Deep Learning with Tensorflow in practice

Démarrage à 9h05
Conférence ENREGISTREE

Vincent Havard,

Enseignant-chercheur,
CESI LINEACT,

Rouen

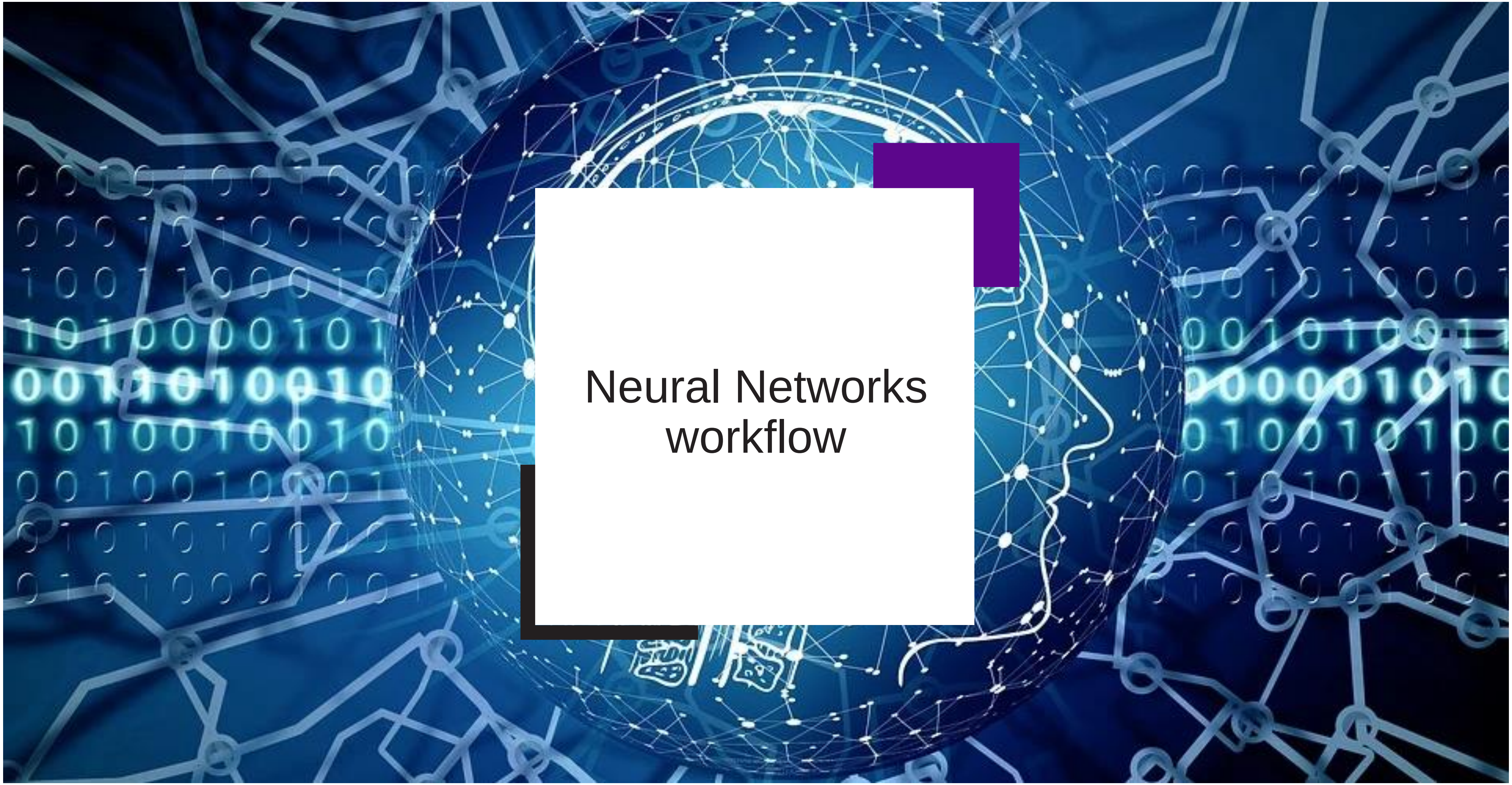
2022

cesi ÉCOLE D'INGÉNIEURS LINEACT cesi BANQUE des TERRITOIRES

« le projet DEFI&Co est cofinancé au titre du Programme d'Investissements d'Avenir lancé par l'Etat »

