



Universidad de
los Andes

> **FACULTAD
DE INGENIERÍA
Y CIENCIAS
APLICADAS**

Reconocimiento Visual con Deep Learning

Informe Tarea 1

Integrantes:

Sebastián Maluk

Javier Soto

Profesor Cátedra:

José Manuel Saavedra

27 de agosto del 2023, Santiago de Chile

Resumen

En este trabajo, se aborda el desafío del reconocimiento visual utilizando dos arquitecturas de redes neuronales convolucionales. Se entrenaron dos modelos: uno basado en una red convolucional simple con tres capas convolucionales y otro que utiliza la arquitectura ResNet-18 más compleja. Estos modelos fueron entrenados para clasificar imágenes de tamaño 128x128 pertenecientes al conjunto de datos de Eitz, que contiene 20.000 imágenes distribuidas en 250 clases. Se utilizó el optimizador Adam con configuración predeterminada para el entrenamiento, que se realizó en aproximadamente 50 épocas con un tamaño de batch de 64. Se realizaron análisis completos de los resultados, incluidos gráficos de pérdida y precisión frente a épocas, matrices de confusión, precisión total y por clase., así como la identificación de casos difíciles y fáciles. Los resultados y observaciones obtenidos en este estudio proporcionan información valiosa sobre el rendimiento de las arquitecturas de red utilizadas en esta tarea de reconocimiento visual.

Índice

Resumen	2
Índice	3
1. Introducción	4
2. Desarrollo	5
2.1 Diseño del Experimento	5
2.2 Implementación	6
2.3 Métricas	10
3. Resultados Experimentales y Discusión	13
3.1 Métricas Generales	13
3.2 Matriz de Confusión	14
3.3 Métricas de Precisión	17
3.4 Las 10 Mejores y Peores Clases	18
3.5 Ejemplos de Casos	19
3.6 Recomendaciones	20
4. Conclusiones	21

1. Introducción

El reconocimiento visual es un campo crucial dentro de la inteligencia artificial, con aplicaciones que abarcan desde la detección de objetos hasta la clasificación de imágenes. En este trabajo, nos centramos en el problema de la clasificación de ‘sketches’ mediante el uso de redes neuronales convolucionales. Dos arquitecturas diferentes fueron empleadas para abordar este desafío.

Uno de los modelos proporcionados, conocido como el modelo Simple, es una arquitectura de red convolucional diseñada específicamente para tareas de clasificación de imágenes. Esta red utiliza capas convolucionales seguidas de capas de Batch Normalization y activaciones ReLU para aprender características clave de las imágenes. Posteriormente, las características extraídas se aplana y se conectan a capas densas para realizar la clasificación. En la capa de salida, se aplica una función de activación softmax para calcular las probabilidades de pertenencia. Por otro lado, el modelo ResNet-18 consta de varias capas residuales, que son bloques de construcción fundamentales en la arquitectura ResNet. Cada bloque residual se compone de capas de Normalización por lotes, activaciones ReLU y capas convolucionales. Es importante destacar que este modelo no tiene una función softmax en la capa de salida, lo que tiene un impacto en el desarrollo posterior del proyecto.

El conjunto de datos utilizados para entrenar y evaluar los modelos proviene del conjunto Eitz, que consta de 20.000 imágenes organizadas en 250 clases. Cada imagen en el conjunto de datos representa un boceto o ‘sketch’ y tiene un tamaño de 128x128 píxeles.

En este informe, presentamos el diseño y la implementación de los modelos, detallamos la metodología de entrenamiento y evaluación, y analizamos exhaustivamente los resultados obtenidos. A través de gráficos de pérdida y precisión a lo largo de las épocas, matrices de confusión y métricas de rendimiento por clase, buscamos comprender cómo se desempeñan las dos arquitecturas. Además, exploramos casos difíciles y fáciles para arrojar luz sobre las limitaciones y fortalezas de los modelos. Los insights derivados de este análisis pueden proporcionar recomendaciones valiosas para futuros trabajos en reconocimiento visual y clasificación de bocetos.

2. Desarrollo

2.1 Diseño del Experimento

En este experimento, se diseñaron e implementaron dos modelos de aprendizaje profundo para clasificación de imágenes. Los modelos se evaluaron en un conjunto de datos que consiste en dibujos hechos a mano de diferentes clases.

Modelo Simple

Un modelo de red neuronal convolucional básico que consta de capas convolucionales, capas de agrupación y una capa completamente conectada para la clasificación.

Modelo ResNet-18

Un modelo de red neuronal convolucional más avanzado que utiliza bloques residuales para facilitar el aprendizaje de características más complejas.

