Applications of Neural Networks for Anomalous Energy Consumption Detection

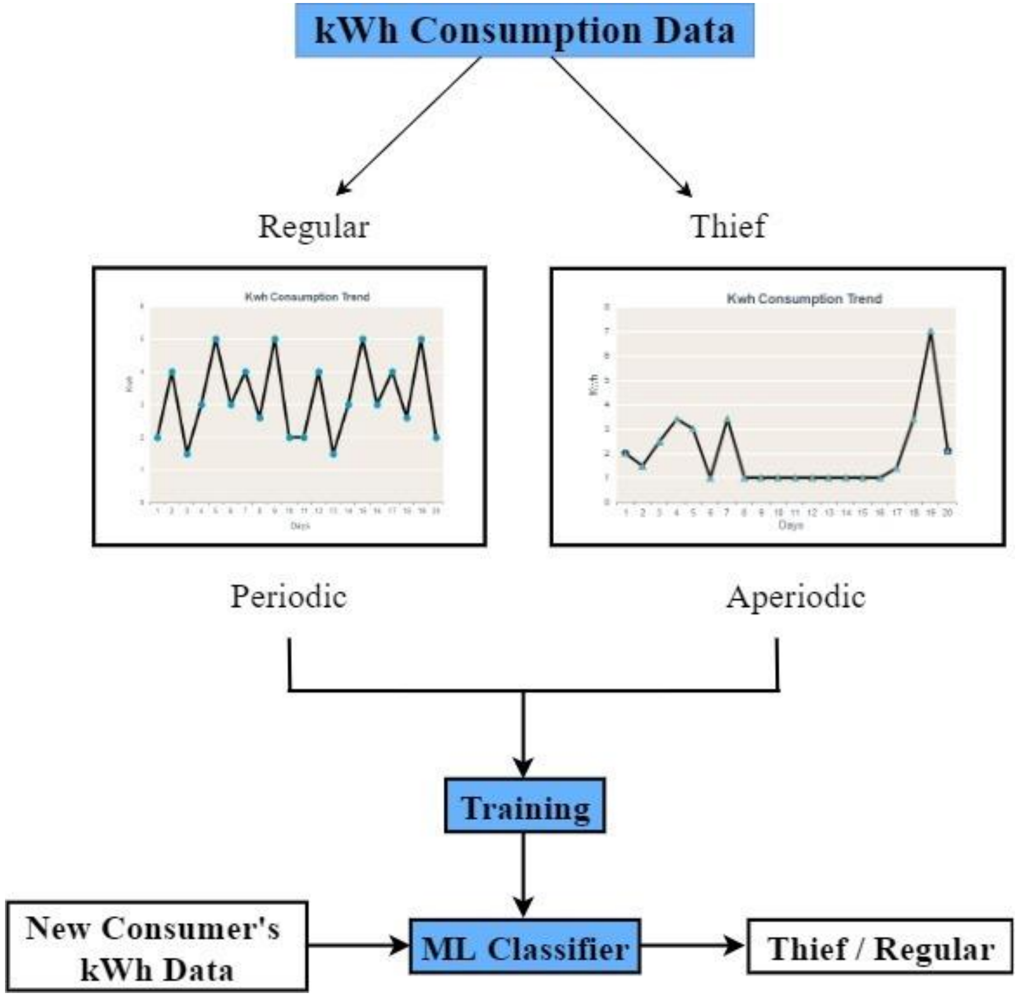
GROUP 13: EE-163, EE-164, EE-177, EE-194

INTERNAL ADVISOR: DR. M. M. ALI BAIG – DEPT. OF ELECTRICAL ENGINEERING, NEDUET

EXTERNAL ADVISOR: MR. SHAHZEB ANWAR – ENI PAKISTAN LIMITED

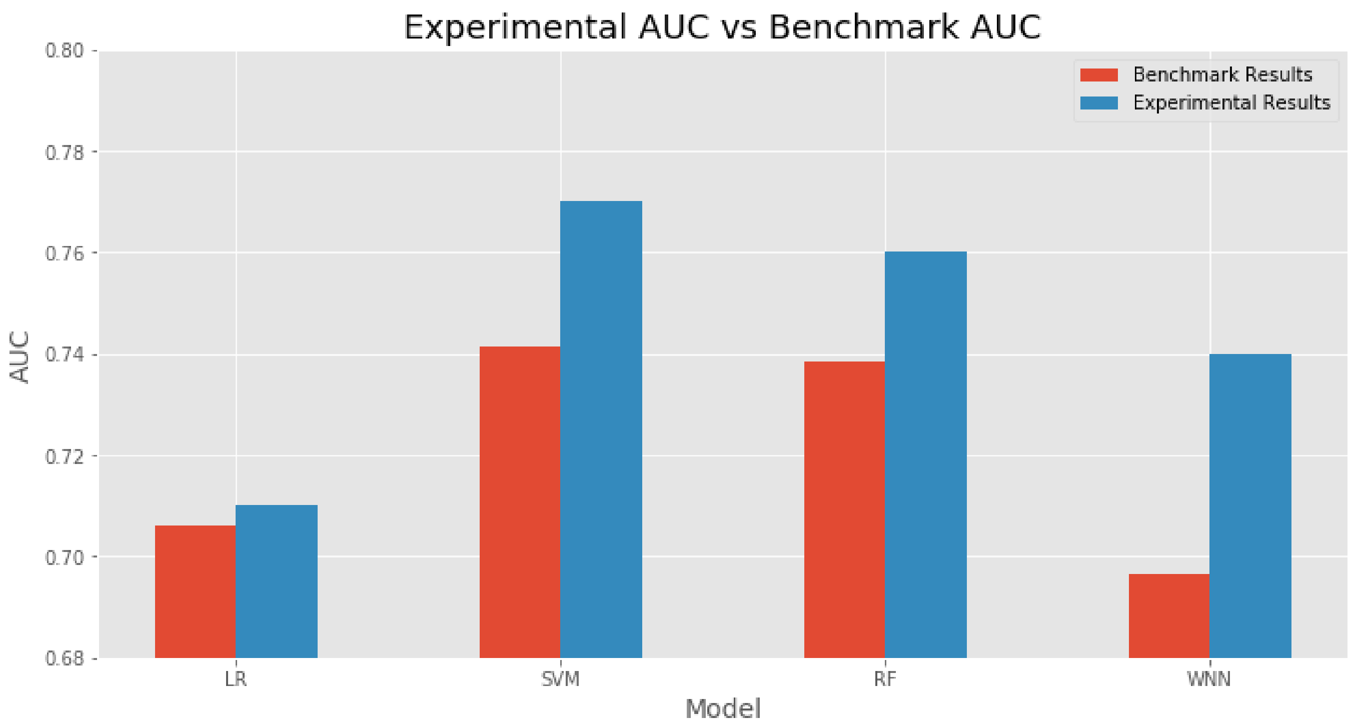
Project Recap

- **Problem**: Electricity theft detection in DX grids.
- **Solution**: Supervised Machine Learning
- **Tools**: Scikit-Learn, Keras



Project Recap

- **Problem**: Electricity theft detection in DX grids.
- **Solution**: Supervised Machine Learning
- **Tools**: Scikit-Learn, Keras
- **Algorithms**:
  Logistic Regression,
  Random Forest,
  Support Vector Machine,
  Wide Neural Network
- **Results**:
  RF, WNN > SVM >> LR



