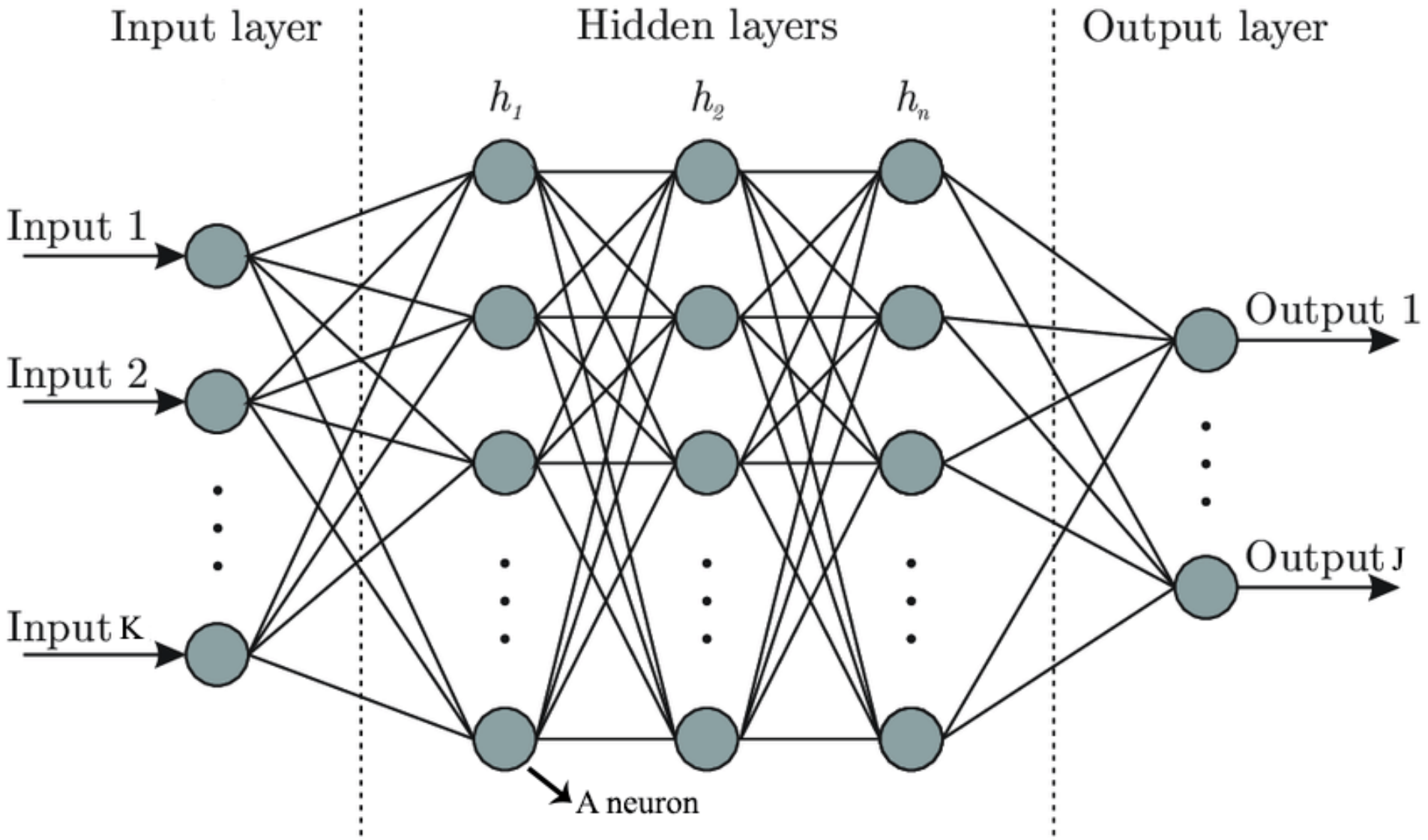
CHAPTER 3
Development of DNN Models with KERAS

What is KERAS?

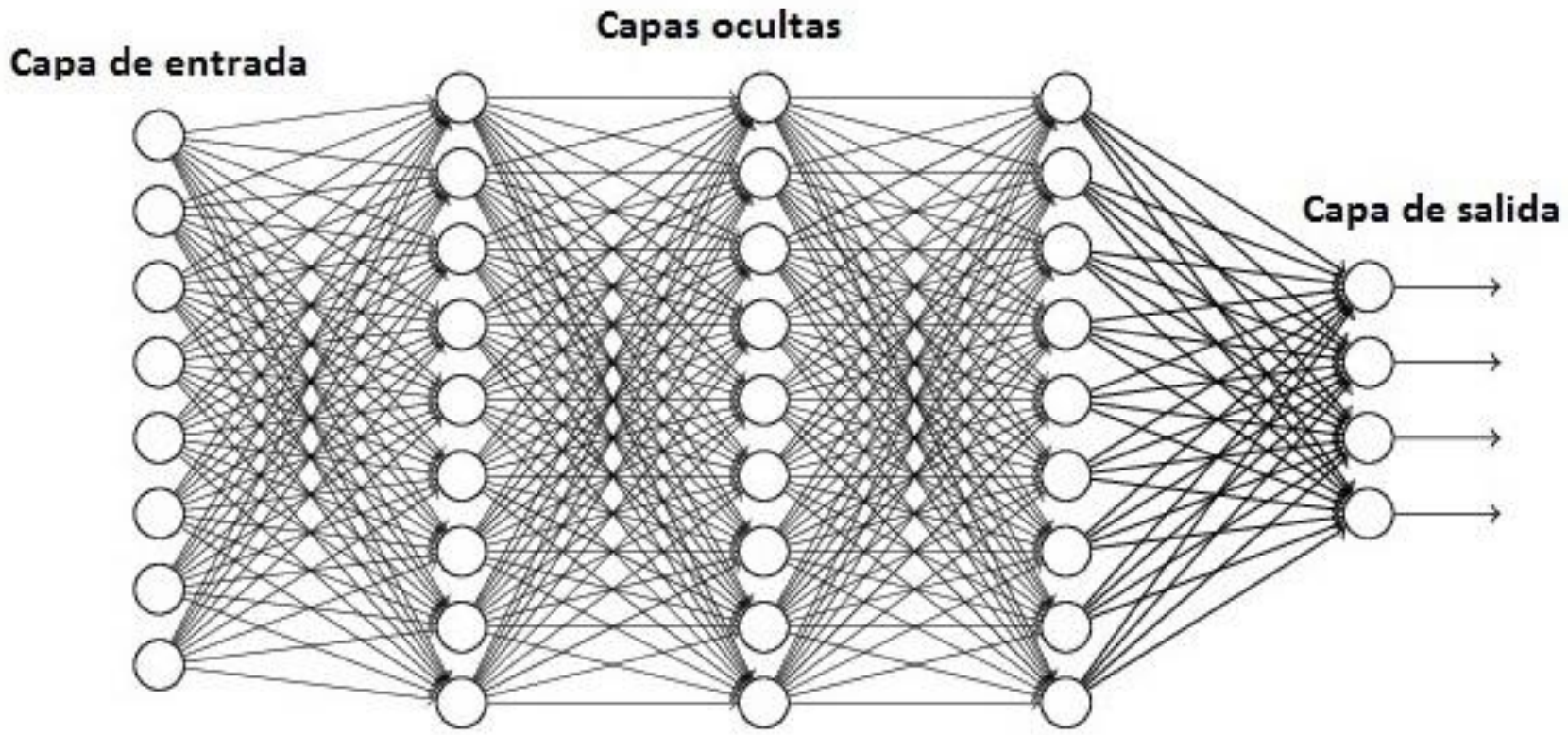
- Keras is an open source neural network library written in high-level Python.
- It is capable of running on top of TensorFlow (and other libraries)
- It is specifically designed to enable experimentation in a relatively short time with Deep Learning networks.
- Its strengths are focused on being user-friendly, modular and extensible.
- Keras contains several implementations of the building blocks of neural networks such as:
  - Layers
  - Activation functions
  - objective functions
  - mathematical optimizers

MultiLayer Perceptrons (MLP)

Structure


Input layer
Hidden layers
Output layer
h_1
h_2
h_n
Input 1
Input 2
Input K
Output 1
Output J
A neuron

Structure



Use (i)

- Import in Python (example)

- `import tensorflow.keras as kr`

- From here all the functions have this prefix:

  `kr.function`

- Instead of

  `tensorflow.keras.function`

Use (ii)

- Start a model (layer structure)
- `model = tf.models.Sequential()`
- “model” is the object where the layers will fit

Use (iii)

- Adding a layer of perceptrons (densely connected)
  - `model.add(kr.layers.Dense(n_neurons))`

