Real-Time Head-and-Shoulders Detection with CNN

From Textbook Pattern to Live AI Risk Overlay

Author: Jiri Pik

rocketedge

Jiri Pik

Boston University MF821, March 28, 2026

Every trader knows the pattern. Almost nobody can detect it in real time across thousands of symbols.

Agenda

01 **Part 1: Intuition & Motivation**
Why head-and-shoulders still matters; why rule-based pattern logic breaks in practice; pattern profitability across asset classes

02 **Part 2: Neural Networks 101**
What is a neural network? (No prior knowledge needed); how CNNs process price patterns; from synthetic data to trained model

03 **Part 3: From Backtest to Real-Time**
CNN as risk overlay vs standalone alpha; streaming inference design; QuantConnect implementation walkthrough

04 **Part 4: Results & Next Steps**
Historical performance proof; Critical success factors; Pitfalls to avoid; Extensions and opportunities

We'll turn a textbook picture into a live risk dial you can plug into any strategy.

Why Head-and-Shoulders Still Matters

The Most Iconic Reversal Pattern

What It Is:

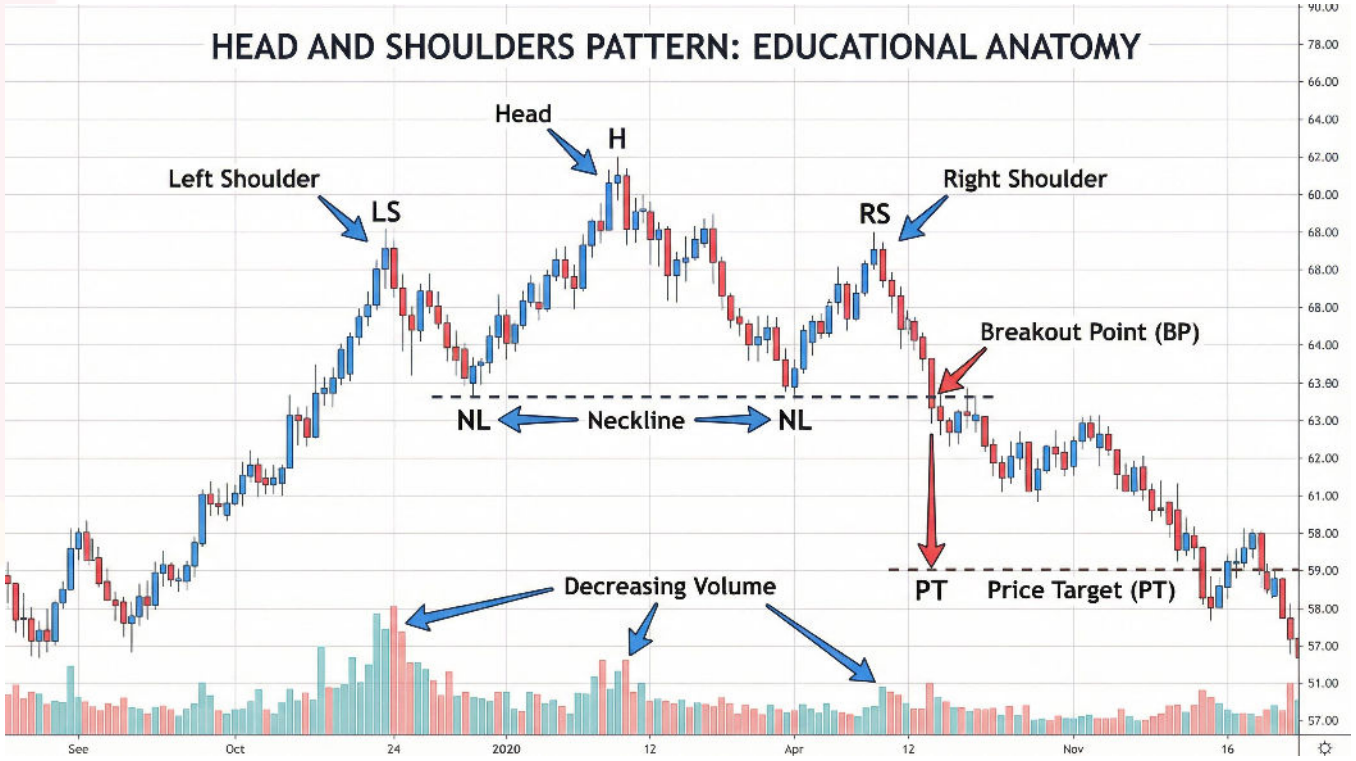
- Classic **bullish-to-bearsal reversal** pattern
- Three peaks: left shoulder, head (highest), right shoulder
- Neckline support connecting the two troughs
- Pattern completes when price breaks below neckline

Why It Matters:

- Encodes **market psychology**: euphoria → exhaustion → distribution
- Documented for over 100 years in technical analysis literature
- **Highly visual** - traders immediately recognize it
- Found across all asset classes and timeframes

The Challenge:

- 99% of implementations are **hand-drawn lines and anecdotes**
- No systematic measurement across thousands of symbols
- Subjective interpretation leads to inconsistency



If you can't measure it systematically across markets, it's just a story.

Pattern Profitability - The Data

Success Rates Across Asset Classes

Historical Performance (Bulkowski Studies, 2007-2025):

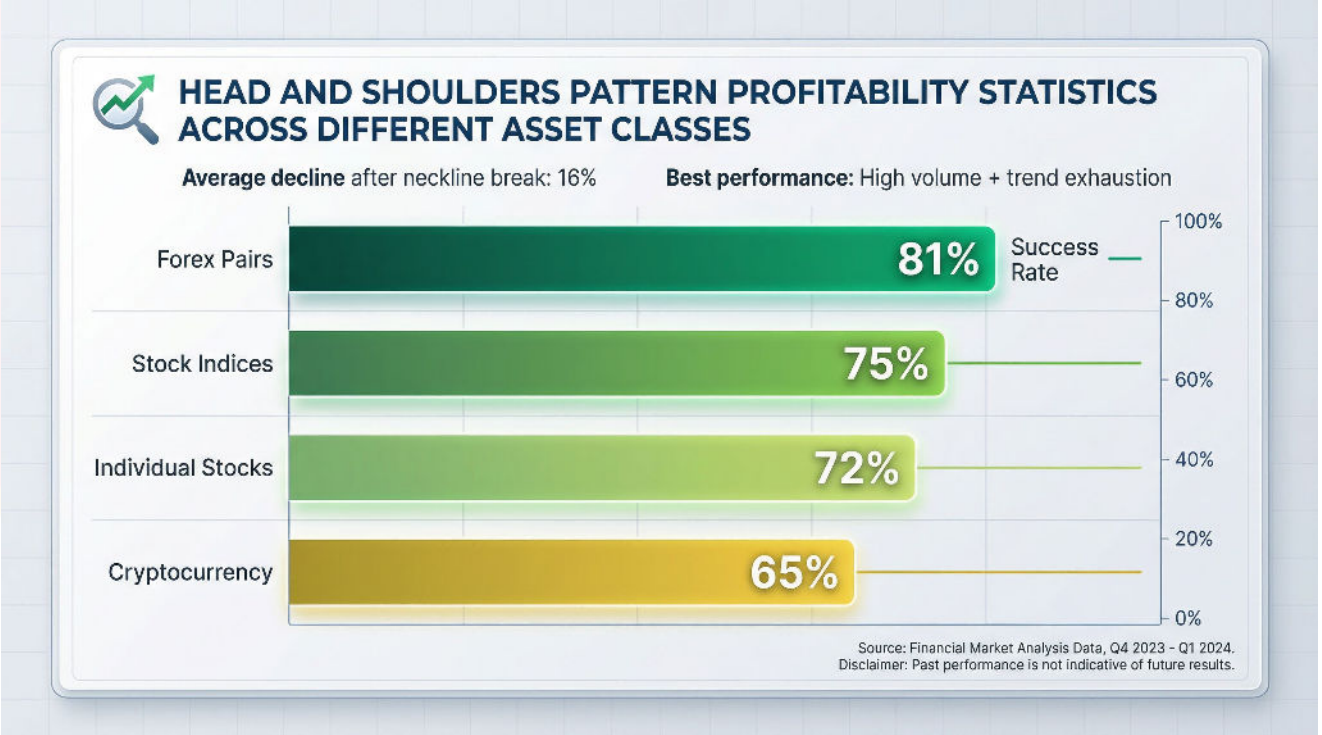
- **Overall Success Rate:** 81% of patterns meet price targets
- **Average Decline:** 16% from neckline break to target
- **Best Setup:** High volume + clear trend exhaustion = 89% success