Bibliotecas Utilizadas

- TensorFlow para implementar y entrenar los modelos
- Pandas para la manipulación de datos
- NumPy para cálculos matemáticos
- Matplotlib para la visualización de datos
- Sklearn para las métricas

2.2 Implementación

Preprocesamiento de Datos

Para cargar y ordenar los datos se utilizó la herramienta de Pandas, su implementación fue necesaria para determinar las imágenes y las etiquetas dentro del dataset.

```
file_path = os.path.join(DATA_PATH, "train.txt")
test_path = os.path.join(DATA_PATH, "test.txt")
mapping_path = os.path.join(DATA_PATH, "mapping.txt")

df = pd.read_csv(file_path, names=["images", "labels"], sep="\t")
df_test = pd.read_csv(test_path, names=["images", "labels"], sep="\t")

classes = pd.read_csv(mapping_path, names=["class", "idx"], sep="\t").to_dict(
    orient="dict"
)["class"]

df["labels"] = df["labels"].map(classes)
df["images"] = df["images"].map(lambda x: os.path.join(DATA_PATH, x))

df_test["labels"] = df_test["labels"].map(classes)
df_test["images"] = df_test["images"].map(lambda x: os.path.join(DATA_PATH, x))
```

Imagen 1: Código de la carga de datos.

Fuente: Elaboración propia.

También se utilizaron generadores de datos de imagen para cargar y preprocesar las imágenes. El IMAGE_SIZE es igual a 128x128 y el BATCH_SIZE es igual a 64, Además se aplicó una división de 80-20 para los datos de entrenamiento y validación, respectivamente. Se quiso tener generadores de entrenamiento y validación para monitorear el rendimiento real del entrenamiento de cada modelo, siendo ambos factores importantes en los gráficos más adelante.

```

datagen_kwarg = dict(rescale=1.0 / 255, validation_split=0.20)

train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(**datagen_kwarg)

train_generator = train_datagen.flow_from_dataframe(
    df,
    x_col="images",
    y_col="labels",
    target_size=IMAGE_SIZE,
    color_mode="grayscale",
    class_mode="categorical",
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=None,
    subset="training",
    classes=list(classes.values()),
)

validation_generator = train_datagen.flow_from_dataframe(
    df,
    x_col="images",
    y_col="labels",
    target_size=IMAGE_SIZE,
    color_mode="grayscale",
    class_mode="categorical",
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=None,
    subset="validation",
    classes=list(classes.values()),
)

```

Imagen 2: Implementación de generadores de datos de imágenes.

Fuente: Elaboración propia.

Construcción, Compilación y Entrenamiento de los Modelos

Modelo Simple

Se implementó utilizando la biblioteca TensorFlow. Consiste en capas convolucionales seguidas de capas de agrupación y una capa densa para la clasificación. Se usó la función de pérdida de entropía cruzada categórica y el optimizador Adam para el entrenamiento.

```

if os.path.exists("models/simple.keras"):
    loaded = True
    model_simple = tf.keras.models.load_model("models/simple.keras")
else:
    loaded = False
    model_simple = simple.SimpleModel(NUM_CLASSES)

shape_data = train_generator[0][0].shape[1:]
model_simple = model_simple.model(shape_data)

model_simple.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
    metrics=[tf.keras.metrics.CategoricalAccuracy()],
)
model_simple.summary()

```

Imagen 3: Construcción y compilación del modelo Simple.

Fuente: Elaboración propia.

La configuración de entrenamiento para el modelo Simple consistió en ingresar como parámetros la cantidad de 50 epochs (asignada a la variable global EPOCHS), los generadores de datos de imagen de entrenamiento y validación. Entonces, el entrenamiento fue de la siguiente manera:

```

if not loaded:
    history_simple = model_simple.fit(
        train_generator, epochs=EPOCHS, validation_data=validation_generator
    )
    model_simple.save("models/simple.keras")

```

Imagen 4: Entrenamiento del modelo Simple.

Fuente: Elaboración propia.

Modelo ResNet-18

También se implementó utilizando TensorFlow. En la construcción del modelo ResNet-18, se define su arquitectura mediante la especificación de los tamaños de bloque y filtros. Luego, se crea una instancia del modelo. El tamaño de entrada previsto para las imágenes de entrada se establece en lotes de tamaño `BATCH_SIZE` igual a 64, imágenes de 128x128 píxeles y un solo canal (escala de grises). Similar al modelo Simple, se utilizó la entropía cruzada categórica y el optimizador Adam. Es importante mencionar que como el

modelo de ResNet-18 no tiene una función softmax en la capa de salida, en la función de entropía cruzada categórica se le agregó el parámetro de `from_logits=True`. El atributo `from_logits=True` informa a la función de pérdida que los valores de salida generados por el modelo no están normalizados, dado que no se les ha aplicado la función softmax para producir una distribución de probabilidad.

```
if os.path.exists("models/resnet.keras"):
    loaded = True
    model_resnet = tf.keras.models.load_model("models/resnet.keras")
else:
    loaded = False
    block_sizes = [2, 2, 2, 2]
    filters = [64, 128, 256, 512]
    model_resnet = resnet.ResNet(block_sizes, filters, NUM_CLASSES)
    input_shape = (BATCH_SIZE, 128, 128, 1)
    model_resnet.build(input_shape=input_shape)
    model_resnet.compile(
        optimizer=tf.keras.optimizers.Adam(),
        loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
        metrics=[tf.keras.metrics.CategoricalAccuracy()],
    )
model_resnet.summary()
```

Imagen 5: Construcción y compilación del modelo ResNet-18.