Results as of midyear evaluation

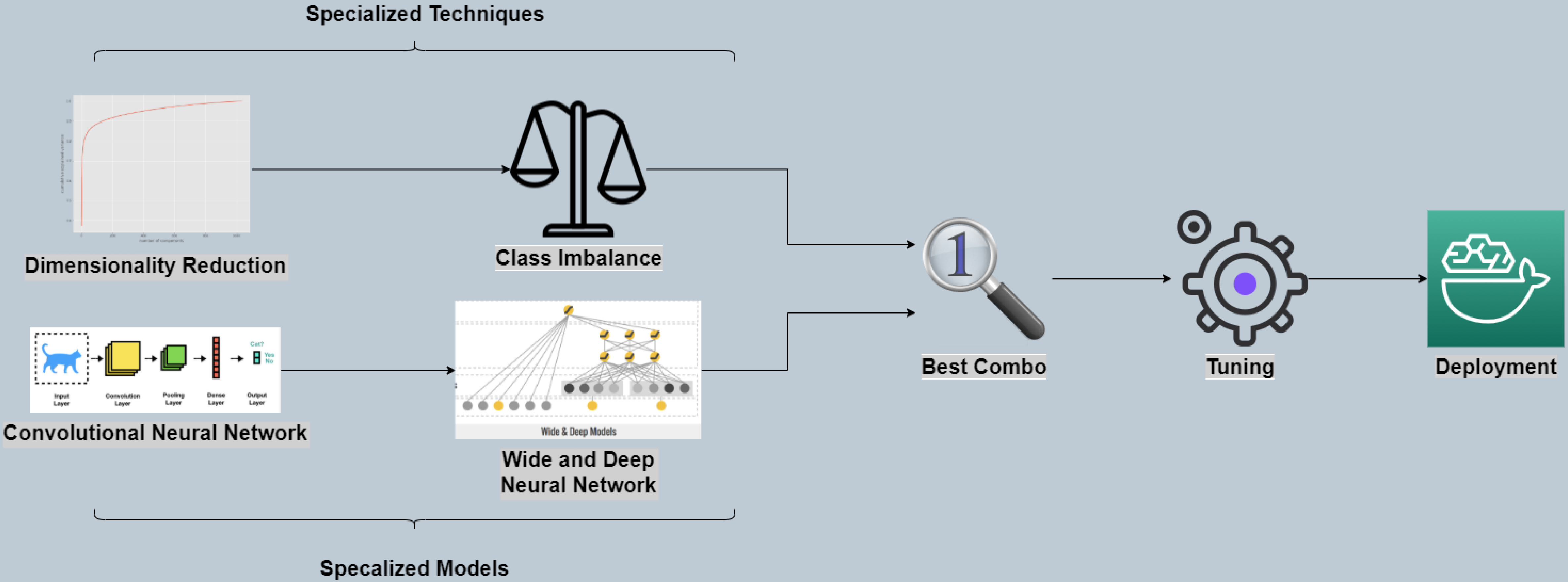
Our Project's Focus: Then and Now

SEMESTER 7
FUNDAMENTALS AND FOUNDATIONS

- Is this project even feasible? **Yes**
- Does data preprocessing make sense? **Yes**
- Does our ML workflow make sense? **Yes**
- Does hyperparameter tuning work? **Yes**
- Deep learning > shallow learning? **Yes**

SEMESTER 8
GOING BEYOND THE BASICS

- How can we circumvent runtime bottlenecks?
- Could a DL model learn from 2D data?
- Which hyperparameters should we tune?
- Experimental → production model: how?



Our Progress This Semester

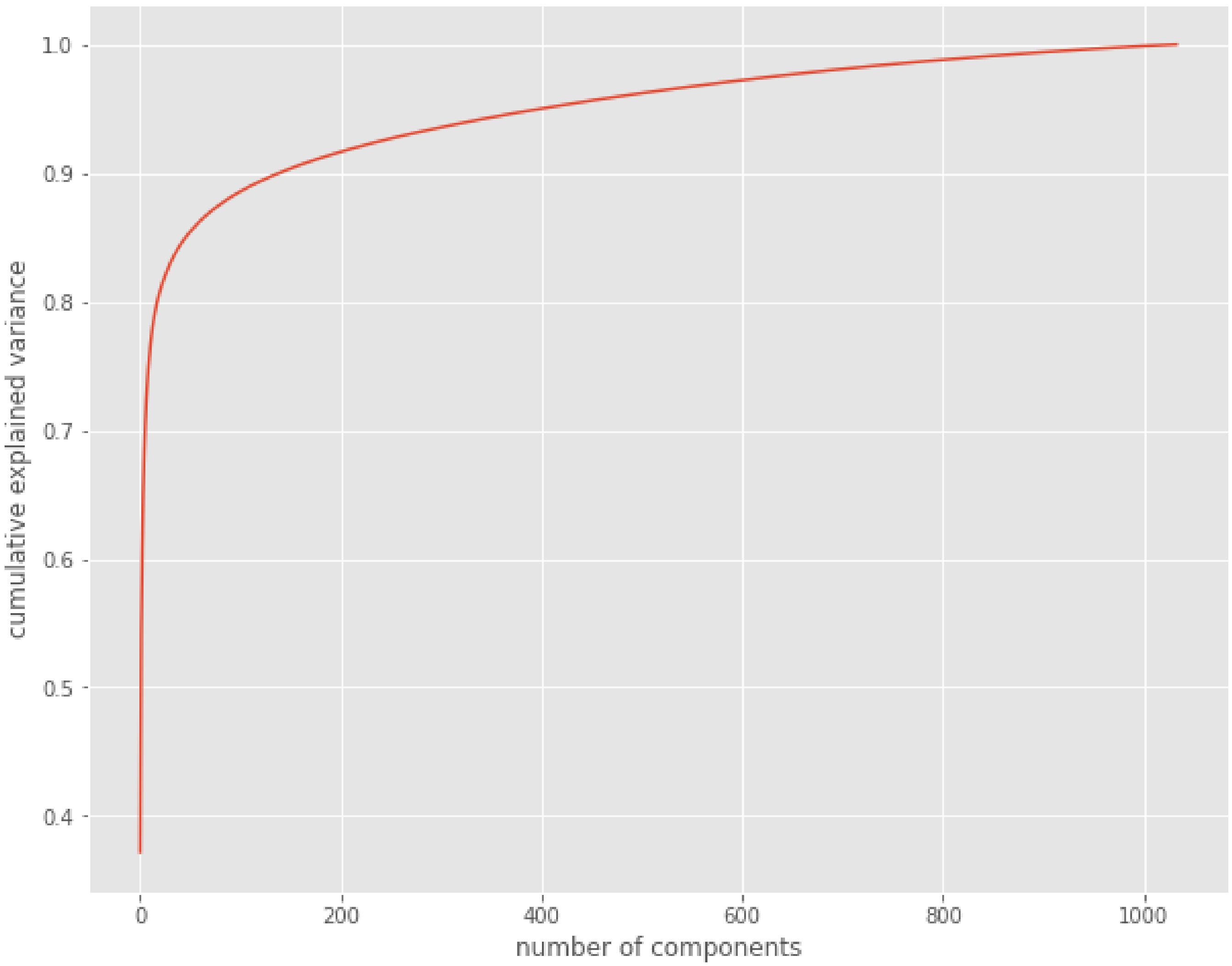
Specialized Techniques + Specialized Models → Best Model/Techniques → Tuning → Deployment

