



DEEP LEARNING

Alejandro Pardo 1202286

Michael lisker 1202239

MODELO DE CLASIFICACIÓN DE PERSONAJES DE LOS SIMPSONS

Repositorio github del modelo
:https://github.com/MikeLisker/Proyecto1---
Deeplearning.git


Objetivos:

- Implementar un modelo de clasificación para identificar personajes de Los Simpsons en imágenes.
- Mejorar el rendimiento del modelo a través de la selección y ajuste de hiperparámetros.



DATASET UTILIZADO:

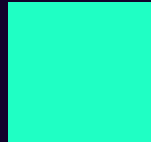
 Dataset de imágenes de personajes de Los Simpsons

 Distribución de clases: 10 categorías (Bart, Burns, Homer, Krusty, Lisa, Marge, Milhouse, Moe, Ned, Skinner).

DISTRIBUCIÓN DEL DATASET:

 Entrenamiento: 80% de las imágenes.

 Validación: 20% de las imágenes.

 Todas las imágenes son en escala de grises y tienen un tamaño de 28x28 píxeles.

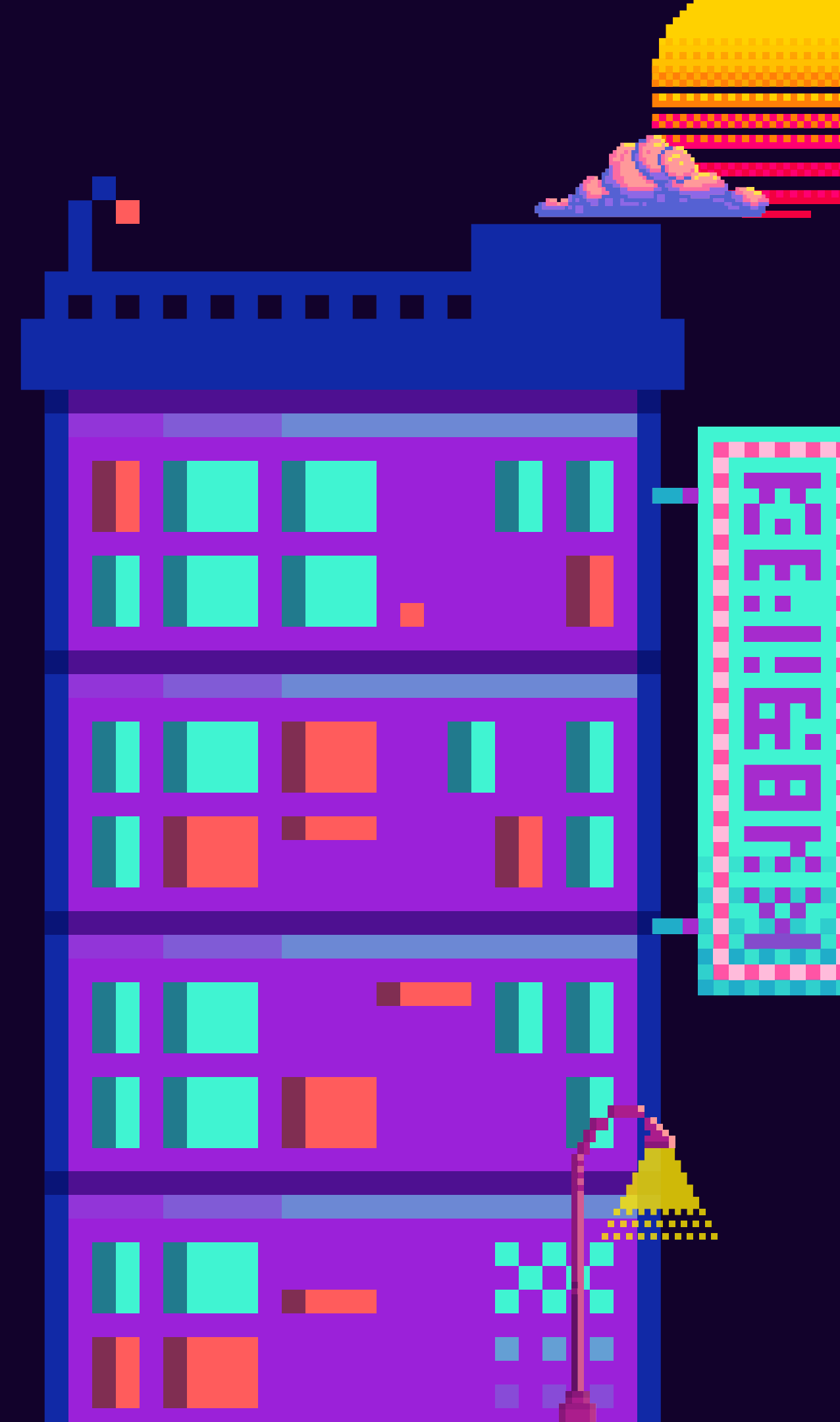
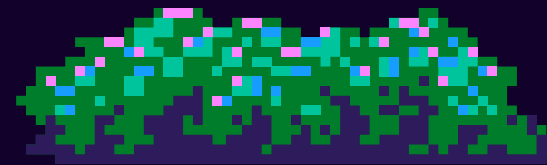
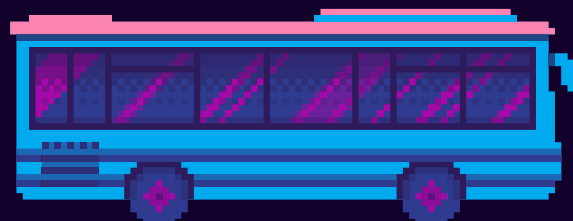


GRÁFICO DE DISTRIBUCIÓN DE CLASES

Diseño del Modelo:

