

# 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/jlai/>



## 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.

1. An overview
2. CNN, VAE, GAN & NLP
3. Transfer Learning
4. Reinforcement Learning
5. Responsible AI
6. Quizzes

## **An overview**

---

- CNNs** (*Convolutional Neural Networks*) are used for image classification and other computer vision tasks because they are able to automatically learn features from raw data. This is useful for tasks where manual feature engineering is difficult or impractical.
- VAEs** (*Variational Autoencoders*) are used for tasks such as image generation and anomaly detection because they are able to learn a compact representation of a dataset and generate new samples from this representation.
- GANs** (*Generative Adversarial Networks*) are used for tasks such as image generation and data augmentation because they are able to generate new data samples that are similar to a given dataset.
- NLP** (*Natural Language Processing*) is important for tasks such as language translation, text classification, and language generation because it allows computers to process and understand human language.

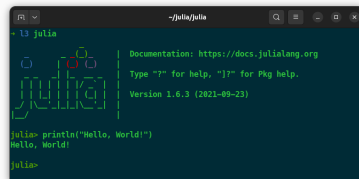


**REMINDER**

# PROGRAMMING LANGUAGE



[julialang.org/](https://julialang.org/)

A screenshot of a terminal window titled "~julia/julia". The prompt is "+ julia". The terminal shows a stylized ASCII art logo of the word "julia" with colored circles. To the right of the logo, the following text is displayed: "Documentation: https://docs.julialang.org", "Type '?' for help, ']' for pkg help.", and "Version 1.6.3 (2021-09-23)". Below this, the user enters the command "julia> println(\"Hello, World!\")" and the output "Hello, World!" is shown. The prompt "julia>" is visible at the bottom.

## 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 repository page for 'a-mhamdi/jlali'. The repository is public and has 2 stars and 3 forks. The main branch is 'main'. The repository contains a README.md file, a LICENSE file, and a .gitignore file. The README.md file is titled 'Fuzzy Logic, Machine Learning and Deep Learning with Julia' and contains a table of contents with links to various sections. The table of contents includes:

- [Fuzzy Logic, Machine Learning and Deep Learning with Julia](#)
- [1. Introduction](#)
- [2. Fuzzy Logic](#)
- [3. Machine Learning](#)
- [4. Deep Learning](#)
- [5. Docker](#)
- [6. Exams](#)
- [7. Slides-Labs](#)
- [8. .gitignore](#)
- [9. LICENSE](#)
- [10. README.md](#)

The repository also has a table of recent commits:

Commit Hash	Message	Time
fde8fca	update Docker README file	2 weeks ago
	Initial commit	4 months ago
	change colors	yesterday
	change colors	yesterday
	rm Docker cheat sheet	3 days ago
	vgg and resnet transfer learning	yesterday
	Update docker-image.yml	2 weeks ago

The repository also has a table of recent issues:

Issue Number	Title	Status
1	Update docker-image.yml	Open

The repository also has a table of recent pull requests:

Pull Request Number	Title	Status
1	Update docker-image.yml	Open

The repository also has a table of recent actions:

Action Name	Status
Build and deploy	Success

The repository also has a table of recent projects:

Project Name	Status
Project 1	Open

The repository also has a table of recent wiki pages:

Wiki Page Name	Status
Wiki Page 1	Open

The repository also has a table of recent security issues:

Security Issue Name	Status
Security Issue 1	Open

The repository also has a table of recent insights:

Insight Name	Status
Insight 1	Open

The repository also has a table of recent settings:

Setting Name	Status
Setting 1	Open

The repository also has a table of recent about information:

Topic	Count
flux	1
machine-learning	1
docker-image	1
fuzzy-logic	1
julialang	1
mjl	1

The repository also has a table of recent languages:

Language	Percentage
Julia	94.3%
Dockerfile	3.4%
Batchfile	2.1%
TeX	0.2%

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



# CONTINUOUS INTEGRATION (CI)

The screenshot shows a web browser window displaying the Docker Hub repository for 'abmhamdi/jlai'. The browser's address bar shows 'hub.docker.com/r/abmhamdi/jlai'. The Docker Hub interface includes a search bar, navigation links for 'Explore', 'Repositories', 'Organizations', and 'Help', and a user profile for 'abmhamdi'. The repository page features a blue cube icon, the name 'abmhamdi/jlai' with a star, and a 'Manage Repository' button. It indicates the repository was updated 21 hours ago by 'abmhamdi' and is associated with 'Artificial Intelligence Labs @ ISETBZ'. The 'Overview' tab is selected, showing a description: 'Fuzzy Logic, Machine Learning and Deep Learning with Julia'. It states that the repository contains slides, labs, and code examples for using Julia to implement artificial intelligence algorithms, running on a Docker image. A status bar shows 'jlai-ci' as 'passing', with 'version latest', 'docker pulls 22', and 'docker stars 0'. A 'Docker Pull Command' box displays the command 'docker pull abmhamdi/jlai'. At the bottom, it instructs users to run the command to pull the Docker image.

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

## CNN, VAE, GAN & NLP

---

# CNN

## MOTIVATING FACTORS

- ▶ 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,.
  - ▶ **CNNs** are composed of several types of layers, including convolutional layers, pooling layers, and fully connected layers.
- ❶ The **convolutional layers** apply filters to the input data, which are used to detect patterns and features in the data.
  - ❷ 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.
  - ❸ The **fully connected layers** combine the features learned by the convolutional and pooling layers to make a prediction.

# CNN

## APPLICATIONS

**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.

**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.

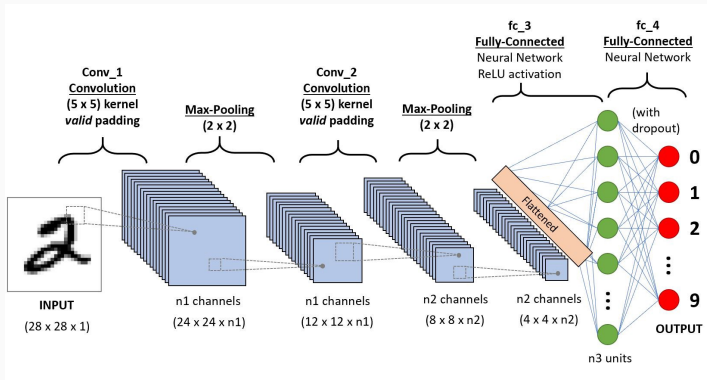
**Robotics** CNNs have been used to process images from robots to enable them to navigate their environment and perform tasks such as grasping objects.

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

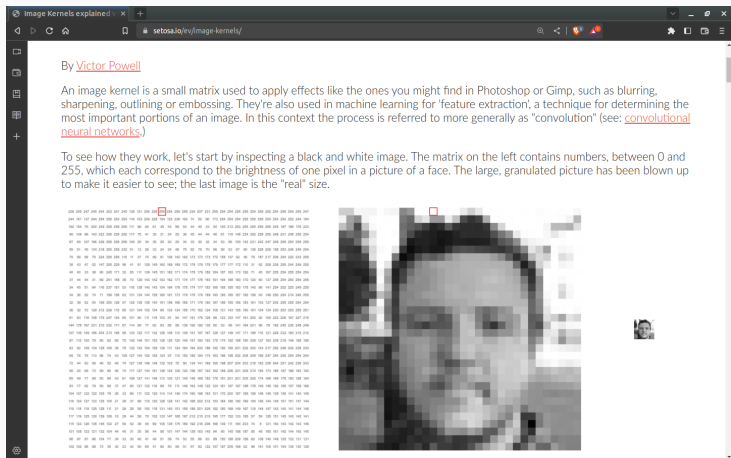


# CNN

## ARCHITECTURE



► Source



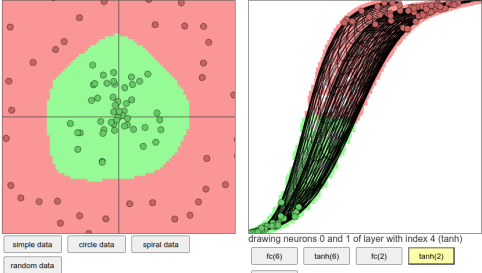
<https://setosa.io/ev/image-kernels/>

## CNN

## CONVNETJS DEMO

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.