- Notes:
  - The number of inputs to each neuron will be the number of outputs of the previous layer.
  - If it is the first layer, you must tell it how many inputs it has (parameters).
  - You can doit in two ways (explicit or implicit)

    explicit

    - `model.add(InputLayer(input_shape=(n,))`
    - `model.add(kr.layers.Dense(n_neurons)`

    implicit

    - `model.add(kr.layers.Dense(n_neuronas, input_shape=(n,))`
  - If it is the last layer, n_neurons must match the number of classes.
    - `model.add(kr.layers.Dense(n_classes)`

Use (iv)

- After addding the layer the activation function must be added

- `model.add(kr.layers.Activation('softmax'))`

- `softmax`
- `softplus`
- `relu`
- `selu`
- `sigmoid`
- `...`
- `CUSTOM`

Use (v)

- Compact way of adding the layer and the activation function at the same time

- `model.add(kr.layers.Dense(n_neuronas)),activation='softmax')`

Use (vi)

- Example for a network of only one layer (plus the input)
  - Number of parameters = 50
  - Number of classes = 10

- `model.add(kr.layers.Dense(10,input_shape=(50,),activation='softmax')`

Use (vii)

- Example (3 layer + input)

```
model.add(kr.layers.Dense(30,input_shape=(20,), activation='relu')
model.add(kr.layers.Dense(50, activation='relu’)
model.add(kr.layers.Dense(10, activation=‘softmax’)
```

- Notes:
  - First layer should inform about the number of inputs
  - Last layer neuron number = number of clases
  - Next layer should be softmax activation
  - In the other layers activation function has more liberty

Use (viii)

- An optimizator shoul be defined in every network
- `OPTIMIZER=kr.optimizers.Adam(learning_rate=0.0001)`

- Compilation of the network
- `model.compile(loss='categorical_crossentropy', optimizer=OPTIMIZER, metrics=['accuracy'])`

- Training start
- `history = model.fit(X_train, Y_train, batch_size=BATCH_SIZE, epochs=NB_EPOCH, verbose=VERBOSE, validation_data=(X_test, Y_test))`

Network evaluation

- When the network is put into production, the output of an element must be predicted. This is called inference.
- The WAV file to be predicted is loaded
- Its parameters are calculated (for example the MEL spectrogram) and stored in the variable X_test

- The function to predict an element is called
- `Y_pred = model.predict(X_test)    #inference, predices output`

