

# Foundations of Statistical and Machine Learning for Actuaries -

## Graphic Data Neural Networks

Edward (Jed) Frees, University of Wisconsin - Madison  
Andrés Villegas Ramirez, University of New South Wales

July 2025

# Schedule

Day and Time	Presenter	Topics	Notebooks for Participant Activity
Monday Morning	Jed	Welcome and Foundations Hello to Google Colab	Auto Liability Claims
Monday Afternoon	Jed	Classical Regression Modeling	Medical Expenditures (MEPS)
	Andrés	Regularization, Resampling, Cross-Validation	Seattle House Sales Data
	Andrés	Classification	Victoria Road Crash Data
Tuesday Morning	Andrés	Trees, Boosting, Bagging	
Tuesday Afternoon	Jed	Big Data, Dimension Reduction and Non-Supervised Learning	Big Data, Dimension Reduction, and Non-Supervised Learning
	Jed	Neural Networks	Seattle House Prices
	Jed	Graphic Data Neural Networks	Claim Counts
Tuesday 4 pm	Fei	Fei Huang Thoughts on Ethics	MNIST Digits Data
Wednesday Morning	Jed	Recurrent Neural Networks, Text Data	Insurer Stock Returns
Wednesday After Lunch	Jed	Artificial Intelligence, Natural Language Processing, and ChatGPT	
	Dani	Dani Bauer Insights	
Wednesday Afternoon	Andrés	Applications and Wrap-Up	

## Tuesday Afternoon 4B. Graphic Data Neural Networks

This module covers:

- applications and history of graphic data neural nets
- convolution architecture, including convolution and pooling layers
- model fitting
- generalization and pretraining

## Graphic Data Neural Nets

### Computer Vision Use Cases

- Every day, we interact with deep vision models — via Google Photos, Google image search, YouTube, video filters in camera apps, OCR software, and many more.
- These models are also at the heart of cutting-edge research in autonomous driving, robotics, AI-assisted medical diagnosis, autonomous retail checkout systems, and even autonomous farming.

A few key insurance industry applications:

- **1. Claims Processing Automation Vehicle Damage Assessment.** After a car accident, policyholders can upload photos of the damaged vehicle. **Property Damage Analysis.** Homeowners upload photos of damaged roofs, interiors, or external structures (from storms, fire, etc.).

## Computer Vision Use Cases 2

- **2. Risk Assessment & Underwriting Aerial & Satellite Imagery.** Use drone or satellite images to evaluate property condition, roof age, pool presence, proximity to vegetation (fire risk), etc. **Building Inspection.** Inspect commercial buildings for compliance, safety risks, or wear and tear.
- **3. Fraud Detection. Photo Tampering Detection.** Ensure claim-related photos have not been digitally manipulated. **Duplicate Claim Detection.** Identify when the same damage image is used in multiple claims.
- **4. Document and Image Analysis.**  
**License Plate & VIN Recognition.** Extract identifiers from vehicle photos for claim validation or theft recovery. **Medical Image Analysis (Health Insurance)**
- **5. Emerging Applications Wildfire Risk Scoring from Visual Inputs.** Combine drone imagery and vegetation classification to model real-time wildfire risks.

## Convolution Neural Networks - A success story

- Convolutional Neural Networks (CNNs) were a **turning point** in the development of AI, especially in the 2010s.
- CNNs are a type of deep learning model especially good at processing **grid-like data**, such as images. They automatically learn to detect features (edges, shapes, patterns) by applying filters (convolutions) across input data.
- A 1998 paper by Yann LeCun et al
  - became widely used by banks to recognize handwritten digits on checks.
  - It introduced two new building blocks: convolutional layers and pooling layers.

## CNNs - A success story 2

- The real explosion came in **2012**, when **AlexNet**, a deep CNN developed by Alex Krizhevsky, won the ImageNet competition (a massive image classification challenge) by a **huge margin** — reducing error from 26% to 15%.
- Why it mattered:
  - **Showed CNNs outperform traditional AI methods** (like hand-crafted features + SVMs).
  - Used **GPUs** to train faster and deeper models.
  - Proved deep learning could scale to **real-world tasks** with large datasets. After AlexNet:
- **Industry adoption exploded** — especially in computer vision (e.g., self-driving cars, facial recognition, medical imaging).
- **CNNs didn't invent AI — but they unlocked its real power**, proving deep learning could solve complex, real-world problems when combined with big data and powerful compute.