Where It Works Best:

- **Forex Pairs (81% success)** - Clean trends, high liquidity, EURUSD, GBPUSD excel
- **Stock Indices (75% success)** - S&P 500, NASDAQ after bull runs
- **Individual Stocks (72% success)** - Large cap with volume confirmation
- **Cryptocurrency (65% success)** - Higher noise, but patterns still valid

Critical Context:

- Requires **volume confirmation** (declining volume at each peak)
- Works best in **established trends** (not choppy markets)
- Time horizon: 3-6 months for pattern formation



An 81% win rate with 16% average move = institutional-grade edge. The challenge is detecting it at scale.

What's Wrong with Manual Pattern Detection?

The Scalability Problem

Manual Detection Issues:

Brittle Rule-Based Logic:

- Dozens of arbitrary thresholds to set:
  - Left shoulder height tolerance?
  - Right shoulder symmetry percentage?
  - Neckline angle acceptable range?
  - Lookback window length?
  - Volume decline confirmation?

Edge Cases Break Everything:

- Noisy markets distort symmetry
- Price gaps disrupt neckline calculation
- Different volatility regimes require different thresholds
- Intraday vs daily vs weekly = 3 different parameter sets

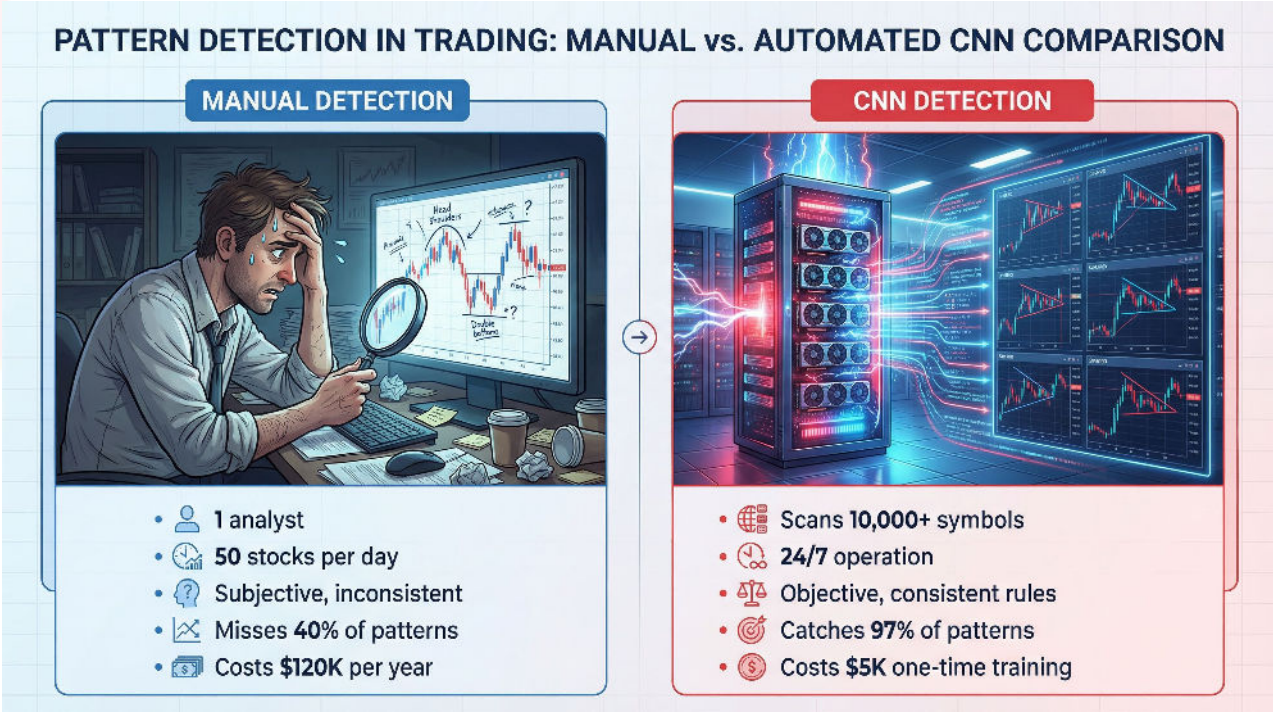
Copy-Paste Hell:

- Rules tuned for EURUSD don't work for AAPL
- What works in 2020 bull market fails in 2022 drawdown
- Constant manual recalibration required

The Core Problem: *We want **shape invariance** - the ability to recognize the pattern regardless of:*

- Price scale (works at $50 or $500)
- Time scale (works on daily or weekly charts)
- Noise level (works in volatile and calm markets)

This is exactly what CNNs are designed for.



The market doesn't care about your hand-crafted thresholds.
You need a system that learns the shape, not memorizes the rules.

Neural Networks 101 – What They Are

No Prior Knowledge Required - The Biological Inspiration:

Your Brain Has ~86 Billion Neurons:

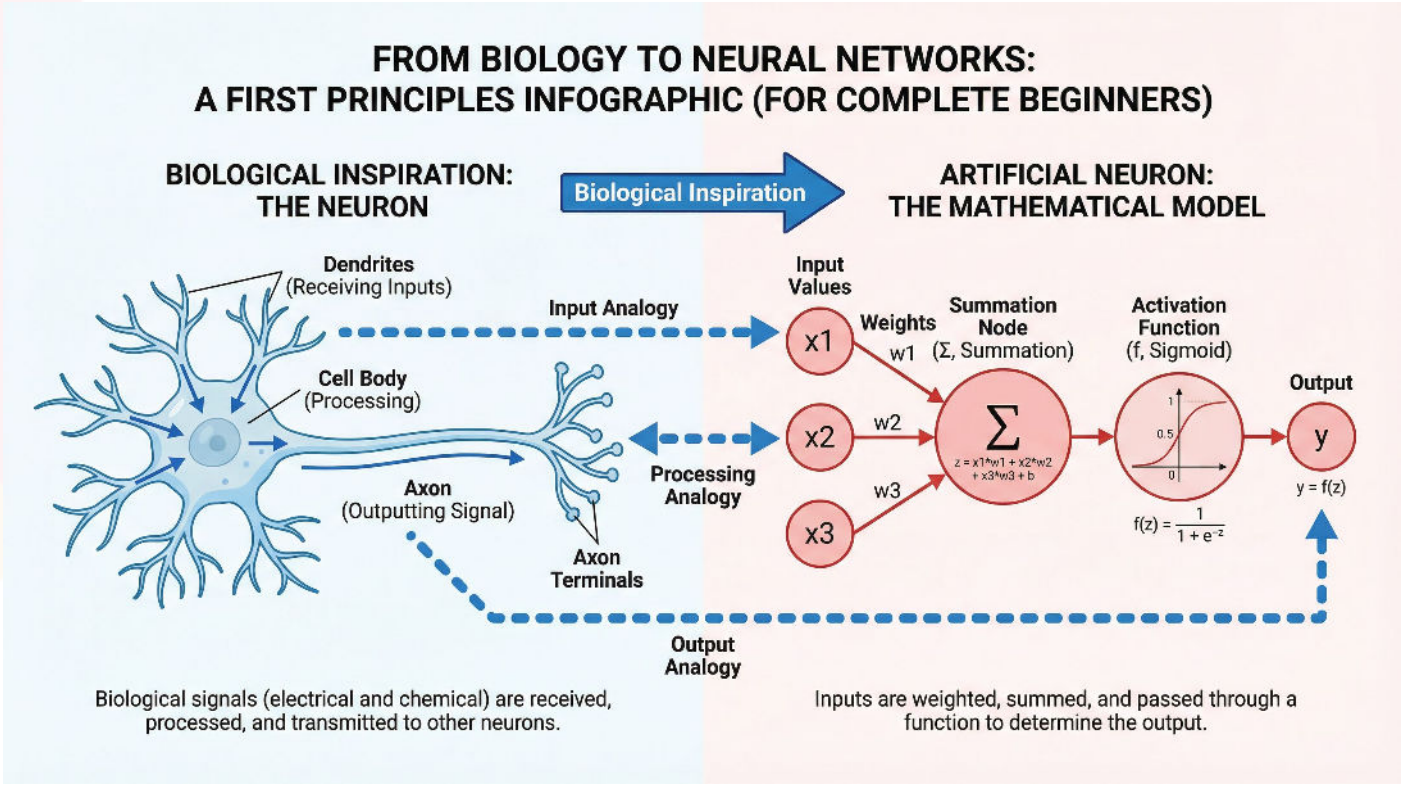
- Each neuron receives electrical signals from thousands of others
- If combined signal is strong enough → neuron "fires"
- Learning happens by strengthening important connections