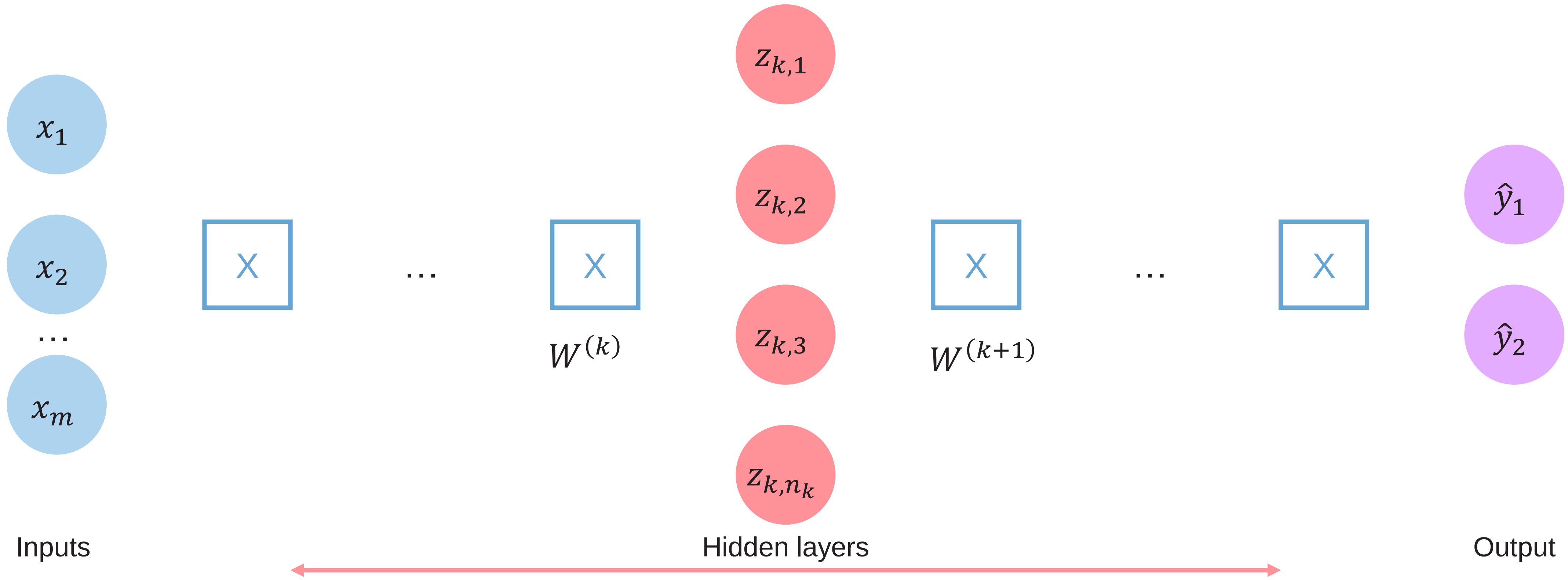
Outline

1. **Neural Networks workflow**
2. **Deep Learning Metrics**
3. **Conclusion**



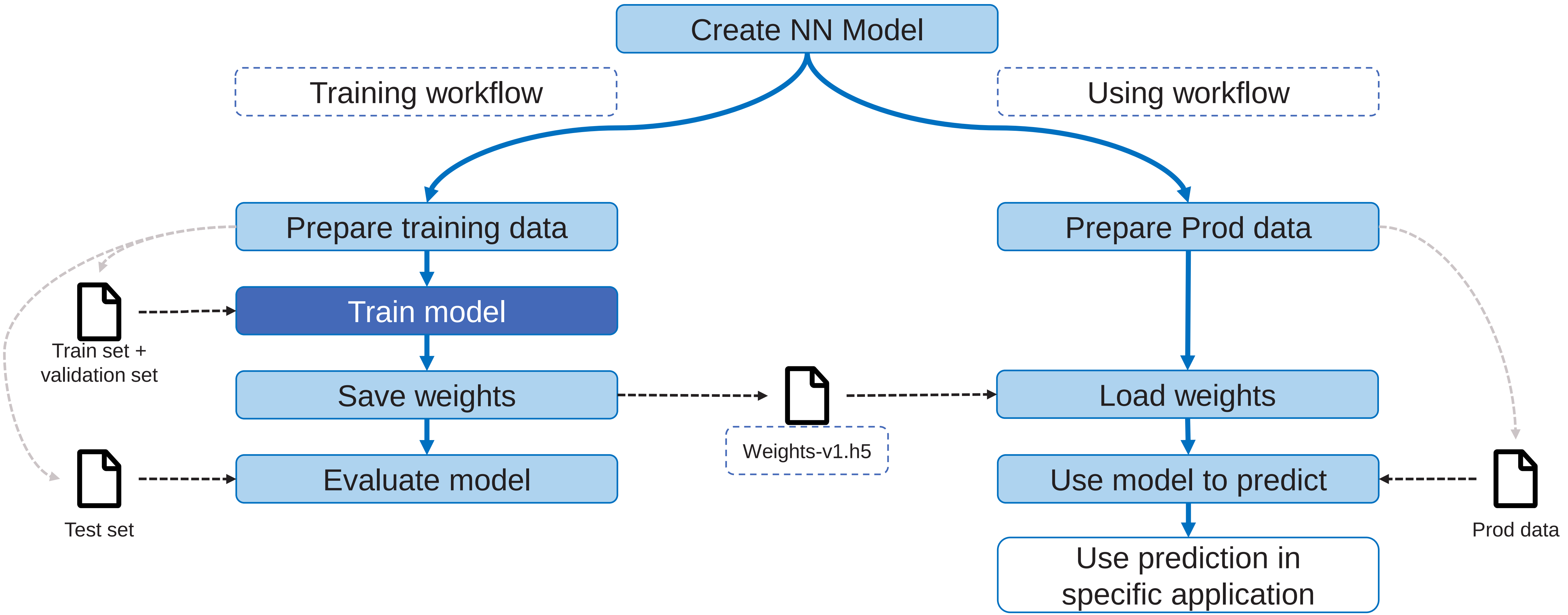
Neural Networks workflow

Deep Neural Network

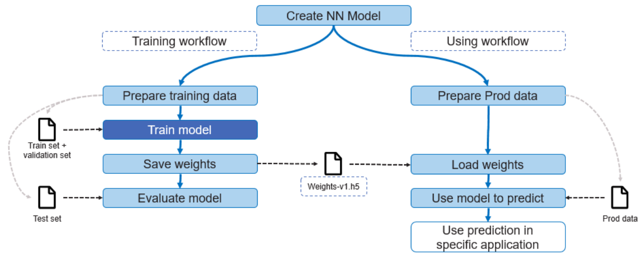


$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_1} g(z_{k-1,j}) w_{j,i}^{(k)}$$

Neural networks workflow



Neural networks workflow



How to **train** a neural network?

1. Prepare your training data in several sets *(training validation, test)*
2. Create your model
   - Define NN
   - Set the Loss function
     Depends on the problematic to solve
   - Set the optimizer
   - Set the metrics
   - and compile
3. Train the neural network on the training set and please wait…
4. Save the NN weights
5. Evaluate on the test set

How to **use** a neural network?

1. Prepare your data to use
2. Create the exactly same model
   - Define NN
   - Set the Loss function
     Depends on the problematic to solve
   - Set the optimizer
   - Set the metrics
   - and compile
3. Load the NN weights
4. *(classification only) transform into probability model*
5. Predict on the data given

Neural networks workflow

How to train a neural network?

1. Prepare your data in several sets *(training validation, test)*
2. Create your model
   - Define NN
   - Set the Loss function
     Depends on the problematic to solve
   - Set the metrics
   - and compile
3. Train the neural network on the training set and please wait…
4. Save the NN weights*
5. Evaluate on the test set

*Save the NN weights during training
https://www.tensorflow.org/tutorials/keras/save_and_load#checkpoint_callback_usage

How to train a neural network?

