

Training and Evaluating CNN Models

Lecture 03

Prepared by

Dr. Hager Saleh

Convolutional Neural Networks (CNNs)

A Convolutional Neural Network (CNN) is a type of deep learning model specifically designed for processing structured grid-like data, such as images. CNNs are widely used in computer vision tasks, including image classification, object detection, and segmentation.

How CNN Works

CNNs process images by learning hierarchical patterns using different layers:

- **Input Layer**

The input is an image represented as a matrix of pixel values. Example: A grayscale image is represented as a 2D matrix, while a color image (RGB) is represented as a 3D matrix.

- **Convolutional Layer**

- Applies filters (kernels) to detect features like edges, textures, and shapes.
- Each filter extracts a different feature.
- The result is called a feature map.
- Example: A 3x3 filter moves across the image and performs a dot product between pixel values and filter values.

- Pooling Layer (Downsampling)
 - Reduces the dimensionality while keeping important features.
 - Common pooling methods:
 - Max Pooling: Takes the maximum value in a region.
 - Average Pooling: Takes the average value in a region.

□ Example: A 2x2 max pooling reduces the size of the feature map while keeping key features.

- Fully Connected Layer
 - Flattens the feature maps into a 1D vector.
- Softmax / Sigmoid (Output Layer)
 - Softmax: Used for multi-class classification (assigns probabilities to classes).
 - Sigmoid: Used for binary classification (outputs 0 or 1).

Different types of CNN models

- AlexNet
- VGGNet (VGG-16, VGG-19)
- Inception
- ResNet
 - ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-152
- DenseNet (Densely Connected CNNs)
- EfficientNet

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
from tensorflow.keras.layers import Dense, Flatten, Dropout,
GlobalAveragePooling2D
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score, classification_report
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

```
def create_data_generators(train_dir, val_dir, test_dir,
img_size=(240, 240), batch_size=32):
    # Data augmentation for training dataset
    train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=360,
        horizontal_flip=True,
        vertical_flip=True,
```

```

        width_shift_range=0.1,
        height_shift_range=0.1,
        zoom_range=(0.75,1),
        brightness_range=(0.75,1.25)
    )

    # Only rescale for validation and testing
    val_test_datagen = ImageDataGenerator(rescale=1./255)

    # Generate augmented training images
    train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=img_size,
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=True
    )

    # Generate validation images
    val_generator = val_test_datagen.flow_from_directory(
        val_dir,
        target_size=img_size,
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=False
    )

    # Generate test images
    test_generator = val_test_datagen.flow_from_directory(
        test_dir,
        target_size=img_size,
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=False
    )

    return train_generator, val_generator, test_generator

```

CNN model

```

def build_cnn_model(input_shape, num_classes):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu',
            input_shape=input_shape),
        MaxPooling2D(2, 2),

```

```

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

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

        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model

# Directories for datasets
train_dir = 'dataset/train'
val_dir = 'dataset/validation'
test_dir = 'dataset/test'

# Create data generators
train_gen, val_gen, test_gen = create_data_generators(train_dir,
val_dir, test_dir)

# Get input shape and number of classes
input_shape = (240, 240, 3)
num_classes = len(train_gen.class_indices)

# Build CNN model
model = build_cnn_model(input_shape, num_classes)

# Train the model
history = model.fit(train_gen, epochs=10, validation_data=val_gen)

```

```

Found 766 images belonging to 3 classes.
Found 164 images belonging to 3 classes.
Found 222 images belonging to 3 classes.

```

```

C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\keras\
src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.

