Received 15 April 2022; revised 30 May 2022; accepted 21 June 2022. Date of publication 29 June 2022; date of current version 15 July 2022.

Digital Object Identifier 10.1109/OJCOMS.2022.3185973

Training Deep Filters for Physical-Layer Frame Synchronization

SARUNAS KALADE, LOUISE H. CROCKETT, AND ROBERT W. STEWART

Department of Electronic and Electrical Engineering, University of Strathclyde, Glasgow G1 1XW, U.K.

CORRESPONDING AUTHOR: S. KALADE (e-mail: sarunas.kalade@strath.ac.uk)

ABSTRACT In this paper we demonstrate the application of Fully Convolutional Neural Network (FCN) for Frame Synchronization (FS) in bursty single carrier transmissions, commonly used in wireless sensor networks and Internet of Things (IoT) applications. Our approach shows greatly improved performance compared to noncoherent correlation-based methods under carrier phase and frequency offsets, especially for shorter preambles. Using a fully convolutional architecture allows the training of a deep filter, which we believe is more suited to signal processing tasks than more commonly used deep learning architectures with fully connected layers. In terms of deployment within a wider communications system, it could be treated similarly to a typical signal processing filter, which means it can be deployed to inputs of arbitrary length. Additionally, because the proposed model is composed only of convolutional layers, the entire model benefits from the weight sharing property of convolutional filters, and results in a greatly reduced memory footprint compared to that of similar models containing fully connected layers.

INDEX TERMS Deep learning, fully convolutional neural network, frame synchronization, Internet of Things.

I. INTRODUCTION

MODERN modulation and coding schemes are nearing the theoretical limit of what is achievable in Bit Error Rate (BER) [1]. However there still exist significant overheads in our communication systems in terms of the added redundancies and training symbols necessary for receivers to function optimally. This introduces necessary, yet undesirable inefficiencies in overall throughput, transmitter power and radio frequency spectrum usage. With more devices being connected than ever before to the Internet of Things (IoT), private 5G networks and wireless sensor networks, it is becoming increasingly important to cut down on the overhead that introduces these inefficiencies [2], [3].

Frame Synchronization (FS) [4] is a crucial step in recovering packet data from wireless transmissions. Usually this is accomplished by prepending a known preamble sequence to the transmitted data symbols and forming a packet, which the receiver can detect by correlating the known preamble with the captured signal. In practice this correlation is generally implemented by a matched filter, and then the output is compared against a threshold to determine whether the preamble has been detected. Adding preamble symbols to the data payload is an extra overhead and ideally we would want to keep that as small as possible without sacrificing throughput. Long preambles will greatly increase the probability of successfully detecting data packets, however this comes at the cost of the transmitter expending more power due to more bits having to be transmitted. Saving the transmitter from emitting as little as 2 bytes in the preamble can have significant power implications [5].

Machine Learning (ML), with Deep Learning (DL) in particular, has shown a great deal of success in recent years when applied to wireless communications problems in the PHY layer [6], such as channel estimation [7], Automatic Modulation Classification (AMC) [8], even learned communication via autoencoder with captured over-the-air data [9], [10], and many others. It is a technology that is seeping into a great number of applications in this field. However, not as much attention has been given to frame synchronization using this technology, and as our results show, there is much to be gained by applying DL techniques at the very edge of the radio physical layer. Perhaps the additional cost of implementing a neural network instead of a matched filter can now be justified, given the increasing

demand for radio spectrum, and the fact that the hardware used for computation is becoming more optimized for these types of neural network models [11], [12].

A. RELATED WORK

ML-based frame synchronization in burst-mode communications has been previously investigated in [13], where the problem is not tackled in a purely data driven approach, but rather ML is used in a way that supports existing analytical methods. In this type of approach a pre-processing stage takes place first, where useful synchronization metrics, such as cross-correlation, are extracted. Then the extracted features are used to train a neural network which cleans the correlation output and reduces the packet detection error rate of the overall system. This approach has a great advantage of lower resource usage than comparable DL models, while improving on classical correlation results, however because it relies on empirical knowledge it is not end-to-end trainable in the way a DNN (Deep Neural Network) could be.