1. Prepare your data in several sets *(training set, validation set, test set)*

[18]:
```
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

2. Create your model
   - Define NN
   - Set the Loss function *(It depends on the problematic to solve)*
   - Set the metrics for evaluation
   - and compile

[16]:
```
def create_model(summary=False, loss_fn_to_use =
                 tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)):
    model = tf.keras.models.Sequential([
      tf.keras.layers.Flatten(input_shape=(28, 28)),
      tf.keras.layers.Dense(128, activation='relu'),
      tf.keras.layers.Dropout(0.2),
      tf.keras.layers.Dense(10)
    ])
    if summary:
        model.summary()
    # Set loss function to use with the model
    loss_fn = loss_fn_to_use

    # and compile
    model.compile(optimizer='adam', loss=loss_fn, metrics=['accuracy',
                                                  tf.metrics.SparseCategoricalAccuracy()])
    return model
```

3. Train the neural network on the training set …*and please wait*
   it is possible to save the NN weights during training see https://www.tensorflow.org/tutorials/keras/save_and_load#checkpoint_callback_usage
4. Save the Neural Network weights
5. Evaluate on the test set

[17]:
```
# Create your model
model = create_model(summary=True)
# Training from a few minutes to days
model.fit(x_train, y_train, epochs=5) # *you can save NN weights during training

# Save weights after training
model.save_weights('./mnist_model_0.98.h5')

# evaluate training on test set
model.evaluate(x_test,  y_test, verbose=2)
```

Neural networks workflow

How to use a neural network?

1. Prepare your data to use
2. Create the exactly same model
   - Define NN
   - Set the Loss function
     Depends on the problematic to solve
   - Set the metrics
   - and compile
3. Load the NN weights
4. *(classification only) transform into probability model*
5. Predict on the data given

How to use a neural network?

1. prepare your data

```
[29]: mnist = tf.keras.datasets.mnist

      (x_train, y_train), (x_test, y_test) = mnist.load_data()
      x_train, x_test = x_train / 255.0, x_test / 255.0
```

2. Create your model
   - Define NN
   - Set the Loss function *(It depends on the problematic to solve)*
   - Set the metrics for evaluation
   - and compile
3. Load the NN weights

```
[ ]: model = create_model(summary=False)

     model.load_weights('./mnist_model_0.98.h5')
     # evaluate to check the model is well loaded
     model.evaluate(x_test,  y_test, verbose=2)
```

4. (classification only) transform into probability model
5. Predict on the data given

```
[ ]: # probability model
     probability_model = tf.keras.Sequential([model, tf.keras.layers.Softmax()])
```

```
[44]: # Prepare the data given
      im_to_test = x_test[0]
      # only when testing on 1 unique sample
      im_to_test = np.expand_dims(im_to_test, axis=0)

      # NOT UNDERSTANDABLE approach
      model_output = model.predict(im_to_test)
      model_output_class = np.argmax(model_output)
      print(model_output)
      print(model_output_class)

      # UNDERSTANDABLE approach
      prob_model_output = probability_model.predict(im_to_test)
      prob_model_output_class = np.argmax(prob_model_output)
      print(prob_model_output)
      print(prob_model_output_class)