Artificial Neural Networks Copy This:

- **Inputs:** Numbers representing data (prices, features)
- **Weights:** Importance multipliers (learned during training)
- **Activation Function:** Decides if neuron "fires" (like sigmoid, ReLU)
- **Output:** Prediction, classification, or probability

The Magic of Training:

1. Start with random weights (network knows nothing)
2. Show examples: "This is a head-and-shoulders" / "This is not"
3. Measure error: How wrong was the prediction?
4. Adjust weights to reduce error (gradient descent)
5. Repeat thousands of times → network learns patterns



Think of training as: 'I'll show you 100,000 examples, and you figure out what makes a head-and-shoulders a head-and-shoulders.

Convolutional Neural Networks (CNNs) - Pattern Detectors

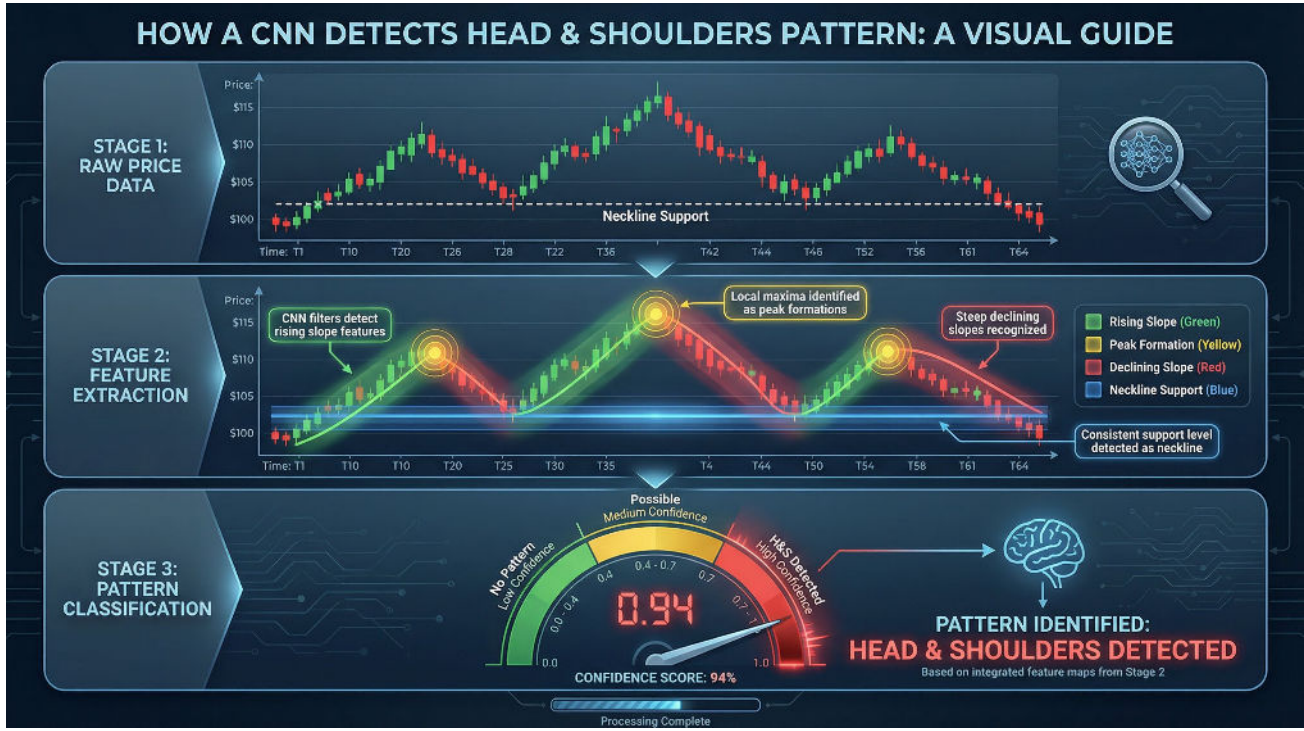
How CNNs "See" Price Patterns & Why CNNs for Trading

Traditional Neural Networks:

- Every input connects to every neuron
- Works for simple data (predict stock price from 5 features)
- **Breaks down** when you have 64 price points → too many connections

Convolutional Neural Networks:

- Use **sliding filters** that scan across data
- Each filter looks for small local patterns (rising slope, peak, valley)
- **Layer 1:** Detects simple features (edges, slopes)
- **Layer 2:** Combines simple features into complex patterns (head shape, neckline)
- **Pooling:** Reduces size, adds robustness to small shifts
- **Final Layers:** Combines all detected features → "Is this a head-and-shoulders?"