Existing works on applying DL for FS typically treat it as a classification problem, which seems like a very natural fit for the data. A dataset for the FS problem will contain an input waveform and a desired response – a single peak at the time offset of the packet. This type of data maps perfectly into DL classification techniques, such as Convolutional Neural Networks (CNNs), which process an input image or waveform, and, ideally, output a single peak corresponding to the predicted class. Using DL for FS in this way was proposed in [14], where the authors approached it as a type of classification problem, and deployed a CNN to predict the correct starting index of a packet in a captured waveform of a fixed size, showing improved results under an AWGN channel. One drawback of this approach is that, once the network is trained, the input frame size must match the frame sizes it was fed during training, it also assumes that there will always be a single packet in a captured waveform. Additionally, the authors in [14] did not consider carrier phase offset in their experiments, which can cause significant overfitting problems, as will be demonstrated later in this paper.

Another example of DL being applied for FS was considered in [15], with a similar approach to [14], except the final layer of the CNN was a single output regression rather than a classification with as many outputs as the input waveform. A drawback of using this type of CNN is that in order to adapt to a changing communication protocol one will require retraining an entirely new network – for example, if the size of the payload according to the standard increased, requiring the receiver to operate on longer waveform captures. Additionally, because these models use fully connected layers, the network size scales proportionally to the input size, which could result in very large/wide models and become problematic when training on long input waveforms.

Recurrent Neural Networks (RNNs) are known to work well with data sequences of variable length – this has been explored in a wireless communications context in [16] using a sequence-to-sequence model for simultaneous modulation classification and demodulation of PSK (Phase Shift Keying) modulated symbols. A clear drawback shown in this work was that RNNs become increasingly difficult to train on raw I/Q samples as the input size increases, even when using cells like LSTM (Long short-term memory) that alleviate this difficulty [17].

Input size variance is not a problem unique to the wireless communications field; this is a big research problem in DL for computer vision, and many other disciplines. Advanced pooling methods, such as Spatial Pyramid Pooling (SPP) have been introduced in [18] and applied in very popular object segmentation models, such as R-CNN [19]. Pyramid pooling works well for classification problems where the number of outputs is set, and the input size is variable. However, this is not the case for FS in wireless communications – typically the output produced by a filter matches the number of samples it received. If we want to train a network to act as a filter, the output of the model will have to match the variable size of the input waveforms.

B. PROPOSED SOLUTION

In this paper, we propose that the FS problem be treated as a filter design challenge, where by using DL, we train a Fully Convolutional Neural Network (FCN) to act as a non-linear deep filter that can be applied to inputs of arbitrary length. In doing so, we can overcome the fixed size limitation of typical CNN approaches, while still being enabled by DL to train powerful non-linear models. We compare the FCN performance to (i) the standard correlation approach, (ii) one of the reference methods derived in [20], that improves upon standard noncoherent correlation by adding some corrective terms, and evaluate all three approaches for multiple preamble lengths. We also compare the FCN to a more common CNN architecture trained on the same dataset, and evaluate the generalization ability and complexity of the two DL methods, with correlation as a baseline. A high level overview of each approach can be seen in Figure 1.

The FCN approach shows improved detection rates over the correlation based methods, when faced with random carrier phase and frequency offsets – this is especially true for shorter preamble lengths. The proposed FCN method also generalizes and consumes less memory than the CNN approach, though it does not outperform it in every scenario. The ability to generalize to longer inputs without having to modify the architecture and re-train is a key advantage of the FCN architecture.