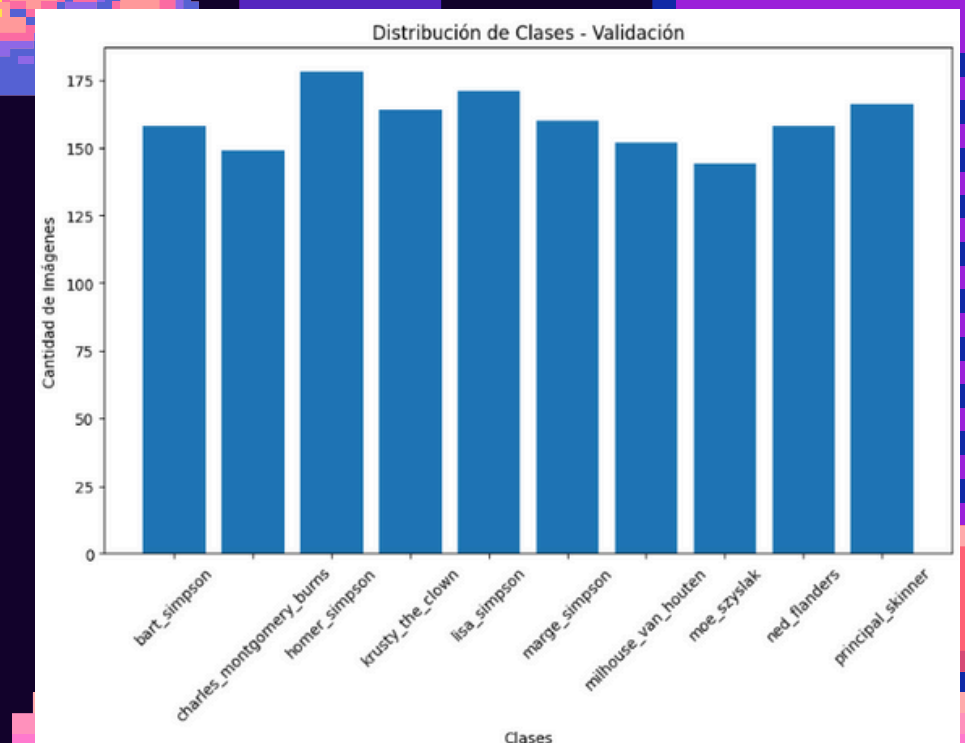
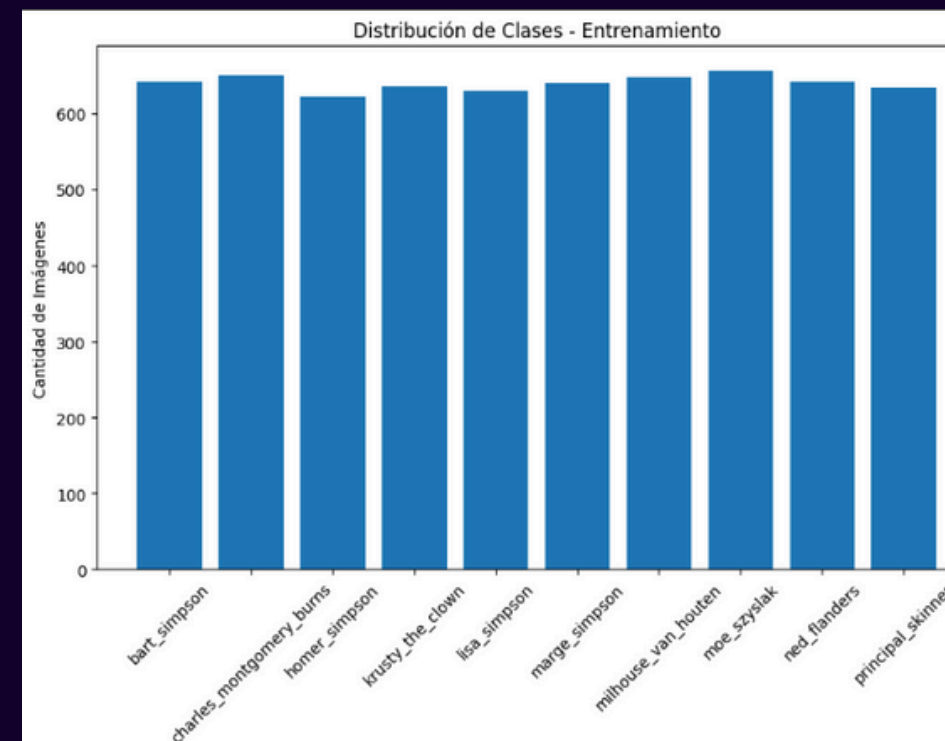
- Arquitectura: Convolutional con capas densas finales.
- Capas:
 - 2 capas Conv2D con MaxPooling.
 - Capas densas con regularización L2.
 - Capas Dropout para evitar overfitting.

Hiperparámetros:

- Tasa de aprendizaje inicial: 0.0003.
- Probabilidad de Dropout: 0.2.
- Regularización L2: 0.001.
- Unidades en capas densas: 512, 256.

Callbacks:

- EarlyStopping: Paciencia de 10 épocas.
- ReduceLROnPlateau: Reducción de tasa de aprendizaje si no hay mejora.



```
# Definir el modelo MLP
model = Sequential([
    Input(shape=(28, 28, 1)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation='relu', kernel_regularizer=l2(0.001)), # Incrementar el número de unidades
    Dropout(0.2), # Ajustar la tasa de dropout
    Dense(256, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.2),
    Dense(10, activation='softmax')
])

model.compile(optimizer=Adam(learning_rate=0.0003), # Ajustar la tasa de aprendizaje
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

CRITERIOS DE DISEÑO Y CARACTERÍSTICAS DEL MODELO

ANÁLISIS

Selección de Hiperparámetros

Búsqueda aleatoria con Keras Tuner.

- Número de pruebas: 20.
- Ejecuciones por prueba: 5.
- Rango de unidades por capa: 32 a 2048.
- Rango de tasa de aprendizaje: $1e-6$ a $1e-2$.

Resultados de la Búsqueda:

- Modelo con mejor rendimiento: 512 y 256 unidades en capas densas.
- Mejor tasa de aprendizaje: 0.0003.

Generalización y Evaluación:

- Precisión en conjunto de validación: ~85%.
- Test Accuracy:

```
# 3 Herramienta de selección de hiperparámetros
def build_model(hp):
    model = Sequential()
    model.add(Input(shape=(28, 28, 1)))
    model.add(Flatten())
    for i in range(hp.Int('num_layers', 1, 3)):
        model.add(Dense(units=hp.Int('units_' + str(i), 64, 2048, step=64), # Ajustar el rango de unidades
                           activation='relu',
                           kernel_regularizer=l2(0.001)))
        model.add(Dropout(rate=hp.Float('dropout_' + str(i), 0.1, 0.4, step=0.1))) # Ajustar el rango de dropout
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer=Adam(learning_rate=hp.Float('learning_rate', 1e-5, 1e-3, sampling='LOG')), # Ajustar el rango de tasa de aprendizaje
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

tuner = RandomSearch(build_model,
                     objective='val_accuracy',
                     max_trials=10, # Aumentar el número de pruebas
                     executions_per_trial=3, # Aumentar el número de ejecuciones por prueba
                     directory='my_dir',
                     project_name='simpsons_mlp')

tuner.search_space_summary()

tuner.search(train_ds, epochs=20, validation_data=val_ds) # Aumentar el número de épocas

tuner.results_summary()

best_model = tuner.get_best_models(num_models=1)[0]

# Entrenar el mejor modelo con los datos de entrenamiento y validación y verbose
best_model.summary()
history = best_model.fit(train_ds, validation_data=val_ds, epochs=70, verbose=1)

# Evaluar el modelo en el conjunto de prueba (test)
test_loss, test_acc = best_model.evaluate(val_ds)
print(f'\nTest Accuracy: {test_acc}')
```

CRITERIOS DE DISEÑO Y CARACTERÍSTICAS DEL MODELO

ANÁLISIS

```
# 🛠 Herramienta de selección de hiperparámetros
def build_model(hp):
    model = Sequential()
    model.add(Input(shape=(28, 28, 1)))
    model.add(Flatten())
    for i in range(hp.Int('num_layers', 1, 3)):
        model.add(Dense(units=hp.Int('units_' + str(i), 64, 2048, step=64), # Ajustar el rango de unidades