Fuente: Elaboración propia.

La configuración de entrenamiento para el modelo ResNet-18 fue la misma que la del modelo Simple. En los parámetros la cantidad de 50 epochs y los generadores de datos de imagen de entrenamiento y validación.

```
if not loaded:
    history = model_resnet.fit(
        train_generator, epochs=EPOCHS, validation_data=validation_generator
    )
    model_resnet.save("models/resnet.keras")
```

Imagen 6: Entrenamiento del modelo ResNet-18.

Fuente: Elaboración propia.

2.3 Métricas

Tras el entrenamiento de ambos modelos se procedió a obtener las métricas y probar cada uno de los modelos. El primer paso fue obtener un generador de datos de la imagen de testeo, donde el IMAGE_SIZE es igual a 128x128 y el BATCH_SIZE es igual a 64.

```
test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(**datagen_kwargs)

test_generator = train_datagen.flow_from_dataframe(
    df_test,
    x_col="images",
    y_col="labels",
    target_size=IMAGE_SIZE,
    color_mode="grayscale",
    class_mode="categorical",
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=None,
    subset="training",
    classes=list(classes.values())),
)
```

Imagen 7: Generador de datos de la imagen de testeo para ambos modelos

Fuente: Elaboración propia.

Precisión

Para evaluar el rendimiento de ambos modelos en esta tarea de clasificación multiclase, se optó por utilizar la métrica de "Precisión Categórica" (Categorical Accuracy).

A diferencia de otras métricas, la precisión categórica es intuitiva y fácil de entender. Simplemente muestra el porcentaje de predicciones correctas, lo que facilita su interpretación en términos prácticos.

Esta métrica es especialmente útil en problemas de clasificación donde hay más de dos clases. Calcula el número de etiquetas verdaderamente positivas en todas las clases y lo divide por el número total de casos, proporcionando una visión integral del rendimiento del modelo.

Al utilizar la misma métrica para evaluar ambos modelos, se facilita la comparación directa de su rendimiento. Esto es especialmente útil cuando se trabaja con arquitecturas de modelos diferentes, como es el caso de nuestro modelo Simple y ResNet-18.

Matriz de Confusión

A continuación se presentan las implementaciones en el código para generar las matrices de confusión para ambos modelos.

Modelo Simple

```
mc_simple = 0
for batch in range(len(test_generator)):
    imgs = test_generator[batch][0]

    y_true = test_generator[batch][1]
    y_pred = model_simple(imgs, training=False)

    # computing confusion_matrix
    mc_simple += metrics.confusion_matrix(y_true, y_pred, NUM_CLASSES)

print(mc_simple)
# mc as percentages
rmc = mc_simple.astype(np.float32) / np.sum(mc_simple, axis=1, keepdims=True)
rmc = (rmc * 100).astype(np.int32) / 100

print(rmc)
```

Imagen 8: Código para generar la matriz de confusión para el modelo Simple

Fuente: Elaboración propia.

Modelo ResNet-18

```
mc_resnet = 0
for batch in range(len(test_generator)):
    imgs = test_generator[batch][0]

    y_true = test_generator[batch][1]
    y_pred = model_resnet(imgs, training=False)

    # computing confusion_matrix
    mc_resnet += metrics.confusion_matrix(y_true, y_pred, NUM_CLASSES)
print(mc_resnet)
# mc as percentages
rmc = mc_resnet.astype(np.float32) / np.sum(mc_resnet, axis=1, keepdims=True)
rmc = (rmc * 100).astype(np.int32) / 100

print(rmc)
```

Imagen 9: Código para generar la matriz de confusión para el modelo ResNet-18

Fuente: Elaboración propia.

3. Resultados Experimentales y Discusión

3.1 Métricas Generales

Épocas vs Pérdida (Loss)

A medida que aumentaban las épocas, observamos una disminución en la pérdida tanto para los modelos Simple como para ResNet-18. Esto es esperado, ya que el modelo aprende a ajustar sus pesos para minimizar la pérdida. Sin embargo, el modelo Simple mostró una pérdida más alta en comparación con el modelo ResNet-18, lo que indica que este último es mejor para capturar las complejidades de los datos.