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/)

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/jlai/>

→ *Codes* → *Julia* → *Part-3* → *cnn.jl*



# VAE

## MOTIVATING FACTORS

- ▶ 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.

# VAE

## APPLICATIONS

**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.

**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.

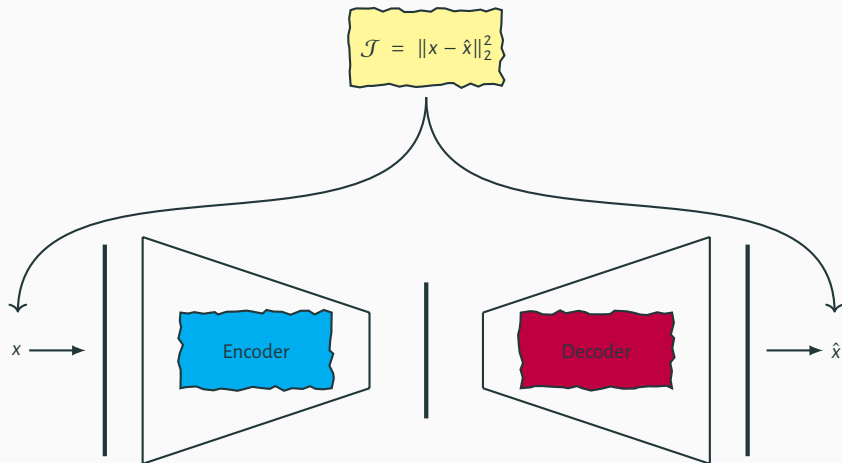
**Representation learning** VAEs can be used to learn meaningful latent representations of data, which can be useful for tasks such as clustering and classification.

**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.

# VAE

## LOSS OF VANILLA AUTOENCODER

MINIMIZE SQUARED ERROR LOSS



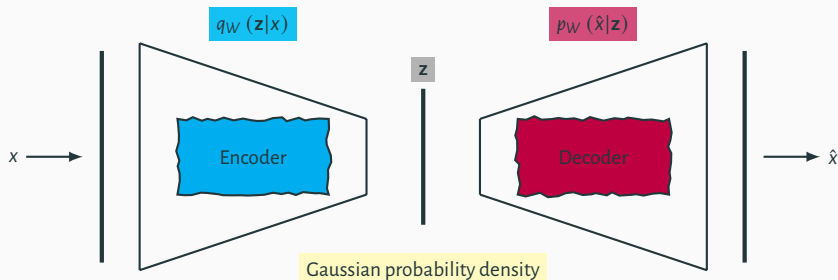
## VAE

## LOSS OF VARIATIONAL AUTOENCODER

$$\mathcal{J} = \underbrace{-\mathbb{E}_{\mathbf{z} \sim q_W(\mathbf{z}|\mathbf{x}^{(i)})} \left[ \log p_W(\mathbf{x}^{(i)}|\mathbf{z}) \right]}_{\text{Expected negative log likelihood term wrt to encoder distribution}} + \underbrace{\mathcal{KL}(q_W(\mathbf{z}|\mathbf{x}^{(i)}) \parallel p(\mathbf{z}))}_{\text{Kullback-Leiber divergence term where } p(\mathbf{z}) \sim \mathcal{N}(\mu=0, \sigma^2=1)}$$