Focus 01: Specialized Techniques

DIMENSIONALITY REDUCTION & ADDRESSING CLASS IMBALANCE

Dimensionality Reduction

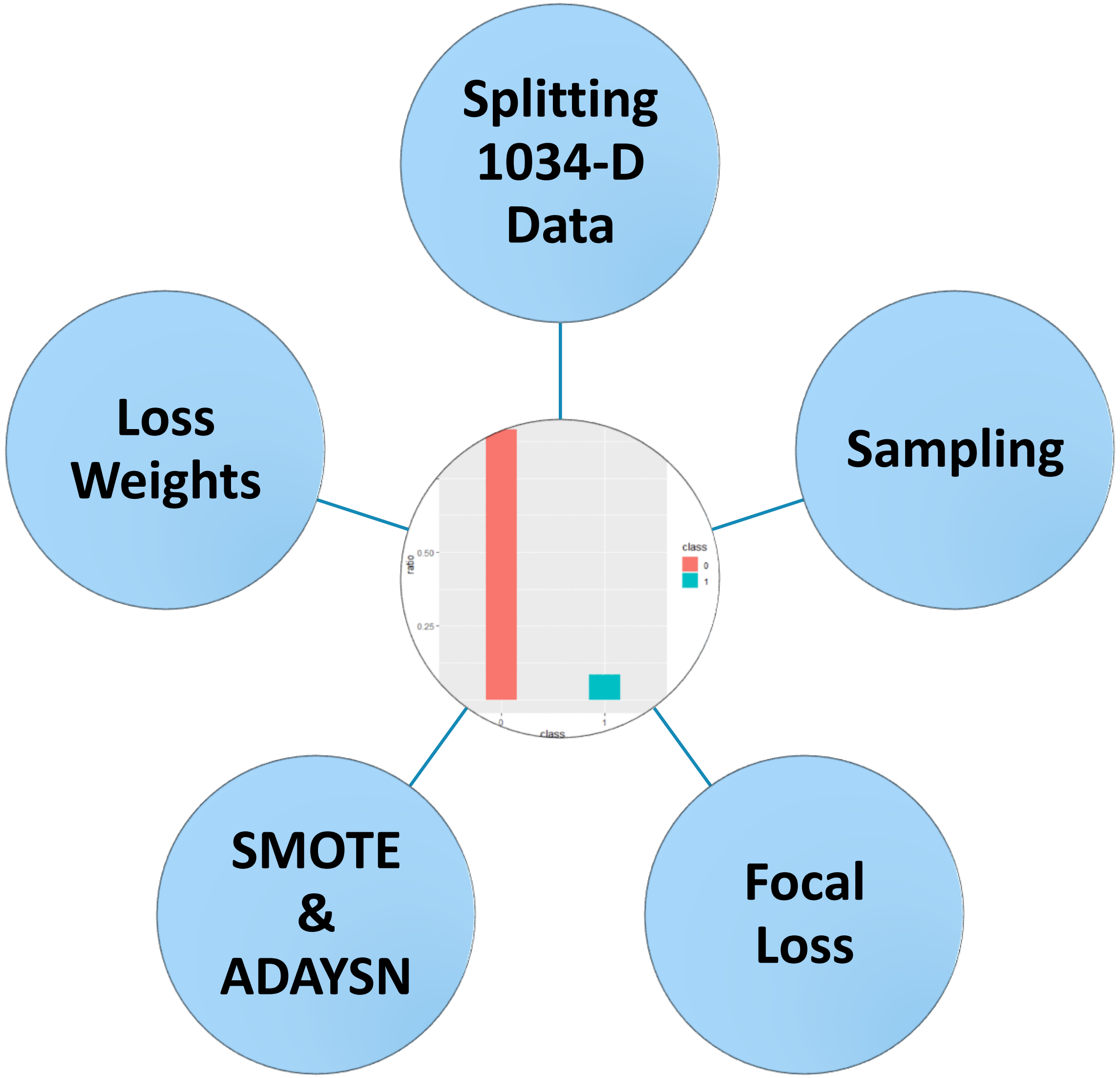
- Curse of dimensionality
- Dims ↑ ⇒ Data Needed, Overfitting, $O(n)$ ↑↑↑!
- SVM → AUC ☺, $O(n)$ ☹
- 1,034-D data necessary?
- Methods: PCA, K-PCA, LLE
- **K-PCA: 600 features ≈ 95% of explained σ^2**
- Convergence ↓, AUC ↓
- Unsuitable for our models



95% explained variance in SGCC dataset attributable to ~600 kWhs

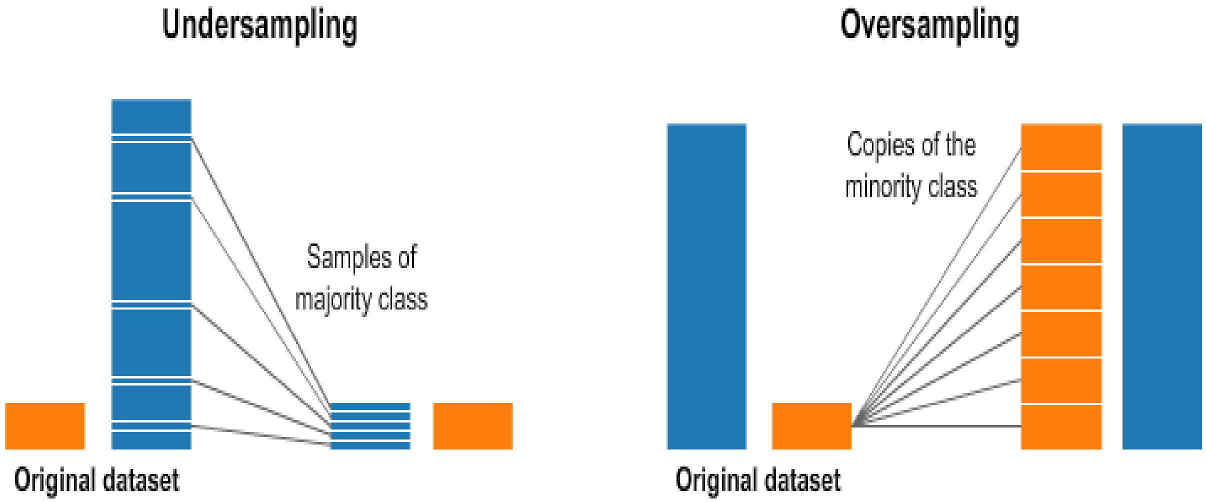
Addressing Class Imbalance

- Class 0:1 = 0.915:0.0815
- Highly imbalanced data
- $y_{majority}$ → prediction ↑
- $y_{minority}$ → prediction ↓
- AUC skewed by $n_{majority}$
- AUC ↑, but FP/FN also ↑
- 5 different techniques
- **Goal: AUC ↑, FP/FN ↓↓**



Addressing Class Imbalance – Oversampling, Undersampling, Split Features

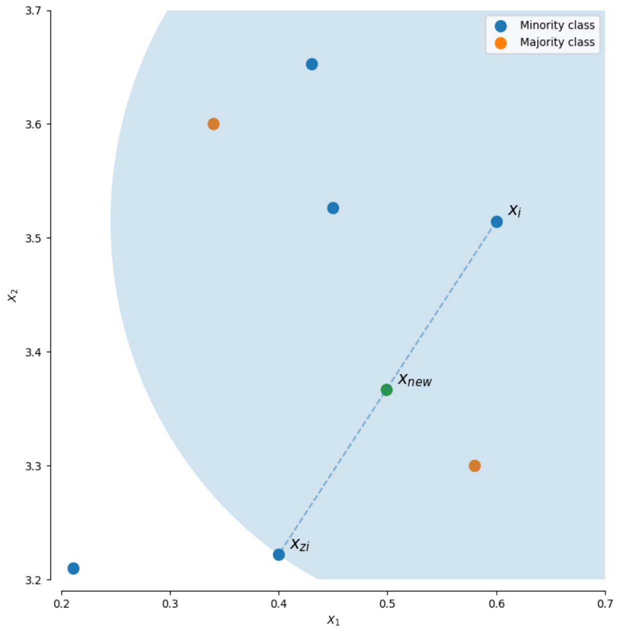
- Oversampling Minority Class
  - Randomly replicate minority class data points
  - $n_{minority} \uparrow = n_{majority}$ possible, but duplicates
  - Massively imbalanced data → overfit minority
- Undersampling Majority Class
  - $\downarrow n_{majority} = n_{minority}$
  - Randomly remove majority class examples
  - Removes potentially useful data.
- Split features
  - (1, 1034) kWh → (3, 345) kWh from same X_i
  - Fully sparse feature vectors
  - Label granularity – labels assumed, not known



Undersampling and Oversampling visualized

Addressing Class Imbalance: SMOTE, ADASYN, & Focal Loss

- SMOTE & ADASYN
  - Synthetic Minority Oversampling Technique
  - Create synthetic $X_{minority}$ in feature space.
  - $x_{new} = x_i + \lambda \times (x_{zi} - x_i), \lambda \in [0,1]$
  - Variants – Borderline/SVM-SMOTE, ADASYN
  - Performance limited by data quality
- Focal Loss – Sigmoid Focal Crossentropy
  - Facebook AI – loss for foreground segmentation
  - Designed for highly imbalanced datasets
  - Misclassified example → class weight ↑
  - Overfitting ↓↓, but AUC ↓ as well



