A World Leading SFI Research Centre

MLGIG Team – Diet Classification Data Challenge

[georgiana.ifrim thach.lenguyen antonio.bevilacqua bhaskar.dhariyal ashish.singh] @insight-centre.org

VistaMilk International Workshop on Spectroscopy and Chemometrics 2022

28/04/2022

HOST INSTITUTION

Teagasc Agriculture and Food Development Authority

PARTNER INSTITUTIONS

DCU UCD NUI Galway Waterford Institute of Technology Tyndall ICBF.com UCC TSSG Insight

FUNDED BY:

An Roinn Talmhaíochta, Bia agus Mara Department of Agriculture, Food and the Marine

European Union European Regional Development Fund

Science Foundation Ireland For what's next

Digitalising Dairy

About MLGIG Team

- **Georgiana** (Assoc Prof@ UCD-CS), **Thach** (postdoc VistaMilk@UCD-CS), **Antonio** (postdoc Insight@UCD-CS), **Bhaskar** (PhD VistaMilk@UCD-CS), **Ashish** (PhD Insight@UCD-CS)

- Many thanks to the organisers, we had lots of fun and learning!

- **All our code is available** (Python Jupyter notebooks): https://github.com/mlgig/vistamilk_diet_challenge

Data and Evaluation

- Removed some samples as recommended by organisers: for both training and test remove samples with col1 < 1.
- Single train/test 60/40 split to check what the model learns (first shuffle the data before split), and where possible what are the important features; 3-fold Cross Validation (3CV) to compare different algorithms.
- 3 balanced classes, compare classification **Accuracy**: fixed train/test split accuracy vs 3CV avgAcc.
- Select best model, train on the full training set, predict on the test set.

- Extra sanity check: Compare predicted class label distribution on test data with the ground truth class distribution on train data.

Modeling Strategies

1. **Tabular Models:** Each sample is a vector of (unordered) features; combine classic ML models with some steps for noise removal (e.g., feature selection).
2. **Time Series Models:** Each sample is a time series (ordered features), extract time series features and learn a classifier. Many variations including Fourier transform features.
3. **Ensemble Methods:** This approach combines time series models and tabular models.
4. **Random Poly Features + Linear Models:** This approach **explicitly generates random polynomial features** from the original features.
5. **Deep Neural Network Models:** This approach **implicitly generates more complex feature interactions.**This family of approaches is based on deep neural networks, both fully connected and convolutional.

Modeling Strategies

We started by evaluating the following models (in **bold** are the best performing models):

- LogisticRegression(), l1 and l2 penalty
- Ridge(), RidgeClassifierCV(),
- RidgeClassifierCV(normalize=True),
- **RidgeClassifierCV(alphas=np.logspace(-5, 5, 10), normalize=True),**
- **LinearDiscriminantAnalysis()**, QuadraticDiscriminantAnalysis(),
- RandomForestClassifier(n_estimators=100), GradientBoostingClassifier(n_estimators=100)
- KNeighborsClassifier(n_neighbors=1),
- MLPClassifier(),
- SVC(kernel='linear'), SVC(kernel='poly'), SVC(kernel='rbf')

Other than **RidgeClassifierCV** and **LinearDiscriminantAnalysis,**
all other models listed above work poorly for this dataset.

Results

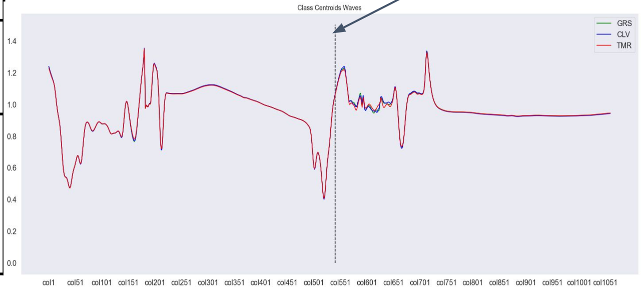
Single split and 3CV average Accuracy for best models

Method	**Acc SingleSplit (60/40)**	**avgAcc 3CV**
RidgeClassifierCV(alphas=np.logspace(-5, 5, 10), normalize=True)	0.771	0.760
LinearDiscriminantAnalysis()	0.737	0.747
SelectFromModel(RidgeClassifierCV) + RidgeClassifierCV (312 features selected of 1060)	0.782	0.777
SelectFromModel(RidgeClassifierCV) + LinearDiscriminantAnalysis()	0.795	0.778
No water region: [0:171, 206:535, 729:747] RidgeClassifierCV(alphas=np.logspace(-5, 5, 10), normalize=True)	0.790	0.777
No water region: [0:171, 206:535, 729:747] LinearDiscriminantAnalysis()	0.795	0.783
Auto noise removal (ranked windows, Ridge, wsize=90, idx=10, removes region [540-720]) + SelectFromModel(RidgeClassifierCV) + LinearDiscriminantAnalysis()	**0.805**	**0.780**

Auto ranked windows by avg validation accuracy:

[('0:90', 0.687),
('90:180', 0.684),
('450:540', 0.577),
('180:270', 0.476),
('270:360', 0.458),
('360:450', 0.440),
('810:900', 0.383),
('900:990', 0.379),
('990:1060', 0.351),
('540:630', 0.349),
('720:810', 0.343),
('630:720', 0.342)]

col540



Take-away

- **Feature selection + RidgeClassifierCV or LDA works well** **(3CV avgAcc: 78%).** Water region removal improves accuracy a bit. **Automatic noise removal through ranking windows seems promising** and points to regions that overlap with the water regions. Time series methods and ensembles do not work well for this problem**.**

- **Explicit feature expansion with polynomial features + feature selection + LDA works very well** **(3CV avgAcc: 86.4%),** with or without noisy region removal. Feature selection is critical if noisy regions were not removed manually from the data.

- **Implicit feature expansion using deep models (FCN) with tuned network architecture works very well when the water region is removed** **(3CV avgAcc: 84.8%).** On the whole data without noise removal, FCN does not work well. Spatial dependencies do not seem to occur among the wave components (CNN lower acc than FCN).

- What happens after wave 540? Using only the waves [0, 360], [450, 540] seems to be enough for high accuracy.

