Carrera de Especialización en Inteligencia Artificial - FIUBA

Visión por Computadora 2

Trabajo Integrador: Modelo para la clasificación de residuos para reciclaje basándose en el dataset TrashNet.

Integrantes:

- Juan Ruiz Otondo
- Gonzalo Fernández
- Maria Fabiana Cid

Motivación y descripción del problema

Motivación:
En el contexto de la creciente generación de residuos sólidos, es fundamental implementar soluciones inteligentes que permitan automatizar su clasificación.

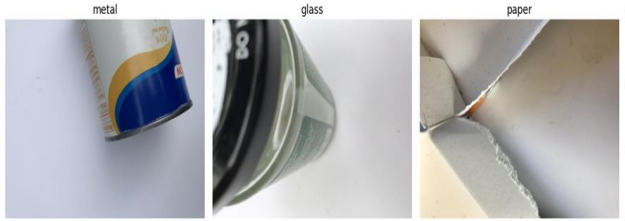
Objetivo academico:
Realizar un modelo para la clasificación de residuos para reciclaje basándose en el dataset TrashNet.

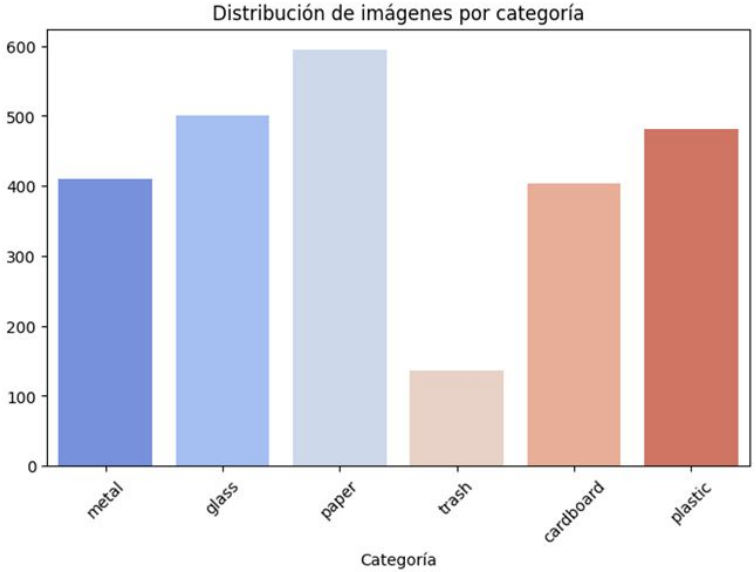
Objetivo práctico:
Es contribuir a una gestión sostenible de residuos mediante el uso de tecnologías de visión por computadora e inteligencia artificial.

Descripción del problema

- Conjunto de datos TrashNet

metal	410
glass	501
paper	594
trash	137
cardboard	403
plastic	482






```
Total number of images: 2527
```

Uso de Data Augmentation

Aplicación de Data Augmentation

Objetivos:
1. Incrementar el tamaño de conjunto de datos
2. Evitar el sobreentrenamiento
3. Permite balancear las clases del conjunto de datos
4. Mejorar las métricas del modelo

El data augmentation se planteará para ser aplicado **online**, es decir, no se generan las nuevas imágenes para guardar como un nuevo dataset.

Original Image

Augmented Image

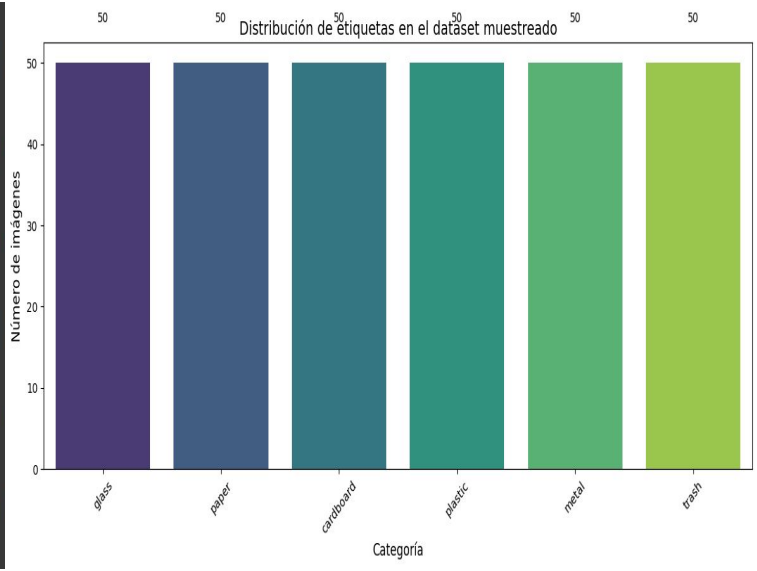
Original Image

Augmented Image

Análisis exploratorio de datos (EDA)

Análisis de distribución de etiquetas

```
Procesando 50 imágenes de la categoría 'glass'...
100% 50/50 [00:02<00:00, 21.32it/s]
Procesando 50 imágenes de la categoría 'paper'...
100% 50/50 [00:02<00:00, 23.66it/s]
Procesando 50 imágenes de la categoría 'cardboard'...
100% 50/50 [00:02<00:00, 24.41it/s]
Procesando 50 imágenes de la categoría 'plastic'...
100% 50/50 [00:02<00:00, 24.05it/s]
Procesando 50 imágenes de la categoría 'metal'...
100% 50/50 [00:02<00:00, 16.15it/s]
Procesando 50 imágenes de la categoría 'trash'...
100% 50/50 [00:02<00:00, 24.54it/s]
Se han cargado 300 imágenes para el análisis.
```

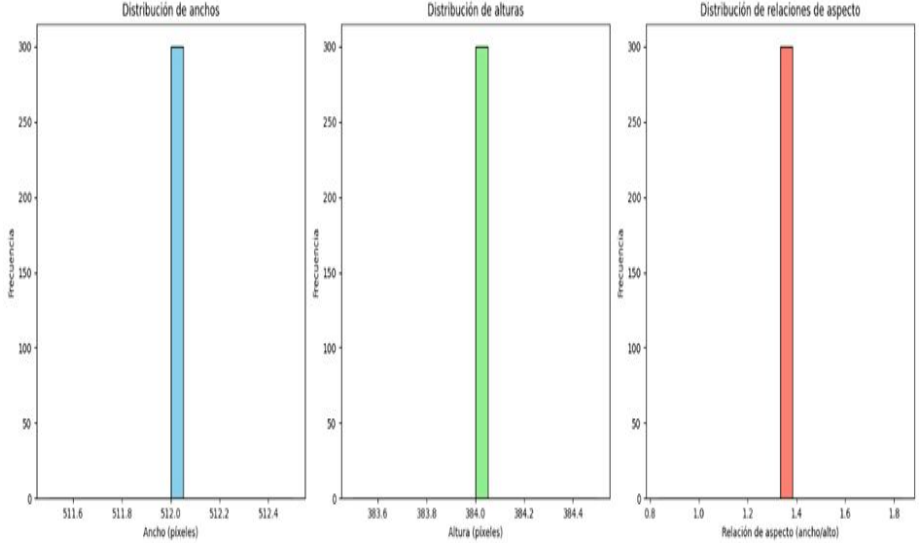


Análisis exploratorio de datos (EDA)

