

# Demystifying Artificial Intelligence Sorcery

(Part 3: Deep Learning)<sup>a</sup>

---

Abdelbacet Mhamdi  
abdelbacet.mhamdi@bizerte.r-iset.tn

*Dr.-Ing. in Electrical Engineering*  
*Senior Lecturer at ISET Bizerte*

---

<sup>a</sup>Available @ <https://github.com/a-mhamdi/isetbz/>



## Disclaimer

This document features some materials gathered from multiple online sources.

Please note no copyright infringement is intended, and I do not own nor claim to own any of the original materials. They are used for educational purposes only.

I have included links solely as a convenience to the reader. Some links within these slides may lead to other websites, including those operated and maintained by third parties. The presence of such a link does not imply a responsibility for the linked site or an endorsement of the linked site, its operator, or its contents.

## ROADMAP

1. An overview
2. Convolutional Neural Network
3. Generative Adversarial Network
4. Variational Autoencoder
5. Natural Language Processing
6. Transfer Learning
7. Quizzes

## **An overview**

---



**REMINDER**

## PROGRAMMING LANGUAGE

[julia-lang.org/](http://julia-lang.org/)

## DEVELOPMENT ENVIRONMENTS



**Pluto.jl**



▲ \$ docker compose up

▼ \$ docker compose down



# JULIA IN A NUTSHELL

- ▲ Fast
- ▲ Dynamic
- ▲ Reproducible
- ▲ Composable
- ▲ General
- ▲ Open Source





# JULIA MICRO-BENCHMARKS (1/2)



<https://julialang.org/benchmarks>



## JULIA MICRO-BENCHMARKS (2/2)

### Geometric Means of Micro-Benchmarks by Language

1	C	1.0
2	Julia	1.17006
3	LuaJIT	1.02931
4	Rust	1.0999
5	Go	1.49917
6	Fortran	1.67022
7	Java	3.46773
8	JavaScript	4.79602
9	Matlab	9.57235
10	Mathematica	14.6387
11	Python	16.9262
12	R	48.5796
13	Octave	338.704





# SOURCE CONTROL MANAGEMENT (SCM)

The screenshot shows the GitHub web interface for the repository 'a-mhamdi/jlai'. The repository is public and has 0 forks and 0 stars. The 'Code' tab is selected, showing a commit history table. The table lists the following files and their commit details:

File	Commit Message	Commit Hash	Time Ago
.github/workflows	fix typo.	996ee27	27 minutes ago
toml	sync *.toml files		last month
.gitignore	add .gitignore file		last month
Dockerfile	change repo's name & references		15 days ago
LICENSE	Initial commit		last month
README.md	ref. to jlai @ dockerhub		37 minutes ago
docker-compose.yml	change repo's name & references		15 days ago
sync-script.sh	sync *.toml files		last month

The repository description is 'Image of julia on ubuntu to run labs of AI.' The 'About' section shows the Readme, MIT license, 0 stars, 1 watching, and 0 forks. The 'Releases' section shows 'No releases published' and a link to 'Create a new release'. The 'Packages' section is also visible.

<https://github.com/a-mhamdi/jlai>



# CONTINUOUS INTEGRATION (CI)

The screenshot shows the Docker Hub interface for the repository `abmhamdi/jlai`. The page includes a search bar, navigation tabs (General, Tags, Builds, Collaborators, Webhooks, Settings), and a description of the repository as 'Artificial Intelligence Labs @ ISETBZ'. It also displays Docker commands for pushing a new tag, a table of tags and scans, and information about automated builds.