While the reduction of the FS overhead is one of the main concerns of this approach, we believe this architecture could be applied to a variety of applications in its future iterations. This architecture could be used for continuous AMC, similarly to image segmentation in computer vision, or to replace RNNs for learned matched filtering and demodulation as shown in [16]. Moreover having a FCN at the front of the receiver pipeline and processing every sample that has been perturbed by the wireless channel could be very

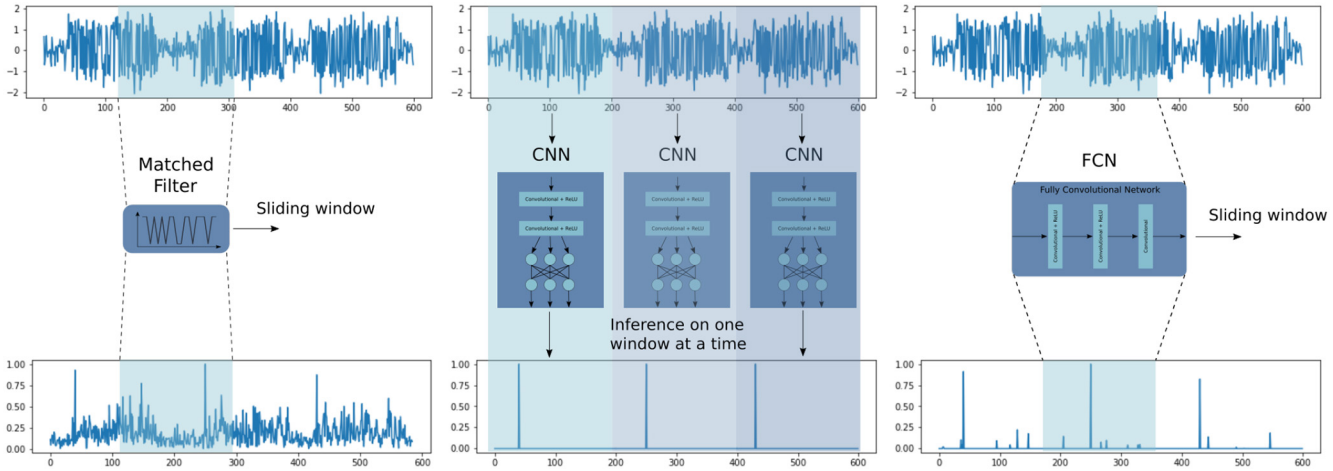


FIGURE 1. Overview of correlation-based and DNN-based frame synchronization models. a) The leftmost pipeline represents correlation, where a sliding matched filter produces an output, b) the middle figure shows the typical DNN approach by employing a CNN – because it can only be deployed on inputs of fixed size, the input has to be segregated into frames and fed into the model individually, c) the rightmost model is the proposed deep filter approach, where it is trained in the same DL methodologies as the CNN, but works in a sliding window approach, exactly like the basic correlation method – combining the best of both worlds.

valuable if other parts of the receiver are replaced by DNNs. In this case we can feed the features extracted by the FCN into one of these synchronizing or demodulating DNNs.

C. RESEARCH CONTRIBUTIONS

The main contributions of this paper are as follows:

- We introduce a fully convolutional neural network architecture, or deep filter, for the problem of FS. This allows us to take advantage of the learning algorithms from DL, while operating as a sliding window seen in signal processing filters, thus overcoming the inherent disadvantage of being locked to a fixed-sized input, which is present in typical DNNs.
- Classification-based CNNs must be trained with a dataset that represents every possible index of preamble start location in a captured waveform, much like in [14]. We demonstrate in our results that the FCN can generalize and accurately predict the time offsets after seeing only a fraction of these possibilities.
- An efficient method of training these networks for FS is presented that avoids the need of a large training dataset, usually composed of a variety of SNR levels [21], while the trained model maintains robustness to different noise levels.
- Using DL for FS allows the training of powerful models, that work best under challenging channel impairments. Our FCN architecture is shown to perform very well under carrier phase and frequency offsets, and for the case of short preambles in fading channels as well.
- A complexity analysis is carried out that demonstrates that the FCN architecture scales significantly better than typical neural networks containing fully connected layers, in terms of the number of parameters required to store these models in memory.

The rest of this paper is organized as follows: the system overview and proposed model are discussed in Section II, and Section III covers the details of training the deep filters for various preamble lengths. Numerical results are then discussed in Section IV, where the FCN architecture is evaluated with multiple preamble lengths, and channel impairments such as AWGN and carrier offsets. The complexity analysis is carried out in Section V, where the scaling of the FCN is compared to correlation and another neural network architecture containing fully connected layers. Finally, Section VI marks our conclusions and in Section VII we discuss possible future work.

