Fine-Grained Activity Detection in the Kitchen with UWB

University of Alberta

Steven Phan

08 April 2023

Contents

Chapter 1

Introduction

Chapter 2

Literature Review

Chapter 3

System Testing and Tuning at the Independent Living Suite

Chapter 4

Classification of Time Series Data

Based on the findings in Chapter 3, it was determined that Machine Learning (ML) would be a suitable method for classifying the data obtained from the Pozyx system. Prior to attempting any experimentation, this chapter will conduct a survey of the current ML techniques available for the classification of Time Series Data. These techniques can be split into the Classic ML methods and Neural Networks.

4.1 Classic Machine Learning Methods

4.1.1 k-Nearest Neighbours (kNN)

Overview

The k-Nearest Neighbours is a classifier that assigns a class to an unlabeled observation by looking at the class of k neighbours and choosing the class that appears the most within these k neighbours. The "nearest neighbours" are determined through a measure of Euclidean-Distance D between a neighbour p and unlabeled point q with n features is shown in Equation 4.1 [81]

$$D(p,q) = \sqrt{(p_1 - q_1)^2 + (p_1 - q_2)^2 + ...(p_n - q_n)^2} \tag{4.1}$$

Using Equation 4.1 the top k neighbours with the lowest distance are considered and the class that occurs the most within these nearest neighbours is the class chosen for the unlabeled observation. In the event that there is a tie between the classes, resolution of the tie depends on the implementation of the library used. In R kNN() resolves ties by picking a random tied candidate class [81] and scikit-learn's kNeighboursClassifier uses scipy's "mode" method which returns the first class that ties in the array [82].

Dynamic Time Warping (DTW) as a Distance

Though Euclidean-Distance may be suitable for problems with a fixed feature set, or vectors with the same length, patterns that manifest in time-series data may be stretched, compressed, or shifted in the temporal domain. Using Euclidean distance, a signal compared with the same signal shifted $t + 1$ may result in large distances even though the signals compared were the same signals [83]. A method called Dynamic Time Warping (DTW), was devised to counter the shortcomings of Euclidean distance and is robust to temporal variation that occurs naturally. It achieves this by calculating all

the distances from one point to every other point and chooses a path that costs the least, finding the minimum distance between two vectors that do not ha e to be the same length. Mathematically, for $X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_m)$ be two time series of length n and m respectively, DTW first begins with calculating the Cost Matrix which is the cost between each pair in the time series [83]

$$\forall_i \in 1, ..., n, \forall_j \in 1, ..., n, C_{ij} = f(i, j) \tag{4.2}$$

Where f is the squared Euclidean-Distance (equation 4.1 squared) for multivariate time-series [83].

$$f(x, y) = D(x, y)^2 \tag{4.3}$$

A warping path is defined as $p = (p_1, ..., p_L)$ through the cost matrix C begins on $p_1 = (1, 1)$, must end on $p_L = (n, m)$ and moves with step $p_{l+1} - p_1 \in \{(1, 0), (1, 1), (0, 1)\}$. The costs in the along the paths are summed And the DTW score is the path that costs the least [83].

$$DTW(X, Y) = \min_{p \in P} C_p(X, Y) \tag{4.4}$$

Though every path does not need to be calculated as dynamic programming can be leveraged, the time complexity is still high at $O(nm)$ [83] and this time complexity must be considered when designing a real-time classification system.

Tunable Parameters

The tunable parameter in kNN is k and controls the number of neighbours selected. The optimal k should be empirically chosen by varying k for the selected dataset. When using kNN, the entire training dataset is used as-is. Therefore, class imbalance should also be considered. For example, a higher number of class A than class B could lead to a higher density of class A. Even though the unlabeled observation may be closer to class B, the classifier may choose class A because class B has "run out" of neighbours. With increasing k, the inaccuracy caused by class imbalance becomes even more apparent. If k exceeds the number of samples in the class with the lower number of samples, then the class with the higher number of samples will always be chosen.

4.1.2 Support Vector Machines (SVM)

Overview

Support Vector Machines (SVM) involves finding a hyperplane that best separates 2 classes. There are two cases, the linearly separable case and the non-linearly separable case. In the linearly separable case, the hyperplane perfectly separates the two classes (see Figure 4.1). The idea is to maximize the "margin" which is the training data that is closest to the hyperplane [84]. Cervantes et al. depict this as maximizing the distance between parallel hyperplanes H1 and H2 (which pass through the "support vector" data-points of the training dataset) in Figure 4.1 to optimize the generalization capability of the model.

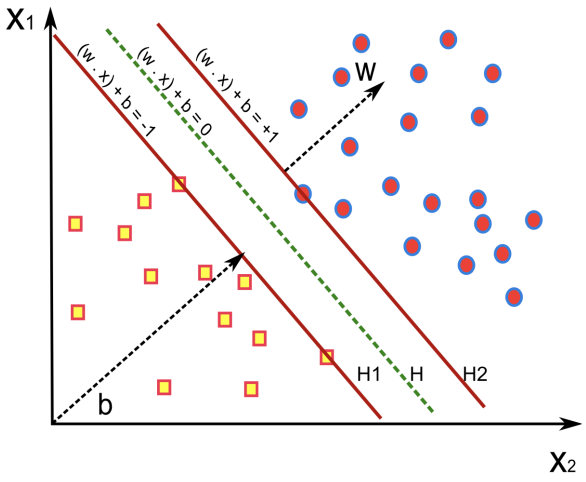


Figure 4.1: SVM Best Hyperplane in the linearly separable case [84].

The linearly separable case is rare in real life. Instead, datapoints from one class may seem to mix with the other class at the optimal boundary (Figure 4.2). To still find this optimal boundary, a positive slack factor ζ_i is introduced and controlled by a parameter C that controls the width of the margin. A lower C allows for a wider margin that may improve generalizability in real life at the cost of more misclassification of the training datatset, whereas a higher C tightens the margin and minimizes the classification errors in the training dataset but may decrease generalizability in real life [84].

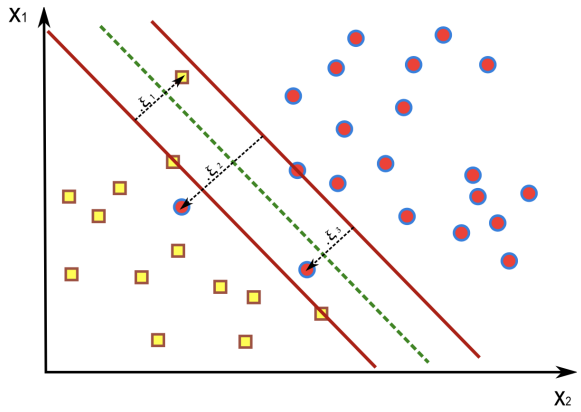


Figure 4.2: SVM Best Hyperplane in the non linearly separable case [84].

SVM can also be used separate data that is not separable by a linear hyperplane (Figure 4.3) by using a kernel. Kernels reframe the problem in a highly dimensional space called the "feature space" and in this feature space, it is simple for the algorithm to find the hyperplane. In the next subsection that will detail the convex optimization problem for SVM, it is found that the function that needs to be optimized depends on the inner product between every sample $\langle x_i \cdot x_j \rangle$. Transforming the original dataset to this feature space through a transfer function ϕ yields an optimization problem that depends on $\langle \phi(x_i) \cdot \phi(x_j) \rangle$. Kernel functions $K(x_i, x_j)$ are special functions that provide an equivalent way to calculate $\phi(x_i) \cdot \phi(x_j)$ (ie $\phi(x_i) \cdot \phi(x_j) = K(x_i, x_j)$) without having to initially transform the original dataset using ϕ since transforming the dataset to the feature space may be a costly operation if there are many features. The following are popular kernels [84]:

1. Linear Kernel: $K(x_i, x_j) = x_i \cdot x_j$
2. Polynomial Kernel: $K(x_i, x_j) = (1 + x_i \cdot x_j)^p$
3. Gaussian Kernel: $K(x_i, x_j) = e^{-\frac{\left\|x_i - x_j\right\|^2}{2\sigma^2}}$
4. RBF Kernel: $K(x_i, x_j) = e^{-\gamma(x_i - x_j)^2}$
5. Sigmoid Kernel: $K(x_i, x_j) = tanh(\eta x_i \cdot x_j + v)$

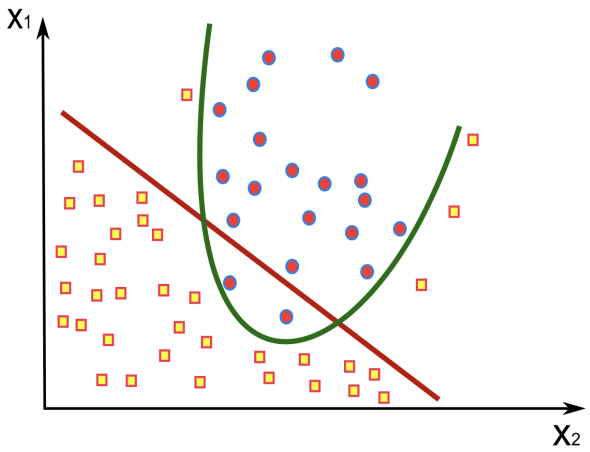


Figure 4.3: SVM non-linear classifier (green line) [84].

Since SVM can only separate between 2 classes as a binary classifier, a further step must be done to separate multiple classes. Scikit-learn uses a one vs. rest strategy by default. For each class a decision boundary between itself and the rest of the samples are created. For an unlabeled point a probability of membership to each class is calculated and an argmax function is used to determine the membership [85].

Theory of SVM: Linearly Separable

Consider a training dataset $X = \{x_i, y_i\}_{i=1}^{n}$ consisting of n samples of feature vectors $x_i \in \mathbb{R}$ and labels $y_i \in (-1, 1)$. In the linearly separable case of SVM, the two classes in our training dataset can be separated perfectly by a hyperplane. The data labelled with class $y = 1$ will be on one side of the hyperplane and the other class $y = -1$ will be on the other side of the hyperplane. The equation for a hyperplane is given in Equation 4.5

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{4.5}$$

Where $x \in \mathbb{R}^d$, and $\mathbf{w}$ the weights for the hyperplane and b the bias. For simplicity of visualization, consider $x \in \mathbb{R}^2$. The SVM optimization problem tries to find $\mathbf{w}$ and b such that the margin (ie. the distance to the closest point(s) to the hyperplane) is maximized. One variation of this margin called the functional margin F uses the y_i label to ensure that the margin is positive for individual samples correctly classified by some hyperplane with parameters ($\mathbf{w}$, b) and is given by the Equation 4.6

$$F = \min_{i=1\ldots n} [y_i(\mathbf{w} \cdot x_i + b)] \tag{4.6}$$

In the SVM optimization problem, this functional margin is set to $F = 1$ as shown in 4.1 and results in the parallel hyperplanes H1 and H2 as the constraint of the problem (Eq. 4.7).

$$y_i(\mathbf{w} \cdot x_i + b) \geq 1 \; \forall \; i \tag{4.7}$$

The goal now is to maximize the distance between the hyperplanes H1 and H2 to obtain the optimal hyperplane H which falls in the middle of H1 and H2. The minimum distance from H to H2 (or H to H1) is called the geometric margin. This minimum distance can be found be taking the vector between any point on H2: p_{H2} and any point on H1: p_{H1} and projecting it onto the unit normal vector of H1 or H2 given by $\mathbf{w}$. Keep in mind that p_{H2} must satisfy $\mathbf{w} \cdot \mathbf{x} + b = 1$ and p_{H1} must satisfy $\mathbf{w} \cdot \mathbf{x} + b = -1$

$$
\begin{aligned}
d &= \|\text{proj}_{\mathbf{w}}(p_{H2} - p_{H1})\| \\
&= \left\| \left((p_{H2} - p_{H1}) \cdot \frac{\mathbf{w}}{||\mathbf{w}||} \right) \frac{\mathbf{w}}{||\mathbf{w}||} \right\| \\
&= \left\| (p_{H2} \cdot \mathbf{w} - p_{H1} \cdot \mathbf{w}) \frac{\mathbf{w}}{||\mathbf{w}||^2} \right\|
\end{aligned}
$$

p_{H2} must satisfy $\mathbf{w} \cdot p_{H2} + b = 1$ (ie $\mathbf{w} \cdot p_{H2} = b - 1$) to be on hyperplane H2 and p_{H1} must satisfy $\mathbf{w} \cdot p_{H1} + b = -1$ to be on hyperplane H1 (ie $\mathbf{w} \cdot p_{H1} = -b - 1$)

$$
\begin{aligned}
&= \left\| ((1 - b) - (-b - 1)) \frac{\mathbf{w}}{||\mathbf{w}||^2} \right\| \\
&= \left\| 2 \frac{\mathbf{w}}{||\mathbf{w}||^2} \right\| \\
d &= \frac{2}{||\mathbf{w}||}
\end{aligned}
\tag{4.8}
$$

To find the optimal hyperplane, the distance $d = 2/||\mathbf{w}||$ must be maximized or equivalently $||\mathbf{w}||^2$ (which is a convex function [84]) can be minimized giving rise to the following convex optimization problem that can be solved using Lagrange Duality. The primal problem is:

$$
\begin{aligned}
&\min ||\mathbf{w}||^2 \\
&\text{s.t. } y_i(\mathbf{w} \cdot x_i + b) \geq 1 \; \forall \; i
\end{aligned}
\tag{4.9}
$$

Converting equation 4.9 to the Lagrange Formulation which causes the constraint to move to the objective function and act as a penalty if the constraint is violated:

$$
L(\mathbf{w}, b, \alpha) = \frac{1}{2} \langle \mathbf{w} \cdot \mathbf{w} \rangle - \sum_{i=1}^{n} \alpha_i [y_i (\langle \mathbf{w} \cdot x_i \rangle + b) - 1] \tag{4.10}
$$

Setting partial derivatives with respect to $\mathbf{w}$ and b equal to zero to minimize the Lagrangian:

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \rightarrow \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i x_i$$
$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{4.11}$$

Substituting these into the Lagrange Formulation and simplifying yields the dual that depends only on the Lagrange Multipliers α [84].

$$L(\mathbf{w}, b, \alpha) = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i y_i \alpha_j y_j \langle x_i \cdot x_j \rangle + \sum_{i=1}^{n} \alpha_i \tag{4.12}$$

Then the dual optimization problem is shown in Equation 4.13 and the solution to the dual is the same as the solution to the primal given that the Karush-Kuhn-Tucker conditions (KKT) are satisfied [84]:

$$\begin{aligned} &\max_{\alpha_i} -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i y_i \alpha_j y_j \langle x_i \cdot x_j \rangle + \sum_{i=1}^{n} \alpha_i \\ &\text{s.t. } \alpha_i \geq 0,\ 1, ..., n \\ &\sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned} \tag{4.13}$$

The solution to the Lagrangian Multipliers α_i can be found using quadratic programming. It is observed that $\alpha_i > 0$ are called the support vectors and all other $\alpha_i = 0$ [84]. Once α_i is found, $\mathbf{w}$ can be found using the training data and α_i in Equation 4.11. Since the support vectors have been found and known to fall on hyperplane H1 and H2, and the normal $\mathbf{w}$ has been found, the optimal b can be found by isolating for b in H2: $(\mathbf{w} \cdot x_{H2}) + b = 1$ or H1: $(\mathbf{w} \cdot x_{H1}) + b = -1$ for a support vector that falls on H2 or H1 respectively.