Confusion Matrix

```
import seaborn as sns

from sklearn.metrics import confusion_matrix

Y_pred = model.predict(X_test)                #predecir (poner la red en producción)
Y_pred_classes = np.argmax(Y_pred,axis = 1)   # pasa de categorical a one-hot los predichos
Y_true = np.argmax(Y_test,axis = 1)           # pasa de categorical a one-hot la verdad de base

confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)

f,ax = plt.subplots(figsize=(10, 10))

sns.heatmap(confusion_mtx, annot=True, linewidths=0.01,cmap="Greens",linecolor="gray", fmt= '.1f',ax=ax)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

Hyperparameters

Hiperparámetros (definición)

- They are parameters external to the model (different from the weights wi)
- They are not learned directly from the dataset during the training process.
- Instead, they must be set before starting training and are used to control the global configuration of the model.
- Hyperparameters are essential for the optimization and fine-tuning of the neural network model.

Hyperparameters (some)

1. Epochs
2. Batch size
3. Lost Function
4. Optimizer
5. Learning rate
6. Regularization
7. Network Architecture
8. Features (quantity and type)

1. Epochs

- An epoch is one complete pass through the training dataset during training.
- The number of epochs is the number of times the model will see the entire dataset during training.
- Too many epochs can lead to overfitting, while too few can result in an undertrained model.
- It is important to experiment with different EPOCHS values during model development to find the optimal number that balances the model's ability to learn and generalize without overfitting.
- This can be monitored by comparing learning performance using the ACCURACY of the training set and the validation set.

2. Batch size

- Indica el número de ejemplos de entrenamiento utilizados en una iteración antes de actualizar los pesos del modelo.
- Los modelos se entrenan de manera más eficiente en lotes en lugar de elementos individuales o en lugar de todo el conjunto de test.

$$1 < \textit{Tamaño del lote} < \textit{Tamaño del conjunto de datos}$$

- El tamaño del lote afecta a la velocidad y la estabilidad del entrenamiento.
  - Lote más grande: +rápido entrenam, +estable, +memoria necesaria
  - Lote más pequeño: +generalización (-overfitting)

3. Lost function

- Es la función que el modelo está tratando de minimizar durante el entrenamiento.
- También conocida como función objetivo o función de costo
- La elección de la función de pérdida depende del tipo de problema, como la clasificación o la regresión
- El objetivo del entrenamiento de una red neuronal es minimizar esta función de pérdida para que el modelo haga predicciones más precisas.
- Ejemplos:
  - loss = MeanSquaredError() #la más sencilla y conocida utilizada en procesado de señal
  - loss = BinaryCrossentropy() #utilizada en problemas de clasificación binaria (2 clases)
  - loss = CategoricalCrossentropy() #utilizada en problemas de clasificación multiclase
  - loss = SparseCategoricalCrossentropy() #clasificación por enteros (no se ha hecho categorical)

4. Optimizer

- Es el algoritmo utilizado para ajustar los pesos del modelo durante el entrenamiento para minimizar la función de pérdida.
- La elección del optimizador es crucial porque afecta cómo se actualizan los pesos del modelo en cada iteración del proceso de entrenamiento.
- Descenso de Gradiente Estocástico (SDG)
  - Uno de los optimizadores más simples. Actualiza los pesos en la dirección opuesta al gradiente de la función de pérdida con respecto a los pesos
- Adam (Adaptive Moment Estimation)