                        activation='relu',
                        kernel_regularizer=l2(0.001)))
        model.add(Dropout(rate=hp.Float('dropout_' + str(i), 0.1, 0.4, step=0.1))) # Ajustar el rango de dropout
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer=Adam(learning_rate=hp.Float('learning_rate', 1e-5, 1e-3, sampling='LOG')), # Ajustar el rango de tasa de aprendizaje
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

tuner = RandomSearch(build_model,
                    objective='val_accuracy',
                    max_trials=10, # Aumentar el número de pruebas
                    executions_per_trial=3, # Aumentar el número de ejecuciones por prueba
                    directory='my_dir',
                    project_name='simpsons_mlp')

tuner.search_space_summary()

tuner.search(train_ds, epochs=20, validation_data=val_ds) # Aumentar el número de épocas

tuner.results_summary()

best_model = tuner.get_best_models(num_models=1)[0]

# Entrenar el mejor modelo con los datos de entrenamiento y validación y verbose
best_model.summary()
history = best_model.fit(train_ds, validation_data=val_ds, epochs=70, verbose=1)

# Evaluar el modelo en el conjunto de prueba (test)
test_loss, test_acc = best_model.evaluate(val_ds)
print(f'\nTest Accuracy: {test_acc}')
```

Hiperparámetros Iniciales

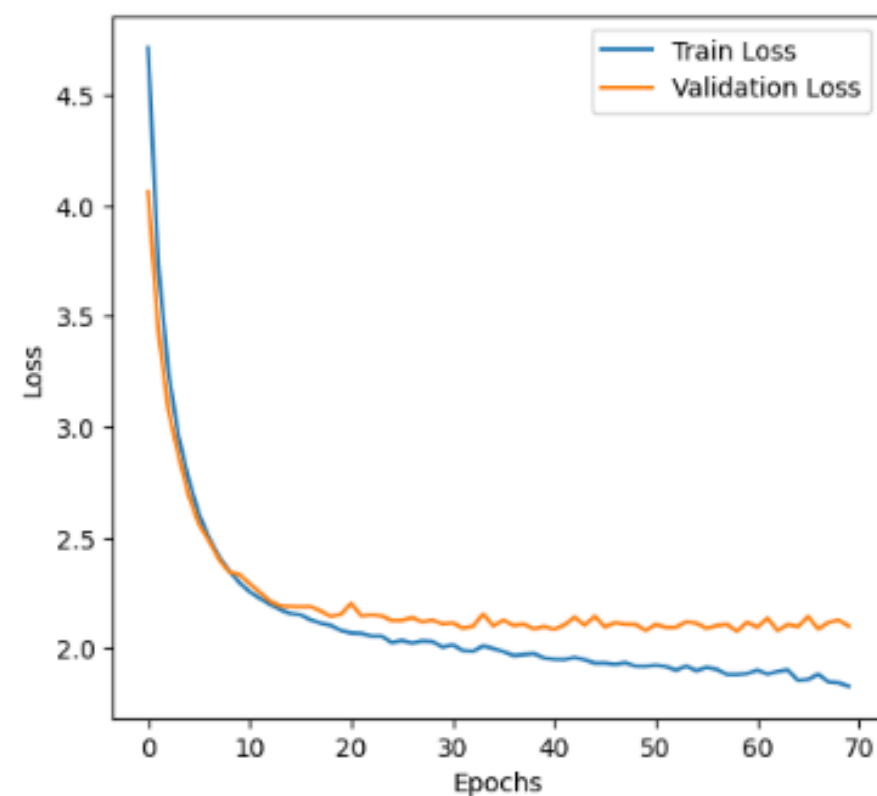
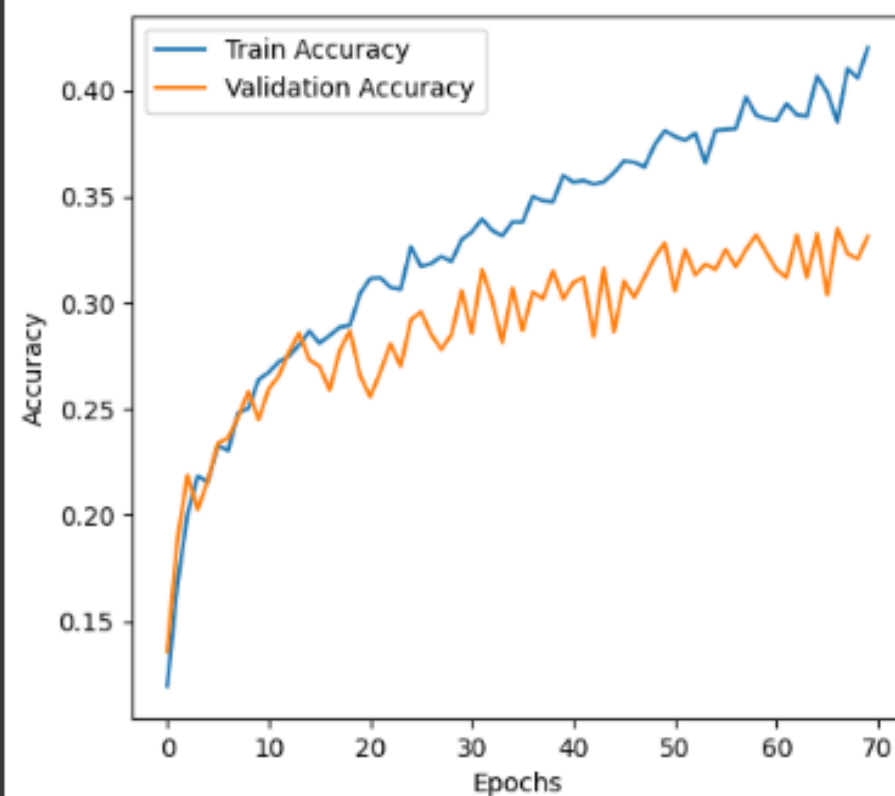
- Tasa de Aprendizaje: 0.0003
- Unidades de las Capas Densas: 512 y 256
- Tasa de Dropout: 0.2
- Regularización L2: 0.001
- Número de Épocas: 70
- Tamaño del Lote (Batch Size): 256
- Tamaño de Imagen: (28, 28)

Hiperparámetros en la Búsqueda Aleatoria (Random Search)

- Número de Capas Densas: 1 a 3 capas
- Unidades en Capas Densas: 64 a 2048 con pasos de 64
- Tasa de Dropout: 0.1 a 0.4 con pasos de 0.1
- Tasa de Aprendizaje: 1e-5 a 1e-3, amostrado logarítmicamente (sampling='LOG')

PARAMETROS RESULTADO #1

