PROYECTO 3.2

UPIIT-IPN

27 de junio 2024

Visión Artificial

Italy Abril Zayas Riojas

Homero Meneses Vázquez

Detección y clasificación del Equipo de Protección Individual (EPI).

Introducción

En este proyecto, se desarrolló un sistema de reconocimiento de objetos para identificar y clasificar razas de perros en imágenes. El sistema se basa en una red neuronal convolucional (CNN). La base de datos es sobre el equipo de protección personal de médicos durante la pandemia. Se emplearon técnicas de preprocesamiento de imágenes y aumento de datos para mejorar la precisión y el rendimiento del modelo. El sistema se evaluó utilizando métricas estándar como la precisión y la tasa de falsos positivos. Los resultados obtenidos demuestran que el sistema propuesto es capaz de detectar y clasificar aunque se vuelve más impreciso con imágenes con condiciones de iluminación y fondo desafiantes. Este sistema tiene el potencial de ser utilizado en diversas aplicaciones, como la inspección automatizada.

Planteamiento del problema.

El brote de COVID-19 ha generado una necesidad crítica de proteger al personal médico que se encuentra en la primera línea de batalla. El Equipo de Protección Individual (EPI) juega un papel crucial en la prevención del contagio. Para garantizar la seguridad del personal y los pacientes, la detección precisa y automatizada del uso del EPI en entornos hospitalarios es una tarea crucial.

La visión artificial se ha convertido en una herramienta poderosa para diversas aplicaciones de detección y clasificación de objetos. En este proyecto, se aplicará la visión artificial para desarrollar un sistema automatizado para detectar el uso de EPI (mascarilla, protector facial, cobertura completa, guantes y gafas) por parte del personal médico antes de ingresar a un área de pacientes COVID-19.

Objetivo:

Desarrollar un programa en Python que realice el reconocimiento de objetos utilizando un conjunto de datos asignado. El programa debe cubrir todas las fases de un flujo de trabajo de visión artificial, incluyendo obtención y depuración de datos, preprocesamiento, segmentación, extracción de características, entrenamiento de modelos de reconocimiento, evaluación y mejora de resultados.

Marco teórico:

Redes Neuronales Convolucionales (CNNs):

Las redes neuronales convolucionales (CNNs) son un tipo de red neuronal artificial (RNA) ampliamente utilizado en tareas de visión artificial, especialmente en el campo de la detección y clasificación de objetos. Las CNNs se caracterizan por su capacidad para extraer características espaciales de las imágenes a través de capas convolucionales y de agrupación.

Las capas convolucionales aplican filtros convolucionales a la imagen de entrada, generando mapas de características que resaltan patrones y detalles relevantes. Las capas de agrupación reducen la dimensionalidad de los mapas de características, controlando la complejidad del modelo y evitando el sobreajuste.

Las CNNs han demostrado un gran éxito en tareas como el reconocimiento facial, la clasificación de imágenes y la detección de objetos en imágenes y videos. Su capacidad para aprender de grandes conjuntos de datos y extraer características de forma automática las convierte en una herramienta poderosa para la visión artificial.

Aprendizaje Transferido:

El aprendizaje transferido es una técnica en el campo del aprendizaje profundo que consiste en reutilizar un modelo pre-entrenado en una tarea para resolver un problema diferente. El modelo pre-entrenado se utiliza como punto de partida, y se ajusta posteriormente a la nueva tarea específica.

En el contexto de la detección de EPI, se puede utilizar una CNN pre-entrenada, como VGG16, en un conjunto de datos masivo de imágenes, para aprovechar su capacidad de extracción de características. Posteriormente, el modelo se ajusta a un conjunto de datos de imágenes de EPI para aprender a identificar la presencia o ausencia de los diferentes elementos del equipo de protección.

El aprendizaje transferido permite aprovechar el conocimiento adquirido en tareas relacionadas, reduciendo el tiempo y los recursos necesarios para entrenar un modelo desde cero. Además, puede mejorar el rendimiento del modelo, especialmente cuando la cantidad de datos disponibles para la tarea específica es limitada.

Evaluación del Modelo

La evaluación del modelo es un paso crucial en el desarrollo de cualquier sistema de aprendizaje automático. Su objetivo es determinar el rendimiento del modelo en la tarea para la que fue entrenado, identificando sus fortalezas y debilidades.