Theory of SVM: Not Linearly Separable

The optimization problem for the not linearly separable case is very similar to the linearly separable case, only now there is the addition of a slack variables ζ_i to the constraint (see Figure 4.2). The optimization problem then becomes

(Equation 4.14) [84] with the squared sum of the slack variables modified by a weight C added to the original cost function:

$$\begin{aligned} &\min ||\mathbf{w}||^2 + C\sum_{i=1}^{n} \zeta_i^2 \\ &\text{s.t. } y_i(\mathbf{w} \cdot x_i + b) \geq 1 - \zeta_i \; \forall \; i \\ &\zeta_i \geq 0 \end{aligned} \tag{4.14}$$

A higher C increases the cost of permitting slack variables and therefore a higher C tends toward perfectly separating the data. The problem is solved in a similar way using Lagrangian Duality and details can be found in Cervantes et al.'s article [84]

Tunable Parameters

In SVM the tunable parameters are C, the type of kernel selected and the kernel's tunable parameter(s). In Section 4.1.2, it was discussed that the larger the C the larger the penalty on slack variables and therefore the classifier tends toward perfect separation of the data points. It doesn't seem like there is agreement on what kernel and what kernel parameter is appropriate for a specific application [84]. However, in practice it is common for datasets with many features to use the basic linear kernel [84]. If the data requires non-linear separation, the Gaussian Kernel is most widely used and its parameter σ can be appropriately chosen using a search [84].

4.1.3 Random Forests

Overview

Random Forests consist of Decision Trees that must be first discussed [86]. A decision tree can be built upon features that are categorical (eg. has astigmatism) or numerical (eg. age). Training a decision tree results in a the structure shown in Figure 4.4 with nodes representing the feature and edges representing the options or ranges of values that the feature can take. Typically, the classifier starts at a starting point called the root node. Edges extend uni-directional from the root node presents a decision that the classifier must make based on the features of the new data point. Picking an option or range of values (choosing one of the edges extending from the node) advances the classifier along that edge to the next node in the next level of the tree. The classifier repeats this decision process until there is no more next node; the classifier has reached a "leaf" node and can classify the data point.

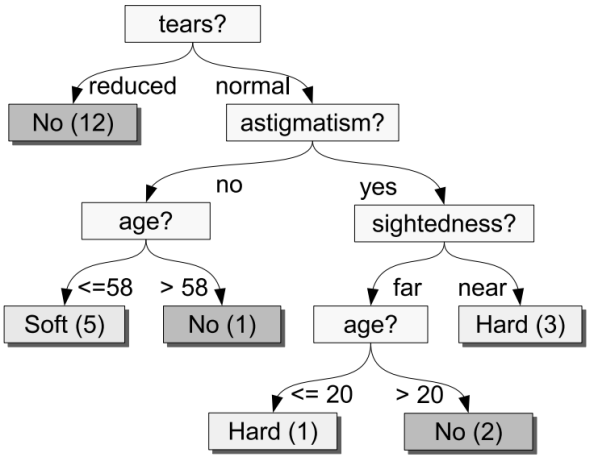
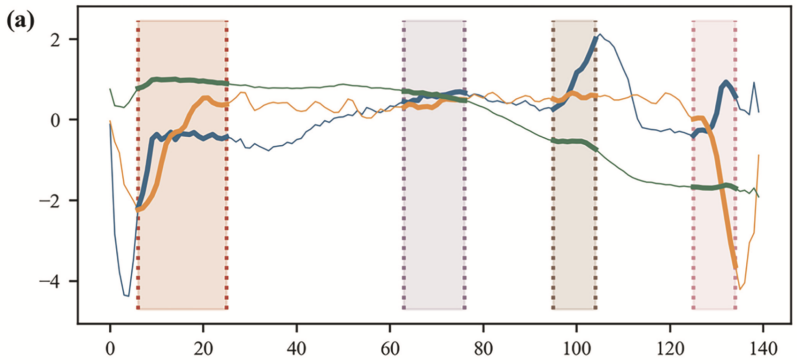


Figure 4.4: Example of a Decision Tree [87]

Random Forests extend the use of Decision Trees by training multiple trees, classifying the new data with each tree and doing a majority vote on the output of each decision tree [88]. Each Decision Tree is trained using a random subset of data drawn from the original training data with replacement. These trees are grown to maximum depth without pruning using the

Classification and Regression Tree (CART) methodology and at each node a random subset of features is used [88]. In situations where there are many features with each feature providing only a small amount of information (eg. medical diagnosis), the single decision tree will have an accuracy slightly better than random classification, whereas using a Random Forest can produce improved accuracy [88].

In time series data, features can be extracted by taking the mean, standard deviation and slope of random subsequences in sequence labelled a certain class 4.5. The number of features is then 3 * the number of subsequences extracted. These features are used to train a random forest and new sequences of time series data can be classified by following the same methodology for feature extraction [83].



(b)

	Interval 1			Interval 2			Interval 3			Interval 4		
	Mean	SD	Slope	Mean	SD	Slope	Mean	SD	Slope	Mean	SD	Slope
Time series 1	-0.61	0.504	0.052	0.58	0.086	0.02	1.004	0.572	0.197	0.189	0.504	0.156
Time series 2	-0.467	0.977	0.158	0.403	0.084	0.017	0.568	0.061	0.011	-1.204	1.305	-0.431
Time series 3	0.944	0.061	0.002	0.604	0.075	-0.018	-0.57	0.065	-0.017	-1.671	0.024	0.003

Figure 4.5: Decision Trees used for time-series classification. Mean, standard deviation and slope of random subsequences in a time-series sequence defining a class are used as features. [83]

Classification and Regression Tree (CART) Methodology

The Classification and Regression Tree (CART) method to grow trees outlined by Breiman et al. has the objectivet of splitting the dataset L at each node such that the descendent nodes–nodes one level down–are "purer" than

the parent node. Purity in sklearn's implementation of decision trees can be evaluated using Gini's Impurity and Entropy [89] and the most common, Gini's Impurity, is shown in Equation 4.15 [89].

$$\mathrm{G} = \sum_{k} p_{mk}(1 - p_{mk}) \tag{4.15}$$

Where p_{mk} is the proportion or probability of observing a class k in subset Q_m

$$p_{mk} = \frac{1}{n_m} \sum_{y \in Q_m} I(y = k) \tag{4.16}$$

At some node m, some subset of the data at node m designated as Q_m, and k classes. At each node, candidate splits $\theta = (j, t_m)$ consisting of a feature j and threshold t_m are evaluated for impurity. According to the source code for sklearn, these candidate splits are chosen by randomly selecting a feature, sorting the feature's values, and using each of the feature's values as a threshold t_m for a candidate split (ie. the thresholds t_m are the values of feature j for the datapoints in the subset Q_m). This process is repeated until the max number of features to be evaluated (an input argument in the Decision Tree Classifier) is reached. The candidate split θ that minimizes the (Gini) impurity is selected to split the data in a binary fashion with 2 descendant nodes [90]. The same process repeats at the descendent nodes recursively until a termination condition (eg. max depth of tree) is reached or the descendant node only contains a single class.

Tunable Parameters

The tunable parameters for Random Forests are similar to Decision Trees and the full list may be found on sklearn's Documentation page for Random Forests [91]. Sklearn mentions that the default values for controlling the size of the tree (eg. max_depth, min_samples_leaf) leads to large, unpruned and fully grown trees which may be very memory intensive. They recommend to control these parameters to limit the memory consumption [91]. Otherwise, similar to other ML methods, a grid search should be used to find the optimal hyperparameters for the training dataset [92]. Parameters that should be considered for tuning in order of highest positive effect on the Area Under the Curve (AUC) is mtry (max_features in sklearn), sample size drawn for

training a tree, and the minimum number of observations in a terminal node (min_samples_leaf in sklearn) [92].

4.1.4 Shapelet Transform

Overview

A shapelet is a small but discriminating subsection of a time series waveform that represents a class [93]. To classify with shapelets, the authors in [93] propose using a decision tree with split criteria based on the inclusion shapelets from a shapelet dictionary (inclusion is determined by the distance between a subsequence from a new time-series signal and the shapelet being less than some threshold–Figure 4.6 shows that the inclusion threshold distance is 5.1). Shapelets address some of the shortcomings of using kNN-DTW presented in Section 4.1.1 that result in large time and space complexity and limit its applicability.

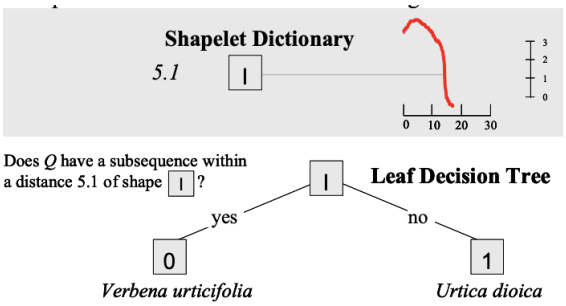


Figure 4.6: Example of a shapelet and the criteria for classifying a new time-series [93].

Determining Shapelets

The authors in [93] mention that the total number of possible shapelets can be calculated by the following equation

$$\sum_{l=MINLEN}^{MAXLEN} \sum_{T_i \in D} (m_i - l + 1) \tag{4.17}$$

for every timeseries T_i in dataset D, l is the length of a candidate shapelet and m_i is the length of the i-th timeseries in the dataset D. For the Trace

dataset with has 200 instances, each with length of 275. If $MINLEN = 3$ and $MAXLEN = 275$, the total number of candidates would be 7,480,200. A brute force approach (Algorithm 1) would be inefficient at finding the best shapelet. The time complexity is $O(\bar{m}^3 k^2)$ for average length of timeseries $\bar{m}$, so the authors propose a pruning strategy to find the best shapelets for classification.

Algorithm 1 Brute Force approach for finding the best shapelet [93], bsf_gain is the information gain calculated using Entropy.

```
candidates ← GenerateCandidates(D, MAXLEN, MINLEN)
bsf_gain ← 0
for each S in candidates do
    gain ← CheckCandidates(D, S)
    if gain > bsf_gain then
        bsf_gain ← gain
        bsf_shapelet ← S
    end if
end for
return bsf_shapelet
```

To speed up the calculations, the authors use the *early abandon* method in the CheckCandidates$(D, \mathcal{S})$ function. Instead of calculating the entire Euclidean distance between the shapelet and the subsection of the time series, a minimum distance variable (initialized to infinity) is used to track the minimum distance calculated. If, at any point in the Euclidean distance calculation, the minimum distance is exceeded, the calculation is stopped and moves onto the next step. If the full distance between the shapelet and the subsection of the timeseries is less than the minimum distance in memory, then the minimum distance is updated [93].

Another optimization involves *entropy pruning* which occurs once again at the CheckCandidates$(D, \mathcal{S})$ function. Another variable is created to store the best so far information gain. Since the minimum distance calculation between the timeseries and the candidate shapelet is the most costly, these distances are calculated one by one. At each distance calculation, the optimal split point is determined for the distances already calculated and the information gain in the most ideal situation (the rest of Class A on one side and the rest of Class B on the other) is calculated. If the information gain in

the most ideal situation is less than the best so far gain, then the candidate shapelet can be pruned and no further minimum distance calculations will need to be done. Otherwise, continue calculating the minimum distances between the timeseries and the candidate shapelet and checking the information gain for the ideal situation at each step. If after calculating all of the distances between the candidate shapelet and the the timeseries dataset and the information gain is still greater than the best so far information gain, then the best so far gain is updated and the process continues until all of the candidate shapelets have been considered [93].

Multi-Class Classification with Shapelets

To classify multiple classes, the authors of [93] proposed growing a decision tree. At each node a shapelet is found using the optimized Algorithm 1 outlined in the previous subsection. The split point identified using the shapelet at the node splits the dataset into a left and right subset of data. On the left and right subset of data, the optimized Algorithm 1 is used to find the shapelet and the split point for that subset of data. The process continues recursively until a leaf node is reached. This process creates a decision tree and a dictionary of shapelets shown in Figure 4.7 that can be used for classifying time-series data.

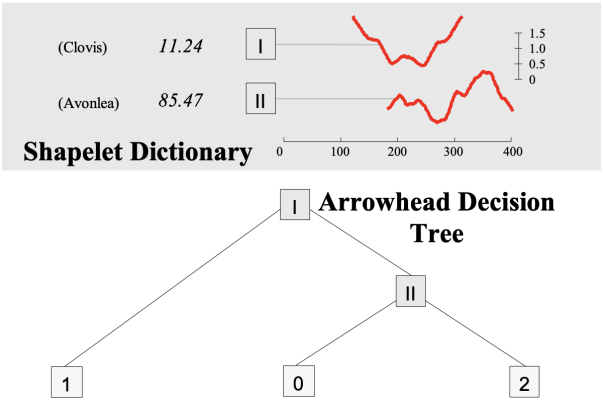


Figure 4.7: An example of a shapelet based decision tree [93].

4.2 Neural Networks

4.2.1 Overview

Neural networks are loosely based on the biological structure of the brain; it consists of nodes called neurons which are connected to each other. Some input x passes through the trained layers of neurons and produces an output $f(x, \theta)$ that can be used in applications such as classification (single point output) and forecasting (vector output) [94]. An example of a neural network is shown in Figure 4.8. Each neuron is a simple function consisting of a weighted sum of the incoming signal added to some bias b, and the result z is passed to an activation function $\sigma(z)$ which determines if the neuron *fires* or not. A very simple example of an activation function called the perceptron is shown in Equation 4.18. Note that due to the loss of information in the perceptron (ie. no information on the strength of activation), however, it is not used in neural networks found in literature [94].

