Documentación del Proceso de Dataset para CNN - Análisis de Wafers Semiconductores

Resumen General

Este código implementa un pipeline completo de procesamiento de datos para análisis de defectos en wafers semiconductores utilizando redes neuronales convolucionales (CNN). El proceso incluye descarga de datos, preprocesamiento, data augmentation con autoencoders y preparación final para entrenamiento.

1. Descarga y Carga del Dataset

Descarga Condicional

```
file_path = "LSWMD.pkl"
file_id = "1JGJ_a7c5Cs4cAGh7NGElfGfhIm90-WUj"
url = f"https://drive.google.com/uc?id={file_id}"
```

Propósito: Descarga automática del dataset LSWMD (Large Scale Wafer Map Dataset) desde Google Drive si no existe localmente.

Características del Dataset:

- Formato: Archivo pickle (.pkl)
- Contenido: Mapas de wafers semiconductores con información de defectos
- Fuente: Dataset público para investigación en análisis de defectos semiconductores

Carga y Exploración Inicial

```
df = pd.read_pickle(file_path)
df.info()
df.head()
df.describe()
```

Funcionalidad:

- Carga el dataset en un DataFrame de pandas
- Realiza exploración inicial para entender la estructura de los datos
- Muestra estadísticas descriptivas básicas

2. Preprocesamiento de Datos

Eliminación de Columnas Innecesarias

```
df = df.drop(['waferIndex'], axis=1)
```

Justificación: El índice del wafer no aporta información relevante para la clasificación de defectos.

Análisis de Dimensiones

```
def find_dim(x):
  dim0 = np.size(x, axis=0)
  dim1 = np.size(x, axis=1)
  return dim0, dim1

df['waferMapDim'] = df.waferMap.apply(find_dim)
```

Propósito:

- Determina las dimensiones de cada mapa de wafer
- Crea una nueva columna con las dimensiones para facilitar el filtrado
- Permite identificar wafers con dimensiones consistentes

Filtrado por Dimensión Estándar

```
sub_df = df.loc[df['waferMapDim'] == (26, 26)]
```

Criterio de Selección:

- Solo se utilizan wafers de 26x26 píxeles
- Asegura uniformidad en el tamaño de entrada para la CNN
- Simplifica el procesamiento al tener dimensiones fijas

Extracción y Preparación de Datos

```
sub_wafer = sub_df['waferMap'].values
sw = np.ones((1, 26, 26))
label = list()

for i in range(len(sub_df)):
  if len(sub_df.iloc[i,:]['failureType']) == 0:
    continue
  sw = np.concatenate((sw, sub_df.iloc[i,:]['waferMap'].reshape(1, 26, 26)))
  label.append(sub_df.iloc[i,:]['failureType'][0][0])
```

Proceso:

1. Extrae los mapas de wafer de 26x26

2. Filtra muestras sin etiqueta de fallo
3. Concatena todos los mapas válidos
4. Extrae las etiquetas de tipo de fallo correspondientes

Formato Final de Datos

```
x = sw[1:]  # Elimina el primer elemento dummy
y = np.array(label).reshape((-1,1))
x = x.reshape((-1, 26, 26, 1))  # Añade dimensión de canal
```

Estructura Resultante:

- x: Array de mapas de wafer con forma (n_samples, 26, 26, 1)
- y: Array de etiquetas con forma (n_samples, 1)

3. Análisis de Tipos de Defectos

Identificación de Categorías

```
faulty_case = np.unique(y)
for f in faulty_case:
  print('{} : {}'.format(f, len(y[y==f])))
```

Tipos de Defectos Identificados:

- Múltiples categorías de defectos específicos
- Categoría "none" para wafers sin defectos
- Distribución desbalanceada entre clases

4. Codificación One-Hot de Características

Transformación de Mapas de Wafer

```
new_x = np.zeros((len(x), 26, 26, 3))
for w in range(len(x)):
  for i in range(26):
    for j in range(26):
      new_x[w, i, j, int(x[w, i, j])] = 1
```

Objetivo:

- Convierte valores categóricos en representación one-hot
- Crea 3 canales para diferentes estados del píxel
- Mejora la capacidad de la CNN para procesar información categórica

5. Data Augmentation con Autoencoder

Arquitectura del Autoencoder

```
# Encoder
input_shape = (26, 26, 3)
input_tensor = Input(input_shape)
encode = layers.Conv2D(64, (3,3), padding='same', activation='relu')(input_tensor)
latent_vector = layers.MaxPool2D()(encode)

# Decoder
decode_layer_1 = layers.Conv2DTranspose(64, (3,3), padding='same', activation='relu')
decode_layer_2 = layers.UpSampling2D()
output_tensor = layers.Conv2DTranspose(3, (3,3), padding='same', activation='sigmoid')
```

Componentes:

- **Encoder**: Comprime el input a una representación latente de 13x13x64
- **Decoder**: Reconstruye la imagen original desde la representación latente
- **Función de pérdida**: MSE (Mean Squared Error)