[[ -6.098753  -13.523815   -4.2792406   1.8365132 -20.36935    -3.3478541
  -22.223114   13.569721   -5.958574   -1.6038301]]
7
[[2.8713529e-09 1.7116818e-12 1.7712962e-08 8.0228556e-06 1.8215655e-15
  4.4955883e-08 2.8534189e-16 9.9999166e-01 3.3034322e-09 2.5716190e-07]]
7
```

One hot Encoding label for classification

One-hot encoding is a way to represent label for classification

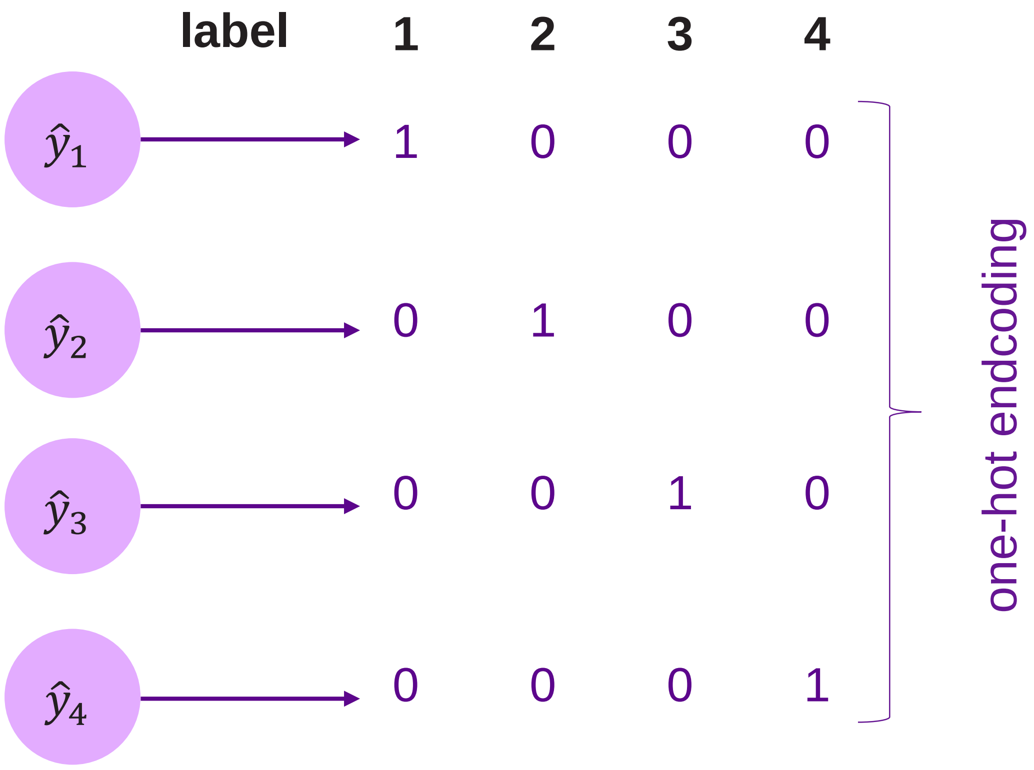
In tensorflow, when you use:

- **CategoricalCrossEntropy**: the label must be encoded in one-hot representation

  y_true = [1,2,3,4,4,2,1, …]

- **SparseCategoricalCrossEntropy**: the label must be in integer and it is <u>automatically</u> converted into on-hot encoding

  y_true = [[1,0,0,0],
  [0,1,0,0],
  [0,0,1,0],
  [0,0,1,0],
  [0,0,0,1],
  [0,1,0,0],
  [1,0,0,0]]

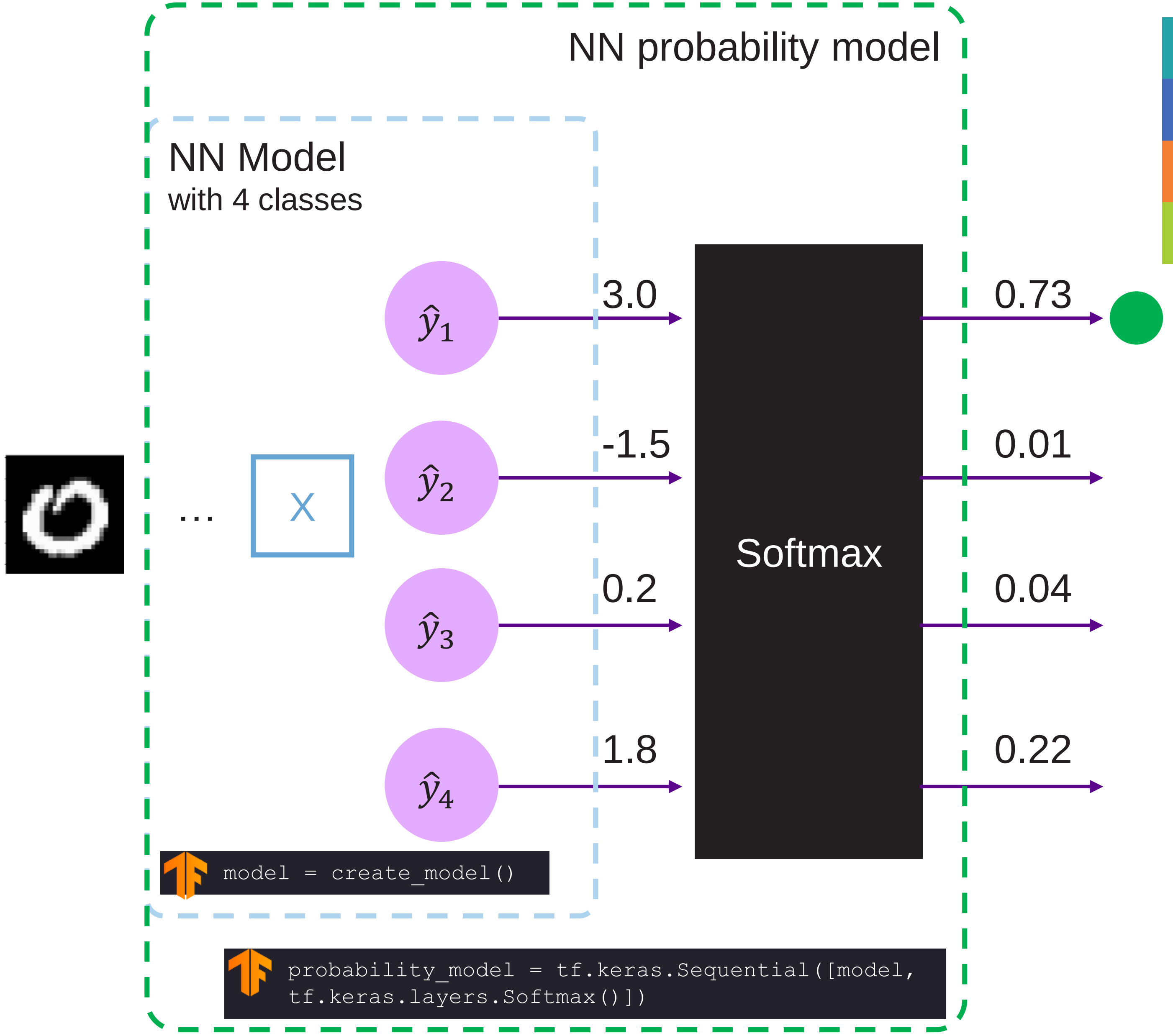


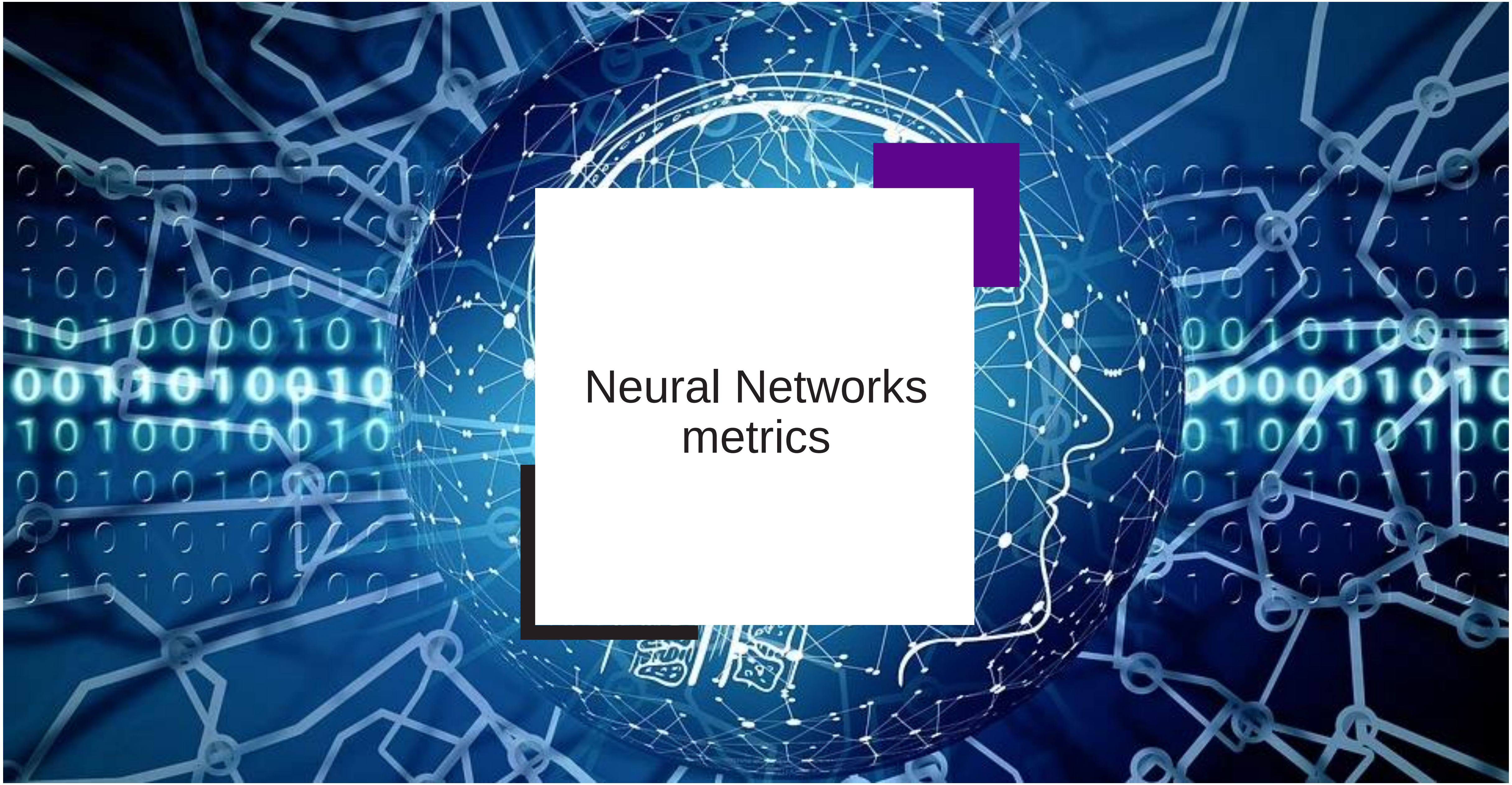
What is a softmax layer?