Random Polynomial Features for Spectroscopy Data

The classic idea: Enriching the feature space with polynomial features e.g., a*a, b*b, a*b

A simple implementation with *sklearn*:

```
clf = Pipeline([
      ('fpoly', PolynomialFeatures(degree=2)),
      ('classification', LinearDiscriminantAnalysis())
])
```

Random Polynomial Features for Spectroscopy Data

The problem: From 1060 original features

- degree=2 creates ~ 500,000 new features.
- degree=3 creates ~ 200,000,000 new features.

⇨ Too expensive to train with the risk of overfitting.

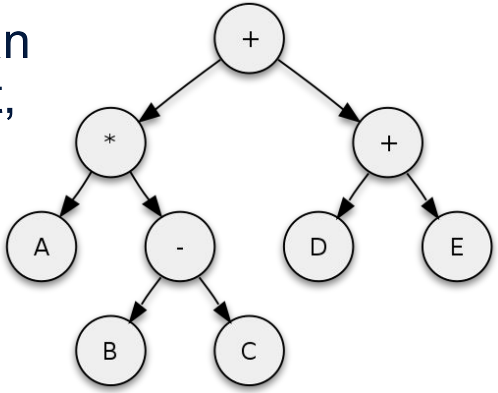
We actually managed to get it run by **adding a feature selection module** to filter the original features first (1060 => 312 features => poly2, 3CV avgAcc: 84.4%):

```
clf = Pipeline([
      ('feature_selection', SelectFromModel(RidgeClassifierCV(alphas=np.logspace(-5,
5, 10)))),
      ('fpoly', PolynomialFeatures(degree=2)),
      ('classification', LinearDiscriminantAnalysis())
])
```

Random Polynomial Features for Spectroscopy Data

Random Polynomial Features:

- Generate random (non-linear) combinations of original features in the form of arithmetic expressions.
    - Random features work well for time series classification problems (e.g. Rocket, MrSQM).
    - More flexible than sklearn PolynomialFeatures (can support mathematical function like cosine, tangent, logarithm, etc.)
    - The number of features can be controlled.
    - Non-deterministic algorithm.



VistaMilk

Random Polynomial Features for Spectroscopy Data

Experiments: Accuracy with 60/40 **single split.**

Pipeline	Whole Series	0:172	206:536	729:748	0:172 206:536	0:172 206:306*	No Water
Rpoly Transformer (k=10000) + StandardScaler + Ridge	0.731	0.775	0.731	0.338	0.794	0.829	0.814
Rpoly Transformer(k=10000) + LDA	0.616	0.825	0.751	0.356	**0.846**	0.842	0.827
Rpoly Transformer(k=5000) + LDA	0.632	**0.839**	0.764	0.352	0.834	**0.847**	0.829
Rpoly Transformer(k=10000) + SelectKBest(k=5000)+ LDA	0.676	0.826	0.767	0.359	0.831	0.842	**0.834**
SelectFromModel + Rpoly Transformer(k=10000) + StandardScaler + SelectKBest(k=5000) + LDA	**0.824**	0.812	0.748	0.356	0.833	0.828	0.822

Random Polynomial Features for Spectroscopy Data

Experiments: Average accuracy with **3-fold CV**.

Pipeline	Whole Series	0:172	206:536	729:748	0:172 206:536	0:172 206:306*	No Water
Rpoly Transformer(k=10000) + StandardScaler + Ridge	0.717	0.76	0.748	0.363	0.805	0.797	0.811
Rpoly Transformer(k=10000) + LDA	0.619	**0.833**	0.766	0.352	0.847	**0.857**	0.833
Rpoly Transformer(k=5000) + LDA	0.631	0.819	0.76	0.349	0.849	0.854	**0.843**
Rpoly Transformer(k=10000) + SelectKBest(k=5000)+ LDA	0.677	0.821	0.754	0.35	0.847	**0.861**	0.836
SelectFromModel + Rpoly Transformer(k=10000) + StandardScaler + SelectKBest(k=5000) + LDA	**0.848**	0.797	0.754	0.349	0.843	0.83	0.835

VistaMilk

Random Polynomial Features for Spectroscopy Data

Tuning Hyperparameters with GridSearchCV:

- Numb_features: Number of nonlinear features created.
- Keep_origin: Whether to keep the original features.
- depth : Maximum depth of the arithmetic expression.
- Kbest: Number of top features to select.

```
Best model: Pipeline(steps=[('bespoke',
                 RandomPolynomialTransformer(keep_origin=True,
                                             num_kernels=17000)),
                 ('normalizer', StandardScaler()),
                 ('constantfilter', VarianceThreshold()),
                 ('selectkbest', SelectKBest(k=7000)),
                 ('model', LinearDiscriminantAnalysis())])
Best Score: 0.8637497627479177
```

Random Polynomial Features for Spectroscopy Data

Examples of important nonlinear features:

```
((X[:,108] - X[:,379] * X[:,3]) * (X[:,4] - (X[:,469] + X[:,47])))
((X[:,200] - (X[:,318] * X[:,439])) * (X[:,113] - X[:,157]))
((X[:,76] - X[:,215]) * (X[:,424] - X[:,409]))
(X[:,47] * X[:,31]) - X[:,461] * X[:,25]
(X[:,485] * X[:,195] * X[:,37]) * (X[:,349] - X[:,171]))
(X[:,326] - (X[:,506] - X[:,280]) * (X[:,303] * X[:,468] - X[:,329]))
```

Interpretation ?

Random Polynomial Features for Spectroscopy Data

Take-away:

- For this problem, (*,-,+) seems to be sufficient to create richer features.
- Faster but as accurate as bruteforce sklearn PolynomialFeatures.
- Quite sensitive to noise in the data. Pairing with feature selection can help.

