INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO, ITAM
Curso: Visión por Computadora

Tarea 5

Pablo Barranco Soto 151528
Sebastián Martínez Santos 176357

Tras hacer varias búsquedas con grid search, cambiando los hiperparametros *funciones de activación*, *batch size*, *epochs*, *loss function*, *número de layers*. Decidimos acomodar nuestros mejores modelos con respecto.

```
for a in activaciones:
  for n in number_of_layers:
    AE = Sequential()
    AE.add(Input(shape=(32, 32, 3)))
    for i in range(n):
      AE.add(Conv2D(8, 3, strides=(2, 2), padding='same', activation=a))
    for i in range(n-1):
      AE.add(Conv2DTranspose(8, 3, strides=(2, 2), padding='same', activation=a))
    AE.add(Conv2DTranspose(3, 3, strides=(2, 2), padding='same', activation='sigmoid'))
    # Compilacion del Modelo
    for l in losses:
      for b in batch_sizes:
        for e in epochs:
          AE.compile(optimizer='adam', loss=l,metrics = ['mse','acc'] )
          h = AE.fit(x=X_train, y=Y_train, batch_size=b, epochs=e, verbose=0, validation_split = 0.2)
          print("{} %".format( round((iteracion/324) *100,2)))
          iteracion = iteracion + 1

          # Guardandamos los resultados
          mse_train.append(h.history['mse'][-1]) # Accuracy entrenamiento
          acc_train.append(h.history['acc'][-1]) # Accuracy validación
          loss_train.append(h.history['loss'][-1]) # Accuracy validación
```

Mejores modelos:

Modelo	Activación	Batch Size	Epoch	Loss Function	N. Layers	Loss
1	ReLu y Sigmoid	500	150	MeanSquaredError	4	0.0060
2	ReLu y Sigmoid	500	50	mean_squared_logarithmic_error	4	0.0038
3	ReLu y Sigmoid	400	150	MeanSquaredError	4	0.0065
4	ReLu y Sigmoid	500	150	MeanSquaredLogarithmicError	4	0.0032
5	Relu y Sigmoid	800	200	MeanSquaredLogarithmicError	4	0.0028
6	SeLu y Sigmoid	500	50	mean_squared_logarithmic_error	4	0.0049
7	SeLu y Sigmoid	500	10	mean_squared_logarithmic_error	4	0.0077

Modelo 1

```
[ ] # Define model
    AE = Sequential()
    AE.add(Input(shape=(32, 32, 3)))
    AE.add(Conv2D(8, 3, strides=(2, 2), padding='same', activation='relu'))
    AE.add(Conv2D(8, 3, strides=(2, 2), padding='same', activation='relu'))
    AE.add(Conv2DTranspose(8, 3, strides=(2, 2), padding='same', activation='relu'))
    AE.add(Conv2DTranspose(3, 3, strides=(2, 2), padding='same', activation='sigmoid'))

    AE.summary()
```