Análisis de dimensiones de imágenes y relación de aspecto

Estadísticas de dimensiones:

	width	height	aspect_ratio
count	300.0	300.0	3.000000e+02
mean	512.0	384.0	1.333333e+00
std	0.0	0.0	4.448312e-15
min	512.0	384.0	1.333333e+00
25%	512.0	384.0	1.333333e+00
50%	512.0	384.0	1.333333e+00
75%	512.0	384.0	1.333333e+00
max	512.0	384.0	1.333333e+00



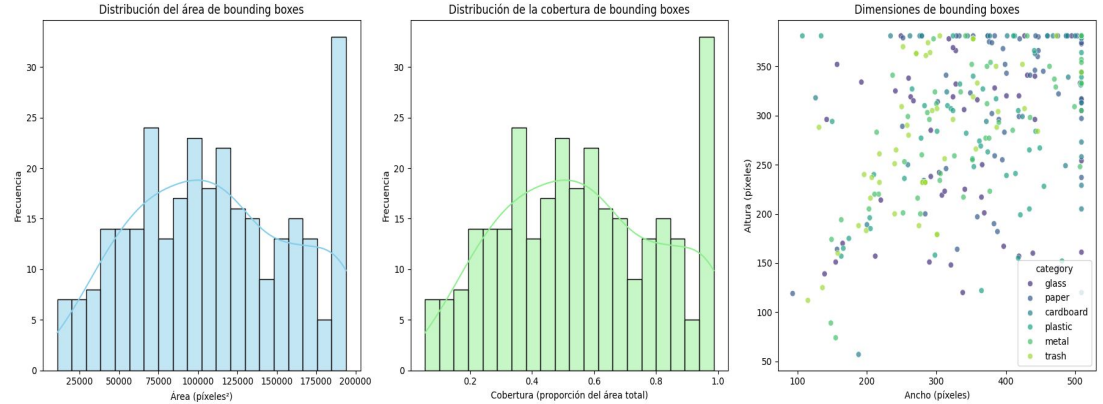
Todas las imágenes del dataset tienen exactamente el mismo tamaño 512x384. La relación de aspecto será 512/384 = 1.33

Análisis exploratorio de datos (EDA)

Análisis de bounding boxes

Estadísticas de bounding boxes:

	bb_width	bb_height	bb_area	bb_coverage
count	300.000000	300.00000	300.000000	300.000000
mean	359.446667	294.12000	109577.990000	0.557342
std	111.264879	77.77215	50035.460285	0.254494
min	93.000000	57.00000	10716.000000	0.054504
25%	280.000000	237.75000	70522.000000	0.358693
50%	363.000000	309.00000	106111.500000	0.539711
75%	454.250000	368.25000	151612.750000	0.771142
max	509.000000	381.00000	193929.000000	0.986374

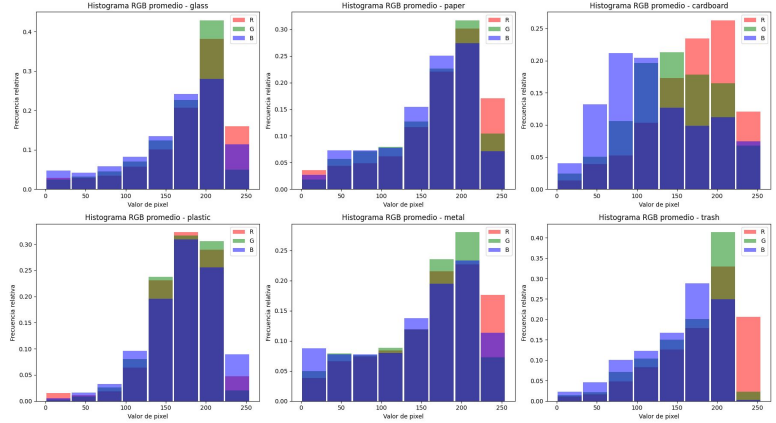


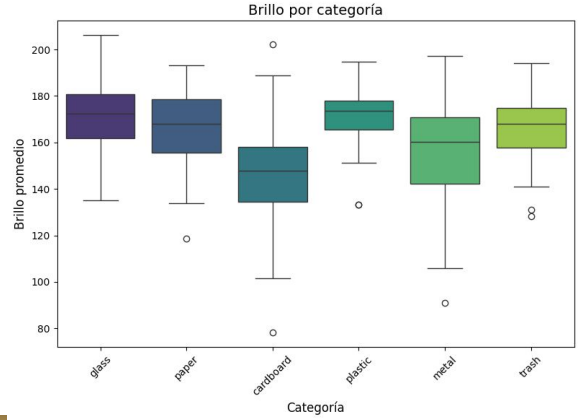
Análisis exploratorio de datos (EDA)

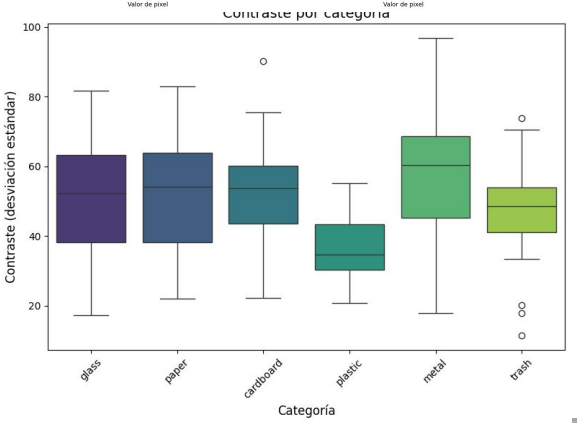
Análisis de distribución de colores

Estadísticas de brillo y contraste:

	brightness	contrast
count	300.000000	300.000000
mean	162.587009	49.641324
std	20.075154	15.377899
min	78.332800	11.504262
25%	152.948933	38.389279
50%	166.634800	49.808609
75%	175.837733	60.884104
max	206.133733	96.718705