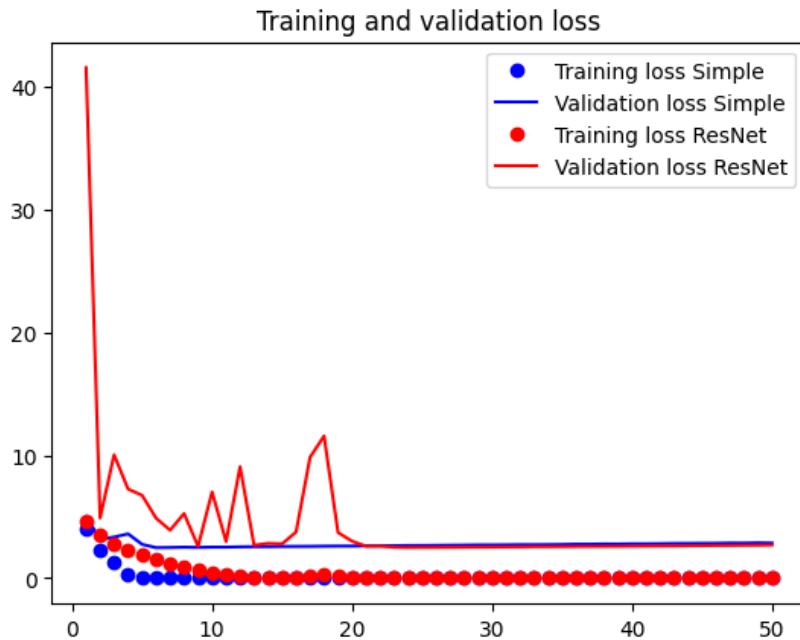


Gráfico 1: Épocas vs Perdida de entrenamiento y validación en ambos modelos.

Fuente: Elaboración propia.

Épocas vs Precisión (Accuracy)

La precisión de ambos modelos mejoró a medida que aumentaba el número de épocas. Esta tendencia sugiere que los modelos estaban aprendiendo las características de manera efectiva. Similar a la pérdida, el modelo ResNet-18 tuvo una mayor precisión en comparación con el modelo Simple.

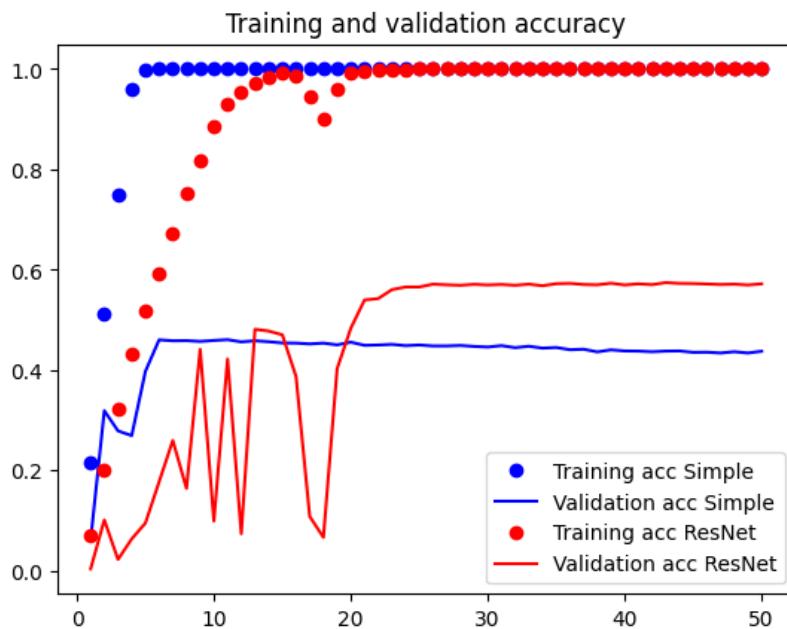


Gráfico 2: Épocas vs Precisión de entrenamiento y validación en ambos modelos.

Fuente: Elaboración propia.

3.2 Matriz de Confusión

Modelo Simple

La matriz de confusión para el modelo Simple mostró una línea diagonal bastante buena, lo que sugiere que las clases están en su mayoría bien separadas. Sin embargo, todavía había algunos elementos fuera de la diagonal que eran relativamente altos, lo que indica clasificaciones erróneas.