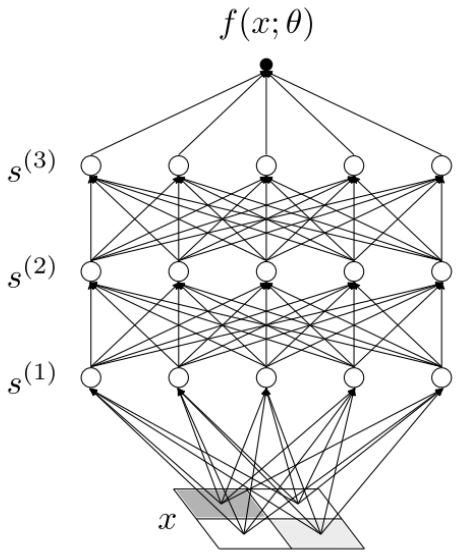


Figure 4.8: A neural network with input x and output $f(x, \theta)$. The input is transformed at each intermediate layer $s^{(1)}, s^{(2)}, s^{(3)}$ until it reaches the output. [94].

$$\sigma(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 1 \end{cases} \tag{4.18}$$

There are also different types of layers depending on the type of input x and the application of the neural network. The following subsections will briefly discuss these different types of layers used in their respective neural network. These layers include the linear layers, convolutional layers, recurrent layers, long short term memory (LSTM) layers and transformer layers.

4.2.2 Deep Neural Networks (DNN)

The Neuron

Deep Neural Networks consist of many stacked layers of neurons [94]. The threshold for considering a neural network "deep" is 3 layers [95]. As mentioned in the Overview, a neuron consists of the weighted sum of the incoming signal x added to some bias b, Equation 4.19. The weights for each layer are stored in a weight matrix W.

$$z_i(x) = b_i + \sum_{j=1}^{n_{in}} W_{ij} x_j \text{ for } i = 1, ..., n_{out} \tag{4.19}$$

Each i-th output goes through an activation function $\sigma_i = \sigma(z_i)$ to determine whether the neuron "fires" to the i-th one in the next layer [94]. Examples of these activation functions can be seen in Figure 4.9.

The inputs and outputs of each layer can then be modeled as follows (Equation 4.20) with superscripts indicating the layer number [94]:

$$\begin{aligned} z_i^{(1)}(x_\alpha) &= b_i^{(1)} + \sum_{j=1}^{n_0} W_{ij}^{(1)} x_{j;\alpha} \text{ for } i = 1, ..., n_1 \\ z_i^{(l+1)}(x_\alpha) &= b_i^{(l+1)} + \sum_{j=1}^{n_l} W_{ij}^{(l+1)} \sigma(z_j^{(l)}(x_\alpha)) \text{ for } i = 1, ..., n_{l+1},\ l = 1, ..., L-1 \end{aligned} \tag{4.20}$$

Where L is the total number of layers (the depth of the neural network), and $n_{l=1,..,L-1}$ is the number of outputs (the width) at layer l. n_0 and n_L

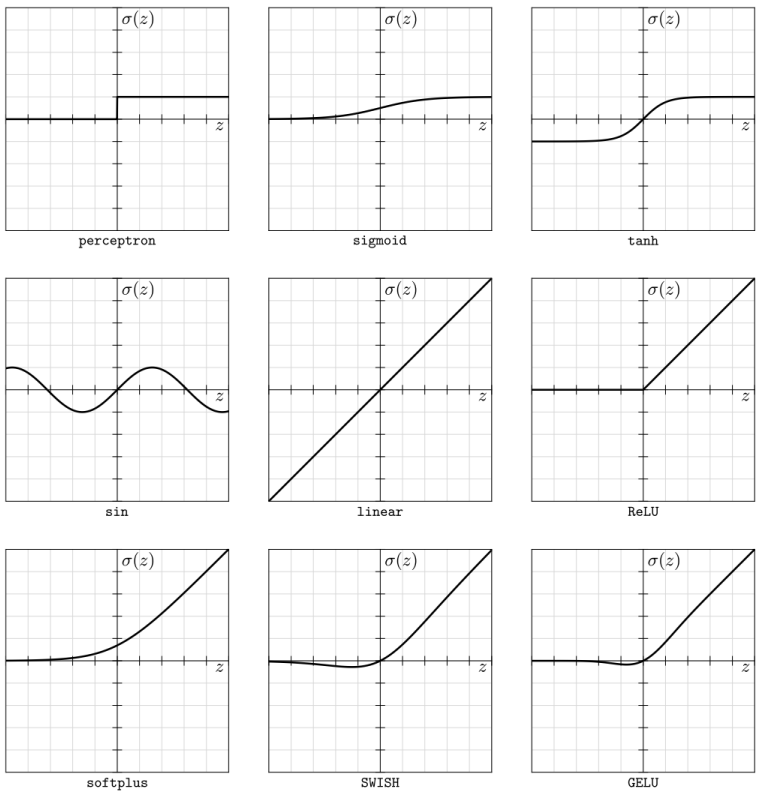


Figure 4.9: Examples of some activation functions [94].

are the input and output dimensions of the neural network respectively [94]. Figure 4.10 depicts the inputs and outputs of a single neuron.

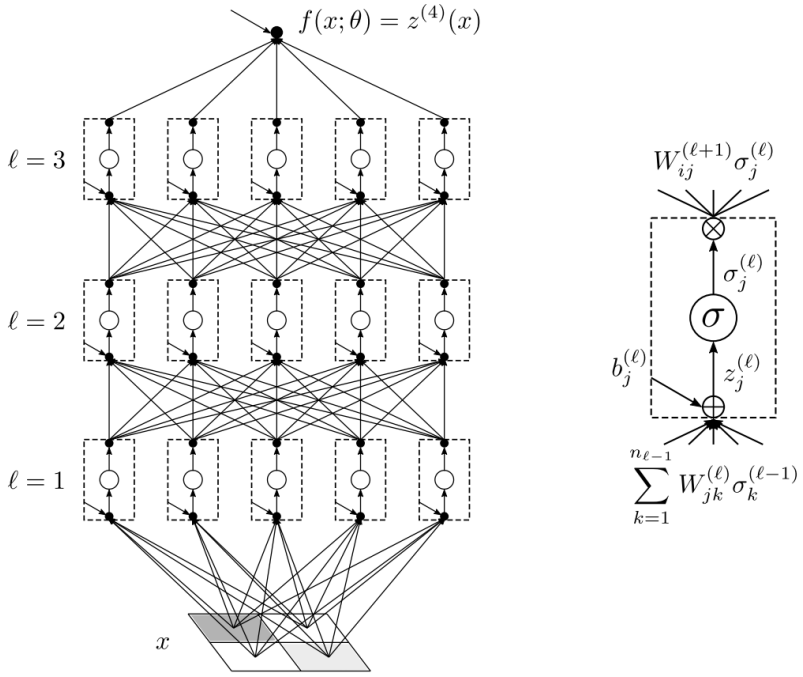


Figure 4.10: Visual representation of the equation of a neuron (Equation 4.20) [94]. The image on the right shows the inputs to the neuron and its corresponding outputs.

Training the Neural Network: Gradient Descent

Training a neural network involves repeatedly updating the weights W_{ij} and biases b_i of the model using a gradient-based method such as gradient descent. Gradient-based methods optimize (minimize) a *loss* function that calculates the error between the output layer result $z^{(L)}(x)$ and the human-annotated labels y; the goal of the optimization of the loss function is to decrease the error between the output layer result and the labels. Though the choice of loss function depends on the application, an example of a loss function $\mathcal{L}$ can mean-squared error (MSE) shown in Equation 4.21[1] [94].

$$\mathcal{L}_{MSE}(z(x_\delta;\theta), y_\delta) = \frac{1}{2}\left[z(x_\delta;\theta) - y_\delta\right]^2 \tag{4.21}$$

With the objective of minimizing the loss in mind, training a simple neural network may have the following steps [94]:

1. Model parameters $W_{ij}^{(l)}$ and $b_i^{(l)}$ (collectively referred to as θ hereon) initialized by independently sampling from an easy to sample distribution such as a zero-mean Gaussian Distribution with variances

$$\begin{aligned}\mathbb{E}[b_{i_1}^{(l)} b_{i_2}^{(l)}] &= \delta_{i_1 i_2} C_b^{(l)} \\ \mathbb{E}[W_{i_1 j_1}^{(l)} W_{i_2 j_2}^{(l)}] &= \delta_{i_1 i_2}\delta_{j_1 j_2}\frac{C_W^{(l)}}{n_l - 1}\end{aligned} \tag{4.22}$$

   With variances C and the above presentation uses the Kronecker Delta $\delta_{i_1 i_2}$ (Equation 4.23) to show that the values for each weight and each bias are drawn independently from each other.

$$\delta_{i_1 i_2} = \begin{cases} 1 & \text{if } i_1 = i_2 \\ 0 & \text{if } i_1 \neq i_2 \end{cases} \tag{4.23}$$

2. Using the initial weights and biases, the data from the training dataset $\mathcal{A}$ is passed through the neural network using Equation 4.20 which predicts the last layer $z^{(L)}(x_\alpha;\theta)_{\alpha\in\mathcal{A}}$. The loss is then calculated as the

[1] Note that the subscript δ in $(z(x_\delta;\theta), y_\delta)$ is a placeholder to show the δ-th data point in a certain dataset that x and y are from (ie. training dataset $\mathcal{A}$ has data points $(x_\alpha, y_\alpha)_{\alpha\in\mathcal{A}}$ or testing dataset $\mathcal{B}$ has data points $(x_\beta, y_\beta)_{\beta\in\mathcal{B}}$)

average of the loss for each training sample in the training dataset $\mathcal{A}$ and is shown in Equation 4.24

$$\mathcal{L}_{\mathcal{A}}(\theta) \equiv \frac{1}{|\mathcal{A}|} \sum_{\alpha \in \mathcal{A}} \mathcal{L}\left(z^{(L)}(x_\alpha; \theta), y_\alpha\right) \tag{4.24}$$

Also note that there can be multiple neurons in the output layer which the label y_α should match in size. The losses from each i-th output neuron with the corresponding i-th portion of the label for the data point α are summed together. In other words:

$$\mathcal{L}_{\mathcal{A}}(\theta) \equiv \frac{1}{|\mathcal{A}|} \sum_{\alpha \in \mathcal{A}} \sum_{i=1}^{n_L} \mathcal{L}\left(z_i^{(L)}(x_\alpha; \theta), y_{i;\alpha}\right) \tag{4.25}$$

3. The gradient of this loss is calculated with respect to each model parameter θ_μ and is used to iteratively update the model parameters in the negative direction of the gradient[2]

$$\theta_\mu(t+1) = \theta_\mu(t) - \eta \left.\frac{d\mathcal{L}_{\mathcal{A}}}{d\theta_\mu}\right|_{\theta_\mu = \theta_\mu(t)} \tag{4.26}$$

Where η is a positive hyperparameter called the learning rate and dictates the size of the step in the direction of the negative gradient, μ is added as a subscript to account for all parameters ($[W_{ij}^{(1)}, b_i^{(1)}, ..., W_{ij}^{(L)}, b_i^{(L)}]$), t is the number of steps in the iterative training process usually starting at $t = 0$.

The derivative $d\mathcal{L}_{\mathcal{A}}/d\theta_\mu$ is done iteratively starting at the last layer and propagating the results backwards layer by layer. This step is called

[2]While Equation 4.26 guarantees at least a local minimum with a small learning rate η. A popular variant of gradient descent called **Stochastic gradient descent** (SGD) uses a random subset of the training data $\mathcal{S}_t \in \mathcal{A}$. $\mathcal{S}_t$ is called a batch and training is organized into "epochs" which is a complete pass through the training dataset. Using SGD has the advantage of better generalization and scalability since training time scales with a fixed size $\mathcal{S}_t$ instead of the entire training set $\mathcal{A}$ (since Equation 4.26 requires the calculation of the gradient for all data points in the data set used, the computation scales linearly with the set used. Assuming that the gradient can be approximated with $\mathcal{S}_t$, computing the gradient with a smaller subset $\mathcal{S}_t$ is faster than the full $\mathcal{A}$ because there are less gradients to calculate).

backpropagation and using the chain rule, it is shown generalized for any parameter θ_μ in Equation 4.27[3]:

$$\frac{d\mathcal{L}_\mathcal{A}}{d\theta_\mu} = \sum_{i=1}^{n_L} \sum_{\alpha \in \mathcal{A}} \frac{\partial \mathcal{L}_\mathcal{A}}{\partial z_{i;\alpha}^{(L)}} \frac{dz_{i;\alpha}^{(L)}}{d\theta_\mu} \tag{4.27}$$

Starting at the last layer L and using the definition of $z_i^{(l+1)}$ in Equation 4.20, parameters associated with the m-th output neuron $m = 1, ..., n_L$ can be calculated as[4]:

$$\begin{aligned} \frac{d\mathcal{L}_\mathcal{A}}{db_m^{(L)}} &= \sum_{i=1}^{n_L} \sum_{\alpha \in \mathcal{A}} \frac{\partial \mathcal{L}_\mathcal{A}}{\partial z_{i;\alpha}^{(L)}} \frac{dz_{i;\alpha}^{(L)}}{db_m^{(L)}} = \sum_{\alpha \in \mathcal{A}} \frac{\partial \mathcal{L}_\mathcal{A}}{\partial z_{m;\alpha}^{(L)}} (1) \\ \frac{d\mathcal{L}_\mathcal{A}}{dW_{mj}^{(L)}} &= \sum_{i=1}^{n_L} \sum_{\alpha \in \mathcal{A}} \frac{\partial \mathcal{L}_\mathcal{A}}{\partial z_{i;\alpha}^{(L)}} \frac{dz_{i;\alpha}^{(L)}}{dW_{mj}^{(L)}} = \sum_{\alpha \in \mathcal{A}} \frac{\partial \mathcal{L}_\mathcal{A}}{\partial z_{m;\alpha}^{(L)}} \sigma_j^{(L-1)} \end{aligned} \tag{4.28}$$

To get the parameters in the subsequent layer $L-1$, the chain rule is once again used:

$$\begin{aligned} \frac{d\mathcal{L}_\mathcal{A}}{db_j^{(L-1)}} &= \sum_{i=1}^{n_L} \sum_{\alpha \in \mathcal{A}} \frac{\partial \mathcal{L}_\mathcal{A}}{\partial z_{i;\alpha}^{(L)}} \frac{dz_{i;\alpha}^{(L)}}{d\sigma_j^{(L-1)}} \frac{d\sigma_j^{(L-1)}}{dz_{j;\alpha}^{(L-1)}} \frac{dz_{j;\alpha}^{(L-1)}}{db_j^{(L-1)}} \\ &= \sum_{i=1}^{n_L} \sum_{\alpha \in \mathcal{A}} \frac{\partial \mathcal{L}_\mathcal{A}}{\partial z_{i;\alpha}^{(L)}} W_{ij}^{(L)} \sigma_j'^{(L-1)} (1) \\ \frac{d\mathcal{L}_\mathcal{A}}{dW_{jk}^{(L-1)}} &= \sum_{i=1}^{n_L} \sum_{\alpha \in \mathcal{A}} \frac{\partial \mathcal{L}_\mathcal{A}}{\partial z_{i;\alpha}^{(L)}} \frac{dz_{i;\alpha}^{(L)}}{d\sigma_j^{(L-1)}} \frac{d\sigma_j^{(L-1)}}{dz_{j;\alpha}^{(L-1)}} \frac{dz_{j;\alpha}^{(L-1)}}{dW_{jk}^{(L-1)}} \\ &= \sum_{i=1}^{n_L} \sum_{\alpha \in \mathcal{A}} \frac{\partial \mathcal{L}_\mathcal{A}}{\partial z_{i;\alpha}^{(L)}} W_{ij}^{(L)} \sigma_j'^{(L-1)} \sigma_k^{(L-2)} \end{aligned} \tag{4.29}$$

[3]Note that the $dz_{i;\alpha}^{(L)}/d\theta_\mu$ term evaluates to zero if the i-th $z_{i;\alpha}^{(L)}$ doesn't depend on the parameter θ_μ.