It transforms a vector of value into a vector of probability value.

For a Neural network with K classes, it is.

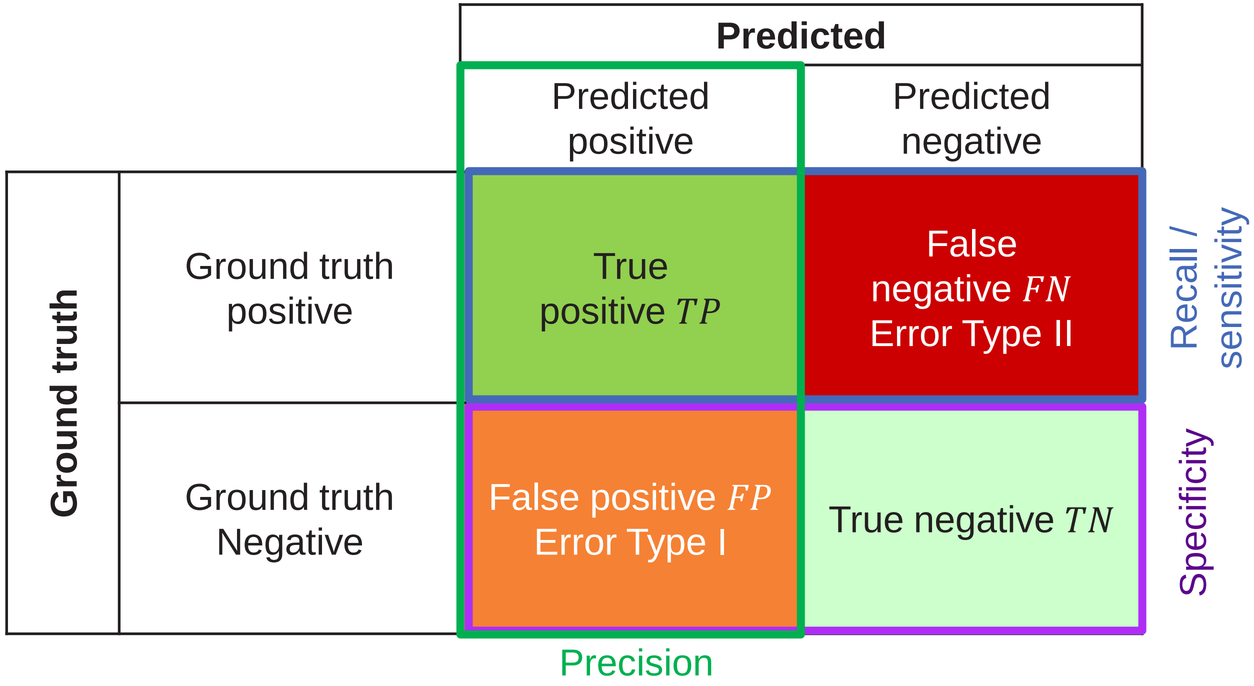
$$\boldsymbol{softmax}(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, for\ i = 1 \ldots K$$





Neural Networks metrics

Binary classification metrics



$$Accuracy = \frac{TP+TN}{ALL}$$

$$precision = \frac{TP}{Pred\ positive} = \frac{TP}{TP+FP}$$

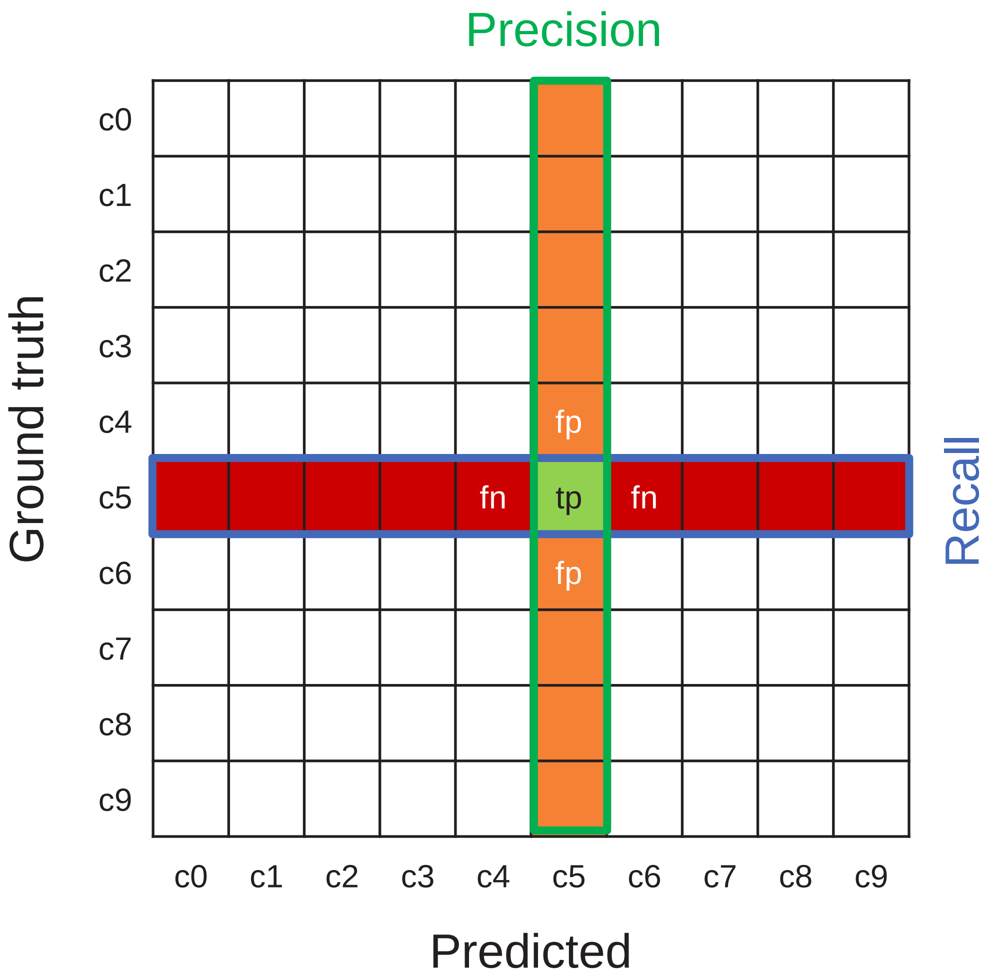
$$TPR = recall = sensitivity = \frac{TP}{GT\ positive} = \frac{TP}{TP+FN}$$

$$specificity = \frac{TN}{GT\ negative} = \frac{TN}{FP+TN}$$

$$FPR = \frac{FP}{GT\ negative} = \frac{FP}{FP+TN}$$

$$F1_{score} = \frac{2 \cdot precision \cdot recall}{precision+recall}$$

Multi classes classification metrics – Confusion matrix



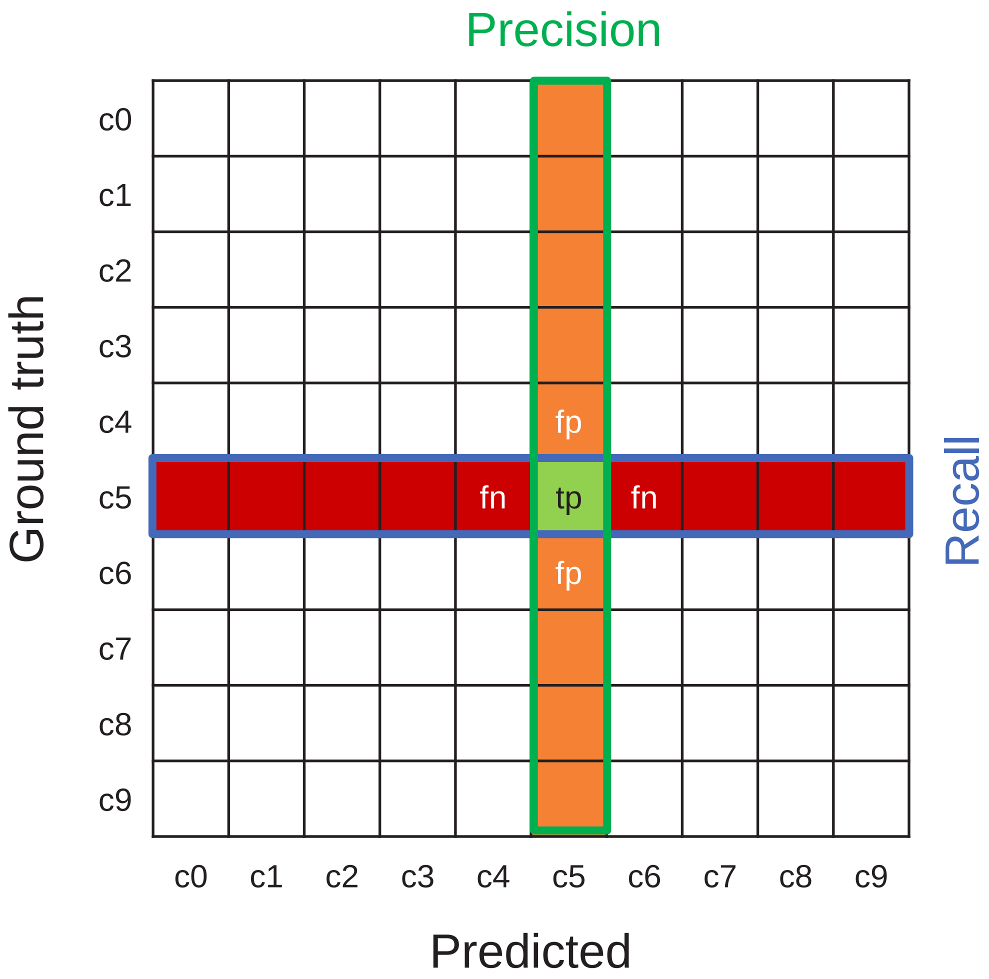
True positive, false positive and false negative on confusion matrix on **c5** class

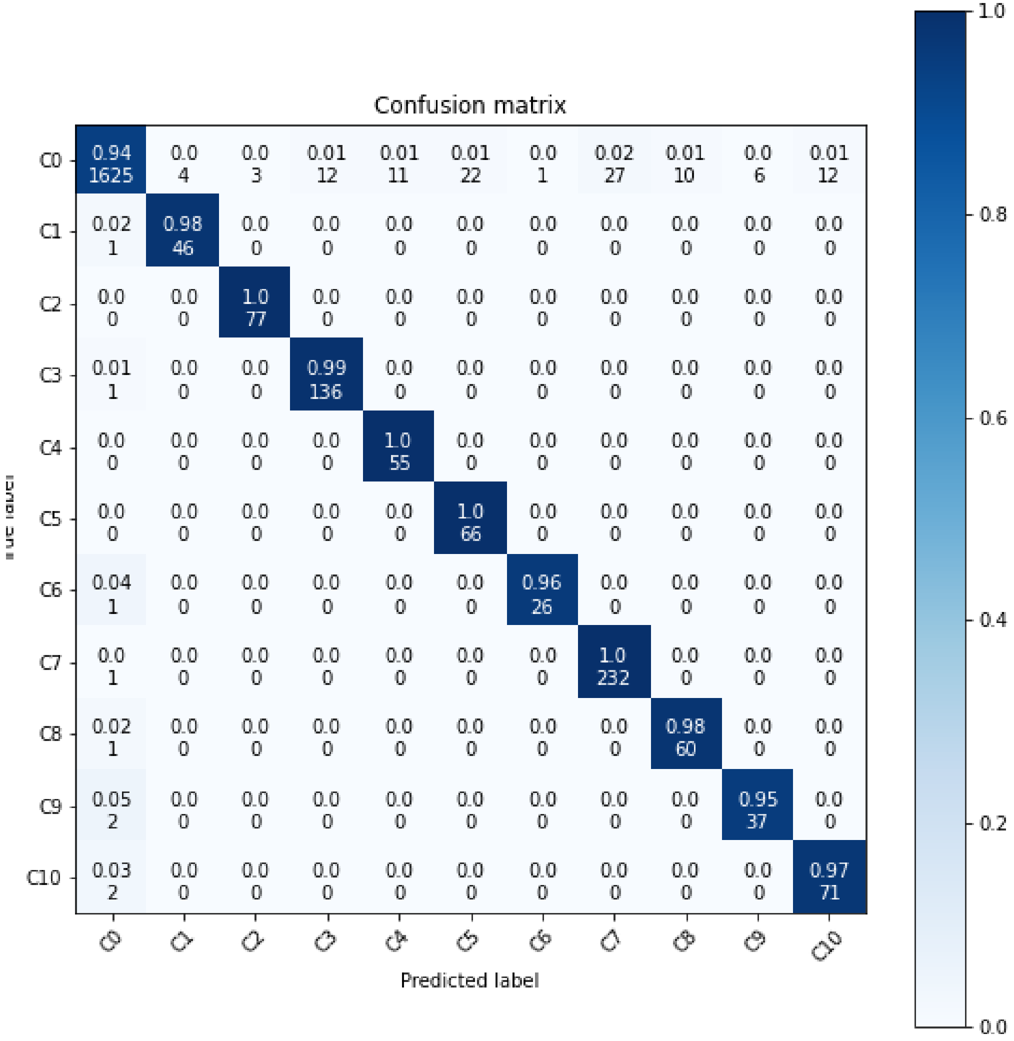
tp: true positive
fn: false negative