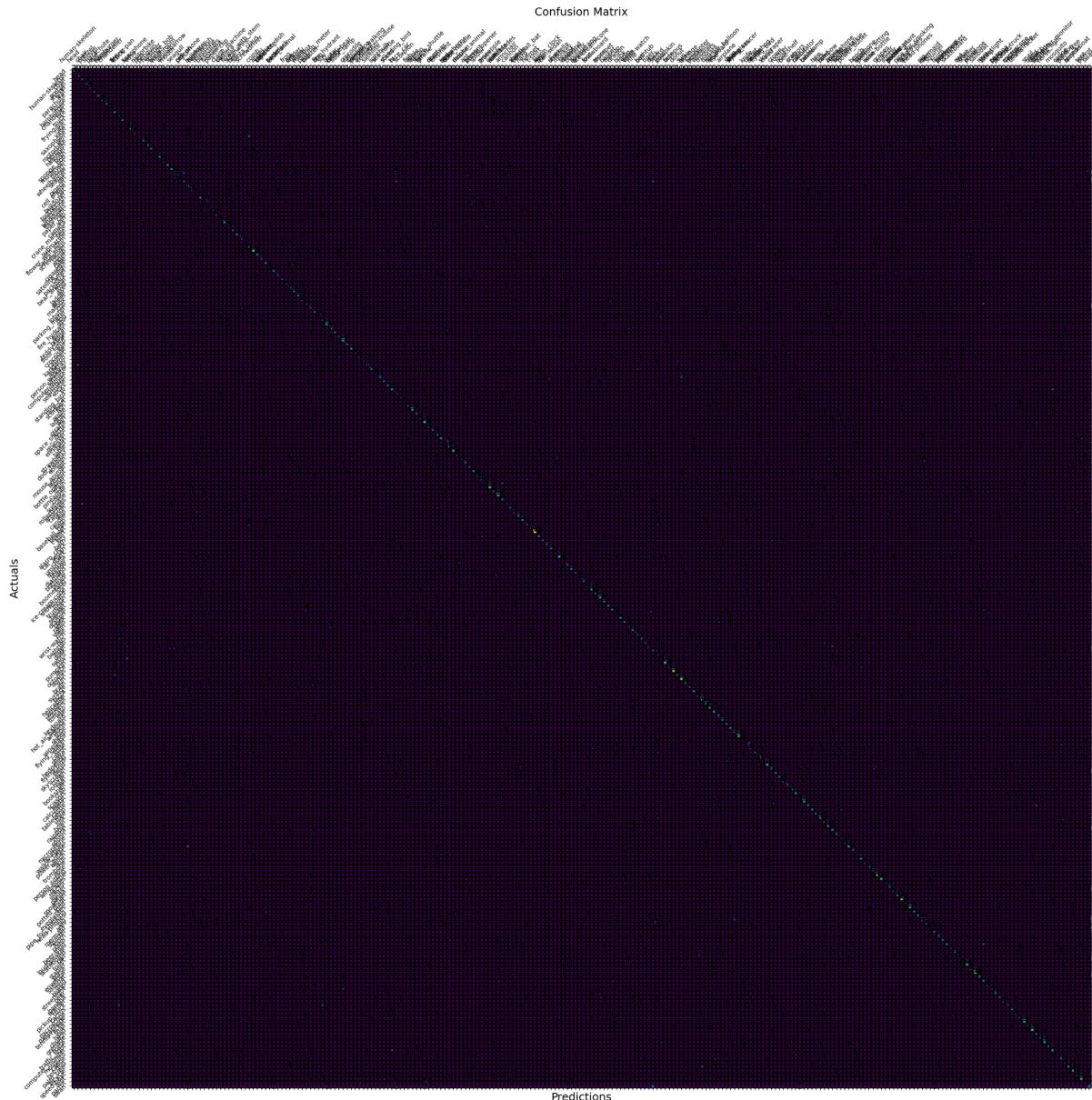


Gráfico 3: Matriz de Confusión graficada para el modelo Simple.

Fuente: Elaboración propia.

Modelo ResNet-18

La matriz de confusión para el modelo ResNet-18 mostró una línea diagonal más fuerte, lo que indica una mayor tasa de clasificaciones correctas. Las clasificaciones erróneas fueron menos en comparación con el modelo Simple.

