Convolutional Neural Network (CNN) for Wafer Edge Automatic Defect Classification (ADC)

Goal

- Predict either of 2 classes: None (No Chipping) or Chipping
  - 0: Represents None
  - 1: Represents Chipping

Data (Pre-Classified/Sorted Manually)

- 2265 Edge Normal images from Klarity and 8 images from previous student's ppt slides
- 2235 None Images and 30 Chipping Images
- In Sample Model Training (Training + Validation): 60 Images
  - Training (In Sample): 80% of 60 = 48 images (24/24 None/Chipping)
  - Validation (In Sample): 20% of 60 = 12 images (6/6 None/Chipping)
  - Testing (Out of Sample): 8 images
  - Rest of the 2205 None images can be used for further testing

Model Employed

- Using deep learning and CNN, the model automatically extracts features from the images and learns to distinguish whether an image fed into the model exhibits signs of chipping
- To improve accuracy and speed of model training, transfer learning was used
  - Transfer learning is using models previously created by very smart researchers and machine learning engineers that have then been trained on millions of images such as cats, dogs, etc.
  - Even though their models were used to classify other kinds of images like animals or vehicles, we can leverage their results for our own use case
  - By cutting off a small part of their model, we can customise their models to classify either wafer chipping or not
- A few popular models available are VGG16, ResNet, Inception, NASNet, YOLO, etc. and they vary in size, complexity, accuracy, speed, among other things

In [1]:
```
import pickle
import pandas as pd
import numpy as np
import cv2
import random
import gc
import os
```

```
import glob
import time

import matplotlib.pyplot as plt
plt.style.use('dark_background')

def duration(start):
    DURATION = round((time.time() - start)/60, 2)
    print(f'\n{DURATION} mins')
```

In [172…

```
from keras.models import Model, Sequential, load_model
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout
from keras.layers import RandomFlip, RandomRotation, RandomZoom, RandomContrast
from keras.callbacks import TensorBoard, LearningRateScheduler, ModelCheckpoint, Reduce

from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

from keras.applications.vgg16 import VGG16
from keras.applications.inception_resnet_v2 import InceptionResNetV2
# from keras.applications.nasnet import NASNetLarge
```

In [184…

```
NUM_CLASSES = 2
SEED = 69
IMG_SIZE = 256
BATCH_SIZE = 8
VAL_SPLIT = 0.2

DEFECT_LIST = ['none', 'chipping']
DEFECT_MAPPING = dict(enumerate(DEFECT_LIST))
DEFECT_MAPPING
```

Out[184… {0: 'none', 1: 'chipping'}

In [185…

```
EN_path = os.path.join(os.getcwd(), 'klarf-EN')
chipping_len = len(glob.glob(os.path.join(EN_path, 'chipping', '*')))
chipping_len
```

Out[185… 30

Load Images from Drive Locally

In [94]:

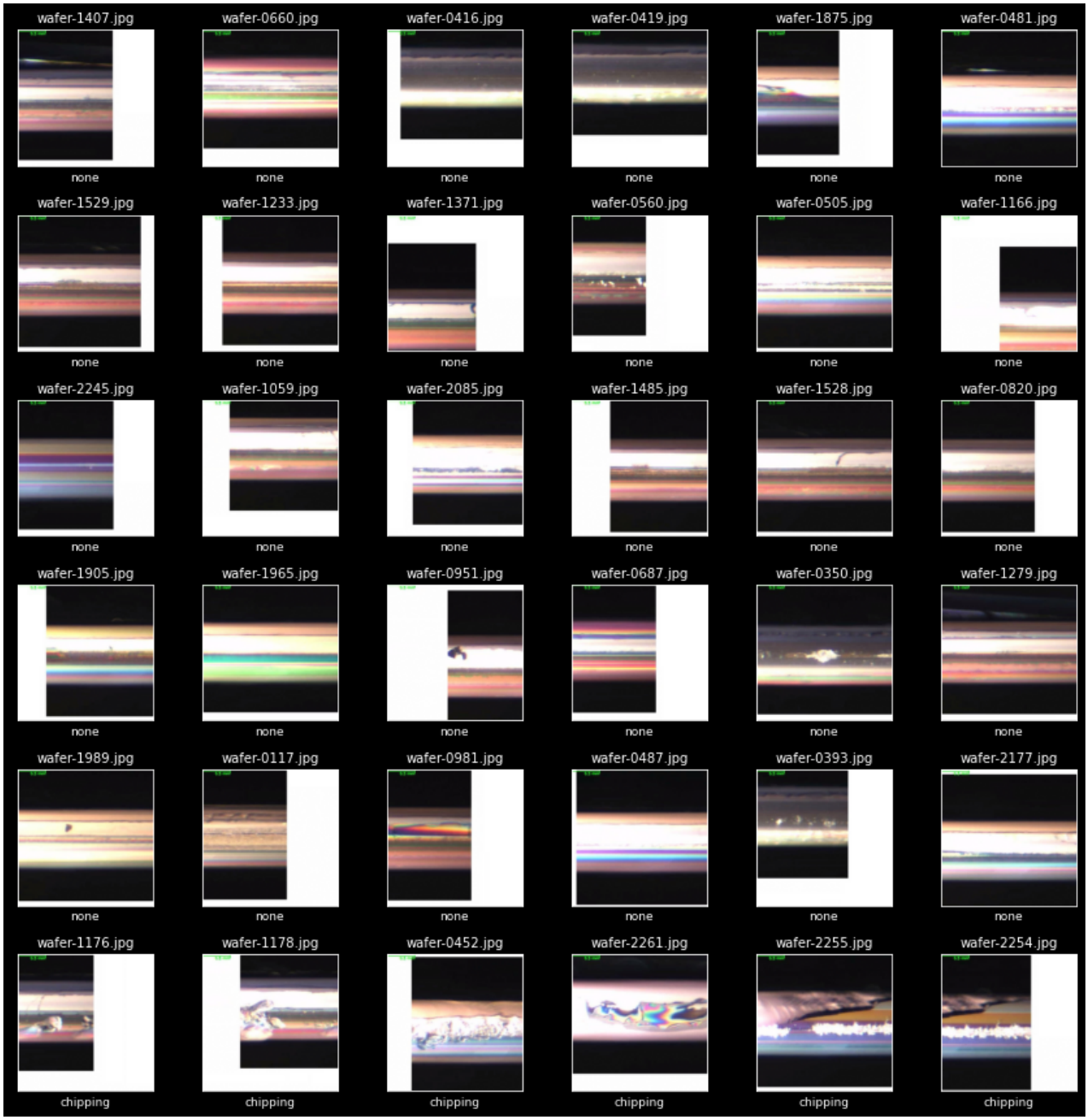
```
x_train, y_train, x_test, y_test, img_paths = [], [], [], [], []

for defect in DEFECT_LIST:
    sample = random.sample(glob.glob(os.path.join(EN_path, defect, '*')), chipping_len)
    for i in range(chipping_len):
        img_paths.append(sample[i])
        img = cv2.imread(sample[i], cv2.IMREAD_COLOR)
        img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
        img = img / 255.0
        if i < chipping_len*(1-VAL_SPLIT):
            x_train.append(img)
            y_train.append(defect)
```

```
        else:
            x_test.append(img)
            y_test.append(defect)

x_train, y_train, x_test, y_test = np.array(x_train), np.array(y_train), np.array(x_tes

p = np.random.permutation(len(x_train))
x_train, y_train = x_train[p], y_train[p]

# img_paths
```

Display 36/60 of the Training/Validation Images

In [168…

```
fig, ax = plt.subplots(nrows=6, ncols=6, figsize=(12, 12))
ax = ax.ravel(order='C')
for i in range(36):
    img = img_paths[i]
    img = cv2.imread(img, cv2.IMREAD_COLOR)
    img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
    img = img / 255.0

    ax[i].imshow(img)

    title = img_paths[i].split('\\')[-1]
    ax[i].set_title(title, fontsize=10)

    xlabel = img_paths[i].split('\\')[-2]
    ax[i].set_xlabel(xlabel, fontsize=9)

    ax[i].set_xticks([])
    ax[i].set_yticks([])

plt.tight_layout()
plt.show()
```



In [96]:
```
def encode(y): return np.array([0 if label=='none' else 1 for label in y])

y_train_encoded = encode(y_train)
y_test_encoded = encode(y_test)
```

In [97]:
```
from tensorflow.keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train_encoded)
y_test_one_hot = to_categorical(y_test_encoded)
```

Load Pretrained Model and Prepare for my own Model Customisation

---

- The pretrained model (built by others) used here is called VGG16
- After loading the model, a few more layers are added to customize to our problem
- The model then trains for 20 epochs (or steps/iterations) and automatically saves the best models
- We can also graph out the learning process (higher accuracy is better, lower loss is better)

In [98]:
```
pretrained_model = VGG16(weights='imagenet', include_top=False, input_shape=(IMG_SIZE,
# pretrained_model = InceptionResNetV2(weights='imagenet', include_top=False, input_sha

# Make loaded layers as non-trainable. This is important as we want to work with pre-tr
for layer in pretrained_model.layers: layer.trainable = False

# pretrained_model.summary() # Trainable parameters will be 0
```