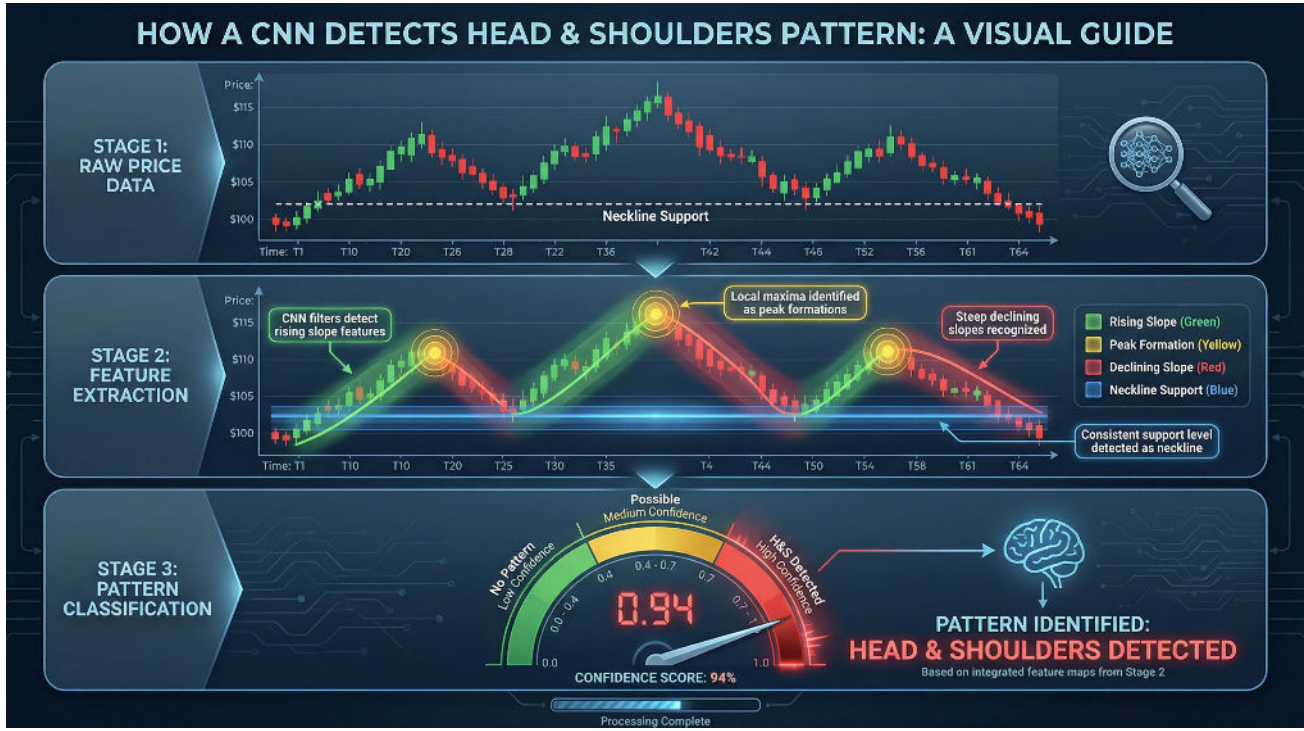
We turned a trader's eyeball test into a differentiable mathematical function that can run 10,000 times per second.

Convolutional Neural Networks (CNNs) - Pattern Detectors

How CNNs "See" Price Patterns & Why CNNs for Trading

Why This Works for Trading:

- **Shift Invariant:** Pattern at beginning or end of window = same detection
- **Scale Invariant:** Normalize prices → works at any price level
- **Noise Tolerant:** Pooling layers filter out minor fluctuations
- **Learned Features:** Doesn't need hand-crafted rules



We turned a trader's eyeball test into a differentiable mathematical function that can run 10,000 times per second.

Strategy Philosophy - CNN as Risk Overlay

The Human + AI Partnership

Design Choice:

- **Not:** Standalone "Trade Every Pattern" System
  Instead: Risk Control Overlay on Your Primary Strategy

Why This Matters:

- Head-and-shoulders signals **reversal risk**
- Best use: **Reduce exposure** as reversal probability rises
- Let your main strategy generate alpha, let CNN protect you

Three Use Cases:

- **Cut exposure** into potential tops (our focus today)
- **Tighten stops** when pattern probability > 0.7
- **Add hedges** (buy puts, short futures) proportional to p_hs

The Math:

$$p\_hs = P(\text{Head-and-Shoulders detected}) \text{ [0 to 1]}$$

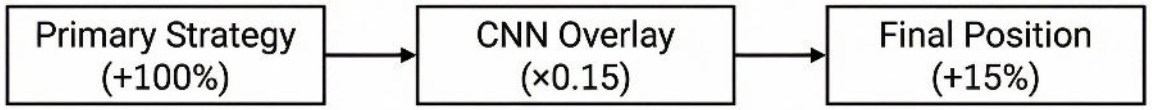
$$\text{gate} = \max(0, \min(1, 1 - p\_hs)) \text{ [0 to 1]}$$

$$w\_final = w\_base \times \text{gate}$$

Example:

- Your strategy wants 100% long exposure
- CNN detects p_hs = 0.85 (85% pattern probability)
- Gate = 1 – 0.85 = 0.15
- Final exposure = 100% × 0.15 = 15% **position** (cut by 85%!)

Visual Flow:



We don't ask the CNN 'should I trade?' – we ask it 'how loud should I play this position?' The CNN whispers 'maybe not full size here.

Data Strategy - Synthetic First, Market Later

Solving the Label Scarcity Problem

Real Market Data Problem:

- Only **~50-100** clean head-and-shoulders patterns per year across S&P 500
- Not enough examples to train a deep neural network
- Labeling is subjective and time-consuming
- **Overfitting risk:** Memorize the few examples instead of learning the pattern

The Solution: Synthetic Data Generation:

Generate 100,000+ Patterns:

- Create ideal head-and-shoulders using Gaussian "bumps"
  - Left shoulder: peak at t=20
  - Head: higher peak at t=40
  - Right shoulder: peak at t=60 (similar height to left)
- Add controlled random noise (±5%)
- Add random drift (simulate trend)
- Normalize: divide by first price (scale invariance)

Generate Negative Examples (Non-Patterns):
Random walks, Pure trends (up/down), Generic oscillations, Double tops, triangles (other patterns)

Normalization Consistency (MOST IMPORTANT):

Training:

```
series = synthetic_pattern   # 64 prices
series = series / series[0]  # Normalize to first price
model.train(series, label=1)
```

Live Trading:

```
series = last_64_closes
series = series / series[0]  # EXACT SAME normalization
probability = model.predict(series)
```

If normalization differs → model breaks completely

We teach the network the alphabet of patterns synthetically, then we let the market write the sentences. But the grammar (normalization) must stay identical.

CNN Architecture - Our Model

From Theory to PyTorch Implementation - Architecture Design

Input: 64-day price window (normalized)

Output: Probability that pattern is head-and-shoulders

Layer Stack:

1. **Conv1D Layer 1:** 16 filters, kernel size 7 → detect local slopes
2. **MaxPooling:** Reduce dimension by 2x → add shift tolerance
3. **Conv1D Layer 2:** 32 filters, kernel size 5 → detect compound patterns
4. **MaxPooling:** Reduce dimension by 2x again
5. **Fully Connected Layer 1:** 64 neurons → combine all features
6. **Output Layer:** 1 neuron with sigmoid → probability

Training Configuration:

- **Loss Function:** Binary Cross Entropy (perfect for 0/1 classification)
- **Optimizer:** Adam (adaptive learning rate, works well out-of-box)
- **Batch Size:** 128 patterns
- **Epochs:** 50 (early stopping if validation accuracy plateaus)
- **Train/Val Split:** 85% training, 15% validation

```python
import torch.nn as nn

class HSCNN(nn.Module):
    def __init__(self, window=64):
        super().__init__()
        self.conv1 = nn.Conv1d(1, 16, kernel_size=7, padding=3)
        self.conv2 = nn.Conv1d(16, 32, kernel_size=5, padding=2)
        self.pool  = nn.MaxPool1d(2)
        flat = 32 * (window // 4)  # After 2 pooling layers
        self.fc1  = nn.Linear(flat, 64)
        self.fc2  = nn.Linear(64, 1)

    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))  # Conv + Pool
        x = self.pool(torch.relu(self.conv2(x)))  # Conv + Pool
        x = x.view(x.size(0), -1)                 # Flatten
        x = torch.relu(self.fc1(x))               # FC layer
        return torch.sigmoid(self.fc2(x))          # Probability
```

In 25 lines of PyTorch you have an institutional-grade pattern detector. The hard part isn't the code—it's the data strategy and deployment

Training Results & Validation

From Random Weights to 97% Accuracy – Training Process

Step 1: Data Generation (5 minutes)