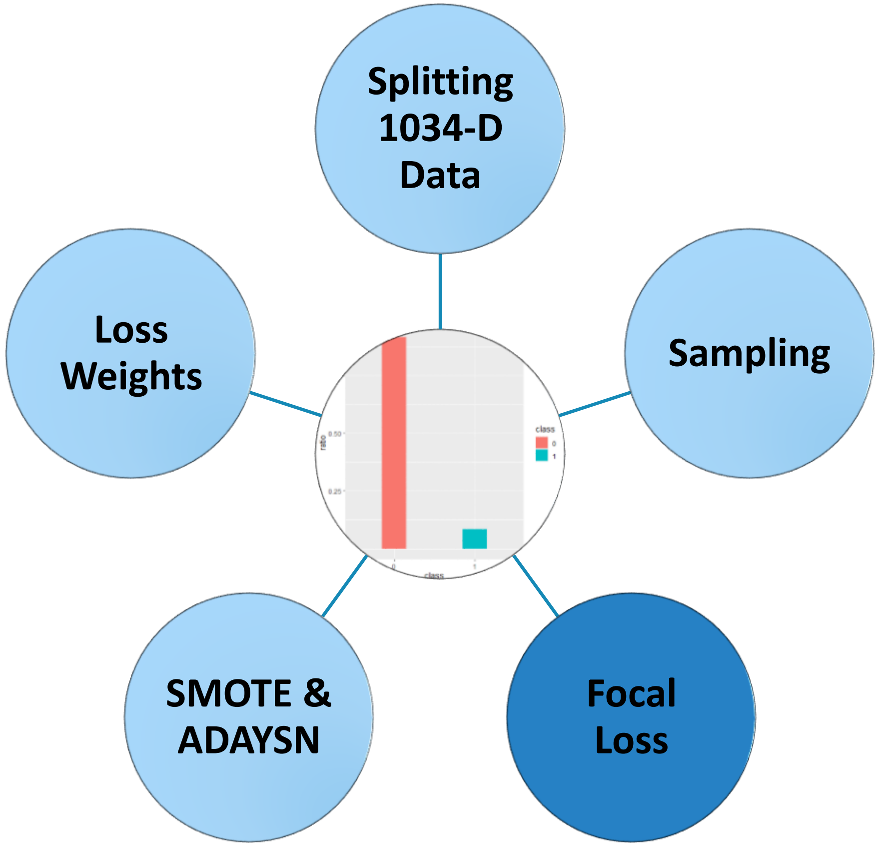
SMOTE Visualized

Addressing Class Imbalance – Loss Weights

- Zheng: both classes as equally important.
- But $n_1 \ll n_0 \Rightarrow$ more "expensive" to misclassify minority than majority class.
- Concretely, if $n_1 << n_0 \Rightarrow \mathcal{L}(X_1) \gg \mathcal{L}(X_0)$
- To make classes equally important for learning, assign weights k_1 and k_0 in loss

$$k_1 \times n_1 = k_0 \times n_0 = 0.5$$

- Make misclassified minorities k times more expensive for loss than misclassified majority
- Best technique for class imbalance → best balance b/w AUC ↑ and FP/FN ↓

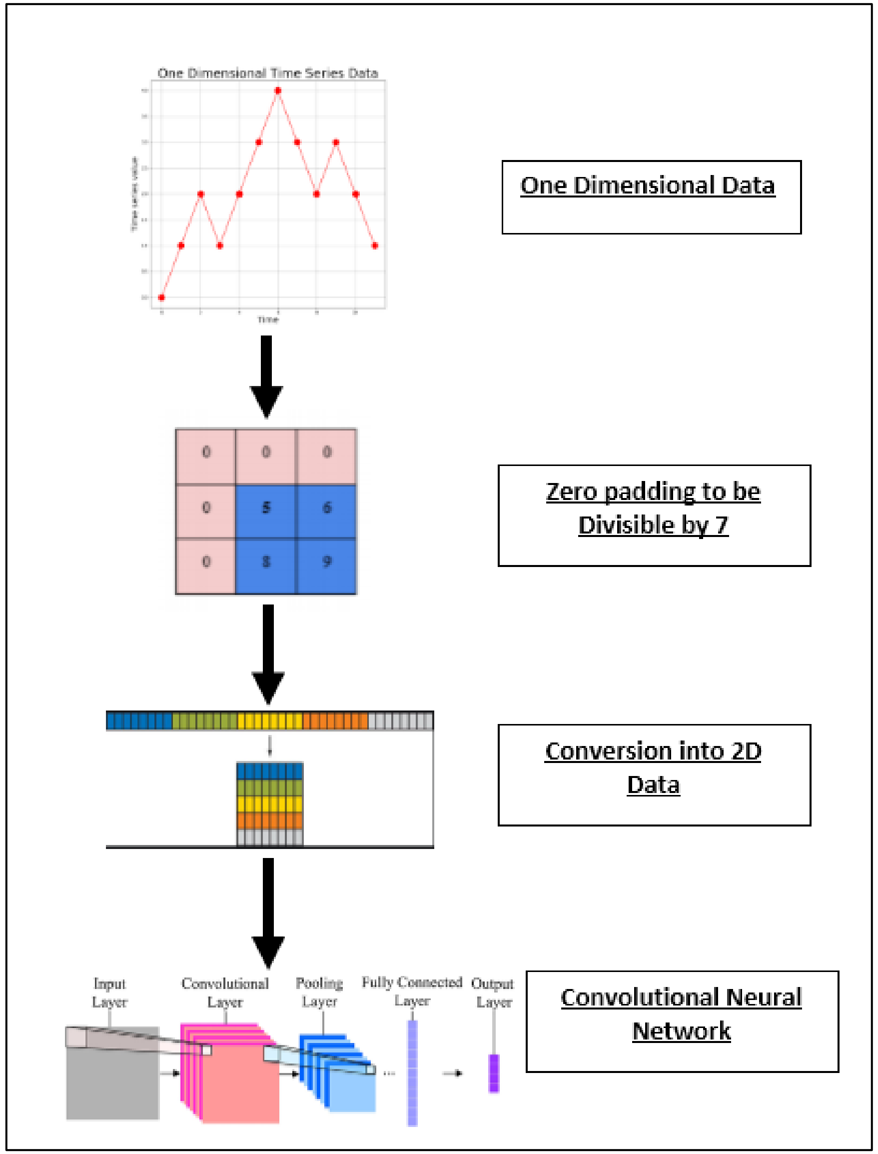


Focus 02: Specialized Models

CONVOLUTIONAL NEURAL NETWORK &
WIDE AND DEEP CONVOLUTIONAL NEURAL NETWORK

Convolutional Neural Network

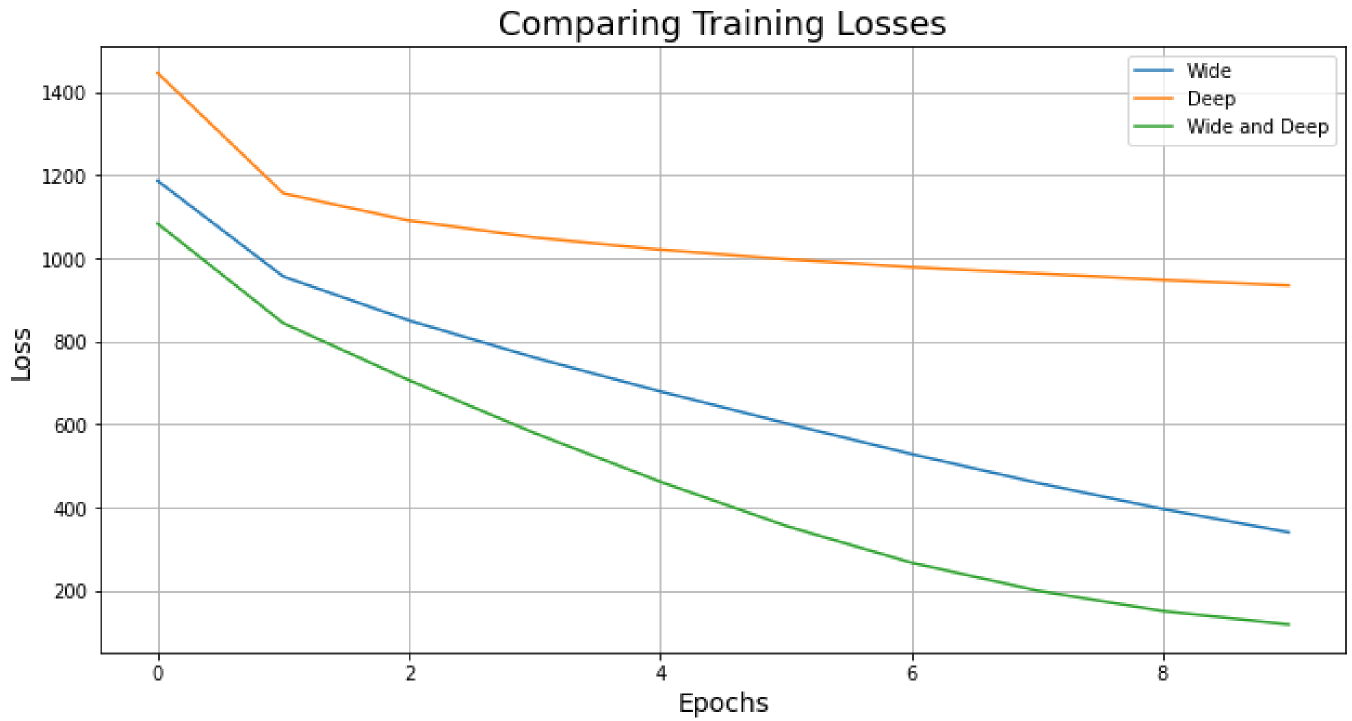
- Hierarchical feature extraction → Generalizability
- **Convolve** 2D windows of multichannel tenors with filters
- **Filters**: learned feature extractors
- **Stride:** distance between centers of adjacent convolutional windows.
- **Padding:** to be divisible by 7
- **Pooling:** strongest feature extracted by a convolution & reduces the dimensionality of the input patch.
- **Zheng**: 1D data (daily) → pad → reshape → 2D data (weekly)