Librerías y Técnicas Seleccionadas

- **TensorFlow y Keras:** Keras es una API de alto nivel construida sobre TensorFlow, que permite la creación de modelos de aprendizaje profundo de forma rápida y eficiente. En este proyecto, se utilizará la arquitectura pre-entrenada VGG16 de TensorFlow para la clasificación de imágenes. VGG16 es una CNN pre-entrenada en un conjunto de datos masivo de imágenes, lo que permite aprovechar su capacidad de extracción de características para la tarea específica de detección de EPI.
- **Procesamiento de Imágenes y Aumento de Datos:** El preprocesamiento de imágenes es fundamental para optimizar el rendimiento del modelo. Se aplicarán técnicas como el cambio de tamaño y la normalización para garantizar la consistencia en las entradas del modelo. Además, para mitigar el desbalance de clases en el dataset (posiblemente existan más imágenes con EPI completo que sin elementos), se utilizará la técnica SMOTE (Synthetic Minority Over-sampling Technique) de la librería imbalanced-learn. SMOTE permite generar muestras

sintéticas de las clases minoritarias, equilibrando la distribución de clases y mejorando el rendimiento del modelo en las clases menos representadas.

- **Evaluación del Modelo:** Para evaluar el desempeño del modelo se utilizarán métricas estándar como la precisión (accuracy_score) y el informe de clasificación (classification_report). La precisión mide el porcentaje de predicciones correctas realizadas por el modelo. El informe de clasificación proporciona una visión detallada del rendimiento del modelo para cada clase de EPI, incluyendo la precisión, el recall y la F1-score.

Primer Bloque: Conversión a Escala de Grises y Segmentación

Listado de Archivos de Imagen

```
image_files = [f for f in os.listdir(folder_path) if f.endswith(('.png', '.jpg', '.jpeg'))]
```

- `os.listdir(folder_path)`: Lista todos los archivos en la carpeta.
- if f.endswith(('.png', '.jpg', '.jpeg'))`: Filtra los archivos para que solo incluya imágenes con las extensiones especificadas.

Función para Convertir a Escala de Grises

```
def convert_to_grayscale(image):
    return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Función para Segmentar la Imagen

```
def segment_image(image):
    gray_image = convert_to_grayscale(image)
    _, thresholded = cv2.threshold(gray_image, 127, 255, cv2.THRESH_BINARY)
    return thresholded
```

- `convert_to_grayscale(image)`: Convierte la imagen a escala de grises.
- `cv2.threshold(gray_image, 127, 255, cv2.THRESH_BINARY)`: Aplica un umbral para convertir la imagen a blanco y negro.

Procesar y Mostrar Imágenes

- `os.path.join(folder_path, image_file)`: Construye la ruta completa a la imagen.
- `cv2.imread(image_path)`: Carga la imagen.
- `if image is None`: Verifica si la imagen se cargó correctamente.
- `segment_image(image)`: Segmenta la imagen.
- `plt.subplot()`, `plt.imshow()`, `plt.title()`, `plt.show()`: Muestra la imagen original y la segmentada usando Matplotlib.

Segundo Bloque: Extracción de Características con VGG16

Listado de Archivos de Imagen

```
image_files = [f for f in os.listdir(folder_path) if f.endswith(('.png', '.jpg', '.jpeg'))]
```

Cargar Modelo VGG16 Preentrenado

```
base_model = VGG16(weights='imagenet', include_top=False)
model = Model(inputs=base_model.input, outputs=base_model.get_layer('block5_pool').output)
```

- `VGG16(weights='imagenet', include_top=False)`: Carga el modelo VGG16 preentrenado sin la última capa de clasificación.
- `Model(inputs=base_model.input, outputs=base_model.get_layer('block5_pool').output)`: Crea un nuevo modelo que produce las salidas del bloque `block5_pool` de VGG16.

Función para Preparar la Imagen para la CNN

```
def prepare_image(img_path):
    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
```

```
    img_array = np.expand_dims(img_array, axis=0)
    return preprocess_input(img_array)
```

- `image.load_img(img_path, target_size=(224, 224))`: Carga y redimensiona la imagen.
- `image.img_to_array(img)`: Convierte la imagen en un arreglo.
- `np.expand_dims(img_array, axis=0)`: Añade una dimensión para representar el lote de imágenes.
- `preprocess_input(img_array)`: Preprocesa la imagen para el modelo VGG16.

Procesar y Extraer Características de las Imágenes