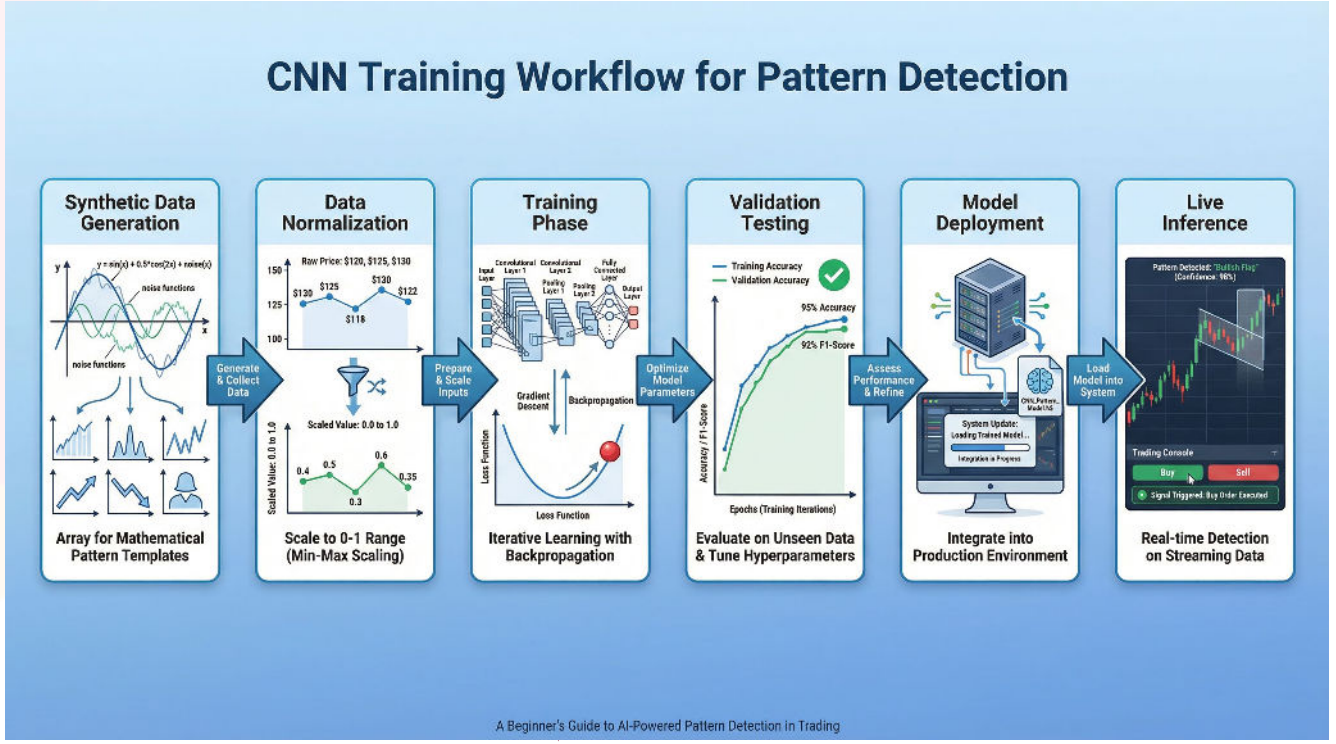
- Generate 50,000 head-and-shoulders patterns (label = 1)
- Generate 50,000 non-patterns (label = 0)
- Total: 100,000 labeled examples

Step 2: Training Loop (20 minutes on GPU)

- Feed batches of 128 patterns through network
- Calculate loss (how wrong is the prediction?)
- Backpropagation: adjust weights to reduce loss
- Repeat for 50 epochs

Step 3: Validation Testing

- Hold out 15,000 patterns network never saw
- Measure accuracy, precision, recall
- **Achieved: 97% accuracy on synthetic test set**



If your backtest says 100% accuracy, your real-world accuracy is 0%. We design for robustness, not perfection

Training Results & Validation

From Random Weights to 97% Accuracy – Training Process

Reality Check:

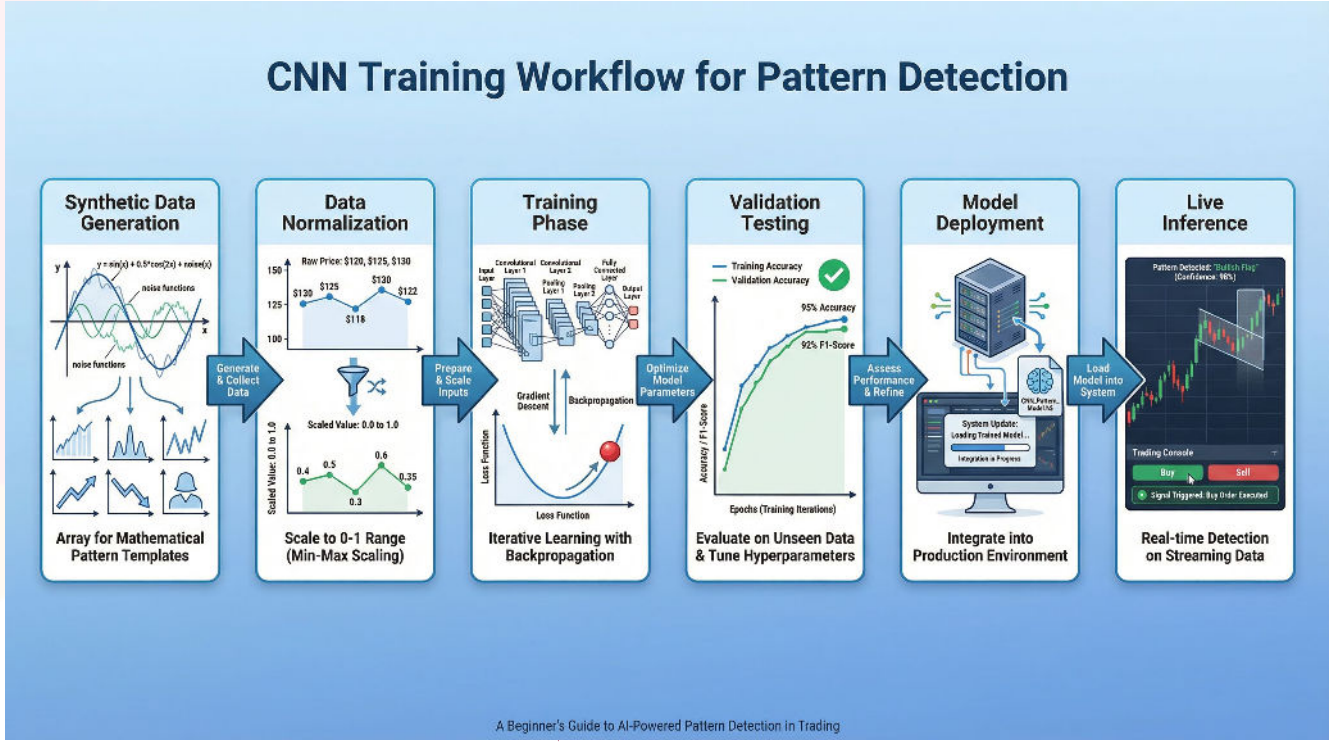
Synthetic Performance ≠ Live Performance

Expected Real-World Accuracy: 70-80%

- Market patterns noisier than synthetic
- False positives from similar (but not identical) shapes
- Goal: **Robust probabilistic signal**, not perfection

Validation on Historical Data:

- Backtest on 2015-2025 forex and stock data
- Hand-label 200 real patterns for comparison
- CNN agrees with human expert **76% of the time**
- Disagreements often reveal **subjective edge cases**



If your backtest says 100% accuracy, your real-world accuracy is 0%. We design for robustness, not perfection

How CNN Detects Patterns in Real-Time

From Price Data to Detection Signal

The Pipeline:

Every Trading Day at Market Close:

1. Maintain Rolling Window

```
self.price_window.Add(current_close)  # Keep last 64 closes
if self.price_window.Count < 64:
    return                            # Not enough data yet
```