CNN data flow

Wide and Deep Neural Network

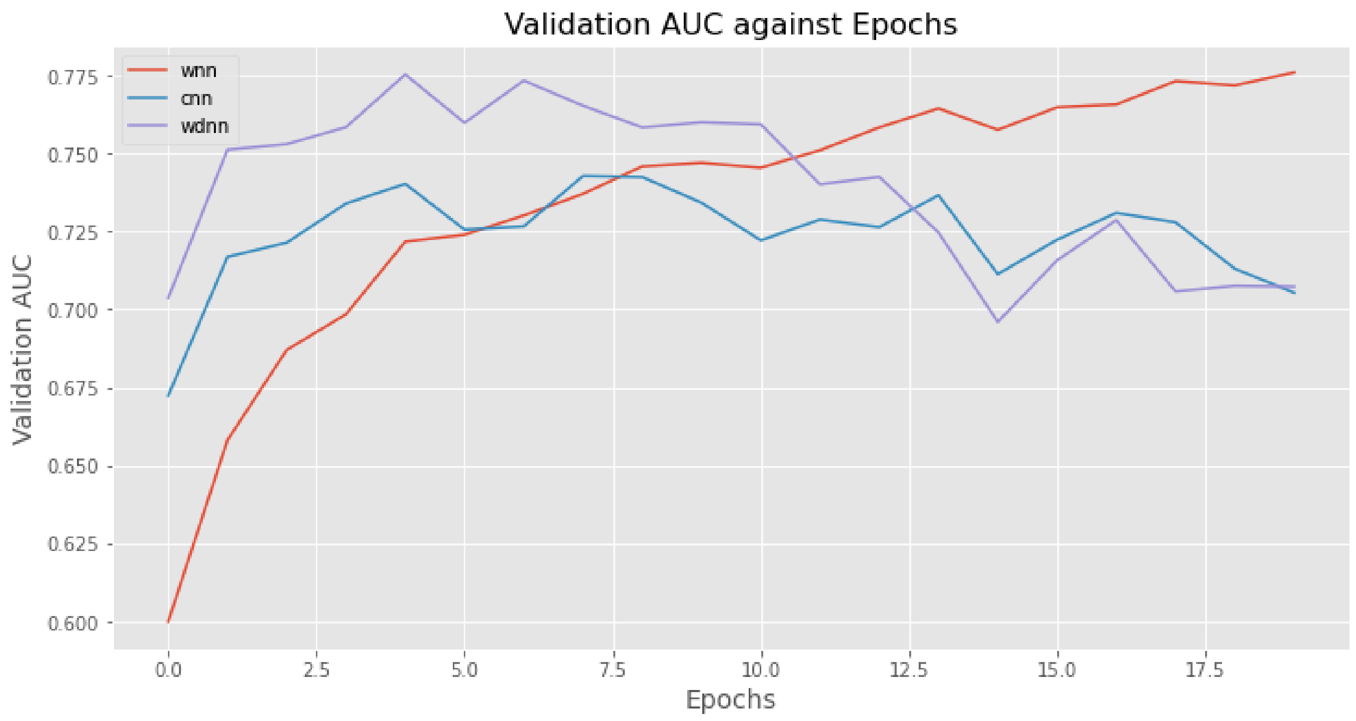
- Pioneered by Google AI
- Combines both WNN & CNN
- Generalization **AND** memorization
- Loss: WDNN < CNN < WNN
- AUC: WNN > CNN > WDNN
- 1D tensor for WNN & 2D/3D tensor for CNN component.
- WNN + CNN O/P → Sigmoid
- **Outperforms other classifiers**



Loss comparison on the canonical California Housing price dataset. WDNN outperforms both wide and deep neural networks.

Wide and Deep Neural Network

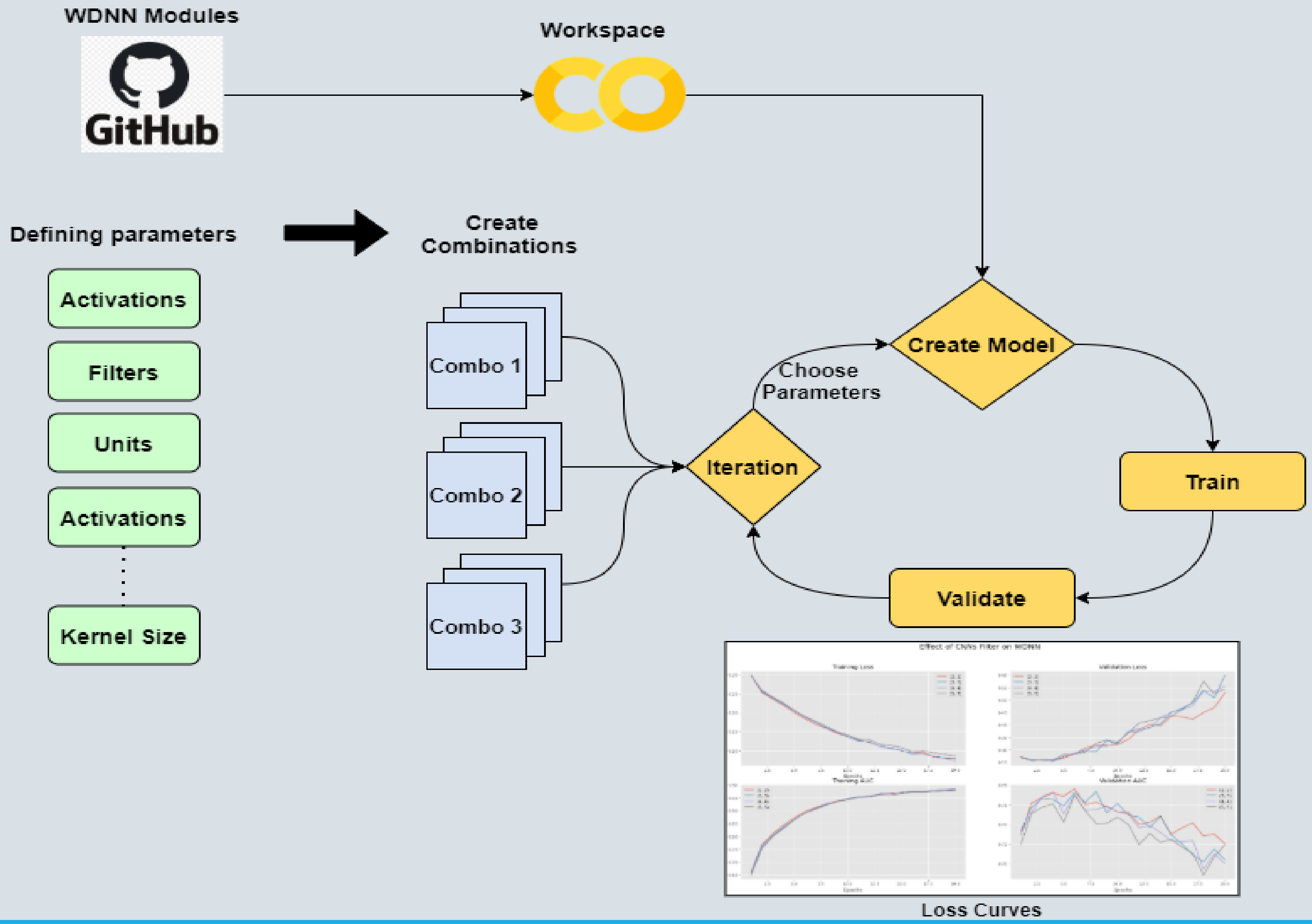
- Pioneered by Google AI
- Combines both WNN & CNN
- Generalization **AND** memorization
- Loss: WDNN < CNN < WNN
- AUC: WNN > CNN > WDNN
- 1D tensor for WNN & 2D/3D tensor for CNN component.
- WNN + CNN O/P → Sigmoid
- **Outperforms other classifiers**
- **Highest compute complexity**
- **Highest propensity to overfit**



AUC on the SGCC (FYP dataset). WDNN outperforms CNN and WNN, but tends to overfit very quickly.