Notations: Lower and upper case letters x and X denote scalars, bold faced lowercase letters $\mathbf{x}$ represent column vectors and bold faced uppercase letters $\mathbf{X}$ denote multi-dimensional tensors or matrices. For a vector $\mathbf{x}$, $\mathbf{x}(n)$ denotes the n^{th} element of the vector. And for a tensor $\mathbf{X}_{i,j,k}$ this operation extracts the $(i, j, k)^{th}$ element of the tensor. $\mathbb{R}$ and $\mathbb{C}$ denote real and complex sets of numbers. $\mathbf{x}^*$ is the complex conjugate of the vector $\mathbf{x}$. τ denotes a time offset. $\mathcal{L}$ represents a loss function, λ denotes a regularization factor that is applied to a loss function, and $\mathbf{w}$ denotes the vector of all the weights present in a neural network model. We refer to complexity measurements as Computational Complexity, denoted in equations as CC.

II. FRAME DETECTION USING FULLY CONVOLUTIONAL NEURAL NETWORKS

A. SYSTEM OVERVIEW

We consider bursty packet transmissions of Binary Phase Shift Keying (BPSK) modulated data. The packets, as illustrated in Figure 2, are made up of random data symbols of length N_d and prepended with a pseudo-random noise (PN) sequence preamble of length N_p. The total waveform length is determined as $M = N_p + N_d + N_t$, where N_t is the total non-symbol padding in the captured frame, for instance when $M = 200$, $N_d = 128$ and $N_p = 8$ bits respectively, $N_t = 64$. We then define τ as the time delay (or front

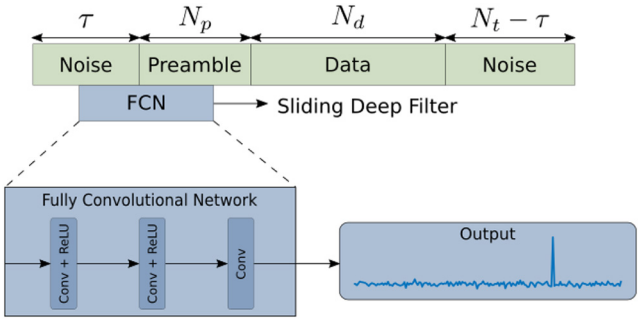


FIGURE 2. Convolutional Network Input Structure.

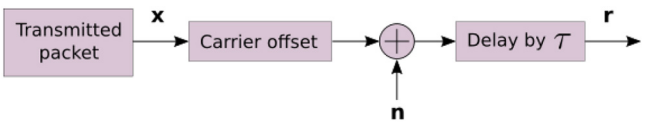


FIGURE 3. System Overview.

padding), and the back padding as $N_t - \tau$. Padding refers to zero valued samples representing the lack of signal at those particular time indexes. The effects on the transmitted frame are illustrated in Figure 3 – it gets perturbed by carrier phase and frequency offsets, and complex Additive White Gaussian Noise (AWGN) $\mathbf{n} \in \mathbb{C}^M$. We assume matched filtering and perfect symbol timing synchronization.

The goal of the frame synchronizer is to identify the location of the preamble symbol sequence at index τ of the received waveform $\mathbf{r} = [r_1, r_2, \ldots, r_M] \in \mathbb{C}^M$. The most common way of performing this detection is to use a non-coherent correlation detector (1) and estimate the peak $\hat{\tau}$ from the absolute cross correlation response,

$$c(\mathbf{r}) = \left| \sum_{i=0}^{M-1} \mathbf{r}_i \mathbf{p}_i^* \right|, \tag{1}$$

where $\mathbf{p} = [p_1, p_2, \ldots, p_{N_p}] \in \mathbb{C}^{N_p}$ is the preamble symbol sequence, M is the number of samples in the received signal and $*$ the complex conjugate. This detection method can be improved further by adding correction terms to (1), as shown in [20], and defined in (2). This method has shown improved FS performance under an AWGN channel with phase offset for PSK (Phase Shift Keying) modulation.

