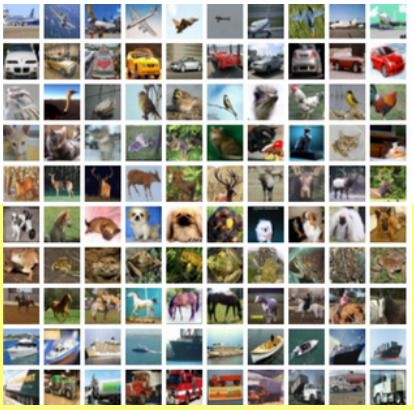
NUCLIO DIGITAL SCHOOL

FEBRERO DE 2023

PROYECTO CIFAR10



Realizado por: Vigee Carolina Rojas Briceño

PROYECTO 1

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(64, activation='relu'))
model.add(ks.layers.Dense(10, activation='softmax'))
```

De acuerdo a lo comentado en clases he agragado otra capa convolucional de 64 para mejorar su precisión. Estas capas convolucionales extraen características de los datos de entrada mediante filtros especiales. Además aumenté el Dense a 64.

ACCURACY:

```
Total params: 282,250
Trainable params: 282,250
Non-trainable params: 0
```

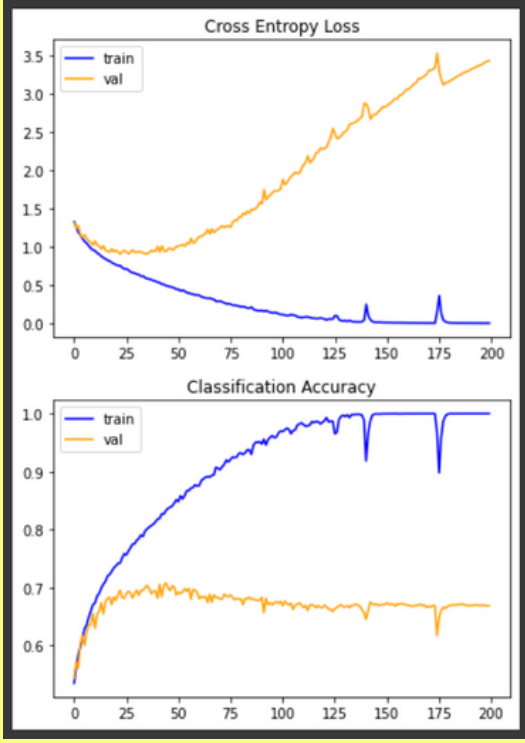
Implementando este aumento de capas hemos conseguido incrementar el accuracy a 65.740, lo cual nos indica que el modelo empieza a predecir. También hemos aumentado los epochs a 200.
Partiendo de lo anterior empezaré a implementar regularizaciones a los modelos para que no haga overfitting.

```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    batch_size= 512, validation_data=(x_val_scaled, y_val))
```

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

```
> 65.740
```

```
plt.title('Cross Entropy Loss')
plt.plot(history.history['loss'], color='blue', label='train')
plt.plot(history.history['val_loss'], color='orange', label='val')
plt.legend()
plt.show()

plt.title('Classification Accuracy')
plt.plot(history.history['accuracy'], color='blue', label='train')
plt.plot(history.history['val_accuracy'], color='orange', label='val')
plt.legend()
plt.show()
```



PROYECTO 2

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))

model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))

model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.Dense(10, activation='softmax'))
```

En este proyecto he aumentado el número de capas, junto con una capa Dense con 128 neuronas, para mejorar el rendimiento del modelo. Realicé esto con la finalidad de la profundidad de la red para que tenga mejor capacidad de aprendizaje, es decir, un mayor número de características y una mejor capacidad para detectar patrones en los datos.

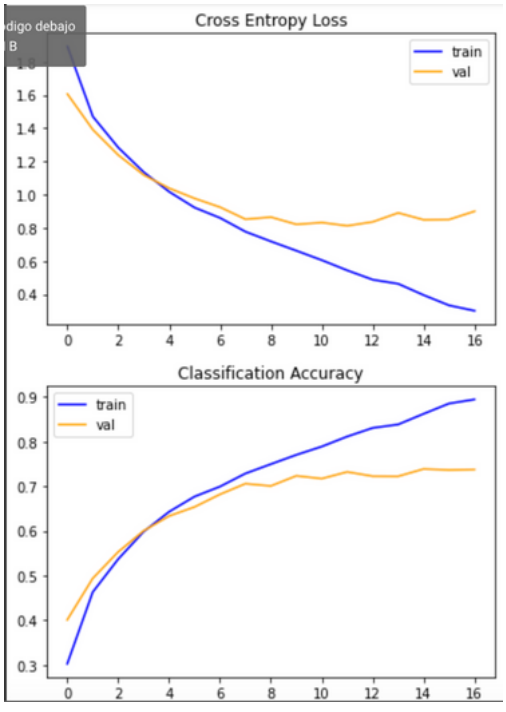
```
=================================================================
Total params: 550,570
Trainable params: 550,570
Non-trainable params: 0
_________________________________________________________________
```