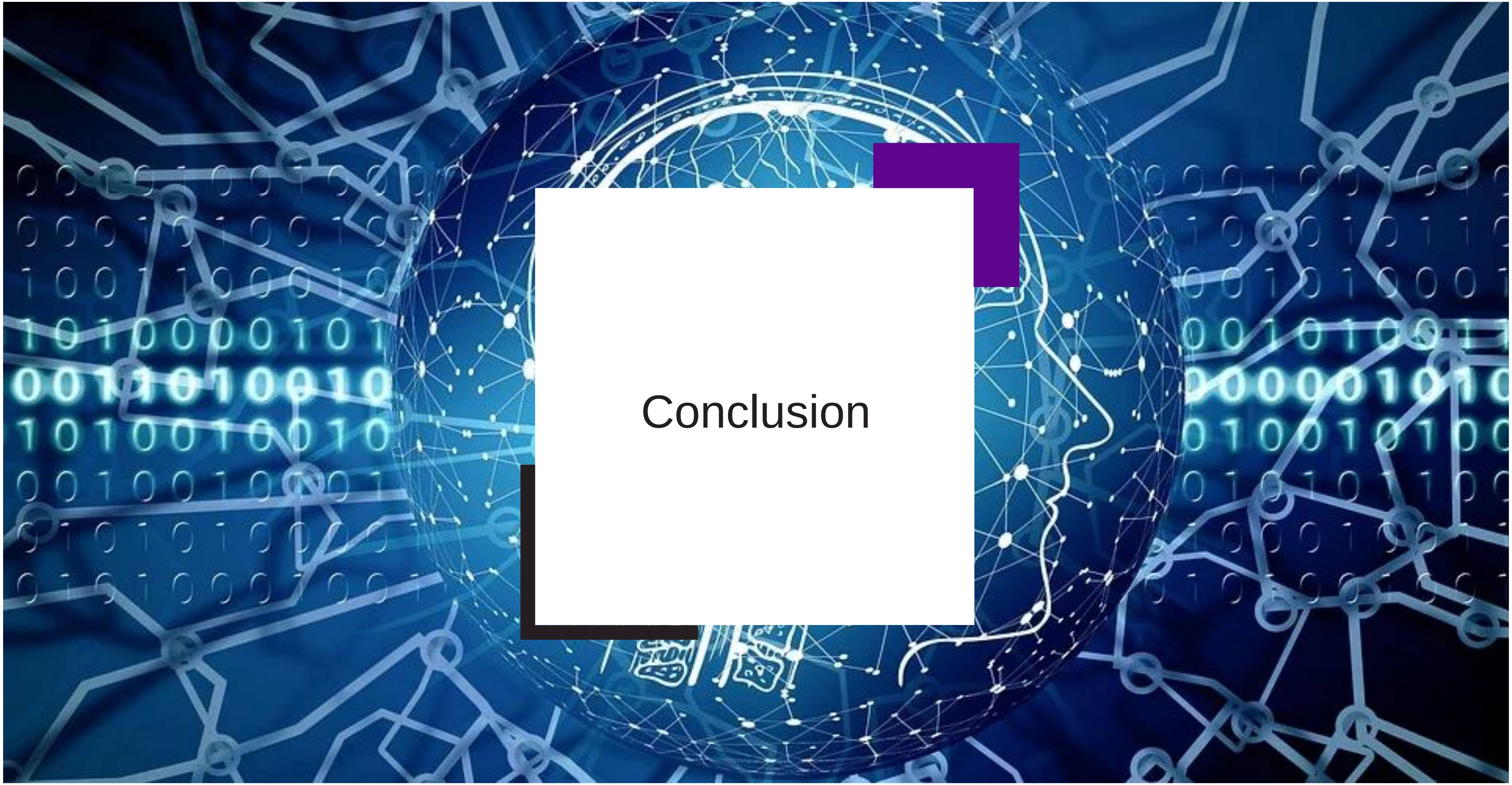
fp: false positive
tn: true negative

True negative is not computable with this representation.

Neural Networks evaluation on classification task with confusion matrix representation





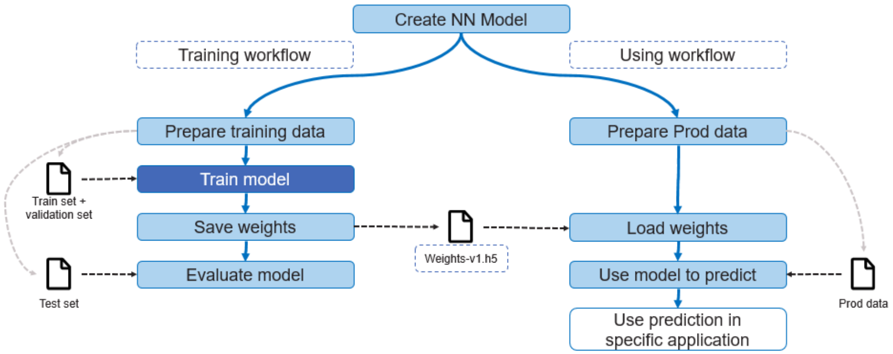


Conclusion

Conclusion

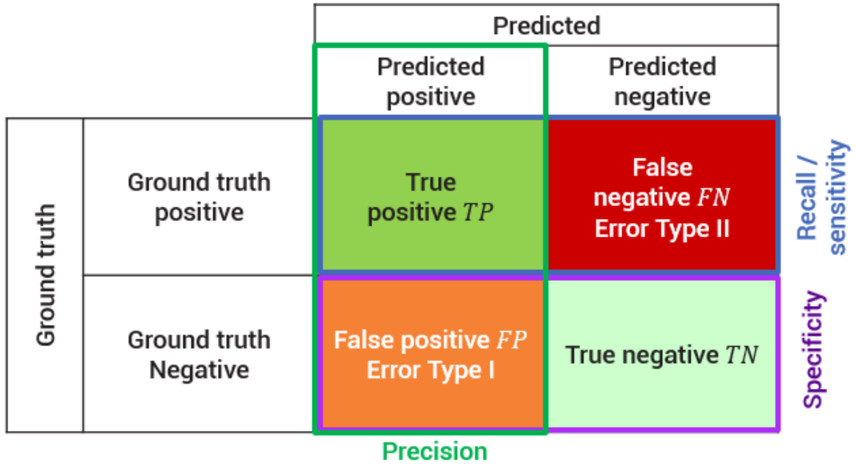
Neural Networks creation workflow

- **Prepare** your data
- **Create** your model
- **Train** the neural network
- **Save** the NN weights
- **Evaluate** on the test set



Evaluating Neural Networks

- Accuracy
- Precision
- Recall = sensitivity
- Specificity
- F1 score



References

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Van Veen, F. & Leijnen, S. (2019). The Neural Network Zoo. Retrieved from https://www.asimovinstitute.org/neural-network-zoo

A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, “A Survey of the Recent Architectures of Deep Convolutional Neural Networks,” *Artif Intell Rev*, vol. 53, no. 8, pp. 5455–5516, Dec. 2020, doi: 10.1007/s10462-020-09825-6.