  - Combina el concepto de momento y la adaptación de la tasa de aprendizaje. Es conocido por su eficiencia y es uno de los más utilizados en la práctica
- RMSprop (Root Mean Square Propagation)
  - Ajusta la tasa de aprendizaje para cada parámetro individualmente basándose en la magnitud de los gradientes

5. Learning rate

- Es un factor que multiplica la magnitud de las actualizaciones de los pesos durante el entrenamiento en los optimizadores.
- Una tasa de aprendizaje demasiado baja puede hacer que el modelo converja lentamente o se quede atascado, mientras que una tasa de aprendizaje demasiado alta puede hacer que el modelo oscile o no converja en absoluto.
- La selección y ajuste adecuado de este hiperparámetro es crucial para lograr un rendimiento óptimo del modelo en una tarea específica.
- Ejemplo de uso
  - optimizer = SDG(learning_rate=0.001)
  - optimizer = Adam(learning_rate=0.005)
  - optimizer = RMSprop (learning_rate=0.002)

6. Regularization

- Es un conjunto de técnicas diseñadas para prevenir el sobreajuste (overfitting) y mejorar la generalización del modelo
- Introduce penalizaciones en la complejidad del modelo para controlar la magnitud de los pesos.

1. Regularización L1 y L2
   - son términos de penalización en la función de pérdida basados en la magnitud de los pesos. L1 penaliza la suma de los valores absolutos de los pesos, mientras que L2 penaliza la suma de los cuadrados de los pesos.
   - `model.add(Dense(64, input_dim=100, activation='relu', kernel_regularizer=l1(0.01)))`
2. Dropout:
   - Durante el entrenamiento, aleatoriamente se "apaga" (pone a cero) una fracción (%) de las neuronas en una capa para evitar que dependan demasiado de ciertas características
   - `model.add(Dense(64, input_dim=100, activation='relu'))`
   - `model.add(Dropout(0.5))`

7. Network Architecture

- La elección de la arquitectura de la red puede tener un impacto significativo en el rendimiento del modelo.
- Se pueden considerar como hiperparámetros de una red:

1. Número de Capas
2. Número de Neuronas en Cada Capa
3. Funciones de Activación
4. Conexiones entre Neuronas (densamente conectadas o conexiones más específicas)

8. Features

- De las señales que queremos clasificar extraemos una serie de características (especialmente en sonido).
- En el caso de espectrogramas, podemos consideran como hiperparámetros.
  - El tamaño de ventana temporal
  - El tamaño de la FFT en espectrograma
  - El número de bandas MEL en espectrogramas MEL
  - El número de coeficientes Mel-Cepstrum MFCC

Convolutional Networks for Audio

Intro

- They are used to recognize images
- Why use them for sound?
  - Because sound is analyzed both in time and in 2D frequency
  - And the spectrogram can be seen as an image (also Mel / MFCC)
  - There are sounds that extend on the time axis
  - There are sounds with a large bandwidth that extend in frequency or leave a harmonic pattern in frequency

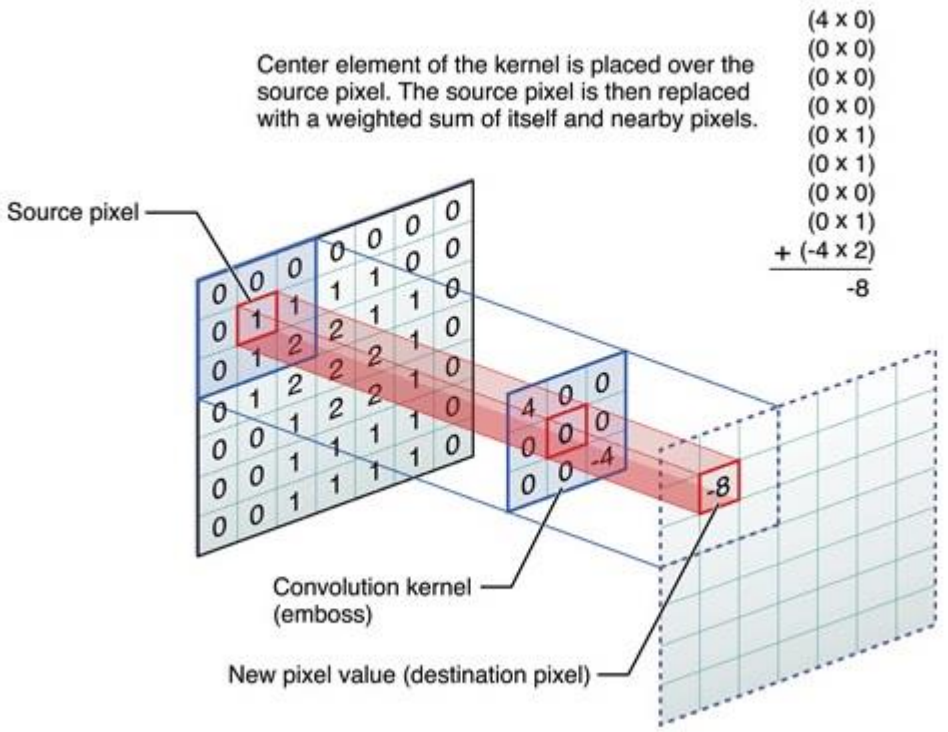
2D Filter

Image pixels

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

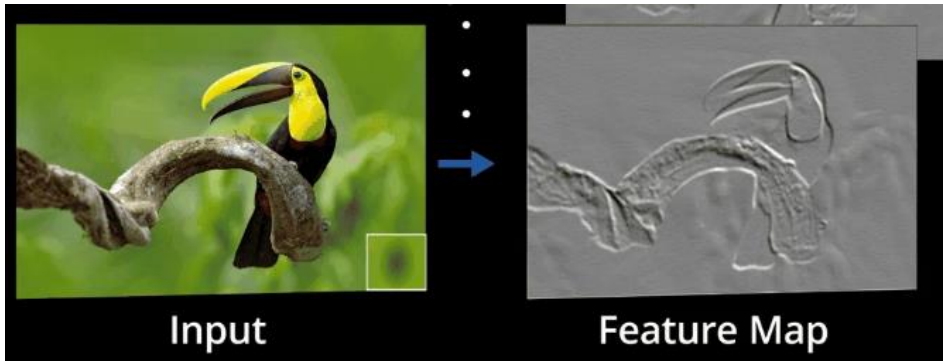
Filter

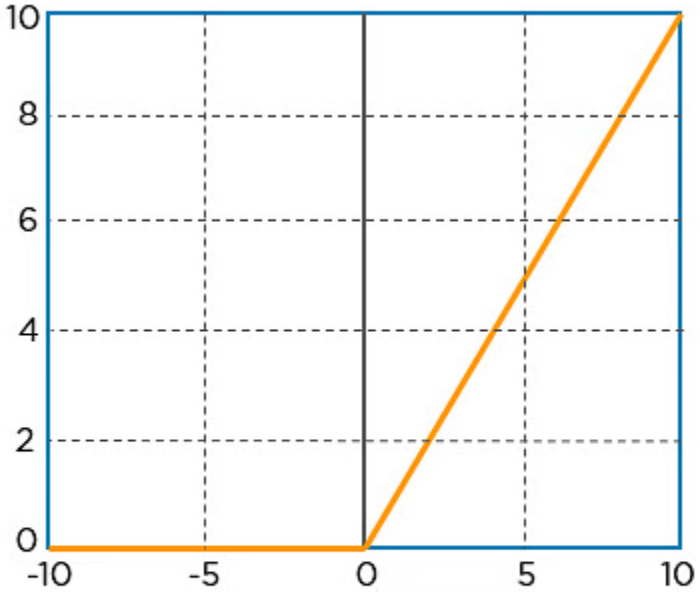
1	0	1
0	1	0
1	0	1

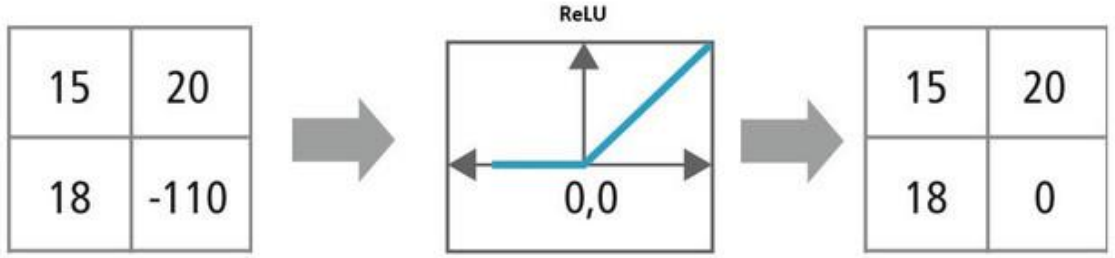


RELU Activation Function for Edge detection

- If the filter is gradient type, negative numbers are removed from the result.
- Helps to identify the edges of objects

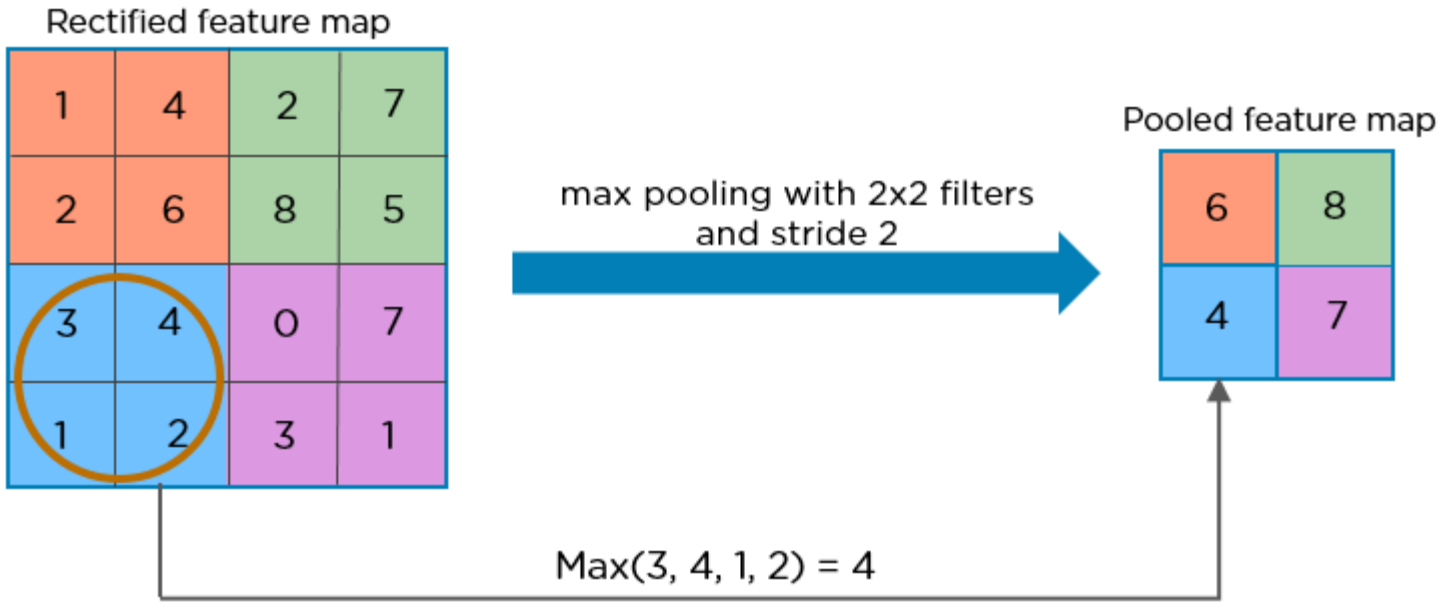






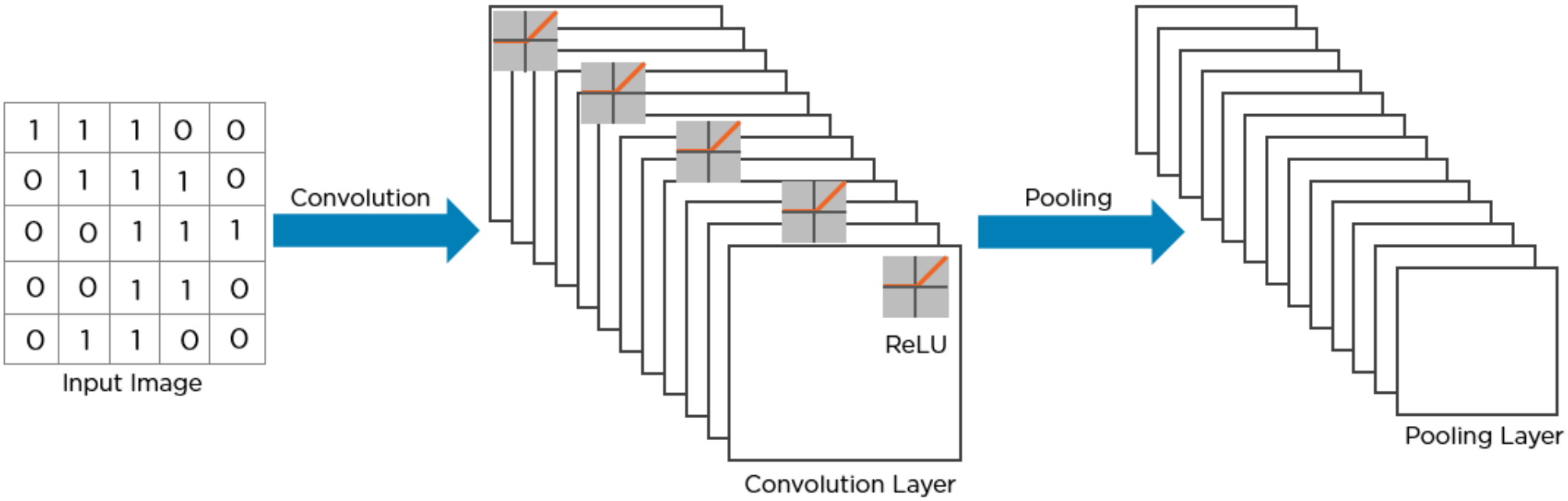
Pooling Layer

- Reduce the size of the images (therefore the number of parameters/weights to train)



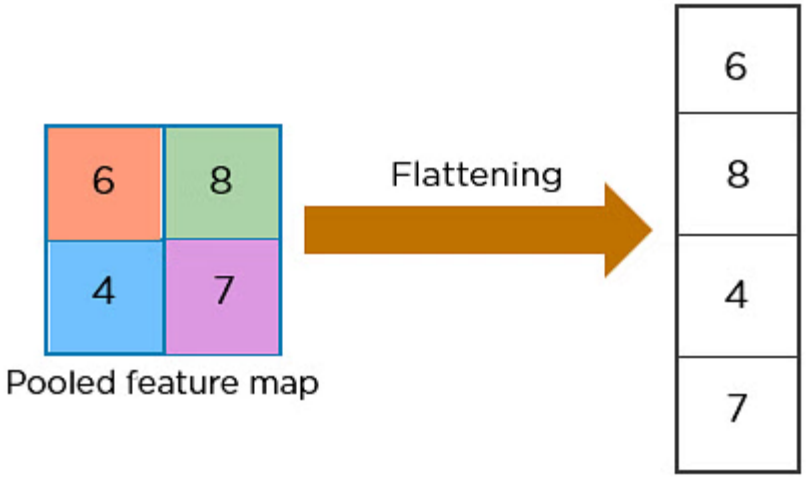
CNN Structure