```

```

    super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

```

```

C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\keras\

```

```
src\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning:
Your `PyDataset` class should call `super().__init__(**kwargs)` in its
constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will
be ignored.
```

```
self._warn_if_super_not_called()
```

```
Epoch 1/10
```

```
24/24 _____ 28s 1s/step - accuracy: 0.4140 - loss:
1.5041 - val_accuracy: 0.5122 - val_loss: 0.9621
```

```
Epoch 2/10
```

```
24/24 _____ 16s 642ms/step - accuracy: 0.5068 - loss:
0.9649 - val_accuracy: 0.5122 - val_loss: 0.9539
```

```
Epoch 3/10
```

```
24/24 _____ 16s 658ms/step - accuracy: 0.4862 - loss:
0.9900 - val_accuracy: 0.5122 - val_loss: 0.9160
```

```
Epoch 4/10
```

```
24/24 _____ 13s 534ms/step - accuracy: 0.5391 - loss:
0.9341 - val_accuracy: 0.4573 - val_loss: 0.9168
```

```
Epoch 5/10
```

```
24/24 _____ 13s 528ms/step - accuracy: 0.4807 - loss:
0.9543 - val_accuracy: 0.5122 - val_loss: 0.9092
```

```
Epoch 6/10
```

```
24/24 _____ 13s 532ms/step - accuracy: 0.5058 - loss:
0.9612 - val_accuracy: 0.5122 - val_loss: 0.9012
```

```
Epoch 7/10
```

```
24/24 _____ 13s 555ms/step - accuracy: 0.5315 - loss:
0.9530 - val_accuracy: 0.4512 - val_loss: 0.9376
```

```
Epoch 8/10
```

```
24/24 _____ 12s 511ms/step - accuracy: 0.5056 - loss:
0.9308 - val_accuracy: 0.5061 - val_loss: 0.9189
```

```
Epoch 9/10
```

```
24/24 _____ 12s 496ms/step - accuracy: 0.5064 - loss:
0.9506 - val_accuracy: 0.5305 - val_loss: 0.8976
```

```
Epoch 10/10
```

```
24/24 _____ 12s 496ms/step - accuracy: 0.5296 - loss:
0.9107 - val_accuracy: 0.5671 - val_loss: 0.9241
```

```
def evaluate_model(model_name, y_true, y_pred):
```

```
    labels = list(test_gen.class_indices.keys())
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    cm = confusion_matrix(y_true, y_pred)
```

```
    print("Accuracy:", accuracy)
```

```

print("Precision :", precision)
print("Recall:", recall)
print("F1-score:" , f1)

print("Classification Report:")
print(classification_report(y_true, y_pred_classes,
target_names=labels))

# Plot confusion matrix
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=labels, yticklabels=labels)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()

```

```

y_pred=model.predict(test_gen)
y_true = test_gen.classes
y_pred_classes = np.argmax(y_pred, axis=1)
evaluate_model('CNN', y_true, y_pred_classes)

```

7/7 ————— 2s 316ms/step

Accuracy: 0.536036036036036

Precision : 0.6047046220456624

Recall: 0.536036036036036

F1-score: 0.4906631664089511

Classification Report:

	precision	recall	f1-score	support
Bengin cases	0.00	0.00	0.00	25
Malignant cases	0.86	0.37	0.52	113
Normal cases	0.45	0.92	0.60	84
accuracy			0.54	222
macro avg	0.43	0.43	0.37	222
weighted avg	0.60	0.54	0.49	222

C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\sklearn\metrics_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

```

_warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))

```

C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\

```
sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
```