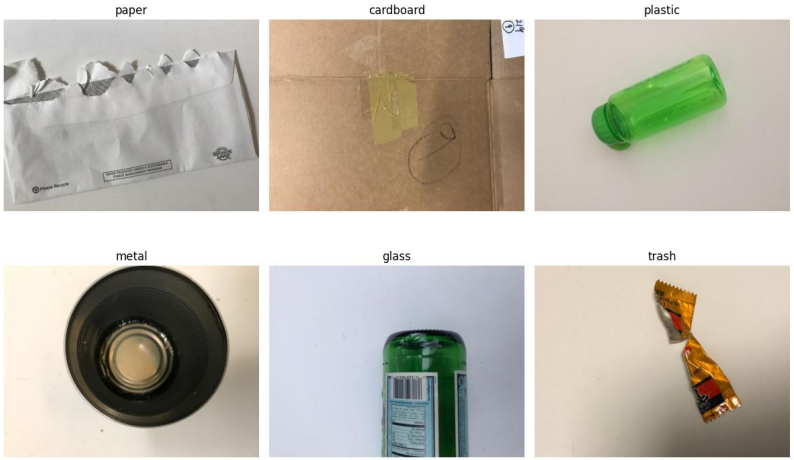






Análisis exploratorio de datos (EDA)

Análisis de calidad de imágenes



- Resolución promedio: 512.0 x 384.0 píxeles
- Relación de aspecto promedio: 1.33
- Brillo promedio: 162.59/255
- Contraste promedio: 49.64
- Cobertura de bounding box promedio: 55.7%

- Visualización de imágenes
- Análisis de integridad de imágenes →Ninguna imagen corrupta
- Chequeo de imágenes duplicadas →No hay imágenes duplicadas
- Análisis de relación de aspecto →Tamaño 512 x 384
- Análisis de brillo y contraste →No hay diferencias claras

Modelo baseline

Para el **baseline** del proyecto se seleccionó una **arquitectura ResNet18**. La selección fue motivada por las siguientes características: es ligero, buen rendimiento, disponible en torchvision.models y admite aprendizaje por transferencia.

Definición del modelo baseline

A continuación se carga el modelo pre-entrenado ResNet18 y se congelan las capas previas:

```
[ ]  def create_empty_model(num_classes):
         model = resnet18(weights=ResNet18_Weights.DEFAULT)
         for param in model.parameters():
             param.requires_grad = False

         model.fc = nn.Linear(model.fc.in_features, num_classes)
         return model


     model = create_empty_model(num_classes=num_classes)
     model.to(device)

     criterion = nn.CrossEntropyLoss()
     optimizer = optim.Adam(model.parameters(), lr=1e-3)
     scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.5)
```

Uso de Transfer Learning: Resnet18

```
model = resnet18(weights=ResNet18_Weights.DEFAULT)
```

ResNet18 con pesos preentrenados en ImageNet. Ahí ocurre el **transfer learning**, porque estamos reutilizando lo que el modelo aprendió en millones de imágenes.

```
for param in model.parameters():
    param.requires_grad = False
```

Esto **congela todas las capas del modelo**, es decir, **no se actualizan durante el entrenamiento**. Así:aprobamos lo que ya aprendió el modelo y solo aprendés la parte nueva (la última capa).
Esto se llama **feature extraction**: usamos la red como un **extractor de características visuales** y entrenamos una capa nueva para la tarea de clasificación.

Uso de Transfer Learning: Resnet18

```
model.fc = nn.Linear(model.fc.in_features, num_classes)
```

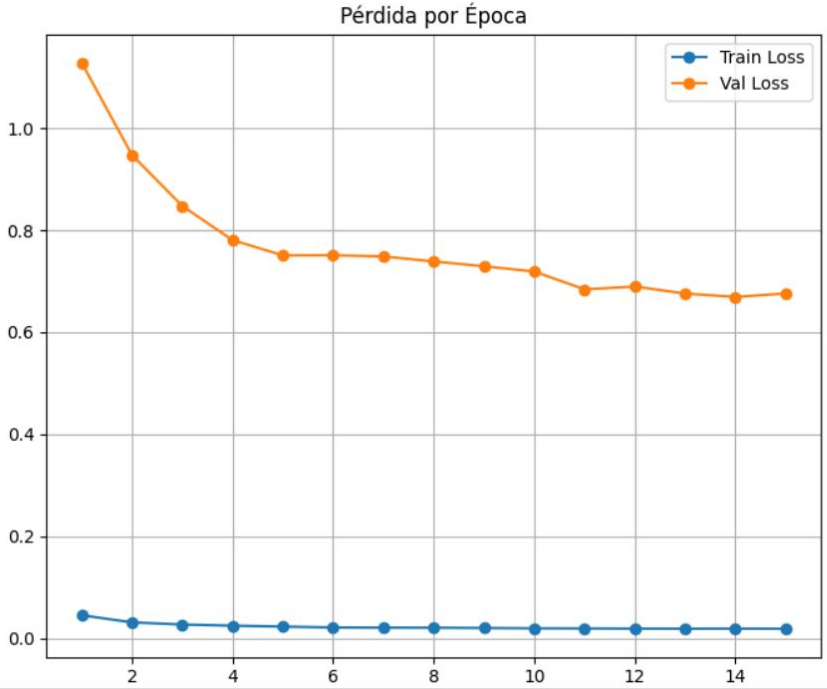
Esto **reemplaza la última capa (fc)** de ResNet18 con una nueva que tiene num_classes salidas, en este caso los 6 tipos de residuos para poder clasificarlos.