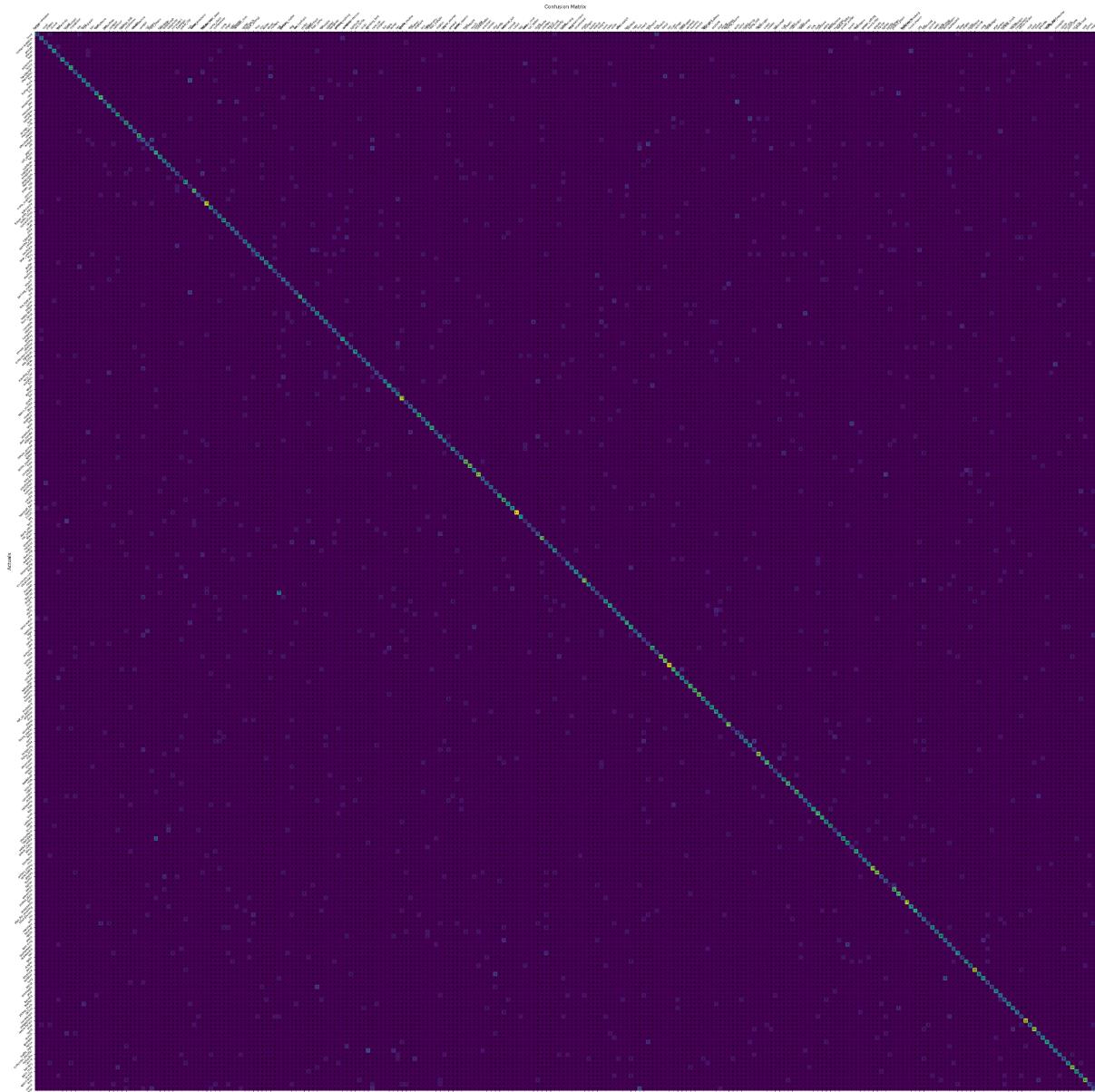


Gráfico 4: Matriz de Confusión graficada para el modelo ResNet-18.

Fuente: Elaboración propia.

3.3 Métricas de Precisión

Precisión Total

El modelo ResNet-18 tuvo una mayor precisión total en comparación con el modelo Simple. Esto podría deberse a que el modelo ResNet-18 es más capaz de aprender características complejas.

Total Accuracy Simple

46.44%

Total Accuracy ResNet

58.19%

Precisión por Clase

El gráfico de barras para la precisión por clase mostró que algunas clases eran más fáciles de predecir que otras para ambos modelos. En general, ResNet-18 tenía mayores precisiones por clase en comparación con el modelo Simple.

Modelo Simple

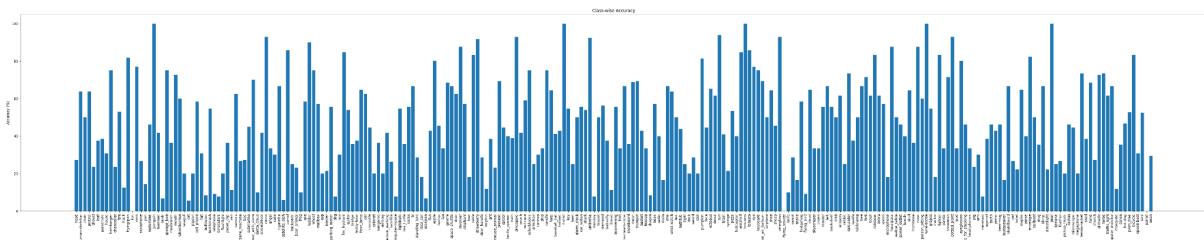


Gráfico 5: Precisión por clase para el modelo Simple.

Fuente: Elaboración propia.

Modelo ResNet-18

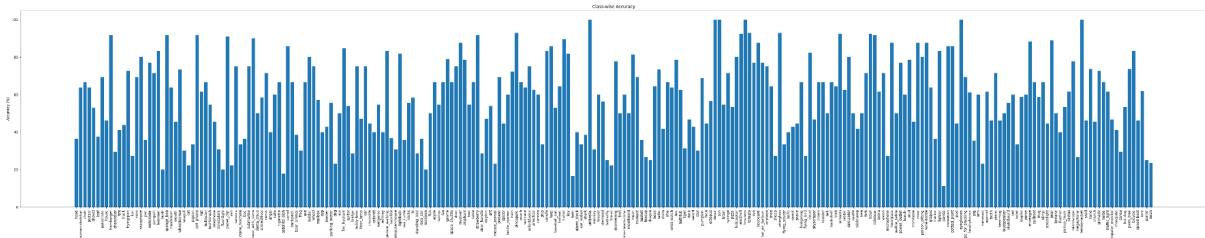


Gráfico 6: Precisión por clase para el modelo ResNet-18.