Next: Deep Learning Models

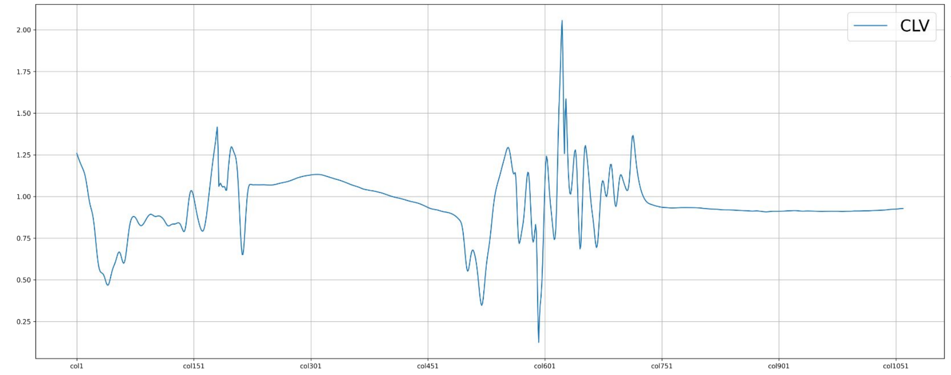
Deep Neural Networks Approach

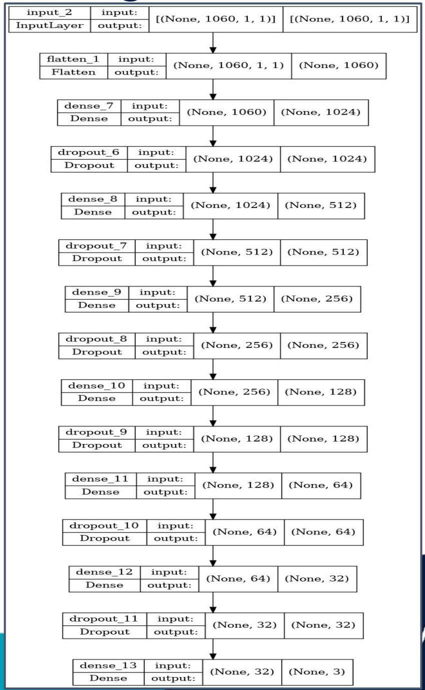
The designed models can be grouped into 2 main categories, depending on the input modality.

- **Fully Connected Networks** (FCNs): input waves are linearly fed into a neural network. No further manipulation is required.
- **Convolutional Neural Networks** (CNNs): input waves are shaped into squared, bidimensional image-like matrices.

Input Modality 1/2: FCN

The input layer of the generated network accepts the sequence of values composing the waves. Depending on whether water regions are included or excluded, input values can be 1060 or 518.





```
input_2       | input:  | [(None, 1060, 1, 1)] | [(None, 1060, 1, 1)]
InputLayer    | output: |
flatten_1     | input:  | (None, 1060, 1, 1)   | (None, 1060)
Flatten       | output: |
dense_7       | input:  | (None, 1060)         | (None, 1024)
Dense         | output: |
dropout_6     | input:  | (None, 1024)         | (None, 1024)
Dropout       | output: |
dense_8       | input:  | (None, 1024)         | (None, 512)
Dense         | output: |
dropout_7     | input:  | (None, 512)          | (None, 512)
Dropout       | output: |
dense_9       | input:  | (None, 512)          | (None, 256)
Dense         | output: |
dropout_8     | input:  | (None, 256)          | (None, 256)
Dropout       | output: |
dense_10      | input:  | (None, 256)          | (None, 128)
Dense         | output: |
dropout_9     | input:  | (None, 128)          | (None, 128)
Dropout       | output: |
dense_11      | input:  | (None, 128)          | (None, 64)
Dense         | output: |
dropout_10    | input:  | (None, 64)           | (None, 64)
Dropout       | output: |
dense_12      | input:  | (None, 64)           | (None, 32)
Dense         | output: |
dropout_11    | input:  | (None, 32)           | (None, 32)
Dropout       | output: |
dense_13      | input:  | (None, 32)           | (None, 3)
Dense         | output: |
```

Input Modality 2/2: CNN

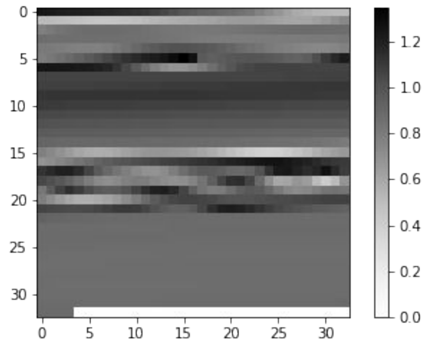
Waves can be reshaped into squared bidimensional structures. This requires some padding to exactly fit a wave into a square.

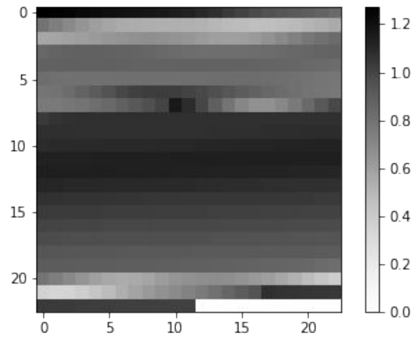
- Full wave: 33 * 33 = 1089 [1060 components + 29 padding]
- No water: 23 * 23 = 529 [518 components + 11 padding]

Input Modality 2/2: CNN

Waves can be reshaped into squared bidimensional structures. This requires some padding to exactly fit a wave into a square.

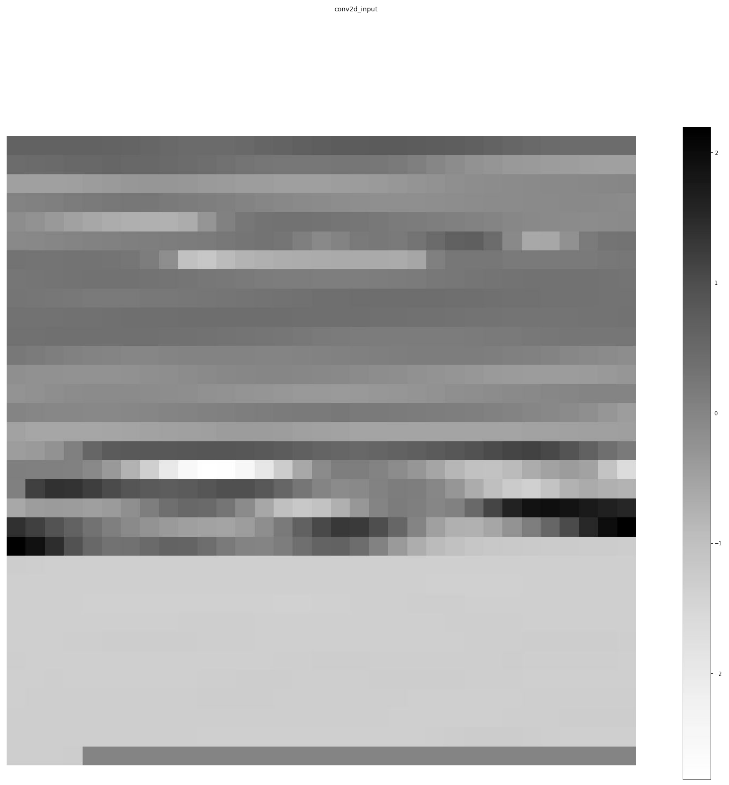
- Full wave: 33 * 33 = 1089 [1060 components + 29 padding]
- No water: 23 * 23 = 529 [518 components + 11 padding]