Expected negative log likelihood term wrt to encoder distribution

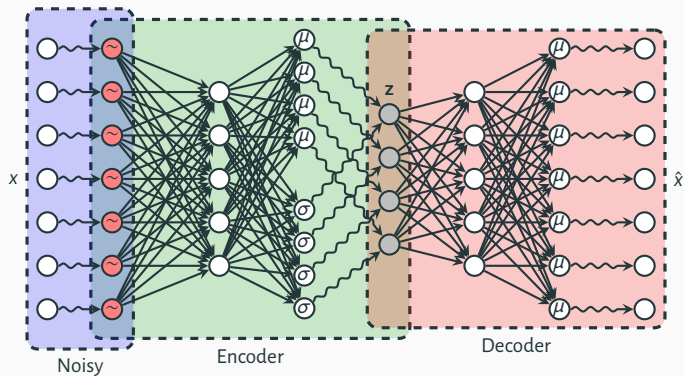
Kullback-Leiber divergence term where  $p(\mathbf{z}) \sim \mathcal{N}(\mu=0, \sigma^2=1)$





# VAE

## ARCHITECTURE OF VARIATIONAL AUTOENCODER



## VAE

CODE SNIPPET



Code is available at <https://github.com/a-mhamdi/jlai/>

→ *Codes* → *Julia* → *Part-3* → *vae.jl*



## VAE

 $\mathcal{KL}$  LOSS DERIVATION

In a VAE, the latent vector  $\mathbf{z}$  is calculated by:

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \quad \text{where} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}_z, \mathbb{1}_{z \times z})$$

$\boldsymbol{\mu}$  and  $\boldsymbol{\sigma}$  denote respectively the mean and variances for the latent vector  $\mathbf{z}$ . The encoder learns to output the two vectors  $\boldsymbol{\mu} \in \mathbb{R}^z$ , and  $\boldsymbol{\sigma} \in \mathbb{R}^z$ . The encoder distribution is

$$q(\mathbf{z}|x) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}(x), \boldsymbol{\Sigma}(x)) \quad \text{where} \quad \boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & 0 & 0 & \cdots \\ 0 & \sigma_2^2 & 0 & \cdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_l^2 \end{bmatrix}$$

The latent prior is given by

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{0}_z, \mathbb{1}_{z \times z})$$

$$\mathcal{KL}(q_W(\mathbf{z}|x) \| p(\mathbf{z})) = \frac{1}{2} \left[ - \sum_i (\log \sigma_i^2 + 1) + \sum_i \sigma_i^2 + \sum_i \mu_i^2 \right]$$

► Source

# GAN

## AN OVERVIEW

- ▶ 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.

# GAN

## APPLICATIONS

**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.

**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.

## GAN

CODE SNIPPET



Code is available at <https://github.com/a-mhamdi/jlai/>

→ *Codes* → *Julia* → *Part-3* → *gan.jl*



# NLP

## PURPOSE OF NLP

- ▶ **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.
- ▶ **NLP** 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.

# NLP

## APPLICATIONS

**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

## GENERAL PROCESS 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.
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.

# NLP

CODE SNIPPET



Code is available at <https://github.com/a-mhamdi/jlai/>

→ *Codes* → *Julia* → *Part-3* → *nlp.jl*



## Transfer Learning

---

# TRANSFER LEARNING

## DRIVING FORCES

- ▶ **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

## APPLICATIONS

**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.

**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.

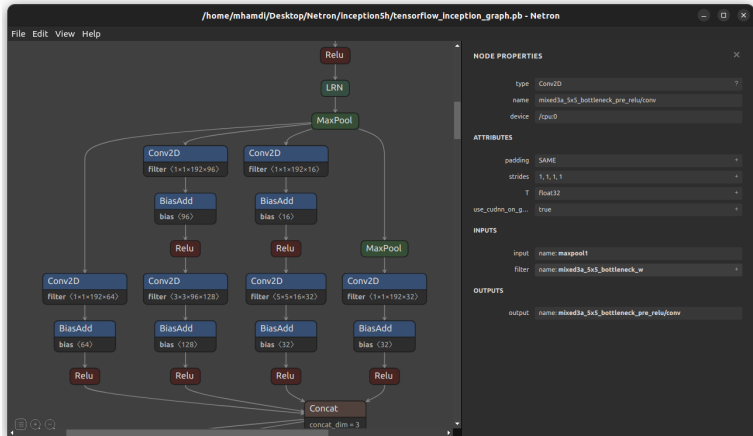
**Speech recognition** Transfer Learning has been used to fine-tune pre-trained speech recognition models on new languages or accents.

