Proyecto IV - Visión

Miroslavna Kovylina Zabyako

Índice

Proyecto IV - Visión 1

Arquitectura 1

```
# Arquitectura 1
model = models.Sequential()

# Capas de captar patrones
# Primera capa convolucional con Batch Normalization
model.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu', input_shape=(32, 32, 3),kernel_initializer=initializers.HeNormal()))
model.add(layers.BatchNormalization())

# Segunda capa convolucional con Batch Normalization, MaxPool y Dropout
model.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu',kernel_initializer=initializers.HeNormal()))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.3))

# Tercera capa convolucional con Batch Normalization
model.add(layers.Conv2D(128, (3, 3),  padding='same', activation='relu',kernel_initializer=initializers.HeNormal()))
model.add(layers.BatchNormalization())

# Cuarta capa convolucional con Batch Normalization, MaxPool y Dropout
model.add(layers.Conv2D(128, (3, 3),  padding='same', activation='relu',kernel_initializer=initializers.HeNormal()))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.3))

# Aplanar y pasar a capas densas
model.add(ks.layers.Flatten())

# Capas de clasificación
# Primera capa
model.add(layers.Dense(256,  activation='relu',kernel_initializer=initializers.HeNormal()))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.4))
# Segunda capa
model.add(layers.Dense(128, activation='relu', kernel_initializer=initializers.HeNormal()))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.4))

# Capa de salida
model.add(layers.Dense(10, activation='softmax'))
```

Resumen de la Arquitectura 1

La arquitectura fue diseñada para mejorar la Accuracy del modelo básico, con dos objetivos principales: aumentar la capacidad para captar patrones y mejorar la capacidad de clasificación:

1. Incremento en la capacidad para captar patrones

Se busca captar patrones desde lo más básico hasta lo más avanzado a través de un diseño de **4 bloques convolucionales**, en lugar de un solo bloque como en el modelo básico:

Primer bloque:

- Primera capa convolucional con **64 filtros** (más parámetros que en un modelo básico), lo que permite capturar más características desde el inicio.

Segundo bloque:

- Segunda capa convolucional con 64 filtros.
- Introduce MaxPooling (2,2) para reducir las dimensiones de la imagen a la mitad, capturando patrones clave y reduciendo el costo computacional. Se añade un Dropout (0.3), que desactiva aleatoriamente el 30% de las neuronas para prevenir el sobreajuste.

Tercer bloque:

- Incrementa la complejidad del modelo utilizando 128 filtros, permitiendo capturar características más avanzadas y detalladas.

Cuarto bloque:

- También incluye 128 filtros para reforzar la captura de patrones complejos.
- Se integra MaxPooling para reducir nuevamente las dimensiones de la imagen y un Dropout (0.3) para evitar el sobreajuste.

BatchNormalization en cada bloque: Esto estabiliza el aprendizaje, acelera la convergencia y evita problemas de gradientes desvanecientes (6 capas en total que está cerca de 7-8 capas cuando BatchNormalization empieza a ser eficaz).

2. Mejora en la capacidad de clasificación

Para mejorar el desempeño en la etapa de clasificación, se incrementa el número de capas densas:

Se añaden 2 capas densas (en lugar de 1) para mejorar la capacidad de aprendizaje y clasificación de características complejas.

Primera capa densa:

- 256 neuronas, lo que aumenta significativamente la capacidad de aprendizaje.
- BatchNormalization estabiliza el aprendizaje y mitiga fluctuaciones y Dropout (0.4) desactiva el 40% de las neuronas para reducir el riesgo de sobreajuste.

Segunda capa densa:

- 128 neuronas para un refinamiento adicional de las características.
- También utiliza BatchNormalization y Dropout (0.4).

Proyecto IV - Visión 1

Arquitectura 1

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 64)	1,792
batch_normalization (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_1 (Conv2D)	(None, 32, 32, 64)	36,928
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d (MaxPooling2D)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 128)	512
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147,584
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 128)	512
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_1 (Dropout)	(None, 8, 8, 128)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 256)	2,097,408
batch_normalization_4 (BatchNormalization)	(None, 256)	1,024
dropout_2 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32,896
batch_normalization_5 (BatchNormalization)	(None, 128)	512
dropout_3 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1,290

Total params: 2,394,826 (9.14 MB)

Trainable params: 2,393,290 (9.13 MB)

Non-trainable params: 1,536 (6.00 KB)

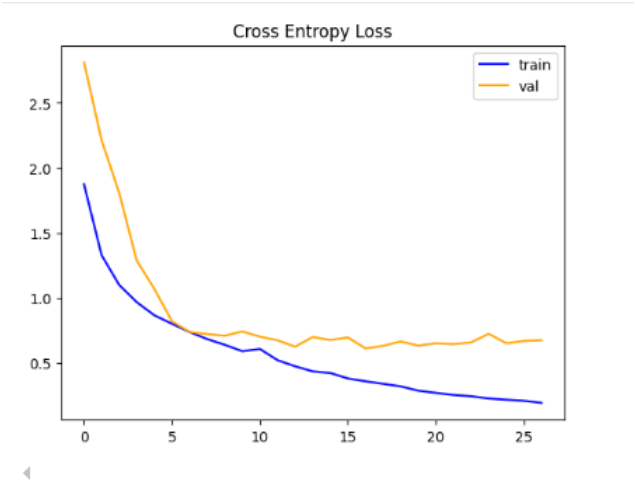
El modelo cuenta con **2,394,826 parámetros**, de los cuales la mayoría son entrenables y están orientados al aprendizaje de características. El modelo tiene una cantidad de parámetros **considerable pero no exagerada**, adecuada para la tarea de clasificación de imágenes en CIFAR-10.

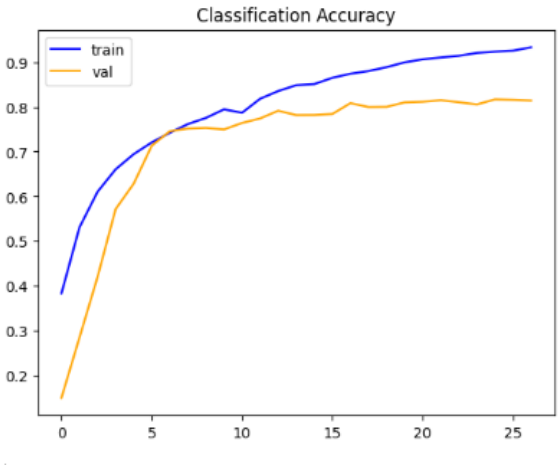
```
# Optimizador, función de error:
model.compile(optimizer='Adam',loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# Callback para early stopping
callback_val_loss=EarlyStopping(monitor="val_loss",patience=10,restore_best_weights=True)
callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10, restore_best_weights=True)
# Epochs
epochs=200
# Model fit
history = model.fit(x_train_scaled, y_train, epochs=epochs,
 batch_size= 512,callbacks=[callback_val_loss, callback_val_accuracy],validation_data=(x_val_scaled, y_val))

# Results
_, acc = model.evaluate(x_test_scaled, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
> 80.530
```

Se consigue el objetivo de alcanzar una Accuracy superior al 80% únicamente con la arquitectura.

Proyecto IV - Visión 1

Arquitectura 1





Comportamiento del entrenamiento: La Accuracy mejora constantemente a lo largo de las épocas, alcanzando aproximadamente 95% al final del entrenamiento (que se da por early stopping).

Comportamiento de la validación: Al principio crece rápidamente, pero luego la mejora se estabiliza y fluctúa ligeramente alrededor de 81%, indicando que el modelo ya no mejora significativamente en términos de generalización.

Ambas curvas muestran una rápida convergencia inicial, indicando un aprendizaje efectivo en las primeras épocas.

Diferencia entre entrenamiento y validación: Sugiere sobreajuste, ya que el modelo aprende mejor los datos del entrenamiento que los del conjunto de validación.

Estabilidad de validación: Después de la época 10, la Accuracy en validación se estabiliza, lo que sugiere que el modelo alcanza su límite en términos de generalización con esta arquitectura y configuración.

Conclusión

Las gráficas indican que el modelo logra un desempeño aceptable tanto en el entrenamiento como en la validación, con una Accuracy en validación que supera el **80%**. Sin embargo, la brecha entre entrenamiento y validación sugiere que podría ser útil:

1. Ajustar la regularización para abordar el problema de sobreajuste.
2. Aplicar aumentación de datos para mejorar la generalización o considerar el uso de transfer learning.

Proyecto IV - Visión 2

Arquitectura 1 mejorada

```
# Arquitectura 1 mejorada

model = models.Sequential()

# Capas de captar patrones

# Primera capa convolucional con Batch Normalization, Dropout

model.add(layers.Conv2D(64, (3, 3), padding='same', input_shape=(32, 32,
3),kernel_initializer=initializers.HeNormal(),kernel_regularizer=regularizers.l2(1e-4)))

model.add(layers.BatchNormalization())

model.add(layers.ReLU())

model.add(layers.Dropout(0.3))

# Segunda capa convolucional con Batch Normalization, MaxPool y Dropout

model.add(layers.Conv2D(64, (3, 3), padding='same',
kernel_initializer=initializers.HeNormal(),kernel_regularizer=regularizers.l2(1e-4)))

model.add(layers.BatchNormalization())

model.add(layers.ReLU())

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Dropout(0.3))

# Tercera capa convolucional con Batch Normalization

model.add(layers.Conv2D(128, (3, 3),  padding='same',
kernel_initializer=initializers.HeNormal(),kernel_regularizer=regularizers.l2(1e-4)))

model.add(layers.BatchNormalization())

model.add(layers.ReLU())

# Cuarta capa convolucional con Batch Normalization, MaxPool y Dropout

model.add(layers.Conv2D(128, (3, 3),  padding='same',
kernel_initializer=initializers.HeNormal(),kernel_regularizer=regularizers.l2(1e-4)))

model.add(layers.BatchNormalization())

model.add(layers.ReLU())

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Dropout(0.4))

# Aplanar y pasar a capas densas

model.add(ks.layers.Flatten())


# Capas de clasificación

# Primera capa

model.add(layers.Dense(256,
kernel_initializer=initializers.HeNormal(),kernel_regularizer=regularizers.l2(1e-4)))

model.add(layers.BatchNormalization())

model.add(layers.ReLU())

model.add(layers.Dropout(0.5))

# Segunda capa

model.add(layers.Dense(128,
kernel_initializer=initializers.HeNormal(),kernel_regularizer=regularizers.l2(1e-4)))

model.add(layers.BatchNormalization())

model.add(layers.ReLU())

model.add(layers.Dropout(0.5))

# Capa de salida

model.add(layers.Dense(10, activation='softmax'))
```

Proyecto IV - Visión 2

Arquitectura 1 mejorada

Resumen de mejoras en la Arquitectura 1:

1. **Regularización:**
   Se introduce kernel_regularizer=regularizers.l2(1e-4) en todas las capas convolucionales y densas.
   Impacto: Esta regularización ayuda a reducir el sobreajuste penalizando los pesos grandes durante el entrenamiento, lo que mejora la capacidad de generalización del modelo.

2. **Dropout:**
   Capas convolucionales: Se añade Dropout en la primera capa convolucional, mejorando la robustez del modelo desde el inicio y reduciendo el sobreajuste en etapas tempranas. El valor de Dropout se incrementa a 0.4 en la cuarta capa convolucional, reforzando la regularización en las capas más profundas.
   Capas densas: El Dropout se incrementa a 0.5, reduciendo aún más el sobreajuste en la etapa final del modelo.

3. **Cambio en la ubicación de la activación ReLU:**
   En la versión mejorada, la activación ReLU se mueve de ser implícita dentro de las capas a convertirse en una capa separada aplicada después de BatchNormalization.
   Justificación técnica: Colocar BatchNormalization antes de la activación ReLU tiene sentido desde el punto de vista matemático. BatchNormalization ajusta las activaciones de la capa previa para que tengan una distribución con media cercana a 0 y desviación estándar de 1. Aplicar ReLU después introduce la no linealidad en los datos ya normalizados, asegurando que la normalización se aplique de forma óptima sin descartar activaciones negativas prematuramente.
   Impacto: Este cambio mejora la estabilidad del modelo, especialmente frente al ruido introducido por los valores de Dropout aumentados. Además, puede acelerar la convergencia, ya que las operaciones matemáticas se realizan en el orden más adecuado.

Proyecto IV - Visión 2

Arquitectura 1 mejorada

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 64)	1,792
batch_normalization (BatchNormalization)	(None, 32, 32, 64)	256
re_lu (ReLU)	(None, 32, 32, 64)	0
dropout (Dropout)	(None, 32, 32, 64)	0
conv2d_1 (Conv2D)	(None, 32, 32, 64)	36,928
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 64)	256
re_lu_1 (ReLU)	(None, 32, 32, 64)	0
max_pooling2d (MaxPooling2D)	(None, 16, 16, 64)	0
dropout_1 (Dropout)	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 128)	512
re_lu_2 (ReLU)	(None, 16, 16, 128)	0
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147,584
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 128)	512
re_lu_3 (ReLU)	(None, 16, 16, 128)	0
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_2 (Dropout)	(None, 8, 8, 128)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 256)	2,097,408
batch_normalization_4 (BatchNormalization)	(None, 256)	1,024
re_lu_4 (ReLU)	(None, 256)	0
dropout_3 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32,896
batch_normalization_5 (BatchNormalization)	(None, 128)	512
re_lu_5 (ReLU)	(None, 128)	0
dropout_4 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1,290

Total params: 2,394,826 (9.14 MB)

Trainable params: 2,393,290 (9.13 MB)

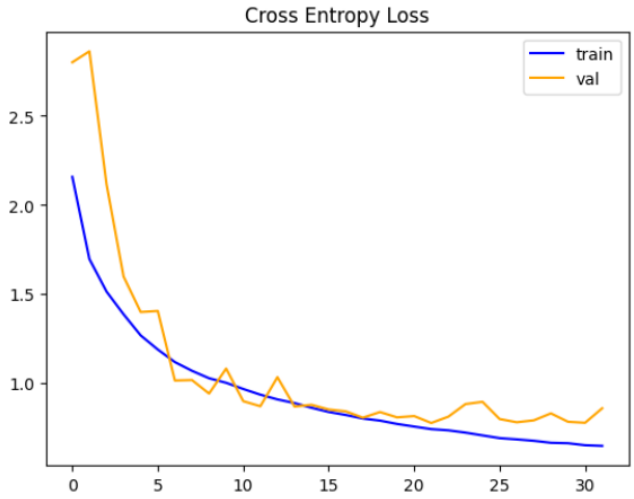
Non-trainable params: 1,536 (6.00 KB)

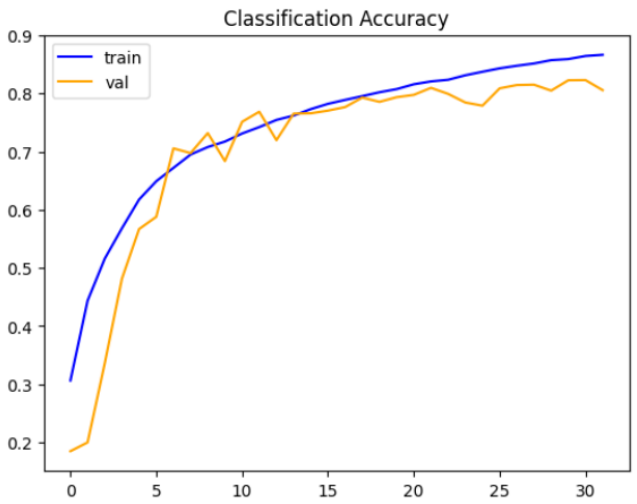
```
# Optimizador, función de error:
model.compile(optimizer='Adam',loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# Callback para early stopping
callback_val_loss=EarlyStopping(monitor="val_loss",patience=10,restore_best_weights=True)
callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10, restore_best_weights=True)
# Epochs
epochs=200
# Model fit
history = model.fit(x_train_scaled, y_train, epochs=epochs,
 batch_size= 512,callbacks=[callback_val_loss, callback_val_accuracy],validation_data=(x_val_scaled, y_val))

# Results
_, acc = model.evaluate(x_test_scaled, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
> 81.890
```

Se consigue el objetivo de alcanzar una Accuracy superior al 80% únicamente con la arquitectura.