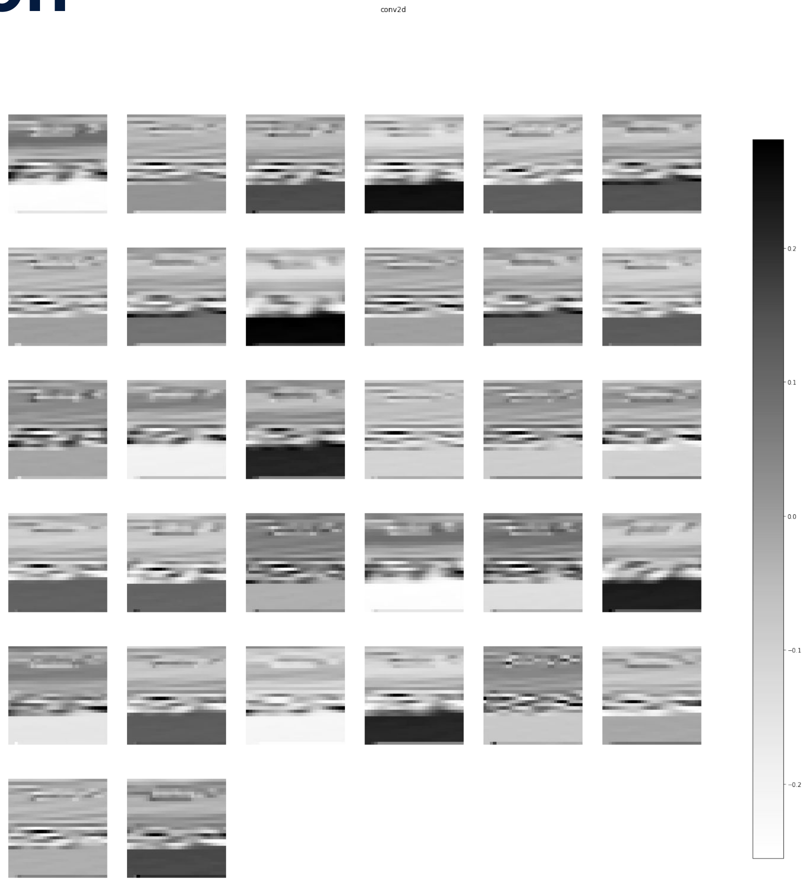


VistaMilk

Filters in Action

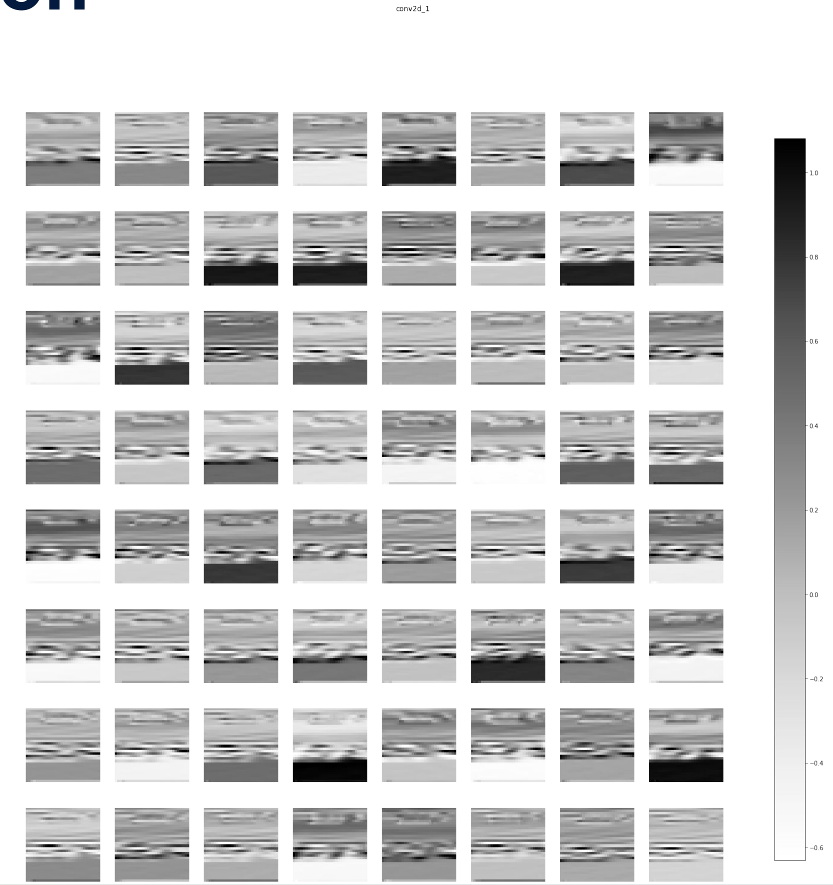


Filters in Action



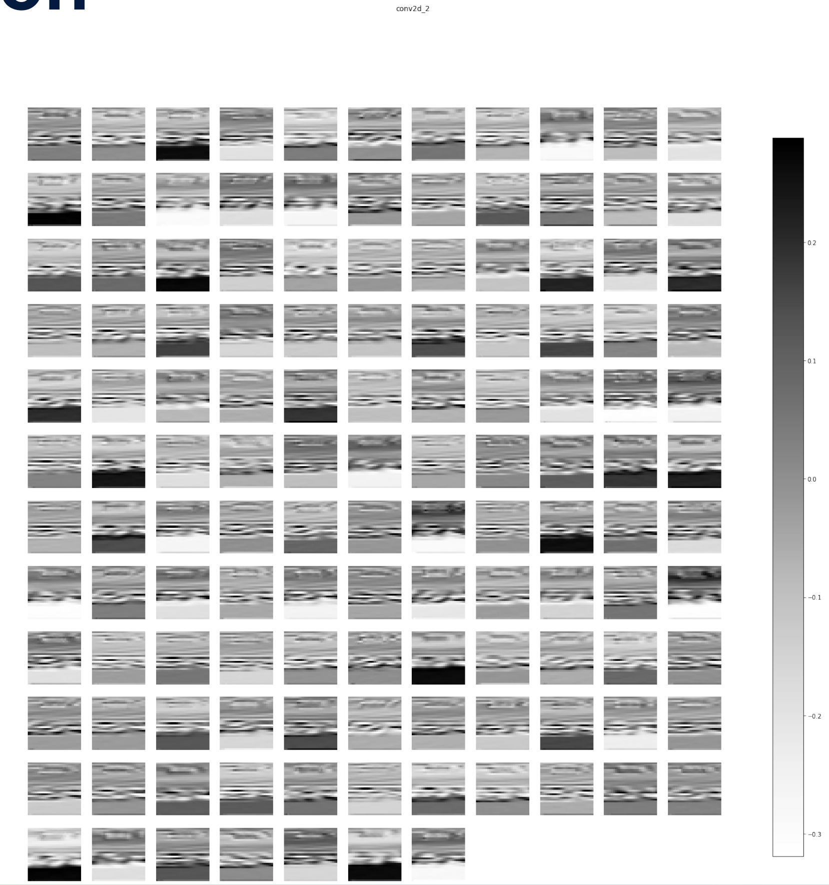
VistaMilk

Filters in Action