Focus 03: WDNN Hyperparameter tuning

MAXIMISING WDNN MODEL PERFORMANCE

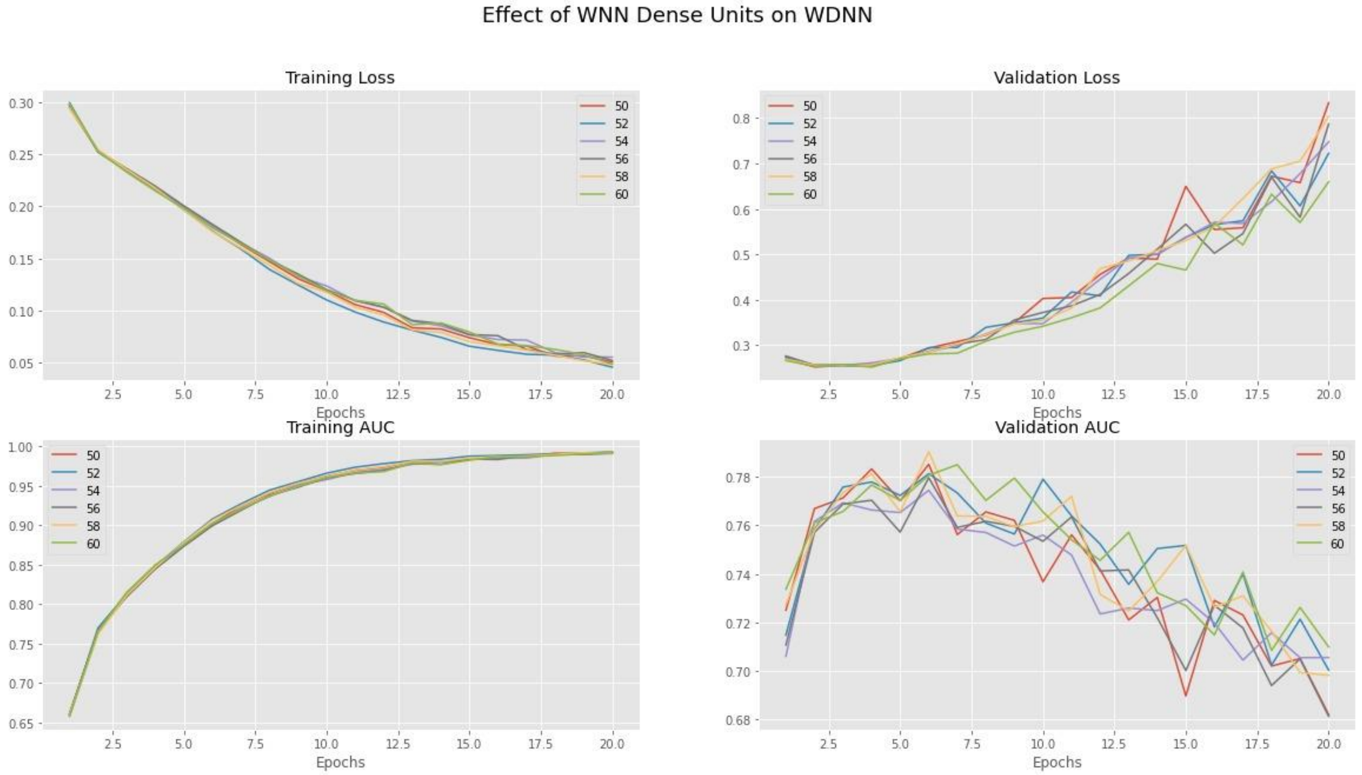


WDNN Hyperparameter tuning

A hyperparameter tuning workflow implemented from scratch for compatibility with Keras Functional API WDNN

WDNN Hyperparameter Tuning

- WDNN → `keras` functional API
- Grid Search API: incompatible
- Implemented hyperparameter tuning from scratch
- Combinations of hyperparameter values → build function.
- One model per combo.
- Plot train/val loss and AUC ∀ combos → visualize trends
- **Max AUC @ 4 /5 epochs → best value**



Tuning Visualized: trends in train/validation loss and AUC for WDNN densely connected units

WDNN Hyperparameter Tuning

- WDNN → `keras` functional API
- Grid Search API: incompatible
- Implemented hyperparameter tuning from scratch
- Combinations of hyperparameter values → build function.
- One model per combo.
- Plot train/val loss and AUC ∀ combos → visualize trends
- **Max AUC @ 4 /5 epochs → best value**

Component	Parameter	Zheng	Tuning	AUC (Optimal)
CNN	Regularizer	NA	DO OR BN	~0.79
	Activation	ReLU	SeLU	~0.76
	No. of Filters	18 ~ 20	8	~0.77 - ~0.78
	Kernel Size	(3, 3)	(3, 3)	~0.77 - ~0.78
	Dense Units	50	54	~0.78
	Dense Activ.	ReLU	SeLU	~0.78 - 0.79
	Pool Size	Max (3, 7)	Max (3, 7)	Not tuned
WNN	Units	64 (60% TR)	54	0.78
	Activation	ReLU	Softmax	0.78

Focus 04: Model Deployment

AMAZON WEB SERVICES (AWS) AND SAGEMAKER → PRODUCTION

Model Deployment: AWS

Amazon Web Services (AWS)

- Cloud computing platform for interfacing by client.
- Serving 34% of the cloud market all over the world.
- On demand computing + custom API interactions.
- Storage, GPUs, RAM & runtime optimization – online

Amazon SageMaker

- Build, train, validate, & deploy models.
- Pre-built ML model images → fine tune if needed.
- Autopilot: Sagemaker chooses best algorithm!

Model Deployment: Web Services Used

S3 Bucket

- Storage logically partitioned and set up based on user's needs.
- bucket that can be linked when a compute instance is launched.
- Bucket data uniquely identifiable by key/prefix pair → call in notebook to load data.

Notebook Instance

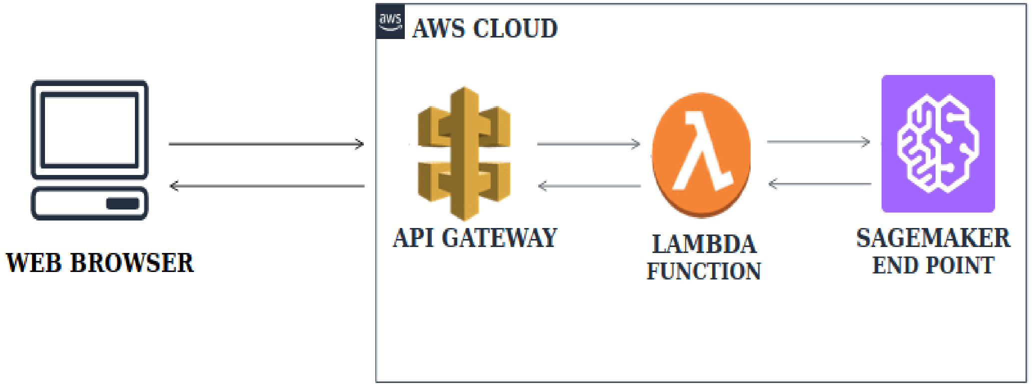
- Jupyter Notebook instances – isolated computing environments running on cloud instances.
- Can also be used for setting up model endpoints that can be invoked for making predictions.

