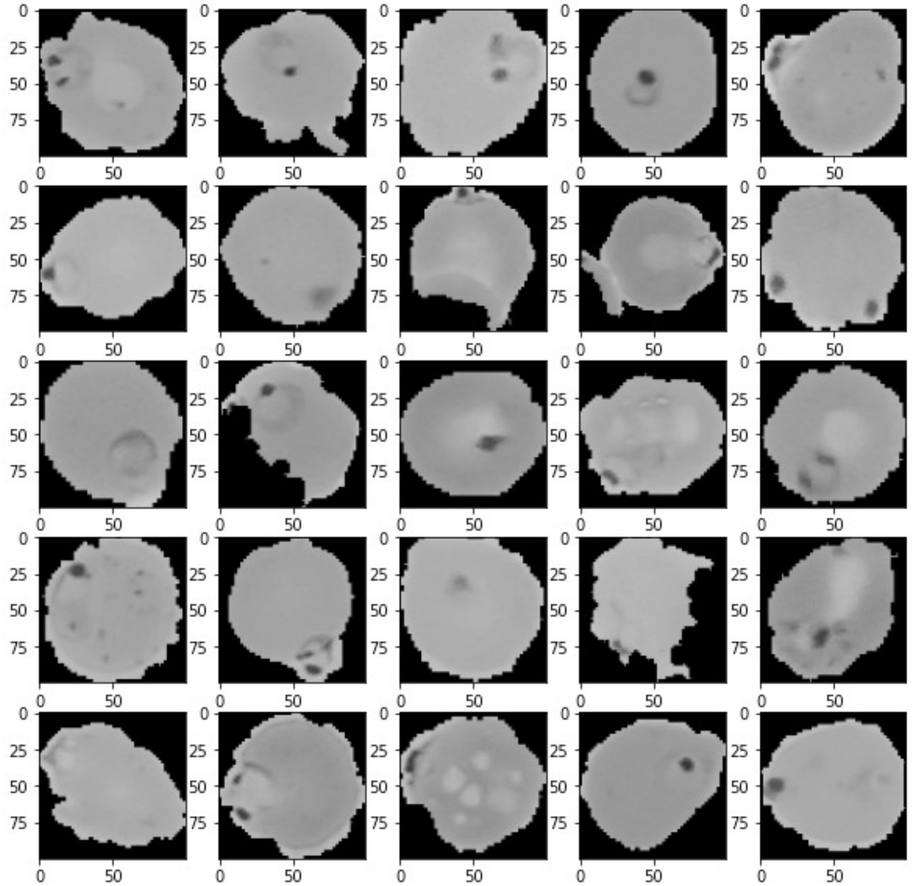
```
(13779,)
```

```python
def img_prep (img):

    h, w = img.shape[:2]

    if h > w:
        max_len = h
        diff_hori = max_len - w
        pad_left = diff_hori//2
        pad_right = diff_hori - pad_left
        img_padded = cv2.copyMakeBorder(img, 0, 0, pad_left, pad_right, cv2.BORDER_CONSTANT, value=0)
    elif h < w:
        max_len = w
        diff_vert = max_len - h
        pad_top = diff_vert//2
        pad_bottom = diff_vert - pad_top
        img_padded = cv2.copyMakeBorder(img, pad_top, pad_bottom, 0, 0, cv2.BORDER_CONSTANT, value=0)
    elif h == w:
        img_padded = img


    return img_padded
```

```python
fig, axes = plt.subplots(5, 5, figsize=(10, 10))
i = 0
for ax in axes.flatten():
    #i = np.random.randint(0, len(images_pos))
    image = images_pos[im[i]]
    #image = image.reshape(400, 400)
    ax.imshow(image, cmap='gray', vmin=0, vmax=255, label=y)
    i += 1
```



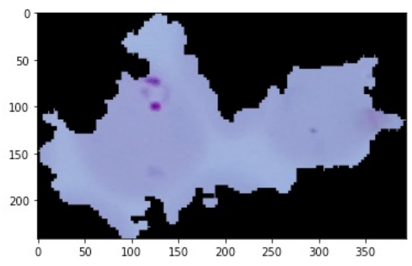
```python
fig, axes = plt.subplots(5, 5, figsize=(10, 10))
i = 0
for ax in axes.flatten():
    image = X[im[i], :]
    image = image.reshape(100, 100)
    ax.imshow(image, cmap='gray', vmin=0, vmax=1, label=y)
    i += 1
    #ax.axis("off")
```