In [110…
```
x = pretrained_model.output
# x = RandomFlip()(x)
# x = RandomZoom(0.2)(x)
# x = RandomContrast(0.2)(x)
# x = Conv2D(64, (3,3), activation='relu')(x)
x = Flatten()(x)
x = Dense(32, activation='relu', kernel_initializer='he_uniform')(x)
x = Dense(64, activation='relu', kernel_initializer='he_uniform')(x)

x = Dropout(0.5)(x)
x = Dense(2, activation='softmax')(x)

cnn_model = Model(inputs=pretrained_model.input, outputs=x)
cnn_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy

##########################################

earlystop = EarlyStopping(monitor='val_loss', patience=5)
lr_stagnate = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-5
checkpoint_loss = ModelCheckpoint(f'models\\model_loss.h5', monitor='val_loss', verbose
checkpoint_acc = ModelCheckpoint(f'models\\model_acc.h5', monitor='val_accuracy', verbo

# Train the CNN model
hist = cnn_model.fit(x_train, y_train_one_hot, epochs=20, batch_size=BATCH_SIZE, valida
```

```
Epoch 1/20
6/6 [==============================] - 22s 4s/step - loss: 1.5907 - accuracy: 0.4583 - v
al_loss: 0.4588 - val_accuracy: 0.9167

Epoch 00001: val_loss improved from inf to 0.45883, saving model to models\model_loss.h5

Epoch 00001: val_accuracy improved from -inf to 0.91667, saving model to models\model_ac
c.h5
Epoch 2/20
6/6 [==============================] - 20s 3s/step - loss: 1.2963 - accuracy: 0.6042 - v
al_loss: 0.4260 - val_accuracy: 0.7500

Epoch 00002: val_loss improved from 0.45883 to 0.42595, saving model to models\model_los
s.h5

Epoch 00002: val_accuracy did not improve from 0.91667
Epoch 3/20
6/6 [==============================] - 20s 4s/step - loss: 0.5432 - accuracy: 0.6667 - v
al_loss: 0.3344 - val_accuracy: 0.8333
```

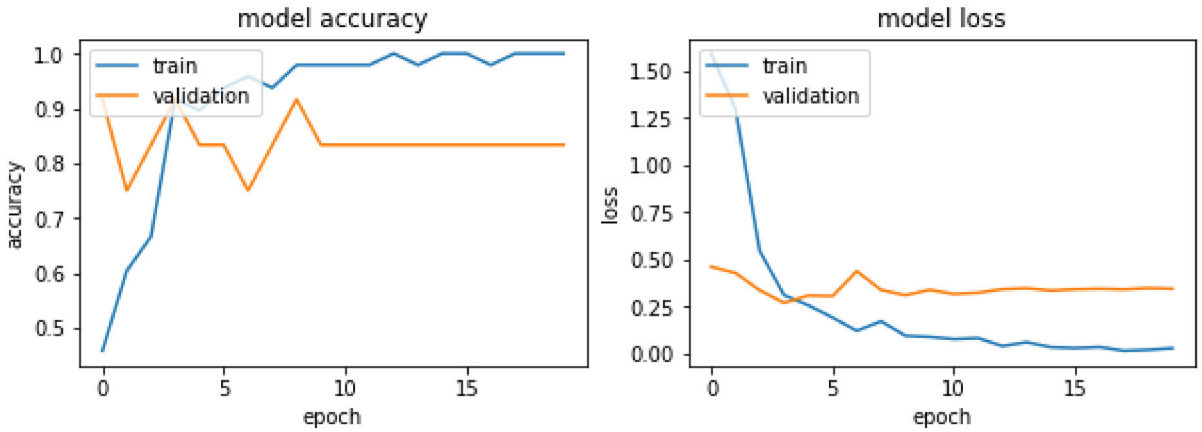
```
Epoch 00003: val_loss improved from 0.42595 to 0.33437, saving model to models\model_los
s.h5

Epoch 00003: val_accuracy did not improve from 0.91667
Epoch 4/20
6/6 [==============================] - 20s 3s/step - loss: 0.3085 - accuracy: 0.9167 - v
al_loss: 0.2670 - val_accuracy: 0.9167

Epoch 00004: val_loss improved from 0.33437 to 0.26699, saving model to models\model_los
s.h5

Epoch 00004: val_accuracy did not improve from 0.91667
Epoch 5/20
6/6 [==============================] - 20s 4s/step - loss: 0.2545 - accuracy: 0.8958 - v
al_loss: 0.3061 - val_accuracy: 0.8333

Epoch 00005: val_loss did not improve from 0.26699

Epoch 00005: val_accuracy did not improve from 0.91667
Epoch 6/20
6/6 [==============================] - 20s 3s/step - loss: 0.1893 - accuracy: 0.9375 - v
al_loss: 0.3030 - val_accuracy: 0.8333

Epoch 00006: val_loss did not improve from 0.26699

Epoch 00006: val_accuracy did not improve from 0.91667
Epoch 7/20
6/6 [==============================] - 20s 3s/step - loss: 0.1180 - accuracy: 0.9583 - v
al_loss: 0.4356 - val_accuracy: 0.7500

Epoch 00007: val_loss did not improve from 0.26699

Epoch 00007: val_accuracy did not improve from 0.91667
Epoch 8/20
6/6 [==============================] - 20s 3s/step - loss: 0.1692 - accuracy: 0.9375 - v
al_loss: 0.3360 - val_accuracy: 0.8333