**abmhamdi /jlai**

**Description**  
Artificial Intelligence Labs @ ISETBZ  
Last pushed: 2 minutes ago

**Docker commands**  
To push a new tag to this repository,  
`docker push abmhamdi/jlai:tagname`

**Tags and scans**  
This repository contains 1 tag(s).  
VULNERABILITY SCANNING - DISABLED [Enable](#)

Tag	OS	Type	Pulled	Pushed
latest	linux	Image	—	2 minutes ago

[See all](#) [Go to Advanced Image Management](#)

**Automated Builds**  
Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.  
Available with Pro, Team and Business subscriptions.  
[Upgrade](#) [Learn more](#)

<https://hub.docker.com/r/abmhamdi/jlai>

## Convolutional Neural Network

---

## CONVNETJS DEMO (1/3)

A Convolutional Neural Network (CNN) is a type of neural network that is particularly well-suited for image classification and object recognition tasks. It is designed to process data with a grid-like topology, such as an image, and is composed of multiple layers that learn to extract features from the input data.

CNNs are composed of several types of layers, including convolutional layers, pooling layers, and fully connected layers. The convolutional layers apply filters to the input data, which are used to detect patterns and features in the data. The pooling layers reduce the spatial dimensions of the data, which helps to reduce the complexity of the model and make it more robust to small translations of the input data. The fully connected layers combine the features learned by the convolutional and pooling layers to make a prediction.

CNNs have been successful in a variety of tasks, including image classification, object detection, and segmentation. They are a powerful tool in the field of artificial intelligence and are widely used in applications such as image and video analysis, robotics, and natural language processing.

Convolutional neural networks (CNNs) are a type of neural network that are particularly well-suited for image-based tasks, such as image classification, object detection, and image segmentation. Here are a few examples of how CNNs have been used in the real world:

**Image classification** CNNs can be used to classify images into different categories, such as identifying objects in an image or labeling the scene depicted in an image.

## CONVNETJS DEMO (2/3)

**Object detection** CNNs can be used to detect objects in images or videos and to localize them by drawing bounding boxes around them.

**Image segmentation** CNNs can be used to segment images into different regions, such as separating the foreground from the background or labeling different objects in an image.

**Medical image analysis** CNNs have been used to analyze medical images, such as CT scans and X-rays, for tasks such as tumor detection and segmentation.

**Self-driving cars** CNNs have been used to process camera images in self-driving cars to detect pedestrians, other vehicles, and traffic signals.

**Robotics** CNNs have been used to process images from robots to enable them to navigate their environment and perform tasks such as grasping objects.

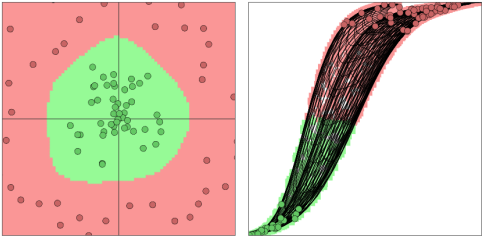
**Natural language processing** CNNs have been used to process text data for tasks such as sentiment analysis and language translation.

These are just a few examples of how CNNs have been used in the real world. CNNs are a powerful tool for image-based tasks and have many potential applications in a wide range of fields.

# CONVNETJS DEMO (3/3)

Feel free to change this, the text area above gets eval()'d when you hit the button and the network gets reloaded. Every 10th of a second, all points are fed to the network multiple times through the trainer class to train the network. The resulting predictions of the network are then "painted" under the data points to show you the generalization.

On the right we visualize the transformed representation of all grid points in the original space and the data, for a given layer and only for 2 neurons at a time. The number in the bracket shows the total number of neurons at that level of representation. If the number is more than 2, you will only see the two visualized but you can cycle through all of them with the cycle button.



simple data   circle data   spiral data

random data

Controls:  
 CLICK: Add red data point  
 SHIFT+CLICK: Add green data point  
 CTRL+CLICK: Remove closest data point

Go [back to ConvNetJS](https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html)

drawing neurons 0 and 1 of layer with index 4 (tanh)

fc(6)   tanh(6)   fc(2)   **tanh(2)**

fc(2)

cycle through visualized neurons at selected layer (if more than 2)

<https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html>





Code is available at <https://github.com/a-mhamdi/isetbz/>  
→ Artificial Intelligence → Codes → Julia → Part-3 → cnn.jl



## **Generative Adversarial Network**

---

## GENERATIVE ADVERSARIAL NETWORK (1/2)

A Generative Adversarial Network (GAN) is a type of deep learning model designed to generate new, synthetic samples of data. It consists of two networks: a generator network and a discriminator network. The generator network generates synthetic samples, while the discriminator network tries to distinguish between the synthetic samples and real samples of data.