[4]note that the sum from i to n_L vanishes because $dz_{i;\alpha}^{(L)}/\theta_\mu^{(L)}$ will evaluate to zero if $i \neq m$

More generally, starting from Equation 4.27, the following Equations 4.30 may be used iteratively by sequential multiplication of the chain rule factors to determine the second term in Equation 4.27 (ie. sensitivity of the output in layer $z_{i;\alpha}^{(L)}$ to parameter θ_μ: $dz_{i;\alpha}^{(L)}/d\theta_\mu$) in the l-th layer $l = L-1, ..., 1$ (once again working backwards from layer L):

$$\begin{aligned}
\frac{dz_{i;\alpha}^{(L)}}{db_j^{(l)}} &= \frac{dz_{i;\alpha}^{(L)}}{dz_{j;\alpha}^{(l)}} \\
\frac{dz_{i;\alpha}^{(L)}}{dW_{jk}^{(l)}} &= \sum_m \frac{dz_{i;\alpha}^{(L)}}{dz_{m;\alpha}^{(l)}} \frac{dz_{m;\alpha}^{(L)}}{dW_{jk}^{(l)}} = \frac{dz_{i;\alpha}^{(L)}}{dz_{j;\alpha}^{(l)}} \sigma_{k;\alpha}^{(l-1)} \\
\frac{dz_{i;\alpha}^{(L)}}{dz_{j;\alpha}^{(l)}} &= \sum_{k=1}^{n_{l+1}} \frac{dz_{i;\alpha}^{(L)}}{dz_{k;\alpha}^{(l+1)}} W_{kj}^{(l+1)} \sigma_{j;\alpha}^{\prime(l)} \quad \text{for} \quad l < L
\end{aligned} \tag{4.30}$$

As a rule of thumb, The change in the loss function $\mathcal{L}_\mathcal{A}$ with respect to some parameter $\theta_\mu^{(l)}$ in layer $l < L$ will depend on all of the neurons in the layer above $l+1$ (causing the summations to stack as the change in the loss function with parameters in lower layers are calculated).

Equation 4.26 is run until there is convergence at a least a local minimum or the number of iterations specified is completed [94].

After the model parameters have been trained, the forward equation (Equation 4.20) can be used on new input data to classify, predict or forecast whatever the model was trained to do.

4.2.3 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) are typically used for inputs that are in the shape of a 2D arrays with multiple channels (ie. the depth of the input) such as a color image with RBG channels. They are primarily used in image recognition and classification tasks [96]. Unlike the DNN in Section 4.2.2 which requires a set of features to input into the model, CNNs extract the features from images by themselves during the training and do not require the user to determine a set of features. Typically, CNNs are composed of 3 layers [96, 97]:

- Convolutional
- Pooling
- Fully Connected (discussed in 4.2.2)

A typical architecture for a CNN is outlined in Figure 4.11.

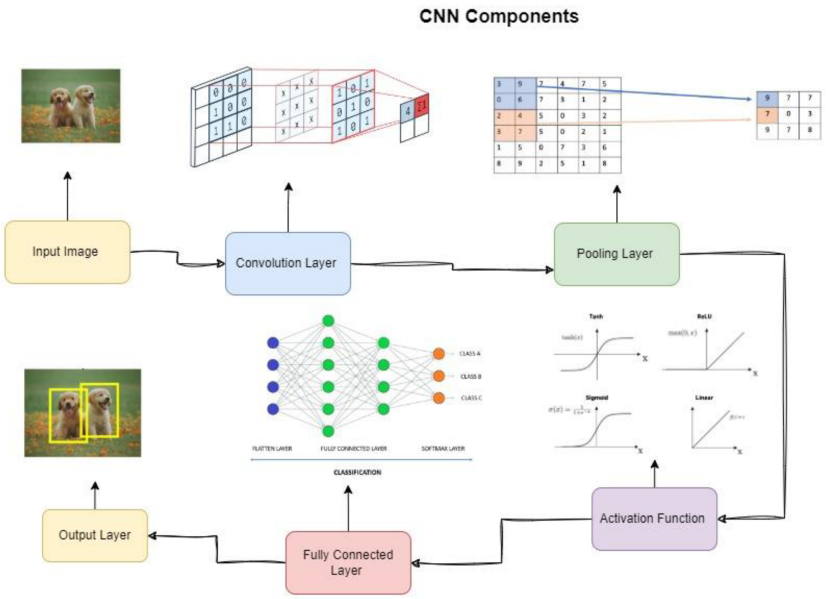


Figure 4.11: Architecture of a common CNN [96].

Convolutional Layer

The convolutional layer involves applying a set of **filters** or **kernels** to the data before it is used (the user can specify the number of kernels k in each layer). These kernels are small 3D arrays whose height and width F are typically smaller than the input array's but have the same depth r as the input array. The kernel's values are called **weights** and are trainable.

Initially, the weights are initialized at random, but as the training progresses, the weights change to enable the kernel to extract meaningful features [96]. In the forward pass, the kernel transforms the image through the dot product between the kernel and the section of the input array. The kernel starts at the top left of the input array and slides left to right, top to bottom calculating the dot product between itself and the section of the input array to produce a single value at each step. How much the filter slides is controlled by the **stride** parameter (ie. a stride of 1 means that the filter will slide to the right 1 element every step where as a stride of 2 means that the filter will stride to the right 2 elements every step). Figure 4.12 shows the result of applying a 2x2 kernel on a 4x4 input with a stride of 1.

As seen in Figure 4.12 the output height and width was less than the input array's height and width. To avoid the shrinkage in size after applying the kernel, the input array can be zero padded P number of layers. Taye's [96] formula to calculate the expected output dimension's height and width O after applying a kernel with height and width F, input array height and width N, and stride S is shown in Equation 4.31

$$O = 1 + \frac{N + 2P - F}{S} \tag{4.31}$$

The convolutional operation from one kernel will produce an array with shape (O,O,1). For k number of kernels that the user defines, the final output shape will be (O,O,k).

After the convolution operation is complete, a bias b is added to the output from each kernel and the values are typically passed through the ReLU activation function σ to remove the negative values[5] [96, 98]. In other words, the output $z^{k,(l)}$ at layer l for the k-th kernel (that is passed to the next layer) is as follows for some input x [99]:

$$z^{k,(l)} = \sigma(W^{k,(l)} * x + b^k) \tag{4.32}$$

[5]though a Leaky ReLU may be used to avoid the "Dying ReLU" issue

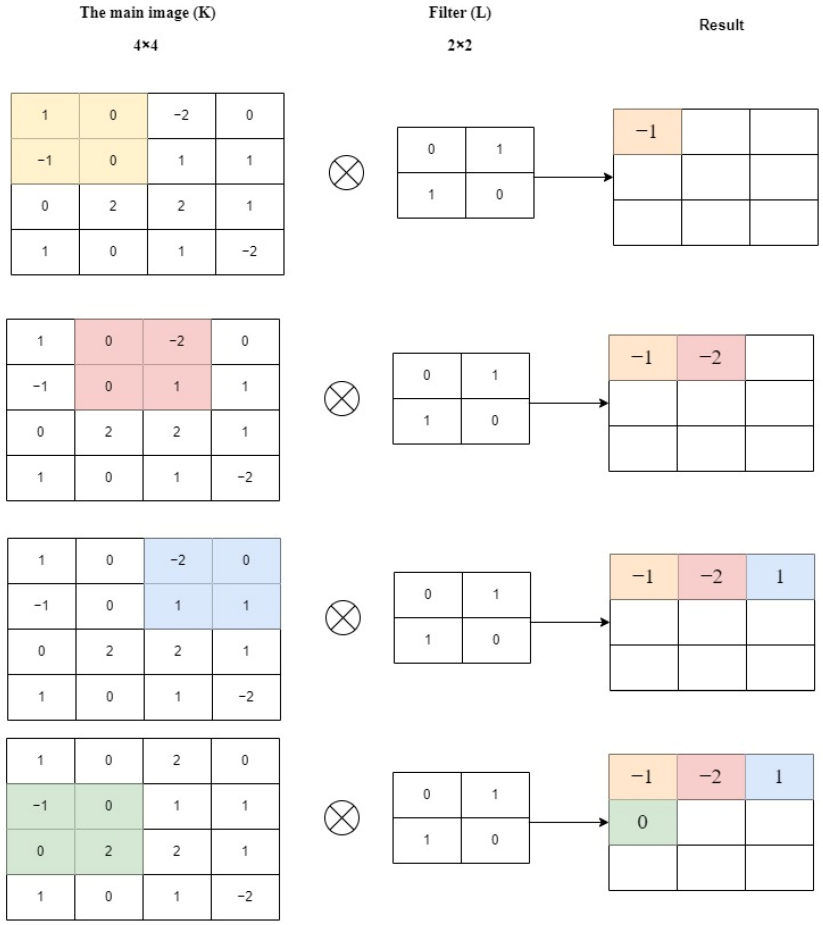


Figure 4.12: Applying a kernel (filter) to the input array [96].

Where $W^{k,(l)}$ are the weights of the k-th kernel, b^k is the bias added to the output of the k-th kernel, and the $*$ symbol represents the convolution operation.

Based on the information discussed on convolutions, given kernel height and width[6] F, the number of filters k and the depth of the input layer r, the number of trainable parameters in one convolutional layer is shown in Equation 4.33

$$\text{Number of Parameters} = ((F * F * r) + 1)k \tag{4.33}$$

[6]height and width of the kernel do not always need to be the same.

Pooling Layer

Pooling downsamples the output to decrease the height and width dimensionality while retaining the most important data. Similar to the idea of a filter or kernel in the previous section, the pooling filter can have the height and width specified (eg. 2x2) as well as the type (eg. max, min, avg). The pooling filter starts at the top left and slides left to right, top to bottom applying the pooling rule to section of the input to the pooling filter. For example, if the pooling type is a max, the maximum value in the section of the input array where the pooling filter is at is taken and becomes the new value in the shrunken output array. Typically, the pooling filter then slides with a stride equal to its dimensions: $S = F$. The pooling layer operates over all of the channels such that the height and width are reduced, but the depth remains the same [96]. Taye provides an example of Max Pooling and Average Pooling with a 2x2 filter size in Figure 4.13. Similar to the convolutional layer, the output height and width O can be calculated using Equation 4.31 with no padding (ie. $P = 0$) and the depth of the output is the same as the input.

Fully Connected Layer

In CNN architectures such as AlexNet and LeNet-5 [100] and the sample CNN architecture in pyTorch [101] the output from the last pooling layer is flattened into a 1D array and fed as an input to fully connected layers discussed in Section 4.2.2. From the literature, it seems that the ReLU activation function is the most popular for its simplicity and time and resource savings [96, 100, 97].

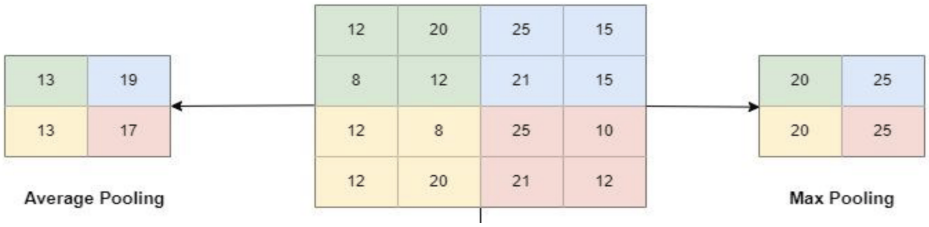


Figure 4.13: The pooling layer [96]

Training the Convolutional Layers

Similar to the training done in the DNN section, training the convolutional layers involve backpropagation and finding the gradient of the cost function with respect to all of the model's parameters. Then using gradient descent, the optimal parameters may be found. Recall Equation 4.26:

$$\theta_\mu(t+1) = \theta_\mu(t) - \eta \left. \frac{d\mathcal{L}_\mathcal{A}}{d\theta_\mu} \right|_{\theta_\mu = \theta_\mu(t)} \tag{4.34}$$

First backpropagation is done through the fully connected layers to find the $d\mathcal{L}_\mathcal{A}/d\theta_\mu$ for weights and biases in the fully connected layers. This has already been discussed in Section 4.2.2 through the use of the chain rule. Considering that a typical CNN architecture involves stacking of the Convolutional layers and the Pooling layers prior to the fully connected layers (refer to Figure 4.14 of AlexNet), calculating the weights and biases that are in the Convolutional layers requires backpropagation to work backwards through both the pooling and the convolutional layers. The treatment of the pooling and convolutional layers will be discussed next.

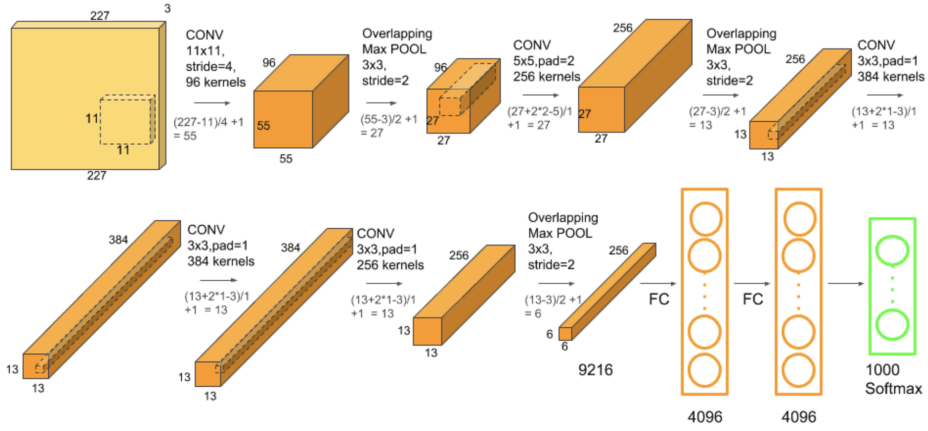


Figure 4.14: Architecture of the AlexNet [102].