AWS Lambda

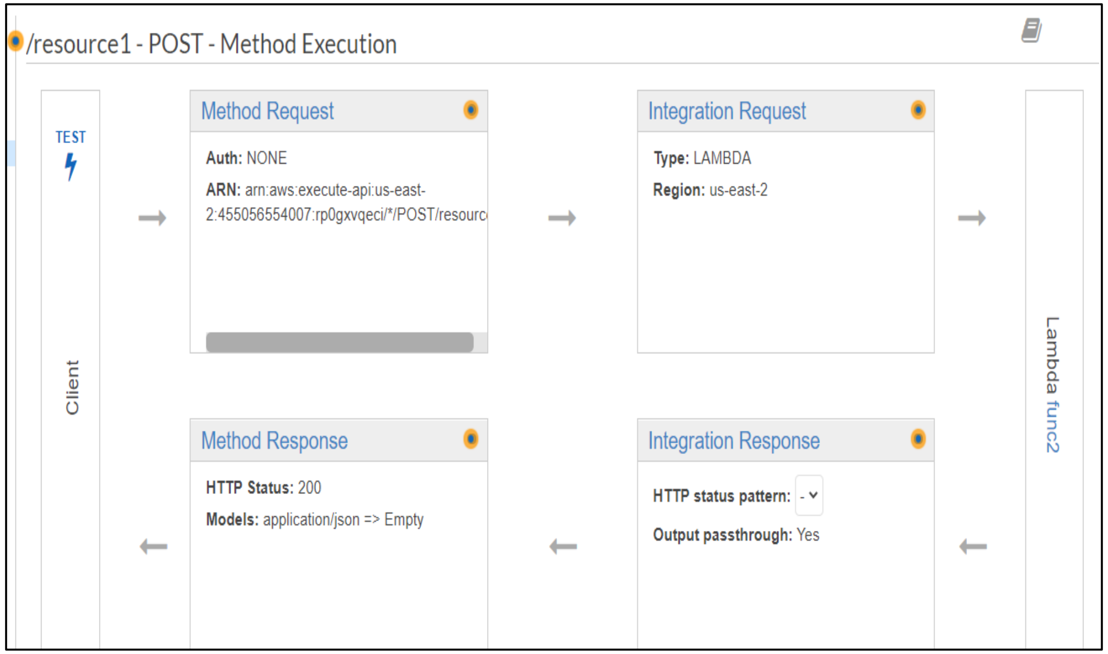
- Automates launching and managing server resources → no human intervention required.
- Instantly reallocates resources in response to a request.

Cloud Processing Workflow

- User ⟷ API Gateway ⟷ λ-function
- `lambda_handler`: stores the features in its `event` argument in the form of either a dictionary, CSV file or a JSON tree.
- `Invoke_endpoint:` client: data → compute instance → prediction
- Prediction = $p(y = 1|data)$
- Prediction → JSON → API Gateway → Response
- **One request per consumer**



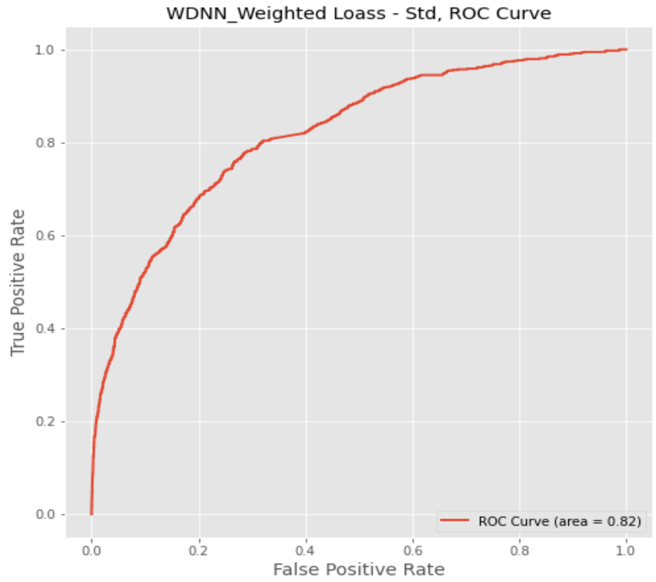
Cloud Processing Workflow



API Structure

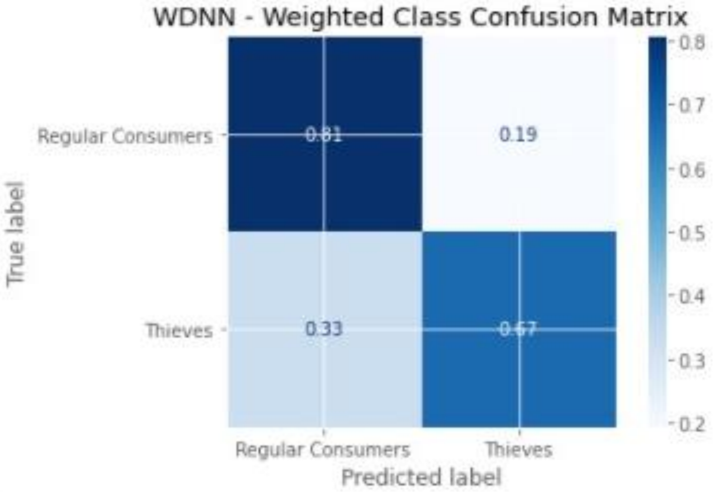
Results: WDNN

- WDNN trained with hyperparameters in slide 18.
- **Best Model: ROC AUC = 0.82**
- **Best Model: Lowest FP/FN**
- Different hyperparameters from Zheng's → ROC AUC ↑.
- Trained on standard scaled, 20 epochs, 54 dense units.
- Weighted binary crossentropy loss function
- **Softmax :** CNN component **SeLU** WNN component.
- Dropout and Batch Norm regularization for overfitting

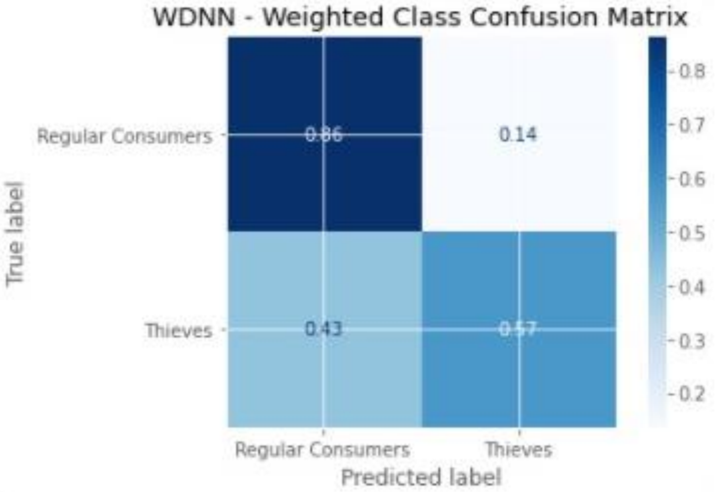


Results: Classification Threshold

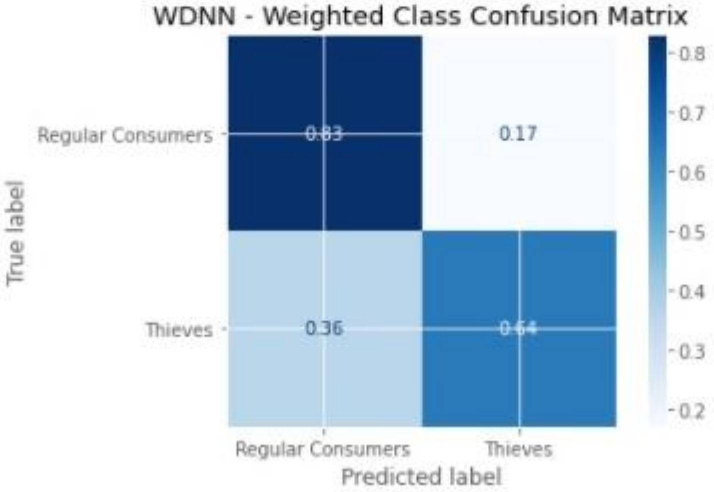
- Classification threshold = ϵ
- $\hat{y}_i = \begin{cases} 1, & P(y_i = 1|X_i) > \epsilon \\ 0, & P(y_i = 1|X_i) \leq \epsilon \end{cases}$
- $\epsilon \uparrow \Rightarrow n(\hat{y}_i = 1) \downarrow, P \downarrow$
- No “optimal” value for ϵ → depends on use case
- $\epsilon \uparrow$: TN ↓, FP ↓, TP ↓
- $\epsilon \downarrow$: TP ↑, FP ↑, TN ↓
- Tested $\epsilon \in [0.5, 0.75]$
- We recommend ϵ = 0.525