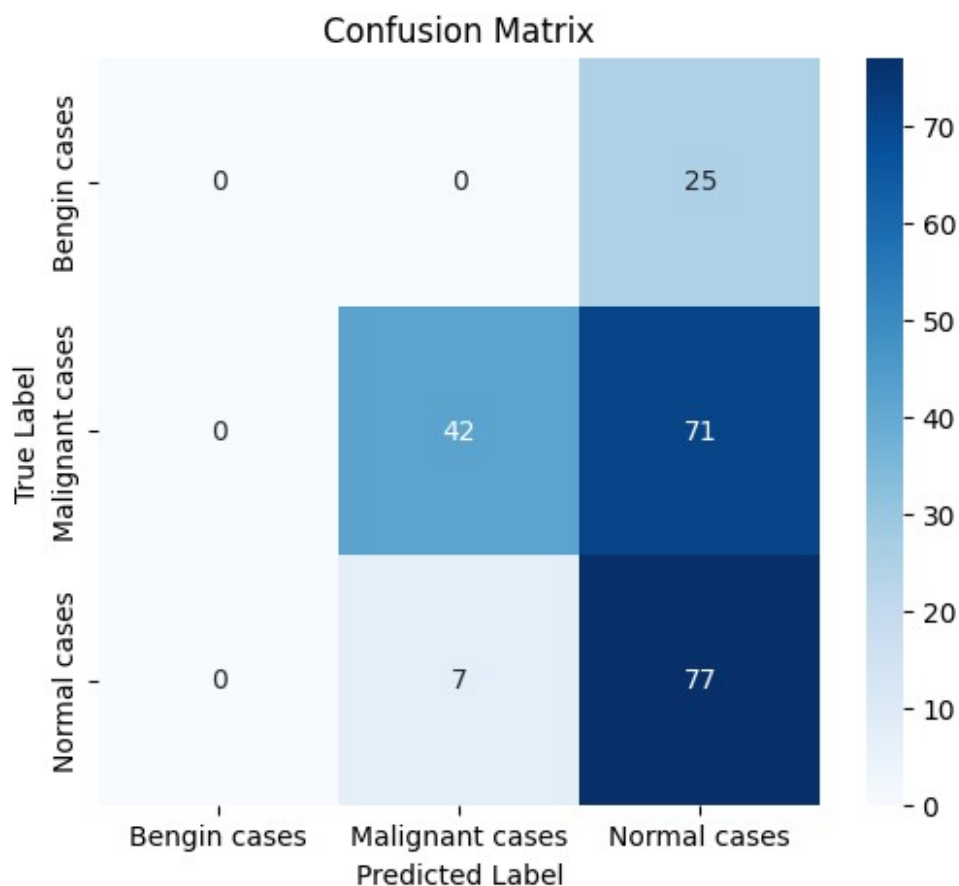
```
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
```

```
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
```

```
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```



ResNet50

```
def build_resnet8_model(input_shape, num_classes):
    base_model = ResNet50(weights='imagenet', include_top=False,
input_shape=input_shape)
    base_model.trainable = False # Freeze the base model

    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model

model = build_resnet8_model(input_shape, num_classes)