- Each filter applied to the image produces a convolution layer.
- The weights of each filter are the parameters to be optimized.



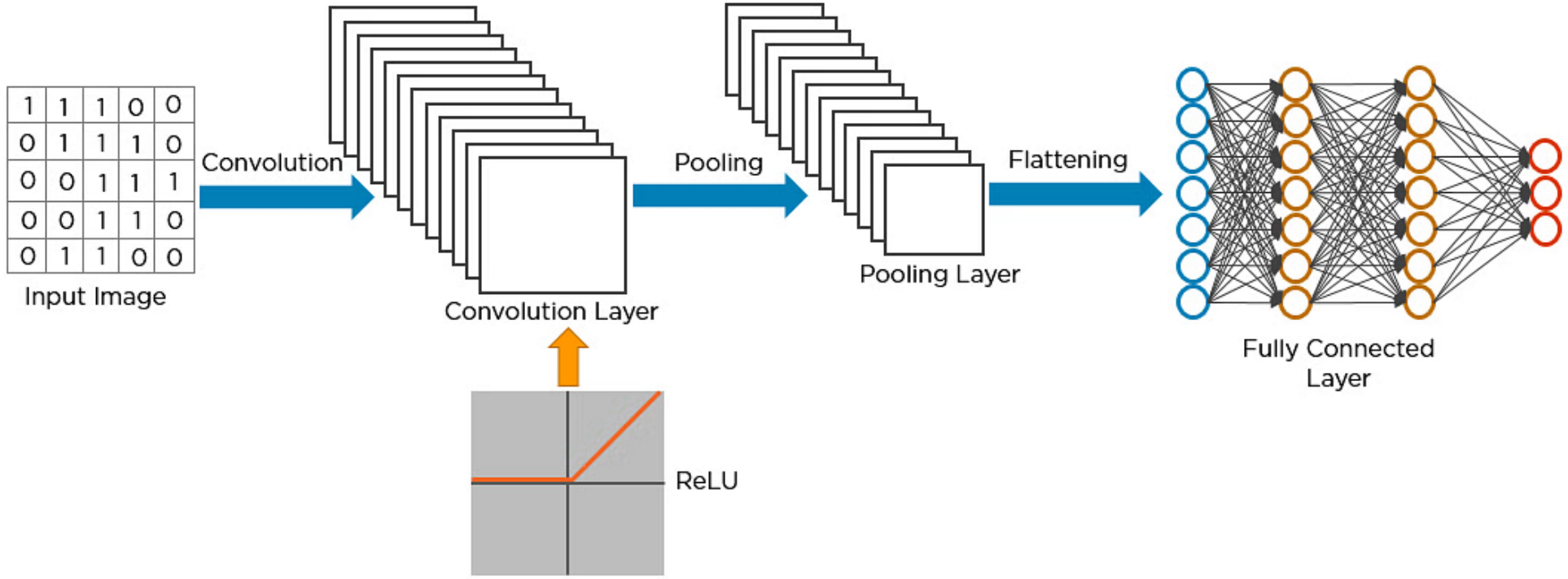
Flattening

- It consists of converting 2D data into 1D data to be processed by a network of 1D perceptrons.

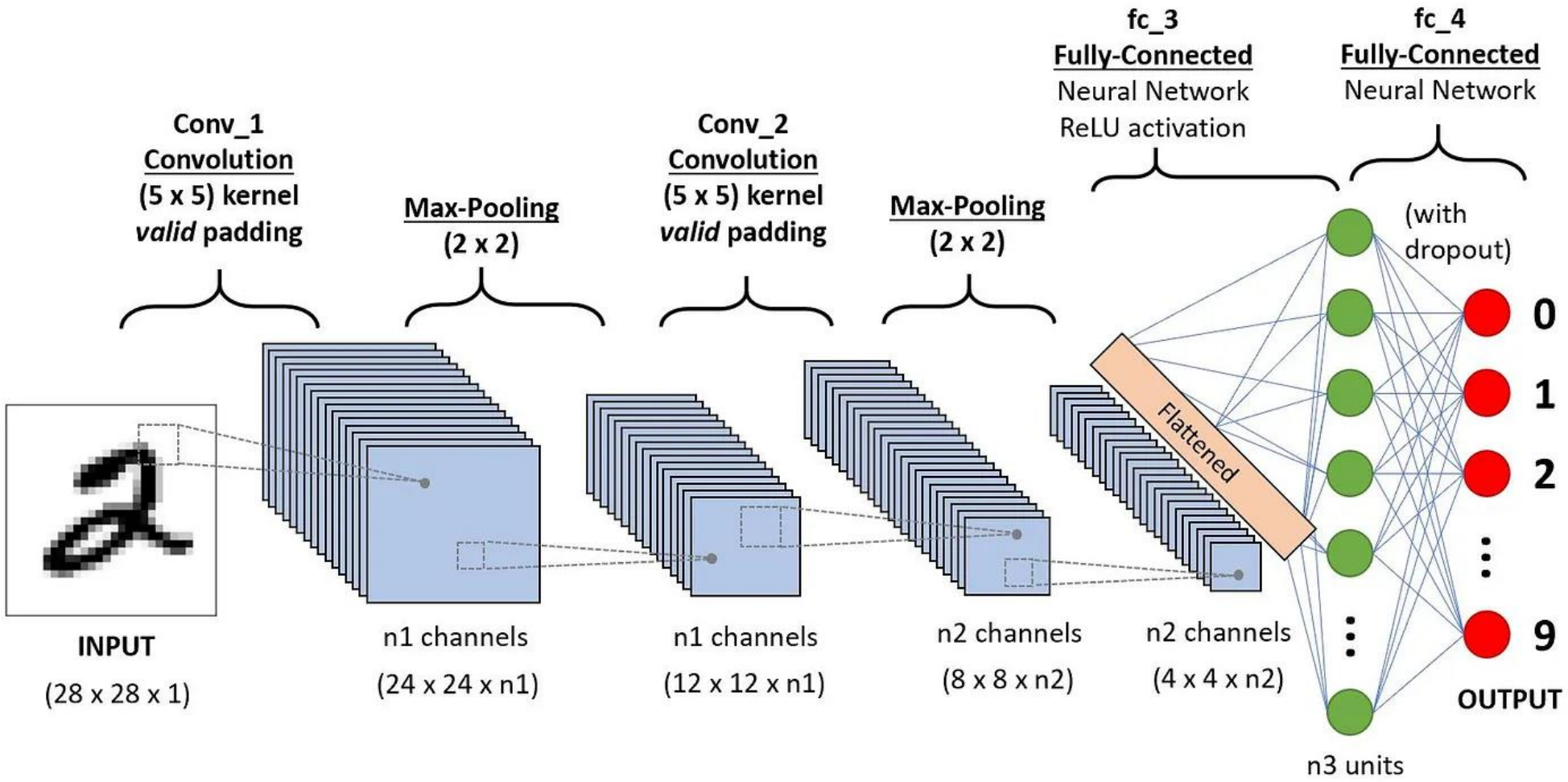


Final CNN Structure

- After flattening we put a densely connected network
- Finally the last layer is the number of classes



Structure with 2 convolutional layer



Convolution layer in Keras

- The number of images stacked in each layer is called channels.
- The number of output channels for each Conv2D layer is controlled by the first argument
- The next argument is the filter size
- `model.add(kr.layers.Conv2D(64,(3,3),activation='relu'))`

- In this layer therefore we will have 64x3x3 weights to optimize

Example

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

- Typically, as the width and height are reduced (by pooling), it can be computationally allowed to add more output channels in each Conv2D layer.
- The input size includes a third dimension for RGB

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36928

Total params: 56,320

Flattening

- Before moving from convolutional to dense, perform flattening
- model.add(tf.layers.Flatten())

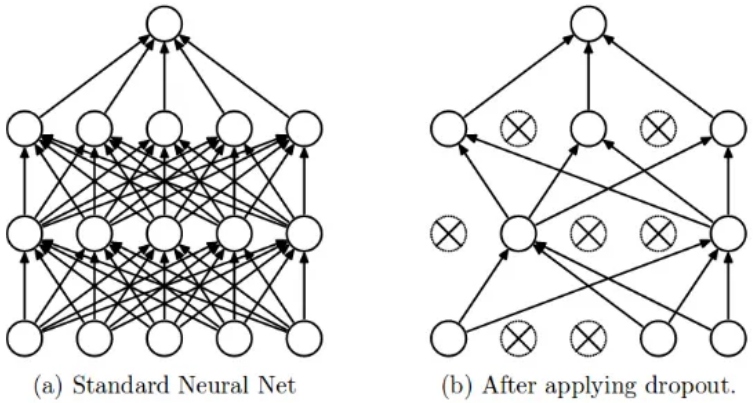
Improving Generalization and Accuracy

Definitions and solutions

- Overfitting:
  - In machine learning, overfitting occurs when an algorithm fits too closely or even exactly to its training data, resulting in a model that cannot make accurate predictions or conclusions from any data other than the training data.
- Generalization
  - The term generalization refers to the model's capability to adapt and react properly to previously unseen, new data, which has been drawn from the same distribution as the one used to build the model.
- Techniques to improve Generalization on the training process:
  - Dropout
  - Regularization
- Data augmentation
  - Data augmentation is the process of artificially generating new data from existing data, primarily to train new machine learning (ML) models.

Dropout

- It is often used to prevent overtraining and generalize the network.
- It consists of randomly disconnecting a % of neurons in each training pass (EPOCH)
- KERAS
- `model.add(tf.layers.Dropout(0.3))`



(a) Standard Neural Net

(b) After applying dropout.

Regularization

- Regularization is a technique used to reduce errors by fitting the function appropriately on the given training set and avoiding overfitting.
- The commonly used regularization techniques are :
  - Lasso Regularization – L1 Regularization
  - Ridge Regularization – L2 Regularization
  - Elastic Net Regularization – L1 and L2 Regularization
- L1 Regularization technique is called LASSO(Least Absolute Shrinkage and Selection Operator) regression.
  - Lasso Regression adds the absolute value of magnitude of the coefficient as a penalty term to the loss function(L).
  - Lasso regression also helps us achieve feature selection by penalizing the weights to approximately equal to zero if that feature does not serve any purpose in the model.
- L2 regularization technique is called Ridge regression.
  - Ridge regression adds the squared magnitude of the coefficient as a penalty term to the loss function(L).

Regularization in KERAS

- 3 keyword arguments:
  - kernel_regularizer: Regularizer to apply a penalty on the layer's kernel
  - bias_regularizer: Regularizer to apply a penalty on the layer's bias
  - activity_regularizer: Regularizer to apply a penalty on the layer's output
- Example