Outliers

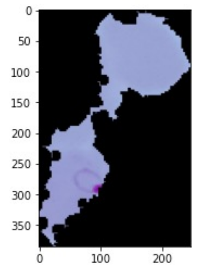
```
plt.imshow(images_pos[index1])
```

<matplotlib.image.AxesImage at 0x7fb9f98ae5b0>



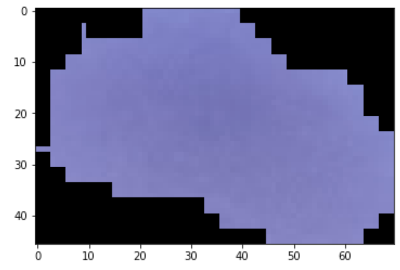
```
plt.imshow(images_pos[index2])
```

<matplotlib.image.AxesImage at 0x7fb9f8bcd910>



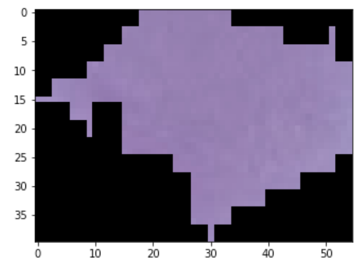
```
plt.imshow(images_pos[index3])
```

<matplotlib.image.AxesImage at 0x7fb9f7b1a700>



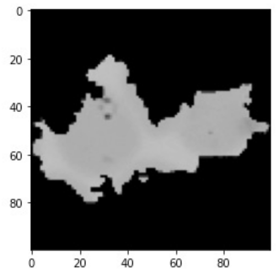
```
plt.imshow(images_pos[index4])
```

<matplotlib.image.AxesImage at 0x7fb9f9da2700>



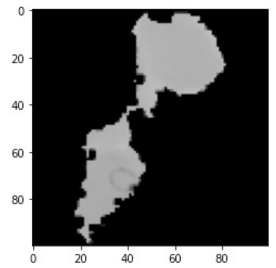
```
plt.imshow(X[index1], cmap='gray', vmin=0, vmax=1)
```

<matplotlib.image.AxesImage at 0x7fb9f737ed60>



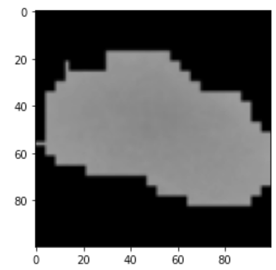
```
plt.imshow(X[index2], cmap='gray', vmin=0, vmax=1)
```

<matplotlib.image.AxesImage at 0x7fb5b424edf0>



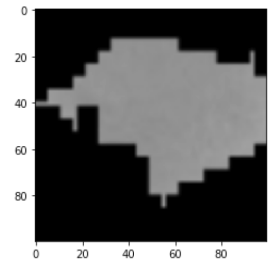
```
plt.imshow(X[index3], cmap='gray', vmin=0, vmax=1)
```

<matplotlib.image.AxesImage at 0x7fb9f913bdf0>



```
plt.imshow(X[index4], cmap='gray', vmin=0, vmax=1)
```

<matplotlib.image.AxesImage at 0x7fb5afa7c8b0>



Train-Test Split

Test-Train Split

```
]:
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_state = 10, shuffle=True,
                                                    stratify=y)

X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)
```

```
np.shape(X)
```

(27558, 128, 128)

```
np.shape(X_train)
```

(22046, 16384)

```
np.shape(X_test)
```

(5512, 16384)

```
np.shape(y_train)
```

(22046,)

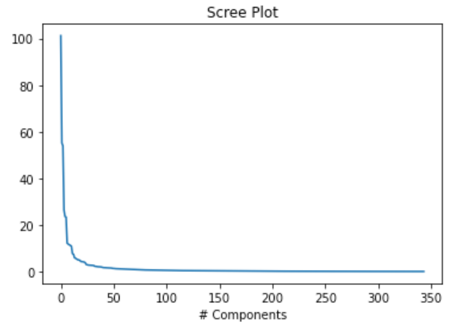
```
np.shape(y_test)
```

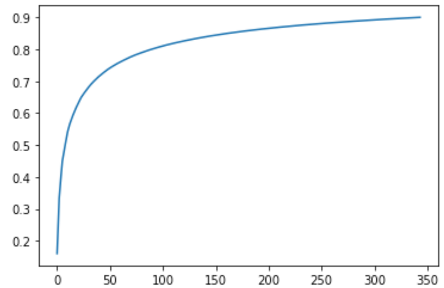
(5512,)

Feature Engineering/Reduction

```
plt.figure()
plt.plot(pca_reduced.explained_variance_)
plt.xlabel("# Components")
plt.title("Scree Plot")

plt.figure()
plt.plot(np.cumsum(pca_reduced.explained_variance_ratio_))
```

```
[<matplotlib.lines.Line2D at 0x7fdacd153b80>]
```