ACCURACY:

```
history = model.fit(x_train_scaled, y_train, batch_size= 512,
          epochs=200, use_multiprocessing=False,
          callbacks=[callback_modelcheckpoint, callback_loss, callback_accuracy],
          validation_data=(x_val_scaled, y_val))
```

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

```
> 72.860
```



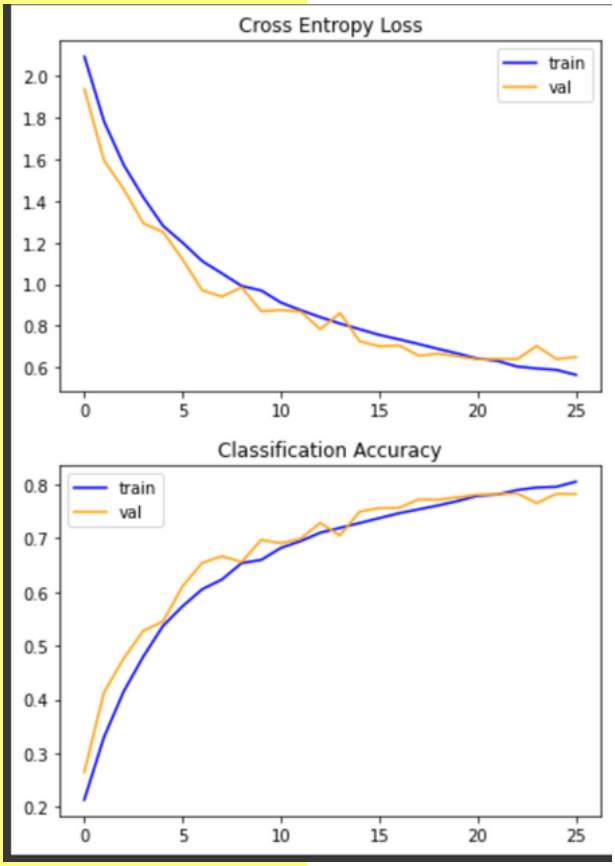
Con los cambios anteriormente mencionados he conseguido un accuracy de 72.860 esto significa que el modelo está funcionando bien, pero se puede mejorar para obtener un mejor accuracy. Por lo que se puede observar en los gráficos las lineas empiezan a separarse dando lugar a un sobreajuste en el que se evidencia que el modelo a pesar de predecir mejor los datos de train no lo está haciendo así de bien con los datos de validación.

PROYECTO 3:

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.Dropout(0.3))

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

```
Total params: 567,082
Trainable params: 567,082
Non-trainable params: 0
```

ACCURACY:




```
history = model.fit(x_train_scaled, y_train, batch_size= 512,
        epochs=200, use_multiprocessing=False,
        callbacks=[callback_modelcheckpoint, callback_loss, callback_accuracy],
        validation_data=(x_val_scaled, y_val))
```

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

```
> 77.540
```

En este proyecto se hicieron varios cambios a la arquitectura de la red. Se agregaron dos capas de Dropout para reducir el sobreajuste, añadiendo ruido a la señal lo que permite regularizar y estabilizar la red neuronal. Además, se agregaron dos capas Densas con 128 neuronas, cada una con una activación relu y una capa de Dropout. Esto se hizo para proporcionar una mayor profundidad a la red, lo que ayudará a la red a aprender mejor características de los datos y mejorar el rendimiento de la red. En el gráfico se evidencia que a diferencia del proyecto anterior las líneas ya no se separan.

PROYECTO 4:

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.BatchNormalization())

model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.BatchNormalization())

model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.BatchNormalization())

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.BatchNormalization())

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

En este proyecto decidí agregar capas de Batch Normalization entre las capas convolucionales y dense. Esto también contribuye a mejorar el rendimiento de la red al reducir el sobreajuste.