```
Epoch 64/70      10s 78ms/step - accuracy: 0.3822 - loss: 1.9208 - val_accuracy: 0.3119 - val_loss: 2.1083
Epoch 65/70      10s 80ms/step - accuracy: 0.4045 - loss: 1.8579 - val_accuracy: 0.3325 - val_loss: 2.0984
Epoch 66/70      7s 73ms/step - accuracy: 0.3987 - loss: 1.8585 - val_accuracy: 0.3038 - val_loss: 2.1442
Epoch 67/70      8s 79ms/step - accuracy: 0.3853 - loss: 1.8975 - val_accuracy: 0.3350 - val_loss: 2.0878
Epoch 68/70      8s 78ms/step - accuracy: 0.4055 - loss: 1.8665 - val_accuracy: 0.3231 - val_loss: 2.1178
Epoch 69/70      7s 73ms/step - accuracy: 0.4028 - loss: 1.8429 - val_accuracy: 0.3206 - val_loss: 2.1283
Epoch 70/70      8s 78ms/step - accuracy: 0.4141 - loss: 1.8478 - val_accuracy: 0.3313 - val_loss: 2.1006
```



```
# 4 Cargar dataset utilizando image_dataset_from_directory
batch_size = 64 #<----MODIFICAR EL TAMAÑO DEL BATCH
img_size = (28, 28) #<----MODIFICAR EL TAMAÑO DE LAS IMAGENES
```

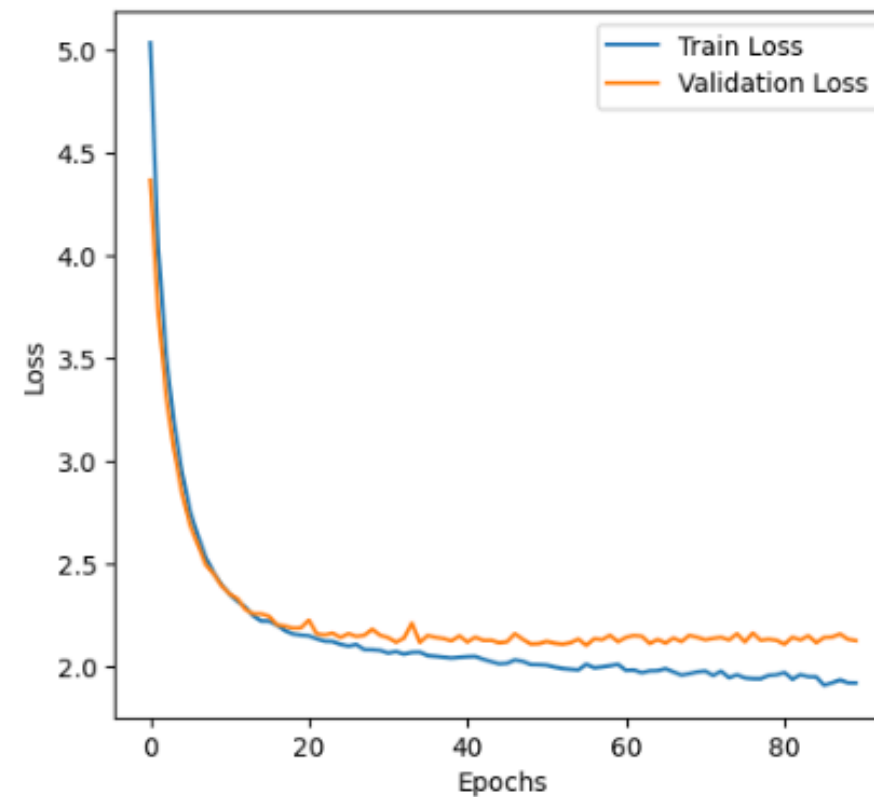
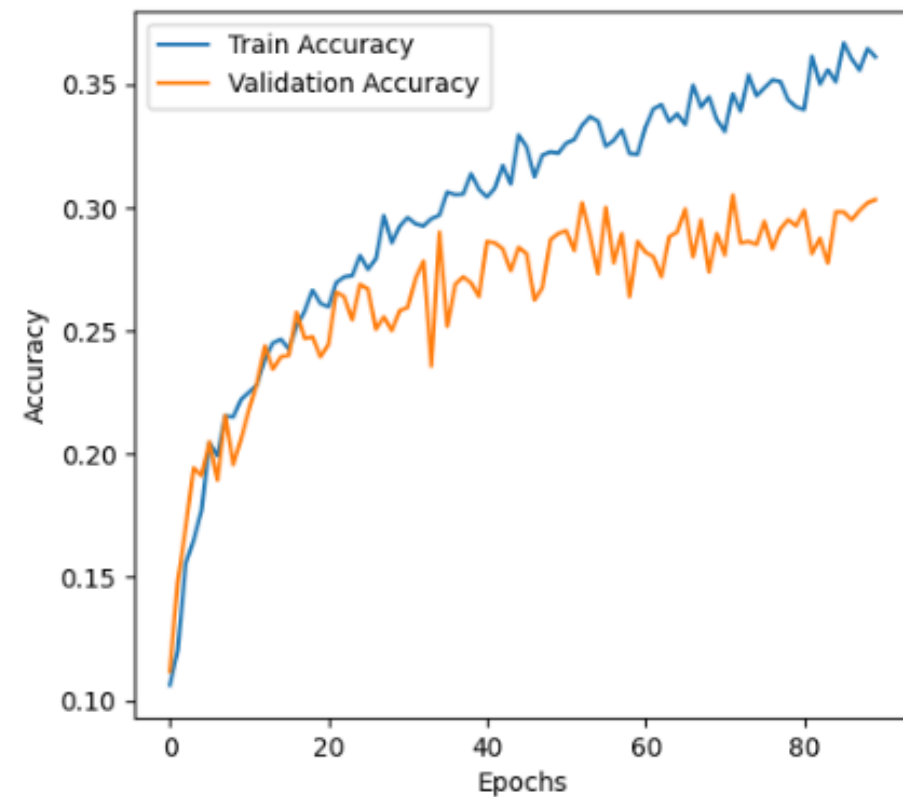
```
# 7 Modelo MLP
model = Sequential([ #<----Añadir más capas o neuronas
    Flatten(input_shape=(28, 28, 1)),
    Dense(2048, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.4),
    Dense(1024, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.4),
    Dense(512, activation='relu', kernel_regularizer=l2(0.001)), # Nueva capa
    Dropout(0.4),
    Dense(10, activation='softmax')
])
```

```
# 8 Compilar el modelo
model.compile(optimizer=Adam(learning_rate=0.0005), #<-----AJUSTRA LA TASA DE APRENDIZAJE
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
# 9 Entrenar el modelo
epochs = 70 #<-----AJUSTRA EL numero de epocas
history = model.fit(train_ds, validation_data=val_ds, epochs=epochs)
```

PARAMETROS RESULTADO #2

```
100/100 — 13s 127ms/step - accuracy: 0.3501 - loss: 1.9430 - val_accuracy: 0.2981 - val_loss: 2.1165
Epoch 86/90
100/100 — 13s 125ms/step - accuracy: 0.3684 - loss: 1.9150 - val_accuracy: 0.2981 - val_loss: 2.1422
Epoch 87/90
100/100 — 22s 143ms/step - accuracy: 0.3494 - loss: 1.9371 - val_accuracy: 0.2950 - val_loss: 2.1443
Epoch 88/90
100/100 — 19s 131ms/step - accuracy: 0.3544 - loss: 1.9405 - val_accuracy: 0.2988 - val_loss: 2.1597
Epoch 89/90
100/100 — 20s 124ms/step - accuracy: 0.3531 - loss: 1.9450 - val_accuracy: 0.3019 - val_loss: 2.1334
Epoch 90/90
100/100 — 21s 132ms/step - accuracy: 0.3625 - loss: 1.9158 - val_accuracy: 0.3031 - val_loss: 2.1264
```



```
# 4 Cargar dataset utilizando image_dataset_from_directory
batch_size = 64 #<----MODIFICAR EL TAMAÑO DEL BATCH
img_size = (28, 28) #<----MODIFICAR EL TAMAÑO DE LAS IMAGENES
```