```
features = []

for image_file in image_files:
    image_path = os.path.join(folder_path, image_file)
    img = prepare_image(image_path)
    features.append(model.predict(img).flatten())
```

- `prepare_image(image_path)`: Prepara la imagen.
- `model.predict(img)`: Predice (extrae características) usando el modelo VGG16.
- `flatten()`: Aplana las características para convertirlas en un vector de una dimensión.

Convertir la Lista de Características en una Matriz

```
feature_matrix = np.array(features)
print("Matriz de características:", feature_matrix.shape)
```

- `np.array(features)`: Convierte la lista de características en una matriz de NumPy.
- `print("Matriz de características:", feature_matrix.shape)`: Muestra las dimensiones de la matriz de características.

Mostrar las Imágenes y sus Características Extraídas

- `cv2.imread(image_path)`: Carga la imagen.
- `plt.subplot()`, `plt.imshow()`, `plt.title()`, `plt.show()`: Muestra la imagen original y un gráfico de barras de las características extraídas.

Tercer Bloque: Clasificación de Imágenes con VGG16 y SMOTE

Cargar Datos de Entrenamiento y Prueba

```
train_path = '/content/train'
test_path = '/content/test'

train_df = pd.read_csv('/content/train_labels.csv')
train_df = train_df[train_df['filename'] != 'Using Personal Protection Equipment (PPE) 072.jpg']

test_df = pd.read_csv('/content/test_labels.csv')
```

- `pd.read_csv()`: Carga los archivos CSV con las etiquetas de entrenamiento y prueba.

Función para Cargar y Preprocesar Imágenes

```
def load_image(row):
    img = image.load_img(os.path.join(train_path, row['filename']), target_size=(224, 224))
    img_array = image.img_to_array(img)
    return img_array
```

- `image.load_img()`, `image.img_to_array()`: Carga y convierte la imagen en un arreglo.

Obtener Imágenes y Etiquetas

```
X = []
y = []

for _, row in train_df.iterrows():
    try:
        img_array = load_image(row)
        X.append(img_array)
        y.append(row['class'])
    except FileNotFoundError:
        print(f"Error: File not found - {os.path.join(train_path, row['filename'])}")

X = np.array(X)
y = np.array(y)
```

- `train_df.iterrows()`: Itera sobre las filas del DataFrame de entrenamiento.
- `load_image(row)`: Carga y preprocesa la imagen.
- `X.append(img_array)`, `y.append(row['class'])`: Añade la imagen y su etiqueta a las listas.
- `np.array()`: Convierte las listas en arreglos de NumPy.

Análisis Exploratorio de Datos

El código a continuación realiza un análisis exploratorio de datos (EDA):

Exploración de datos y visualización

Este código realiza un análisis exploratorio de datos (EDA) completo en dos conjuntos de datos, probablemente representando datos de entrenamiento y prueba para una tarea de clasificación de imágenes. El objetivo es comprender la estructura, características y distribución de los datos para prepararlos mejor para el entrenamiento del modelo de clasificación.

1. Forma de los datos y valores faltantes:

- Se determina la forma (número de filas y columnas) de los conjuntos de datos de entrenamiento y prueba (`train_df` y `test_df`).
- Se cuenta el número total de valores faltantes (valores nulos) en cada conjunto de datos.
- Se identifican el número total de columnas con valores faltantes y una lista de los nombres de esas columnas para cada conjunto de datos.

2. Creación de un resumen detallado:

- Se crea un nuevo marco de datos (`detailed_db`) para almacenar información detallada sobre ambos conjuntos de datos. Esto incluye:
    - Nombre del conjunto de datos ("Train" o "Test")
    - Número de filas
    - Número de columnas
    - Número total de valores faltantes
    - Lista de columnas con valores faltantes
    - Número de columnas con valores faltantes

3. Impresión de información del conjunto de datos:

- Se imprime información sobre la forma del conjunto de datos, el número de valores faltantes y los tipos de datos de cada columna.
- Se imprime un separador de línea doble para mayor claridad.

4. Exploración de características categóricas:

- Se imprime el número de valores únicos para cada columna en los conjuntos de datos. Esto ayuda a comprender la distribución de las características categóricas.

5. Resumen de características categóricas con gráficos:

- Se crea un resumen y un gráfico para las características categóricas (posiblemente la clase objetivo). Esto podría implicar visualizar la distribución de las etiquetas de clase.

6. Visualización de valores faltantes:

- Se crea una visualización (probablemente un mapa de calor) para mostrar la distribución de los valores faltantes en todas las características de los conjuntos de datos.

7. Conteo de imágenes y objetos:

- Se imprime el número de nombres de archivo únicos en cada conjunto de datos, indicando el número de imágenes únicas.
- Se imprime el número total de filas (objetos) en cada conjunto de datos.

8. Exploración de la resolución de la imagen:

- Se traza la distribución del ancho y alto de la imagen para ambos conjuntos de datos.

9. Relación entre clase y tamaño de imagen:

- Se utiliza `seaborn.relplot` para crear gráficos relacionales entre el ancho, la altura de la imagen y la clase objetivo. Esto ayuda a visualizar si existe una correlación entre el tamaño de la imagen y las etiquetas de clase.

10. Distribución del tamaño de la imagen por clase:

- Se utiliza `seaborn.jointplot` para crear gráficos de dispersión individuales para cada clase, mostrando la distribución del ancho y la altura dentro de cada clase.

11. Distribución del área de la imagen por clase:

- Se utiliza `seaborn.violinplot` para visualizar la distribución del área de la imagen (calculada como ancho * alto) en diferentes clases.

12. Cálculo del tamaño promedio de la imagen:

- Se calcula y potencialmente se imprime el ancho y alto promedio de la imagen para ambos conjuntos de datos.

Dibujar Rectángulos en la Imagen