Epoch 00008: val_loss did not improve from 0.26699

Epoch 00008: val_accuracy did not improve from 0.91667
Epoch 9/20
6/6 [==============================] - 20s 3s/step - loss: 0.0920 - accuracy: 0.9792 - v
al_loss: 0.3077 - val_accuracy: 0.9167

Epoch 00009: val_loss did not improve from 0.26699

Epoch 00009: val_accuracy did not improve from 0.91667
Epoch 10/20
6/6 [==============================] - 20s 3s/step - loss: 0.0861 - accuracy: 0.9792 - v
al_loss: 0.3362 - val_accuracy: 0.8333

Epoch 00010: val_loss did not improve from 0.26699

Epoch 00010: val_accuracy did not improve from 0.91667
Epoch 11/20
6/6 [==============================] - 20s 3s/step - loss: 0.0742 - accuracy: 0.9792 - v
al_loss: 0.3143 - val_accuracy: 0.8333

Epoch 00011: val_loss did not improve from 0.26699

Epoch 00011: val_accuracy did not improve from 0.91667
Epoch 12/20
6/6 [==============================] - 20s 3s/step - loss: 0.0794 - accuracy: 0.9792 - v
al_loss: 0.3210 - val_accuracy: 0.8333
```

```
Epoch 00012: val_loss did not improve from 0.26699

Epoch 00012: val_accuracy did not improve from 0.91667
Epoch 13/20
6/6 [==============================] - 20s 3s/step - loss: 0.0372 - accuracy: 1.0000 - v
al_loss: 0.3396 - val_accuracy: 0.8333

Epoch 00013: val_loss did not improve from 0.26699

Epoch 00013: val_accuracy did not improve from 0.91667
Epoch 14/20
6/6 [==============================] - 20s 4s/step - loss: 0.0575 - accuracy: 0.9792 - v
al_loss: 0.3447 - val_accuracy: 0.8333

Epoch 00014: val_loss did not improve from 0.26699

Epoch 00014: val_accuracy did not improve from 0.91667
Epoch 15/20
6/6 [==============================] - 20s 3s/step - loss: 0.0312 - accuracy: 1.0000 - v
al_loss: 0.3327 - val_accuracy: 0.8333

Epoch 00015: val_loss did not improve from 0.26699

Epoch 00015: val_accuracy did not improve from 0.91667
Epoch 16/20
6/6 [==============================] - 20s 3s/step - loss: 0.0267 - accuracy: 1.0000 - v
al_loss: 0.3392 - val_accuracy: 0.8333

Epoch 00016: val_loss did not improve from 0.26699

Epoch 00016: val_accuracy did not improve from 0.91667
Epoch 17/20
6/6 [==============================] - 20s 3s/step - loss: 0.0321 - accuracy: 0.9792 - v
al_loss: 0.3421 - val_accuracy: 0.8333

Epoch 00017: val_loss did not improve from 0.26699

Epoch 00017: val_accuracy did not improve from 0.91667
Epoch 18/20
6/6 [==============================] - 21s 4s/step - loss: 0.0123 - accuracy: 1.0000 - v
al_loss: 0.3389 - val_accuracy: 0.8333

Epoch 00018: val_loss did not improve from 0.26699

Epoch 00018: val_accuracy did not improve from 0.91667
Epoch 19/20
6/6 [==============================] - 20s 4s/step - loss: 0.0170 - accuracy: 1.0000 - v
al_loss: 0.3457 - val_accuracy: 0.8333

Epoch 00019: val_loss did not improve from 0.26699

Epoch 00019: val_accuracy did not improve from 0.91667
Epoch 20/20
6/6 [==============================] - 21s 4s/step - loss: 0.0257 - accuracy: 1.0000 - v
al_loss: 0.3424 - val_accuracy: 0.8333

Epoch 00020: val_loss did not improve from 0.26699

Epoch 00020: val_accuracy did not improve from 0.91667
```