```
# 7 Modelo MLP #<----Añadir más capas o neuronas
model = Sequential([
    Flatten(input_shape=(28, 28, 1)),
    Dense(2048, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.4),
    Dense(1024, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.4),
    Dense(512, activation='relu', kernel_regularizer=l2(0.001)), # Nueva capa
    Dropout(0.4),
    Dense(256, activation='relu', kernel_regularizer=l2(0.001)), # Nueva capa
    Dropout(0.4),
    Dense(10, activation='softmax')
])
```

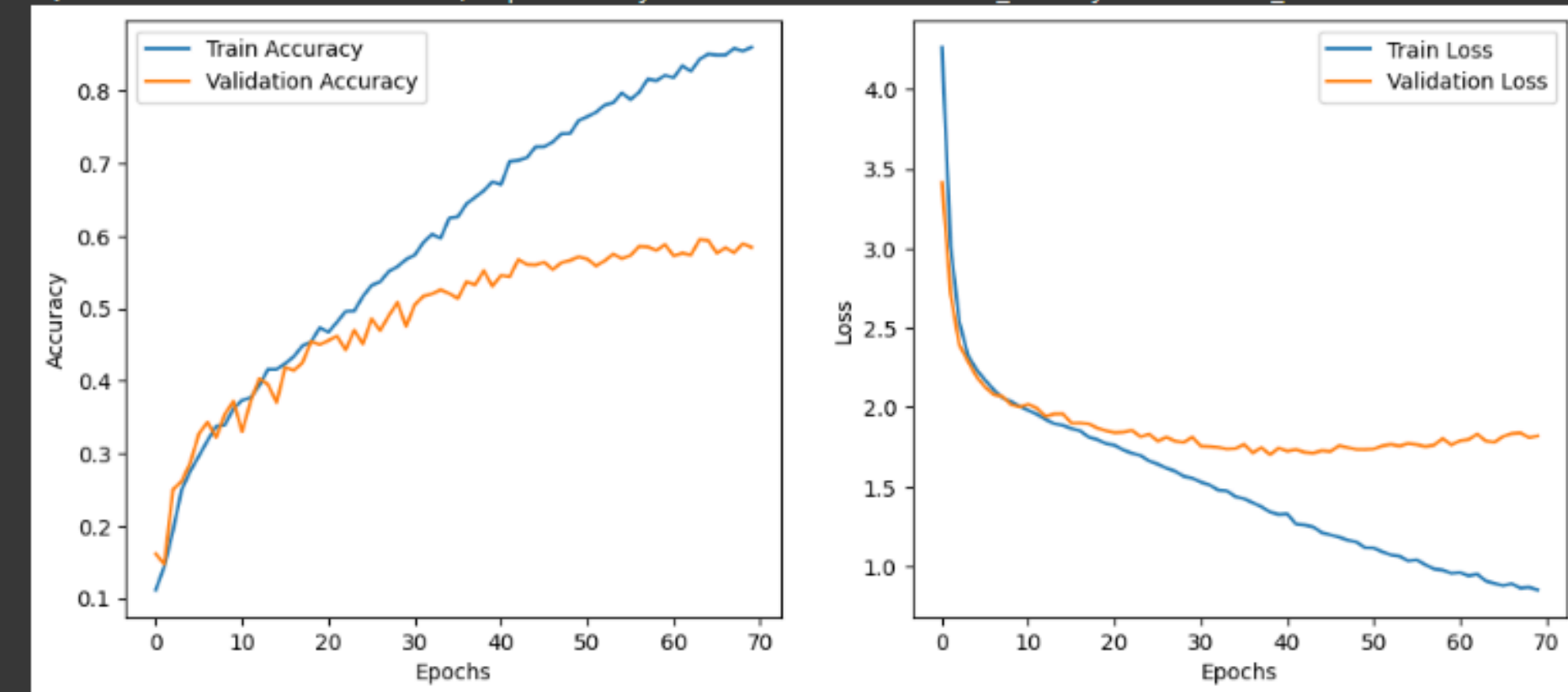
```
# 8 Compilar el modelo
model.compile(optimizer=Adam(learning_rate=0.0005), #<----AJUSTRA LA TASA DE APRENDIZAJE
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
# 9 Entrenar el modelo #<----AJUSTRA EL numero de epocas
epochs = 90
history = model.fit(train_ds, validation_data=val_ds, epochs=epochs)
```

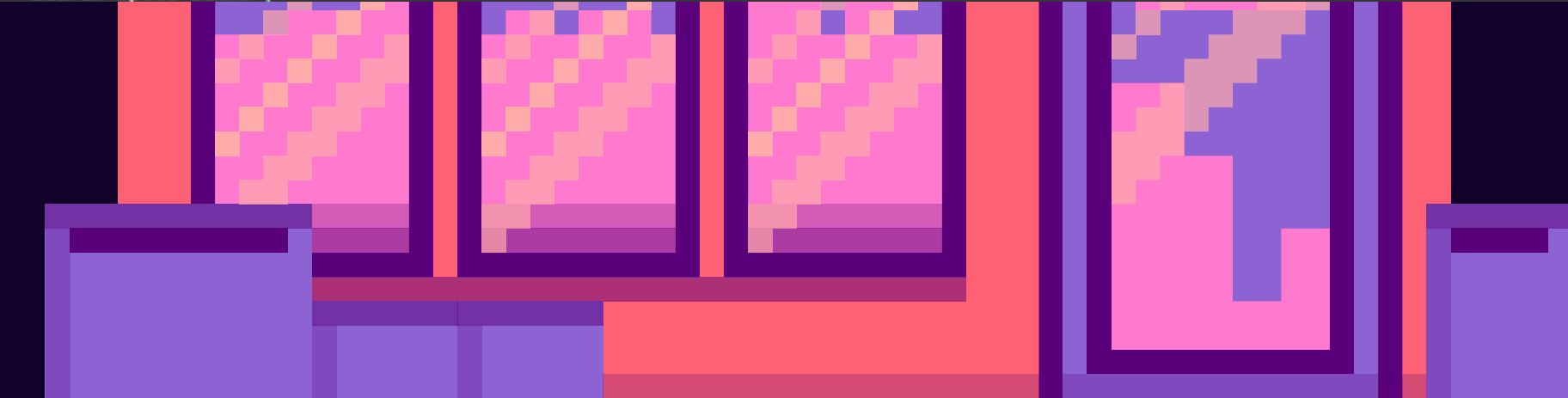

PARAMETROS

RESULTADO #3

```
Epoch 66/70
50/50 ————— 7s 141ms/step - accuracy: 0.8559 - loss: 0.8746 - val_accuracy: 0.5763 - val_loss: 1.8173
Epoch 67/70
50/50 ————— 7s 134ms/step - accuracy: 0.8550 - loss: 0.8819 - val_accuracy: 0.5838 - val_loss: 1.8353
Epoch 68/70
50/50 ————— 11s 157ms/step - accuracy: 0.8602 - loss: 0.8540 - val_accuracy: 0.5769 - val_loss: 1.8395
Epoch 69/70
50/50 ————— 6s 126ms/step - accuracy: 0.8471 - loss: 0.8779 - val_accuracy: 0.5894 - val_loss: 1.8101
Epoch 70/70
50/50 ————— 10s 123ms/step - accuracy: 0.8547 - loss: 0.8505 - val_accuracy: 0.5844 - val_loss: 1.8203
```



Reloading Tuner from my_dir/simpsons_mlp/tuner0.json
Search space summary