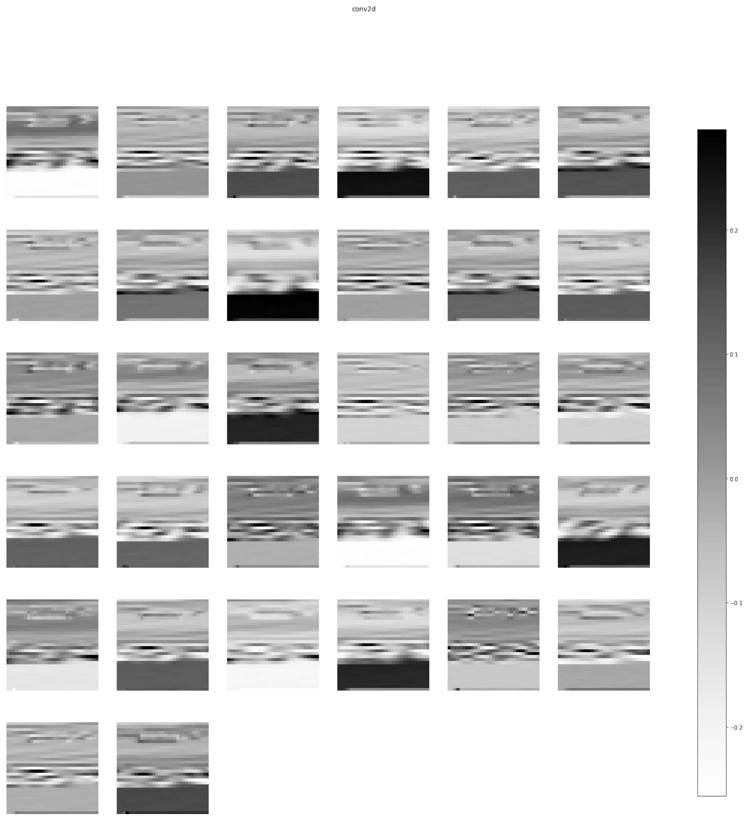
VistaMilk

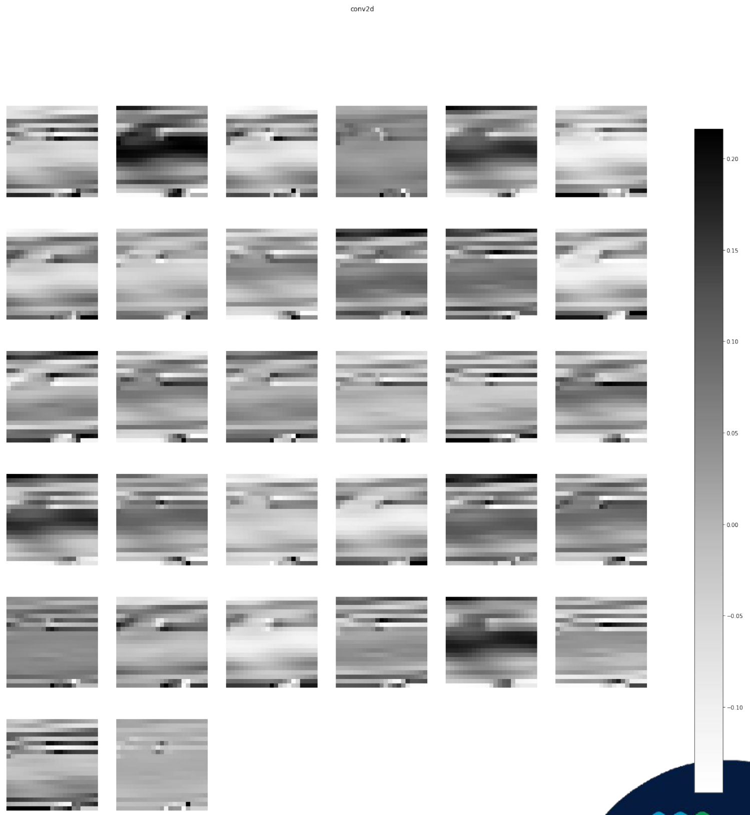
Filters in Action



VistaMilk

Full Wave vs. Reduced Wave

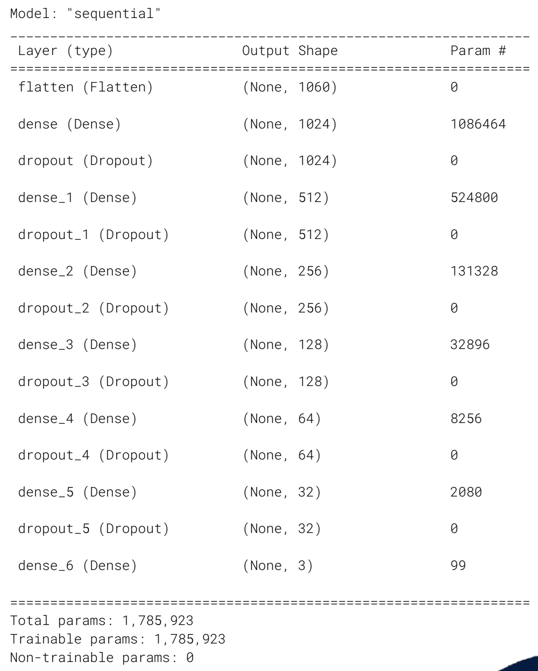




Fully Connected Network

Basic FCN:

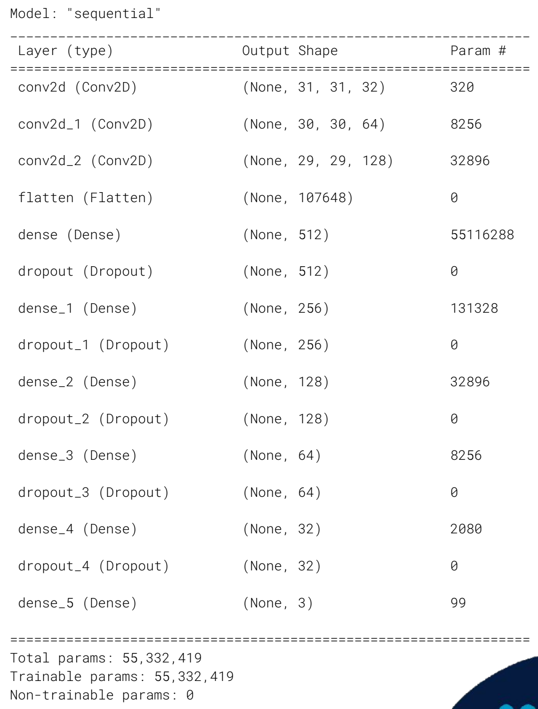
- Dense layers with 1024, 512, 256, 128, 64, 32 units
- Interleaving dropout layers with 0.2 drop rate
- Output layer with 3 units
- `elu` activation function for intermediate layers
- `softmax` activation function for output layer



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 1060)              0

 dense (Dense)               (None, 1024)              1086464

 dropout (Dropout)           (None, 1024)              0

 dense_1 (Dense)             (None, 512)               524800

 dropout_1 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 256)               131328

 dropout_2 (Dropout)         (None, 256)               0

 dense_3 (Dense)             (None, 128)               32896

 dropout_3 (Dropout)         (None, 128)               0

 dense_4 (Dense)             (None, 64)                8256

 dropout_4 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 32)                2080

 dropout_5 (Dropout)         (None, 32)                0

 dense_6 (Dense)             (None, 3)                 99

=================================================================
Total params: 1,785,923
Trainable params: 1,785,923
Non-trainable params: 0
```

Convolutional Neural Network

Basic CNN:
- Convolutional layers with 32, 64 and 128 filters, of shape (3, 3), (2, 2) and (2, 2)
- Flattening layer
- Dense layers of 512, 256, 128, 64, and 32 units
- Output layer with 3 units
- `elu` activation function for intermediate layers
- `softmax` activation function for output layer

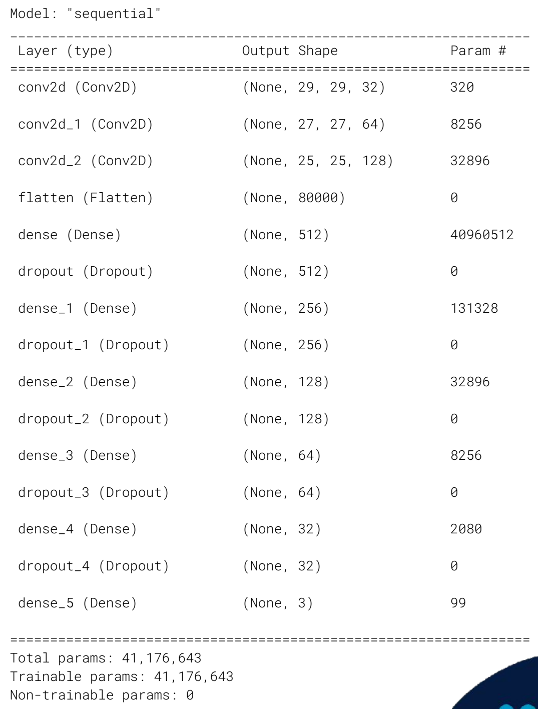


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 31, 31, 32)        320

 conv2d_1 (Conv2D)           (None, 30, 30, 64)        8256

 conv2d_2 (Conv2D)           (None, 29, 29, 128)       32896

 flatten (Flatten)           (None, 107648)            0

 dense (Dense)               (None, 512)               55116288

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 128)               32896

 dropout_2 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 64)                8256

 dropout_3 (Dropout)         (None, 64)                0

 dense_4 (Dense)             (None, 32)                2080

 dropout_4 (Dropout)         (None, 32)                0

 dense_5 (Dense)             (None, 3)                 99

=================================================================
Total params: 55,332,419
Trainable params: 55,332,419
Non-trainable params: 0
```

Dilated CNN

Similar to previous CNN, but convolutional kernels are build with a dilation rate of (2, 2).

This could help spotting spatial dependencies among components further apart in the spectrum.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 29, 29, 32)        320

 conv2d_1 (Conv2D)           (None, 27, 27, 64)        8256

 conv2d_2 (Conv2D)           (None, 25, 25, 128)       32896

 flatten (Flatten)           (None, 80000)             0

 dense (Dense)               (None, 512)               40960512

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 128)               32896

 dropout_2 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 64)                8256