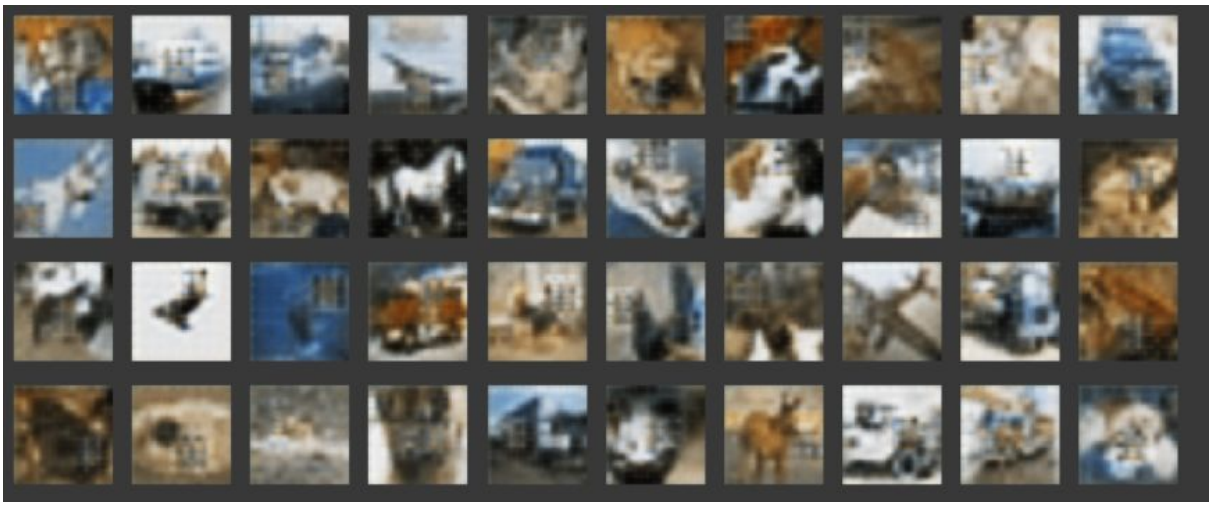
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 16, 16, 8)         224
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 8, 8, 8)           584
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 16, 16, 8)         584
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 32, 32, 3)         219
=================================================================
Total params: 1,611
Trainable params: 1,611
Non-trainable params: 0
_________________________________________________________________
```

```
[ ] # Compile and train
    AE.compile(optimizer='adam', loss=tf.keras.losses.MeanSquaredError())
    h = AE.fit(x=x_train, y=y_train, batch_size=500, epochs=150, validation_split=0.2)
```

Modelo 2

```
AE = Sequential()
AE.add(Input(shape=(32, 32, 3)))
AE.add(Conv2D(8, 3, strides=(2, 2), padding='same', activation='relu'))
AE.add(Conv2D(8, 3, strides=(2, 2), padding='same', activation='relu'))
AE.add(Conv2DTranspose(8, 3, strides=(2, 2), padding='same', activation='relu'))
AE.add(Conv2DTranspose(3, 3, strides=(2, 2), padding='same', activation='sigmoid'))

#Compile
AE.compile(optimizer='Adam', loss='mean_squared_logarithmic_error')
h = AE.fit(x=x_train, y=y_train, batch_size=500, epochs=50, validation_split=0,verbose=0)
h.history['loss'][-1]
```



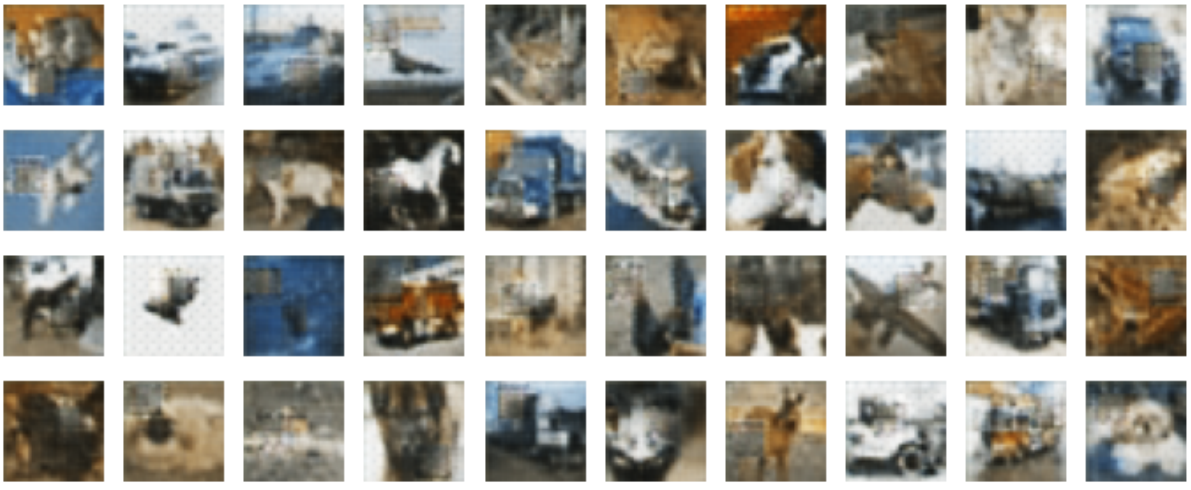
Modelo 3