During training, the generator and discriminator networks are trained concurrently, with the generator trying to generate synthetic samples that are indistinguishable from real samples, and the discriminator trying to correctly classify the samples as either real or synthetic. The generator is trained to improve its synthetic samples based on the feedback from the discriminator, and the discriminator is trained to become more sensitive to synthetic samples.

The goal of a GAN is to learn a generative model that can produce synthetic samples that are similar to the training data. GANs have been used for a variety of tasks, including image generation, audio synthesis, and natural language processing.

GANs have been used for a variety of tasks in the real world, including:

**Image generation** GANs can be used to generate realistic images of objects, people, and scenes.

**Image style transfer** GANs can be used to transfer the style of one image to another image, while preserving the content of the original image.

## GENERATIVE ADVERSARIAL NETWORK (2/2)

**Text-to-image synthesis** GANs can be used to generate images from textual descriptions, such as generating images of objects based on their names.

**Video prediction** GANs can be used to predict future frames in a video, given the past frames.

**Text-to-speech synthesis** GANs can be used to generate speech audio from text.

**Super-resolution** GANs can be used to enhance the resolution of images or videos.

**Data augmentation** GANs can be used to generate additional training data for other machine learning models.

**Medical image analysis** GANs have been used to generate synthetic medical images for tasks such as segmentation and classification.

**Domain adaptation** GANs can be used to translate images from one domain (e.g., synthetic images) to another domain (e.g., real images) to improve the performance of machine learning models.

These are just a few examples of how GANs have been used in the real world. GANs are a powerful tool for generating synthetic data and have many potential applications in a wide range of fields.



Code is available at <https://github.com/a-mhamdi/isetbz/>  
→ Artificial Intelligence → Codes → Julia → Part-3 → gan.jl



## Variational Autoencoder

---

## VARIATIONAL AUTO-ENCODER (1/3)

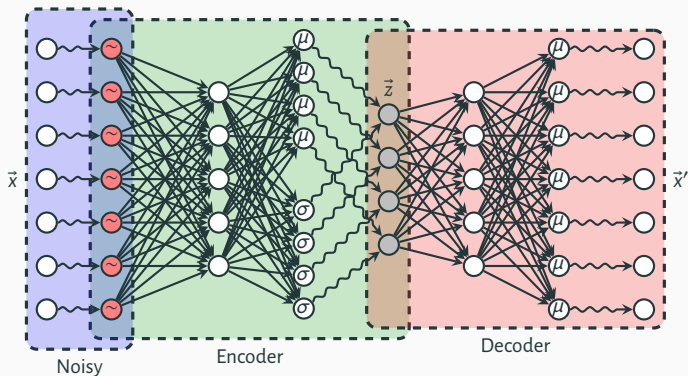
A variational autoencoder (VAE) is a type of deep learning model that is used to learn latent representations of data. It is a generative model, which means that it can generate new samples of data that are similar to the training data.

VAEs are trained to encode the data into a low-dimensional latent space and then decode the latent representation back into the original data space. During training, the VAE learns to reconstruct the input data, while also trying to enforce a constraint on the latent space that encourages it to represent the data in a meaningful way.

The constraint that is used in a VAE is called the variational lower bound. This lower bound is maximized during training, which encourages the latent space to be structured in a way that is useful for generating samples that are similar to the training data.

VAEs have been used for a variety of tasks, including image generation, natural language processing, and unsupervised anomaly detection.

## VARIATIONAL AUTO-ENCODER (2/3)



**Generative modeling** VAEs can be used to generate new samples of data that are similar to the training data. This can be useful for tasks such as image generation, audio synthesis, and natural language generation.