Proyecto IV - Visión 2

Arquitectura 1 mejorada





Comparación de las gráficas: Arquitectura 1 original vs. mejorada

Comportamiento del entrenamiento:
En la arquitectura mejorada, la Accuracy de entrenamiento es más controlada (la curva es más suave), alcanzando aproximadamente el **90%** después de 30 épocas.
Esto sugiere que el aprendizaje es más estable y robusto gracias al cambio de posición de **ReLU** con respecto a **BatchNormalization** y a las técnicas de regularización adicionales, como el aumento de **Dropout** y la implementación de **L2**.

Comportamiento de la validación:
La curva de validación muestra un crecimiento continuo y no se estanca al inicio, lo que refleja una mejor capacidad de generalización.

Convergencia:
La convergencia inicial es similar en ambas arquitecturas. Sin embargo, en la versión mejorada, la precisión en validación continúa mejorando de manera más consistente a lo largo de las épocas. Esto indica que las mejoras en la arquitectura, como la regularización y el ajuste de Dropout, favorecen un aprendizaje más efectivo al reducir la memorización y priorizar la generalización (mejor control del sobreajuste).

Conclusiones

1. **Generalización:**
   La arquitectura mejorada muestra una mejor capacidad de generalización, reduciendo la brecha entre entrenamiento y validación (del 14% en la original al 5% en la mejorada).
2. **Regularización efectiva:**
   Las técnicas de regularización añadidas (L2, Dropout aumentado y ajustes en ReLU y BatchNormalization) han reducido el sobreajuste, resultando en un modelo más robusto y estable.
3. **Precisión final:**
   La arquitectura mejorada logra una mayor precisión en validación (**85%**) en comparación con la original (**81%-82%**).

En general, las mejoras introducidas en la arquitectura han tenido un impacto positivo tanto en la estabilidad como en el desempeño en validación, lo que hace que el modelo mejorado sea más adecuado para datos no vistos, lo que ha resultado en un valor en Test más alto.

Proyecto IV - Visión 3

Arquitectura 2

```
# Arquitectura 2
model = ks.Sequential()
# Capas de captar patrones
# Primera capa convolucional con Batch Normalization, MaxPool y Dropout
model.add(layers.Conv2D(32, (3, 3), padding='same',input_shape=(32, 32,
3),kernel_initializer=initializers.HeNormal()))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.25))

# Segunda capa convolucional con Batch Normalization, MaxPool y Dropout
model.add(layers.Conv2D(64, (3, 3), padding='same',kernel_initializer=initializers.HeNormal()))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.3)

# Tercera capa convolucional con Batch Normalization, MaxPool y Dropout
model.add(layers.Conv2D(128, (3, 3), padding='same',kernel_initializer=initializers.HeNormal()))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.3))

# Cuarta capa convolucional con Batch Normalization, Dropout, sin MaxPooling para conservar
detalles espaciales complejas
model.add(layers.Conv2D(128, (3, 3),padding='same',kernel_initializer=initializers.HeNormal()))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.Dropout(0.4))

# Aplanar y pasar a capas densas
model.add(layers.Flatten())

# Capas de clasificación
# Primera capa
model.add(layers.Dense(256,kernel_initializer=initializers.HeNormal()))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.Dropout(0.4))
# Segunda capa
model.add(layers.Dense(128,kernel_initializer=initializers.HeNormal()))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.Dropout(0.4))
# Capa de salida
model.add(layers.Dense(10, activation='softmax'))
```

Proyecto IV - Visión 3

Arquitectura 2

Resumen de cambios con respecto a la Arquitectura 1:

La idea detrás de la segunda arquitectura era probar una versión más ligera (menos parámetros y menor costo computacional) de la red neuronal.

I. Cambios en la primera capa convolucional:

1. Filtros: Tiene 32 filtros en lugar de 64, lo que limita la cantidad de características que se capturan en esta etapa inicial. Esto es adecuado si los patrones básicos (bordes, texturas simples) no son demasiado complejos en los datos.

- **Impacto positivo:** Beneficia la reducción de complejidad y el costo computacional.
- **Impacto negativo:** Podría afectar la capacidad del modelo para captar patrones iniciales más complejos.

2. MaxPooling: Al incluir MaxPooling en la primera capa, reduce rápidamente las dimensiones espaciales, disminuyendo el costo computacional. Sin embargo, esto podría ocasionar la pérdida de detalles importantes desde el inicio del modelo.

3. Dropout: La reducción de Dropout (de 0.3 a 0.25) sugiere un enfoque más conservador en la regularización en esta etapa inicial.

II. Cambios en la tercera y cuarta capas convolucionales:

En la tercera capa se aplica MaxPooling, en lugar de hacerlo en la cuarta.

La **cuarta capa** (última antes de las capas densas) no tiene MaxPooling, lo que permite conservar patrones más complejos y pasarlos directamente a la etapa de clasificación.

III. Cambios generales:

1.Regularización L2:

- La ausencia de regularización L2 en esta arquitectura reduce la cantidad de cálculos necesarios durante el entrenamiento, ya que no es necesario calcular ni aplicar la penalización L2 en cada actualización de los pesos.
- **Impacto negativo:** Esto podría aumentar el riesgo de sobreajuste, especialmente en datos más complejos. Además, la reducción del costo computacional puede ser marginal en comparación con la pérdida de capacidad de generalización.

2. Dropout:

Se mantienen Dropouts en todas las capas, pero con valores ligeramente menores en algunas de ellas (por ejemplo, en la primera capa convolucional y las capas densas).

Proyecto IV - Visión 3

Arquitectura 2

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128
re_lu (ReLU)	(None, 32, 32, 32)	0
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
dropout (Dropout)	(None, 16, 16, 32)	0
conv2d_1 (Conv2D)	(None, 16, 16, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 16, 16, 64)	256
re_lu_1 (ReLU)	(None, 16, 16, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_1 (Dropout)	(None, 8, 8, 64)	0
conv2d_2 (Conv2D)	(None, 8, 8, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 8, 8, 128)	512
re_lu_2 (ReLU)	(None, 8, 8, 128)	0
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
conv2d_3 (Conv2D)	(None, 4, 4, 128)	147,584
batch_normalization_3 (BatchNormalization)	(None, 4, 4, 128)	512
re_lu_3 (ReLU)	(None, 4, 4, 128)	0
dropout_3 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 256)	524,544
batch_normalization_4 (BatchNormalization)	(None, 256)	1,024
re_lu_4 (ReLU)	(None, 256)	0
dropout_4 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32,896
batch_normalization_5 (BatchNormalization)	(None, 128)	512
re_lu_5 (ReLU)	(None, 128)	0
dropout_5 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1,290

Total params: 802,506 (3.06 MB)

Trainable params: 801,034 (3.06 MB)

Non-trainable params: 1,472 (5.75 KB)

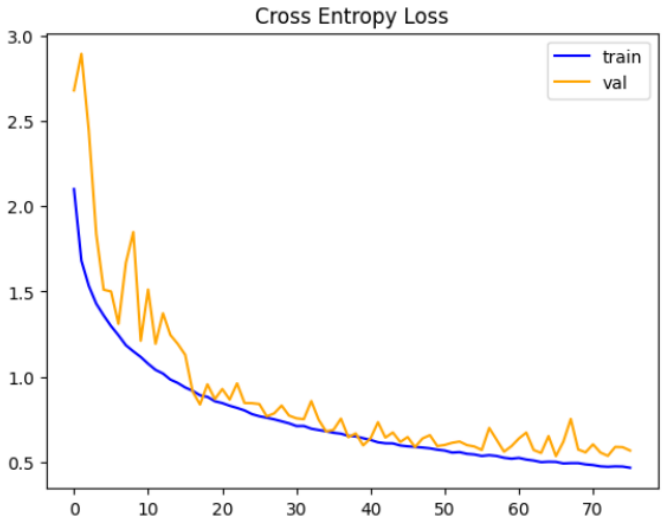
El modelo cuenta con **802,506 parámetros**, de los cuales la mayoría son entrenables y están orientados al aprendizaje de características. El modelo tiene una cantidad de parámetros **menor** que la Arquitectura 1.

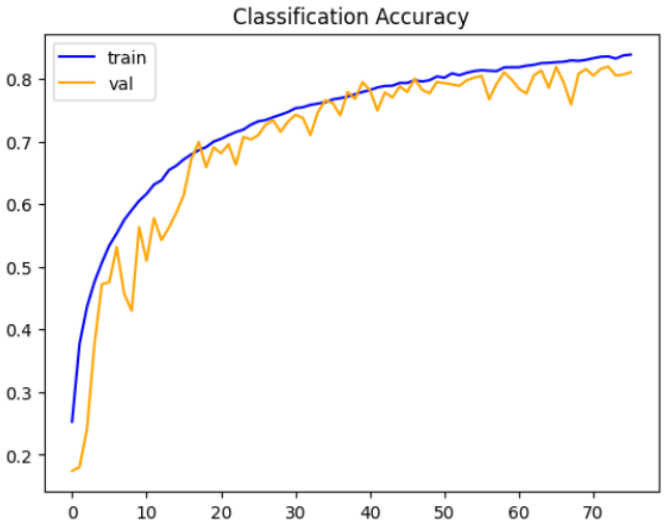
```
# Optimizador, función de error:
model.compile(optimizer='Adam',loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# Callback para early stopping
callback_val_loss=EarlyStopping(monitor="val_loss",patience=10,restore_best_weights=True)
callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10, restore_best_weights=True)
# Epochs
epochs=200
# Model fit
history = model.fit(x_train_scaled, y_train, epochs=epochs,
 batch_size= 512,callbacks=[callback_val_loss, callback_val_accuracy],validation_data=(x_val_scaled, y_val))

# Results
_, acc = model.evaluate(x_test_scaled, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
> 81.770
```

Se consigue el objetivo de alcanzar una Accuracy superior al 80% únicamente con la arquitectura y es el mejor resultado de todos.

Proyecto IV - Visión 3

Arquitectura 2





Comportamiento del entrenamiento:
La curva de entrenamiento alcanza una Accuracy de ~85% después de 70 épocas. El aprendizaje es más gradual y constante comparando con la Arquitectura 1, mostrando que el modelo sigue aprendiendo, aunque de forma más lenta después de las primeras 30 épocas.

Comportamiento de la validación:
La Accuracy en validación alcanza ~81%-82% y muestra fluctuaciones a lo largo del tiempo. Sin embargo, la curva de validación es consistente y se mantiene cercana a la de entrenamiento, lo que indica un buen balance entre aprendizaje y generalización.

Convergencia:
La curva de validación converge de manera más progresiva y estable, continuando su mejora durante un período más largo.

Conclusión comparando con la Arquitectura 1:

Ventajas: modelo más ligero, menor riesgo de sobreajuste, aprendizaje más gradual.
Desventajas: Accuracy final ligeramente inferior (81%-82%) y mayor tiempo de entrenamiento. Oscilaciones en la curva de validación que sugieren que le cuesta salir de mínimos locales en datos de validación.

Proyecto IV - Visión 4

Arquitectura 2 Con LeakyReLu

```
# Arquitectura 2
model = ks.Sequential()
# Capas de captar patrones
# Primera capa convolucional con Batch Normalization, MaxPool y Dropout
model.add(layers.Conv2D(32, (3, 3), padding='same',input_shape=(32, 32,
3),kernel_initializer=initializers.HeNormal()))
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU(alpha=0.1))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.25))

# Segunda capa convolucional con Batch Normalization, MaxPool y Dropout
model.add(layers.Conv2D(64, (3, 3), padding='same',kernel_initializer=initializers.HeNormal()))
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU(alpha=0.1))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.3)

# Tercera capa convolucional con Batch Normalization, MaxPool y Dropout
model.add(layers.Conv2D(128, (3, 3), padding='same',kernel_initializer=initializers.HeNormal()))
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU(alpha=0.1))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.3))

# Cuarta capa convolucional con Batch Normalization, Dropout, sin MaxPooling para conservar
detalles espaciales complejas
model.add(layers.Conv2D(128, (3, 3),padding='same',kernel_initializer=initializers.HeNormal()))
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU(alpha=0.1))
model.add(layers.Dropout(0.4))

# Aplanar y pasar a capas densas
model.add(layers.Flatten())

# Capas de clasificación
# Primera capa
model.add(layers.Dense(256,kernel_initializer=initializers.HeNormal()))
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU(alpha=0.1))
model.add(layers.Dropout(0.4))
# Segunda capa
model.add(layers.Dense(128,kernel_initializer=initializers.HeNormal()))
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU(alpha=0.1))
model.add(layers.Dropout(0.4))
# Capa de salida
model.add(layers.Dense(10, activation='softmax'))
```

Proyecto IV - Visión 4

Arquitectura 2
Con LeakyReLu

La idea detrás de reemplazar ReLU con Leaky ReLU en la arquitectura 2 es mejorar la capacidad del modelo para escapar de mínimos locales, mitigando problemas causados por unidades muertas o gradientes cero.