```
[11] # Define model
     AE = Sequential()
     AE.add(Input(shape=(32, 32, 3)))
     AE.add(Conv2D(8, 3, strides=(2, 2), padding='same', activation='relu'))
     AE.add(Conv2D(8, 3, strides=(2, 2), padding='same', activation='relu'))
     AE.add(Conv2DTranspose(8, 3, strides=(2, 2), padding='same', activation='relu'))
     AE.add(Conv2DTranspose(3, 3, strides=(2, 2), padding='same', activation='sigmoid'))

     AE.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 16, 16, 8)         224
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 8, 8, 8)           584
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 16, 16, 8)         584
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 32, 32, 3)         219
=================================================================
Total params: 1,611
Trainable params: 1,611
Non-trainable params: 0
_________________________________________________________________
```

```
[22] # Compile and train
     AE.compile(optimizer='adam', loss=tf.keras.losses.MeanSquaredError())
     h = AE.fit(x=x_train, y=y_train, batch_size=400, epochs=150, validation_split=0.2)
```



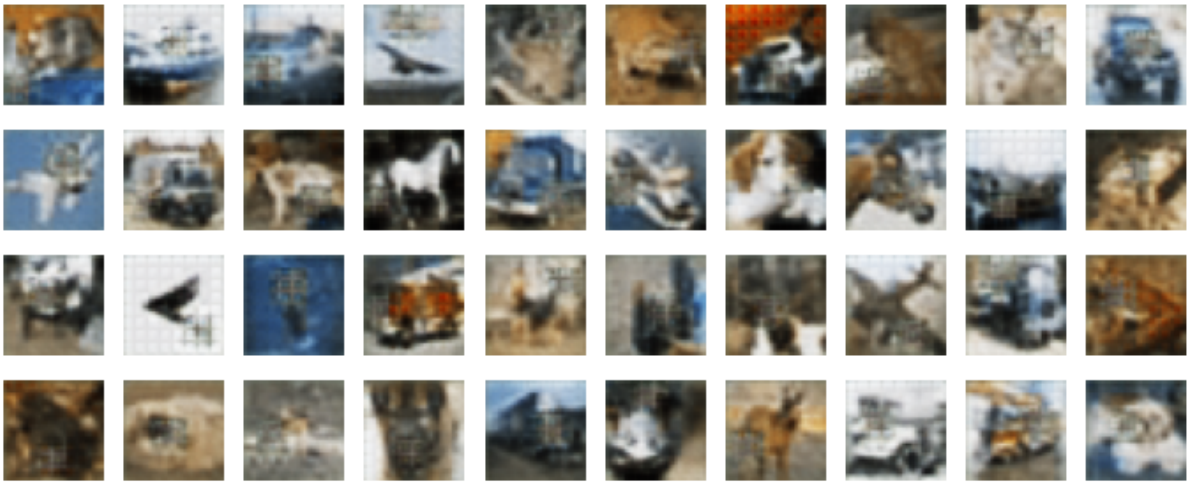
Modelo 4

```
# Define model
AE = Sequential()
AE.add(Input(shape=(32, 32, 3)))
AE.add(Conv2D(8, 3, strides=(2, 2), padding='same', activation='relu'))
AE.add(Conv2D(8, 3, strides=(2, 2), padding='same', activation='relu'))
AE.add(Conv2DTranspose(8, 3, strides=(2, 2), padding='same', activation='relu'))
AE.add(Conv2DTranspose(3, 3, strides=(2, 2), padding='same', activation='sigmoid'))

AE.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 16, 16, 8)         224
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 8, 8, 8)           584
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 16, 16, 8)         584
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 32, 32, 3)         219
=================================================================
Total params: 1,611
Trainable params: 1,611
Non-trainable params: 0
_________________________________________________________________
```