In [117…

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 3))

# plot training and validation accuracy against epochs using matplotlib
```

```
ax1.plot(hist.history['accuracy'])
ax1.plot(hist.history['val_accuracy'])
ax1.set_title('model accuracy')
ax1.set_ylabel('accuracy')
ax1.set_xlabel('epoch')
ax1.legend(['train', 'validation'], loc='upper left')

# plot training and validation loss against epochs using matplotlib
ax2.plot(hist.history['loss'])
ax2.plot(hist.history['val_loss'])
ax2.set_title('model loss')
ax2.set_ylabel('loss')
ax2.set_xlabel('epoch')
ax2.legend(['train', 'validation'], loc='upper left')

plt.show()
```



Using Trained Model to do Predictions

- Training/Validation accuracies will generally be high because the model has seen these images before
- Test (out of sample) images will be more reflective of how the model will predict new images
- There are also ~2000 None images that were unused during training/vaidation that can further give insights into the model's performance

Validation Accuracy

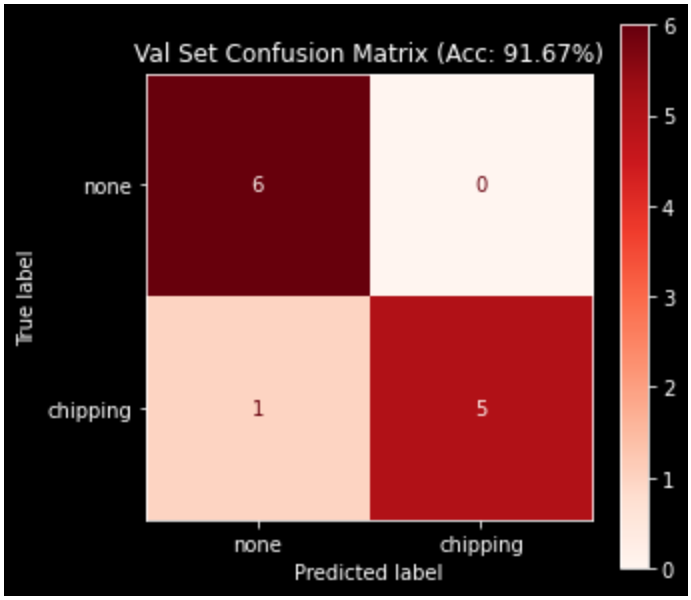
In [193…

```
model = load_model('models/model_loss.h5')
# model = load_model('models/model_acc.h5')
# model = cnn_model

valpred = model(x_test)
valpred = np.argmax(valpred, axis=1).tolist()
val_accuracy = (y_test_encoded == valpred).sum() / y_test_encoded.size
valpred
```

Out[193…

```
[0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1]
```

In [194…
```
cm = confusion_matrix(y_test_encoded, valpred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=DEFECT_LIST)

fig, ax = plt.subplots(figsize=(5,5))
ax.set_title(f'Val Set Confusion Matrix (Acc: {val_accuracy:.2%})')

disp.plot(cmap=plt.cm.Reds, ax=ax)
plt.show()
```

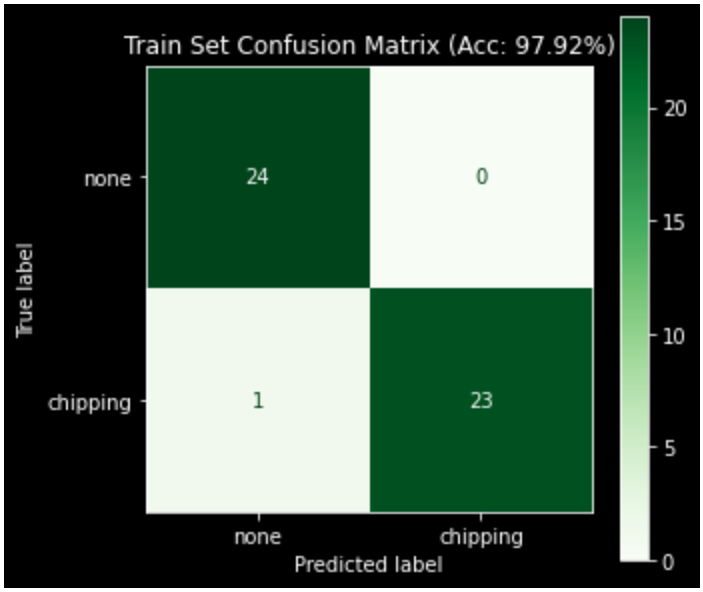


Training Accuracy

In [189…
```
trainpred = model.predict(x_train)
trainpred = np.argmax(trainpred, axis=1).tolist()
train_accuracy = (y_train_encoded == trainpred).sum() / y_train_encoded.size
train_accuracy
```

Out[189… 0.9791666666666666

In [191…
```
cm = confusion_matrix(y_train_encoded, trainpred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=DEFECT_LIST)

fig, ax = plt.subplots(figsize=(5,5))
ax.set_title(f'Train Set Confusion Matrix (Acc: {train_accuracy:.2%})')

disp.plot(cmap=plt.cm.Greens, ax=ax)
plt.show()
```

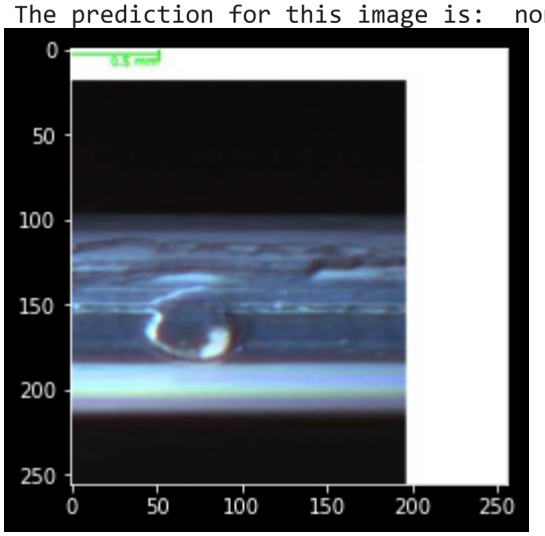


Single Image Accuracy

In [183…

```
# testimg = glob.glob(os.path.join(os.getcwd(), 'test', '*.png'))[8]
testimg = glob.glob(os.path.join(EN_path, 'none', '*.jpg'))[100] #69
# testimg = glob.glob(os.path.join(EN_path, 'none', '*.jpg'))[1000]
testimg = cv2.imread(testimg, cv2.IMREAD_COLOR)