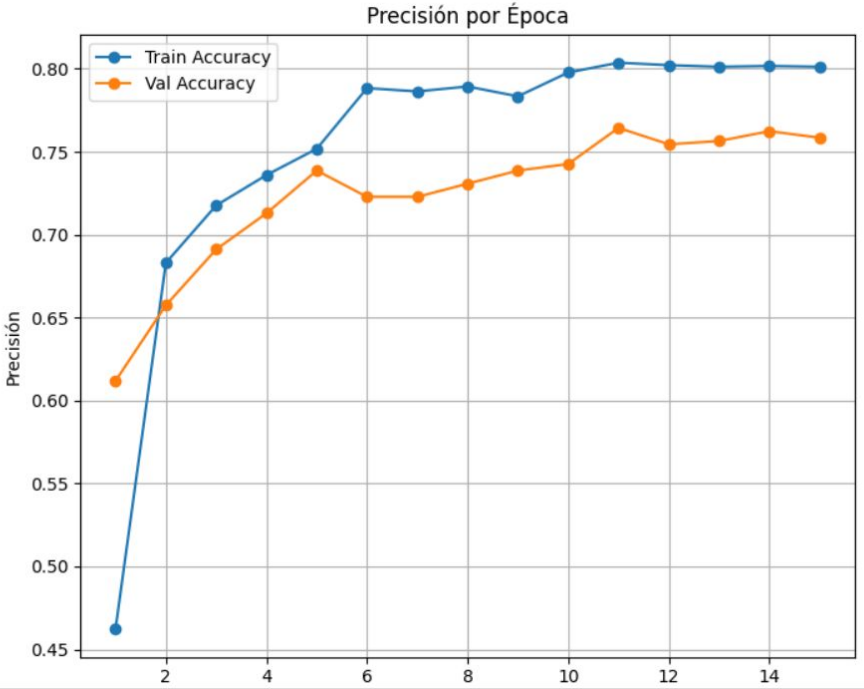
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
```

Aunque model.parameters() incluye todas las capas, **solo se entrenará la capa final**, porque las otras están congeladas (requires_grad=False).

Modelo baseline

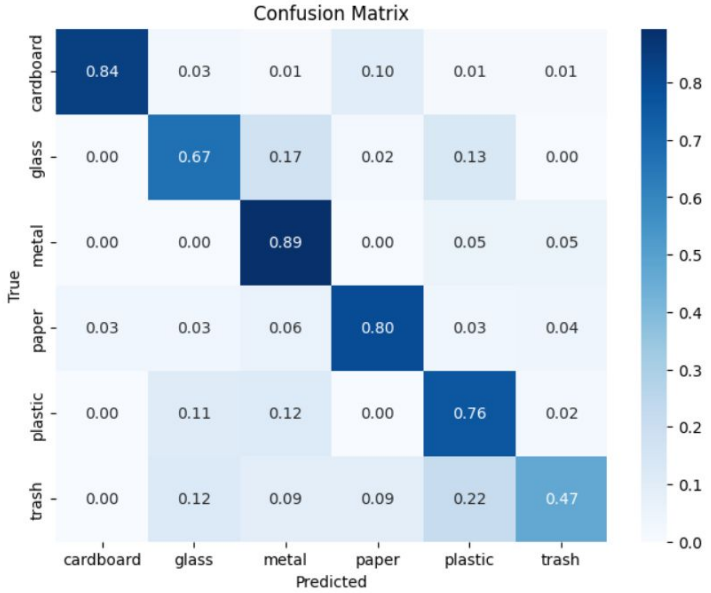
Arquitectura ResNet18. Resultados Sin Data Augmentation





Modelo baseline

Arquitectura ResNet18. Resultados Sin Data Augmentation



Classification report:

	precision	recall	f1-score	support
0	0.94	0.84	0.89	80
1	0.78	0.67	0.72	104
2	0.63	0.89	0.74	75
3	0.88	0.80	0.84	119
4	0.71	0.76	0.73	95
5	0.56	0.47	0.51	32
accuracy			0.76	505
macro avg	0.75	0.74	0.74	505
weighted avg	0.78	0.76	0.77	505

Modelo baseline

Arquitectura ResNet18. Con Data Augmentation

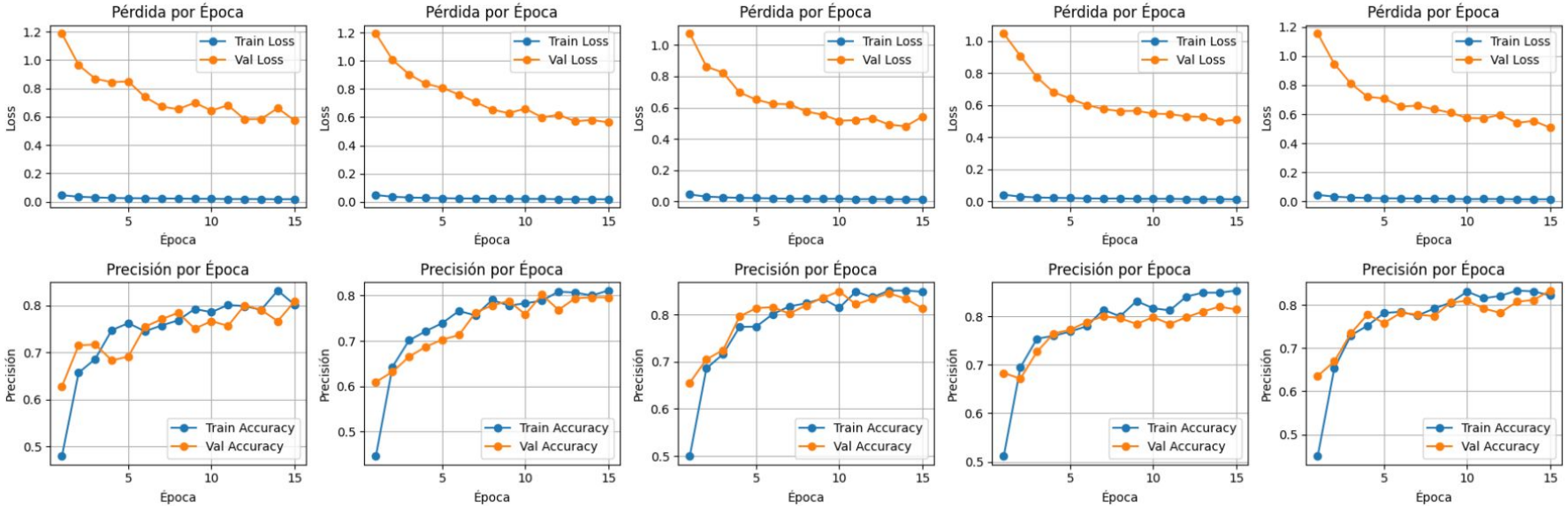
Definición de diferentes transformaciones y ejecución del entrenamiento y validación:

```
[ ]  train_transform0 = transforms.Compose(
        [
            transforms.RandomRotation(25),
            transforms.ToTensor(),
            transforms.Normalize([0.5] * 3, [0.5] * 3),
        ]
    )

    train_transform1 = transforms.Compose(
        [
            transforms.RandomRotation(25),
            transforms.ToTensor(),
            transforms.Normalize([0.5] * 3, [0.5] * 3),
        ]
    )

    train_transform2 = transforms.Compose(
        [
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            transforms.Normalize([0.5] * 3, [0.5] * 3),
        ]
    )
```

```
train_transform3 = transforms.Compose(
    [
        transforms.RandomVerticalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.5] * 3, [0.5] * 3),
    ]
)

train_transform4 = transforms.Compose(
    [
        transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
        transforms.ToTensor(),
        transforms.Normalize([0.5] * 3, [0.5] * 3),
    ]
)

train_transform5 = transforms.Compose(
    [
        transforms.RandomRotation(25),
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.5] * 3, [0.5] * 3),
    ]
)