history = model.fit(train_gen, epochs=50, validation_data=val_gen)
```

Epoch 1/50
24/24 ————— 26s 929ms/step - accuracy: 0.4613 - loss: 1.1936 - val_accuracy: 0.5122 - val_loss: 0.9798

Epoch 2/50
24/24 ————— 21s 884ms/step - accuracy: 0.4838 - loss: 0.9886 - val_accuracy: 0.5122 - val_loss: 0.9511

Epoch 3/50
24/24 ————— 21s 859ms/step - accuracy: 0.5036 - loss: 0.9602 - val_accuracy: 0.5305 - val_loss: 0.9605

Epoch 4/50
24/24 ————— 21s 873ms/step - accuracy: 0.5248 - loss: 0.9517 - val_accuracy: 0.5122 - val_loss: 0.9522

Epoch 5/50
24/24 ————— 21s 864ms/step - accuracy: 0.5024 - loss: 0.9767 - val_accuracy: 0.5122 - val_loss: 0.9354

Epoch 6/50
24/24 ————— 21s 886ms/step - accuracy: 0.5386 - loss: 0.9284 - val_accuracy: 0.5122 - val_loss: 0.9343

Epoch 7/50
24/24 ————— 21s 857ms/step - accuracy: 0.5160 - loss: 0.9554 - val_accuracy: 0.5122 - val_loss: 0.9375

Epoch 8/50
24/24 ————— 21s 861ms/step - accuracy: 0.5423 - loss: 0.9455 - val_accuracy: 0.5122 - val_loss: 0.9347

Epoch 9/50

24/24 _____ 21s 873ms/step - accuracy: 0.4989 - loss: 0.9668 - val_accuracy: 0.5122 - val_loss: 0.9441
Epoch 10/50

24/24 _____ 21s 867ms/step - accuracy: 0.4789 - loss: 0.9613 - val_accuracy: 0.5122 - val_loss: 0.9339
Epoch 11/50

24/24 _____ 21s 856ms/step - accuracy: 0.5031 - loss: 0.9926 - val_accuracy: 0.5183 - val_loss: 0.9282
Epoch 12/50

24/24 _____ 22s 935ms/step - accuracy: 0.4774 - loss: 0.9681 - val_accuracy: 0.5122 - val_loss: 0.9241
Epoch 13/50

24/24 _____ 21s 873ms/step - accuracy: 0.5107 - loss: 0.9946 - val_accuracy: 0.5061 - val_loss: 0.9339
Epoch 14/50

24/24 _____ 21s 873ms/step - accuracy: 0.4979 - loss: 0.9671 - val_accuracy: 0.5122 - val_loss: 0.9269
Epoch 15/50

24/24 _____ 21s 866ms/step - accuracy: 0.5038 - loss: 0.9363 - val_accuracy: 0.5122 - val_loss: 0.9280
Epoch 16/50

24/24 _____ 21s 857ms/step - accuracy: 0.5094 - loss: 0.9412 - val_accuracy: 0.5122 - val_loss: 0.9351
Epoch 17/50

24/24 _____ 21s 875ms/step - accuracy: 0.5238 - loss: 0.9246 - val_accuracy: 0.5305 - val_loss: 0.9469
Epoch 18/50

24/24 _____ 23s 955ms/step - accuracy: 0.5304 - loss: 0.9450 - val_accuracy: 0.5671 - val_loss: 0.9296
Epoch 19/50

24/24 _____ 21s 869ms/step - accuracy: 0.4950 - loss: 0.9735 - val_accuracy: 0.5122 - val_loss: 0.9493
Epoch 20/50

24/24 _____ 22s 915ms/step - accuracy: 0.5104 - loss: 0.9445 - val_accuracy: 0.5122 - val_loss: 0.9543
Epoch 21/50

24/24 _____ 24s 993ms/step - accuracy: 0.5247 - loss: 0.9439 - val_accuracy: 0.5122 - val_loss: 0.9356
Epoch 22/50

24/24 _____ 22s 907ms/step - accuracy: 0.5186 - loss: 0.9668 - val_accuracy: 0.5122 - val_loss: 0.9251
Epoch 23/50

24/24 _____ 23s 976ms/step - accuracy: 0.5289 - loss: 0.9525 - val_accuracy: 0.5183 - val_loss: 0.9381
Epoch 24/50

24/24 _____ 23s 943ms/step - accuracy: 0.5236 - loss: 0.9368 - val_accuracy: 0.5122 - val_loss: 0.9466
Epoch 25/50

24/24 _____ 22s 935ms/step - accuracy: 0.5296 - loss:

0.9433 - val_accuracy: 0.5305 - val_loss: 0.9303
Epoch 26/50
24/24 _____ 22s 928ms/step - accuracy: 0.5399 - loss: 0.8934 - val_accuracy: 0.5122 - val_loss: 0.9315
Epoch 27/50
24/24 _____ 21s 879ms/step - accuracy: 0.4970 - loss: 0.9832 - val_accuracy: 0.5122 - val_loss: 0.9135
Epoch 28/50
24/24 _____ 21s 863ms/step - accuracy: 0.5238 - loss: 0.9271 - val_accuracy: 0.5122 - val_loss: 0.9295
Epoch 29/50
24/24 _____ 21s 872ms/step - accuracy: 0.5045 - loss: 0.9306 - val_accuracy: 0.5244 - val_loss: 0.9165
Epoch 30/50
24/24 _____ 21s 872ms/step - accuracy: 0.4603 - loss: 0.9704 - val_accuracy: 0.5122 - val_loss: 0.9185
Epoch 31/50