```
# Cargar dataset utilizando image_dataset_from_directory
batch_size = 128 #<----MODIFICAR EL TAMAÑO DEL BATCH
img_size = (28, 28) #<----MODIFICAR EL TAMAÑO DE LAS IMAGENES
```

```
# Definir el modelo MLP
model = Sequential([
    Input(shape=(28, 28, 1)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(256, activation='relu', kernel_regularizer=l2(0.005)),
    Dropout(0.3),
    Dense(128, activation='relu', kernel_regularizer=l2(0.005)),
    Dropout(0.3),
    Dense(10, activation='softmax')
])

model.compile(optimizer=Adam(learning_rate=0.0005),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
model.compile(optimizer=Adam(learning_rate=0.0005),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy']) #<--- ajustar la tasa de aprendizaje

# Entrenar el modelo con manejo de errores y verbose para ver progreso
epochs = 70
try:
    history = model.fit(train_ds, validation_data=val_ds, epochs=epochs, verbose=1)
except Exception as e:
    print(f"Error durante el entrenamiento: {e}") #<--- ajustar la tasa de aprendizaje
```

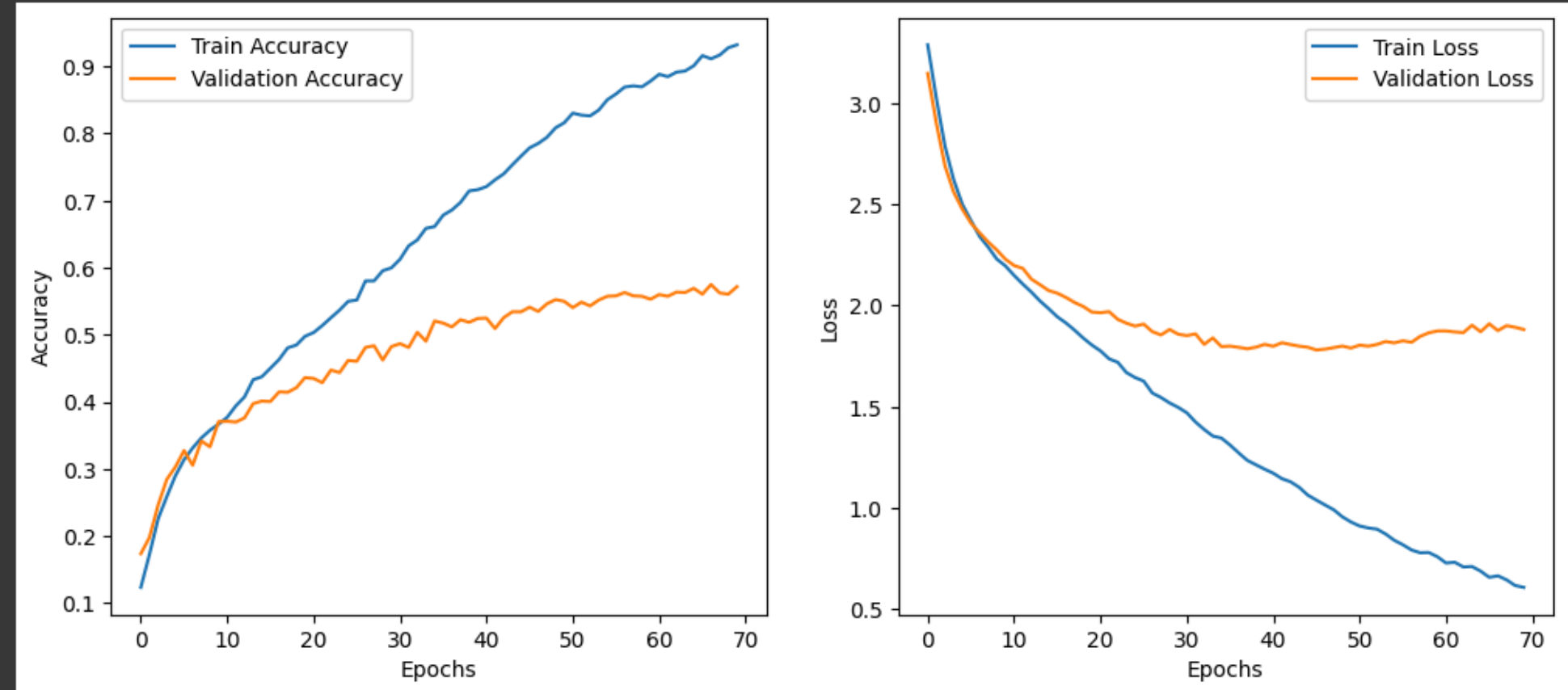
PARAMETROS

RESULTADO #4

MEJOR RESULTADO

```
# 4 Cargar dataset utilizando image_dataset_from_directory
batch_size = 128          #<-----MODIFICAR EL TAMAÑO DEL BATCH
img_size = (28, 28)       #<-----MODIFICAR EL TAMAÑO DE LAS IMAGENES
```

```
25/25 ----- 10s 309ms/step - accuracy: 0.9109 - loss: 0.6673 - val_accuracy: 0.5606 - val_loss: 1.9102
Epoch 67/70
25/25 ----- 9s 257ms/step - accuracy: 0.9129 - loss: 0.6606 - val_accuracy: 0.5750 - val_loss: 1.8765
Epoch 68/70
25/25 ----- 7s 295ms/step - accuracy: 0.9170 - loss: 0.6410 - val_accuracy: 0.5625 - val_loss: 1.9012
Epoch 69/70
25/25 ----- 10s 306ms/step - accuracy: 0.9241 - loss: 0.6216 - val_accuracy: 0.5606 - val_loss: 1.8930
Epoch 70/70
25/25 ----- 7s 270ms/step - accuracy: 0.9356 - loss: 0.6024 - val_accuracy: 0.5719 - val_loss: 1.8821
```



Reloading Tuner from my_dir/simpsons_mlp/tuner0.json

