**EJEMPLO DE EXAMEN TEST VIDEO TEÓRICO**

1. Respecto al Variational Autoencoder, indique qué afirmación es falsa:
2. El uso de un término KL en el espacio latente ayuda a mejorar la reconstrucción de la imagen. (el KL es el error de regularización/ordenación del espacio latente, no el de reconstrucción)
3. La función de pérdidas tiene dos términos: el error de reconstrucción, y la regularización del espacio latente.
4. El embedding de la entrada está compuesto por dos vectores. (se refería a los vectores mu y std, que están a la salida del encoder, la pregunta no está bien formulada)
5. En cada iteración se debe realizar una reparametrización de la compresión realizada por el encoder, que ayuda a mejorar la consistencia de dicho espacio.

# En un data generator hecho manualmente para cargar un dataset de imágenes, el método \_getitem\_() devuelve:

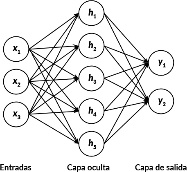
# Una imagen y su etiqueta asociada.

# Un conjunto de imágenes con sus respectivas etiquetas.

# Una instancia del generator.

# Una imagen tras aplicarle data augmentation.

**EJEMPLO DE EXAMEN FINAL - TIPO TEST**

1. ¿Cuántos parámetros entrenables tiene la siguiente arquitectura de red neuronal artificial?
2. 36
3. 32
4. 25
5. 16

# Se quiere desarrollar una red neuronal para clasificar una población en grupos de edades, a partir de unos datos fisiológicos. ¿Qué combinación de activación de salida y función de pérdidas es la más adecuada de las siguientes?

1. Lineal + MSE
2. Tanh + Categorical Cross Entropy
3. Sigmoide + MSE
4. Softmax + Categorical Cross Entropy

# En una red neuronal convolucional, ¿cuáles serán las dimensiones del mapa de activación a la salida de una capa Conv2D compuesta por 32 filtros con *kernel*3 × 3 si se emplea un valor de *stride*= 2 y el parámetro *zero padding* se establece a *same*? Tenga en cuenta que la imagen de entrada es de 16 × 16 pixeles.

1. 16 x 16 x 32
2. 16 x 16 x 16
3. 32 x 32 x 16
4. 8 x 8 x 32

# ¿Cuáles son las dimensiones del tensor de entrada a una red neuronal convolucional para albergar 32 videos de 18 *frames*en color de dimensiones espaciales 256 x 128?

# Es necesario un tensor 4D de dimensiones (256, 128, 18, 32)

# Es necesario un tensor 4D de dimensiones (256, 128, 18)

# Es necesario un tensor 4D de dimensiones (32, 256, 128, 18)

# Es necesario un tensor 5D -> en color (32,256,128,3,18)

# Acerca de las arquitecturas de red vistas para segmentación y su entrenamiento, indique qué afirmación es falsa.

# Normalmente se emplea como función de pérdidas los índices de Jaccard o Dice. (normalmente se usa IoU y otro tipo de función de pérdidas como las que se mencionan en la pregunta)

# Se pueden emplear tanto funciones de pérdidas de clasificación, como la entropía cruzada, como otras especializadas en segmentación

# El uso de skip-connections permite mantener nivel de detalle entre la codificación y decodificación de las características espaciales.

# Las arquitecturas de segmentación presentan un decoder con bloques convolucionales. (no todas tienen por qué tener decoder, se podría usar un clasificador normal de top model y en el base model una clasificación a nivel de pixel [Flatten -> Resize(shape de la imagen) -> Clasificación a nivel de pixel], ej. Fully Connected Network -> diapositivas)

**EJEMPLO DE EXAMEN FINAL – PREGUNTA DESARROLLO**

1. Describa el objetivo del método Word2Vec, las etapas de entrenamiento, y la diferencia en sus variantes.

El objetivo del método Word2Vec es obtener un embedding manejable por una red neuronal a partir de un texto dado.

La diferencia en sus variantes skipgram y continue bag of words según la aplicación.

1. Describa el objetivo, arquitectura y optimización del algoritmo R-CNN. ¿Qué principal novedad introdujo posteriormente Fast R-CNN?

El objetivo de R-CNN es, además de una mera clasificación hacer una localización de los objetos pertenecientes a las clases con las que se le ha entrenado. Esto lo hace mediante un Bounding Box.

**EJEMPLO DE EXAMEN FINAL – PREGUNTA PSEUDOCÓDIGO**

1. Implemente una arquitectura de red neuronal para llevar a cabo un problema de regresión a partir de imágenes. En concreto, se quiere realizar una predicción del precio de un coche a partir de una colección de imágenes X\_train, y un vector Y que contiene los precios de dicho dataset. Se pide hacer un fine-tuning de una red VGG19, utilizando como función de pérdidas la binary cross entropy. Escriba el pseudocódigo que refleje el procesamiento de los datos, carga del modelo, modificación del mismo, y entrenamiento, haciendo uso del optimizador que más oportuno considere, así como otros hiperparámetros.

# Imports

from datasets.coches import load\_data

from models import Model, Sequential, VGG19

from layers import Input, Flatten, Dense

from utilities import DataGen

from optimizers import Adam

from preprocessing import Normalizer

# Carga del dataset

X\_train, y\_train, X\_test, y\_test = load\_data()

X\_train = X\_train / 255 # normalizamos imagenes

X\_test = X\_test / 255 # normalizamos imágenes

# para poder usar el binary cross entropy hay que normalizar los valores

# de salida de la red entre 0 y 1

normalizer = Normalizer ()

y\_train\_std = normalizer.fit\_transform(y\_train)

# Exploración del dataset y data cleaning

# Definimos generadores con data augmentation para evitar el overfitting

train\_gen = DataGen(X\_train, y\_train\_std, val\_split=0.2, parámetros\_de\_data\_augmentation)

test\_gen = DataGen(X\_test)

# Hiperparámetros

input\_shape = X\_train[0].shape

epochs = 10

learning\_rate = 1e-4

batch\_size = 128

steps\_per\_epoch = X\_train.shape[0] // batch\_size

# Base Model

base\_model = VGG(input\_shape)

# congelamos todo menos el último bloque convolucional

for layer in base\_model:

if layer.name == ‘conv1\_last\_block’:

break

else:

layer.trainable = False

# Top Model

top\_model = Sequential()

top\_model.add(Flatten())

top\_model.add(Dense(512, activation=’relu’))

top\_model.add(Dense(128, activation=’relu’))

top\_model.add(Dense(32, activation=’relu’))

top\_model.add(Dense(1, activation=’sigmoid’))

# Modelo completo

model = Sequential()

model.add(base\_model)

model.add(top\_model)

# Compilamos el modelo

model.compile(optimizar=Adam(learning\_rate), loss=”binary\_crossentropy” metrics=[‘accuracy’])

# Entrenamos el modelo

model.fit(train\_gen, epochs, batch\_size)

# Evaluamos el modelo

y\_test\_std = normalizer.transform(y\_test)

model.evaluate(test\_gen, y\_test\_std)

# evaluar resultados

# Predicción

…