## VARIATIONAL AUTO-ENCODER (3/3)

- Anomaly detection** VAEs can be used to identify anomalies in data by reconstructing the input data and measuring the reconstruction error. Data points that are poorly reconstructed are likely to be anomalous.
- Data compression** VAEs can be used to compress data by encoding it into a lower-dimensional latent space and then reconstructing it. The encoder can be used as a compression function, and the decoder can be used as a decompression function.
- Representation learning** VAEs can be used to learn meaningful latent representations of data, which can be useful for tasks such as clustering and classification.
- Semi-supervised learning** VAEs can be used in a semi-supervised setting, where only a small portion of the data is labeled. The VAE can use the labeled data to learn a meaningful latent representation of the data, and then use this representation to make predictions on the unlabeled data.



Code is available at <https://github.com/a-mhamdi/isetbz/>  
→ Artificial Intelligence → Codes → Julia → Part-3 → vae.jl



## Natural Language Processing

---

## NATURAL LANGUAGE PROCESSING (1/2)

Natural Language Processing (NLP) is a field of artificial intelligence and computer science that focuses on the interaction between computers and humans using natural language. It involves the development of algorithms and models that can understand, interpret, and generate human language.

NLP is used in a wide range of applications, including machine translation, question answering, text summarization, text classification, and sentiment analysis. Some examples of NLP tasks include:

Part-of-speech tagging: Identifying the parts of speech (e.g., noun, verb, adjective) in a sentence

Named entity recognition: Identifying and labeling named entities (e.g., people, organizations, locations) in a text

Sentiment analysis: Determining the sentiment (e.g., positive, neutral, negative) of a piece of text

Machine translation: Translating text from one language to another

Text summarization: Generating a concise summary of a longer piece of text

NLP is a rapidly evolving field and has seen significant progress in recent years due to advances in machine learning and computational power.

Here's some pseudocode that outlines the general process of performing NLP tasks in Julia:

1. Preprocess the text data by lowercasing, removing punctuation, and splitting the text into individual tokens (e.g., words or subwords).
2. Build a vocabulary of the most common tokens in the text data.

## NATURAL LANGUAGE PROCESSING (2/2)

3. Encode the text data as a sequence of integers using the vocabulary.
4. Pad the encoded sequences to the same length to make them suitable for input to a model.
5. Define the NLP model using a library such as Flux.jl or Knet.jl.
6. Train the model using gradient descent and a suitable loss function.
7. Use the trained model to make predictions on new data.

\*\*\*



Code is available at <https://github.com/a-mhamdi/isetbz/>  
→ Artificial Intelligence → Codes → Julia → Part-3 → nlp.jl



## Transfer Learning

---

## TRANSFER LEARNING (1/4)

Transfer learning is a machine learning technique in which a model that has been trained on one task is re-purposed on a second related task. Transfer learning can be used to improve the performance of the second task by leveraging the knowledge learned from the first task.

One common use of transfer learning is to fine-tune a pre-trained model on a new dataset. For example, a pre-trained image classification model that has been trained on a large dataset such as ImageNet can be fine-tuned on a smaller dataset of a different but related task, such as detecting objects in medical images. Fine-tuning the pre-trained model on the new dataset can lead to improved performance compared to training a model from scratch on the smaller dataset.

Transfer learning is useful because it allows a machine learning model to learn from a large amount of data, even if the data is not directly related to the task at hand. It can also be used to speed up the training process, since the model does not need to be trained from scratch. Transfer learning has been applied to a variety of tasks, including image classification, natural language processing, and speech recognition.

Here are a few examples of how transfer learning has been applied in the real world:

**Image classification** Transfer learning has been used to fine-tune pre-trained image classification models on new datasets, such as identifying plant species from images of leaves or detecting objects in medical images.



## TRANSFER LEARNING (2/4)

**Natural language processing** Transfer learning has been used to fine-tune pre-trained language models on new tasks, such as sentiment analysis or text classification.