Fuente: Elaboración propia.

3.4 Las 10 Mejores y Peores Clases

Modelo Simple

Las 10 mejores clases tuvieron precisiones muy altas, posiblemente porque estas clases tenían características distintivas que eran más fáciles de aprender para el modelo. Las 10 peores clases probablemente tenían características más ambiguas o similares a otras clases, lo que las hacía más difíciles de predecir.

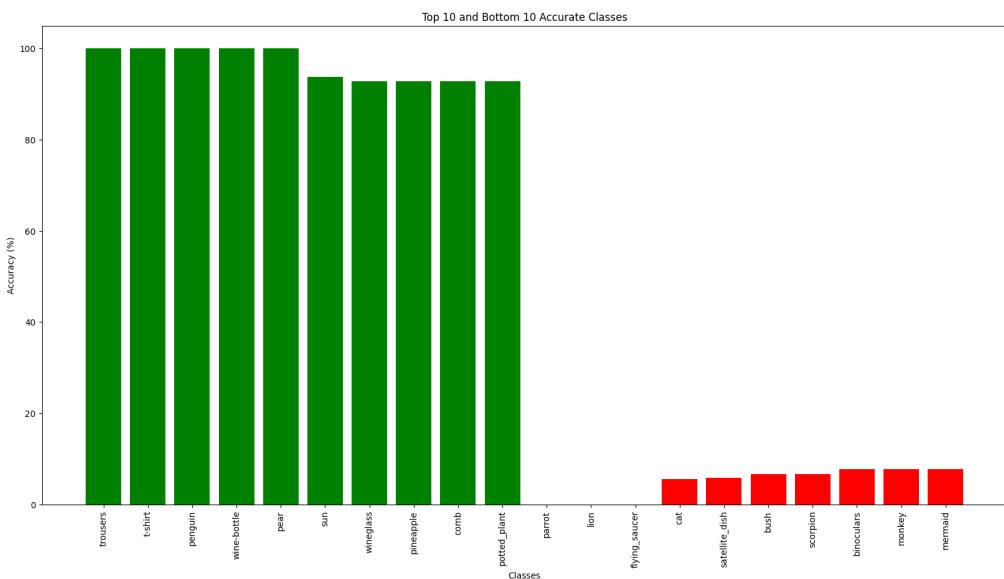


Gráfico 7: 10 mejores y peores clases para el modelo Simple.

Fuente: Elaboración propia.

Modelo ResNet-18

Se observaron tendencias similares, pero el modelo ResNet-18 tenía mayores precisiones tanto para las 10 mejores como para las 10 peores clases en comparación con el modelo Simple.

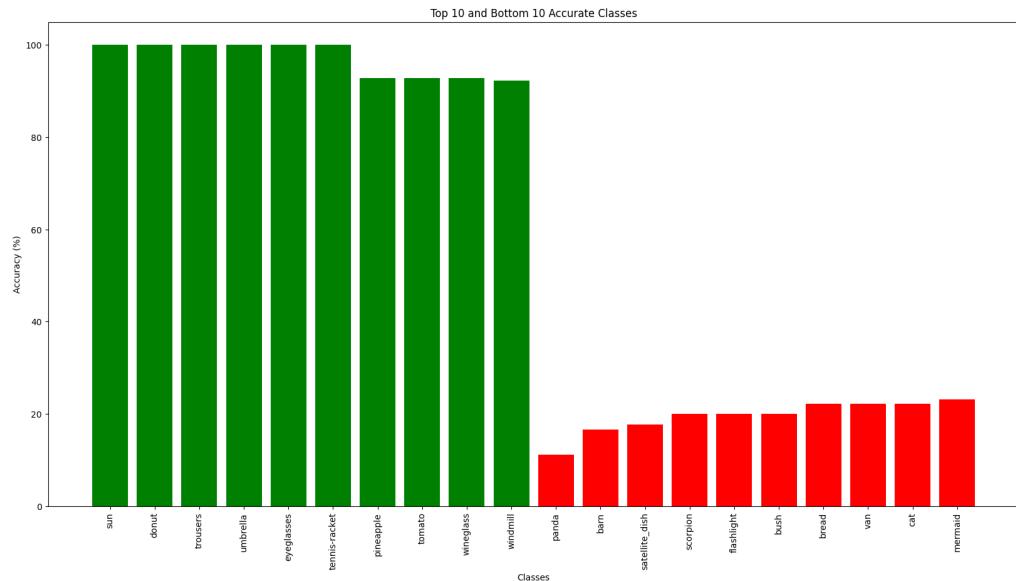


Gráfico 8: 10 mejores y peores clases para el modelo ResNet-18.