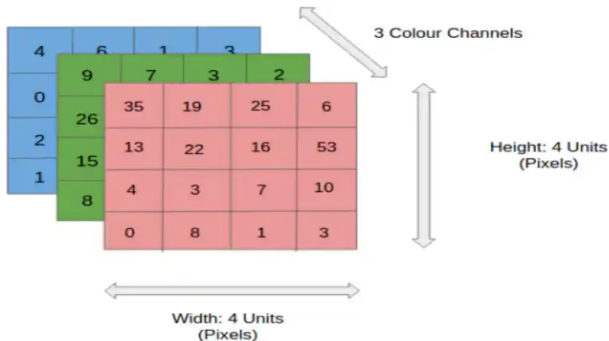
## Graphic Data

- Images are stored in a computer as a matrix of numbers known as **pixel** values.
  - These pixel values represent the intensity of each pixel.
  - In grayscale images, a pixel value of 0 represents black, and 255 represents white.
- A **channel** is a matrix of pixel values, and we have only one channel in the case of a grayscale image.
  - The dimension of this image will be  $24 \times 16$ , we mean it would be 24 pixels across the height and 16 pixels across the width.
- Almost all colors can be generated from the three primary colors – Red, Green, and Blue.
  - Therefore, we can say that each colored image is a unique composition of these three colors or 3 channels – Red, Green, and Blue.



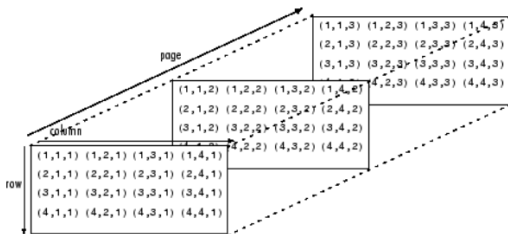
## Graphic Data 2

- The numbers for each image are organized in a three-dimensional array called a **feature map**.



{ Source: <https://www.analyticsvidhya.com/blog/> }

- Naturally, it will be straightforward to extend these ideas to higher dimensional data.
- For example, you might want to have images move over time, treating data as a 4-dimensional array.



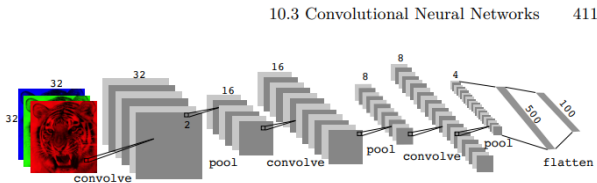
{ Source: <https://https://wiki.pathmind.com/convolutional-network> }

## Convolutional Architecture

- The network first identifies low-level features in the input image, such as small edges, patches of color, and the like.
  - These low-level features are then combined to form higher-level features, such as parts of ears, eyes, and so on.
  - Eventually, the presence or absence of these higher-level features contributes to the probability of any given output class.
- How does a convolutional neural network build up this hierarchy? It combines two specialized types of hidden layers, called **convolution layers** and **pooling layers**.
  - Convolution layers search for instances of small patterns in the image, whereas
  - pooling layers downsample these to select a prominent subset.

## Convolutional Architecture 2

- In order to achieve state-of-the-art results, contemporary neural network architectures make use of many convolution and pooling



**FIGURE 10.8.** Architecture of a deep CNN for the CIFAR100 classification task. Convolution layers are interspersed with  $2 \times 2$  max-pool layers, which reduce the size by a factor of 2 in both dimensions.

{ Credit: From Chapter 10 of - James et al. (2023), An Introduction to Statistical Learning with Applications in Python }

## Convolutions

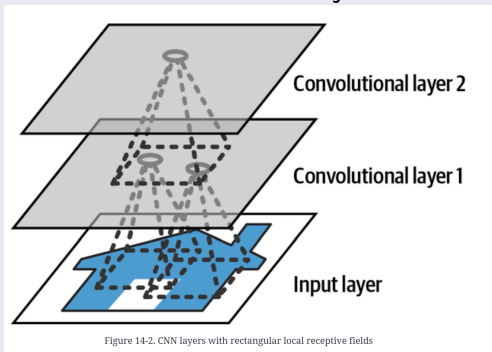
- Consider a feature  $x(t)$  at position  $t$  that is a bit noisy.
  - For simplicity, for now assume that  $x$  is one dimensional.
  - We can smooth the feature with neighboring values of  $x(\cdot)$  using

$$s(t) = \sum_{a=-\infty}^{a=\infty} x(a)w(t-a)$$

- where the weight function  $w$  is called a **kernel**. This operation is known as a **convolution**.