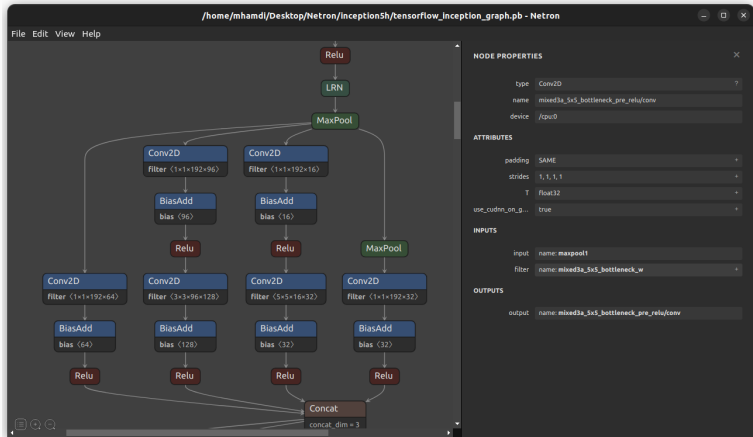
**Speech recognition** Transfer learning has been used to fine-tune pre-trained speech recognition models on new languages or accents.

**Computer vision** Transfer learning has been used to fine-tune pre-trained models for tasks such as object detection and image segmentation.

**Robotics** Transfer learning has been used to fine-tune pre-trained models for tasks such as grasping objects or navigating a new environment.

These are just a few examples of how transfer learning has been applied in the real world. Transfer learning is a powerful tool that can be used to improve the performance of machine learning models on new tasks by leveraging knowledge learned from previous tasks.

# TRANSFER LEARNING (3/4)



<https://github.com/lutzroeder/netron>

Here's some pseudocode that outlines the general process of transfer learning in Julia:

1. Load the pre-trained model (e.g., a convolutional neural network trained on ImageNet).

## TRANSFER LEARNING (4/4)

2. Replace the final layer (or layers) of the pre-trained model with a new, untrained layer (or layers) that is suitable for your target task.
3. Freeze the weights of the pre-trained layers to prevent them from being updated during training.
4. Load your dataset and split it into training and validation sets.
5. Use the training set to fine-tune the weights of the new layer (or layers) using gradient descent and a suitable loss function.
6. Monitor the performance of the model on the validation set and adjust the hyperparameters (e.g., learning rate) as needed.
7. When you're satisfied with the performance of the model on the validation set, you can use it to make predictions on the test set or on new data.

# TRANSFER LEARNING

CODE SNIPPET



Code is available at <https://github.com/a-mhamdi/isetbz/>  
→ Artificial Intelligence → Codes → Julia → Part-3 → transfer-learning.jl



## Quizzes

---

## MCQ (1/1)

1. Your supervisor asks you to create a machine learning system that will help your human resources department classify jobs applicants into well-defined groups. What type of system are you more likely to recommend?
  - ✗ an unsupervised machine learning system that clusters together the best candidates.
  - ✗ you would not recommend a machine learning system for this type of project.
  - ✗ a deep learning artificial neural network that relies on petabytes of employment data.
  - ✓ a supervised machine learning system that classifies applicants into existing groups.
2. Your data science team must build a binary classifier, and the number one criterion is the fastest possible scoring at deployment. It may even be deployed in real time. Which technique will produce a model that will likely be fastest for the deployment team use to new cases?
  - ✗ random forest
  - ✓ logistic regression
  - ✗ KNN
  - ✗ deep neural network
3. The famous data scientist Andrew Ng has been quoted as saying, "Applied machine learning is basically feature engineering." What is feature engineering?
  - ✗ scraping new features from web data
  - ✓ creating new variables by combining and modifying the original variables
  - ✗ designing innovative new user features to add to software
  - ✗ using deep learning to find features in the data

## SOME USEFUL LINKS

1. <https://setosa.io/ev/>
2. <https://karpathy.ai/>
3. <http://yann.lecun.com/>
4. <https://www.hackingnote.com/>
5. <https://machinelearningmastery.com/>
6. <https://stanford.edu/~shervine/teaching/>
7. <https://www.ibm.com/downloads/cas/GB8ZMQZ3>
8. <https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/>

## FURTHER READING (1/1)