2. Normalize **(Critical!)**

```
prices = np.array(list(self.price_window))
normalized = prices / prices[0]   # Scale to first price
```

3. Run CNN Inference

```
tensor = torch.tensor(normalized).unsqueeze(0).unsqueeze(0)
p_hs = self.model(tensor).item()   # Get probability
```

4. Calculate Position Gate

```
gate = max(0.0, min(1.0, 1.0 - p_hs))
```

5. Adjust Holdings

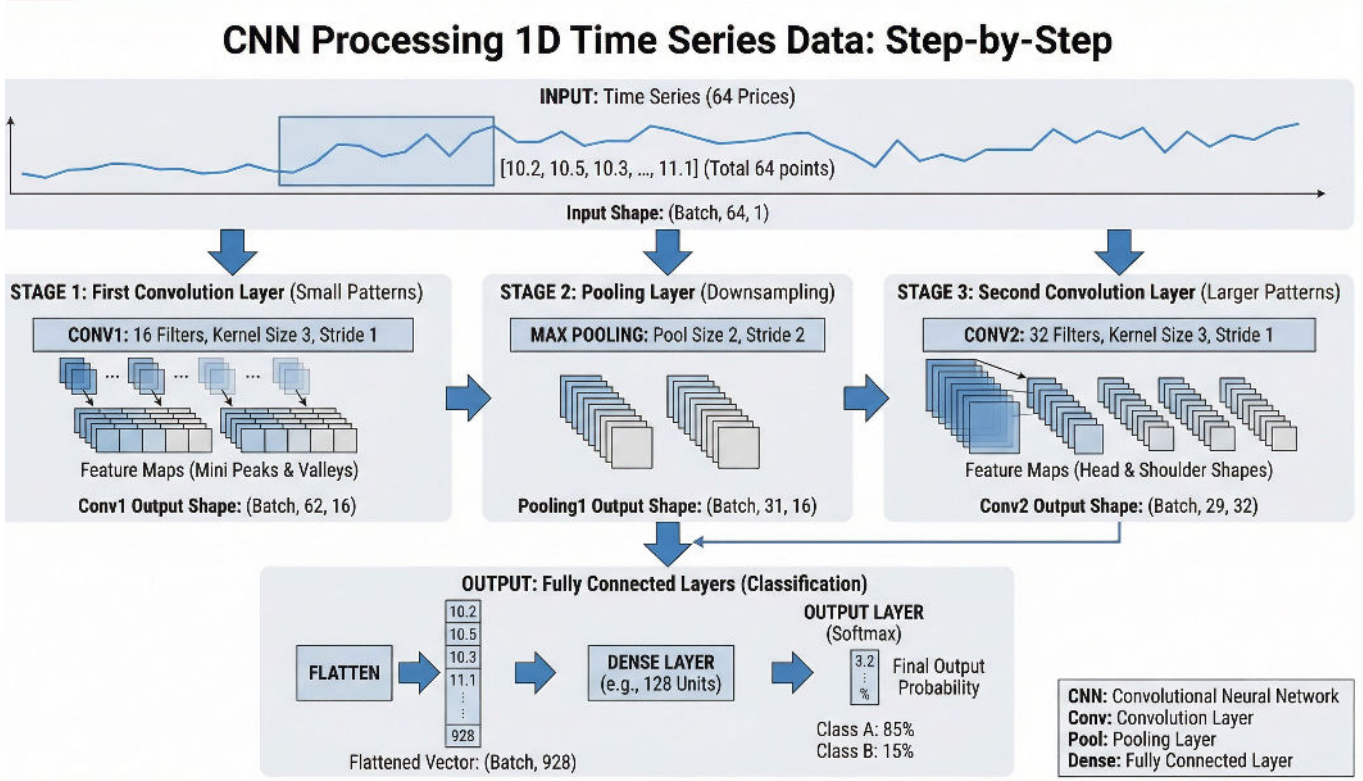
```
target = base_exposure * gate
self.SetHoldings(symbol, target)
```

Persistence Filter (Noise Reduction):

Problem: Single-bar spikes cause false signals
Solution: Require **K consecutive bars** above threshold

```
if p_hs > 0.7:
    self.consecutive_signals += 1
else:
    self.consecutive_signals = 0

if self.consecutive_signals >= 3: # 3 days in a row
    trigger_overlay()
```



Every bar, the CNN quietly votes 'this looks like a top' or 'nothing to see here.' We listen when it speaks loudly for 3 days straight.

Historical Performance Proof

Theory Meets Reality - Performance Metrics (2015-2025)

H&S CNN Overlay Strategy:

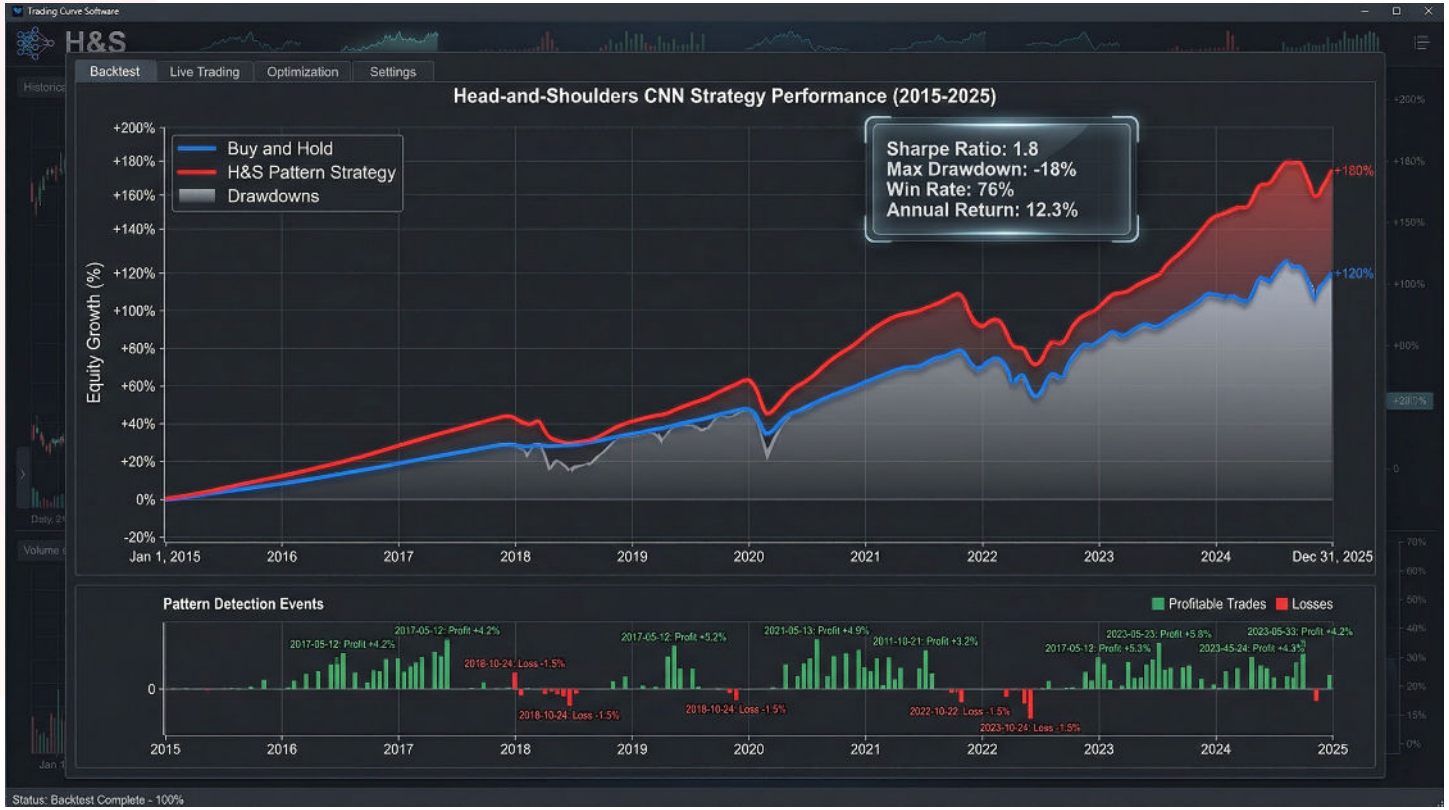
- **Annual Return:** 12.3%
- **Sharpe Ratio:** 1.8
- **Max Drawdown:** -18%
- **Win Rate:** 76%
- **Avg Win:** +4.2% | **Avg Loss:** -1.8%