Since there are no parameters in the pooling layer, we must look back to the convolutional layer prior to this pooling layer to see how the pooling layer affects the backpropagation. Each cell $z_{ij}^{k,(l)}$ in the output from the convolutional layer is dependent on all the weights in the k-th kernel. Using

ij subscripts to denote the position in the output 2D array, k to denote the k-th kernel, F as the width and height of the kernel, r the depth of the kernel and O the output height and width, Equation 4.32 expanded is shown in Equation 4.35 at layer l:

$$z_{ij}^{k,(l)} = \sigma\left(b^{k,(l)} + \sum_{m=1}^{F}\sum_{n=1}^{F}\sum_{d=1}^{r} W_{mnd}^{k,(l)} x_{[m+i-1][n+j-1][d]}\right) \quad \text{for} \quad i,j = 1,...,O \tag{4.35}$$

Since each output cell $z_{ij}^{k,(l)}$ from the k-th kernel depends on each weight in the k-th kernel, to calculate the sensitivity of the loss function to some weight parameter in the k-th kernel of convolutional layer l ($d\mathcal{L}_{\mathcal{A}}/dW_{abc}^{k,(l)}$), the sum of the partial derivatives of all the outputs from the convolution between the k-th kernel and the input with respect to some weight $W_{abc}^{k,(l)}$ must be calculated:

$$\frac{d\mathcal{L}_{\mathcal{A}}}{dW_{abc}^{k,(l)}} = \sum_{i=1}^{O}\sum_{j=1}^{O} \frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial z_{ij}^{k,(l)}} \frac{\partial z_{ij}^{k,(l)}}{\partial W_{abc}^{k,(l)}} \tag{4.36}$$

With the equations for sensitivity of loss function to the weights and bias in the convolutional layer, the effect of the Pooling layer can be investigated. As seen in AlexNet (Figure 4.14), there is a MaxPool layer applied to the output of the convolutional layer before being flattened and fed into the fully connected layers. In the forward pass, this pooling layer selects the "winning" cell (in this case the max value in the area selected by the pooling filter) before sliding to the next area. In terms of the output from the convolutional layer in Equation 4.35, only changes in the output $z_{ij}^{k,(l)}$ that have been selected as the "winning" cell by the pooling layer will have an effect on the loss $\mathcal{L}_{\mathcal{A}}$ [103]. Thus, in terms of Equation 4.36, rather calculate all $i,j = 1,...,O$, only the set of the winning cells C_{win} (or winning indices) are calculated:

$$\frac{d\mathcal{L}_{\mathcal{A}}}{dW_{abc}^{k,(l)}} = \sum_{ij \in C_{\text{win}}} \frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial z_{ij}^{k,(l)}} \frac{\partial z_{ij}^{k,(l)}}{\partial W_{abc}^{k,(l)}} \tag{4.37}$$

There might also be an Average Pooling layer which has a similar effect to adding a convolutional layer without any additional parameters. In each step, the pooling filter averages the section that it is on:

$$\text{POOL}_{ij}^{k} = \frac{1}{F \cdot F} \sum_{m=1}^{F} \sum_{n=1}^{F} x_{[m+i-1][n+j-1]}^{k} \quad \text{for} \quad i,j = 1,...,O, \quad k = 1,...,r \tag{4.38}$$

This pooling function would be added to the chain rule in Equation 4.36 to properly account for the effects of the average pooling layer. For the output dimensions of the average pooling layer O_{AVG} and the output dimensions of the convolutional layer O the equation to find the sensitivity of the loss function to a weight in the convolutional layer is:

$$\frac{d\mathcal{L}_{\mathcal{A}}}{dW_{abc}^{k,(l)}} = \sum_{m=1}^{O} \sum_{n=1}^{O} \sum_{i=1}^{O_{AVG}} \sum_{j=1}^{O_{AVG}} \frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial POOL_{ij}^{k}} \frac{\partial POOL_{ij}^{k}}{\partial z_{mn}^{k,(l)}} \frac{\partial z_{mn}^{k,(l)}}{\partial W_{abc}^{k,(l)}} \tag{4.39}$$

Once again, using the chain rule and Equations 4.36 and 4.37 determined for backpropagating through the convolutional and pooling layers, the sensitivity of the loss function to the remaining parameters can be found.

4.2.4 Recurrent Neural Networks (RNN)

Overview

Long Short Term Memory (LSTM) Neural Networks have impressive benchmarks in literature for tasks including language modeling, speech-to-text transcription, and machine translation [104]. The Recurrent Neural Network (RNN) is a predecessor to and includes as a special case this LSTM neural network [104]. Since LSTM is a type of RNN, RNN will be discussed first to serve as a foundation for LSTM that will be discussed in the next section.

RNNs were developed out of a need for a model that could handle sequential data: data like time-series data, a sentence, data where the value at some time-step t, x_t, depends on the data at the previous time-steps (eg. $x_{t-1}, x_{t-2}, x_{t-3}...$). The authors in [105] present the input as a sequence of vectors that each have m features. The sequential input to the RNN can have N time-steps, real or fictitious, $t = 0, ..., N$. The size of the output hidden state can be specified at some number n. At $t = 0$ the output is calculated very similarly to the neuron Equation 4.19 in section 4.2.2 and passed

through an activation function, in RNN this is typically tanh or ReLU [105]. In vector form with W as an $n \times m$ matrix:[7]

$$h_0(x_0) = \sigma(b + Wx_0) \tag{4.40}$$

What is different from the regular neural network is that h_0, in addition to being used as a potential output, is used as a hidden state. This hidden state is weighted by the hidden state weight $n \times n$ matrix U and used to calculate the output hidden state of the next time step:

$$h_1(x_1) = \sigma(Uh_0 + Wx_1 + b) \tag{4.41}$$

More generally [105][8]:

$$h_t(x_t) = \sigma(Uh_{t-1} + Wx_t + b), \quad t = 0, ..., N \tag{4.42}$$

For hidden states used as an output, they are passed through a linear layer which results in the output y_t vector with a size of r.

$$y_t(h_t) = Vh_t + c, \quad t = 0, ..., N \tag{4.43}$$

Where V is another weight matrix with dimensions $r \times n$ and c is the bias with a size of r.

Note that the weight matrices U, W, and V in the RNN are reused in each time step for all $t = 0, ..., N$. In other words, the number of parameters will not grow as the length of the sequence increases or N grows.

A visual depiction of the architecture of the RNN is shown in Figure 4.15 [106].

Training the Simple Recurrent Neural Network

Once again training the recurrent neural network involves the idea of backpropagation. For RNNs, in addition to backpropagating through the layers, there will need to be back propagation through time (BPTT) [105]. The authors in [105] provide equations for the calculation of the RNN parameters:

[7] note that the subscript now indicate a time-step

[8] Note that typically $h_{-1} = 0$ as there is no negative time

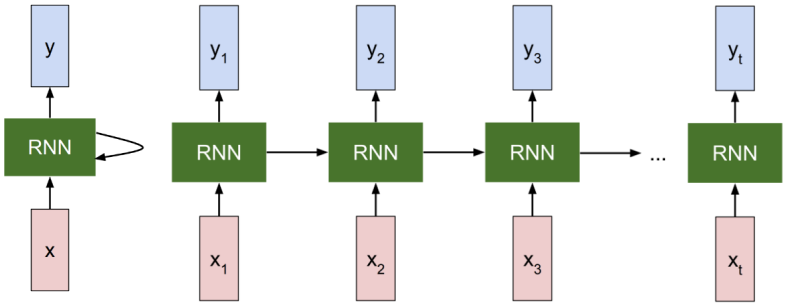


Figure 4.15: A visual depiction of an RNN. The far left shows the simplified RNN with x being passed in, the middle looping arrow represents the calculation of the hidden state at each time step and being passed to the calculation for the hidden state in the next time step. Each hidden state can be used to calculate an output y using Equation 4.43. The connected structure to the right of the simplified RNN shows the "unrolled" version of the RNN. The arrow to the right shows the hidden state being passed onto the calculation in the subsequent time step [106].

$$\begin{aligned} \frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial V} &= \sum_{t=0}^{N} \left(\frac{\mathcal{L}_{\mathcal{A}}^{t}}{\partial y_t} \right) \frac{\partial y_t}{\partial V} = \sum_{t=0}^{N} \left(\frac{\mathcal{L}_{\mathcal{A}}^{t}}{\partial y_t} \right) h_t^T \\ \left(\frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial c} \right)^T &= \sum_{t=0}^{N} \left(\frac{\mathcal{L}_{\mathcal{A}}^{t}}{\partial y_t} \right)^T \frac{\partial y_t}{\partial c} = \sum_{t=0}^{N} \left(\frac{\mathcal{L}_{\mathcal{A}}^{t}}{\partial y_t} \right)^T (1) \end{aligned} \tag{4.44}$$

The partial derivatives for the bias c is transposed to retain the chain-rule derivative for scalar [105].

For the gradients of the parameter U, W and b, the partial derivative of the loss with respect to the intermediate function in Equation 4.42 ($z_t = Uh_{t-1} + Wx_t + b$) will be defined. Note that to find the gradient of the parameters, the algorithm is required to step backwards in time starting from the $t = N$ and iteratively using Equation 4.45.

$$\begin{aligned} \lambda_t &= \frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial z_t} \\ &= (U\sigma_t')^T \lambda_{t+1} + (V\sigma_t')^T \frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial y_t}, \quad 0 < t \leq N-1 \end{aligned} \tag{4.45}$$

With the intermediate partial derivative λ_t defined the gradients of the other parameters are as follows:

$$\begin{aligned}
\frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial U} &= \sum_{t=0}^{N} \left(\frac{\mathcal{L}_{\mathcal{A}}^{t}}{\partial z_t} \right) \frac{\partial z_t}{\partial U} = \sum_{t=0}^{N} \lambda_t h_{t-1}^{T} \\
\frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial W} &= \sum_{t=0}^{N} \left(\frac{\mathcal{L}_{\mathcal{A}}^{t}}{\partial z_t} \right) \frac{\partial z_t}{\partial W} = \sum_{t=0}^{N} \lambda_t x_t^{T} \\
\left(\frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial b} \right)^{T} &= \sum_{t=0}^{N} \left(\frac{\mathcal{L}_{\mathcal{A}}^{t}}{\partial z_t} \right)^{T} \frac{\partial z_t}{\partial b} = \sum_{t=0}^{N} (\lambda_t)^{T}
\end{aligned} \tag{4.46}$$

From inspecting the equation for λ_t, it can be seen that gradient of the parameter matrix U and W will require U and V to be nested multiple times. For example with expansion to the next time step:

$$\lambda_t = (U\sigma_t')^T \left[(U\sigma_{t+1}')^T \lambda_{t+2} + (V\sigma_{t+1}')^T \frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial y_{t+1}} \right] + (V\sigma_t')^T \frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial y_t} \tag{4.47}$$

The nesting of the parameter matrices in the calculation of λ_t causes small changes in the these parameter matrices W and U to be greatly amplified as the sequence of data gets longer leading to the vanishing gradient or exploding gradient issues that typically make these simple RNNs unsuitable for practical use [105]. Although workarounds such as clipping or setting a threshold on the upper and lower bound of the parameter values seems to alleviate some of these issues, an approach more widely used are LSTMs discussed in the next section [105].

4.2.5 Long Short Term Memory (LSTM) Neural Networks

Long Short Term Memory (LSTM) Neural Networks build upon idea of the simple RNN and allow for information for the sequence of data to be learned while mitigating the issue of the vanishing gradient and the exploding gradient. The LSTM adds a **memory cell** c_t to the simple RNN architecture. In comparing with the simple RNN, the original calculation for h_t in the simple RNN becomes an intermediate output $\tilde{c}_t$ in the LSTM that is combined with

the memory cell before being output as a hidden state to be used in the calculation for subsequent time steps. The equations for the LSTM are as follows [105]:

$$\begin{aligned} \tilde{c}_t &= g(U_c h_{t-1} + W_c x_t + b) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\ h_t &= o_t \odot g(c_t) \end{aligned} \tag{4.48}$$

With $\odot$ representing element-wise multiplication and i_t, f_t and o_t representing the *input, forget, output* gates respectively. The equations for these gates are as follows:

$$\begin{aligned} i_t &= \sigma(U_i h_{t-1} + W_i x_t + b_i) \\ f_t &= \sigma(U_f h_{t-1} + W_f x_t + b_f) \\ o_t &= \sigma(U_o h_{t-1} + W_o x_t + b_o) \end{aligned} \tag{4.49}$$

With the following variables

- g: an activation function such as logistic function, tanh, or ReLU. Typically tanh [105]
- σ: an activation function such as logistic function, tanh, or ReLU. Typically Logistic to facilitate the "gating" function of forcing the output of the gate to be between 0 and 1 [105].
- U_{step}: The U parameter matrix for *step* in the LSTM. Values of the $step \in \{c, i, f, o\}$ for the *memory cell, input, forget* and *output* steps respectively
- W_{step}: The W parameter matrix for the *step* in the LSTM. Values of the $step \in \{c, i, f, o\}$ for the *memory cell, input, forget* and *output* steps respectively
- b_{step}: The b bias parameter for the *step* in the LSTM. Values of the $step \in \{c, i, f, o\}$ for the *memory cell, input, forget* and *output* steps respectively

Similar to the RNN the parameters in the memory cell, input, forget and output steps are reused at calculation for each time step so the number of parameters do not grow as the sequence gets longer. The total number of parameters for the LSTM is 4 times that of a RNN: $n_{\text{LSTM, params}} = 4(n^2 + nm + n)$ (recall that n is the number of hidden state features and m is the input vector number of features).