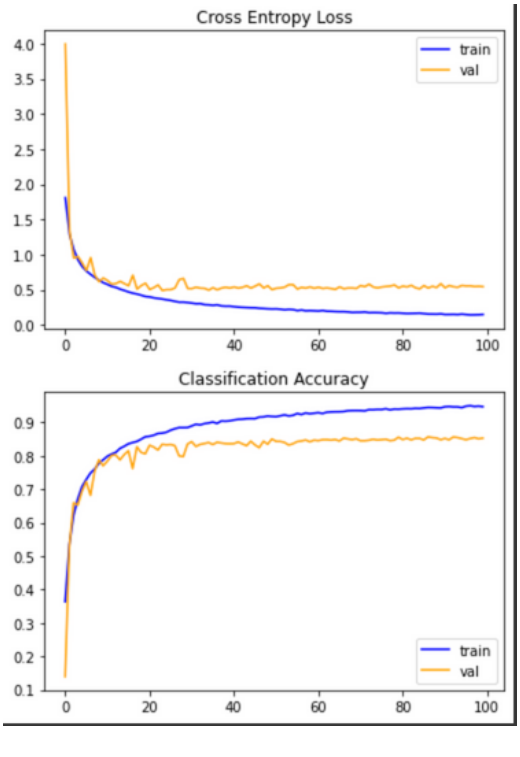
ACCURACY:

```
history = model.fit(x_train_scaled, y_train, batch_size= 128,
          epochs=100, use_multiprocessing=False,
          validation_data=(x_val_scaled, y_val))
```

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

```
> 83.320
```

Reduje el número de epochs de 200 a 100 esto para reducir el tiempo de entrenamiento sin afectar significativamente el rendimiento del modelo. También se redujo el tamaño de batch_size de 512 a 128 lo que también ayuda a reducir el tiempo de entrenamiento sin embargo se puede observar en el gráfico que las líneas se separan un poco al realizar este cambio. En este proyecto hemos conseguido un 83.320 de accuracy.



PROYECTO 5:

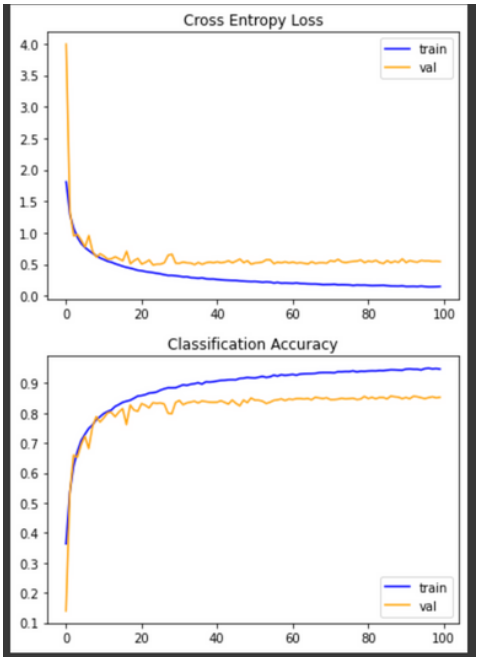
```
model = ks.Sequential()

model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.BatchNormalization())

model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.BatchNormalization())

model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.BatchNormalization())

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(256, activation='relu'))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.BatchNormalization())

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

ACCURACY:

```
history = model.fit(x_train_scaled, y_train, batch_size= 128,
          epochs=100, use_multiprocessing=False,
          callbacks=[callback_modelcheckpoint, callback_loss, callback_accuracy],
          validation_data=(x_val_scaled, y_val))
```

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