```
data_df = train_df.copy()

data_df['file'] = train_path + '/' + data_df['filename']
```

```
data_df.head()


ldf = data_df[data_df['file'] == data_df['file'][0]]


f = ldf.iloc[0].file
img = Image.open(f)
draw = ImageDraw.Draw(img)
xres, yres = img.size[0], img.size[1]


for i in range(len(ldf)):
    draw.rectangle([int(ldf.iloc[i]['xmin']),
                    int(ldf.iloc[i]['ymin']),
                    int(ldf.iloc[i]['xmax']),
                    int(ldf.iloc[i]['ymax'])], outline=(255, 0, 0), width=3)
plt.imshow(np.array(img))
```

- `data_df = train_df.copy()`: Hace una copia del DataFrame de entrenamiento.
- `ldf = data_df[data_df['file'] == data_df['file'][0]]`: Filtra las filas que corresponden a la primera imagen.
- `Image.open(f)`: Abre la imagen.
- `ImageDraw.Draw(img)`: Crea un objeto para dibujar sobre la imagen.
- `draw.rectangle()`: Dibuja un rectángulo en la imagen.

Modelo de Clasificación

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
for layer in base_model.layers:
    layer.trainable = False


model = Sequential([
    base_model,
```

```
    Flatten(),
    Dense(1024, activation='relu'),
    Dense(1, activation='sigmoid')
])

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

- `VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))`: Carga el modelo VGG16 preentrenado.
- `for layer in base_model.layers: layer.trainable = False`: Congela las capas del modelo base.
- `Sequential()`: Define un modelo secuencial con capas adicionales.
- `model.compile()`: Compila el modelo con el optimizador, la función de pérdida y las métricas.

Preparar Datos para Entrenamiento

```
X = []
y = []

for _, row in train_df.iterrows():
    try:
        img_array = load_image(row)
        X.append(img_array)
        y.append(row['class'])
    except FileNotFoundError:
        print(f"Error: File not found - {os.path.join(train_path, row['filename'])}")

X = np.array(X)
y = np.array(y)

from sklearn.preprocessing import LabelEncoder
```

```
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
y =

 np.array(y_encoded).astype(np.float32)

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, stratify=y)

smote = SMOTE()
X_train_sm, y_train_sm = smote.fit_resample(X_train.reshape(X_train.shape[0], -1), y_train)
X_train_sm = X_train_sm.reshape(X_train_sm.shape[0], 224, 224, 3)
```

- `X, y`: Listas para almacenar imágenes y etiquetas.
- `load_image(row)`: Función para cargar y preprocesar imágenes.
- `LabelEncoder()`: Codifica etiquetas en valores numéricos.
- `train_test_split()`: Divide datos en conjuntos de entrenamiento y validación.
- `SMOTE()`: Realiza sobremuestreo en el conjunto de entrenamiento.

Entrenar y Evaluar el Modelo