**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.

# TRANSFER LEARNING

## NETRON



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

# TRANSFER LEARNING

## GENERAL PROCESS IN JULIA

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

→ Codes → Julia → Part-3 → transfer-learning-\*.jl





## Reinforcement Learning

---

# REINFORCEMENT LEARNING

## SYNOPSIS

- ▶ **Reinforcement Learning** is a type of machine learning in which an agent learns to interact with its environment in order to maximize a reward. It involves learning to map situations (called states) to actions that will maximize a reward. The agent receives feedback in the form of rewards and penalties for its actions, which it uses to adjust its behavior accordingly.
- ▶ In **reinforcement Learning**, the goal is to learn a policy that maximizes the cumulative reward over time. The agent learns this policy through **trial and error**, by exploring different actions in different states and receiving feedback in the form of rewards or penalties.
- ▶ **Reinforcement Learning** is used in a variety of applications, including control systems, game playing, and natural language processing. It has been successful in a number of tasks, including teaching a computer to play chess and Go at a high level.

# REINFORCEMENT LEARNING

CODE SNIPPET



Code is available at <https://github.com/a-mhamdi/jlai/>

→ Codes → Julia → Part-3 → reinforcement-learning.jl



## Responsible AI

---



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



## FURTHER READING (1/2)

### References

---

- [Alz+21] L. Alzubaidi et al. “Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions”. In: *Journal of Big Data* 8.1 (2021). DOI: 10.1186/s40537-021-00444-8.
- [Azu21] P. Azunre. *Transfer Learning for Natural Language Processing*. Manning Publications Co. LLC, 2021, p. 272.
- [Goo+17] I. Goodfellow et al. *Deep Learning*. MIT Press, 2017.
- [HZM16] X. Hao, G. Zhang, and S. Ma. “Deep Learning”. In: *International Journal of Semantic Computing* 10.03 (2016), pp. 417–439. DOI: 10.1142/s1793351x16500045.
- [KW13] D. P. Kingma and M. Welling. “Auto-Encoding Variational Bayes”. In: (Dec. 2013). arXiv: 1312.6114 [stat.ML].
- [LBH15] Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539.
- [LD19] B. Lauwens and A. B. Downey. *Think Julia : How to Think Like a Computer Scientist. How to Think Like a Computer Scientist*. O'Reilly Media, 2019, p. 298.

## FURTHER READING (2/2)

- [PWP20] D. Phil Winder Ph. *Reinforcement Learning Industrial Applications of Intelligent Agents. Industrial Applications of Intelligent Agents*. O'Reilly Media, Incorporated, 2020.
- [Sar21] I. H. Sarker. "Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions". In: *SN Computer Science* 2.6 (2021). DOI: 10.1007/s42979-021-00815-1.
- [SB] R. S. Sutton and A. G. Barto. *Reinforcement Learning An Introduction. An Introduction*. A Bradford Book, p. 552.
- [SC21] F. F. L. da Silva and A. H. R. A. H. R. Costa. *Transfer Learning for Multiagent Reinforcement Learning Systems*. Springer International Publishing AG, 2021.
- [Ser21] L. Serrano. *Grokking Machine Learning*. Manning Publications Co. LLC, 2021, p. 498.
- [SKP] M. Sewak, M. R. Karim, and P. Pujari. *Practical Convolutional Neural Networks: Implement advanced deep learning models using Python*. Packt Publishing - ebooks Account, p. 218.
- [SR] J. Silge and D. Robinson. *Text Mining with R: A Tidy Approach*. O'Reilly Media, p. 194.