Proyecto IV - Visión 4

Arquitectura 2 Con LeakyReLu

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128
leaky_re_lu (LeakyReLU)	(None, 32, 32, 32)	0
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
dropout (Dropout)	(None, 16, 16, 32)	0
conv2d_1 (Conv2D)	(None, 16, 16, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 16, 16, 64)	256
leaky_re_lu_1 (LeakyReLU)	(None, 16, 16, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_1 (Dropout)	(None, 8, 8, 64)	0
conv2d_2 (Conv2D)	(None, 8, 8, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 8, 8, 128)	512
leaky_re_lu_2 (LeakyReLU)	(None, 8, 8, 128)	0
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
conv2d_3 (Conv2D)	(None, 4, 4, 128)	147,584
batch_normalization_3 (BatchNormalization)	(None, 4, 4, 128)	512
leaky_re_lu_3 (LeakyReLU)	(None, 4, 4, 128)	0
dropout_3 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 256)	524,544
leaky_re_lu_4 (LeakyReLU)	(None, 256)	0
dropout_4 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32,896
leaky_re_lu_5 (LeakyReLU)	(None, 128)	0
dropout_5 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1,290

Total params: 800,970 (3.06 MB)

Trainable params: 800,266 (3.05 MB)

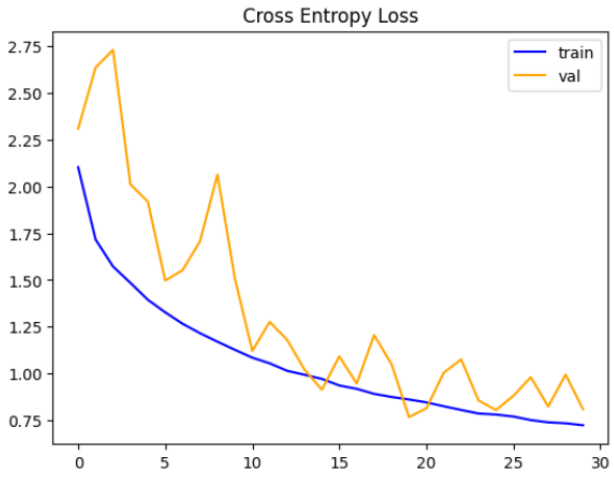
Non-trainable params: 704 (2.75 KB)

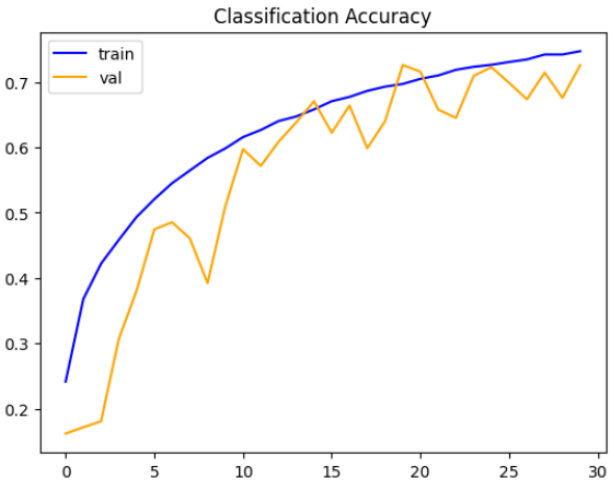
```
# Optimizador, función de error:
model.compile(optimizer='Adam',loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# Callback para early stopping
callback_val_loss=EarlyStopping(monitor="val_loss",patience=10,restore_best_weights=True)
callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10, restore_best_weights=True)
# Epochs
epochs=200
# Model fit
history = model.fit(x_train_scaled, y_train, epochs=epochs,
 batch_size= 512,callbacks=[callback_val_loss, callback_val_accuracy],validation_data=(x_val_scaled, y_val))

# Results
_, acc = model.evaluate(x_test_scaled, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
> 72.840
```

No se consigue el objetivo de alcanzar una Accuracy superior al 80% en el caso de usar LeakyReLU.

Proyecto IV - Visión 4

Arquitectura 2 Con LeakyReLu





La curva de validación muestra una mayor fluctuación y una Accuracy máxima significativamente menor (~70%) en comparación con ReLU (~81%-82%). Esto sugiere que el modelo con Leaky ReLU tiene dificultades para generalizar en el conjunto de validación.
Con Leaky ReLU, el aprendizaje parece ser más lento y menos efectivo.
En general, la Arquitectura 2 con Leaky ReLU muestra un **desempeño notablemente inferior** en comparación con la misma arquitectura utilizando ReLU, tanto en la precisión de entrenamiento como en la de validación. Esto podría atribuirse a:

1. La falta de regularización implícita que proporciona ReLU al "eliminar" activaciones negativas.
2. El flujo continuo de gradientes negativos, que puede amplificar el ruido en los datos. Permitir un flujo de gradientes pequeños para valores negativos (controlado por el parámetro alpha) puede resultar en activaciones más densas. Este comportamiento podría hacer que el modelo aprenda patrones no significativos, afectando negativamente la generalización.
3. Si los datos contienen características que generan muchas activaciones negativas en las capas iniciales, Leaky ReLU puede amplificarlas ligeramente debido al flujo de gradientes en la parte negativa. Esto podría llevar a un entrenamiento menos efectivo al preservar valores que no son útiles para la tarea.
4. El valor predeterminado de alpha=0.1 puede no ser adecuado para la tarea de clasificar CIFAR-10.
5. Leaky ReLU podría ser sensible a la normalización previa de los datos. Si la normalización no es adecuada, puede exacerbar el ruido en las activaciones negativas. Esto podría deberse a usar BatchNormalization antes de Leaky ReLU.

Arquitectura 3

```
# Arquitectura 3
model = models.Sequential()
# Capas de captar patrones
# Primera bicapa convolucional de 64 filtros con Batch Normalization, MaxPool y Dropout
model.add(layers.Conv2D(64, (3, 3), padding='same', input_shape=(32, 32,
3),kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.Dropout(0.3))
model.add(layers.Conv2D(64, (3, 3), padding='same',
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.3))

# segunda bicapa convolucional de 128 filtros con Batch Normalization, MaxPool y Dropout
model.add(layers.Conv2D(128, (3, 3),  padding='same',
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.Dropout(0.3))
model.add(layers.Conv2D(128, (3, 3),  padding='same',
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.4))

# tercera bicapa convolucional de 128 filtros con Batch Normalization, MaxPool y Dropout
model.add(layers.Conv2D(128, (3, 3),  padding='same',
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.Dropout(0.3))
model.add(layers.Conv2D(128, (3, 3),  padding='same',
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.4))

# Aplanar y pasar a capas densas
model.add(layers.Flatten())

# Capas de clasificación
# Primera capa
model.add(layers.Dense(256, kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-
4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.Dropout(0.5))
# Segunda capa
model.add(layers.Dense(128, kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-
4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.Dropout(0.5))
# Capa de salida
model.add(layers.Dense(10, activation='softmax'))
```

Arquitectura 3

Arquitectura 3: Mejoras respecto a la Arquitectura 2

La Arquitectura 3 introduce varias mejoras significativas en comparación con la Arquitectura 2, enfocándose en optimizar el aprendizaje, mejorar la capacidad de generalización y minimizar el sobreajuste para el conjunto de datos CIFAR-10.

1. **Uso de "bicapas" convolucionales:**
   En cada bloque convolucional, se incluyen dos capas convolucionales consecutivas en lugar de una sola. Esto permite capturar patrones más complejos y ricos en características gracias a la combinación de dos operaciones convolucionales antes de aplicar MaxPooling. Mejora la capacidad del modelo para identificar detalles finos y jerárquicos en las imágenes, lo que es esencial para un conjunto de datos CIFAR-10.
2. **Organización jerárquica más eficiente:**
   Primera bicapa convolucional: Utiliza 64 filtros en cada capa, junto con Batch Normalization y MaxPooling en la última capa de las dos. Reduce las dimensiones espaciales desde el principio, optimizando el procesamiento de imágenes pequeñas como las de CIFAR-10.
   Segunda y tercera bicapas convolucionales: Incremento a 128 filtros, lo que permite capturar características más abstractas y avanzadas. Cada bicapa incluye dos convoluciones (la segunda capa convolucional con Batch Normalization, MaxPooling y Dropout), logrando un equilibrio entre extracción de características y regularización.

Características heredadas de la Arquitectura 2:
Se mantienen las características importantes que han demostrado ser efectivas para el buen rendimiento del modelo:

1. kernel_initializer=initializers.HeUniform(): Este método está diseñado para inicializar pesos de manera óptima en redes neuronales profundas que usan activaciones ReLU.
2. Regularización L2 (kernel_regularizer=1e-4): Penaliza pesos excesivamente grandes, lo que ayuda a reducir el riesgo de sobreajuste y mejora la capacidad de generalización del modelo. Esto es crucial para evitar memorizar patrones específicos de CIFAR-10 y garantizar un buen rendimiento en validación.
3. Activación ReLU después de BatchNormalization: Este orden mejora la estabilidad y robustez del modelo, ya que normaliza los pesos antes de aplicar la no linealidad.
4. MaxPooling: Cada bicapa reduce gradualmente las dimensiones espaciales, preservando información clave sin perder detalles esenciales.
5. Dropout:Los valores seleccionados (0.3 y 0.4) en las capas convolucionales están cuidadosamente ajustados para evitar el sobreajuste sin ralentizar significativamente el aprendizaje.
6. Diseño de capas de clasificación y salida:Se mantienen las mismas capas de clasificación y salida utilizadas en los modelos anteriores, con 256 y 128 neuronas densas respectivamente, y un Dropout de 0.5 para regularización adicional.

Arquitectura 3

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_10 (Conv2D)	(None, 32, 32, 64)	1,792
batch_normalization_14 (BatchNormalization)	(None, 32, 32, 64)	256
re_lu_14 (ReLU)	(None, 32, 32, 64)	0
dropout_13 (Dropout)	(None, 32, 32, 64)	0
conv2d_11 (Conv2D)	(None, 32, 32, 64)	36,928
batch_normalization_15 (BatchNormalization)	(None, 32, 32, 64)	256
re_lu_15 (ReLU)	(None, 32, 32, 64)	0
max_pooling2d_5 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout_14 (Dropout)	(None, 16, 16, 64)	0
conv2d_12 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_16 (BatchNormalization)	(None, 16, 16, 128)	512
re_lu_16 (ReLU)	(None, 16, 16, 128)	0
dropout_15 (Dropout)	(None, 16, 16, 128)	0
conv2d_13 (Conv2D)	(None, 16, 16, 128)	147,584
batch_normalization_17 (BatchNormalization)	(None, 16, 16, 128)	512
re_lu_17 (ReLU)	(None, 16, 16, 128)	0
max_pooling2d_6 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_16 (Dropout)	(None, 8, 8, 128)	0
conv2d_14 (Conv2D)	(None, 8, 8, 128)	147,584
batch_normalization_18 (BatchNormalization)	(None, 8, 8, 128)	512
re_lu_18 (ReLU)	(None, 8, 8, 128)	0
dropout_17 (Dropout)	(None, 8, 8, 128)	0
conv2d_15 (Conv2D)	(None, 8, 8, 128)	147,584
batch_normalization_19 (BatchNormalization)	(None, 8, 8, 128)	512
re_lu_19 (ReLU)	(None, 8, 8, 128)	0
max_pooling2d_7 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_18 (Dropout)	(None, 4, 4, 128)	0
flatten_2 (Flatten)	(None, 2048)	0
dense_6 (Dense)	(None, 256)	524,544
batch_normalization_20 (BatchNormalization)	(None, 256)	1,024
re_lu_20 (ReLU)	(None, 256)	0
dropout_19 (Dropout)	(None, 256)	0
dense_7 (Dense)	(None, 128)	32,896
batch_normalization_21 (BatchNormalization)	(None, 128)	512
re_lu_21 (ReLU)	(None, 128)	0
dropout_20 (Dropout)	(None, 128)	0
dense_8 (Dense)	(None, 10)	1,290

Total params: 1,118,154 (4.27 MB)

Trainable params: 1,116,106 (4.26 MB)

Non-trainable params: 2,048 (8.00 KB)

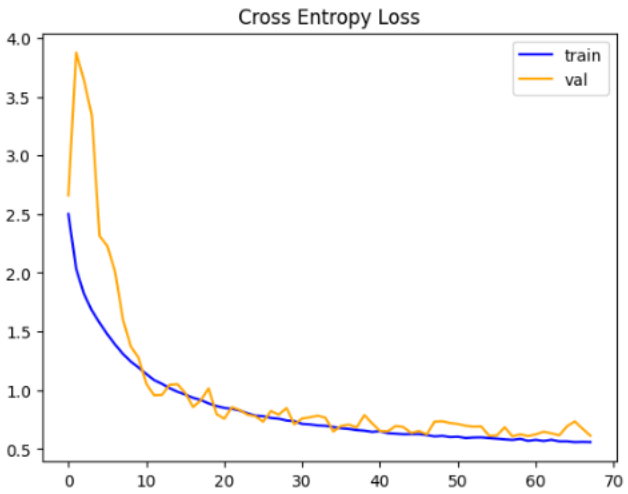
El modelo cuenta con **1,118,154 parámetros**, de los cuales la mayoría son entrenables y están orientados al aprendizaje de características.

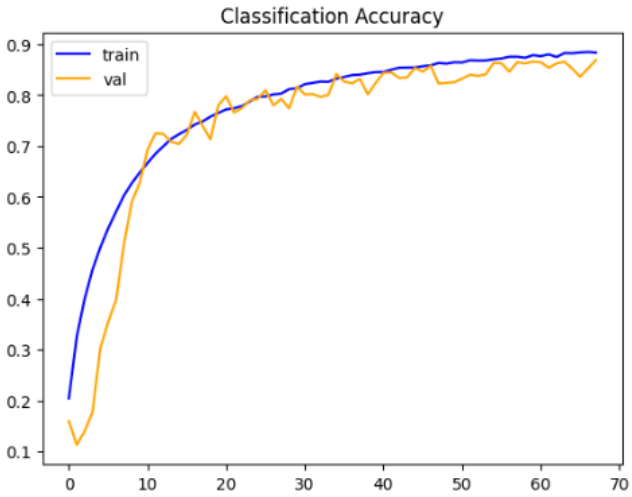
```
# Optimizador, función de error:
model.compile(optimizer='Adam',loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# Callback para early stopping
callback_val_loss=EarlyStopping(monitor="val_loss",patience=10,restore_best_weights=True)
callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10, restore_best_weights=True)
# Epochs
epochs=200
# Model fit
history = model.fit(x_train_scaled, y_train, epochs=epochs,
 batch_size= 512,callbacks=[callback_val_loss, callback_val_accuracy],validation_data=(x_val_scaled, y_val))

# Results
_, acc = model.evaluate(x_test_scaled, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
> 86.390
```

Se logra el objetivo de alcanzar una Accuracy superior al 80% únicamente con la arquitectura, siendo este el mejor resultado obtenido.

Proyecto IV - Visión 5

Arquitectura 3





Analysis de Arquitectura 3:

1. Desempeño:

- La Accuracy tanto en entrenamiento como en validación es notablemente más consistente que en la Arquitectura 2. El modelo muestra una mejora gradual a lo largo de las épocas y una mayor estabilidad en ambas curvas.
- En validación, la Accuracy alcanza valores cercanos al 85%, con una brecha mínima respecto a la de entrenamiento. Esto indica que el modelo es más generalizable y menos susceptible al sobreajuste, lo cual era un problema evidente en la Arquitectura.
- Este comportamiento sugiere que los mecanismos de regularización (Dropout, L2 y bicapas convolucionales) están funcionando adecuadamente para evitar un ajuste excesivo a los datos de entrenamiento

2. Convergencia:

- La convergencia inicial ocurre alrededor de las primeras 8 épocas, lo que demuestra un aprendizaje eficiente desde el inicio.
- Después de este punto, la Accuracy sigue mejorando de manera constante durante las épocas posteriores, indicando que el modelo continúa extrayendo información útil del conjunto de datos. Este comportamiento sugiere un aprendizaje más profundo y efectivo.

3. Estabilidad:

- La Accuracy es más estable en comparación con la Arquitectura 2, reflejando un mejor balance entre aprendizaje y generalización.
- Las bicapas convolucionales parecen ser clave para mejorar la capacidad de aprendizaje del modelo sin comprometer su estabilidad. Estas permiten que el modelo maneje mejor características complejas del conjunto de datos, a la vez que evitan un entrenamiento errático.
- A pesar de la mejora general, se observan pequeñas oscilaciones en la curva de validación, lo que sugiere que el modelo podría enfrentarse a mínimos locales en ciertos momentos.