24/24 _____ 21s 856ms/step - accuracy: 0.5105 - loss: 0.9201 - val_accuracy: 0.5122 - val_loss: 0.9183
Epoch 32/50
24/24 _____ 21s 873ms/step - accuracy: 0.5294 - loss: 0.9249 - val_accuracy: 0.5061 - val_loss: 0.9352
Epoch 33/50
24/24 _____ 21s 871ms/step - accuracy: 0.5278 - loss: 0.9386 - val_accuracy: 0.5122 - val_loss: 0.9119
Epoch 34/50
24/24 _____ 21s 869ms/step - accuracy: 0.4978 - loss: 0.9705 - val_accuracy: 0.5122 - val_loss: 0.9090
Epoch 35/50
24/24 _____ 21s 882ms/step - accuracy: 0.5639 - loss: 0.9006 - val_accuracy: 0.5305 - val_loss: 0.9293
Epoch 36/50
24/24 _____ 21s 873ms/step - accuracy: 0.5528 - loss: 0.9126 - val_accuracy: 0.5244 - val_loss: 0.9340
Epoch 37/50
24/24 _____ 21s 873ms/step - accuracy: 0.5109 - loss: 0.9382 - val_accuracy: 0.5793 - val_loss: 0.9236
Epoch 38/50
24/24 _____ 21s 866ms/step - accuracy: 0.5409 - loss: 0.9234 - val_accuracy: 0.5610 - val_loss: 0.9193
Epoch 39/50
24/24 _____ 21s 866ms/step - accuracy: 0.5210 - loss: 0.9117 - val_accuracy: 0.5183 - val_loss: 0.9017
Epoch 40/50
24/24 _____ 21s 862ms/step - accuracy: 0.5091 - loss: 0.9512 - val_accuracy: 0.5671 - val_loss: 0.9171
Epoch 41/50
24/24 _____ 21s 877ms/step - accuracy: 0.5115 - loss: 0.9520 - val_accuracy: 0.5488 - val_loss: 0.9162

Epoch 42/50
 24/24 ————— 21s 867ms/step - accuracy: 0.5413 - loss: 0.9307 - val_accuracy: 0.5671 - val_loss: 0.8901
 Epoch 43/50
 24/24 ————— 21s 875ms/step - accuracy: 0.5193 - loss: 0.9109 - val_accuracy: 0.6159 - val_loss: 0.9090
 Epoch 44/50
 24/24 ————— 21s 893ms/step - accuracy: 0.5141 - loss: 0.9371 - val_accuracy: 0.5244 - val_loss: 0.9003
 Epoch 45/50
 24/24 ————— 21s 876ms/step - accuracy: 0.5434 - loss: 0.9246 - val_accuracy: 0.5366 - val_loss: 0.9251
 Epoch 46/50
 24/24 ————— 22s 898ms/step - accuracy: 0.5300 - loss: 0.9015 - val_accuracy: 0.5793 - val_loss: 0.9063
 Epoch 47/50
 24/24 ————— 44s 2s/step - accuracy: 0.5671 - loss: 0.8792 - val_accuracy: 0.4878 - val_loss: 0.9345
 Epoch 48/50
 24/24 ————— 21s 890ms/step - accuracy: 0.5346 - loss: 0.9049 - val_accuracy: 0.5427 - val_loss: 0.9221
 Epoch 49/50
 24/24 ————— 23s 962ms/step - accuracy: 0.5007 - loss: 0.9308 - val_accuracy: 0.4756 - val_loss: 0.9265
 Epoch 50/50
 24/24 ————— 23s 946ms/step - accuracy: 0.5442 - loss: 0.9096 - val_accuracy: 0.5000 - val_loss: 0.9221

```
y_pred=model.predict(test_gen)
y_true = test_gen.classes
y_pred_classes = np.argmax(y_pred, axis=1)
evaluate_model('ResNet50', y_true, y_pred_classes)
```

7/7 ————— 7s 888ms/step

Accuracy: 0.5135135135135135
 Precision : 0.5231381500171057
 Recall: 0.5135135135135135
 F1-score: 0.47458762867083354

Classification Report:

	precision	recall	f1-score	support
Bengin cases	0.00	0.00	0.00	25
Malignant cases	0.70	0.40	0.51	113
Normal cases	0.44	0.82	0.57	84
accuracy			0.51	222
macro avg	0.38	0.41	0.36	222
weighted avg	0.52	0.51	0.47	222

```
C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\
sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this
behavior.
```

```
    _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

```
C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\
sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this
behavior.
```

```
    _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

```
C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\
sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this
behavior.
```

```
    _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

```
C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\
sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this
behavior.
```

```
    _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