"Anatomy and Physiology" by the US National Cancer Institute's Surveillance, Epidemiology and End Results (SEER) Program . Neuron description, Licence CC BY-SA 3.0

H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, “Visualizing the Loss Landscape of Neural Nets,” in *Advances in Neural Information Processing Systems*, 2018, vol. 31, pp. 6389–6399, [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/a41b3bb3e6b050b6c9067c67f663b915-Paper.pdf.

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1, No. 2). Cambridge: MIT press.

References

Datasets:

A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, T. Duerig, and V. Ferrari. The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale. IJCV, 2020.

R. Benenson, S. Popov, and V. Ferrari. Large-scale interactive object segmentation with human annotators. CVPR, 2019.

Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. (* = equal contribution) **ImageNet Large Scale Visual Recognition Challenge**. *IJCV,* 2015

J. Carreira, E. Noland, C. Hillier, and A. Zisserman, *A Short Note on the Kinetics-700 Human Action Dataset*. 2019.

Ambika Choudhury, 10 Open Datasets You Can Use For Computer Vision Projects, available at https://analyticsindiamag.com/10-open-datasets-you-can-use-for-computer-vision-projects/

J. Fritsch, T. Kuehnl, and A. Geiger, “A New Performance Measure and Evaluation Benchmark for Road Detection Algorithms,” 2013.

A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite,” 2012.

M. Menze and A. Geiger, “Object Scene Flow for Autonomous Vehicles,” 2015.

A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets Robotics: The KITTI Dataset,” *International Journal of Robotics Research (IJRR)*, 2013.

Google Research, available at https://research.google/tools/datasets/, access on Nov. 10, 2019