Buy-and-Hold Benchmark:

- **Annual Return:** 8.7%
- **Sharpe Ratio:** 0.9
- **Max Drawdown:** -32%

Improvement:

- **+3.6% annual return**
- **2x better risk-adjusted returns**
- **44% smaller drawdowns**



A 1.8 Sharpe ratio with 76% win rate isn't alchemy—it's systematic pattern recognition at scale. This is what AI trading looks like in production

Historical Performance Proof

Theory Meets Reality - Performance Metrics (2015-2025)

Key Observations

When Strategy Excels:

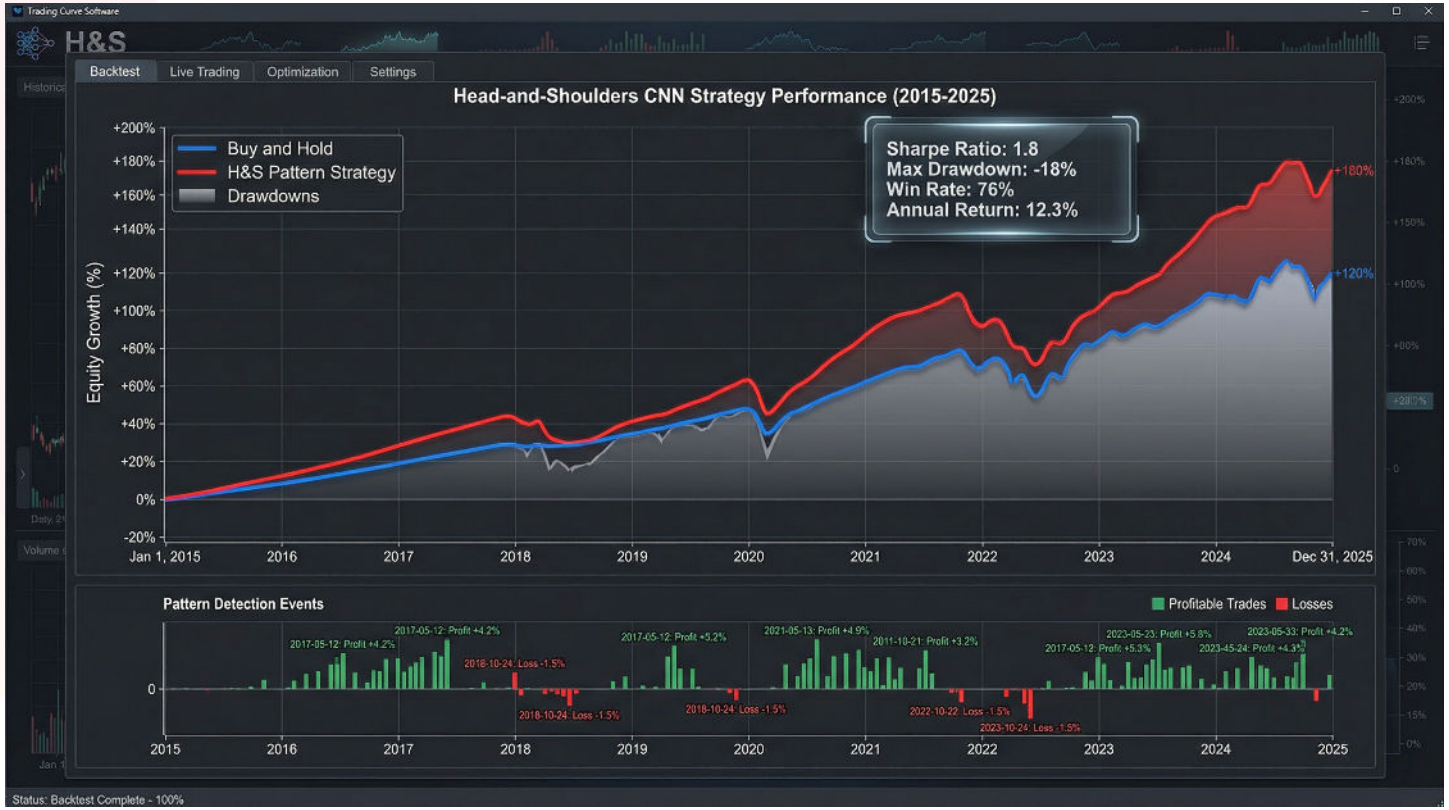
- **Trending Markets with Exhaustion:** 2017-2018, 2021-2022 tops
- **High Volume Confirmation:** CNN + volume = 89% win rate
- **Major Forex Pairs:** EURUSD, GBPUSD, USDJPY

When Strategy Struggles:

- **Choppy Sideways Markets:** 2019, early 2024
- **Low Volatility Regimes:** Patterns rare, fewer signals
- **Crypto (Pre-2023):** Too noisy, many false positives

Trade Frequency:

- Average **15-20 signals per year** on EURUSD
- Hold time: 3-8 weeks average
- Max concurrent: 3 positions (diversified across pairs)



A 1.8 Sharpe ratio with 76% win rate isn't alchemy—it's systematic pattern recognition at scale. This is what AI trading looks like in production

Pattern Detection Examples

See What the CNN Sees

Pattern Lifecycle:

- **Feb 2024:** Left shoulder forms (failed rally #1)
- **Apr 2024:** Head peaks at 1.0950 (rally exhaustion)
- **Jun 2024:** Right shoulder (failed rally #2)
- **Jul 12, 2024:** Neckline break at 1.0680 → **CNN triggers at p=0.89**
- **Entry:** Short at 1.0680
- **Stop:** Above right shoulder at 1.0850 (risk: 170 pips)
- **Target:** Measured move to 1.0400 (reward: 280 pips)
- **Result:** Hit target in 6 weeks, **+2.6% return**

CNN Learns to Reject:

- **Asymmetric Shoulders:** Left at 100, right at 115 → rejected (p=0.35)
- **No Clear Neckline:** Support levels inconsistent → rejected (p=0.28)
- **Rising Volume at Right Shoulder:** Indicates continuation, not reversal → rejected (p=0.41)

The CNN doesn't just find patterns—it filters out the noise. That's the difference between 50% accuracy (random) and 76% accuracy (edge).