test_transform = transforms.Compose(
    [transforms.ToTensor(), transforms.Normalize([0.5] * 3, [0.5] * 3)]
```

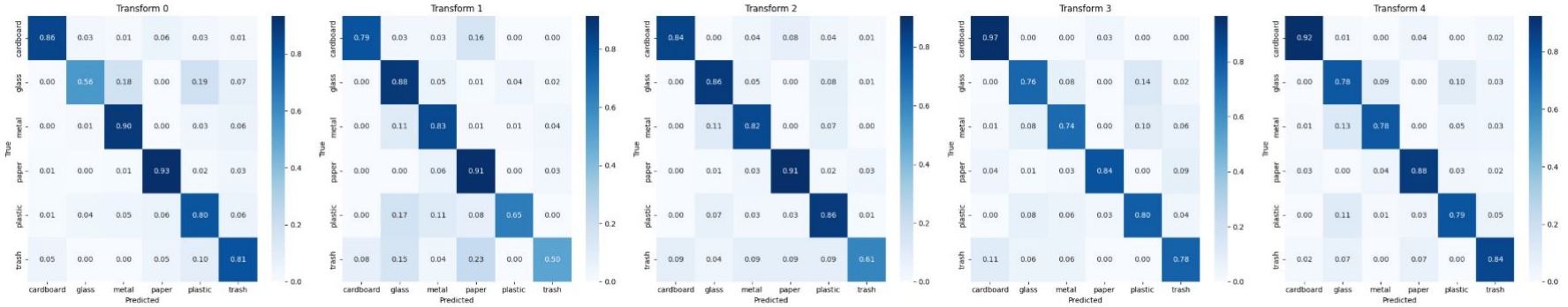
Modelo baseline

Arquitectura ResNet18. Resultados Con Data Augmentation



Modelo baseline

Arquitectura ResNet18. Resultados Con Data Augmentation



Comparando las diferentes matrices de confusión se justifica el uso de las siguientes transformaciones para el Data Augmentation:

- Espejado random horizontal
- Espejado random vertical
- Rotación random

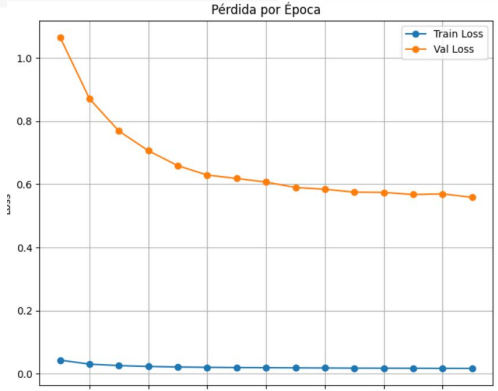
Modelo baseline

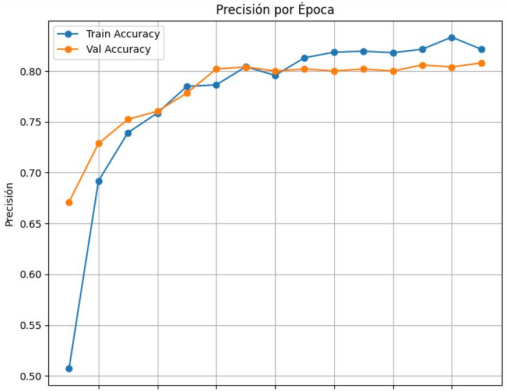
Arquitectura ResNet18. Búsqueda de Hiperparametros con Optuna

Ejecución de la búsqueda.

```
[ ]  study = optuna.create_study(direction="maximize")
     study.optimize(objective, n_trials=20)
```

```
Best trial:
  F1 score: 0.8134532928419761
  Params:
    lr: 0.004367922954598241
    weight_decay: 8.843939602412726e-06
    batch_size: 32
    step_size: 3
    gamma: 0.13565654529804916
```





Modelo EfficientNet-B0

Modelo Elegido: EfficientNet-B0 (https://arxiv.org/abs/1905.11946)

EfficientNet-B0 es una a**rquitectura de red neuronal convolucional (CNN) diseñada para lograr alta precisión usando menos parámetros y menos cómputo que otras redes tradicionales.**
- Muy eficiente: ideal para dispositivos móviles o proyectos con pocos recursos
- Usa Batch Normalization, Swish activations y depthwise separable convolutions
- Escalable a modelos más grandes (B1, B2, ..., B7)

Razones por las que se seleccionó EfficientNet-B0
- TrashNet es **un dataset relativamente pequeño:2527 imágenes**
- EfficientNet-B0 tiene pocos parámetros, lo que ayuda a reducir el riesgo de overfitting
- Ofrece alta precisión con bajo costo computacional

Modelo EfficientNet-B0

Característica	ResNet18	EfficientNet-B0
Año de publicación	2015	2019
Diseñadores	Microsoft Research	Google Brain
Tipo de arquitectura	Red residual (ResNet)	Arquitectura escalada compuesta (depth, width, resolution)
Bloques básicos	Bloques residuales simples (BasicBlock)	MBConv (Mobile Inverted Bottleneck Convolution) con squeeze-and-excitation (SE)
Parámetros aprox.	~11.7 millones	**~5.3 millones**
Profundidad	18 capas (conexiones residuales simples)	Arquitectura más profunda y compleja, pero compacta gracias al escalamiento compuesto
Tamaño de entrada	224x224	**224x224 (pero escalable)**

Característica	ResNet18	EfficientNet-B0
BatchNorm	Después de cada conv	También usa BatchNorm + Swish
Función de activación	ReLU	**Swish** (más suave, no linealidad adaptativa)
Conexiones residuales	Sí	Sí, dentro de los MBConv
Regularización interna	No	Sí (DropConnect)
Estrategia de escalamiento	No aplica	**Sí**: escala ancho, profundidad y resolución de forma conjunta (modelo compuesto)
Velocidad de inferencia	Más rápida en CPU	Más eficiente en GPU (mejor para producción móvil o edge)
Precisión top-1 (ImageNet)	~69.8%	**~77.1%** (mejor rendimiento con menos parámetros)

Modelo EfficientNet-B0

Definición del modelo EfficientNet-B0

```
[ ]  def create_empty_model(num_classes):
         model = timm.create_model("efficientnet_b0", pretrained=True)
         for param in model.parameters():
             param.requires_grad = False
         model.classifier = nn.Linear(model.classifier.in_features, num_classes)
         return model


     model = create_empty_model(num_classes=num_classes)
     model.to(device)

     criterion = nn.CrossEntropyLoss()
     optimizer = optim.Adam(model.parameters(), lr=1e-3)
     scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.5)
```

**timm: Pytorch Image Models*

Uso de Transfer Learning: EfficientNet-B0

```
model = timm.create_model("efficientnet_b0", pretrained=True)
```

Esto crea un modelo **EfficientNet-B0 con pesos preentrenados**. Es decir, la red ya fue entrenada para reconocer 1.000 clases de ImageNet.