```
layer = layers.Dense(
    units=64,
    kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-4),
    bias_regularizer=regularizers.l2(1e-4),
    activity_regularizer=regularizers.l2(1e-5)
)
```

Data Augmentation with Librosa

- https://www.kaggle.com/code/huseinzol05/sound-augmentation-librosa
- Change pitch and speed
- Change pitch only
- Change speed only
- Change volume / dynamic range
- Add noise
- Apply Filters
- Shift silent

Recursive Networks in Audio
RNN

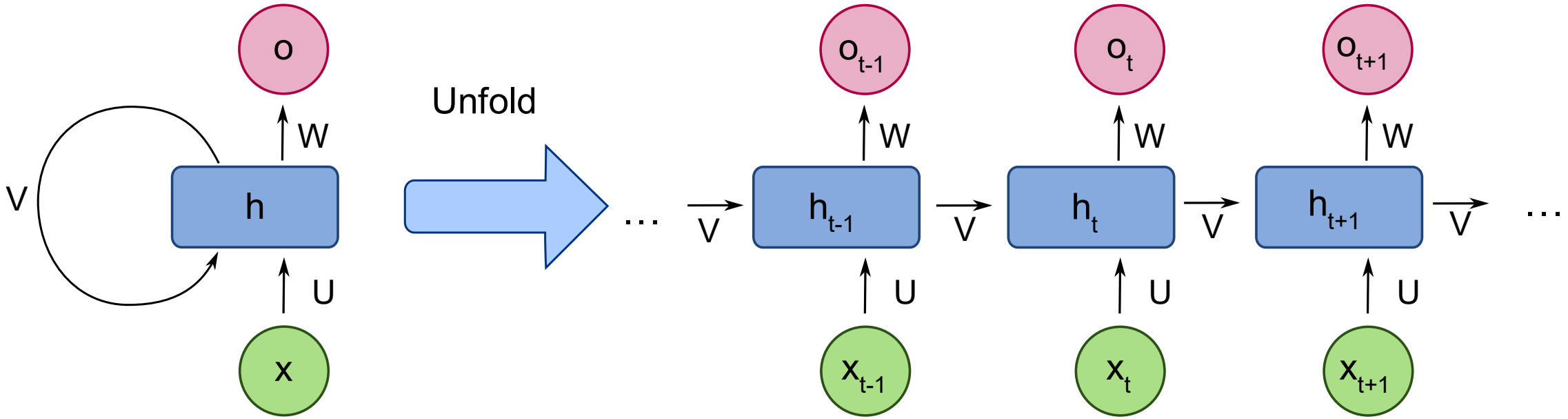
Problem in sound

- When defining a traditional DNN, the number of inputs is fixed in advance.
- Time series such as sound do not have a fixed duration.
- It is difficult to work with sound in these cases.
- In addition, there are cases where processing the entire sound fragment at once would be very expensive *(e.g. Style Recognition, 30 s)*
- A network that can be fed data little by little and that has memory would be interesting.

RNN Definition

- It is a type of network architecture that implements some kind of memory, and therefore, has a sense of time.
- It is achieved by implementing a type of neurons that receive an input from the output of a hidden layer of the previous temporal state that is reinjected.

RNN recursiveness



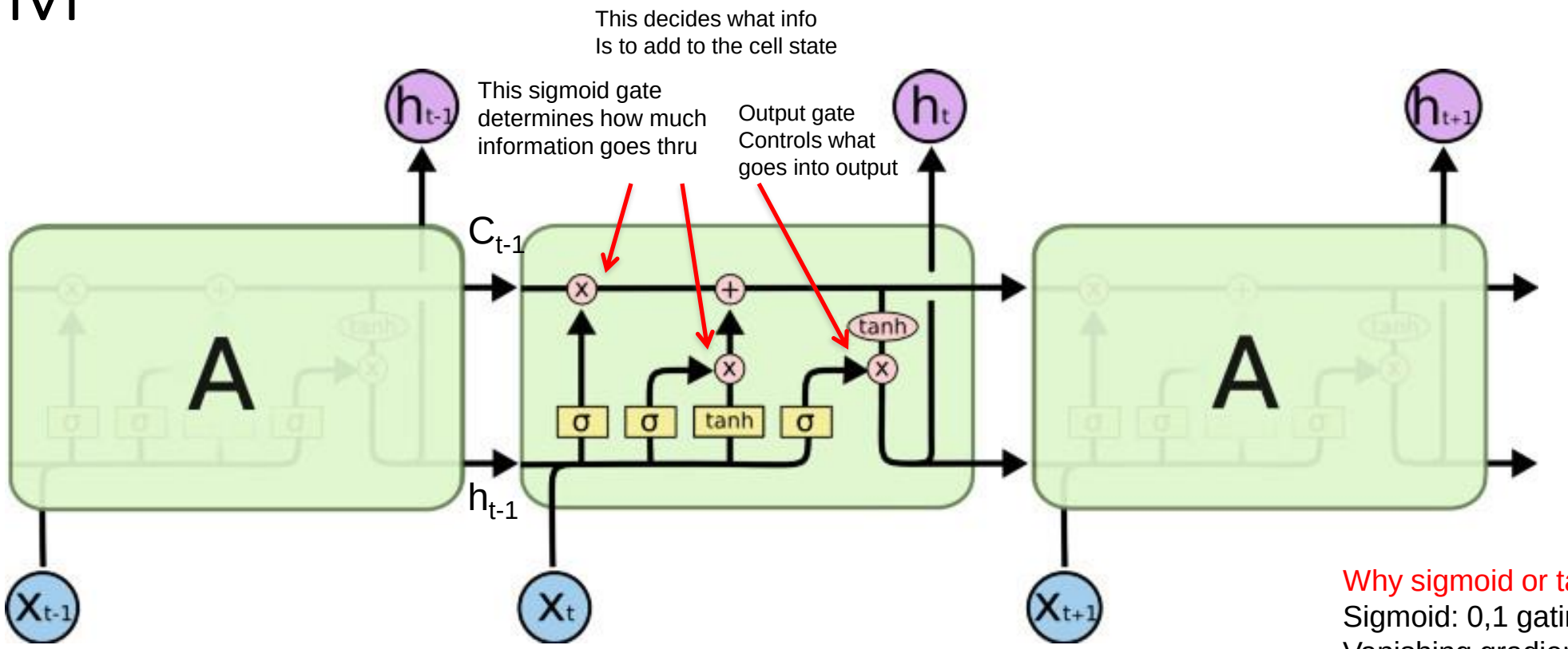
Problems with simple RNN

- Feedback causes the data from N past time interactions to lose importance.
- The network gradually loses memory.
- It is only valid for short time iterations.
- “Gradient vanishing” problem.

LSTM networks (Long-Short Term Memory)

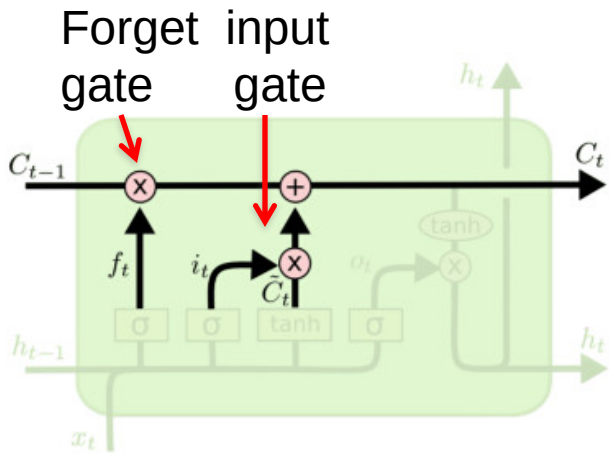
- They have a retention and forgetting mechanism.
- They can preserve past information very well.
- They can forget past information instantly if more relevant information comes in.
- They have a longer long-term memory than simple RNNs.

LSTM

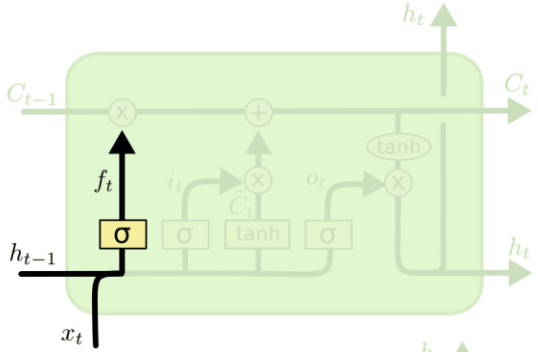


Why sigmoid or tanh:
Sigmoid: 0,1 gating as switch.
Vanishing gradient problem in LSTM is handled already.
ReLU replaces tanh ok?

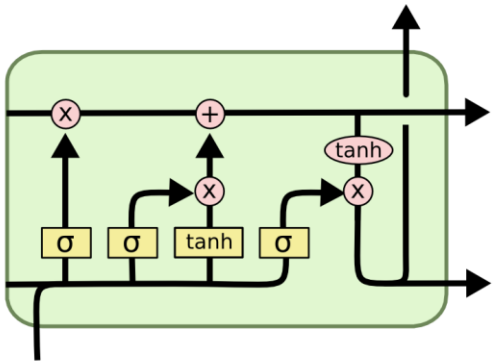
The core idea is this cell state C_t, it is changed slowly, with only minor linear interactions. It is very easy for information to flow along it unchanged.

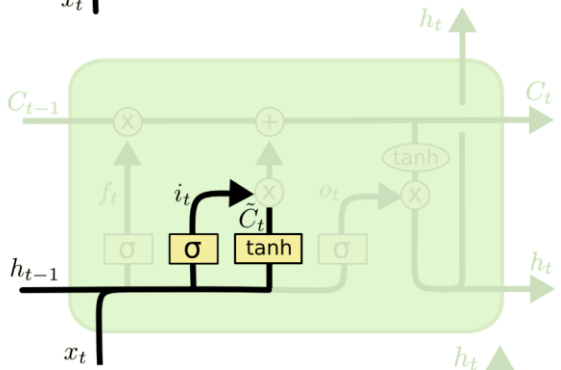


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



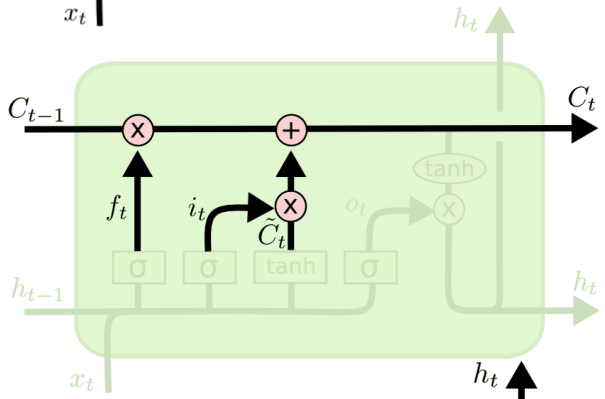
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$





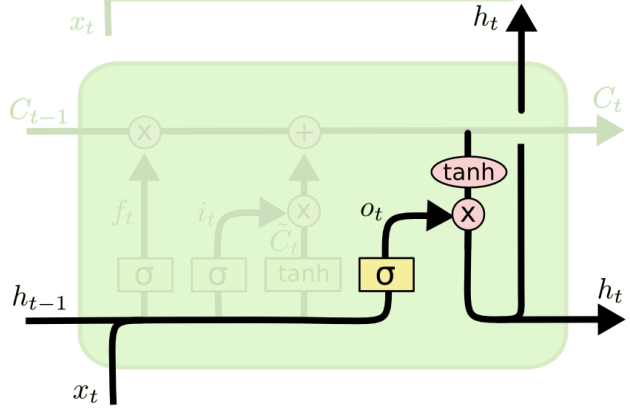
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

i_t decides what component is to be updated.
C'_t provides change contents



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

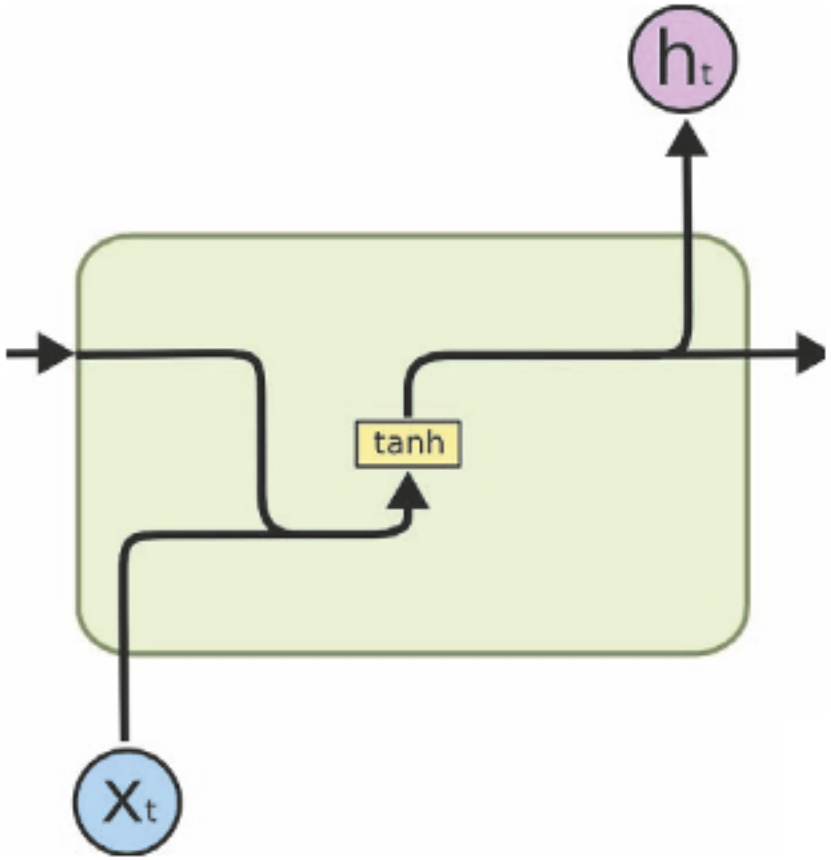
Updating the cell state



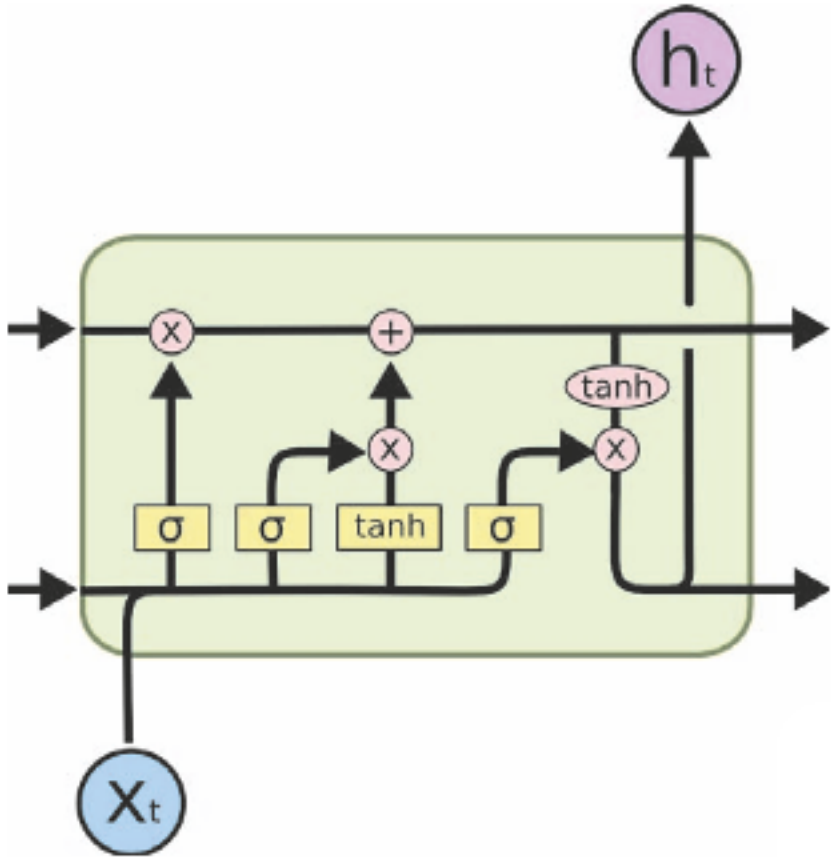
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Decide what part of the cell state to output