En resumen, la Arquitectura 3 es claramente superior, alcanzando un alto rendimiento en CIFAR-10 al abordar las limitaciones de la Arquitectura 2.

Proyecto IV - Visión 6

Arquitectura 3

Data Augmentation peor

```
# Arquitectura 3
model = models.Sequential()
# Capas de captar patrones
# Primera bicapa convolucional de 64 filtros con Batch Normalization, MaxPool y Dropout
model.add(layers.Conv2D(64, (3, 3), padding='same', input_shape=(32, 32,
3),kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.Dropout(0.3))
model.add(layers.Conv2D(64, (3, 3), padding='same',
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.3))

# segunda bicapa convolucional de 128 filtros con Batch Normalization, MaxPool y Dropout
model.add(layers.Conv2D(128, (3, 3),  padding='same',
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.Dropout(0.3))
model.add(layers.Conv2D(128, (3, 3),  padding='same',
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.4))

# tercera bicapa convolucional de 128 filtros con Batch Normalization, MaxPool y Dropout
model.add(layers.Conv2D(128, (3, 3),  padding='same',
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.Dropout(0.3))
model.add(layers.Conv2D(128, (3, 3),  padding='same',
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.4))

# Aplanar y pasar a capas densas
model.add(layers.Flatten())

# Capas de clasificación
# Primera capa
model.add(layers.Dense(256, kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-
4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.Dropout(0.5))
# Segunda capa
model.add(layers.Dense(128, kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-
4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.Dropout(0.5))
# Capa de salida
model.add(layers.Dense(10, activation='softmax'))
```

Proyecto IV - Visión 6

Arquitectura 3

Data Augmentation peor

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_10 (Conv2D)	(None, 32, 32, 64)	1,792
batch_normalization_14 (BatchNormalization)	(None, 32, 32, 64)	256
re_lu_14 (ReLU)	(None, 32, 32, 64)	0
dropout_13 (Dropout)	(None, 32, 32, 64)	0
conv2d_11 (Conv2D)	(None, 32, 32, 64)	36,928
batch_normalization_15 (BatchNormalization)	(None, 32, 32, 64)	256
re_lu_15 (ReLU)	(None, 32, 32, 64)	0
max_pooling2d_5 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout_14 (Dropout)	(None, 16, 16, 64)	0
conv2d_12 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_16 (BatchNormalization)	(None, 16, 16, 128)	512
re_lu_16 (ReLU)	(None, 16, 16, 128)	0
dropout_15 (Dropout)	(None, 16, 16, 128)	0
conv2d_13 (Conv2D)	(None, 16, 16, 128)	147,584
batch_normalization_17 (BatchNormalization)	(None, 16, 16, 128)	512
re_lu_17 (ReLU)	(None, 16, 16, 128)	0
max_pooling2d_6 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_16 (Dropout)	(None, 8, 8, 128)	0
conv2d_14 (Conv2D)	(None, 8, 8, 128)	147,584
batch_normalization_18 (BatchNormalization)	(None, 8, 8, 128)	512
re_lu_18 (ReLU)	(None, 8, 8, 128)	0
dropout_17 (Dropout)	(None, 8, 8, 128)	0
conv2d_15 (Conv2D)	(None, 8, 8, 128)	147,584
batch_normalization_19 (BatchNormalization)	(None, 8, 8, 128)	512
re_lu_19 (ReLU)	(None, 8, 8, 128)	0
max_pooling2d_7 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_18 (Dropout)	(None, 4, 4, 128)	0
flatten_2 (Flatten)	(None, 2048)	0
dense_6 (Dense)	(None, 256)	524,544
batch_normalization_20 (BatchNormalization)	(None, 256)	1,024
re_lu_20 (ReLU)	(None, 256)	0
dropout_19 (Dropout)	(None, 256)	0
dense_7 (Dense)	(None, 128)	32,896
batch_normalization_21 (BatchNormalization)	(None, 128)	512
re_lu_21 (ReLU)	(None, 128)	0
dropout_20 (Dropout)	(None, 128)	0
dense_8 (Dense)	(None, 10)	1,290

Total params: 1,118,154 (4.27 MB)

Trainable params: 1,116,106 (4.26 MB)

Non-trainable params: 2,048 (8.00 KB)

El modelo cuenta con **1,118,154 parámetros**, de los cuales la mayoría son entrenables y están orientados al aprendizaje de características.

```
# Optimizador, función de error:
model.compile(optimizer='Adam',loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# Callback para early stopping
callback_val_loss=EarlyStopping(monitor="val_loss",patience=10,restore_best_weights=True)
callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10, restore_best_weights=True)
```

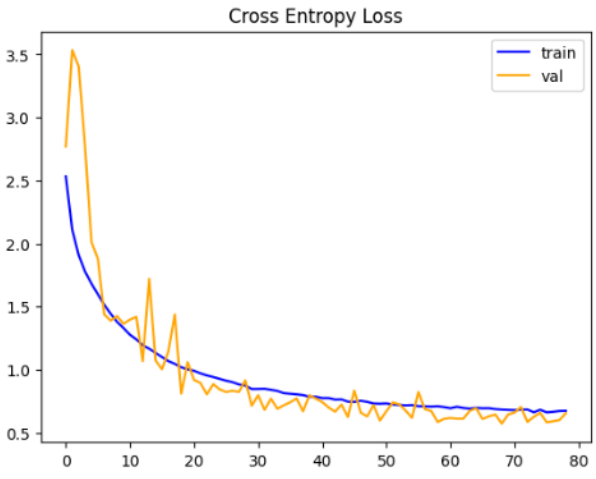
Arquitectura 3

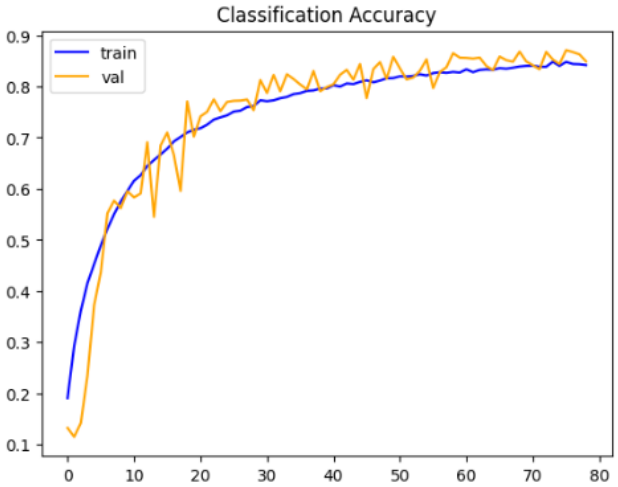
Data Augmentation peor

```
# generación de imagenes distorsionadas
train_datagen = ImageDataGenerator(
   rotation_range=5,
   horizontal_flip=True,
   width_shift_range=0.1,
   height_shift_range=0.1,
   brightness_range=(0.9, 1.2),  # Brightness adjustment
   rescale=1./255,)
train_generator = train_datagen.flow(
   x_train,
   y_train,
   batch_size=512)
validation_datagen = ImageDataGenerator(
   rescale=1./255)
validation_generator = validation_datagen.flow(
   x_val,
   y_val,
   batch_size=512)
```

```
# Epochs
epochs=200
# Model fit
history = model.fit(train_generator, epochs=epochs, validation_data=validation_generator,
            steps_per_epoch=80,validation_steps=80, callbacks=[callback_val_loss,
callback_val_accuracy],
            )
# Results
_, acc = model.evaluate(x_test_scaled, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
> 86.1
```

La Accuracy ha disminuido debido a un exceso de distorsión en las imágenes.





Resumen Data Augmentation con Arquitectura 3 :

El uso de data augmentation puede introducir un mayor nivel de variación en los datos, lo cual es beneficioso para aumentar la robustez del modelo y prevenir el sobreajuste.

Sin embargo, como se observa en las gráficas, cuando las transformaciones son demasiado agresivas, como en este caso, los datos distorsionados dificultan que el modelo identifique patrones claros y consistentes.

Esto resulta en una curva de validación menos estable, con fluctuaciones notables, y en una ligera disminución de la precisión general en comparación con el entrenamiento sin data augmentation.

Estas oscilaciones reflejan que el modelo enfrenta dificultades para generalizar correctamente, ya que las imágenes transformadas pueden diferir significativamente de las distribuciones originales, complicando la convergencia y reduciendo la efectividad del aprendizaje.

Proyecto IV - Visión 7

Arquitectura 3

Data Augmentation mejor

```
# Arquitectura 3
model = models.Sequential()
# Capas de captar patrones
# Primera bicapa convolucional de 64 filtros con Batch Normalization, MaxPool y Dropout
model.add(layers.Conv2D(64, (3, 3), padding='same', input_shape=(32, 32,
3),kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.Dropout(0.3))
model.add(layers.Conv2D(64, (3, 3), padding='same',
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.3))

# segunda bicapa convolucional de 128 filtros con Batch Normalization, MaxPool y Dropout
model.add(layers.Conv2D(128, (3, 3),  padding='same',
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.Dropout(0.3))
model.add(layers.Conv2D(128, (3, 3),  padding='same',
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.4))

# tercera bicapa convolucional de 128 filtros con Batch Normalization, MaxPool y Dropout
model.add(layers.Conv2D(128, (3, 3),  padding='same',
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.Dropout(0.3))
model.add(layers.Conv2D(128, (3, 3),  padding='same',
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.4))

# Aplanar y pasar a capas densas
model.add(layers.Flatten())

# Capas de clasificación
# Primera capa
model.add(layers.Dense(256, kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-
4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.Dropout(0.5))
# Segunda capa
model.add(layers.Dense(128, kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-
4)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.Dropout(0.5))
# Capa de salida
model.add(layers.Dense(10, activation='softmax'))
```

Proyecto IV - Visión 7

Arquitectura 3

Data Augmentation mejor

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_10 (Conv2D)	(None, 32, 32, 64)	1,792
batch_normalization_14 (BatchNormalization)	(None, 32, 32, 64)	256
re_lu_14 (ReLU)	(None, 32, 32, 64)	0
dropout_13 (Dropout)	(None, 32, 32, 64)	0
conv2d_11 (Conv2D)	(None, 32, 32, 64)	36,928
batch_normalization_15 (BatchNormalization)	(None, 32, 32, 64)	256
re_lu_15 (ReLU)	(None, 32, 32, 64)	0
max_pooling2d_5 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout_14 (Dropout)	(None, 16, 16, 64)	0
conv2d_12 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_16 (BatchNormalization)	(None, 16, 16, 128)	512
re_lu_16 (ReLU)	(None, 16, 16, 128)	0
dropout_15 (Dropout)	(None, 16, 16, 128)	0
conv2d_13 (Conv2D)	(None, 16, 16, 128)	147,584
batch_normalization_17 (BatchNormalization)	(None, 16, 16, 128)	512
re_lu_17 (ReLU)	(None, 16, 16, 128)	0
max_pooling2d_6 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_16 (Dropout)	(None, 8, 8, 128)	0
conv2d_14 (Conv2D)	(None, 8, 8, 128)	147,584
batch_normalization_18 (BatchNormalization)	(None, 8, 8, 128)	512
re_lu_18 (ReLU)	(None, 8, 8, 128)	0
dropout_17 (Dropout)	(None, 8, 8, 128)	0
conv2d_15 (Conv2D)	(None, 8, 8, 128)	147,584
batch_normalization_19 (BatchNormalization)	(None, 8, 8, 128)	512
re_lu_19 (ReLU)	(None, 8, 8, 128)	0
max_pooling2d_7 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_18 (Dropout)	(None, 4, 4, 128)	0
flatten_2 (Flatten)	(None, 2048)	0
dense_6 (Dense)	(None, 256)	524,544
batch_normalization_20 (BatchNormalization)	(None, 256)	1,024
re_lu_20 (ReLU)	(None, 256)	0
dropout_19 (Dropout)	(None, 256)	0
dense_7 (Dense)	(None, 128)	32,896
batch_normalization_21 (BatchNormalization)	(None, 128)	512
re_lu_21 (ReLU)	(None, 128)	0
dropout_20 (Dropout)	(None, 128)	0
dense_8 (Dense)	(None, 10)	1,290

Total params: 1,118,154 (4.27 MB)

Trainable params: 1,116,106 (4.26 MB)

Non-trainable params: 2,048 (8.00 KB)

El modelo cuenta con **1,118,154 parámetros**, de los cuales la mayoría son entrenables y están orientados al aprendizaje de características.

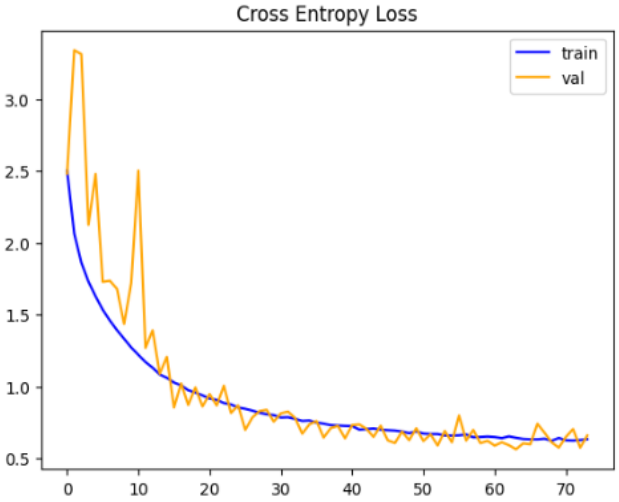
```
# Optimizador, función de error:
model.compile(optimizer='Adam',loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# Callback para early stopping
callback_val_loss=EarlyStopping(monitor="val_loss",patience=10,restore_best_weights=True)
callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10, restore_best_weights=True)
```

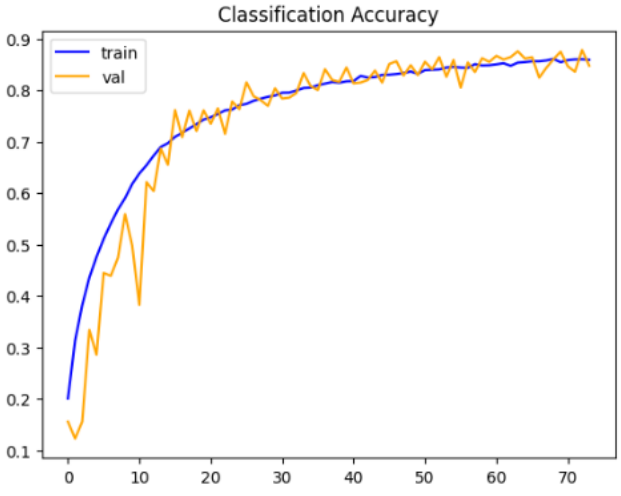
Arquitectura 3

Data Augmentation mejor

```
# generación de imagenes distorsionadas
train_datagen = ImageDataGenerator(
   rotation_range=5,
   horizontal_flip=True,
   brightness_range=(0.9, 1.2),  # Brightness adjustment
   rescale=1./255,)
train_generator = train_datagen.flow(
   x_train,
   y_train,
   batch_size=512)
validation_datagen = ImageDataGenerator(
   rescale=1./255)
validation_generator = validation_datagen.flow(
   x_val,
   y_val,
   batch_size=512)

# Epochs
epochs=200
# Model fit
history = model.fit(train_generator, epochs=epochs, validation_data=validation_generator,
            steps_per_epoch=80,validation_steps=80, callbacks=[callback_val_loss,
callback_val_accuracy])
# Results
_, acc = model.evaluate(x_test_scaled, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
> 87.170
```

Al reducir la cantidad de distorsiones en comparación con el proyecto anterior, se ha logrado incrementar la Accuracy (87.17) en relación con la obtenida en la Arquitectura 3 (86.39).