Fuente: Elaboración propia.

3.5 Ejemplos de Casos

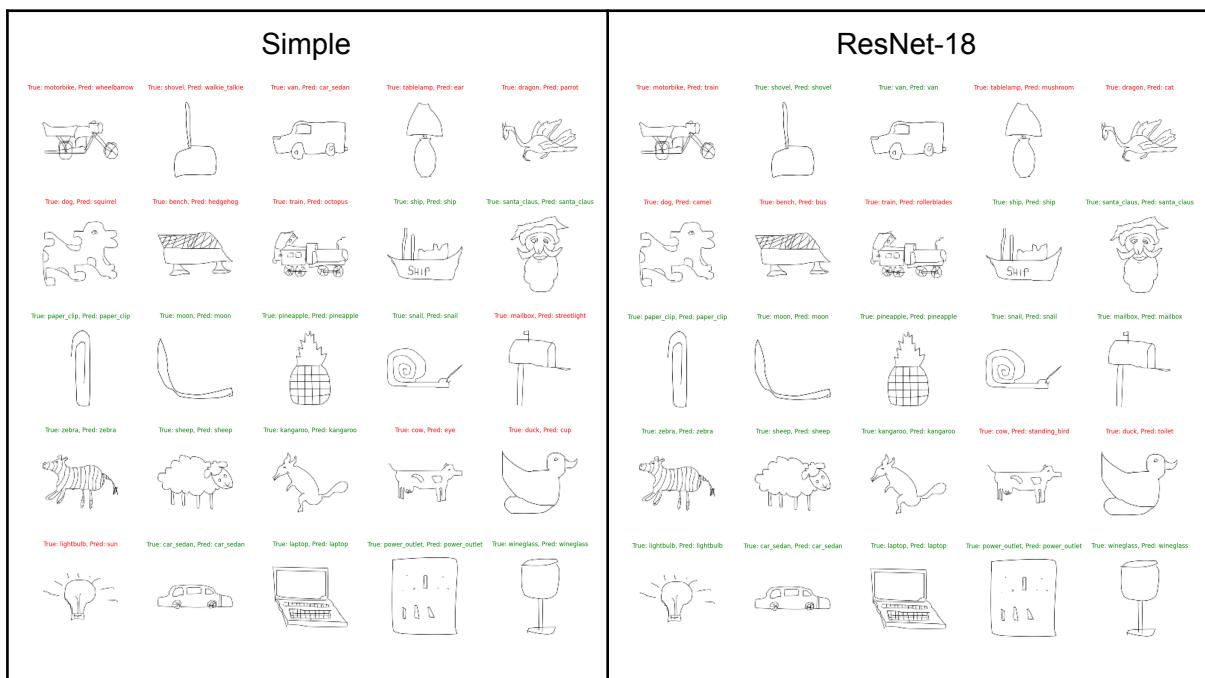


Tabla 1: Ejemplos de casos para ambos modelos

Fuente: Elaboración propia.

Casos Difíciles

Algunas clases fueron consistentemente difíciles de predecir para ambos modelos, como *cat*, *bush* y *scorpion*. Estos casos involucran clases con formas similares o aquellas que no tienen una característica distintiva con la cual el modelo las pueda identificar.

Casos Fáciles

Las clases con características únicas y distintivas fueron más fáciles de predecir para ambos modelos.

3.6 Recomendaciones

Para tareas más simples, el modelo Simple debería ser suficiente, ya que es más fácil de interpretar y más rápido de entrenar.

Para tareas complejas con alta variabilidad entre clases, usar un modelo más complejo como ResNet-18 probablemente daría mejores resultados.

Se podrían usar técnicas de aumento de datos para mejorar la capacidad del modelo para generalizar, especialmente para clases que son difíciles de predecir.

4. Conclusiones

El campo del reconocimiento visual ha avanzado significativamente con la introducción y evolución de las redes neuronales convolucionales. En este estudio, se exploraron dos arquitecturas distintas para abordar la tarea de clasificar 'sketches': un modelo Simple y ResNet-18.

A pesar de su simplicidad, el modelo Simple demostró ser competente en la tarea, lo que subraya la eficacia de las redes neuronales convolucionales incluso en su forma más básica. Sin embargo, ResNet-18 superó en rendimiento al modelo Simple, evidenciando las ventajas de una arquitectura más profunda y sofisticada, en particular para tareas de clasificación complejas.

Las matrices de confusión y las métricas de precisión revelaron áreas de fortaleza y debilidad para ambos modelos. Algunas clases fueron consistentemente difíciles de predecir, mientras que otras se clasificaron con alta precisión en ambos modelos.