# testimg = x_train[np.random.randint(0, x_train.shape[0])]

testimg = cv2.resize(testimg, (IMG_SIZE, IMG_SIZE))
testimg = testimg / 255.0

plt.imshow(testimg)
testimg = np.expand_dims(testimg, axis=0) # Expand dims so the input is (num_images, (I
testpred = model.predict(testimg)
testpred = np.argmax(testpred, axis=1)
print("The prediction for this image is: ", DEFECT_MAPPING.get(testpred[0]))
```

```
The prediction for this image is:  none
```



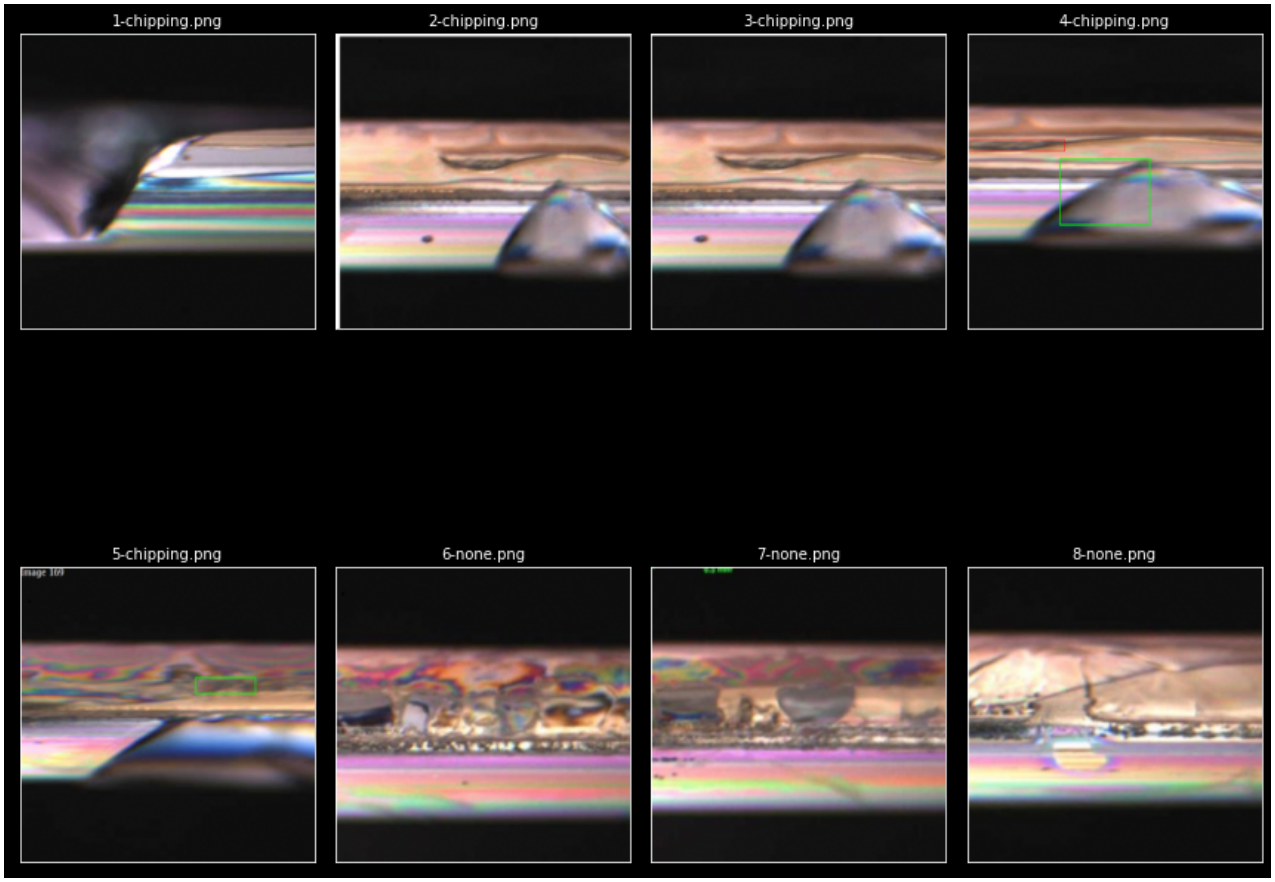
Out of Sample Test Accuracy

Testing on the 8 images extracted from the ppt to simulate new images coming in

In [71]:
```
test_imgs, test_labels = [], []

for test_img in glob.glob(os.path.join(EN_path, 'test', '*')):
    test_labels.append(test_img.split('-')[-1].split('.')[0])
    test_img = cv2.imread(test_img)
    test_img = cv2.resize(test_img, (IMG_SIZE, IMG_SIZE))
    test_img = test_img / 255.0
    test_imgs.append(test_img)