Resumen Data Augmentation con Arquitectura 3 mejorado:

Las imágenes de CIFAR-10, debido a su baja resolución, requieren un enfoque de data augmentation suave para mejorar la Accuracy sin comprometer la capacidad del modelo para captar patrones significativos.

Tras probar más de 20 configuraciones de data augmentation, esta combinación específica ha sido la única que ha logrado mejorar la Accuracy, gracias a la aplicación de transformaciones moderadas como rotación, volteo horizontal y ajustes en el brillo (brightness).

Este enfoque se ve reflejado en una curva de validación más estable, con menores oscilaciones y una mejor convergencia con la curva de entrenamiento.

En contraste, otras transformaciones más agresivas resultaron contraproducentes, ya que distorsionaban excesivamente las imágenes, dificultando que el modelo identificara las características esenciales del conjunto de datos, lo que afectaba negativamente tanto la estabilidad como la generalización.

Proyecto IV - Visión 8

Transfer Learning Basic con resizing

```
# Creamos una red que será extracción de features basada en VGG16 entrenada con ImageNet (224x224x3)
# Definir entrada original de CIFAR-10
input_layer = layers.Input(shape=(32, 32, 3))
# Redimensionar la entrada a 224x224
resized_input = layers.Resizing(224, 224, interpolation="bilinear")(input_layer)
model_vgg16 = vgg16.VGG16(include_top=False, weights='imagenet', input_tensor=resized_input,
input_shape=(224, 224, 3))
# output
output = model_vgg16.layers[-1].outpu
# output_layer
output_layer = layers.Flatten()(output)
model_prevgg16 = Model(model_vgg16.input, output_layer)
model_prevgg16.summary()
```

```
Model: "functional_10"
```

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 32, 32, 3)	0
resizing_1 (Resizing)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten_1 (Flatten)	(None, 25088)	0

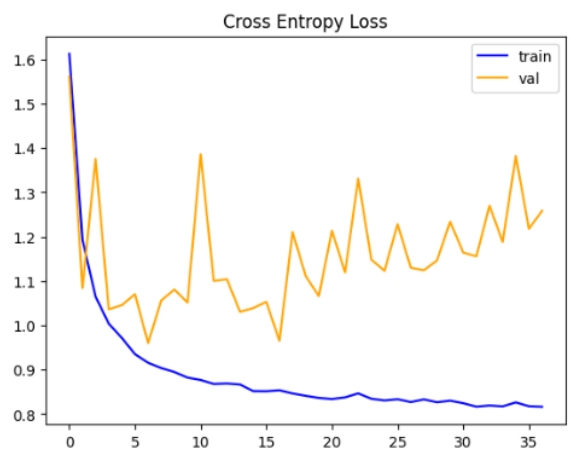
```
Total params: 14,714,688 (56.13 MB)

Trainable params: 14,714,688 (56.13 MB)

Non-trainable params: 0 (0.00 B)
```

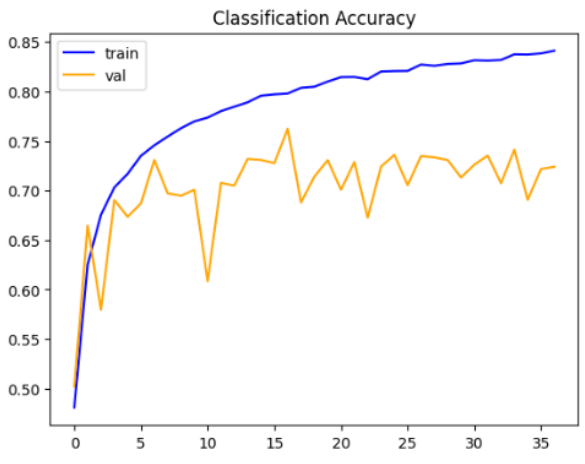
Red de classificacion

```
# Capas de clasificación
output_from_vgg16=25088

model_post_vgg = models.Sequential()
model_post_vgg.add(layers.InputLayer(input_shape=(output_from_vgg16,)))
# Primera capa
model_post_vgg.add(layers.Dense(256,
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model_post_vgg.add(layers.BatchNormalization())
model_post_vgg.add(layers.ReLU())
model_post_vgg.add(layers.Dropout(0.6))
# Segunda capa
model_post_vgg.add(layers.Dense(128,
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model_post_vgg.add(layers.BatchNormalization())
model_post_vgg.add(layers.ReLU())
model_post_vgg.add(layers.Dropout(0.6))
# Capa de salida
model_post_vgg.add(layers.Dense(10, activation='softmax'))
```

Transfer Learning Basic con resizing

```
# Optimizer
model_post_vgg.compile(optimizer='Adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

def llevar_a_cuello_de_botella(model, datos):
  return model.predict(datos) # model(datos)

# Pasar los datos por el cuello de de botella
x_train_postvgg16 = llevar_a_cuello_de_botella(model_prevgg16, x_train_scaled)
x_val_postvgg16 = llevar_a_cuello_de_botella(model_prevgg16, x_val_scaled)
x_test_postvgg16 = llevar_a_cuello_de_botella(model_prevgg16, x_test_scaled)

# Callbacks
callback_val_loss=EarlyStopping(monitor="val_loss",patience=30, restore_best_weights=True)
callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=30, restore_best_weights=True)

# Epochs
epochs=500

# Model fitting
history = model_post_vgg.fit(x=x_train_postvgg16, y=y_train, batch_size=256,
                             epochs=epochs, callbacks=[callback_val_loss, callback_val_accuracy],
                             validation_data=(x_val_postvgg16, y_val))

# Results
_, acc = model_post_vgg.evaluate(x_test_scaled, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
> 75.19
```





Transfer Learning de VGG16 Básico con Resizing:

Comportamiento del entrenamiento:

La Accuracy en entrenamiento mejora consistentemente a lo largo de las épocas y alcanza alrededor del 85% al final. Esto sugiere que el modelo está aprendiendo las características del conjunto de entrenamiento de manera progresiva.

Comportamiento de la validación:

La curva de validación es mucho más inestable y no supera el 75%, mostrando una gran variabilidad entre épocas. Esto indica que el modelo tiene problemas de generalización, lo que puede estar relacionado con varios factores:

1.Sobreajuste: El modelo puede estar aprendiendo demasiado los detalles del conjunto de entrenamiento y no generaliza bien para los datos de validación.

2.No es capaz de salir de los mínimos locales.

3.El cambio de tamaño puede estar introduciendo ruido en las imágenes y el modelo está aprendiendo patrones irrelevantes debido a interpolaciones.

Resumen:

El modelo se estanca y no aprende lo suficiente. Hay que probar fine tunning.

Proyecto IV - Visión 9

Transfer Learning Fine Tuning con Data Augmentation VGG16

```
# Crear una red que será extracción de features basada en VGG16 entrenada con ImageNet (224x224x3)
model_vgg16 = vgg16.VGG16(include_top=False, weights='imagenet', input_shape=(224, 224, 3))
output = model_vgg16.layers[-1].output
output_layer = ks.layers.Flatten()(output)
model_prevgg16 = Model(model_vgg16.input, output_layer)

# Descongelar las ultimas 30 capas
trainable = False
for layer in model_prevgg16.layers:
  if layer.name == "block5_conv1":
    trainable = True
  layer.trainable = trainable


layers = [(layer, layer.name, layer.trainable) for layer in model_prevgg16.layers]
pd.DataFrame(layers, columns=["Layer", "Name", "Is trainable?"])
```

	Layer	Name	Is trainable?
0	<InputLayer name=input_layer, built=True>	input_layer	False
1	<Conv2D name=block1_conv1, built=True>	block1_conv1	False
2	<Conv2D name=block1_conv2, built=True>	block1_conv2	False
3	<MaxPooling2D name=block1_pool, built=True>	block1_pool	False
4	<Conv2D name=block2_conv1, built=True>	block2_conv1	False
5	<Conv2D name=block2_conv2, built=True>	block2_conv2	False
6	<MaxPooling2D name=block2_pool, built=True>	block2_pool	False
7	<Conv2D name=block3_conv1, built=True>	block3_conv1	False
8	<Conv2D name=block3_conv2, built=True>	block3_conv2	False
9	<Conv2D name=block3_conv3, built=True>	block3_conv3	False
10	<MaxPooling2D name=block3_pool, built=True>	block3_pool	False
11	<Conv2D name=block4_conv1, built=True>	block4_conv1	False
12	<Conv2D name=block4_conv2, built=True>	block4_conv2	False
13	<Conv2D name=block4_conv3, built=True>	block4_conv3	False
14	<MaxPooling2D name=block4_pool, built=True>	block4_pool	False
15	<Conv2D name=block5_conv1, built=True>	block5_conv1	True
16	<Conv2D name=block5_conv2, built=True>	block5_conv2	True
17	<Conv2D name=block5_conv3, built=True>	block5_conv3	True
18	<MaxPooling2D name=block5_pool, built=True>	block5_pool	True
19	<Flatten name=flatten, built=True>	flatten	True

```
Model: "functional"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0

```
Total params: 14,714,688 (56.13 MB)

Trainable params: 7,079,424 (27.01 MB)

Non-trainable params: 7,635,264 (29.13 MB)
```

Proyecto IV - Visión 9

Transfer Learning Fine Tuning con Data Augmentation VGG16

```
# Crear una red de clasificación
# Definir entrada original de CIFAR-10
model_post_vgg = ks.Sequential()
# Define la capa de entrada con la forma de las imágenes de CIFAR-10
model_post_vgg.add(ks.layers.InputLayer(input_shape=(32,32,3),))
# Agregar una capa de redimensionamiento para cambiar el tamaño a (224, 224, 3)
model_post_vgg.add(ks.layers.Resizing(224,224,interpolation="bilinear"))
# Agregar model_prevgg16
model_post_vgg.add(model_prevgg16)
# Primera capa
model_post_vgg.add(ks.layers.Dense(256,
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model_post_vgg.add(ks.layers.BatchNormalization())
model_post_vgg.add(ks.layers.ReLU())
model_post_vgg.add(ks.layers.Dropout(0.5))
# Segunda capa
model_post_vgg.add(ks.layers.Dense(128,
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model_post_vgg.add(ks.layers.BatchNormalization())
model_post_vgg.add(ks.layers.ReLU())
model_post_vgg.add(ks.layers.Dropout(0.5))
# Capa de salida
model_post_vgg.add(ks.layers.Dense(10, activation='softmax'))
```

```
model_post_vgg.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
resizing (Resizing)	(None, 224, 224, 3)	0
functional (Functional)	(None, 25088)	14,714,688
dense (Dense)	(None, 256)	6,422,784
batch_normalization (BatchNormalization)	(None, 256)	1,024
re_lu (ReLU)	(None, 256)	0
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32,896
batch_normalization_1 (BatchNormalization)	(None, 128)	512
re_lu_1 (ReLU)	(None, 128)	0
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1,290

Total params: 21,173,194 (80.77 MB)

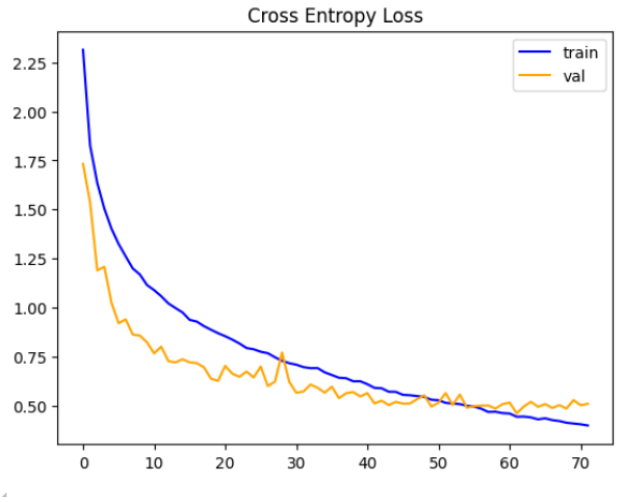
Trainable params: 13,537,162 (51.64 MB)

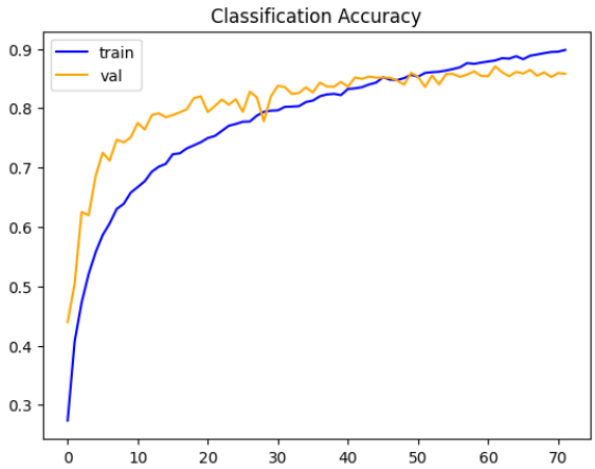
Non-trainable params: 7,636,032 (29.13 MB)

```
# Optimizer
model_post_vgg.compile(optimizer=Adam(learning_rate=2e-5),
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])

# Generación de imagenes distorsionadas
train_datagen = ImageDataGenerator(
   rotation_range=10,
   width_shift_range=0.1,
   height_shift_range=0.1,
   horizontal_flip=True,
   brightness_range=(0.9, 1.2),
   rescale=1./255,)
train_generator = train_datagen.flow(
   x_train_scaled,
   y_train,
   batch_size=128)
validation_datagen = ImageDataGenerator(
   rescale=1./255)
validation_generator = validation_datagen.flow(
   x_val_scaled,
   y_val,
   batch_size=128)
```

Transfer Learning Fine Tuning con Data Augmentation VGG16

```
# Callback para early stopping
callback_val_loss=EarlyStopping(monitor="val_loss",patience=10,restore_best_weights=True)
callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10, restore_best_weights=True)

# Model fit
epochs=500
history = model_post_vgg.fit(train_generator, epochs=epochs, validation_data=validation_generator,
  steps_per_epoch=200,validation_steps=200, callbacks=[callback_val_loss, callback_val_accuracy])

# Results
_, acc = model_post_vgg.evaluate(x_test_scaled, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
> 87.16
```





Transfer Learning de VGG16 con Fine Tuning y Data Augmentation:

Comportamiento del entrenamiento:

- La Accuracy de entrenamiento aumenta rápidamente durante las primeras 10-30 épocas, alcanzando alrededor del 80%, lo que corresponde a que el modelo está aprendiendo características generales del dataset.
- Después de las 30 épocas, la curva de entrenamiento muestra un crecimiento más lento, acercándose al 90%. Esto indica que el modelo sigue mejorando y aprendiendo características más específicas.

Comportamiento de la validación:

- La Accuracy de validación mejora rápidamente durante las primeras épocas, alcanzando alrededor del 80% en las primeras 10 épocas, lo cual indica que el modelo generaliza bien inicialmente.
- A partir de la época 30, la Accuracy de validación comienza a oscilar ligeramente alrededor del 85-88%, mostrando cierta estabilidad con leves fluctuaciones.