```
> 83.360
```



En este proyecto decidí sólo realizar cambios en el Dense, pasarlo de 128 a 256, esto se hizo para aumentar la profundidad de la red, lo que significa que la red tendrá una mejor capacidad de aprendizaje, es decir, un mayor número de características y una mejor capacidad para detectar patrones en los datos. Podemos observar en este gráficos que se suavizan un poco mas los picos a pesar de sólo haber conseguido aumentar mínimamente el accuracy.

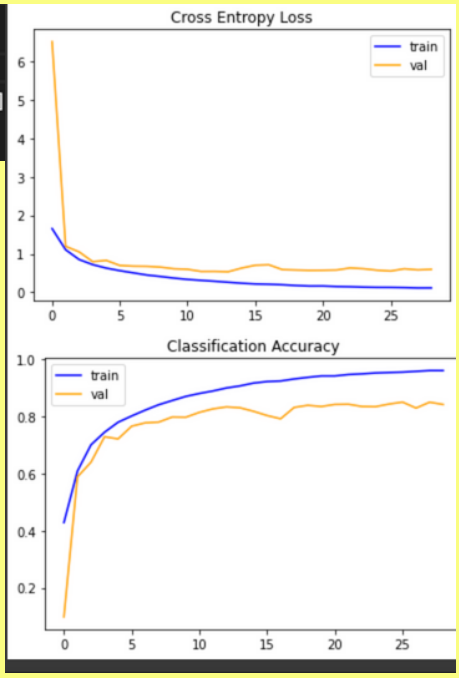
PROYECTO 6:

```
model = ks.Sequential()


model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.BatchNormalization())

model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.BatchNormalization())

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.BatchNormalization())


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


model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.BatchNormalization())
```

ACCURACY:

```
history = model.fit(x_train_scaled, y_train, batch_size= 128,
        epochs=100, use_multiprocessing=False,
        callbacks=[callback_modelcheckpoint, callback_loss, callback_accuracy],
        validation_data=(x_val_scaled, y_val))
```

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

     > 83.390
```



Los cambios realizados en el proyecto 6 consisten en agregar capas adicionales a la arquitectura de la red neuronal. Se agregaron cuatro capas convolucionales adicionales, cada una con 64, 128 y 256 filtros respectivamente. Estas capas ayudan a evitar el sobreajuste del modelo y mejorar la generalización.

PROYECTO 7:

```
model = ks.Sequential()


model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.BatchNormalization())

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.BatchNormalization())

model.add(ks.layers.Conv2D(512, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.Conv2D(512, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.BatchNormalization())


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


model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.BatchNormalization())

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

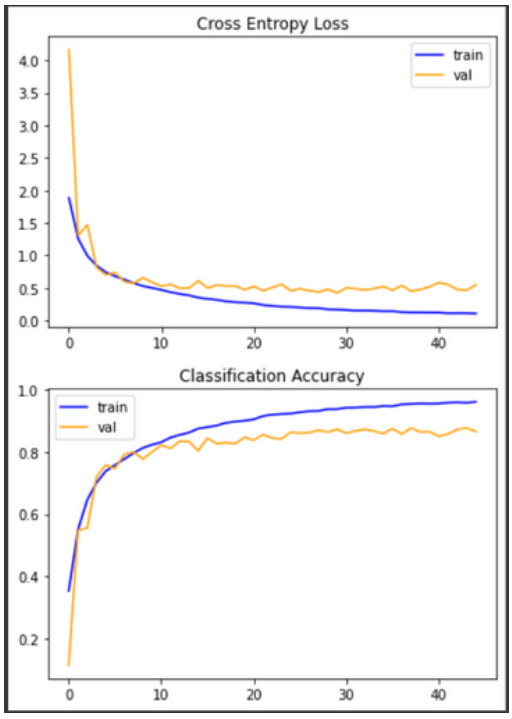
Se agregaron tres capas convolucionales adicionales, cada una con 128, 256 y 512 filtros respectivamente. También se agregaron capas de Dropout las cuales aumenté del 0.3 a 0.5 esto ayuda a reducir el sobreajuste del modelo, ya que la capa de Dropout elimina aleatoriamente algunas de las neuronas de la red para evitar que las mismas se ajusten demasiado a los datos de entrenamiento.

```
=================================================================
Total params: 8,781,962
Trainable params: 8,779,146
Non-trainable params: 2,816
```

ACCURACY:

```
[ ] history = model.fit(x_train_scaled, y_train, batch_size= 128,
            epochs=100, use_multiprocessing=False,
            callbacks=[callback_modelcheckpoint, callback_loss, callback_accuracy],
            validation_data=(x_val_scaled, y_val))
```

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

    > 85.950
```