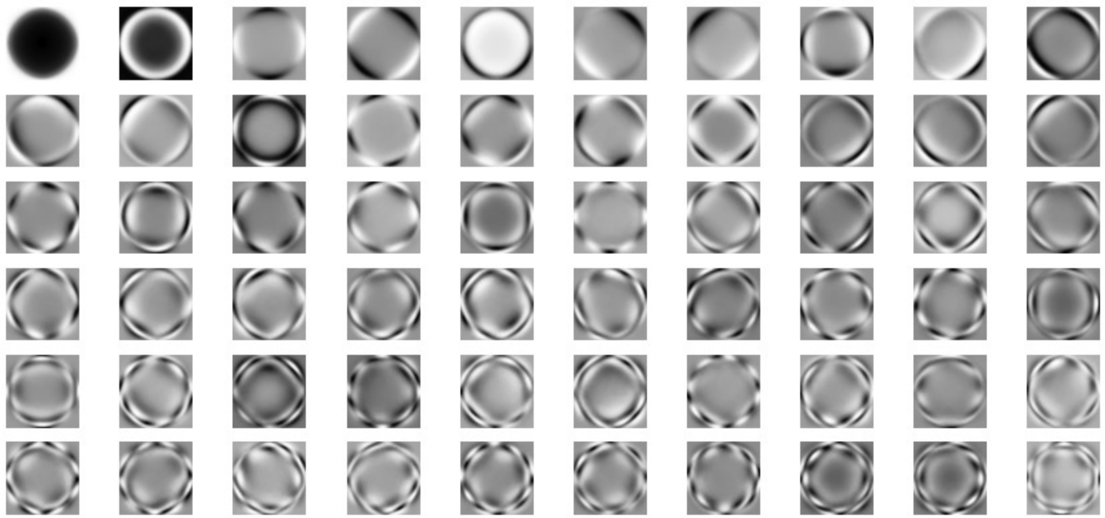



```
pca_reduced = PCA(n_components = 0.9)
```

```
pca_reduced.fit(X_train)
```

```
PCA(n_components=0.9)
```

```
fig, axes = plt.subplots(6, 10, figsize=(15, 7))

for i, ax in enumerate(axes.flatten()):
    if i==0:
        component = pca_reduced.mean_
    else:
        component = pca_reduced.components_[i-1, :]
    ax.imshow(component.reshape(128, 128), cmap="Greys")
    ax.axis("off")
```



```
print(pca_reduced.components_.shape)
```

```
(344, 16384)
```

Support Vector Classifier

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, plot_confusion_matrix
```

```
svm_pca = SVC(kernel="rbf", gamma="scale")
```

```
# Modell fitten
svm_pca.fit(X_train_transformed, y_train)
```

```
SVC()
```

```
# Modell evaluieren
svm_pca.score(X_test_transformed, y_test)
```

```
0.7117198838896952
```

```
svm_pca.get_params(deep=True)
```

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

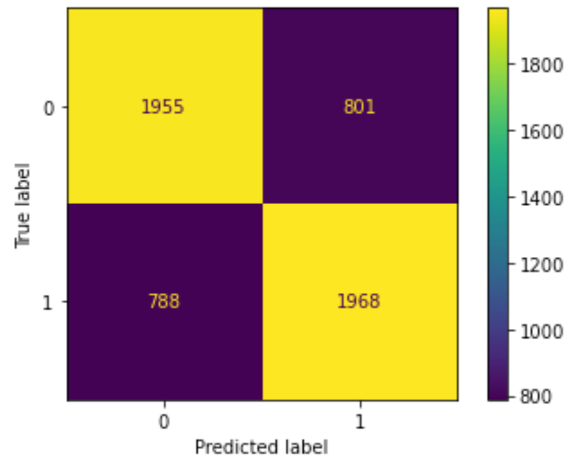
```
y_pred = svm_pca.predict(X_test_transformed)
```

```
accuracy_score(y_test, y_pred)
```

```
0.7117198838896952
```

```
plot_confusion_matrix(svm_pca, X_test_transformed, y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
```



Logistische Regression + GridSearch

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
```

```
param_grid = {
    'penalty':["l2"], #"l1",
    'C': [10e-4, 10e-3, 10e-2, 10e-1, 1.0] ,
    'solver': [ 'lbfgs', 'liblinear', 'newton-cg', 'sag', 'saga']
}

# Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.
```

```
def make_logistic_regression(C=0.001, solver="lbfgs", penalty='l2'):

    log_model = LogisticRegression(C=C, solver=solver , penalty=penalty, random_state=10, max_iter=200)

    return log_model
```

```
meta_model_hp_tuning = GridSearchCV(
    make_logistic_regression(),
    param_grid=param_grid,
)
```

```
meta_model_hp_tuning.fit(X_train_transformed, y_train)
```

```
GridSearchCV(estimator=LogisticRegression(C=0.001, max_iter=200,
                                          random_state=10),
             param_grid={'C': [0.001, 0.01, 0.1, 1.0, 1.0], 'penalty': ['l2'],
                         'solver': ['lbfgs', 'liblinear', 'newton-cg', 'sag',
                                    'saga']})
```

```
meta_model_hp_tuning.best_score_
```

```
0.6610723376066991
```

```
meta_model_hp_tuning.best_params_
```

```
{'C': 0.001, 'penalty': 'l2', 'solver': 'lbfgs'}
```