## Convolutions 2

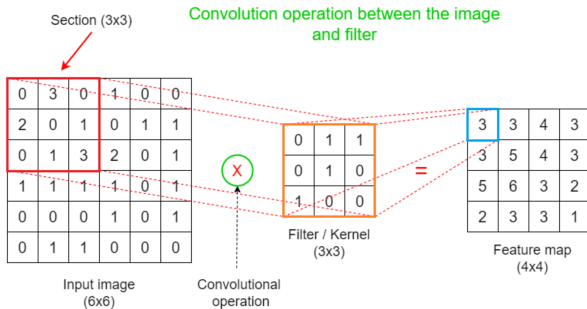
- Particularly with graphic data, the smoothing need not extend so far. We can limit the index  $a$  to just a few instances.



{ Credit: This is an example from Chapter 14 of [Géron, A. \(2022\). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow \(3rd ed.\). O'Reilly Media.](#) }

In the same way, if we have a two-dimensional image data  $x$  as our input, we use a two-dimensional kernel  $w$ :






$$s(t_1, t_2) = \sum_{a=-\infty}^{a=\infty} \sum_{b=-\infty}^{b=\infty} x(a, b) w(t_1 - a, t_2 - b).$$



## Interpretations:

- If a  $3 \times 3$  submatrix of the original image resembles the convolution filter, then it will have a large value in the convolved image; otherwise, it will have a small value.
- Thus, the convolved image *highlights regions of the original image* that resemble the convolution filter.

Impact of  
Kernels

Operation	Kernel $\omega$	Image result $g(x,y)$
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Ridge or edge detection	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

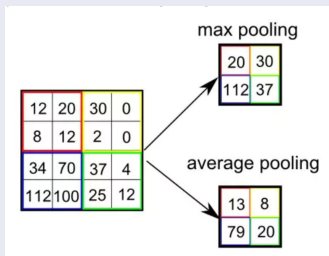


## Sparse Interactions and Parameter Sharing

- The basic feedforward neural network displays fully connected (dense) layers. That is, there exists a separate parameter describing the interaction between each input unit and each output unit. This means that every output unit interacts with every input unit.
  - Convolutional networks, however, typically have **sparse interactions** (also referred to as sparse connectivity or sparse weights).
- In addition, one uses the **same** kernel for all input units at each layer. **Parameter sharing** refers to using the same parameter for more than one function in a model.
- This sparsity and parameter sharing greatly reduce the number of parameters to be estimated, compared to the basic neural net.

## Pooling

- *Pooling* represents another way of decreasing the size of a layer.
- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.
- Their goal is to subsample (i.e., shrink) the input image in order to reduce the computational load, the memory usage, and the number of parameters (thereby limiting the risk of overfitting).



## Model Fitting

### Review of Classification Metrics

- The label is a categorical variable  $M$  with different classes, say,  $0, 1, \dots, M - 1$ 
  - One uses *one-hot encoding* (dummy variables) to convert the label to a binary vector.
  - So *label* = 3 corresponds to  $y = [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]$
- The activation function is generally the **softmax** function
  - Let  $a_1, \dots, a_K$  be the fitted values of the nodes at the last stage.
  - The output values are defined as

$$\hat{y}_i = \frac{\exp(a_i)}{\sum_{j=1}^K \exp(a_j)}$$

- So, a large value of  $a_i$  means a large value of  $\hat{y}_i$ .
- This gives a binary vector of predicted probabilities.

## Review of Classification Metrics 2

- The predicted value is simply the class with the largest predicted probability (`np.argmax()` in python).  
Mathematically

$$\hat{M} = \arg \max_j \hat{y}_j$$

- The loss function **categorical cross-entropy** is simply the negative log multinomial likelihood function.