 dropout_3 (Dropout)         (None, 64)                0

 dense_4 (Dense)             (None, 32)                2080

 dropout_4 (Dropout)         (None, 32)                0

 dense_5 (Dense)             (None, 3)                 99

=================================================================
Total params: 41,176,643
Trainable params: 41,176,643
Non-trainable params: 0
```

VistaMilk

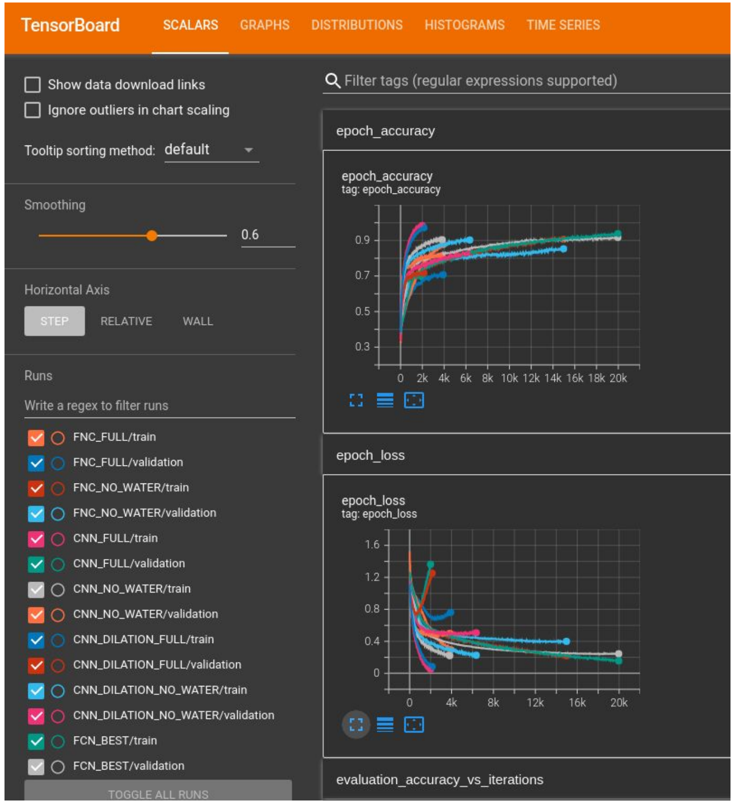
Implementation and Training

- Training runs for a maximum of 50k epochs
- Early stopping is used to prevent overfitting and save time
- Target metric to manage training is validation loss

- All the networks are implemented using Tensorflow 2.8.0
- Training data can be stored and evaluated using Tensorboard
- Reasonably fast (on GPU!): ~2hr for full-wave CNN (55M params)

Metric Visualization

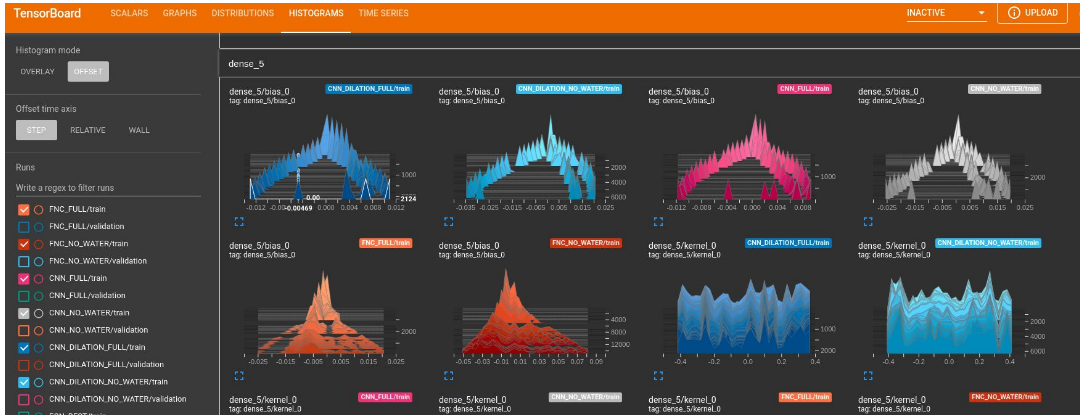
- Automatically store training info and display them live (while training is in progress).
- Useful to explore value distribution across layers, over epochs (histograms).



Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis: [✓ STEP] RELATIVE WALL

Runs
Write a regex to filter runs

- [✓] FNC_FULL/train
- [✓] FNC_FULL/validation
- [✓] FNC_NO_WATER/train
- [✓] FNC_NO_WATER/validation
- [✓] CNN_FULL/train
- [✓] CNN_FULL/validation
- [✓] CNN_NO_WATER/train
- [✓] CNN_NO_WATER/validation
- [✓] CNN_DILATION_FULL/train
- [✓] CNN_DILATION_FULL/validation
- [✓] CNN_DILATION_NO_WATER/train
- [✓] CNN_DILATION_NO_WATER/validation
- [✓] FCN_BEST/train
- [✓] FCN_BEST/validation



Histogram mode: OVERLAY [✓ OFFSET]
Offset time axis: [✓ STEP] RELATIVE WALL

Runs
Write a regex to filter runs

- [✓] FNC_FULL/train
- [ ] FNC_FULL/validation
- [✓] FNC_NO_WATER/train
- [ ] FNC_NO_WATER/validation
- [✓] CNN_FULL/train
- [ ] CNN_FULL/validation
- [✓] CNN_NO_WATER/train
- [ ] CNN_NO_WATER/validation