```
history = model.fit(X_train_sm, y_train_sm, validation_data=(X_val, y_val), epochs=10, batch_size=32)

y_pred = model.predict(X_val)
y_pred_classes = np.where(y_pred > 0.5, 1, 0)

print(classification_report(y_val, y_pred_classes))
print("Accuracy:", accuracy_score(y_val, y_pred_classes))
```

- `model.fit()`: Entrena el modelo con los datos de entrenamiento.
- `model.predict()`: Realiza predicciones en el conjunto de validación.

- `classification_report()`: Muestra un reporte de clasificación con precisión, recall y F1-score.
- `accuracy_score()`: Calcula la precisión del modelo.

Resultados

Notebook 1: Visualización de Imágenes y Características

En la primera notebook, se imprimieron las imágenes originales junto a las segmentadas. Las coordenadas de los ejes representan los píxeles de la imagen, y se muestran en intervalos de 200 píxeles. Las características extraídas se presentan en un gráfico que compara las imágenes originales con las características obtenidas, mostrando una escala mucho mayor en el eje x, lo cual indica la cantidad de datos generados por la extracción de características utilizando la red CNN.

Interpretación de los ejes:

Eje x de las imágenes originales: valores de 0 a 1000 o 1200, representando la resolución de la imagen en píxeles.
Eje y de las imágenes originales: valores de 0 a 600, representando la resolución de la imagen en píxeles.
Eje x de las características extraídas: valores de 0 a 25000, representando la cantidad de datos generados por la red CNN.
Eje y de las características extraídas: la escala es más amplia, indicando la mayor cantidad de datos y características capturadas.
Las características extraídas representan los atributos detectados por la CNN, los cuales son utilizados en el proceso de reconocimiento.

Notebook 2: Análisis Estadístico y Visualización

En la segunda notebook, se generaron gráficos que muestran la distribución de las etiquetas y los valores faltantes en los datos.

Gráfico Countplot:

Eje x: categorías de objetos (gloves, coverall, mask, goggles, face_shield).
Eje y: número de ocurrencias de cada categoría.
Gráfico de Pastel:

Porcentaje de ocurrencias de cada categoría.
Gráfico de Valores Faltantes:

Muestra la distribución de valores faltantes en el conjunto de datos, con tonos de gris representando la cantidad de datos presentes en cada columna.
Gráfico de Barras:

Eje x: valores discretos (0 a 12).
Eje y: frecuencia de cada valor.
Estos gráficos proporcionan una visión clara de la distribución de las etiquetas y los valores faltantes, así como la frecuencia de los valores en los datos, lo cual es crucial para entender el estado del conjunto de datos antes del entrenamiento de los modelos.

Conclusión

Matriz de Confusión:

La matriz de confusión proporciona una representación visual de la distribución de las predicciones del modelo. Cada celda de la matriz representa el número de ejemplos de una clase real que fueron clasificados como otra clase por el modelo.

Análisis de Errores:

Es importante analizar los errores cometidos por el modelo para identificar posibles sesgos o limitaciones. Esto puede hacerse mediante la inspección manual de ejemplos mal clasificados o utilizando técnicas de análisis de errores más sofisticadas.

La evaluación del modelo es un proceso continuo que debe realizarse a lo largo del desarrollo del sistema. Los resultados de la evaluación permiten identificar áreas de mejora y realizar ajustes en el modelo, las técnicas de entrenamiento o el preprocesamiento de datos para optimizar su rendimiento.

La evaluación del modelo es un componente esencial para garantizar la confiabilidad y la robustez de un sistema de detección de EPI. La utilización de métricas adecuadas, curvas de validación, matrices de confusión y análisis de errores permite evaluar el desempeño del modelo de manera integral y tomar decisiones informadas para su mejora continua.

El proyecto implementado cubrió todas las fases del flujo de trabajo de visión artificial. A través de la obtención, depuración, preprocesamiento, segmentación, extracción de características, entrenamiento y evaluación, se lograron resultados significativos en el reconocimiento de objetos. Las mejoras propuestas apuntan a optimizar aún más el rendimiento del sistema, proporcionando una base sólida para futuras investigaciones y desarrollos en el campo de la visión artificial.