PROYECTO 8:

cifar10_V8_model.ipynb

Notebook Data Logs Comments (0) Settings

In [3]:

```
model = ks.Sequential()


model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.BatchNormalization())

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.BatchNormalization())

model.add(ks.layers.Conv2D(512, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.Conv2D(512, (3, 3), strides=1, activation='relu',
                           padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.BatchNormalization())


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


model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.BatchNormalization())

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

En este proyecto me he enfocado en modificar el optimizador del modelo. Se ha aplicado un learning rate de 0.001, es capaz de ajustar la tasa de aprendizaje de forma dinámica para llegar a los mejores resultados.

In [5]:

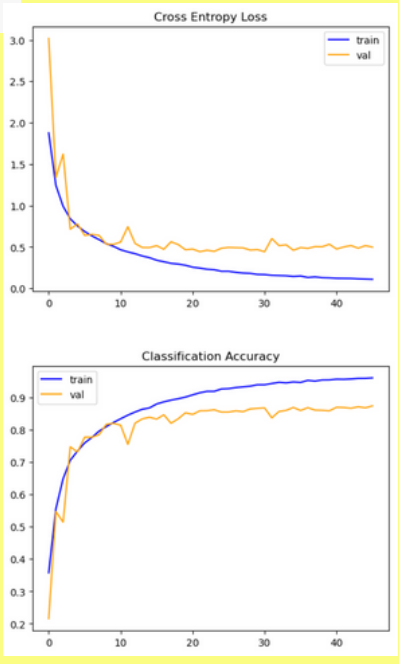
```
opt = keras.optimizers.Adam(learning_rate=0.001)
model.compile(loss='sparse_categorical_crossentropy', metrics=['accuracy'], optimizer=opt)
# model.compile(loss='Binary_Crossentropy', metrics=['accuracy'], optimizer=opt)
```

ACCURACY:

```
history = model.fit(x_train_scaled, y_train, batch_size= 128,
          epochs=100, use_multiprocessing=False,
          callbacks=[callback_modelcheckpoint, callback_loss, callback_accuracy],
          validation_data=(x_val_scaled, y_val))
```

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

```
> 87.150
```



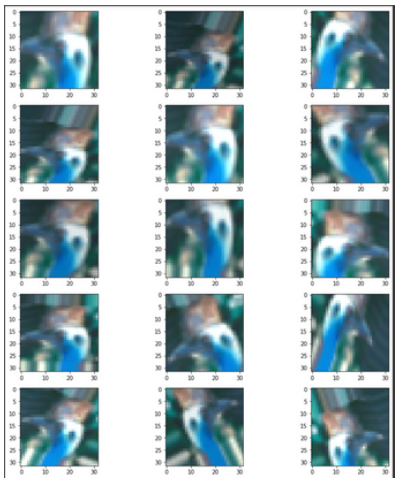
Con este cambio he conseguido un 87.150 de accuracy, sin embargo vemos que hay un gap entre train y val, Para reducir el gap entre el entrenamiento y la validación, podemos intentar disminuir el tamaño del lote y agregar elementos de regularización y normalización por lotes. Esto podría ayudar a mejorar la precisión de la red al reducir el sobreajuste del modelo y aumentar la generalización

PROYECTO 9:

```
[ ] model = ks.Sequential()


    model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu', padding='same', input_shape=(32,32,3)))
    model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
    model.add(ks.layers.MaxPooling2D((2, 2)))
    model.add(ks.layers.Dropout(0.3))
    model.add(ks.layers.BatchNormalization())

    model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
    model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
    model.add(ks.layers.MaxPooling2D((2, 2)))
    model.add(ks.layers.Dropout(0.3))
    model.add(ks.layers.BatchNormalization())

    model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same'))
    model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same'))
    model.add(ks.layers.MaxPooling2D((2, 2)))
    model.add(ks.layers.Dropout(0.3))
    model.add(ks.layers.BatchNormalization())


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


    model.add(ks.layers.Dense(512, activation='relu'))
    model.add(ks.layers.Dropout(0.3))
    model.add(ks.layers.BatchNormalization())

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

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range = 0.5,
    zoom_range = 0.3,
    horizontal_flip = True,
    brightness_range = (0.9, 1.3),
    width_shift_range = 0.2,
    rotation_range = 30