```
for param in model.parameters():
    param.requires_grad = False
```

Las capas preentrenadas **no se entrenan de nuevo** (**feature extraction)**.
Solo usamos la red como un "detector de patrones visuales" ya entrenado.

Uso de Transfer Learning: EfficientNet-B0

```
model.classifier = nn.Linear(model.classifier.in_features, num_classes)
```

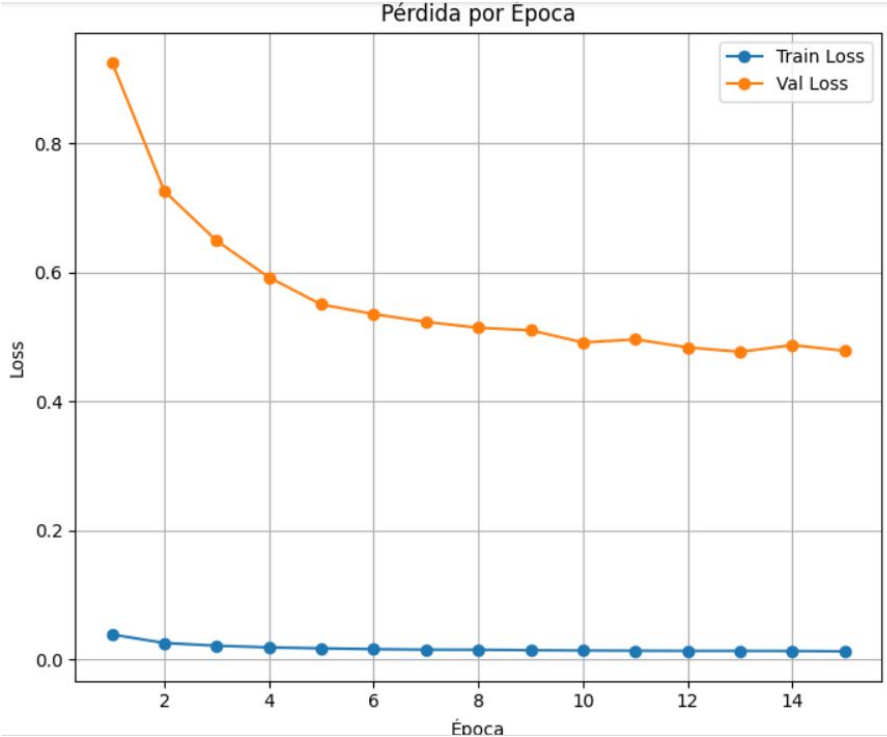
Esto **reemplaza la última capa (fc)** de EfficientNet-B0 con una nueva que tiene num_classes salidas, en este caso los 6 tipos de residuos para poder clasificarlos.

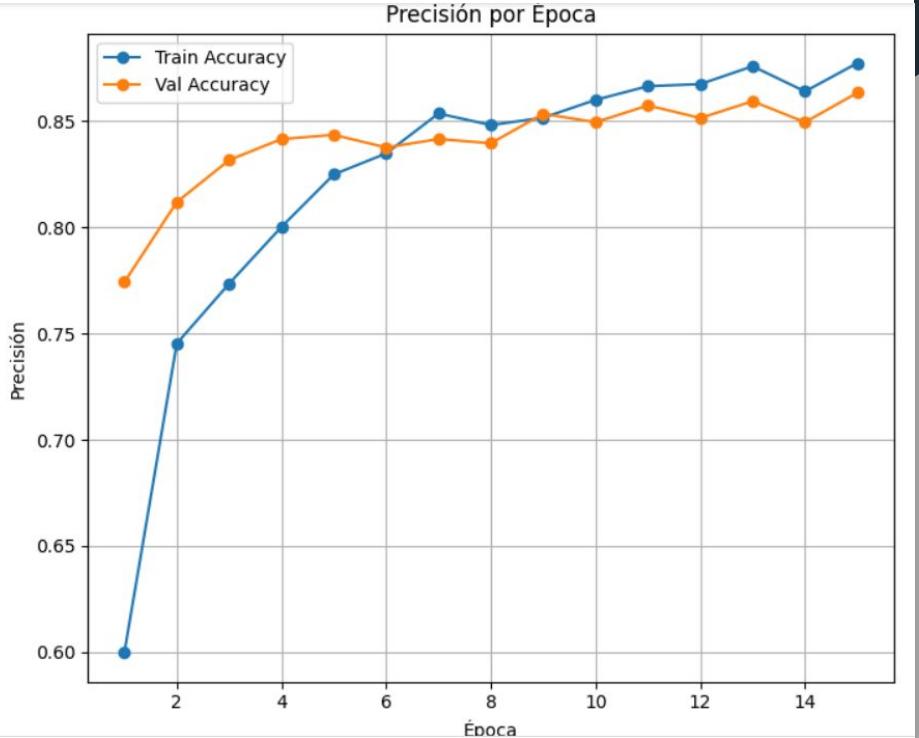
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
```

Aunque model.parameters() incluye todo el modelo, **solo los parámetros de la capa final tienen requires_grad=True**, así que solo se actualiza esa capa