$$c(\mathbf{r}) = \left| \sum_{i=0}^{M-1} \mathbf{r}_i \mathbf{p}_i^* \right| - max\left[\sum_{i=0}^{M-1} \left| \mathbf{r}_i^I \right|, \sum_{i=0}^{M-1} \left| \mathbf{r}_i^Q \right|, \frac{1}{\sqrt{2}} \sum_{i=0}^{M-1} \left| \mathbf{r}_i^I + \mathbf{r}_i^Q \right|, \frac{1}{\sqrt{2}} \sum_{i=0}^{M-1} \left| \mathbf{r}_i^I - \mathbf{r}_i^Q \right| \right]. \tag{2}$$

In the following experiments, both methods will be used as baselines when evaluating the FCN performance.

B. FULLY CONVOLUTIONAL NEURAL NETWORKS

FCNs are most prominently used for dense pixel-wise classification in image segmentation [22]. This type of network is constructed entirely from convolutional layers, pooling layers and activation functions. Since they have no fully connected layers they are not constrained to a fixed sized input, as is the case with feedforward networks, which work by performing matrix multiplication of their learned weights by the entire input at once. The convolution operation is ubiquitous in signal processing, which makes it a natural fit for the problems of filtering and recovering digitally modulated waveforms, especially in time-varying channels, where defining analytical models is difficult. A single convolutional layer output of the i^{th} filter at input lags of j, k, is defined as

$$y_{i,j,k} = \sum_{c,m,n} \mathbf{X}_{c,j-m,k-n} \mathbf{W}_{i,c,m,n} + \mathbf{b}_i, \tag{3}$$

where $\mathbf{W} \in \mathbb{R}^{K \times C \times L \times H}$ is the weight tensor, K is the number of filters in the convolutional layer, C is the number of input channels, L and H are the length and height of the 2-D filters respectively, and $\mathbf{X}$ is the input tensor of length M. Additionally, a bias term $\mathbf{b}$ is added for each filter. The output is often passed through a non-linear activation function, such as the Rectified Linear Unit (ReLU), defined as $max(x, 0)$.

The FCN presented in this work is composed of 3 convolutional layers, the parameters of which are shown in Table 1. The first two layers are followed by a ReLU function, while the last one remains linear to extend the output range to negative and large positive values. The design does not use any pooling layers, and the padding of each convolutional layer is set to match the input size of the previous layer, ensuring that the final output length N is of the same length as the input.

Since the largest preamble size tackled in this work is 32 bits long, we made sure that the length of the filters of the first layer were at least large enough to learn to correlate this preamble sequence. We found that, for our network and dataset, using more than 16 filters in the first two layers does not improve performance. It is important that the last layer contains only a single filter, which then transforms the intermediary feature maps produced by the second convolutional layer into a single-dimensional vector that we can easily interpret in the same way as a correlation output.

Selecting good parameters (number of filters, size of kernels, number of layers) is a very difficult problem spanning the field of ML. We have not done an exhaustive search in the parameter space, but empirically determined a network that performed well. It is very likely that even better results could be achieved by experimenting with different DNN parameters like activation functions or numbers of layers.

III. TRAINING

Two neural network architectures (illustrated in Figure 4) and the training data required for each are discussed in this section. The CNN model is trained for comparison using typical methods for training an ML classifier. The individual layer settings for each network are characterised by the parameters given in Tables 1 and 2. The proposed FCN model utilizes

TABLE 1. FCN parameters for input length *N*.

Layer	Parameters	Output Shape
Input		$2 \times N$
Conv + ReLU	Kernel = (16, 1, 35, 2)	$16 \times 1 \times N$
Conv + ReLU	Kernel = (16, 16, 35, 1)	$16 \times 1 \times N$
Conv	Kernel = (1, 16, 25, 1)	$1 \times 1 \times N$

TABLE 2. CNN parameters for input length 200.