    )

train_generator = train_datagen.flow(
    x_train,  # Aquí hay que usar datos NO re-escalados
    y_train,
    batch_size=256
)
```

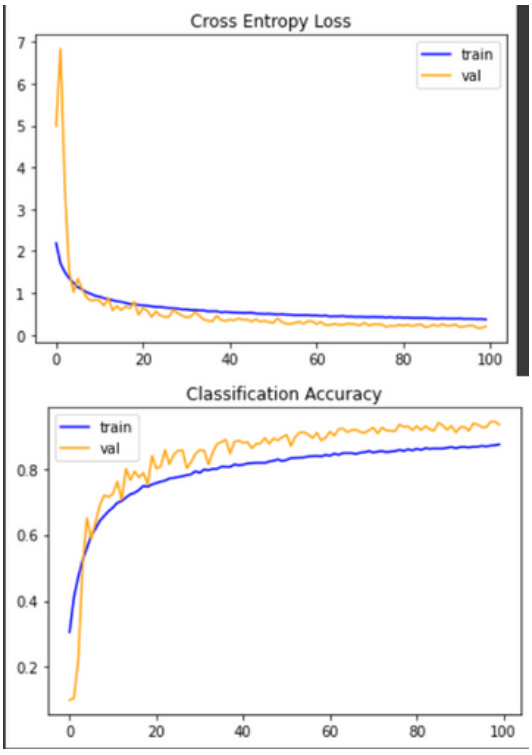
```
validation_datagen = ImageDataGenerator(
    rescale=1./255
    )
validation_generator = validation_datagen.flow(
    x_val,
    y_val,
    batch_size=128
)
```



ACCURACY:

```
history = model.fit(train_generator,
        epochs=100, use_multiprocessing=False,
        callbacks=[callback_modelcheckpoint, callback_loss, callback_accuracy],
        validation_data=validation_generator, steps_per_epoch = 156,
              validation_steps = 78)
```

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

    > 88.370
```



En este proyecto he aplicado las técnicas de data augmentation impartidas en clases, lo que creará versiones modificadas de las imágenes originales con lo que lograremos mejorar la generalización del modelo y a prevenir el sobreajuste.

PROYECTO 10:

```
output_from_vgg16 = 0
```

```
model_vgg16_final.output
```

```
<KerasTensor: shape=(None, 512) dtype=float32 (created by layer 'flatten_1')>
```

```
model_post_vgg = ks.Sequential()

model_post_vgg.add(ks.layers.InputLayer(input_shape=(512)))
model_post_vgg.add(ks.layers.Dense(1024, activation='relu', input_shape=(512,)))
model_post_vgg.add(ks.layers.Dropout(0.5))
model_post_vgg.add(ks.layers.Dense(1024, activation='relu'))
model_post_vgg.add(ks.layers.Dropout(0.5))
model_post_vgg.add(ks.layers.Dense(512, activation='relu'))
model_post_vgg.add(ks.layers.Dropout(0.5))
model_post_vgg.add(ks.layers.Dense(10, activation='softmax'))
```

En el último proyecto he implementado técnicas avanzadas de transfer learning aplicaremos una red neuronal pre -entrenada llamada VGG16

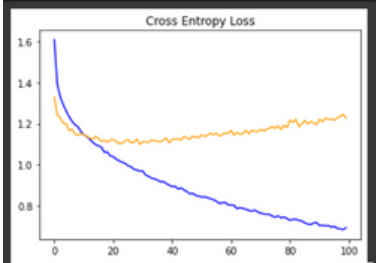
ACCURACY:

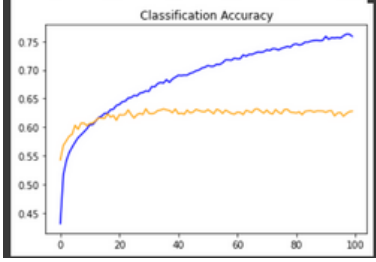
```
history = model_post_vgg.fit(x=x_train_postvgg16, y=y_train_encoded, batch_size=64,
                             epochs=100,
                             validation_data=(x_val_postvgg16, y_val_encoded))
```

```
_, acc = model_post_vgg.evaluate(x_test_postvgg16, y_test_encoded, verbose=0)
print('Modelo con Basic Transfer Learning > %.3f' % (acc * 100.0))
```

```
Modelo con Basic Transfer Learning > 61.790
```

```
plt.title('Cross Entropy Loss')
plt.plot(history.history['loss'], color='blue', label='train
plt.plot(history.history['val_loss'], color='orange', label=
plt.show()

plt.title('Classification Accuracy')
plt.plot(history.history['accuracy'], color='blue', label='t
plt.plot(history.history['val_accuracy'], color='orange', la
plt.show()
```





Implementando VGG16 no hemos logrado mejorar nuestro modelo al contrario hemos empeorado este, esto puede deberse a que no he dado con la configuración correcta y por ello el modelo se sobreajusta, lo que resulta en una precisión peor que la del modelo sin VGG16.