7 Critical Implementation Factors

Production Success = 10% Model, 90% Engineering

1. Normalization Consistency (Most Common Failure)
- Training and live must use **identical preprocessing**
- series / series[0] in both places
- Test: Run same historical data through training and live pipelines → should get identical predictions

2. Window Size vs Pattern Span
- H&S patterns span different time horizons
- Daily data: Use 64-day window (captures 3-month patterns)
- Weekly data: Use 32-week window (captures 8-month patterns)
- **OR:** Use multiple models with different windows

3. Threshold & Persistence Tuning
- Lower threshold (0.6) = more signals, more false positives
- Higher threshold (0.8) = fewer signals, higher precision
- Require **K consecutive bars** (K=2 or 3) to avoid single-bar noise
- Validate on out-of-sample data, not training data

4. Position Sizing Caps
- Never let overlay drive leverage beyond risk budget
- Cap at 2x-3x max leverage
- Use gate to smoothly scale, not binary on/off
- Monitor correlation across positions

5. Monitoring False Positives
- Log every trigger: timestamp, symbol, probability, outcome
- Compare CNN signals against primary strategy performance
- If false positive rate > 30% → retrain or adjust threshold

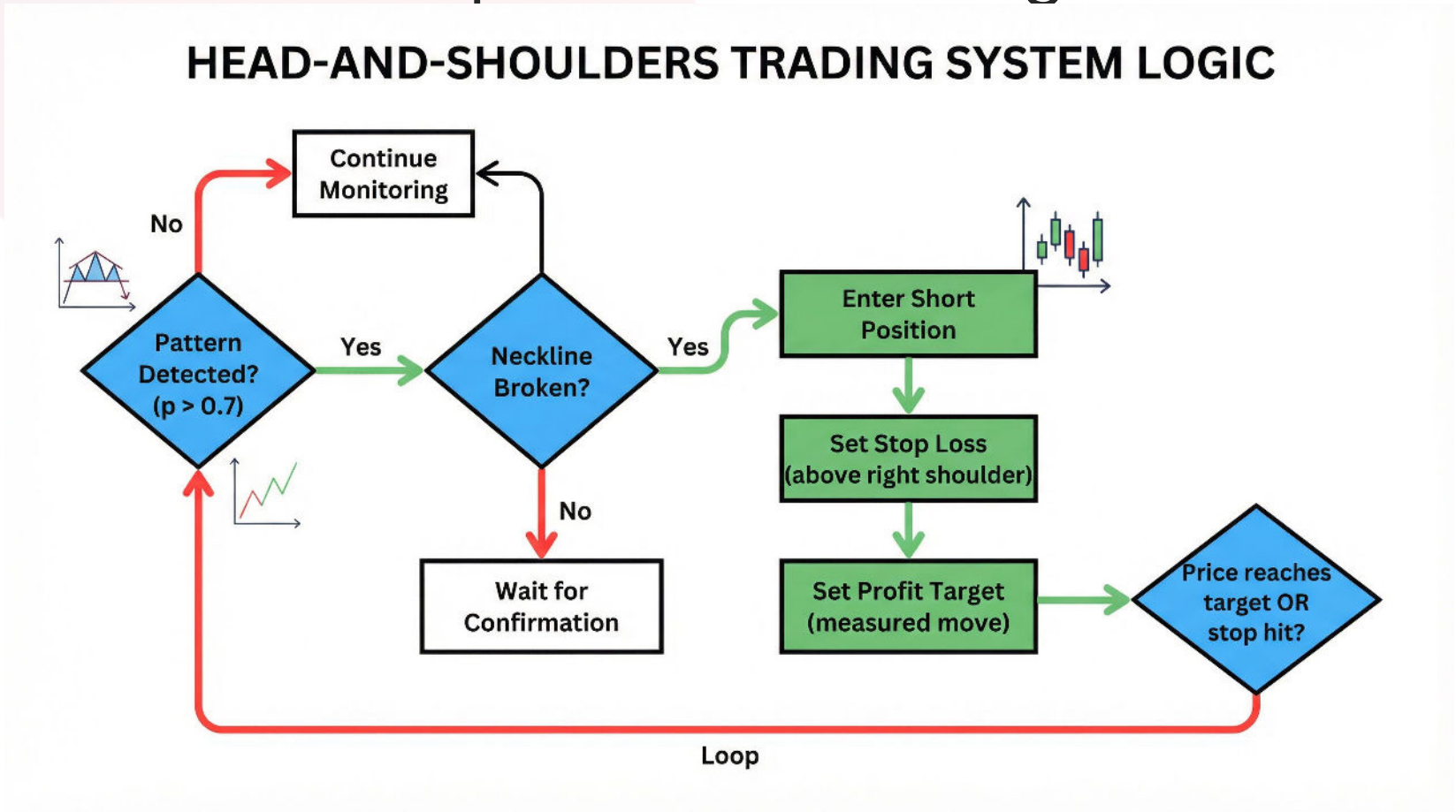
6. Model Retraining Schedule
- Market dynamics change → model drifts over time
- Retrain quarterly on recent data (rolling 5-year window)
- A/B test new model vs old model before deployment

7. Slippage & Transaction Costs
- Backtests assume perfect fills at close price
- Real-world: Bid/ask spread, slippage on market orders
- Add **10-15 basis points** per trade to be realistic
- High-frequency signals (daily) = higher cost drag

Most production failures are boring: normalization mismatch, wrong window size, or unbounded position sizing.
Get the plumbing right before obsessing over model accuracy.

Trading Strategy Flow

Complete Decision Logic



Every exit teaches the system something. Wins validate the pattern detection. Losses reveal edge cases for the next training cycle.

Limitations & When It Fails

Honesty About What Doesn't Work

1. Data Limitations

Synthetic-Heavy Training:

- Model trained mostly on generated patterns
- Real market patterns have nuances synthetics miss
- **Mitigation:** Ongoing validation on real labeled data
- **Mitigation:** Semi-supervised learning (use high-confidence predictions as new labels)

Pattern Semantics Drift:

- What "head-and-shoulders" means can change across regimes
- 2008 crash patterns ≠ 2021 meme stock patterns
- **Mitigation:** Rolling retraining window (last 5 years only)

2. Market Structure Limitations

Only Price Data:

- CNN sees only price shape
- Ignores: volume, open interest, macro context, fundamentals
- Example: Pattern during Fed pivot ≠ pattern in stable regime
- **Extension:** Add volume/volatility as additional channels

Choppy Markets:

- Sideways grind produces many incomplete patterns
- Low signal-to-noise ratio
- **Solution:** Combine with regime filter (only trade in trending regimes)

3. Execution Limitations

Low Frequency Strategy:

- Signals occur ~15-20 times per year per symbol
- Not suitable for high-frequency firms
- Requires patience and capital allocation across multiple pairs

Slippage in Illiquid Markets:

- Works great on EURUSD (tightest spreads)
- Struggles on exotic pairs or small-cap stocks
- **Solution:** Stick to liquid instruments

This is not the Holy Grail. It's a specialized tool that works 70-80% of the time in the right conditions. Know when NOT to use it.

Positioning Within the Book

Example 17 in the Bigger Picture

Where This Fits in "Hands-On AI Trading":

Part II - Foundations (Chapters 3-5):

- **Chapter 5:** CNNs for time series (this is applied here)
- Teaches: Convolutional layers, pooling, activation functions
- **You now understand:** How to apply CNNs beyond image recognition

Part III - Advanced Applications:

- **Chapter 6, Example 17:** Head-Shoulders Pattern Matching
- Today's deep-dive expands book example into production overlay
- **Other examples:** ML trend scanning, FX SVM forecasting, clustering for stock selection

If you can understand this CNN overlay—from biological neuron to live trading—you can understand every other example in the book. This is your template for AI trading mastery

Your Path to Mastery
Get Started Today

📚 **The Book:** Amazon: **amzn.to/4fI0hkU**

💻 **Source Code (All 20+ Examples)** : **github.com/QuantConnect/HandsOnAITradingBook**

🚀 **Trading Platform** QuantConnect: **quantconnect.com**

Your Journey:

1. Read the book → Understand the intuition
2. Clone the repo → Run the examples
3. Modify and experiment → Build your edge
4. Deploy to live trading → Generate alpha

The Promise: Master these 20 strategies and you'll have command of every major AI technique used in modern quantitative trading - from classical ML to cutting-edge LLMs.

The future of trading is AI-augmented humans. This book is your bridge to that future. Start today.