Layer	Parameters	Output Shape
Input		2×200
Conv + ReLU	Kernel = (16, 1, 35, 2)	$16 \times 1 \times 200$
Conv + ReLU	Kernel = (16, 16, 35, 1)	$16 \times 1 \times 200$
Flatten		1×3200
FC + ReLU	Size = 128	1×128
FC + Softmax	Size = 200	1×200

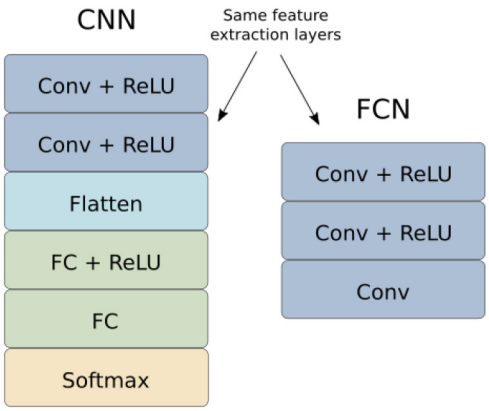


FIGURE 4. CNN and FCN model architectures.

the same dataset parameters and preprocessing steps as the CNN, but is trained as a regressor, rather than a classifier.

Training of the models consists of optimizing the weights of the neural network models by feeding them training examples, which are composed of simulated transmissions of BPSK modulated packets at random delay intervals, and corresponding desired responses (e.g., a peak at the frame start index). Each training example consists of a waveform that is 200 samples long and a response of the same length.

The models were trained on BPSK packets, consisting of 128-bit payloads with 8, 16 and 32-bit PN synchronizing sequences, all affected by AWGN, random time delay $\tau \in \{0, 100\}$, random phase offset $\phi \in \{-\pi, \pi\}$ and CFO between 0–10kHz. We found that an effective training strategy was to generate data at a relatively high SNR, and apply regularization, rather than train on overly noisy data or ranges of SNRs.

If trained with an SNR that is too high, the network will not learn to be robust to noise. However if the SNR is too low, our model will just learn noise (garbage in – garbage out). If a baseline is available, such as the correlation performance, we found through experimentation that a good guideline is to stay within the range of SNRs where the baseline accuracy is at 60-90%, as illustrated in Figure 5. A good starting point for training the FCN model is to generate data at the SNR where the baseline model reaches around

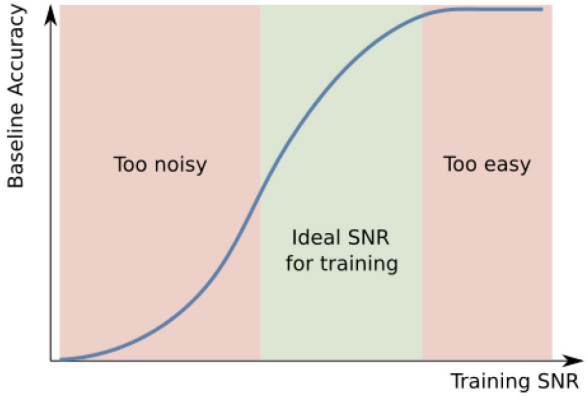


FIGURE 5. Finding the best training dataset SNR.

90% of its maximum accuracy. However, if the baseline never achieves very good results, as is the case with extremely short preambles, it is a good idea to start with a high SNR value of around 15dB, then incrementally reduce the SNR until performance stops improving. A similar strategy of initially training on a high SNR and progressively introducing more noise has been presented in [23].

A. FCN

In order to evaluate how well the network fits the training data, an appropriate loss function should be selected. The Mean Squared Error (MSE) loss works well for regression tasks, where the outputs can take on ranges of values, rather than discrete categorical outcomes (where a log loss would be more appropriate). The MSE loss function used to train the networks is defined as

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{n=0}^{N} \left| \mathbf{y}(n) - \hat{\mathbf{y}}(n) \right|^2, \tag{4}$$

where N is the number of training examples in a batch, $\mathbf{y}(n)$ is the desired response of the system (ideally all 0 values, except for the preamble index), and $\hat{\mathbf{y}}(n)$ is the response predicted by the FCN model. This is generally a good choice for regression tasks where the output is not discrete. While basic, it turns out that this loss function is more than good enough to produce a very desirable output function for this problem. Using MSE to optimize the deep filters also avoids the limitation of having examples of only 1 packet capture in the training set, as seen in other softmax-based classification methods discussed earlier. This enables future expansion of the architecture with various multi-packet configurations.