The architecture of the LSTM unit is shown in Figure 4.16

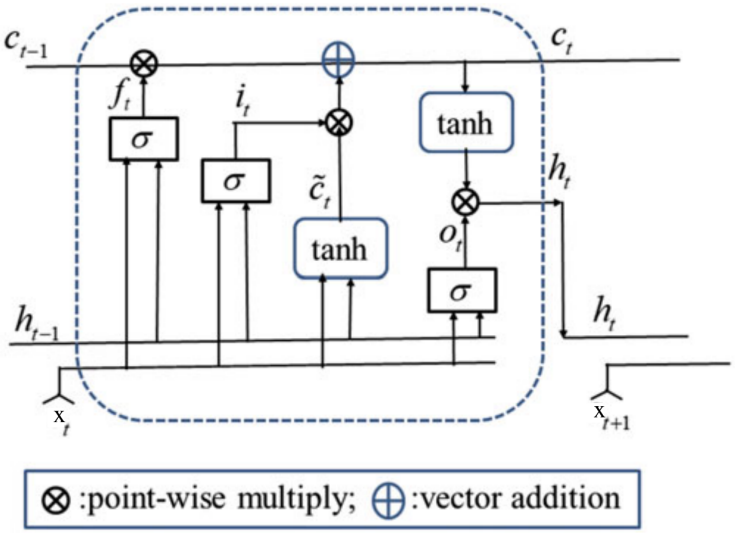


Figure 4.16: LSTM Architecture [105]. Note that the activation function g is shown as the tanh function.

Training LSTMs

The author in [105] does not provide equations for the backpropagation of the LSTM. However, since the LSTM is a type of RNN, the backpropagation through time (BPTT) would be used to find the gradients of the parameters $U_{step}, W_{step}, b_{step}$ for all of the steps $\{c, i, f, o\}$ by working backwards from the loss function $\mathcal{L}_{\mathcal{A}}$ to the point when these parameters are used.

4.2.6 Transformer Neural Networks

One limitation of the simple RNN and the LSTM is the sequential nature of the calculations. According to Vaswani et al. the sequential nature of the calculation (ie. h_t depends on the calculation of h_{t-1}) prevents any form of parallelization to occur limiting training speed and memory requirements increase with length of the sequences [107].

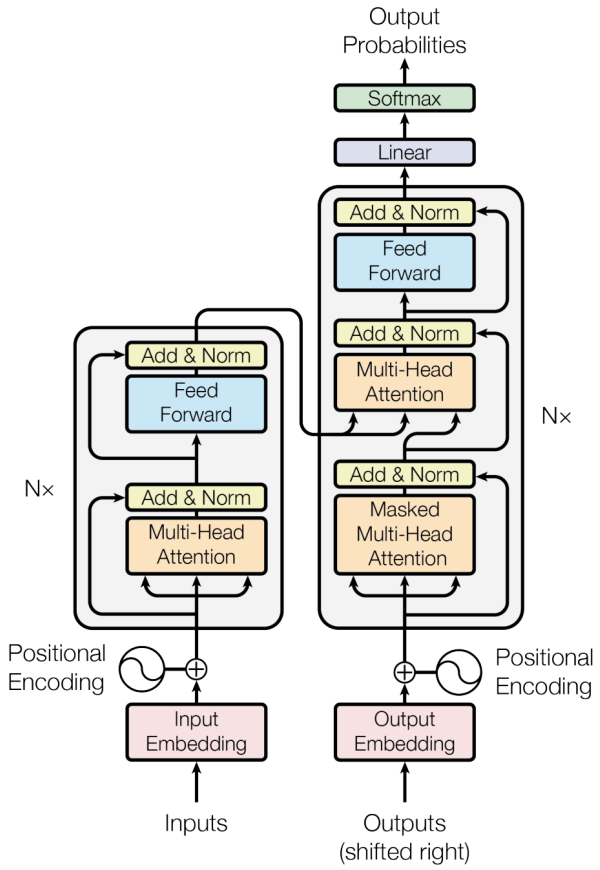


Figure 4.17: Transformer Architecture [107]

The transformer architecture that is proposed in [107] eliminates the sequential dependence and uses only the attention mechanism to draw global dependencies between the input and the output [107]. Although Vaswani et al. proposed the architecture for the task of machine translation (and more

recently natural language processing), the idea of leveraging contextual information in a sequence may be useful in the classification of the time series data in this thesis. The transformer architecture is shown in Figure 4.17.

Positional Encoding

The first step in the transformer architecture (Figure 4.17) is the positional encoding step. An input matrix X of d_{model} features and N time steps[9] (ie. $X \in \mathbb{R}^{N \times d_{model}}$) needs to have some sort of positional information injected to make use of the sequence since the transformer does not use recurrence or convolution [107]. Sine and Cosine functions were used to positionally encode each vector in X [107]:

$$\begin{aligned} PE_{(pos,2i)} &= sin(pos/10000^{2i/d_model}) \\ PE_{(pos,2i+1)} &= cos(pos/10000^{2i/d_model}) \end{aligned} \tag{4.50}$$

Where i represents the i-th dimension or feature, and pos represents the time-step or position. The $PE \in \mathbb{R}^{N \times d_{model}}$ matrix is added to X to positionally encode the input matrix.

Attention

After positionally encoding the input matrix, Vaswani et al. use a form of the attention mechanism as a "Scaled Dot-Product Attention" in [107]. In general, the input to the attention function consists of queries and keys of dimension d_k and values with dimensions d_v. The queries and keys are dot-product together, divided by $\sqrt{d_k}$, and put through a softmax to obtain the weights on the value. Queries, Keys and Values are compactly represented in the form $Q \in \mathbb{R}^{N \times d_k}, K \in \mathbb{R}^{N \times d_k}, V \in \mathbb{R}^{N \times d_v}$ respectively[10] and are used as follows in Equation[11] 4.51

[9] Note that in natural language processing (NLP), these time steps are tokens–words or special characters depending on the tokenization function [108]

[10] Note that N used here refers to the number of time steps, where as in [107] N refers to the number of stacked Attention + Feed Forward layers

[11] note that there is a "masked" version of this equation. Masking ensures that positions don't attend to subsequent positions (since early positions shouldn't have knowledge of future positions). As seen in Figure 4.18, masking is applied after the scaling step and before softmax in the scaled Dot-product attention. for some query vector q_j at position j and key vector k_l at position l, if $l > j$ then $q_j \cdot k_l = -\infty$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4.51}$$

A visual diagram of the Scaled Dot-Product Attention is shown in Figure 4.18.

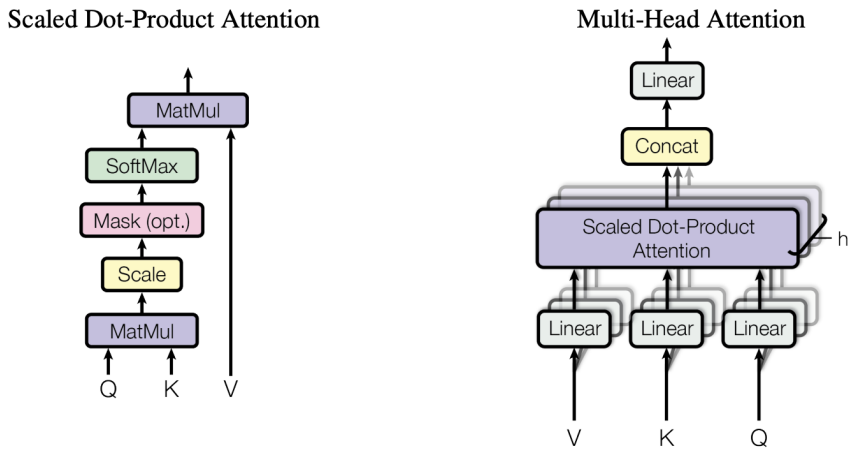


Figure 4.18: Attention. (left) Scaled-Dot Product Attention, (right) Multi-head attention. [107]

Computing Equation 4.51 once is referred to a single head of attention [107]. For a model with an input dimension of d_{model} Vaswani et al. employ a multi-headed attention in their transformer architecture by linearly projecting the queries, keys and values to a lower dimension d_k, d_k and d_v respectively. Each i head of attention has an associated trainable query weights W_i^Q, key weights W_i^K, and value weights W_i^V. Given a model input dimension d_{model} and total number of heads h, the lower dimensions d_k and d_v are calculated as follows: $(d_k = d_v = d_{model}/h)$.

For example, in [107], the input to the transformer are embeddings (a numerical representation of words that encode semantic and contextual information [109]) with 512 dimensions. Using 8 heads $h = 8$, the dimensions of the query, key and value for each head is $d_k = d_v = 512/8 = 64$ [107]. The resulting values from the calculation of each head of attention are concatenated together and matrix multiplied with an output weight matrix W^O.

The equation for multi-head attention in [107] is modified for some input matrix $X \in \mathbb{R}^{N \times d_{model}}$ and shown in Equation 4.52

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \\ \text{where} \quad \text{head}_i &= \text{Attention}(XW_i^Q, XW_i^K, XW_i^V) \end{aligned} \tag{4.52}$$

Where $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ [107]. These weight matrices are all learned in practice [107].

The output of the multi-head attention block is a matrix that is the same shape as the input matrix X and contains information on how the surrounding context should affect the values in each embedding vector. The output from the multi-head attention and X are added together and normalized before being passed onto the next block: the Fully Connected layer.

Aside: Attention Learning Intuition in NLP

Based on Equation 4.52, each embedding in X gets converted into a query q, key k and value v row vector using the respective weight matrices. These row vectors are vertically stacked forming Q, K and V. In the step with the matrix multiplication between QK^T in Equation 4.51, each query is dot-product with each key resulting in a $N \times N$ matrix that shows how similar the j-th query is to the l-th key.

$$QK^T = \begin{bmatrix} q_1 \cdot k_1 & \dots & q_1 \cdot k_N \\ \vdots & \ddots & \vdots \\ q_N \cdot k_1 & \dots & q_N \cdot k_N \end{bmatrix} \tag{4.53}$$

In consideration of the QK^T step, the weight matrices for the query and key (W_i^Q and W_i^K respectively) are trained so that if there is a relationship between a query at j: q_j and key at l: k_l, then the vectors q_j and k_l will be similar (ie. produce relatively large positive value when dot-product). The QK^T step is responsible determining which elements in the sequence should affect which other elements in the sequence and the magnitude of the effect.

Since the "what" is handled by the QK^T, the "how" is handled by V and W^O. Through training, W_i^V and W^O together should learn the direction (in terms of vectors) that the l-th embedding (from the key) should add to the j-th embedding (from the query) that had relationships discovered in the QK^T step. Since

$$V = \begin{bmatrix} v_1 \\ \vdots \\ v_N \end{bmatrix} \tag{4.54}$$

And treating the softmax and division by $\sqrt{d_k}$ as a function $f()$, the output of a single head of attention is shown in Equation 4.55[12]

$$f(QK^T)V = \begin{bmatrix} f(q_1 \cdot k_1) * v_1 + \cdots + f(q_1 \cdot k_N) * v_N \\ \vdots \\ f(q_N \cdot k_1) * v_1 + \cdots + f(q_N \cdot k_N) * v_N \end{bmatrix} \tag{4.55}$$

All the heads of attention are concatenated together and matrix multiplied by W^O to project the heads of attention from d_v back to d_{model} (meaning W^O has a part in influencing the "direction" as well).

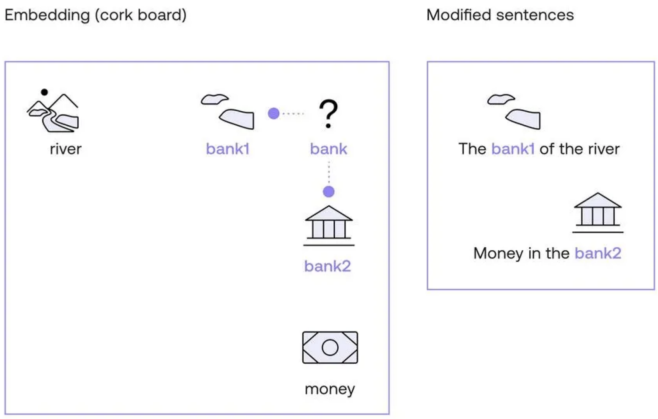


Figure 4.19: The river bank pushes the embedding of bank to the left closer to the river related words. Money pushes "bank" embedding toward the financial related words [110].

In terms of NLP, the output of the multi-head attention nudges the original embeddings in X in the direction that is more semantically appropriate based on the context. [110] uses the example of the word "bank". Based on

[12]considering how each row is an embedding, notice how the value and key subscripts increment in tandem. This can be interpreted as the embedding of key's effect on the embedding of the query.

the context "bank" could be either a "the bank of the river" or "money in the bank". Keeping in mind Equation 4.53, well-trained query weights and key weights should produce similar vectors when "bank" is the query and "river" or "money" is the key. With value vectors defining a direction that's added to the original embeddings, in this example, "river's" value vector should nudge the "bank's" embedding closer to "river" related embeddings and "money's" value vector should nudge "bank's" embedding closer to the finance related embeddings, Figure 4.19 [110].

Feed Forward (Fully Connected)

After the input is modified by the output of the multi-head attention layer. A fully connected layer is applied to each position separately and identically with a ReLU activation in between [107].

$$\text{FFN}(x) = \max\left(0, xW_1 + b_1\right)W_2 + b_2 \tag{4.56}$$

The output from the feed-forward fully connected layer is then added and normalized with the output from the previous layer as shown in the transformer architecture Figure 4.17. The feed forward and attention layers are repeated as many times as the user specifies and eventually outputs the probabilities (prediction).

Training a Transformer

The authors in [107] do not go into the mathematical theory for training the transformer architecture. It is assumed that similar to the other architectures, backpropagation is used to train the weights W^O, and each of the W_i^Q, W_i^K and W_i^V in the multi-headed attention step. Liang in [111] uses calculus to find the gradients of these parameter matrices and has also validated it with pyTorch's implementation of the MultiHeadAttention.

4.3 Classification Strategy

The objective of the thesis is to use positional + IMU sensor data to create a system that can classify fine-grained cooking tasks in real-time. Once again, this real-time classification system would enable ADL quality monitoring which would allow for the early detection of physical or cognitive decline [61].

Model and Input

Time series data is a series of data that each consists of a value at an associated timestamp. In terms of the system investigated in this thesis, this could be a series of position in the x-axis measured at time $t_1, t_2, t_3...$ respectively. The data can be said to be indexed by time [112] typically in ascending order.

Table 4.1: Summary of the models discussed in this section along with their Input. Acronyms used: Feature Extraction (FE), Time-series (TS)

Model	**FE?**	**Input**
kNN	Yes	Extracted features from a subsequence
kNN with DTW	No	The full subsequence
SVM	Yes	Extracted features from a subsequence
Random Forests	Yes	Extracted features from a subsequence
Shapelet Transform	No	The full subsequence
DNN with Features	Yes	Extracted features from a subsequence
DNN with TS	No	TS with constant $n_{samples}$
1D CNN [113]	No	TS with constant $n_{samples}$
LSTM	No	TS with variable $n_{samples}$ [114]
Transformer NN	No	TS with variable $n_{samples}$

Based on the models investigated in this section, some can take the time-series input as is, and some requires "Feature Extraction" or compressing the data down into a feature vector that can be used with the model. In the section for Random Forests, random subsequences of a time-series (with

a minimum length) was taken and the subsequence was compressed into a feature vector consisting of the Mean, Standard Deviation and the Slope [83]. The Time-series Feature extraction library (TSFEL) is a library that Barandas et al. built for the time-series feature extraction task and has over 60 features across temporal, statistical and spectral domains [115]. In addition to the mean, standard deviation, and slope, features can include the min, max, median, absolute energy, max frequency, median frequency and many others [116]. Table 4.1 summarizes the models discussed and the type of input required.