$\epsilon = 0.62$



$\epsilon = 0.72$

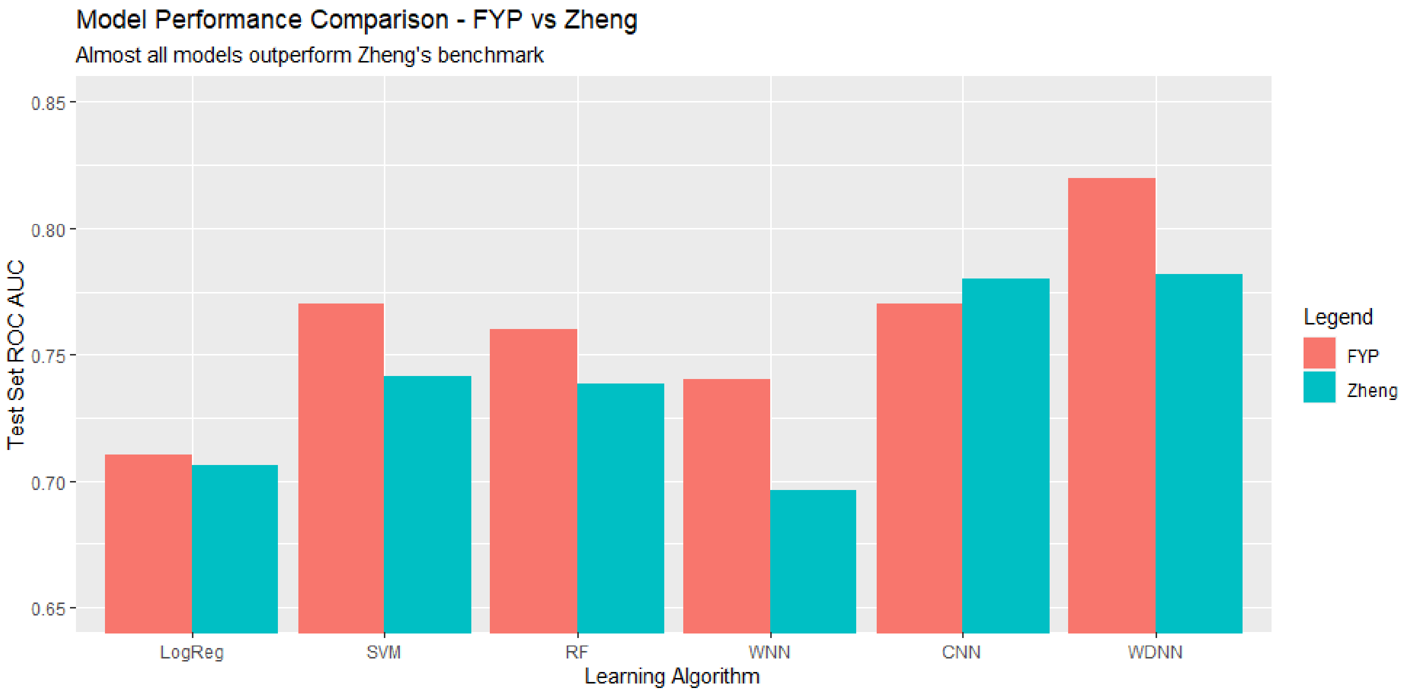


$\epsilon = 0.65$

$\boldsymbol{\epsilon = 0.525}$

Results

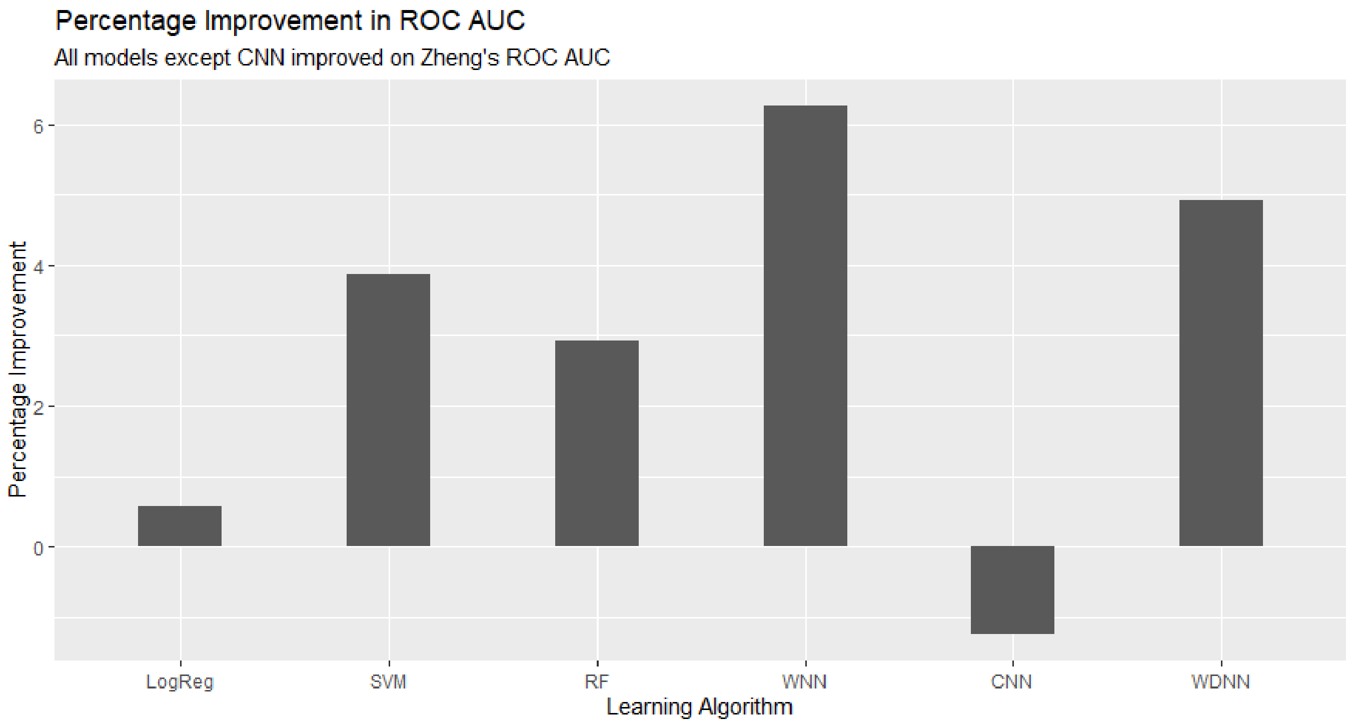
- 5/6 models: $AUC_{FYP} > AUC_{Zheng}$
- CNN results → no tuning, not using manual filter design.
- WDNN AUC ↑: feature scaling, weighed loss, extensive tuning
- Greatest improvement in WNN followed by WDNN then SVM.
- WDNN > CNN > SVM > RF > WNN > LR in terms of AUC
- WDNN ROC AUC ↑ , FP ↓ , FN ↓
- **WDNN best model for the FYP.**



Comparing ROC AUC of our model's against Zheng's benchmark

Results

- 5/6 models: $AUC_{FYP} > AUC_{Zheng}$
- CNN results → no tuning, not using manual filter design
- WDNN AUC ↑: feature scaling, weighed loss, extensive tuning
- Greatest improvement in WNN followed by WDNN then SVM.
- WDNN > CNN > SVM > RF > WNN > LR in terms of AUC
- WDNN ROC AUC ↑ , FP ↓ , FN ↓
- **WDNN best model for the FYP.**
- **Up to 6% improvement on Zheng's benchmark**



Achieved up to 6% improvement on Zheng's ROC AUC. CNN performance is lower because no hyperparameter tuning.

Conclusions

- Successful proof of concept: ML is a viable technique for detecting energy theft using kWh data.
- DL > ML: multiple non-linear transformations of feature space → richer representations of data
- FYP > Zheng: Our AUCs are up to 6% higher than Zheng's on 5/6 models.
- Best Model: WDNN: highest ROC AUC (0.82), lowest FP/FN by true label
- Class Imbalance techniques: Weighted loss by class weights > focal loss, SMOTE, sampling
- Dimensionality Reduction: PCA, KPCA unsuitable, and possibly unnecessary for this project.
- Deployment: AWS is very useful, but can be unreliable e.g. `tf`, `py` version inconsistencies
- **The Importance of Data Quality**
  - Good data >> Good models: Model performance, utility of SMOTE, PCA → all limited by data quality.
  - Data Quality ↑ if fewer missing data, more timeseries data → performance ↑

Future Work

- Sequence Models
  - Memory and state → learn meaningful, long-term dependencies and patterns in timeseries data
  - 1D-CNN, bidirectional LSTM, transformers
- Anomaly Detection
  - Semi-supervised learning: need labels for few, not all, examples
  - Identify kWhs that deviate “sufficiently” from the norm → outliers, anomalies.
  - Pre-built algorithms in Scikit-Learn
- AWS Batch Transform
  - Inference right now: one request → one consumer.
  - For scalability, one request → multiple consumers
  - AWS Batch Transform: concurrency + parallel processing → inference on multiple examples.

Thank You

QUESTIONS?