```
# Compile and train
AE.compile(optimizer='adam', loss=tf.keras.losses.MeanSquaredLogarithmicError())
h = AE.fit(x=x_train, y=y_train, batch_size=500, epochs=150, validation_split=0.2)
```



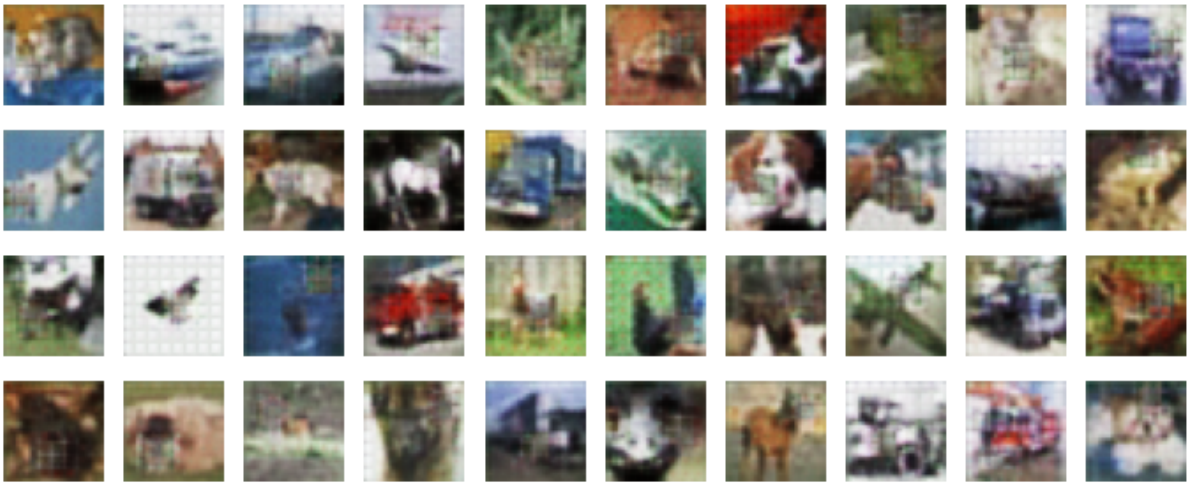
Modelo 5

```
# Define model
AE = Sequential()
AE.add(Input(shape=(32, 32, 3)))
AE.add(Conv2D(8, 3, strides=(2, 2), padding='same', activation='relu'))
AE.add(Conv2D(8, 3, strides=(2, 2), padding='same', activation='relu'))
AE.add(Conv2DTranspose(8, 3, strides=(2, 2), padding='same', activation='relu'))
AE.add(Conv2DTranspose(3, 3, strides=(2, 2), padding='same', activation='sigmoid'))

AE.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 16, 16, 8)         224
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 8, 8, 8)           584
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 16, 16, 8)         584
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 32, 32, 3)         219
=================================================================
Total params: 1,611
Trainable params: 1,611
Non-trainable params: 0
_________________________________________________________________
```

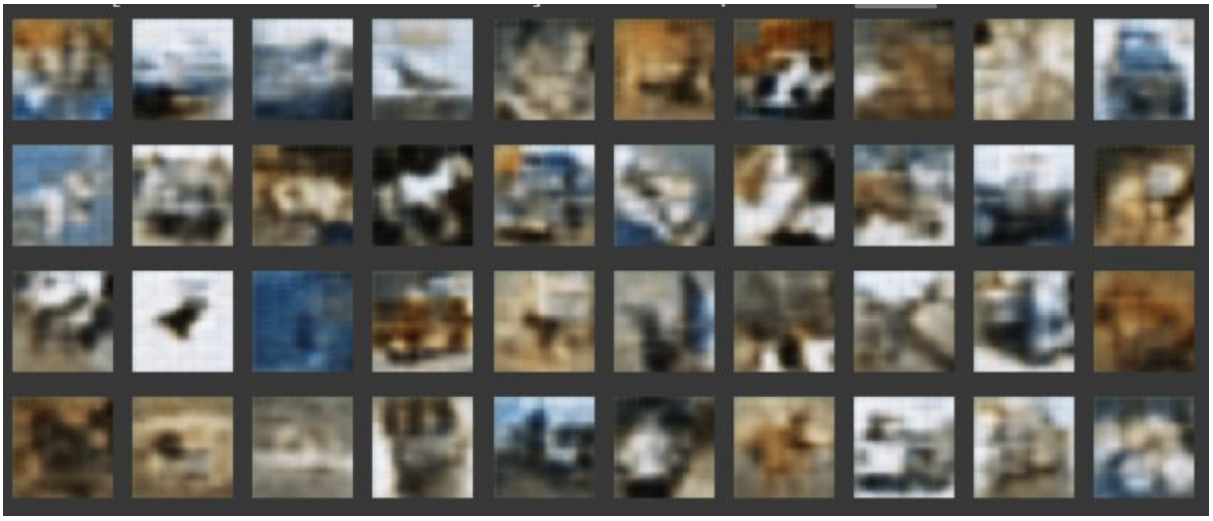
```
# Compile and train
AE.compile(optimizer='adam', loss=tf.keras.losses.MeanSquaredLogarithmicError())
h = AE.fit(x=x_train, y=y_train, batch_size=800, epochs=200, validation_split=0.2)
```



Modelo 6