Modelo EfficientNet-B0

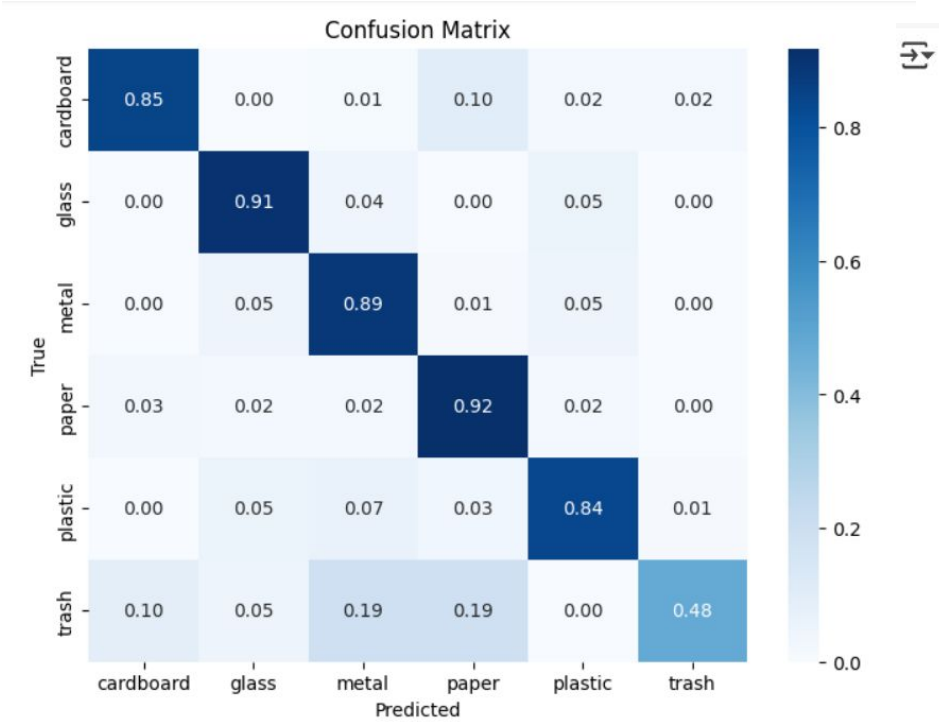
Resultados Sin Data Augmentation





Modelo EfficientNet-B0

Resultados Sin Data Augmentation

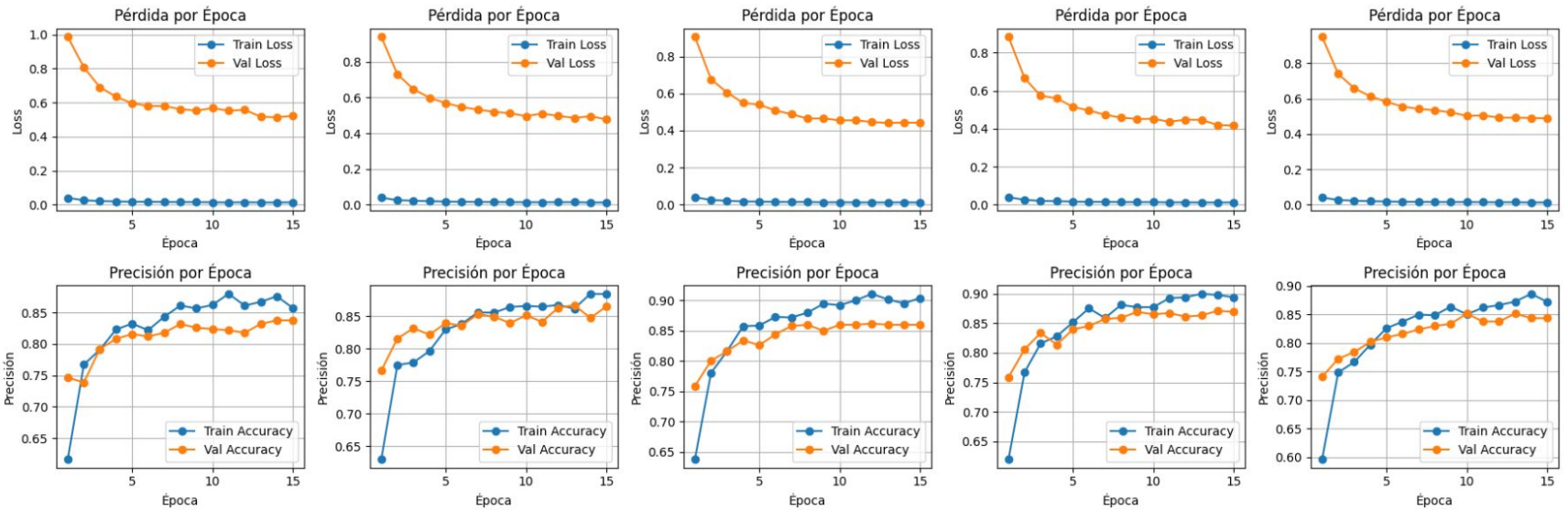


Classification report:

	precision	recall	f1-score	support
0	0.95	0.85	0.90	106
1	0.88	0.91	0.89	95
2	0.81	0.89	0.85	81
3	0.84	0.92	0.88	110
4	0.86	0.84	0.85	92
5	0.77	0.48	0.59	21
accuracy			0.86	505
macro avg	0.85	0.81	0.82	505
weighted avg	0.86	0.86	0.86	505

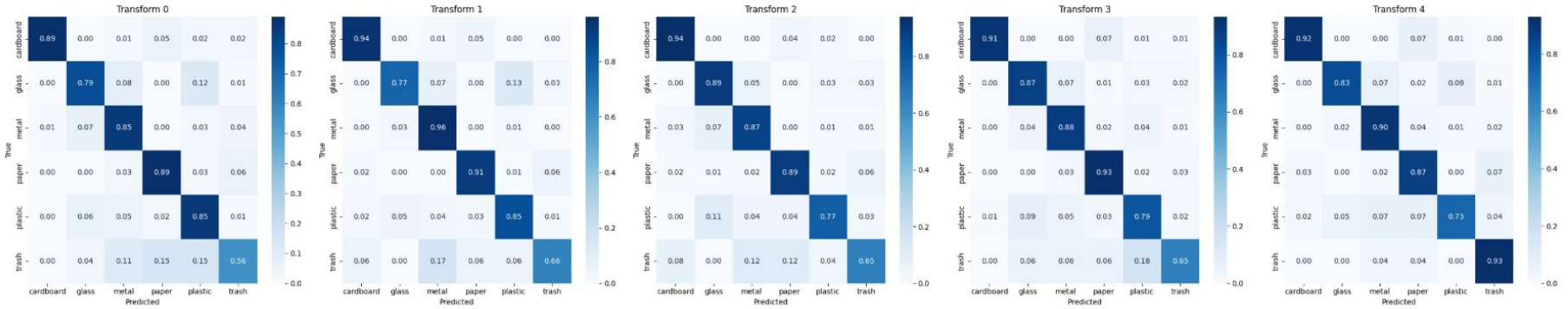
Modelo EfficientNet-B0

Resultados Con Data Augmentation



Modelo EfficientNet-B0

Resultados Con Data Augmentation



A partir de las matrices de confusión se puede observar cómo con data augmentation se mejora considerablemente el desempeño del modelo al clasificar "trash", la clase desbalanceada.

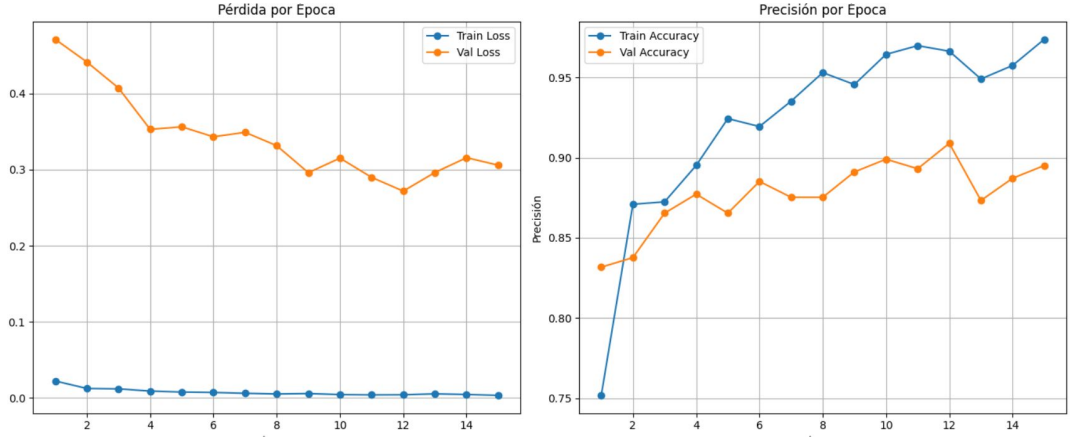
Además, se observa que l**a transformación 2 es con la que se obtiene mejores resultados**. Por lo que solo se utilizará el espejado horizontal como técnica de augmentation.

Modelo EfficientNet-B0

Búsqueda de Hiperparametros con Optuna

```
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=20)
```

```
Best trial:
  F1 score: 0.8762210943161396
  Params:
    lr: 0.009550666304303759
    weight_decay: 0.0007685471197628106
    batch_size: 32
    step_size: 7
    gamma: 0.6275531729477372
```



Métricas de evaluación (indicadores del rendimiento del modelo):

- El **F1 score** combina *precisión* y *recall*, ideal para clases desbalanceadas.

  Cuanto más alto (cerca de 1), mejor.