test_imgs = np.array(test_imgs)
test_labels = encode(test_labels)
test_labels
```

Out[71]:
```
array([1, 1, 1, 1, 1, 0, 0, 0])
```

In [206…
```
fig, ax = plt.subplots(nrows=2, ncols=4, figsize=(12, 12))
ax = ax.ravel(order='C')
all_test_imgs = glob.glob(os.path.join(EN_path, 'test', '*'))
for i in range(len(all_test_imgs)):
    img = cv2.imread(all_test_imgs[i], cv2.IMREAD_COLOR)
    img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
    img = img / 255.0
    ax[i].imshow(img)
    title = all_test_imgs[i].split('\\')[-1]
    ax[i].set_title(title, fontsize=10)
    ax[i].set_xticks([])
    ax[i].set_yticks([])

plt.tight_layout()
plt.show()
```



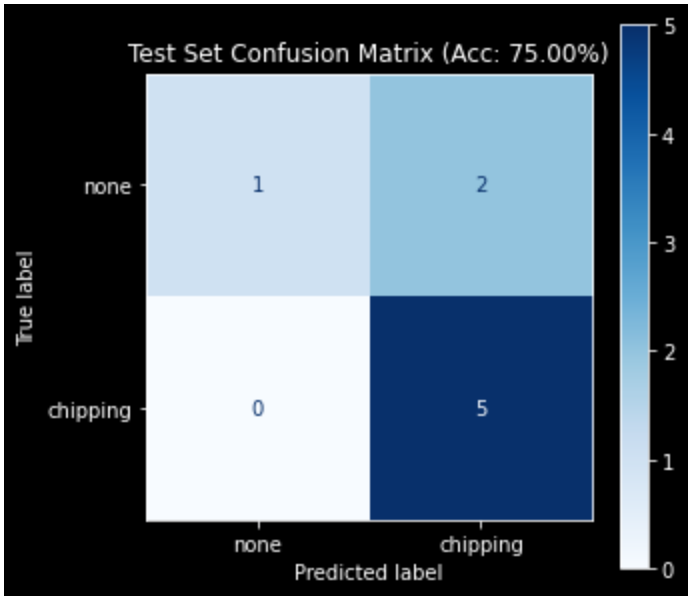
In [112…

```
testmodel = load_model('models/model_loss.h5')
test_preds = testmodel(test_imgs)
test_preds = np.argmax(test_preds, axis=1).tolist()
test_accuracy = (test_labels == test_preds).sum() / test_labels.size
test_preds
```

Out[112… [1, 1, 1, 1, 1, 1, 0, 1]

In [182…

```
cm = confusion_matrix(test_labels, test_preds)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=DEFECT_LIST)

fig, ax = plt.subplots(figsize=(5,5))
ax.set_title(f'Test Set Confusion Matrix (Acc: {test_accuracy:.2%})')

disp.plot(cmap=plt.cm.Blues, ax=ax)
plt.show()
```



None Images Accuracy

Testing on all the 2235 None images, so the predictions should be all 0s (all no chipping)

In [36]:
```
none_all = []

for none_img in glob.glob(os.path.join(EN_path, 'none', '*')):
    none_img = cv2.imread(none_img)
    none_img = cv2.resize(none_img, (IMG_SIZE, IMG_SIZE))
    none_img = none_img / 255.0
    none_all.append(none_img)

none_all = np.array(none_all)
none_all.shape
```

Out[36]: (2235, 256, 256, 3)

In [131…
```
start = time.time()
nonepreds = model.predict(none_all)
nonepreds = np.argmax(nonepreds, axis=1).tolist()
print(f'{1-sum(nonepreds)/(len(nonepreds)):.2%}')
duration(start)
```