- [✓] CNN_DILATION_FULL/train
- [ ] CNN_DILATION_FULL/validation
- [✓] CNN_DILATION_NO_WATER/train
- [ ] CNN_DILATION_NO_WATER/validation

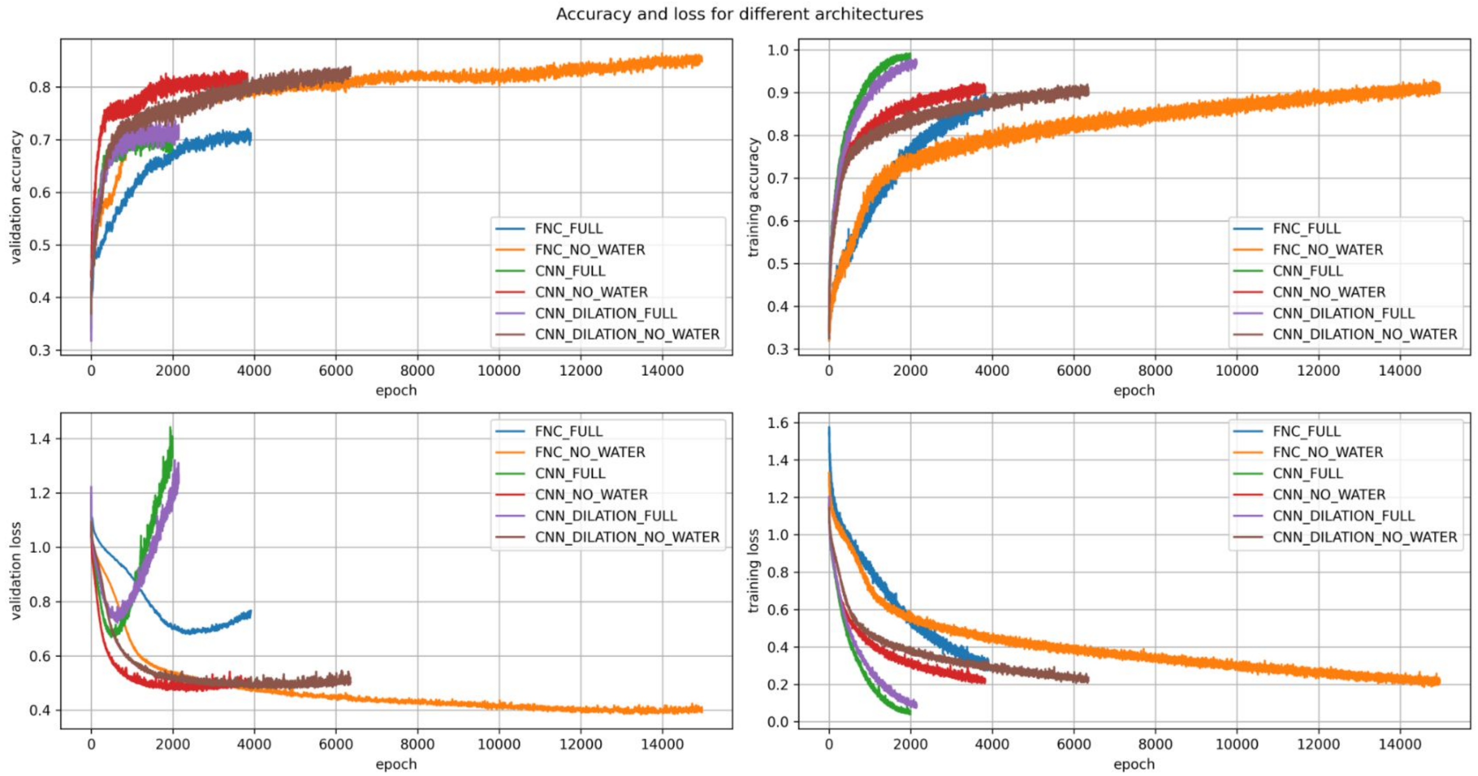
VistaMilk

Single Split Results (60/40)

The training set was further partitioned and a 20% was used as validation set.

Architecture	Data type	Val. acc.	Val. loss	Epochs	**Test acc.**	Test loss
FCN	FULL	0.6871	0.6797	~2400	0.6625	0.7689
FCN	**NO_WATER**	0.8564	0.3845	~13500	**0.8482**	0.4088
CNN	FULL	0.6692	0.6706	~500	0.6664	0.7222
CNN	NO_WATER	0.8153	0.4714	~2300	0.8243	0.4296
CNN_DILATED	FULL	0.6846	0.7412	~1100	0.6517	0.7792
CNN_DILATED	NO_WATER	0.8102	0.4884	~5300	0.8243	0.4821

Single Split Results (60/40)



VistaMilk

3CV Results

Architecture	Data type	1 acc.	2 acc.	3 acc.	avg.
FCN	FULL	0.67	0.6771	0.6753	0.6741
FCN	**NO_WATER**	**0.8548**	**0.851**	**0.8371**	**0.8477**
CNN	FULL	0.6866	0.6845	0.6706	0.6806
CNN	NO_WATER	0.8059	0.8362	0.8325	0.8249
CNN_DILATED	FULL	0.6783	0.6845	0.6521	0.6716
CNN_DILATED	NO_WATER	0.8243	0.8122	0.8075	0.8147

Lessons Learned

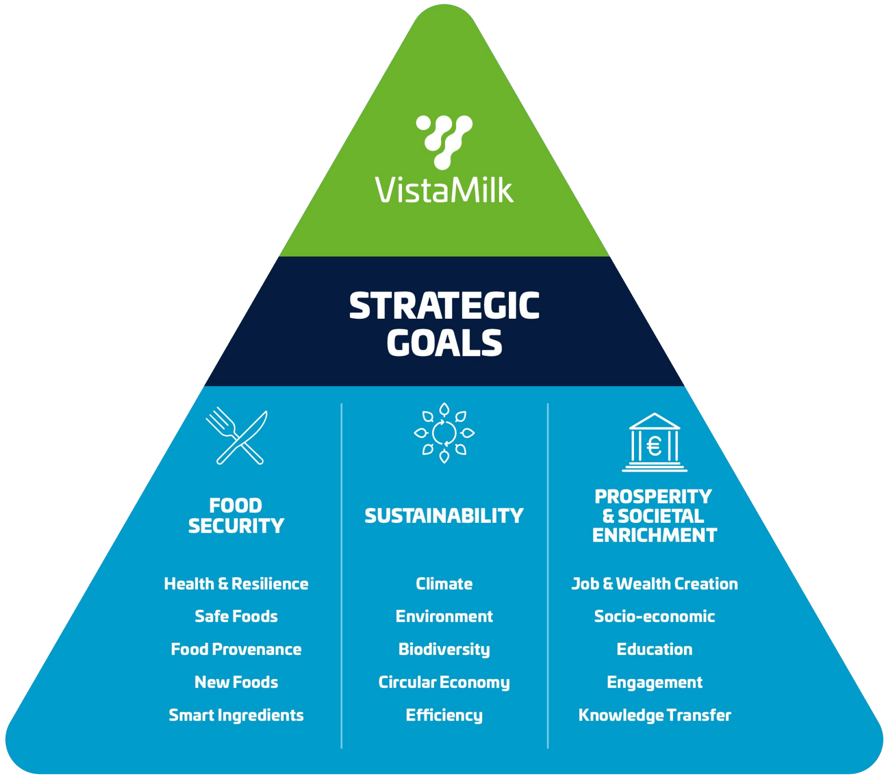
- Deep models are very sensitive to data normalisation and noisy regions. All tested architectures exhibited an increase in accuracy of 15-20% when water regions were removed from the data.

- It seems like spatial dependencies among wave components are not particularly interesting: overall, CNNs do not outperform FCNs.

- CNNs are the most sensitive to noisy data, and this might be caused by the fact that local receptive fields are often likely to cover noisy regions.

Thank you!

QA


VistaMilk
STRATEGIC GOALS
FOOD SECURITY
Health & Resilience
Safe Foods
Food Provenance
New Foods
Smart Ingredients
SUSTAINABILITY
Climate
Environment
Biodiversity
Circular Economy
Efficiency
PROSPERITY & SOCIETAL ENRICHMENT
Job & Wealth Creation
Socio-economic
Education
Engagement
Knowledge Transfer