Diferencia entre Entrenamiento y Validación:

La diferencia entre las curvas es de aproximadamente 3%, lo que sugiere que el modelo está generalizando bien a los datos de validación sin sobreajustarse significativamente, ya que implica que el modelo no está memorizando los datos de entrenamiento.

Resumen:

La Accuracy de validación se estabiliza antes de alcanzar el 90%, lo que sugiere que el modelo ha alcanzado su límite con la configuración actual.

Parece ser que el modelo VGG16 no es el más óptimo para un Dataset de imágenes pequeños y de baja resolución como Cifar 10. Hay que implementar otras redes preentrenadas.

Transfer Learning Fine Tuning con Data Augmentation DenseNet121

```
# Crear una red que será extracción de features basada en DenseNet121 entrenada con ImageNet (224x224x3)
base_model = DenseNet121(include_top=False, weights='imagenet', input_shape=(224, 224, 3))
output = base_model.layers[-1].output
output_layer = ks.layers.Flatten()(output)
model_prebase= Model(base_model.input, output_layer)
# Descongelamos las ultimas 30 capas
trainable = False
# Iterar por todas las capas y establecer trainable en False
for layer in model_prebase.layers:
        layer.trainable = trainable
# Iterar a través de las últimas 30 capas de model_prebase y establecer trainable en True
for layer in model_prebase.layers[-30:]:
        layer.trainable = True

layers = [(layer, layer.name, layer.trainable) for layer in model_prebase.layers]
pd.DataFrame(layers, columns=["Layer", "Name", "Is trainable?"])
```

	Layer	Name	Is trainable?
0	<InputLayer name=input_layer_2, built=True>	input_layer_2	False
1	<ZeroPadding2D name=zero_padding2d_2, built=True>	zero_padding2d_2	False
2	<Conv2D name=conv1_conv, built=True>	conv1_conv	False
3	<BatchNormalization name=conv1_bn, built=True>	conv1_bn	False
4	<Activation name=conv1_relu, built=True>	conv1_relu	False
...	...	...	...
423	<Conv2D name=conv5_block16_2_conv, built=True>	conv5_block16_2_conv	True
424	<Concatenate name=conv5_block16_concat, built=True>	conv5_block16_concat	True
425	<BatchNormalization name=bn, built=True>	bn	True
426	<Activation name=relu, built=True>	relu	True
427	<Flatten name=flatten_1, built=True>	flatten_1	True