RNN vs LSTM



(a) RNN



(b) LSTM

How to use in KERAS

- To introduce the fist layer LSTM

- `model.add(kr.layers.LSTM(64, input_shape=(timesteps, features)))`

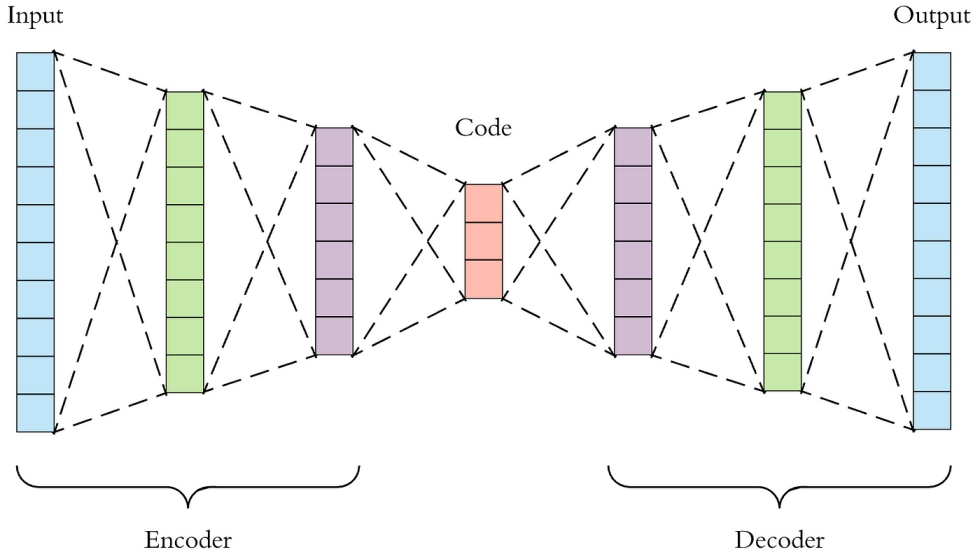
- The input parameters are two-dimensional, with the first coordinate being considered time (to apply recursion).

Encoder-Decoder Architectures

Encoder-Decoder (background)

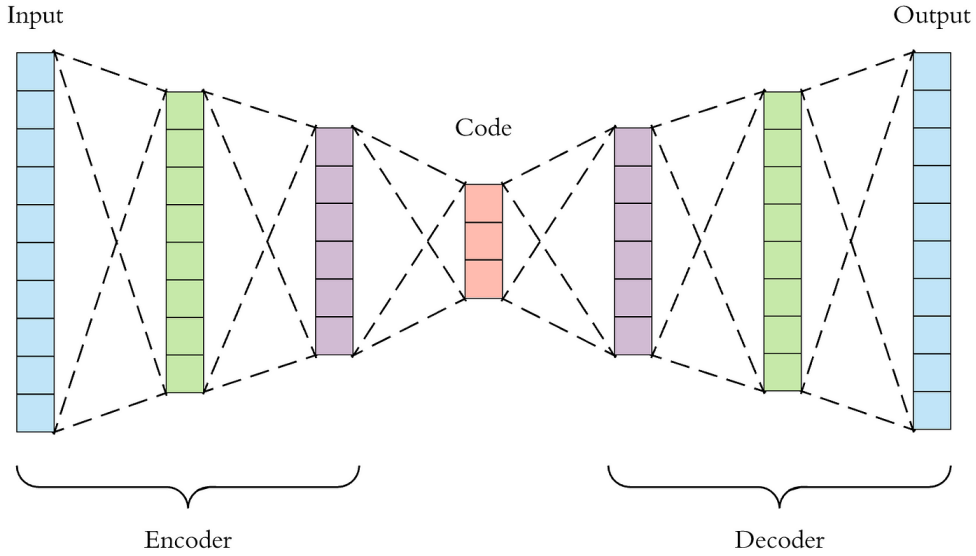
- They are DNN architectures that are widely used in sequence-to-sequence problems.
- For example, in: machine translation, text generation, text summarization, and also in audio.
- Encoder-Decoder networks are especially effective for tasks where the input and output are of variable length.
- They handle input and output sequences flexibly, capturing the relevant information in the latent representation and generating coherent output sequences.

Structure



- Encoder:
  - The task of the encoder is to transform the input (e.g., a sequence of words in a language) into a lower-dimensional but more meaningful latent representation or feature vector.
  - Each time step (or word) in the input is processed and encoded into an internal representation. Important information is captured in the latent representation.
  - Upon completion of the input sequence, the encoder produces a single vector (or set of vectors) that captures the key information of the input.

Structure

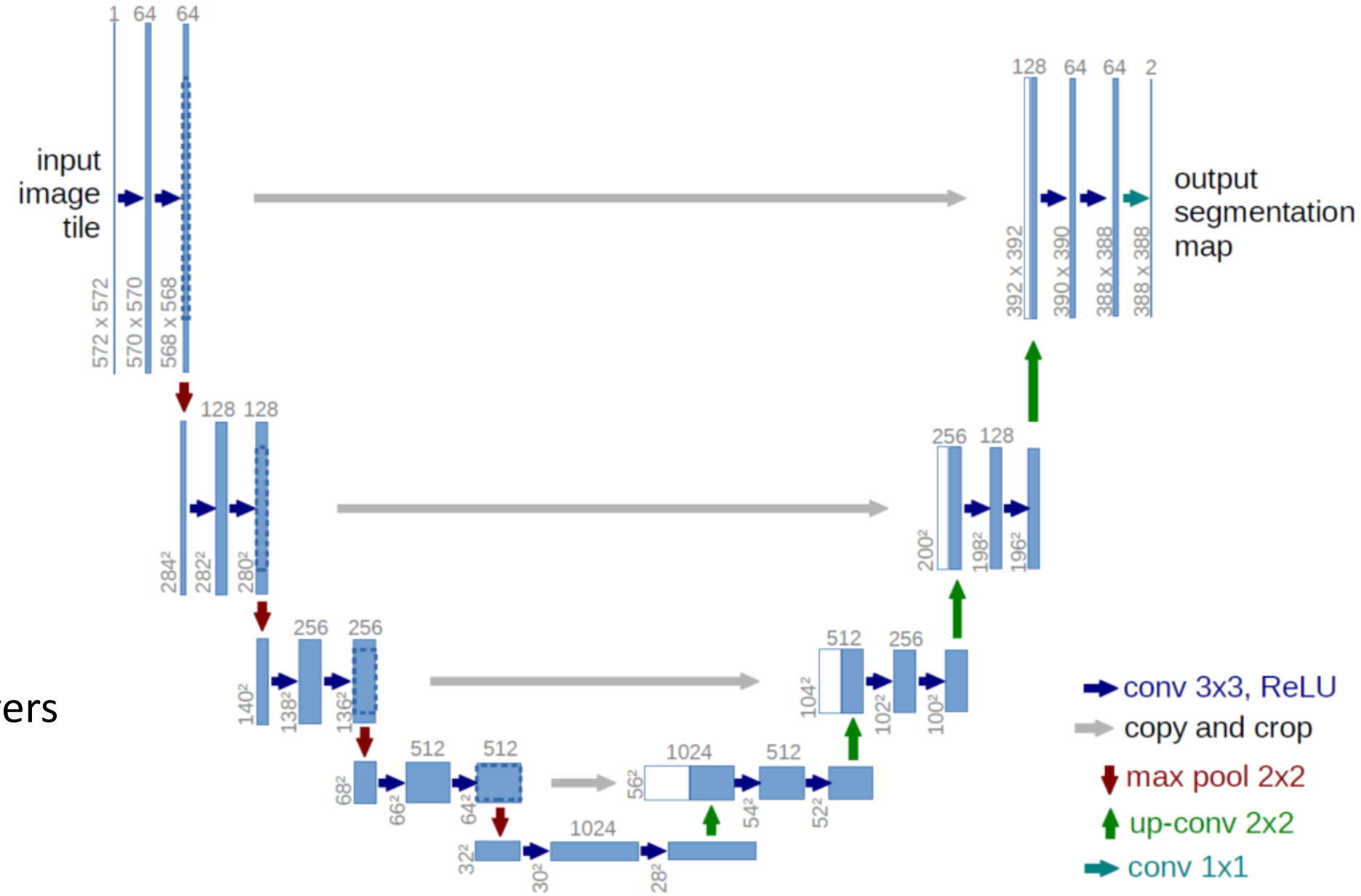


- Decoder:
  - The decoder takes the latent vector generated by the encoder and uses it to generate the desired output (for example, a sequence of words in another language).
  - It starts by generating the first part of the output using the latent vector and then, at each subsequent time step, incorporates information from the latent vector and the output generated up to that point.
  - The output is generated step by step and is adjusted based on the information learned by the encoder.

Training

- During the training phase, the network is fed known input-output pairs.
- The network learns to adjust the weights of its connections such that the output generated by the decoder is as close as possible to the desired output, taking into account the information captured by the encoder.
- A loss function is used to measure the discrepancy between the generated output and the desired output. This function guides the model during training to minimize error.

U-Net



- Famous encoder-decoder network
- Convolucional layers are employed

Spleeter

- It is a sound separation algorithm in songs based on the U-Net architecture.
- It is written in Python and Tensorflow.
- It allows sounds to be separated in 3 ways:
  - Vocals (singing voice) / accompaniment separation (2 stems)
  - Vocals / drums / bass / other separation (4 stems)
  - Vocals / drums / bass / piano / other separation (5 stems)