Real-Time Considerations

In addition to the classification model that must be evaluated, the system's practicality in real-time application must be investigated. The following are considerations for real-time classification:

- The latency of the model or the speed of the classification. The latency of classification must be lower than the time taken to collect the data for classification. If the latency of classification is higher than the data collection interval, then the classification system will not be able to keep up with the data input.
- Robustness to temporal variation. The task speed is expected to vary greatly depending on multiple factors such as skill and health conditions. For example, able-bodied adults would be expected to complete tasks faster than older adults with comorbidities.
- Real-time activity boundary detection. Since each task inherently varies in time, opening a door versus continuous chopping, how can the start and end of an input be selected to capture a specific task.

Chapter 5

Pilot Testing of Detecting Activities to make a Sandwich

Chapter 6

PASS Tasks

Bibliography

[1] Pozyx, "Creator One Kit for research and prototyping - Pozyx." [Online]. Available: https://www.pozyx.io/creator-one-kit

[2] S. C. Government of Canada, "The Daily — Population projections: Canada, the provinces and territories, 2013 to 2063," Sep. 2014, last Modified: 2014-09-17. [Online]. Available: https://www150.statcan.gc.ca/n1/daily-quotidien/140917/dq140917a-eng.htm

[3] L. M. Tijsen, E. W. Derksen, W. P. Achterberg, and B. I. Buijck, "Challenging rehabilitation environment for older patients," *Clinical Interventions in Aging*, vol. Volume 14, pp. 1451–1460, Aug. 2019. [Online]. Available: https://www.dovepress.com/challenging-rehabilitation-environment-for-older-patients-peer-reviewed-article-CIA

[4] I. Wilkinson and A. Harper, "Comprehensive geriatric assessment, rehabilitation and discharge planning," *Medicine*, vol. 49, no. 1, pp. 10–16, Jan. 2021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1357303920302711

[5] P. F. Edemekong, D. L. Bomgaars, S. Sukumaran, and S. B. Levy, "Activities of Daily Living," in *StatPearls*. Treasure Island (FL): StatPearls Publishing, 2022. [Online]. Available: http://www.ncbi.nlm.nih.gov/books/NBK470404/

[6] E. Jaul and J. Barron, "Age-Related Diseases and Clinical and Public Health Implications for the 85 Years Old and Over Population," *Frontiers in Public Health*, vol. 5, p. 335, Dec. 2017. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5732407/

[7] J. Hewitt, S. Long, B. Carter, S. Bach, K. McCarthy, and A. Clegg, "The prevalence of frailty and its association with clinical outcomes in general surgery: a systematic review and meta-analysis," *Age and Ageing*, vol. 47, no. 6, pp. 793–800, Nov. 2018. [Online]. Available: https://doi.org/10.1093/ageing/afy110

[8] S. Conroy, "Defining frailty — the holy grail of geriatric medicine," *The Journal of Nutrition, Health and Aging*, vol. 13, no. 4, pp. 389–389, Apr. 2009. [Online]. Available: http://link.springer.com/10.1007/s12603-009-0050-9

[9] X. Chen, G. Mao, and S. X. Leng, "Frailty syndrome: an overview," *Clinical Interventions in Aging*, vol. 9, pp. 433–441, Mar. 2014. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3964027/

[10] "Diagnosis and Management of Dementia: Review | Dementia and Cognitive Impairment | JAMA | JAMA Network." [Online]. Available: https://jamanetwork.com/journals/jama/fullarticle/2753376

[11] I. Arevalo-Rodriguez, N. Smailagic, M. Roqué I Figuls, A. Ciapponi, E. Sanchez-Perez, A. Giannakou, O. L. Pedraza, X. Bonfill Cosp, and S. Cullum, "Mini-Mental State Examination (MMSE) for the detection of Alzheimer's disease and other dementias in people with mild cognitive impairment (MCI)," *The Cochrane Database of Systematic Reviews*, vol. 2015, no. 3, p. CD010783, Mar. 2015.

[12] M. Desai, L. A. Pratt, H. Lentzner, and K. N. Robinson, "Trends in vision and hearing among older Americans," *Aging Trends (Hyattsville, Md.)*, no. 2, pp. 1–8, Mar. 2001.

[13] J. R. Evans, A. E. Fletcher, R. P. L. Wormald, E. S.-W. Ng, S. Stirling, L. Smeeth, E. Breeze, C. J. Bulpitt, M. Nunes, D. Jones, and A. Tulloch, "Prevalence of visual impairment in people aged 75 years and older in Britain: results from the MRC trial of assessment and management of older people in the community," *The British Journal of Ophthalmology*, vol. 86, no. 7, pp. 795–800, Jul. 2002.

[14] B. K. Swenor, M. J. Lee, V. Varadaraj, H. E. Whitson, and P. Y. Ramulu, "Aging With Vision Loss: A Framework for Assessing the

Impact of Visual Impairment on Older Adults," *The Gerontologist*, vol. 60, no. 6, pp. 989–995, Aug. 2020. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7427480/

[15] M. E. Tinetti, C. S. Williams, and T. M. Gill, "Dizziness among older adults: a possible geriatric syndrome," *Annals of Internal Medicine*, vol. 132, no. 5, pp. 337–344, Mar. 2000.

[16] S. Iwasaki and T. Yamasoba, "Dizziness and Imbalance in the Elderly: Age-related Decline in the Vestibular System," *Aging and Disease*, vol. 6, no. 1, pp. 38–47, Feb. 2014. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4306472/

[17] G. Savarese and L. H. Lund, "Global Public Health Burden of Heart Failure," *Cardiac Failure Review*, vol. 3, no. 1, pp. 7–11, Apr. 2017. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5494150/

[18] B. Ziaeian and G. C. Fonarow, "Epidemiology and aetiology of heart failure," *Nature Reviews. Cardiology*, vol. 13, no. 6, pp. 368–378, Jun. 2016.

[19] R. D. S. Watson, C. R. Gibbs, and G. Y. H. Lip, "Clinical features and complications," *BMJ : British Medical Journal*, vol. 320, no. 7229, pp. 236–239, Jan. 2000. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1117436/

[20] I. o. M. U. C. o. S. S. C. D. Criteria, "Ischemic Heart Disease," in *Cardiovascular Disability: Updating the Social Security Listings*. National Academies Press (US), 2010. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK209964/

[21] M. V. Madhavan, B. J. Gersh, K. P. Alexander, C. B. Granger, and G. W. Stone, "Coronary Artery Disease in PatientsÂ â‰¥80 Years of Age," *Journal of the American College of Cardiology*, vol. 71, no. 18, pp. 2015–2040, May 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0735109718336167

[22] A. Sapra and P. Bhandari, "Diabetes Mellitus," in *StatPearls*. Treasure Island (FL): StatPearls Publishing, 2021. [Online]. Available: http://www.ncbi.nlm.nih.gov/books/NBK551501/

[23] "Glucose tolerance tests: What exactly do they involve?" in *InformedHealth.org [Internet].* Institute for Quality and Efficiency in Health Care (IQWiG), Oct. 2020. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK279331/

[24] M. J. Lespasio, A. A. Sultan, N. S. Piuzzi, A. Khlopas, M. E. Husni, G. F. Muschler, and M. A. Mont, "Hip Osteoarthritis: A Primer," *The Permanente Journal*, vol. 22, pp. 17–084, Jan. 2018. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5760056/

[25] J. M. Hootman and C. G. Helmick, "Projections of US prevalence of arthritis and associated activity limitations," *Arthritis and Rheumatism*, vol. 54, no. 1, pp. 226–229, Jan. 2006.

[26] D. Prieto-Lhambra, D. Hunter, and N. Arden, *Osteoarthritis: The Facts*, second edition ed., ser. The Facts Series. Oxford, New York: Oxford University Press, Sep. 2014.

[27] D. J. Hunter and S. Bierma-Zeinstra, "Osteoarthritis," *The Lancet*, vol. 393, no. 10182, pp. 1745–1759, Apr. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140673619304179

[28] R. Altman, E. Asch, D. Bloch, G. Bole, D. Borenstein, K. Brandt, W. Christy, T. D. Cooke, R. Greenwald, and M. Hochberg, "Development of criteria for the classification and reporting of osteoarthritis. Classification of osteoarthritis of the knee. Diagnostic and Therapeutic Criteria Committee of the American Rheumatism Association," *Arthritis and Rheumatism*, vol. 29, no. 8, pp. 1039–1049, Aug. 1986.

[29] J. E. Compston, M. R. McClung, and W. D. Leslie, "Osteoporosis," *The Lancet*, vol. 393, no. 10169, pp. 364–376, Jan. 2019, publisher: Elsevier. [Online]. Available: https://www-thelancet-com.login.ezproxy.library.ualberta.ca/journals/lancet/article/PIIS0140-6736(18)32112-3/fulltext

[30] J. L. Porter and M. Varacallo, "Osteoporosis," in *StatPearls*. Treasure Island (FL): StatPearls Publishing, 2023. [Online]. Available: http://www.ncbi.nlm.nih.gov/books/NBK441901/

[31] D. L. Glaser and F. S. Kaplan, "Osteoporosis: Definition and Clinical Presentation," *Spine*, vol. 22, no. 24, p. 12S, Dec. 1997.

[32] M. Pashmdarfard and A. Azad, "Assessment tools to evaluate Activities of Daily Living (ADL) and Instrumental Activities of Daily Living (IADL) in older adults: A systematic review," *Medical Journal of the Islamic Republic of Iran*, vol. 34, p. 33, Apr. 2020. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7320974/

[33] E. McMahon, "Katz Index of Independence in Activities of Daily Living," p. 2.

[34] P. P. Katz for the Association of Rheumatology Health Professionals Outcomes Measures Task Force, "Measures of adult general functional status: The Barthel Index, Katz Index of Activities of Daily Living, Health Assessment Questionnaire (HAQ), MACTAR Patient Preference Disability Questionnaire, and Modified Health Assessment Questionnaire (MHAQ)," *Arthritis Care & Research*, vol. 49, no. S5, pp. S15–S27, 2003, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/art.11415. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/art.11415

[35] AHS, "Barthel Index of Activities of Daily Living."

[36] S. Katz, A. B. Ford, R. W. Moskowitz, B. A. Jackson, and M. W. Jaffe, "Studies of Illness in the Aged: The Index of ADL: A Standardized Measure of Biological and Psychosocial Function," *JAMA*, vol. 185, no. 12, pp. 914–919, Sep. 1963. [Online]. Available: https://doi.org/10.1001/jama.1963.03060120024016

[37] M. Davidson, "Functional Independence Measure," in *Encyclopedia of Quality of Life and Well-Being Research*, A. C. Michalos, Ed. Dordrecht: Springer Netherlands, 2014, pp. 2373–2376.

[38] "FIM(TM)." [Online]. Available: https://www.tbims.org/combi/FIM/

[39] E. Dutil, A. Forget, M. Vanier, and C. Gaudreault, "Development of the ADL Profile: An Evaluation for Adults with Severe Head Injury," *Occupational Therapy In Health Care*, vol. 7, no. 1, pp. 7–22, Jan. 1990.

[40] C. L. Bottari, C. Dassa, C. M. Rainville, and E. Dutil, "The IADL profile: development, content validity, intra- and interrater agreement," *Canadian Journal of Occupational Therapy. Revue Canadienne D'ergotherapie*, vol. 77, no. 2, pp. 90–100, Apr. 2010.

[41] N. Kennedy, A. Barion, A. Rademaker, G. Rehkemper, and S. Weintraub, "The Activities of Daily Living Questionnaire A Validation Study in Patients with Dementia," *Alzheimer disease and associated disorders*, vol. 18, pp. 223–30, Oct. 2004.

[42] A. Perry, M. Morris, C. Unsworth, S. Duckett, J. Skeat, K. Dodd, N. Taylor, and K. Reilly, "Therapy outcome measures for allied health practitioners in Australia: the AusTOMs," *International Journal for Quality in Health Care*, vol. 16, no. 4, pp. 285–291, Aug. 2004. [Online]. Available: https://doi.org/10.1093/intqhc/mzh059

[43] S. A. Haymes, A. W. Johnston, and A. D. Heyes, "The Development of the Melbourne Low-Vision ADL Index: A Measure of Vision Disability," *Investigative Ophthalmology & Visual Science*, vol. 42, no. 6, pp. 1215–1225, May 2001.

[44] R. G. Brown, B. MacCarthy, M. Jahanshahi, and C. D. Marsden, "Accuracy of Self-Reported Disability in Patients With Parkinsonism," *Archives of Neurology*, vol. 46, no. 9, pp. 955–959, Sep. 1989. [Online]. Available: https://doi.org/10.1001/archneur.1989.00520450025014

[45] M. Holbrook and C. E. Skilbeck, "AN ACTIVITIES INDEX FOR USE WITH STROKE PATIENTS," *Age and Ageing*, vol. 12, no. 2, pp. 166–170, 1983. [Online]. Available: https://academic.oup.com/ageing/article-lookup/doi/10.1093/ageing/12.2.166

[46] M. P. Lawton and E. M. Brody, "Assessment of older people: self-maintaining and instrumental activities of daily living," *The Gerontologist*, vol. 9, no. 3, pp. 179–186, 1969.

[47] J. C. Rogers, "Performance Assessment of Self-Care Skills," Apr. 2014. [Online]. Available: http://doi.apa.org/getdoi.cfm?doi=10.1037/t16068-000

[48] D. Chisholm, P. Toto, K. Raina, M. Holm, and J. Rogers, "Evaluating capacity to live independently and safely in the community: Performance Assessment of Self-care Skills," *The British journal of occupational therapy*, vol. 77, no. 2, pp. 59–63, Feb. 2014. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4186770/

[49] C. M. Cullum, K. Saine, L. D. Chan, K. Martin-Cook, K. F. Gray, and M. F. Weiner, "Performance-Based instrument to assess functional capacity in dementia: The Texas Functional Living Scale," *Neuropsychiatry, Neuropsychology, and Behavioral Neurology*, vol. 14, no. 2, pp. 103–108, 2001.