428 rows × 3 columns

```
model_prebase.summary()
```

```
Model: "functional_12"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_2 (InputLayer)	(None, 224, 224, 3)	0	-
zero_padding2d_2 (ZeroPadding2D)	(None, 230, 230, 3)	0	input_layer_2[0][0]
conv1_conv (Conv2D)	(None, 112, 112, 64)	9,408	zero_padding2d_2[0][0]
conv1_bn (BatchNormalization)	(None, 112, 112, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 112, 112, 64)	0	conv1_bn[0][0]
zero_padding2d_3 (ZeroPadding2D)	(None, 114, 114, 64)	0	conv1_relu[0][0]
pool1 (MaxPooling2D)	(None, 56, 56, 64)	0	zero_padding2d_3[0][0]
conv2_block1_0_bn (BatchNormalization)	(None, 56, 56, 64)	256	pool1[0][0]
conv2_block1_0_relu (Activation)	(None, 56, 56, 64)	0	conv2_block1_0_bn[0][…
conv2_block1_1_conv (Conv2D)	(None, 56, 56, 128)	8,192	conv2_block1_0_relu[0…
conv2_block1_1_bn (BatchNormalization)	(None, 56, 56, 128)	512	conv2_block1_1_conv[0…
conv2_block1_1_relu (Activation)	(None, 56, 56, 128)	0	conv2_block1_1_bn[0][…
conv2_block1_2_conv (Conv2D)	(None, 56, 56, 32)	36,864	conv2_block1_1_relu[0…

Transfer Learning Fine Tuning con Data Augmentation DenseNet121

conv2_block1_concat (Concatenate)	(None, 56, 56, 96)	0	pool1[0][0], conv2_block1_2_conv[0…
conv2_block2_0_bn (BatchNormalization)	(None, 56, 56, 96)	384	conv2_block1_concat[0…
conv2_block2_0_relu (Activation)	(None, 56, 56, 96)	0	conv2_block2_0_bn[0][…
conv2_block2_1_conv (Conv2D)	(None, 56, 56, 128)	12,288	conv2_block2_0_relu[0…
conv2_block2_1_bn (BatchNormalization)	(None, 56, 56, 128)	512	conv2_block2_1_conv[0…
conv2_block2_1_relu (Activation)	(None, 56, 56, 128)	0	conv2_block2_1_bn[0][…
conv2_block2_2_conv (Conv2D)	(None, 56, 56, 32)	36,864	conv2_block2_1_relu[0…
conv2_block2_concat (Concatenate)	(None, 56, 56, 128)	0	conv2_block1_concat[0… conv2_block2_2_conv[0…
conv2_block3_0_bn (BatchNormalization)	(None, 56, 56, 128)	512	conv2_block2_concat[0…
conv2_block3_0_relu (Activation)	(None, 56, 56, 128)	0	conv2_block3_0_bn[0][…
conv2_block3_1_conv (Conv2D)	(None, 56, 56, 128)	16,384	conv2_block3_0_relu[0…
conv2_block3_1_bn (BatchNormalization)	(None, 56, 56, 128)	512	conv2_block3_1_conv[0…
conv2_block3_1_relu (Activation)	(None, 56, 56, 128)	0	conv2_block3_1_bn[0][…
conv2_block3_2_conv (Conv2D)	(None, 56, 56, 32)	36,864	conv2_block3_1_relu[0…
conv2_block3_concat (Concatenate)	(None, 56, 56, 160)	0	conv2_block2_concat[0… conv2_block3_2_conv[0…
conv2_block4_0_bn (BatchNormalization)	(None, 56, 56, 160)	640	conv2_block3_concat[0…
conv2_block4_0_relu (Activation)	(None, 56, 56, 160)	0	conv2_block4_0_bn[0][…
conv2_block4_1_conv (Conv2D)	(None, 56, 56, 128)	20,480	conv2_block4_0_relu[0…
conv2_block4_1_bn (BatchNormalization)	(None, 56, 56, 128)	512	conv2_block4_1_conv[0…
conv2_block4_1_relu (Activation)	(None, 56, 56, 128)	0	conv2_block4_1_bn[0][…
conv2_block4_2_conv (Conv2D)	(None, 56, 56, 32)	36,864	conv2_block4_1_relu[0…
conv2_block4_concat (Concatenate)	(None, 56, 56, 192)	0	conv2_block3_concat[0… conv2_block4_2_conv[0…
conv2_block5_0_bn (BatchNormalization)	(None, 56, 56, 192)	768	conv2_block4_concat[0…
conv2_block5_0_relu (Activation)	(None, 56, 56, 192)	0	conv2_block5_0_bn[0][…
conv2_block5_1_conv (Conv2D)	(None, 56, 56, 128)	24,576	conv2_block5_0_relu[0…
conv2_block5_1_bn (BatchNormalization)	(None, 56, 56, 128)	512	conv2_block5_1_conv[0…
conv2_block5_1_relu (Activation)	(None, 56, 56, 128)	0	conv2_block5_1_bn[0][…
conv2_block5_2_conv (Conv2D)	(None, 56, 56, 32)	36,864	conv2_block5_1_relu[0…
conv2_block5_concat (Concatenate)	(None, 56, 56, 224)	0	conv2_block4_concat[0… conv2_block5_2_conv[0…

conv3_block6_0_bn (BatchNormalization)	(None, 28, 28, 288)	1,152	conv3_block5_concat[0…
conv3_block6_0_relu (Activation)	(None, 28, 28, 288)	0	conv3_block6_0_bn[0][…
conv3_block6_1_conv (Conv2D)	(None, 28, 28, 128)	36,864	conv3_block6_0_relu[0…
conv3_block6_1_bn (BatchNormalization)	(None, 28, 28, 128)	512	conv3_block6_1_conv[0…
conv3_block6_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_block6_1_bn[0][…
conv3_block6_2_conv (Conv2D)	(None, 28, 28, 32)	36,864	conv3_block6_1_relu[0…
conv3_block6_concat (Concatenate)	(None, 28, 28, 320)	0	conv3_block5_concat[0… conv3_block6_2_conv[0…
conv3_block7_0_bn (BatchNormalization)	(None, 28, 28, 320)	1,280	conv3_block6_concat[0…
conv3_block7_0_relu (Activation)	(None, 28, 28, 320)	0	conv3_block7_0_bn[0][…
conv3_block7_1_conv (Conv2D)	(None, 28, 28, 128)	40,960	conv3_block7_0_relu[0…
conv3_block7_1_bn (BatchNormalization)	(None, 28, 28, 128)	512	conv3_block7_1_conv[0…
conv3_block7_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_block7_1_bn[0][…
conv3_block7_2_conv (Conv2D)	(None, 28, 28, 32)	36,864	conv3_block7_1_relu[0…
conv3_block7_concat (Concatenate)	(None, 28, 28, 352)	0	conv3_block6_concat[0… conv3_block7_2_conv[0…
conv3_block8_0_bn (BatchNormalization)	(None, 28, 28, 352)	1,408	conv3_block7_concat[0…
conv3_block8_0_relu (Activation)	(None, 28, 28, 352)	0	conv3_block8_0_bn[0][…
conv3_block8_1_conv (Conv2D)	(None, 28, 28, 128)	45,056	conv3_block8_0_relu[0…
conv3_block8_1_bn (BatchNormalization)	(None, 28, 28, 128)	512	conv3_block8_1_conv[0…
conv3_block8_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_block8_1_bn[0][…
conv3_block8_2_conv (Conv2D)	(None, 28, 28, 32)	36,864	conv3_block8_1_relu[0…
conv3_block8_concat (Concatenate)	(None, 28, 28, 384)	0	conv3_block7_concat[0… conv3_block8_2_conv[0…
conv3_block9_0_bn (BatchNormalization)	(None, 28, 28, 384)	1,536	conv3_block8_concat[0…
conv3_block9_0_relu (Activation)	(None, 28, 28, 384)	0	conv3_block9_0_bn[0][…
conv3_block9_1_conv (Conv2D)	(None, 28, 28, 128)	49,152	conv3_block9_0_relu[0…
conv3_block9_1_bn (BatchNormalization)	(None, 28, 28, 128)	512	conv3_block9_1_conv[0…
conv3_block9_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_block9_1_bn[0][…
conv3_block9_2_conv (Conv2D)	(None, 28, 28, 32)	36,864	conv3_block9_1_relu[0…

conv4_block9_concat (Concatenate)	(None, 14, 14, 544)	0	conv4_block8_concat[0… conv4_block9_2_conv[0…
conv4_block10_0_bn (BatchNormalization)	(None, 14, 14, 544)	2,176	conv4_block9_concat[0…
conv4_block10_0_relu (Activation)	(None, 14, 14, 544)	0	conv4_block10_0_bn[0]…
conv4_block10_1_conv (Conv2D)	(None, 14, 14, 128)	69,632	conv4_block10_0_relu[…
conv4_block10_1_bn (BatchNormalization)	(None, 14, 14, 128)	512	conv4_block10_1_conv[…
conv4_block10_1_relu (Activation)	(None, 14, 14, 128)	0	conv4_block10_1_bn[0]…
conv4_block10_2_conv (Conv2D)	(None, 14, 14, 32)	36,864	conv4_block10_1_relu[…
conv4_block10_concat (Concatenate)	(None, 14, 14, 576)	0	conv4_block9_concat[0… conv4_block10_2_conv[…
conv4_block11_0_bn (BatchNormalization)	(None, 14, 14, 576)	2,304	conv4_block10_concat[…
conv4_block11_0_relu (Activation)	(None, 14, 14, 576)	0	conv4_block11_0_bn[0]…
conv4_block11_1_conv (Conv2D)	(None, 14, 14, 128)	73,728	conv4_block11_0_relu[…
conv4_block11_1_bn (BatchNormalization)	(None, 14, 14, 128)	512	conv4_block11_1_conv[…
conv4_block11_1_relu (Activation)	(None, 14, 14, 128)	0	conv4_block11_1_bn[0]…
conv4_block11_2_conv (Conv2D)	(None, 14, 14, 32)	36,864	conv4_block11_1_relu[…
conv4_block11_concat (Concatenate)	(None, 14, 14, 608)	0	conv4_block10_concat[… conv4_block11_2_conv[…
conv4_block12_0_bn (BatchNormalization)	(None, 14, 14, 608)	2,432	conv4_block11_concat[…
conv4_block12_0_relu (Activation)	(None, 14, 14, 608)	0	conv4_block12_0_bn[0]…
conv4_block12_1_conv (Conv2D)	(None, 14, 14, 128)	77,824	conv4_block12_0_relu[…
conv4_block12_1_bn (BatchNormalization)	(None, 14, 14, 128)	512	conv4_block12_1_conv[…
conv4_block12_1_relu (Activation)	(None, 14, 14, 128)	0	conv4_block12_1_bn[0]…
conv4_block12_2_conv (Conv2D)	(None, 14, 14, 32)	36,864	conv4_block12_1_relu[…
conv4_block12_concat (Concatenate)	(None, 14, 14, 640)	0	conv4_block11_concat[… conv4_block12_2_conv[…
conv4_block13_0_bn (BatchNormalization)	(None, 14, 14, 640)	2,560	conv4_block12_concat[…
conv4_block13_0_relu (Activation)	(None, 14, 14, 640)	0	conv4_block13_0_bn[0]…
conv4_block13_1_conv (Conv2D)	(None, 14, 14, 128)	81,920	conv4_block13_0_relu[…
conv4_block13_1_bn (BatchNormalization)	(None, 14, 14, 128)	512	conv4_block13_1_conv[…
conv4_block13_1_relu (Activation)	(None, 14, 14, 128)	0	conv4_block13_1_bn[0]…

conv4_block13_2_conv (Conv2D)	(None, 14, 14, 32)	36,864	conv4_block13_1_relu[…
conv4_block13_concat (Concatenate)	(None, 14, 14, 672)	0	conv4_block12_concat[… conv4_block13_2_conv[…
conv4_block14_0_bn (BatchNormalization)	(None, 14, 14, 672)	2,688	conv4_block13_concat[…
conv4_block14_0_relu (Activation)	(None, 14, 14, 672)	0	conv4_block14_0_bn[0]…
conv4_block14_1_conv (Conv2D)	(None, 14, 14, 128)	86,016	conv4_block14_0_relu[…
conv4_block14_1_bn (BatchNormalization)	(None, 14, 14, 128)	512	conv4_block14_1_conv[…
conv4_block14_1_relu (Activation)	(None, 14, 14, 128)	0	conv4_block14_1_bn[0]…
conv4_block14_2_conv (Conv2D)	(None, 14, 14, 32)	36,864	conv4_block14_1_relu[…
conv4_block14_concat (Concatenate)	(None, 14, 14, 704)	0	conv4_block13_concat[… conv4_block14_2_conv[…
conv4_block15_0_bn (BatchNormalization)	(None, 14, 14, 704)	2,816	conv4_block14_concat[…
conv4_block15_0_relu (Activation)	(None, 14, 14, 704)	0	conv4_block15_0_bn[0]…
conv4_block15_1_conv (Conv2D)	(None, 14, 14, 128)	90,112	conv4_block15_0_relu[…
conv4_block15_1_bn (BatchNormalization)	(None, 14, 14, 128)	512	conv4_block15_1_conv[…
conv4_block15_1_relu (Activation)	(None, 14, 14, 128)	0	conv4_block15_1_bn[0]…
conv4_block15_2_conv (Conv2D)	(None, 14, 14, 32)	36,864	conv4_block15_1_relu[…
conv4_block15_concat (Concatenate)	(None, 14, 14, 736)	0	conv4_block14_concat[… conv4_block15_2_conv[…
conv4_block16_0_bn (BatchNormalization)	(None, 14, 14, 736)	2,944	conv4_block15_concat[…
conv4_block16_0_relu (Activation)	(None, 14, 14, 736)	0	conv4_block16_0_bn[0]…
conv4_block16_1_conv (Conv2D)	(None, 14, 14, 128)	94,208	conv4_block16_0_relu[…
conv4_block16_1_bn (BatchNormalization)	(None, 14, 14, 128)	512	conv4_block16_1_conv[…
conv4_block16_1_relu (Activation)	(None, 14, 14, 128)	0	conv4_block16_1_bn[0]…
conv4_block16_2_conv (Conv2D)	(None, 14, 14, 32)	36,864	conv4_block16_1_relu[…
conv4_block16_concat (Concatenate)	(None, 14, 14, 768)	0	conv4_block15_concat[… conv4_block16_2_conv[…
conv4_block17_0_bn (BatchNormalization)	(None, 14, 14, 768)	3,072	conv4_block16_concat[…
conv4_block17_0_relu (Activation)	(None, 14, 14, 768)	0	conv4_block17_0_bn[0]…
conv4_block17_1_conv (Conv2D)	(None, 14, 14, 128)	98,304	conv4_block17_0_relu[…
conv4_block17_1_bn (BatchNormalization)	(None, 14, 14, 128)	512	conv4_block17_1_conv[…
conv4_block17_1_relu (Activation)	(None, 14, 14, 128)	0	conv4_block17_1_bn[0]…

conv4_block17_2_conv (Conv2D)	(None, 14, 14, 32)	36,864	conv4_block17_1_relu[…
conv4_block17_concat (Concatenate)	(None, 14, 14, 800)	0	conv4_block16_concat[… conv4_block17_2_conv[…
conv4_block18_0_bn (BatchNormalization)	(None, 14, 14, 800)	3,200	conv4_block17_concat[…
conv4_block18_0_relu (Activation)	(None, 14, 14, 800)	0	conv4_block18_0_bn[0]…
conv4_block18_1_conv (Conv2D)	(None, 14, 14, 128)	102,400	conv4_block18_0_relu[…
conv4_block18_1_bn (BatchNormalization)	(None, 14, 14, 128)	512	conv4_block18_1_conv[…
conv4_block18_1_relu (Activation)	(None, 14, 14, 128)	0	conv4_block18_1_bn[0]…
conv4_block18_2_conv (Conv2D)	(None, 14, 14, 32)	36,864	conv4_block18_1_relu[…
conv4_block18_concat (Concatenate)	(None, 14, 14, 832)	0	conv4_block17_concat[… conv4_block18_2_conv[…
conv4_block19_0_bn (BatchNormalization)	(None, 14, 14, 832)	3,328	conv4_block18_concat[…
conv4_block19_0_relu (Activation)	(None, 14, 14, 832)	0	conv4_block19_0_bn[0]…
conv4_block19_1_conv (Conv2D)	(None, 14, 14, 128)	106,496	conv4_block19_0_relu[…
conv4_block19_1_bn (BatchNormalization)	(None, 14, 14, 128)	512	conv4_block19_1_conv[…
conv4_block19_1_relu (Activation)	(None, 14, 14, 128)	0	conv4_block19_1_bn[0]…
conv4_block19_2_conv (Conv2D)	(None, 14, 14, 32)	36,864	conv4_block19_1_relu[…
conv4_block19_concat (Concatenate)	(None, 14, 14, 864)	0	conv4_block18_concat[… conv4_block19_2_conv[…
conv4_block20_0_bn (BatchNormalization)	(None, 14, 14, 864)	3,456	conv4_block19_concat[…
conv4_block20_0_relu (Activation)	(None, 14, 14, 864)	0	conv4_block20_0_bn[0]…
conv4_block20_1_conv (Conv2D)	(None, 14, 14, 128)	110,592	conv4_block20_0_relu[…
conv4_block20_1_bn (BatchNormalization)	(None, 14, 14, 128)	512	conv4_block20_1_conv[…
conv4_block20_1_relu (Activation)	(None, 14, 14, 128)	0	conv4_block20_1_bn[0]…
conv4_block20_2_conv (Conv2D)	(None, 14, 14, 32)	36,864	conv4_block20_1_relu[…
conv4_block20_concat (Concatenate)	(None, 14, 14, 896)	0	conv4_block19_concat[… conv4_block20_2_conv[…
conv4_block21_0_bn (BatchNormalization)	(None, 14, 14, 896)	3,584	conv4_block20_concat[…
conv4_block21_0_relu (Activation)	(None, 14, 14, 896)	0	conv4_block21_0_bn[0]…
conv4_block21_1_conv (Conv2D)	(None, 14, 14, 128)	114,688	conv4_block21_0_relu[…
conv4_block21_1_bn (BatchNormalization)	(None, 14, 14, 128)	512	conv4_block21_1_conv[…

Transfer Learning Fine Tuning con Data Augmentation DenseNet121

conv4_block21_1_relu (Activation)	(None, 14, 14, 128)	0	conv4_block21_1_bn[0]…
conv4_block21_2_conv (Conv2D)	(None, 14, 14, 32)	36,864	conv4_block21_1_relu[…
conv4_block21_concat (Concatenate)	(None, 14, 14, 928)	0	conv4_block20_concat[… conv4_block21_2_conv[…
conv4_block22_0_bn (BatchNormalization)	(None, 14, 14, 928)	3,712	conv4_block21_concat[…
conv4_block22_0_relu (Activation)	(None, 14, 14, 928)	0	conv4_block22_0_bn[0]…
conv4_block22_1_conv (Conv2D)	(None, 14, 14, 128)	118,784	conv4_block22_0_relu[…
conv4_block22_1_bn (BatchNormalization)	(None, 14, 14, 128)	512	conv4_block22_1_conv[…
conv4_block22_1_relu (Activation)	(None, 14, 14, 128)	0	conv4_block22_1_bn[0]…
conv4_block22_2_conv (Conv2D)	(None, 14, 14, 32)	36,864	conv4_block22_1_relu[…
conv4_block22_concat (Concatenate)	(None, 14, 14, 960)	0	conv4_block21_concat[… conv4_block22_2_conv[…
conv4_block23_0_bn (BatchNormalization)	(None, 14, 14, 960)	3,840	conv4_block22_concat[…
conv4_block23_0_relu (Activation)	(None, 14, 14, 960)	0	conv4_block23_0_bn[0]…
conv4_block23_1_conv (Conv2D)	(None, 14, 14, 128)	122,880	conv4_block23_0_relu[…
conv4_block23_1_bn (BatchNormalization)	(None, 14, 14, 128)	512	conv4_block23_1_conv[…
conv4_block23_1_relu (Activation)	(None, 14, 14, 128)	0	conv4_block23_1_bn[0]…
conv4_block23_2_conv (Conv2D)	(None, 14, 14, 32)	36,864	conv4_block23_1_relu[…
conv4_block23_concat (Concatenate)	(None, 14, 14, 992)	0	conv4_block22_concat[… conv4_block23_2_conv[…
conv4_block24_0_bn (BatchNormalization)	(None, 14, 14, 992)	3,968	conv4_block23_concat[…
conv4_block24_0_relu (Activation)	(None, 14, 14, 992)	0	conv4_block24_0_bn[0]…
conv4_block24_1_conv (Conv2D)	(None, 14, 14, 128)	126,976	conv4_block24_0_relu[…
conv4_block24_1_bn (BatchNormalization)	(None, 14, 14, 128)	512	conv4_block24_1_conv[…
conv4_block24_1_relu (Activation)	(None, 14, 14, 128)	0	conv4_block24_1_bn[0]…
conv4_block24_2_conv (Conv2D)	(None, 14, 14, 32)	36,864	conv4_block24_1_relu[…
conv4_block24_concat (Concatenate)	(None, 14, 14, 1024)	0	conv4_block23_concat[… conv4_block24_2_conv[…
pool4_bn (BatchNormalization)	(None, 14, 14, 1024)	4,096	conv4_block24_concat[…
pool4_relu (Activation)	(None, 14, 14, 1024)	0	pool4_bn[0][0]
pool4_conv (Conv2D)	(None, 14, 14, 512)	524,288	pool4_relu[0][0]
pool4_pool (AveragePooling2D)	(None, 7, 7, 512)	0	pool4_conv[0][0]
conv5_block1_0_bn (BatchNormalization)	(None, 7, 7, 512)	2,048	pool4_pool[0][0]

Transfer Learning Fine Tuning con Data Augmentation DenseNet121

conv5_block1_0_relu (Activation)	(None, 7, 7, 512)	0	conv5_block1_0_bn[0][…
conv5_block1_1_conv (Conv2D)	(None, 7, 7, 128)	65,536	conv5_block1_0_relu[0…
conv5_block1_1_bn (BatchNormalization)	(None, 7, 7, 128)	512	conv5_block1_1_conv[0…
conv5_block1_1_relu (Activation)	(None, 7, 7, 128)	0	conv5_block1_1_bn[0][…
conv5_block1_2_conv (Conv2D)	(None, 7, 7, 32)	36,864	conv5_block1_1_relu[0…
conv5_block1_concat (Concatenate)	(None, 7, 7, 544)	0	pool4_pool[0][0], conv5_block1_2_conv[0…
conv5_block2_0_bn (BatchNormalization)	(None, 7, 7, 544)	2,176	conv5_block1_concat[0…
conv5_block2_0_relu (Activation)	(None, 7, 7, 544)	0	conv5_block2_0_bn[0][…
conv5_block2_1_conv (Conv2D)	(None, 7, 7, 128)	69,632	conv5_block2_0_relu[0…
conv5_block2_1_bn (BatchNormalization)	(None, 7, 7, 128)	512	conv5_block2_1_conv[0…
conv5_block2_1_relu (Activation)	(None, 7, 7, 128)	0	conv5_block2_1_bn[0][…
conv5_block2_2_conv (Conv2D)	(None, 7, 7, 32)	36,864	conv5_block2_1_relu[0…
conv5_block2_concat (Concatenate)	(None, 7, 7, 576)	0	conv5_block1_concat[0… conv5_block2_2_conv[0…
conv5_block3_0_bn (BatchNormalization)	(None, 7, 7, 576)	2,304	conv5_block2_concat[0…
conv5_block3_0_relu (Activation)	(None, 7, 7, 576)	0	conv5_block3_0_bn[0][…
conv5_block3_1_conv (Conv2D)	(None, 7, 7, 128)	73,728	conv5_block3_0_relu[0…
conv5_block3_1_bn (BatchNormalization)	(None, 7, 7, 128)	512	conv5_block3_1_conv[0…
conv5_block3_1_relu (Activation)	(None, 7, 7, 128)	0	conv5_block3_1_bn[0][…
conv5_block3_2_conv (Conv2D)	(None, 7, 7, 32)	36,864	conv5_block3_1_relu[0…
conv5_block3_concat (Concatenate)	(None, 7, 7, 608)	0	conv5_block2_concat[0… conv5_block3_2_conv[0…
conv5_block4_0_bn (BatchNormalization)	(None, 7, 7, 608)	2,432	conv5_block3_concat[0…
conv5_block4_0_relu (Activation)	(None, 7, 7, 608)	0	conv5_block4_0_bn[0][…
conv5_block4_1_conv (Conv2D)	(None, 7, 7, 128)	77,824	conv5_block4_0_relu[0…
conv5_block4_1_bn (BatchNormalization)	(None, 7, 7, 128)	512	conv5_block4_1_conv[0…
conv5_block4_1_relu (Activation)	(None, 7, 7, 128)	0	conv5_block4_1_bn[0][…
conv5_block4_2_conv (Conv2D)	(None, 7, 7, 32)	36,864	conv5_block4_1_relu[0…
conv5_block4_concat (Concatenate)	(None, 7, 7, 640)	0	conv5_block3_concat[0… conv5_block4_2_conv[0…

conv5_block5_0_bn (BatchNormalization)	(None, 7, 7, 640)	2,560	conv5_block4_concat[0…
conv5_block5_0_relu (Activation)	(None, 7, 7, 640)	0	conv5_block5_0_bn[0][…
conv5_block5_1_conv (Conv2D)	(None, 7, 7, 128)	81,920	conv5_block5_0_relu[0…
conv5_block5_1_bn (BatchNormalization)	(None, 7, 7, 128)	512	conv5_block5_1_conv[0…
conv5_block5_1_relu (Activation)	(None, 7, 7, 128)	0	conv5_block5_1_bn[0][…
conv5_block5_2_conv (Conv2D)	(None, 7, 7, 32)	36,864	conv5_block5_1_relu[0…
conv5_block5_concat (Concatenate)	(None, 7, 7, 672)	0	conv5_block4_concat[0… conv5_block5_2_conv[0…
conv5_block6_0_bn (BatchNormalization)	(None, 7, 7, 672)	2,688	conv5_block5_concat[0…
conv5_block6_0_relu (Activation)	(None, 7, 7, 672)	0	conv5_block6_0_bn[0][…
conv5_block6_1_conv (Conv2D)	(None, 7, 7, 128)	86,016	conv5_block6_0_relu[0…
conv5_block6_1_bn (BatchNormalization)	(None, 7, 7, 128)	512	conv5_block6_1_conv[0…
conv5_block6_1_relu (Activation)	(None, 7, 7, 128)	0	conv5_block6_1_bn[0][…
conv5_block6_2_conv (Conv2D)	(None, 7, 7, 32)	36,864	conv5_block6_1_relu[0…
conv5_block6_concat (Concatenate)	(None, 7, 7, 704)	0	conv5_block5_concat[0… conv5_block6_2_conv[0…
conv5_block7_0_bn (BatchNormalization)	(None, 7, 7, 704)	2,816	conv5_block6_concat[0…
conv5_block7_0_relu (Activation)	(None, 7, 7, 704)	0	conv5_block7_0_bn[0][…
conv5_block7_1_conv (Conv2D)	(None, 7, 7, 128)	90,112	conv5_block7_0_relu[0…
conv5_block7_1_bn (BatchNormalization)	(None, 7, 7, 128)	512	conv5_block7_1_conv[0…
conv5_block7_1_relu (Activation)	(None, 7, 7, 128)	0	conv5_block7_1_bn[0][…
conv5_block7_2_conv (Conv2D)	(None, 7, 7, 32)	36,864	conv5_block7_1_relu[0…
conv5_block7_concat (Concatenate)	(None, 7, 7, 736)	0	conv5_block6_concat[0… conv5_block7_2_conv[0…
conv5_block8_0_bn (BatchNormalization)	(None, 7, 7, 736)	2,944	conv5_block7_concat[0…
conv5_block8_0_relu (Activation)	(None, 7, 7, 736)	0	conv5_block8_0_bn[0][…
conv5_block8_1_conv (Conv2D)	(None, 7, 7, 128)	94,208	conv5_block8_0_relu[0…
conv5_block8_1_bn (BatchNormalization)	(None, 7, 7, 128)	512	conv5_block8_1_conv[0…
conv5_block8_1_relu (Activation)	(None, 7, 7, 128)	0	conv5_block8_1_bn[0][…
conv5_block8_2_conv (Conv2D)	(None, 7, 7, 32)	36,864	conv5_block8_1_relu[0…
conv5_block8_concat (Concatenate)	(None, 7, 7, 768)	0	conv5_block7_concat[0… conv5_block8_2_conv[0…

conv5_block9_0_bn (BatchNormalization)	(None, 7, 7, 768)	3,072	conv5_block8_concat[0…
conv5_block9_0_relu (Activation)	(None, 7, 7, 768)	0	conv5_block9_0_bn[0][…
conv5_block9_1_conv (Conv2D)	(None, 7, 7, 128)	98,304	conv5_block9_0_relu[0…
conv5_block9_1_bn (BatchNormalization)	(None, 7, 7, 128)	512	conv5_block9_1_conv[0…
conv5_block9_1_relu (Activation)	(None, 7, 7, 128)	0	conv5_block9_1_bn[0][…
conv5_block9_2_conv (Conv2D)	(None, 7, 7, 32)	36,864	conv5_block9_1_relu[0…
conv5_block9_concat (Concatenate)	(None, 7, 7, 800)	0	conv5_block8_concat[0… conv5_block9_2_conv[0…
conv5_block10_0_bn (BatchNormalization)	(None, 7, 7, 800)	3,200	conv5_block9_concat[0…
conv5_block10_0_relu (Activation)	(None, 7, 7, 800)	0	conv5_block10_0_bn[0]…
conv5_block10_1_conv (Conv2D)	(None, 7, 7, 128)	102,400	conv5_block10_0_relu[…
conv5_block10_1_bn (BatchNormalization)	(None, 7, 7, 128)	512	conv5_block10_1_conv[…
conv5_block10_1_relu (Activation)	(None, 7, 7, 128)	0	conv5_block10_1_bn[0]…
conv5_block10_2_conv (Conv2D)	(None, 7, 7, 32)	36,864	conv5_block10_1_relu[…
conv5_block10_concat (Concatenate)	(None, 7, 7, 832)	0	conv5_block9_concat[0… conv5_block10_2_conv[…
conv5_block11_0_bn (BatchNormalization)	(None, 7, 7, 832)	3,328	conv5_block10_concat[…
conv5_block11_0_relu (Activation)	(None, 7, 7, 832)	0	conv5_block11_0_bn[0]…
conv5_block11_1_conv (Conv2D)	(None, 7, 7, 128)	106,496	conv5_block11_0_relu[…
conv5_block11_1_bn (BatchNormalization)	(None, 7, 7, 128)	512	conv5_block11_1_conv[…
conv5_block11_1_relu (Activation)	(None, 7, 7, 128)	0	conv5_block11_1_bn[0]…
conv5_block11_2_conv (Conv2D)	(None, 7, 7, 32)	36,864	conv5_block11_1_relu[…
conv5_block11_concat (Concatenate)	(None, 7, 7, 864)	0	conv5_block10_concat[… conv5_block11_2_conv[…
conv5_block12_0_bn (BatchNormalization)	(None, 7, 7, 864)	3,456	conv5_block11_concat[…
conv5_block12_0_relu (Activation)	(None, 7, 7, 864)	0	conv5_block12_0_bn[0]…
conv5_block12_1_conv (Conv2D)	(None, 7, 7, 128)	110,592	conv5_block12_0_relu[…
conv5_block12_1_bn (BatchNormalization)	(None, 7, 7, 128)	512	conv5_block12_1_conv[…
conv5_block12_1_relu (Activation)	(None, 7, 7, 128)	0	conv5_block12_1_bn[0]…
conv5_block12_2_conv (Conv2D)	(None, 7, 7, 32)	36,864	conv5_block12_1_relu[…

Transfer Learning Fine Tuning con Data Augmentation DenseNet121

conv5_block12_concat (Concatenate)	(None, 7, 7, 896)	0	conv5_block11_concat[… conv5_block12_2_conv[…
conv5_block13_0_bn (BatchNormalization)	(None, 7, 7, 896)	3,584	conv5_block12_concat[…
conv5_block13_0_relu (Activation)	(None, 7, 7, 896)	0	conv5_block13_0_bn[0]…
conv5_block13_1_conv (Conv2D)	(None, 7, 7, 128)	114,688	conv5_block13_0_relu[…
conv5_block13_1_bn (BatchNormalization)	(None, 7, 7, 128)	512	conv5_block13_1_conv[…
conv5_block13_1_relu (Activation)	(None, 7, 7, 128)	0	conv5_block13_1_bn[0]…
conv5_block13_2_conv (Conv2D)	(None, 7, 7, 32)	36,864	conv5_block13_1_relu[…
conv5_block13_concat (Concatenate)	(None, 7, 7, 928)	0	conv5_block12_concat[… conv5_block13_2_conv[…
conv5_block14_0_bn (BatchNormalization)	(None, 7, 7, 928)	3,712	conv5_block13_concat[…
conv5_block14_0_relu (Activation)	(None, 7, 7, 928)	0	conv5_block14_0_bn[0]…
conv5_block14_1_conv (Conv2D)	(None, 7, 7, 128)	118,784	conv5_block14_0_relu[…
conv5_block14_1_bn (BatchNormalization)	(None, 7, 7, 128)	512	conv5_block14_1_conv[…
conv5_block14_1_relu (Activation)	(None, 7, 7, 128)	0	conv5_block14_1_bn[0]…
conv5_block14_2_conv (Conv2D)	(None, 7, 7, 32)	36,864	conv5_block14_1_relu[…
conv5_block14_concat (Concatenate)	(None, 7, 7, 960)	0	conv5_block13_concat[… conv5_block14_2_conv[…
conv5_block15_0_bn (BatchNormalization)	(None, 7, 7, 960)	3,840	conv5_block14_concat[…
conv5_block15_0_relu (Activation)	(None, 7, 7, 960)	0	conv5_block15_0_bn[0]…
conv5_block15_1_conv (Conv2D)	(None, 7, 7, 128)	122,880	conv5_block15_0_relu[…
conv5_block15_1_bn (BatchNormalization)	(None, 7, 7, 128)	512	conv5_block15_1_conv[…
conv5_block15_1_relu (Activation)	(None, 7, 7, 128)	0	conv5_block15_1_bn[0]…
conv5_block15_2_conv (Conv2D)	(None, 7, 7, 32)	36,864	conv5_block15_1_relu[…
conv5_block15_concat (Concatenate)	(None, 7, 7, 992)	0	conv5_block14_concat[… conv5_block15_2_conv[…
conv5_block16_0_bn (BatchNormalization)	(None, 7, 7, 992)	3,968	conv5_block15_concat[…
conv5_block16_0_relu (Activation)	(None, 7, 7, 992)	0	conv5_block16_0_bn[0]…
conv5_block16_1_conv (Conv2D)	(None, 7, 7, 128)	126,976	conv5_block16_0_relu[…
conv5_block16_1_bn (BatchNormalization)	(None, 7, 7, 128)	512	conv5_block16_1_conv[…
conv5_block16_1_relu (Activation)	(None, 7, 7, 128)	0	conv5_block16_1_bn[0]…
conv5_block16_2_conv (Conv2D)	(None, 7, 7, 32)	36,864	conv5_block16_1_relu[…

conv5_block16_concat (Concatenate)	(None, 7, 7, 1024)	0	conv5_block15_concat[_ conv5_block16_2_conv[_
bn (BatchNormalization)	(None, 7, 7, 1024)	4,096	conv5_block16_concat[_
relu (Activation)	(None, 7, 7, 1024)	0	bn[0][0]
flatten_1 (Flatten)	(None, 50176)	0	relu[0][0]

Total params: 7,037,504 (26.85 MB)

Trainable params: 639,616 (2.44 MB)

Non-trainable params: 6,397,888 (24.41 MB)