- Scientific References
  - https://www.theoj.org/joss-papers/joss.02154/10.21105.joss.02154.pdf
  - https://archives.ismir.net/ismir2019/latebreaking/000036.pdf

Spleeter (Tech details)

- It uses 12 U-Net layers, 6 for the encoder and 6 for the decoder.
- The loss function is an L1-norm between the input spectrogram and the output spectrograms.
- Proprietary databases with separate instruments and the full mixture were used to train.
- It took 1 week to train on a GPU.
- It can work in production at 100x on an RTX 2080 GPU
- https://github.com/deezer/spleeter (GITHUB)

Spleeter (MS-DOS version)

Usage: spleeter [options] <input_file_path>

Options:

-m, --model Spleeter model name
(2stems, 4stems, 5stems-16kHz, ..., default: 2stems)

-o, --output Output base file name
(in the format of filename.ext, default: <input_file_path>)

-b, --bitrate Output file bit rate
(128k, 192000, 256k, ..., default: 256k)

--overwrite Overwrite when the target output file exists

-h, --help Display this help and exit

-v, --version Display program version and exit

DataBases / Corpus

Environmental Sounds

- ESC-50
- https://github.com/karolpiczak/ESC-50
- 50 types of environmental sounds
- 40 examples per class
- Total 2000 sounds of maximum 5 seconds duration
- The 50 classes are grouped into 5 categories
- Is it possible to work in one category only (10 classes per cat.)

Musical Instruments

- BASIC
- https://github.com/seth814/Audio-Classification/
- 10 different musical instruments
- 30 samples of each instrument

- IRMAS
- 12 different musical instruments(including voice)
- https://zenodo.org/records/1290750
- 3,2 GB (6705 files)

Speech Emotion

- CREMAD
- https://www.kaggle.com/datasets/ejlok1/cremad/
- It is a data set of 7,442 original clips from 91 actors.
- 48 male and 43 female actors between the ages of 20 and 74
- Actors spoke from a selection of 12 sentences.
- The sentences have six different emotions
  - Anger, Disgust, Fear, Happy, Neutral, Sad

Music separated tracks by instruments

- MUSDB18
- 4 separated tracks
- 26 GB

- https://sigsep.github.io/datasets/musdb.html

Musical genres

- GTZAN
- 10 musical genres
- 100 songs per genre
- Features
- https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification/data
- Features computation
- https://www.kaggle.com/code/gastngadea/generate-features-csv-files-from-gtzan-dataset/notebook

Speech Emotion

- https://www.kaggle.com/code/ashishsingh226/speech-emotion-recognition-using-lstm/notebook

- Speech emotion / Mood with LSTM

Musical Genres

- https://www.kaggle.com/code/andradaolteanu/work-w-audio-data-visualise-classify-recommend

- Mixed features (MFCC & More)

- Clasic machine learning vs DNN
- Comparison
- Best: XGBoost

Genres of Popular Songs

- https://kaavyamaha12.medium.com/training-a-tensorflow-model-for-predicting-the-genres-of-popular-songs-4286d87d71f6