Cross-Validation

```
from sklearn.model_selection import cross_val_score

train_score = np.mean(cross_val_score(meta_model_hp_tuning, X_train_transformed, y_train))
```

```
train_score
```

```
0.6609816140866266
```

```
best_poly_regression = meta_model_hp_tuning.best_estimator_
```

```
best_poly_regression
```

```
LogisticRegression(C=0.001, max_iter=200, random_state=10)
```

```
y_test_pred = best_poly_regression.predict(X_test_transformed)
```

```
accuracy_test = (y_test_pred == y_test).mean()
accuracy_test
```

```
0.6696298984034833
```

Generalisierung

Neural Networks

Prediction using NNs

```python
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import TensorDataset, DataLoader
```

```python
class SimpleNet(nn.Module):
    def __init__(self, in_features=sample_width*sample_height):
        super(SimpleNet, self).__init__()

        self.layer1 = nn.Linear(in_features=in_features, out_features=270)
        self.layer2 = nn.Linear(in_features=270, out_features=80)
        self.layer3 = nn.Linear(in_features=80, out_features=2)

    def forward(self, x):
        x = self.layer1(x)
        x = F.relu(x)
        x = self.layer2(x)
        x = F.relu(x)
        x = self.layer3(x)
        return x
```

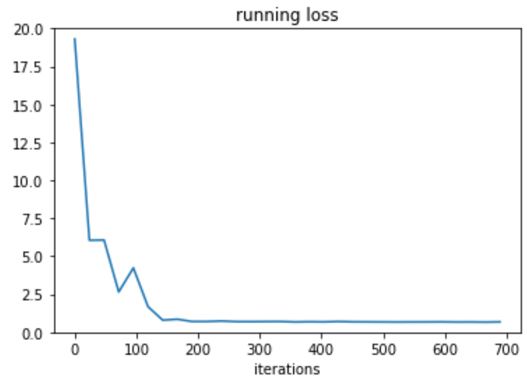
prediction using CNNs

```python
import torch.nn as nn
import torch.nn.functional as F


class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5, padding=2)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.conv3 = nn.Conv2d(16, 24, 5)
        self.pool3 = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(24 * 7 * 7, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 2)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
        x = F.relu(self.conv2(x))
        x = self.pool2(x)
        x = F.relu(self.conv3(x))
        x = self.pool3(x)
        x = x.view(-1, 24 * 7 * 7)  # ähnlich wie reshape
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Neuronal Networks

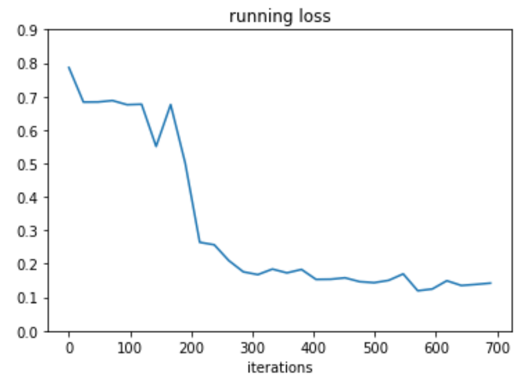
Prediction using NNs

Hidden layers:	*2*
Input neurons:	*6400*
Hidden neurons L1:	*270*
Hidden neurons L2:	*80*
Output neurons:	*2*
Max. Accuracy:	*60%*



prediction using CNNs

Convolutional layers:	*3*
Hidden layers:	*2*
Input neurons:	*1176*
Hidden neurons L1:	*120*
Hidden neurons L2:	*84*
Output neurons:	*2*
Max. Accuracy:	*94%*



Ergebnisse

- Mit SVC und Logistischer Regression ist die Generalisierung-Score ca. 70%
- Bei der Convolutional Neuronal Netzwerken ist die Einschätzung auf 94% Accuracy deutlich besser
- Die Konvergenzgeschwindigkeit des Fehlers ist abhängig von der Anzahl von Neuronen in der Versteckten Schichten. Die Größe des Fehlers kann man mit mehreren versteckten Schichten leicht reduzieren.
- Kleine Details und Merkmale auf der Training- Samples sind leichter mit Faltungsnetzwerke zu Erkennen

Empfehlungen

- Falls mehr Rechenressourcen vorhanden sind, höhere Auslösung der Bildern benutzen
- Benutzung der RGB Spektrum, um eine genauere Vorhersage zu schätzen
- Die Architektur des Neuronale-Netzwerks zu optimieren
- Verteilung der Bildergröße