Entrenamiento del Autoencoder

```
ae.fit(new_x, new_x, batch_size=1024, epochs=15, verbose=2)
```

Parámetros de Entrenamiento:

- Épocas: 15
- Batch size: 1024
- Optimizador: Adam
- Objetivo: Reconstrucción perfecta de los mapas de wafer

Separación de Encoder y Decoder

```
encoder = models.Model(input_tensor, latent_vector)
decoder_input = Input((13, 13, 64))
# ... configuración del decoder independiente
decoder = models.Model(decoder_input, output_tensor(decode))
```

Utilidad:

- Permite usar el encoder y decoder por separado
- Facilita la generación controlada de nuevos datos
- Optimiza el proceso de augmentation

6. Generación de Datos Sintéticos

Función de Augmentation

```
def gen_data(wafer, label):
  encoded_x = encoder.predict(wafer)
  gen_x = np.zeros((1, 26, 26, 3))

  for i in range((2000//len(wafer)) + 1):
    noised_encoded_x = encoded_x + np.random.normal(loc=0, scale=0.1, size=(len(encoded_x), 13, 13, 64))
    noised_gen_x = decoder.predict(noised_encoded_x)
    gen_x = np.concatenate((gen_x, noised_gen_x), axis=0)

  gen_y = np.full((len(gen_x), 1), label)
  return gen_x[1:], gen_y[1:]
```

Proceso de Augmentation:

1. **Codificación**: Convierte wafers a representación latente
2. **Adición de Ruido**: Añade ruido gaussiano (μ=0, σ=0.1) al espacio latente
3. **Decodificación**: Genera nuevos wafers desde representaciones ruidosas
4. **Escalado**: Genera hasta 2000 muestras por clase minoritaria

Aplicación a Todas las Clases Defectuosas

```
for f in faulty_case:
  if f == 'none':
    continue
  gen_x, gen_y = gen_data(new_x[np.where(y==f)[0]], f)
  new_x = np.concatenate((new_x, gen_x), axis=0)
  y = np.concatenate((y, gen_y))
```

Estrategia:

- Augmenta solo clases con defectos (excluye "none")
- Balancea el dataset aumentando clases minoritarias
- Mantiene la diversidad mediante variaciones controladas

7. Balanceado del Dataset

Reducción de Clase Mayoritaria

```
none_idx = np.where(y=='none')[0][np.random.choice(len(np.where(y=='none')[0]), size=11000, replace=False)]
new_x = np.delete(new_x, none_idx, axis=0)
new_y = np.delete(y, none_idx, axis=0)
```

Objetivo:

- Reduce la clase "none" (sin defectos) a 11,000 muestras
- Evita el sesgo hacia la clase mayoritaria
- Mejora la capacidad de detección de defectos minoritarios

8. Preparación Final para CNN

Codificación Numérica de Etiquetas

```
for i, l in enumerate(faulty_case):
  new_y[new_y==l] = i
new_y = to_categorical(new_y)
```

Transformación:

- Convierte etiquetas string a índices numéricos
- Aplica codificación one-hot para clasificación multiclase
- Prepara formato compatible con funciones de pérdida categórica

División Train/Test

```
x_train, x_test, y_train, y_test = train_test_split(new_x, new_y, test_size=0.33, random_state=2019)
```

Configuración:

- 67% datos de entrenamiento
- 33% datos de prueba
- Semilla fija (2019) para reproducibilidad
- Estratificación implícita para mantener proporción de clases

9. Resultados y Métricas del Pipeline

Dimensiones Finales

- **Entrada original**: Mapas de wafer 26x26 píxeles
- **Formato procesado**: (n_samples, 26, 26, 3) con codificación one-hot
- **Dataset balanceado**: Múltiples clases con distribución equilibrada

Beneficios del Proceso

1. **Data Augmentation Inteligente**: Usa autoencoder para generar variaciones realistas
2. **Balanceado de Clases**: Evita sesgo hacia clases mayoritarias
3. **Formato Optimizado**: Estructura ideal para CNNs
4. **Reproducibilidad**: Semillas fijas y procesos determinísticos

10. Consideraciones Técnicas

Parámetros Críticos

- **Ruido Gaussiano**: σ=0.1 para augmentation controlada
- **Dimensión Latente**: 13x13x64 preserva información esencial
- **Batch Size**: 1024 para entrenamiento eficiente del autoencoder

Limitaciones y Mejoras Potenciales

- **Dimensión Fija**: Solo procesa wafers 26x26
- **Augmentation Simple**: Podría incorporar transformaciones geométricas
- **Balanceado Manual**: Podría usar técnicas más sofisticadas como SMOTE

Este pipeline proporciona una base sólida para el entrenamiento de CNNs en detección de defectos semiconductores, combinando técnicas modernas de deep learning con principios sólidos de preparación de datos.