- **Accuracy (Train y Validation)**:**Train Acc:** Eficiencia del modelo sobre datos de entrenamiento y **Val Acc:** Precisión sobre datos nunca vistos.

- **Loss (Train y Validation)**:Cuantifica el error de predicción del modelo.

Hiperparámetros optimizados

Hiperparámetros evaluados (optimizados con *Optuna*):

- **lr**: Tasa de aprendizaje.
- **weight_decay**: Regularización para evitar overfitting.
- **batch_size**: Tamaño del lote de entrenamiento.
- **step_size** y **gamma**: Parámetros del scheduler para ajustar el *learning rate* dinámicamente.

Resultados y conclusiones

Resultados para Modelo Resnet (Modelo Baseline)

Best trial:

F1 score: **0.8134532928419761** Params: lr: 0.004367922954598241 weight_decay: 8.843939602412726e-06 batch_size: 32 step_size: 3 gamma: 0.13565654529804916

Resultados para EfficientNet-B0

Best trial: F1 score: **0.8762210943161396** Params: lr: 0.009550666304303759 weight_decay: 0.0007685471197628106 batch_size: 32 step_size: 7 gamma: 0.6275531729477372

Resultado final EfficientNet-B0 : Epoch 15/15 | Train Loss: 0.0034, Train Acc: 0.9738 | Val Loss: 0.3059, Val Acc: 0.8950

Resultados y conclusiones

- EfficientNet-B0 **supera a ResNet en F1 score y en precisión de validación**, lo que indica que aprendió mejor a clasificar.
- **El F1 score final de 0.876 es muy bueno**, especialmente si las clases estaban desbalanceadas.

- **Train Acc:** EfficientNet-B0: `0.9738` → Aprendió bien.
- **Val Acc:** EfficientNet-B0: `0.8950` → Generaliza bastante bien.
- **Loss (Train y Validation)**:**Train Loss** bajo: el modelo aprendió.**Val Loss** algo más alto: indica cierto overfitting, pero no crítico.

Resultados y conclusiones

EfficientNet-B0 supera a ResNet18 (Baseline) tanto **en precisión (F1) como en eficiencia, a pesar de tener menos parámetros entrenables.**

EfficientNet fue diseñada explícitamente para ser eficiente y precisa con menor complejidad computacional.

Ambos modelos estamos usando **feature extraction (congelando parámetros) Transfer Learning**.Esto es útil si el dataset no es muy grande, pero si hay suficientes datos, se podrian descongelar parcialmente capas intermedias (fine-tuning profundo) para sacar aún más provecho de **EfficientNet.**

El mejor desempeño **EfficientNet-B0** se debe a **la arquitectura**, no a los hiperparámetros en este caso.

El modelo **EfficientNet-B0** es superior en términos de F1-score (+7% mejor), lo que indica que generaliza mejor en el conjunto de validación.

Resultados y conclusiones

- **EfficientNet-B0** es **superior** en **tareas reales de clasificación**, especialmente si buscás **mayor precisión con menos recursos computacionales**.

- **ResNet18** sigue siendo una **excelente opción** cuando querés algo simple, **rápido de entrenar y fácil de modificar**, **ideal para prototipos** o cuando la interpretabilidad importa más que la precisión.

Pruebas de Predicción con imágenes distintas al dataset

Carga del modelo EfficientNet B0

```
[ ]  model = create_empty_model(num_classes=6)
     model.load_state_dict(torch.load("output/trashnet_efficientnet_b0.pth", map_location=device))
     model.to(device)
     model.eval()
```

```
EfficientNet(
  (conv_stem): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
  (bn1): BatchNormAct2d(
    32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
    (drop): Identity()
    (act): SiLU(inplace=True)
  )
  (blocks): Sequential(
    (0): Sequential(
      (0): DepthwiseSeparableConv(
        (conv_dw): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32, bias=False)
        (bn1): BatchNormAct2d(
          32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
          (drop): Identity()
          (act): SiLU(inplace=True)
        )
        (aa): Identity()
        (se): SqueezeExcite(
          (conv_reduce): Conv2d(32, 8, kernel_size=(1, 1), stride=(1, 1))
          (act1): SiLU(inplace=True)
          (conv_expand): Conv2d(8, 32, kernel_size=(1, 1), stride=(1, 1))
          (gate): Sigmoid()
        )
        (conv_pw): Conv2d(32, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn2): BatchNormAct2d(
          16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
          (drop): Identity()
          (act): Identity()
        )
        (drop_path): Identity()
      )
```

```
    )
    (1): Sequential(
      (0): InvertedResidual(
        (conv_pw): Conv2d(16, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNormAct2d(
          96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
          (drop): Identity()
          (act): SiLU(inplace=True)
        )
        (conv_dw): Conv2d(96, 96, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), groups=96, bias=False)
        (bn2): BatchNormAct2d(
          96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
          (drop): Identity()
          (act): SiLU(inplace=True)
        )
        (aa): Identity()
        (se): SqueezeExcite(
          (conv_reduce): Conv2d(96, 4, kernel_size=(1, 1), stride=(1, 1))
          (act1): SiLU(inplace=True)
          (conv_expand): Conv2d(4, 96, kernel_size=(1, 1), stride=(1, 1))
          (gate): Sigmoid()
        )
        (conv_pwl): Conv2d(96, 24, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNormAct2d(
          24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
          (drop): Identity()
          (act): Identity()
```

Pruebas de Predicción con imágenes distintas al dataset

Residuo1.jpg

Residuo2.jpg

Residuo3.jpg

Pruebas de Predicción con imágenes distintas al dataset

Residuo4.jpg

Residuo5.jpg

Residuo6.jpg

Pruebas de Predicción con imágenes distintas al dataset

Se aplica una **transformación del input equivalente a la utilizada en entrenamiento.**

Predicción

```
def predict_image(image_path):
    image = Image.open(image_path).convert("RGB")
    input_tensor = transform(image).unsqueeze(0).to(device)  # agrega batch dimension y lleva a device
    with torch.no_grad():
        output = model(input_tensor)
        probs = F.softmax(output, dim=1)
        confidence, pred_class = torch.max(probs, dim=1)
    return class_names[pred_class.item()], confidence.item()
```

```
[ ] plt.figure(figsize=(12, 8))
    for i, image_name in enumerate(image_paths):
        image_path = os.path.join(INPUT_PATH, image_name)
        pred_class, confidence = predict_image(image_path)
        image = Image.open(image_path).convert("RGB")
        plt.subplot(2, 3, i + 1)
        plt.imshow(image)
        plt.title(f"{pred_class} ({confidence*100:.1f}%)")
        plt.axis("off")

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

Pruebas de Predicción con imágenes distintas al dataset

cardboard (96.1%)

metal (99.3%)

plastic (91.3%)

Pruebas de Predicción con imágenes distintas al dataset

paper (97.6%)

glass (98.6%)

trash (98.1%)

Muchas Gracias!