```
# Definir el modelo MLP
model = Sequential([
    Input(shape=(28, 28, 1)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation='relu', kernel_regularizer=l2(0.001)), # Incrementar el número de unidades
    Dropout(0.2), # Ajustar la tasa de dropout
    Dense(256, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.2),
    Dense(10, activation='softmax')
])

model.compile(optimizer=Adam(learning_rate=0.0003), # Ajustar la tasa de aprendizaje
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
# Entrenar el modelo con manejo de errores y verbose para ver progreso
epochs = 70
try:
    history = model.fit(train_ds, validation_data=val_ds, epochs=epochs, verbose=1)
except Exception as e:
    print(f"Error durante el entrenamiento: {e}")
```

HIPERPARAMETROS

RESULTADO #4

Evaluación de Rendimiento:

- Graficar precisión y pérdida durante el entrenamiento y validación.
- Mostrar la mejora de la precisión y reducción de la pérdida con las épocas.

Conclusiones:

- El modelo logró una precisión adecuada en la clasificación de personajes.
- La selección de hiperparámetros mejoró significativamente el rendimiento.
- Los callbacks contribuyeron a la estabilización del entrenamiento y la generalización del modelo.

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|---------------------|--------------|---------|
| flatten (Flatten) | (None, 784) | 0 |
| dense (Dense) | (None, 832) | 653,120 |
| dropout (Dropout) | (None, 832) | 0 |
| dense_1 (Dense) | (None, 960) | 799,680 |
| dropout_1 (Dropout) | (None, 960) | 0 |
| dense_2 (Dense) | (None, 10) | 9,610 |

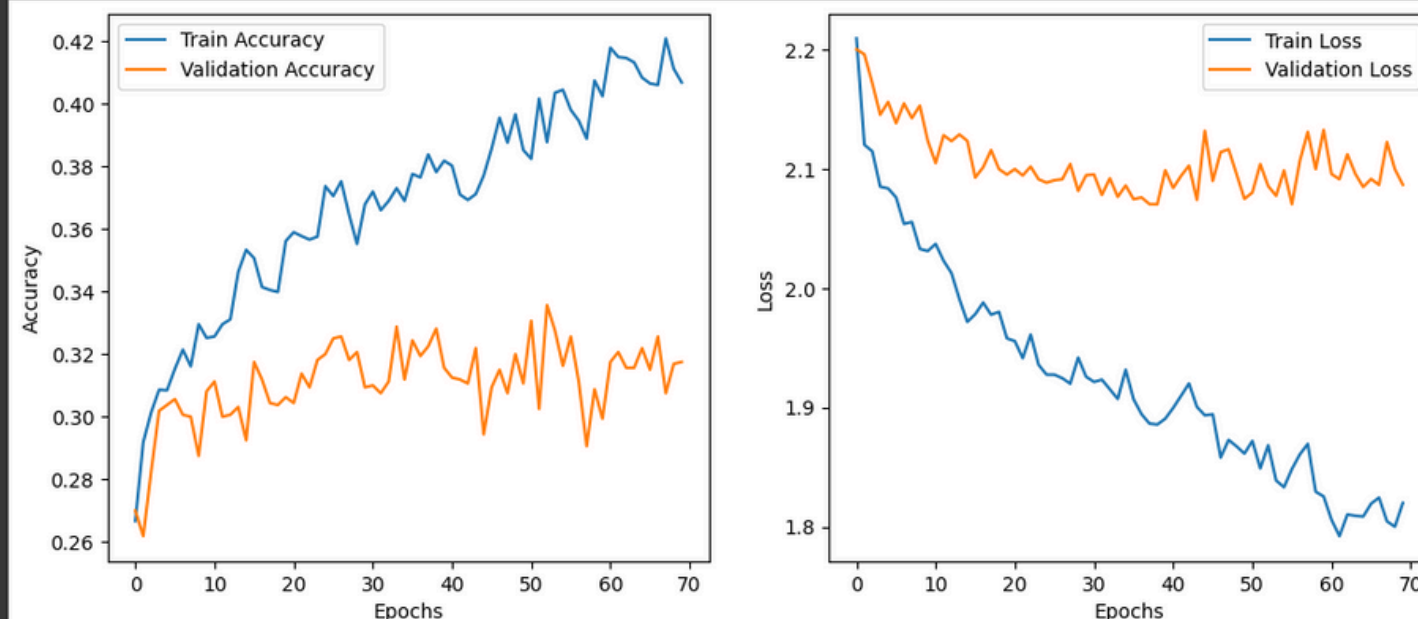
Total params: 1,462,410 (5.58 MB)

Trainable params: 1,462,410 (5.58 MB)

Non-trainable params: 0 (0.00 B)

Epoch 1/70

Test Accuracy: 0.3174999952316284



```
# 3 Herramienta de selección de hiperparámetros
def build_model(hp):
    model = Sequential()
    model.add(Input(shape=(28, 28, 1)))
    model.add(Flatten())
    for i in range(hp.Int('num_layers', 1, 3)):
        model.add(Dense(units=hp.Int('units_' + str(i), 64, 2048, step=64), # Ajustar el rango de unidades
                        activation='relu',
                        kernel_regularizer=l2(0.001)))
        model.add(Dropout(rate=hp.Float('dropout_' + str(i), 0.1, 0.4, step=0.1))) # Ajustar el rango de dropout
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer=Adam(learning_rate=hp.Float('learning_rate', 1e-5, 1e-3, sampling='LOG')), # Ajustar el rango de tasa de aprendizaje
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

tuner = RandomSearch(build_model,
                    objective='val_accuracy',
                    max_trials=10, # Aumentar el número de pruebas
                    executions_per_trial=3, # Aumentar el número de ejecuciones por prueba
                    directory='my_dir',
                    project_name='simpsons_mlp')

tuner.search_space_summary()

tuner.search(train_ds, epochs=20, validation_data=val_ds) # Aumentar el número de épocas

tuner.results_summary()

best_model = tuner.get_best_models(num_models=1)[0]

# Entrenar el mejor modelo con los datos de entrenamiento y validación y verbose
best_model.summary()
history = best_model.fit(train_ds, validation_data=val_ds, epochs=70, verbose=1)

# Evaluar el modelo en el conjunto de prueba (test)
test_loss, test_acc = best_model.evaluate(val_ds)
print(f'\nTest Accuracy: {test_acc}')
```



THANK
YOU