```
# Vemos como lucen los mejores modelos
AE = Sequential()
AE.add(Input(shape=(32, 32, 3)))
AE.add(Conv2D(8, 3, strides=(2, 2), padding='same', activation='selu'))
AE.add(Conv2D(8, 3, strides=(2, 2), padding='same', activation='selu'))
AE.add(Conv2D(8, 3, strides=(2, 2), padding='same', activation='selu'))
AE.add(Conv2DTranspose(8, 3, strides=(2, 2), padding='same', activation='selu'))
AE.add(Conv2DTranspose(8, 3, strides=(2, 2), padding='same', activation='selu'))
AE.add(Conv2DTranspose(3, 3, strides=(2, 2), padding='same', activation='sigmoid'))

#Compile
AE.compile(optimizer='Adam', loss='mean_squared_logarithmic_error')
h = AE.fit(x=x_train, y=y_train, batch_size=500, epochs=50, validation_split=0.2,verbose = 0)
```



Modelo 7

```
AE = Sequential()
AE.add(Input(shape=(32, 32, 3)))
AE.add(Conv2D(8, 3, strides=(2, 2), padding='same', activation='selu'))
AE.add(Conv2D(8, 3, strides=(2, 2), padding='same', activation='selu'))
AE.add(Conv2D(8, 3, strides=(2, 2), padding='same', activation='selu'))
AE.add(Conv2DTranspose(8, 3, strides=(2, 2), padding='same', activation='selu'))
AE.add(Conv2DTranspose(8, 3, strides=(2, 2), padding='same', activation='selu'))
AE.add(Conv2DTranspose(3, 3, strides=(2, 2), padding='same', activation='sigmoid'))

#Compile
AE.compile(optimizer='Adam', loss='mean_squared_logarithmic_error')
h = AE.fit(x=x_train, y=y_train, batch_size=500, epochs=10, validation_split=0.2,verbose = 0)
```

Experiencias aprendidas:

1. Se necesita un balance para obtener buenos resultados. Es decir, más no significa mejor. Pues el aumentar el número de capas en la red neuronal o el incrementar la cantidad de “batches” y de épocas no se traduce en un mejor desempeño de la red neuronal para rellenar huecos.

2. Las funciones de activación “ReLu” y “Sigmoid” son en las que observamos mejor desempeño del algoritmo. La activación selu, difumina demasiado la imagen y a pesar de tener buenos resultados numéricos, al comparar con la imagen original, la salida de la red está muy borrosa.

3. Los resultados numéricos no siempre empatan con los mejores resultados visuales. Esto se debe a que aun cuando la perdida *loss* fue menor para algunos experimentos no siempre se veian tan bien los resultados. Para esto lo mejor seria escoger una métrica adecuada.

4. Tuvimos dificultades para modificar el número de neuronas por capa ya que estas están sujetas a la arquitectura del autoencoder.

5. Aunque la red neuronal sea buena identificando bordes, muchas veces no tiene un buen desempeño en preservar la tonalidad de los colores de las imágenes originales.