[50] A. Staff, "The (Original) Barthel Index of ADLs," Sep. 2008. [Online]. Available: https://www.elitelearning.com/resource-center/rehabilitation-therapy/the-original-barthel-index-of-adls/

[51] "Measures of adult general functional status: The Barthel Index, Katz Index of Activities of Daily Living, Health Assessment Questionnaire (HAQ), MACTAR Patient Preference Disability Questionnaire, and Modified Health Assessment Questionnaire (MHAQ)." [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/art.11415

[52] K. M. Hall, B. B. Hamilton, W. A. Gordon, and N. D. Zasler, "Functional Independence Measure and Functional Assessment Measure," Jun. 2014, institution: American Psychological Association. [Online]. Available: http://doi.apa.org/getdoi.cfm?doi=10.1037/t28782-000

[53] "ADL Profile â€“ Strokengine." [Online]. Available: https://strokengine.ca/en/assessments/adl/

[54] cunsworth2014, "AusTOMs," Nov. 2019. [Online]. Available: https://austoms.com/

[55] "Self-assessment Parkinsonâ€™s Disease Disability Scale | RehabMeasures Database," May 2013. [Online]. Available: https://www.sralab.org/rehabilitation-measures/self-assessment-parkinsons-disease-disability-scale

[56] J. Schuling, R. De Haan, M. Limburg, and K. H. Groenier, "The Frenchay Activities Index. Assessment of functional status in stroke patients." *Stroke*, vol. 24, no. 8, pp. 1173–1177, Aug. 1993. [Online]. Available: https://www.ahajournals.org/doi/10.1161/01.STR.24.8.1173

[57] E. McMahon, "Lawton â€“Brody Instrumental Activities of Daily Living Scale (IADL)."

[58] "Performance Assessment of Self-Care Skills | RehabMeasures Database," Jun. 2015. [Online]. Available: https://www.sralab.org/rehabilitation-measures/performance-assessment-self-care-skills

[59] U. F. O. Themes, "Texas Functional Living Scale (TFLS)," Jul. 2017. [Online]. Available: https://nursekey.com/texas-functional-living-scale-tfls/

[60] N. Camp, M. Lewis, K. Hunter, J. Johnston, M. Zecca, A. Di Nuovo, and D. Magistro, "Technology Used to Recognize Activities of Daily Living in Community-Dwelling Older Adults," *International Journal of Environmental Research and Public Health*, vol. 18, no. 1, p. 163, Jan. 2021, number: 1 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/1660-4601/18/1/163

[61] N. Gadey, P. Pataunia, A. Chan, and A. RÃos RincÃ³n, "Technologies for monitoring activities of daily living in older adults: a systematic review," *Disability and Rehabilitation. Assistive Technology*, pp. 1–10, Mar. 2023.

[62] S. A. Sikkes and J. d. Rotrou, "A qualitative review of instrumental activities of daily living in dementia: what's cooking?" *Neurodegenerative Disease Management*, vol. 4, no. 5, pp. 393–400, Oct. 2014. [Online]. Available: https://www.futuremedicine.com/doi/10.2217/nmt.14.24

[63] B. Bouchard, K. Bouchard, and A. Bouzouane, "A smart cooking device for assisting cognitively impaired users," *Journal of Reliable Intelligent Environments*, vol. 6, no. 2, pp. 107–125, Jun. 2020. [Online]. Available: https://link.springer.com/10.1007/s40860-020-00104-3

[64] E. Dubuc, M. Gagnon-Roy, M. Couture, N. Bier, S. Giroux, and C. Bottari, "Perceived needs and difficulties in meal preparation of people living with traumatic brain injury in a chronic phase: Supporting long-term services and interventions," *Australian Occupational Therapy Journal*, vol. 66, no. 6, pp. 720–730, 2019, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1440-1630.12611. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/1440-1630.12611

[65] P. Malani, J. Kullgren, E. Solway, J. Wolfson, C. Leung, D. Singer, and M. Kirch, "The Joy of Cooking and its Benefits for Older Adults," University of Michigan, Tech. Rep., Jun. 2020.

[66] R. Yared, B. Abdulrazak, T. Tessier, and P. Mabilleau, "Cooking risk analysis to enhance safety of elderly people in smart kitchen," in *Proceedings of the 8th ACM International Conference on PErvasive Technologies Related to Assistive Environments*. Corfu Greece: ACM, Jul. 2015, pp. 1–4. [Online]. Available: https://dl.acm.org/doi/10.1145/2769493.2769516

[67] D. J. Cook, "Learning Setting-Generalized Activity Models for Smart Spaces," *IEEE intelligent systems*, vol. 2010, no. 99, p. 1, Sep. 2010. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3068197/

[68] B. Logan and J. Healey, "Sensors to Detect the Activities of Daily Living," in *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*, Aug. 2006, pp. 5362–5365, iSSN: 1557-170X.

[69] N. Sarma, S. Chakraborty, and D. S. Banerjee, "Activity Recognition through Feature Learning and Annotations using LSTM," in *2019 11th International Conference on Communication Systems & Networks (COMSNETS)*. Bengaluru, India: IEEE, Jan. 2019, pp. 444–447. [Online]. Available: https://ieeexplore.ieee.org/document/8711147/

[70] M. Mokhtari, B. Abdulrazak, and H. Aloulou, Eds., *Smart Homes and Health Telematics, Designing a Better Future: Urban Assisted Living: 16th International Conference, ICOST 2018, Singapore, Singapore, July 10-12, 2018, Proceedings*, ser. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, vol. 10898. [Online]. Available: http://link.springer.com/10.1007/978-3-319-94523-1

[71] K. Yordanova, S. Lüdtke, S. Whitehouse, F. Krüger, A. Paiement, M. Mirmehdi, I. Craddock, and T. Kirste, "Analysing Cooking Behaviour in Home Settings: Towards Health Monitoring," *Sensors*, vol. 19, no. 3, p. 646, Jan. 2019, number: 3 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/1424-8220/19/3/646

[72] D. J. Cook and M. Schmitter-Edgecombe, "Assessing the Quality of Activities in a Smart Environment," *Methods of Information in Medicine*, vol. 48, no. 05, pp. 480–485, 2009. [Online]. Available: http://www.thieme-connect.de/DOI/DOI?10.3414/ME0592

[73] P. N. Dawadi, D. J. Cook, M. Schmitter-Edgecombe, and C. Parsey, "Automated Assessment of Cognitive Health Using Smart Home Technologies," *Technology and health care : official journal of the European Society for Engineering and Medicine*, vol. 21, no. 4, pp. 323–343, 2013. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4143248/

[74] P.-W. Chen, N. A. Baune, I. Zwir, J. Wang, V. Swamidass, and A. W. Wong, "Measuring Activities of Daily Living in Stroke Patients with Motion Machine Learning Algorithms: A Pilot Study," *International Journal of Environmental Research and Public Health*, vol. 18, no. 4, p. 1634, Feb. 2021. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7915561/

[75] S. Pan, M. Berges, J. Rodakowski, P. Zhang, and H. Y. Noh, "Fine-Grained Activity of Daily Living (ADL) Recognition Through Heterogeneous Sensing Systems With Complementary Spatiotemporal Characteristics," *Frontiers in Built Environment*, vol. 6, 2020. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fbuil.2020.560497

[76] J. Chung, G. Demiris, and H. J. Thompson, "Ethical Considerations Regarding the Use of Smart Home Technologies for Older Adults: An Integrative Review," *Annual Review of Nursing Research*, vol. 34, no. 1, pp. 155–181, Jan. 2016. [Online]. Available: http://connect.springerpub.com/lookup/doi/10.1891/0739-6686.34.155

[77] G. Demiris, "Privacy and social implications of distinct sensing approaches to implementing smart homes for older adults," *Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual International Conference*, vol. 2009, pp. 4311–4314, 2009.

[78] J. Leo and D. Goodwin, "Negotiated Meanings of Disability Simulations in an Adapted Physical Activity Course: Learning From Student Reflections," *Adapted Physical Activity Quarterly*, vol. 31, no. 2, pp. 144–161, Apr. 2014. [Online]. Available: https://journals.humankinetics.com/view/journals/apaq/31/2/article-p144.xml

[79] Pozyx, "Hardware setup." [Online]. Available: https://docs.pozyx.io/creator/hardware-setup

[80] ——, "Configuration of the UWB parameters (Arduino)." [Online]. Available: https://docs.pozyx.io/creator/configuration-of-the-uwb-parameters-arduino

[81] Z. Zhang, "Introduction to machine learning: K-nearest neighbors," vol. 4, no. 11, p. 218. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4916348/

[82] Scipy.stats.mode — SciPy v0.7 Reference Guide (DRAFT). [Online]. Available: https://docs.scipy.org/doc/scipy-0.7.x/reference/generated/scipy.stats.mode.html

[83] J. Faouzi, "Time Series Classification: A Review of Algorithms and Implementations," in *Time Series Analysis - Recent Advances, New Perspectives and Applications*, J. Rocha, C. M. Viana, and S. Oliveira, Eds. IntechOpen. [Online]. Available: https://www.intechopen.com/chapters/1185930

[84] J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, and A. Lopez, "A comprehensive survey on support vector machine classification: Applications, challenges and trends," vol. 408, pp. 189–215. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0925231220307153

[85] 1.12. Multiclass and multioutput algorithms. scikit-learn. [Online]. Available: https://scikit-learn/stable/modules/multiclass.html

[86] Y. Liu, Y. Wang, and J. Zhang, "New Machine Learning Algorithm: Random Forest," in *Information Computing and Applications*, B. Liu, M. Ma, and J. Chang, Eds. Springer, pp. 246–252.

[87] V. E. Lee and L. Liu, "Decision Trees: Theory and Algorithms."

[88] L. Breiman, "Random Forests," vol. 45, no. 1, pp. 5–32. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[89] 1.10. Decision Trees. scikit-learn. [Online]. Available: https://scikit-learn/stable/modules/tree.html

[90] Scikit-learn/sklearn/tree/splitter.pyx at main · scikit-learn/scikit-learn · GitHub. [Online]. Available: https://github.com/scikit-learn/scikit-learn/blob/0b0b90b75102226d27e3947b9766729325d7042c/sklearn/tree/_splitter.pyx

[91] RandomForestClassifier. scikit-learn. [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[92] P. Probst, M. Wright, and A.-L. Boulesteix, "Hyperparameters and Tuning Strategies for Random Forest," vol. 9, no. 3, p. e1301. [Online]. Available: http://arxiv.org/abs/1804.03515

[93] L. Ye and E. Keogh, "Time series shapelets: A new primitive for data mining," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 947–956. [Online]. Available: https://dl.acm.org/doi/10.1145/1557019.1557122

[94] D. A. Roberts, S. Yaida, and B. Hanin, *The Principles of Deep Learning Theory*. [Online]. Available: http://arxiv.org/abs/2106.10165

[95] What is a Neural Network? — IBM. [Online]. Available: https://www.ibm.com/topics/neural-networks

[96] M. M. Taye, "Theoretical Understanding of Convolutional Neural Network: Concepts, Architectures, Applications, Future Directions," vol. 11, no. 3, p. 52. [Online]. Available: https://www.mdpi.com/2079-3197/11/3/52

[97] K. B. Prakash, R. Kannan, S. Alexander, and G. R. Kanagachidambaresan, Eds., *Advanced Deep Learning for Engineers and Scientists: A Practical Approach*, ser. EAI/Springer Innovations in Communication and Computing. Springer International Publishing. [Online]. Available: https://link.springer.com/10.1007/978-3-030-66519-7

[98] P. Purwono, A. Ma'arif, W. Rahmaniar, H. Imam, H. I. K. Fathurrahman, A. Frisky, and Q. M. U. Haq, "Understanding of Convolutional Neural Network (CNN): A Review," vol. 2, pp. 739–748.

[99] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions," vol. 8, no. 1, p. 53. [Online]. Available: https://doi.org/10.1186/s40537-021-00444-8

[100] M. Swapna, D. Y. Sharma, and B. Prasad, "CNN Architectures: Alex Net, Le Net, VGG, Google Net, Res Net," vol. 8, pp. 953–959.

[101] Training a Classifier — PyTorch Tutorials 2.4.0+cu121 documentation. [Online]. Available: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

[102] R. Vasudev. Understanding and Calculating the number of Parameters in Convolution Neural Networks (CNNs). Medium. [Online]. Available: https://towardsdatascience.com/understanding-and-calculating-the-number-of-parameters-in-convolution-neural-networks-cnns-fc88790d530d

[103] N. Zakaria. Backpropagation in CNN; A Mathematically Explicit Exposition. Medium. [Online]. Available: https://nordinzakaria.medium.com/backpropagation-in-cnn-a-mathematically-explicit-exposition-3f723f0ad9a0

[104] A. Sherstinsky, "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network," vol. 404, p. 132306. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167278919305974

[105] F. M. Salem, *Recurrent Neural Networks: From Simple to Gated Architectures*. Springer International Publishing. [Online]. Available: https://link.springer.com/10.1007/978-3-030-89929-5

[106] CS231n Convolutional Neural Networks for Visual Recognition. [Online]. Available: https://cs231n.github.io/rnn/

[107] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need. [Online]. Available: http://arxiv.org/abs/1706.03762

[108] Torchtext.data.utils — Torchtext 0.18.0 documentation. [Online]. Available: https://pytorch.org/text/stable/data_utils.html

[109] T. H. Teng, K. D. Varathan, and F. Crestani, "A comprehensive review of cyberbullying-related content classification in online social media," vol. 244, p. 122644. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417423031469

[110] Amanatullah. The Attention Mechanism. Medium. [Online]. Available: https://medium.com/@amanatulla1606/the-attention-mechanism-797e51c45d46

[111] Longx. Transformer Attention Layer gradient. Longxiang He. [Online]. Available: https://say-hello2y.github.io/2022-09-07/attention-gradient

[112] Y. Yin, "Prediction and analysis of time series data based on granular computing," vol. 17. [Online]. Available: https://www.frontiersin.org/journals/computational-neuroscience/articles/10.3389/fncom.2023.1192876/full

[113] Z. Wang, W. Yan, and T. Oates. Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline. [Online]. Available: http://arxiv.org/abs/1611.06455

[114] LSTM — PyTorch 2.5 documentation. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

[115] M. Barandas, D. Folgado, L. Fernandes, S. Santos, M. Abreu, P. Bota, H. Liu, T. Schultz, and H. Gamboa, "TSFEL: Time Series Feature Extraction Library," vol. 11, p. 100456. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711020300017

[116] List of available features — TSFEL 0.1.9 documentation. [Online]. Available: https://tsfel.readthedocs.io/en/latest/descriptions/feature_list.html