```
98.97%

12.15 mins
```

In [ ]:
```
start = time.time()
saved_model = load_model('models/vgg16-test.h5')
nonepreds = saved_model.predict(none_all)
nonepreds = np.argmax(nonepreds, axis=1).tolist()
print(f'{1-sum(nonepreds)/(len(nonepreds)):.2%}')
duration(start)
```

Save Model

In [120…
```
model.save(os.path.join(os.getcwd(), 'models', 'vgg16-98.97.h5'))
```

Deep Dive into the Wrongly Predicted None Images

Some of the wrong images seem a bit arbitrary but it is expected of any machine learning model not to be 100% perfect

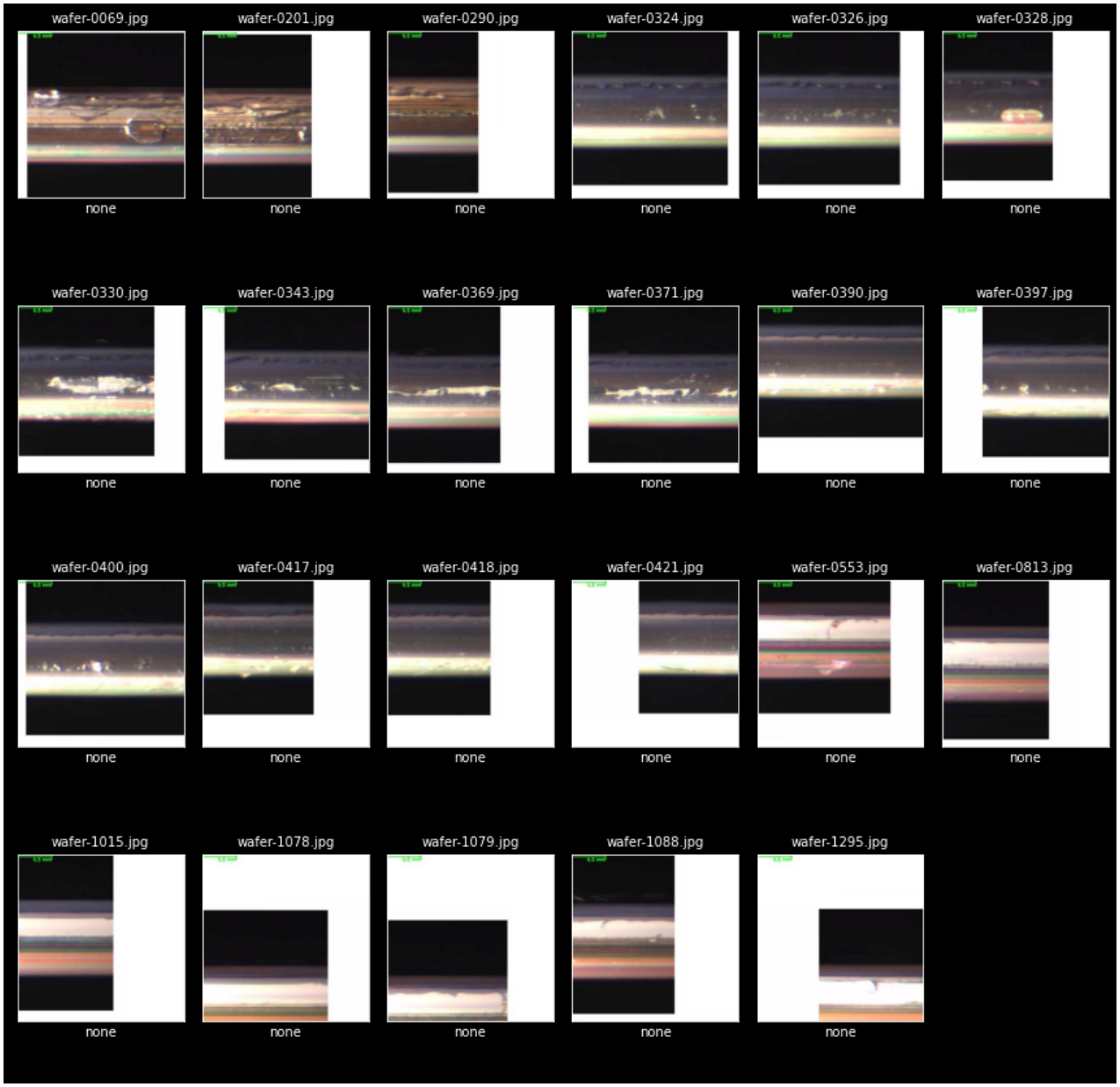
In [160…
```
incorrects = [i for i in range(len(nonepreds)) if nonepreds[i] == 1]
print(incorrects)
```

```
[69, 201, 290, 324, 326, 328, 330, 343, 369, 371, 390, 397, 400, 417, 418, 421, 553, 81
3, 1015, 1078, 1079, 1088, 1295]
```

In [136…
```
len(incorrects)
```

Out[136… 23

In [171…
```
fig, ax = plt.subplots(nrows=len(incorrects)//6+1, ncols=6, figsize=(12, 12))
ax = ax.ravel(order='C')
for a in ax: a.set_axis_off()
i = 0
for none_img in glob.glob(os.path.join(EN_path, 'none', '*')):
    if int(none_img.split('-')[-1].split('.')[0]) in incorrects:
        img = cv2.imread(none_img, cv2.IMREAD_COLOR)
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
        img = img / 255.0
        ax[i].imshow(img)
        ax[i].set_axis_on()
        ax[i].set_title(f'wafer-{none_img.split("-")[-1]}', fontsize=10)
        ax[i].set_xlabel('none', fontsize=10)
        ax[i].set_xticks([])
        ax[i].set_yticks([])
        i += 1

plt.tight_layout()
plt.show()
```


wafer-0069.jpg
none
wafer-0201.jpg
none
wafer-0290.jpg
none
wafer-0324.jpg
none
wafer-0326.jpg
none
wafer-0328.jpg
none
wafer-0330.jpg
none
wafer-0343.jpg
none
wafer-0369.jpg
none
wafer-0371.jpg
none
wafer-0390.jpg
none
wafer-0397.jpg
none
wafer-0400.jpg
none
wafer-0417.jpg
none
wafer-0418.jpg
none
wafer-0421.jpg
none
wafer-0553.jpg
none
wafer-0813.jpg
none
wafer-1015.jpg
none
wafer-1078.jpg
none
wafer-1079.jpg
none
wafer-1088.jpg
none
wafer-1295.jpg
none