$$-\sum_j y_j \log(\hat{p}_j)$$

- The **accuracy** is simply the proportion of correct predictions

$$\text{accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions}}$$

## Fitting the Model - Sample Python Code

```
img_rows, img_cols = 28, 28
x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)

model = Sequential()      # Define a network as a linear stack of layers
model.add(Conv2D(32,      # Add 1st pooling layer with kernel shape: 3 x 3
                kernel_size=(3, 3), activation='relu',
                input_shape=(img_rows, img_cols, 1)) )
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32,      # Add 2st convolutional layer with:
                kernel_size=(3, 3), activation='relu',
                input_shape=(img_rows/2, img_cols/2, 1)))
model.add(Conv2D(16,
                kernel_size=(3, 3), activation='relu',
                input_shape=(img_rows/2, img_cols/2, 1)))
model.add(Flatten())
model.add(Dense(24, activation='relu'))
                                # Add output layer with softmax activation
                                # with 10 output classes
model.add(Dense(num_classes, activation='softmax'))

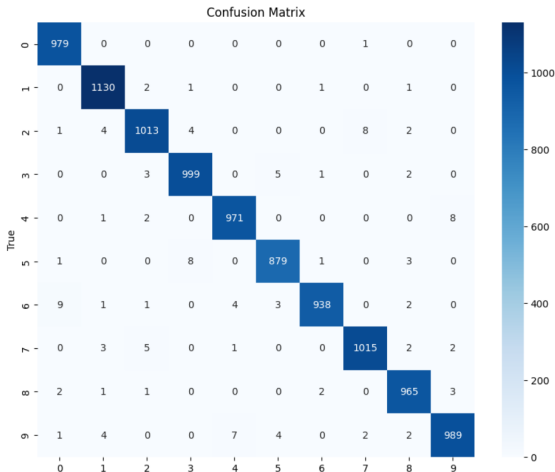
model.compile(loss='categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])
model.summary()

history = model.fit(x_train, y_train, epochs=20,
                   validation_data=(x_test, y_test))
```

This model has accuracy:

- 99.73 % for the training sample
- 98.78 % for the test sample.

Summarize the results with a confusion matrix.



## Some Reminders about Generalization

- A neural network is a function that describes the relationship between features and outputs.
- The traditional strategy to address it is to split the sample into two groups: training and testing.
  - The training sample is used to train the model and learn the mathematical function relating features to targets.
  - The testing sample is used only to test the model and obtain a measure of generalization.
- When the training and testing accuracy are really close, we say the model predictions generalize well.
  - Otherwise, we say the model is **overfitting** the data. This is, the learned equation “fits” the data so well, that now is unable to make good predictions for out-of-sample exemplars.
  - A model is **underfit** when it is so simple that has poor accuracy in the training set, and mediocre to poor in the testing set.

## Generalization 2

- Overfitting is usually approached with a mix of techniques in neural networks: data augmentation, dropout, weight decay, early stopping, and others.
  - Overfitting is often interpreted as a sign of excessive model complexity,
- No amount of regularization, cross-validation, etc. will fix poorly sampled data

Figure 13: overfitting, underfitting, and good-fitting





## Pretrained Models

- CNN models have fewer parameters than comparable feedforward neural net models due to sparsity and parameter sharing
- Nonetheless, there are many layers to CNN models. The number of parameters can become large quickly.
- For example, even in our simple MNIST digits example, there are 45,570 parameters to be estimated.

```
[ ] model.summary()
```

➡ Model: "sequential\_6"

Layer (type)	Output Shape	Param #
conv2d_10 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_11 (Conv2D)	(None, 11, 11, 32)	9,248
conv2d_12 (Conv2D)	(None, 9, 9, 16)	4,624
flatten_5 (Flatten)	(None, 1296)	0
dense_15 (Dense)	(None, 24)	31,128
dense_16 (Dense)	(None, 10)	250

Total params: 45,570 (178.01 KB)

Trainable params: 45,570 (178.01 KB)

Non-trainable params: 0 (0.00 KB)

## Representation Learning and Pretrained Models

- The idea underpinning *representational learning* is that the model learns features, not just predictions
  - In our CNN graphic data case, we believe that the lower layers represent low-level features in the input image such as small edges, patches of color, and the like.
  - Later, we will take a look at textual data. There, for example, we will argue that *positions of words* preserve semantic meaning; e.g. synonyms should appear near each other.

## Pretrained Models and Transfer Learning

- For standard problems such as computer vision / graphic data and text analysis, extensive studies exist that model these low level features.
  - For example, in CNN, there are large models, e.g. ResNet, VGG, MobileNet, that are already trained on a large dataset like ImageNet.
  - One can use the fitted parameters from such a study in one's own work.
  - They are called **pretrained**.
  - Intuitively, these models can be used to fit low level features, freeing up your data to fit higher level features.
- Re-using or fine-tuning representations from one task to another is an example of **transfer learning**.

# Session 4B. Graphic Data Neural Networks

## Summary

This module covered:

- applications and history of graphic data neural nets
- convolution architecture, including convolution and pooling layers
- model fitting
- generalization and pretraining

During lab, participants may follow the notebook [MNIST Digits](#) .