```
model_post = ks.Sequential()
# Definir la capa de entrada con la forma de las imágenes de CIFAR-10
model_post.add(ks.layers.InputLayer(input_shape=(32,32,3),))
# Agregar una capa de redimensionamiento para cambiar el tamaño a (224, 224, 3)
model_post.add(ks.layers.Resizing(224,224,interpolation="bilinear"))
# Agregar model_prebase
model_post.add(model_prebase)
#Red de classificacion
# Capas de clasificación
# Primera capa
model_post.add(ks.layers.Dense(256,
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model_post.add(ks.layers.BatchNormalization())
model_post.add(ks.layers.ReLU())
model_post.add(ks.layers.Dropout(0.4))
# Segunda capa
model_post.add(ks.layers.Dense(128,
kernel_initializer=initializers.HeUniform(),kernel_regularizer=regularizers.l2(1e-4)))
model_post.add(ks.layers.BatchNormalization())
model_post.add(ks.layers.ReLU())
model_post.add(ks.layers.Dropout(0.4))
# Capa de salida
model_post.add(ks.layers.Dense(10, activation='softmax'))
```

```
model_post.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
resizing_1 (Resizing)	(None, 224, 224, 3)	0
functional_12 (Functional)	(None, 50176)	7,037,504
dense_3 (Dense)	(None, 256)	12,845,312
batch_normalization_2 (BatchNormalization)	(None, 256)	1,024
re_lu_2 (ReLU)	(None, 256)	0
dropout_2 (Dropout)	(None, 256)	0
dense_4 (Dense)	(None, 128)	32,896
batch_normalization_3 (BatchNormalization)	(None, 128)	512
re_lu_3 (ReLU)	(None, 128)	0
dropout_3 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 10)	1,290

Total params: 19,918,538 (75.98 MB)

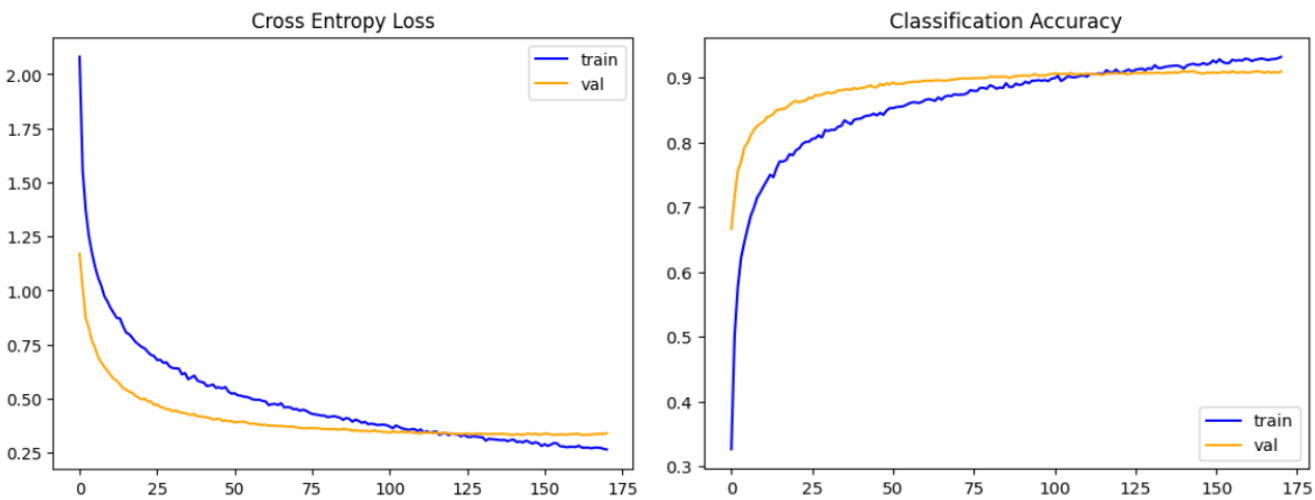
Trainable params: 13,519,882 (51.57 MB)

Non-trainable params: 6,398,656 (24.41 MB)

Transfer Learning Fine Tuning con Data Augmentation DenseNet121

```
# Model compile con learning rate bajo para poder mejorar el aprendizaje
model_post.compile(optimizer=Adam(learning_rate=1e-5),
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])

# Generación de imágenes distorsionadas
train_datagen = ImageDataGenerator(
   rotation_range=10,
   width_shift_range=0.1,
   height_shift_range=0.1,
   horizontal_flip=True,
   brightness_range=(0.9, 1.2),
   rescale=1./255,)
train_generator = train_datagen.flow(x_train_scaled,y_train, batch_size=128)
validation_datagen = ImageDataGenerator(rescale=1./255)
validation_generator = validation_datagen.flow( x_val_scaled,y_val,batch_size=128)

# Callbacks
callback_val_loss=EarlyStopping(monitor="val_loss",patience=30,restore_best_weights=True)
callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=30, restore_best_weights=True)

# Epochs
epochs=500
# Model fit
history = model_post.fit(train_generator, epochs=epochs, validation_data=validation_generator,
   steps_per_epoch=200,validation_steps=200, callbacks=[callback_val_loss, callback_val_accuracy])

# Results
_, acc = model_post.evaluate(x_test_scaled, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
```

> 90.370 Se consigue el objetivo de obtener más de 90% en el Test.



Transfer Learning de DenseNet 121 con Fine Tuning y Data Augmentation.

Comportamiento del entrenamiento:

Hay un crecimiento constante en la Accuracy a medida que avanzan las épocas, alcanzando más del 90% al final del entrenamiento. Esto indica que el modelo es capaz de aprender de los datos de entrenamiento de manera efectiva.

Comportamiento de la validación:

La Accuracy de validación es más alta que la de entrenamiento inicialmente, lo que podría indicar un modelo regularizado con buena inicialización, mostrando cierta estabilidad con despreciables fluctuaciones. La curva de validación se estabiliza alrededor de la época 50,sigue creciente lentamente y alcanza valores mayores del 90%. La validación muestra un buen ajuste del modelo, ya que no hay señales de sobreajuste evidente.

Comparación entre entrenamiento y validación:

Brecha inicial: Hay una diferencia significativa en las primeras épocas, con la precisión de validación siendo más alta. Esto se explicaría la situación cuando el modelo aún no ha aprendido adecuadamente los patrones del conjunto de entrenamiento.

Convergencia: A partir de la época 75, las curvas comienzan a converger, mostrando que el modelo generaliza bien y no está sobreajustando los datos de entrenamiento. Al final del entrenamiento, las precisiones de entrenamiento y validación son casi iguales (mayores al 90%).

Razón del éxito:

Esta red ha permitido alcanzar más del 90% de precisión gracias a la combinación de:

1.Transfer Learning: Uso de DenseNet 121 preentrenada para aprovechar características previamente aprendidas en otro dominio.

2.Fine Tuning: Ajuste de las últimas 30 capas de la red para adaptarla al nuevo conjunto de datos.

3.Data Augmentation: Aplicación de técnicas no muy agresivas, lo que evita distorsionar en exceso las imágenes y permite aumentar la generalización del modelo.

Resumen:

La implementación de Transfer Learning con Fine Tuning y Data Augmentation permitió alcanzar más del 90% de precisión en el conjunto de test, logrando un modelo eficiente y bien generalizado.