Regularization is important to avoid overfitting the model to the training dataset – this is desirable as it enables networks to actually perform well on cases they have not seen before [28]. We found L2 penalization (also known as weight decay [29]), to be a simple and effective method of regularizing the models. Weight decay can be implemented

by adding a regularization term to (4):

$$\mathcal{L}_{reg}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_{n=0}^{N}|\mathbf{y}(n) - \hat{\mathbf{y}}(n)|^2 + \frac{1}{2}\lambda\sum_{i=0}^{N_w}\mathbf{w}_i^2, \qquad (5)$$

where λ is the regularization factor, N_w is the number of weights, and $\mathbf{w}$ is the weights vector that contains all the weight values of the neural network. In this case λ is a hyper parameter, as it is not learned, and has to be manually adjusted. The regularization term penalizes the network if the weights of the convolutional filters grow too large, preventing it from developing a bias towards a single input feature.

For FCN validation we used simulated data with higher SNR than the training set, with no carrier offsets. This is different from the usual method where the validation dataset is obtained by slicing off a portion of the training set. The goal of this approach was to ensure that the resulting networks are capable of generalizing to milder channel conditions and are not only functional when affected by very specific channel impairments that may have been over-represented in the training set. A variety of combinations for training and validation datasets can be deployed, and finding the optimal strategy is an ongoing research problem to be considered for future work.

B. CNN

Instead of having the last convolutional layer compress all the previous feature maps into a single output, the CNN uses a flattening layer to convert all previous feature maps into a vector that can be fed into fully connected layers. The final layer is another fully connected layer that matches the input length of the captured waveform, followed by a *softmax* activation function. The softmax activation for the j-th neuron of the output defined in (6) is given by

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \qquad (6)$$

where σ is the softmax activation function, $\mathbf{z}$ represents the outputs of the final fully connected layer and K is the number of outputs. A convenient property of the softmax function is that the output vector is transformed into a probability distribution where the sum of all outputs equals 1. This has a useful effect of being more easily interpretable, and makes it easier to work with certain logarithmic loss functions.

A very commonly applied loss function for classification tasks after the softmax normalization is categorical cross-entropy loss [24], or log loss. Similarly to (5), we define the cross-entropy loss as $\mathcal{L}_c$ with an L2 penalty applied in (7).

$$\mathcal{L}_c(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_i \mathbf{y}_i \log \hat{\mathbf{y}}_i + \frac{1}{2}\lambda\sum_{i=0}^{N_w}\mathbf{w}_i^2. \qquad (7)$$

As in previous equations, $\mathbf{y}$ and $\hat{\mathbf{y}}$ represent the labels and model predictions respectively, whereas $\mathbf{w}$ represents the weights of the network. This has been the choice of loss function for similar previous works [14]. Applying L2 penalization is similar to (4), where the regularization term λ and weights of the neural network are added to the main loss function.

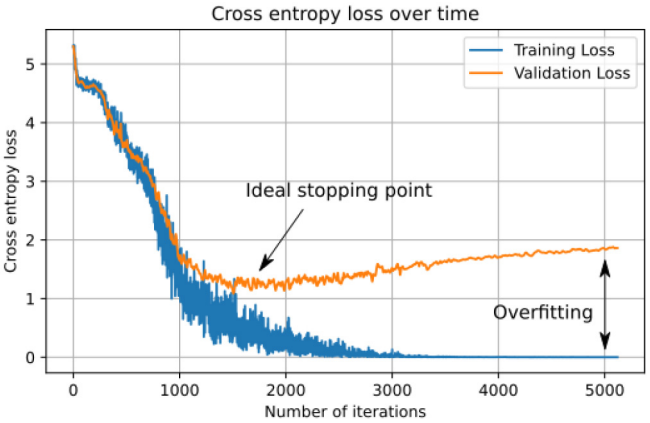


FIGURE 6. Example of CNN overfitting to the training set. With early stopping, the model should be saved at the ideal stopping point, where validation loss is lowest.

TABLE 3. Training hyper-parameters.

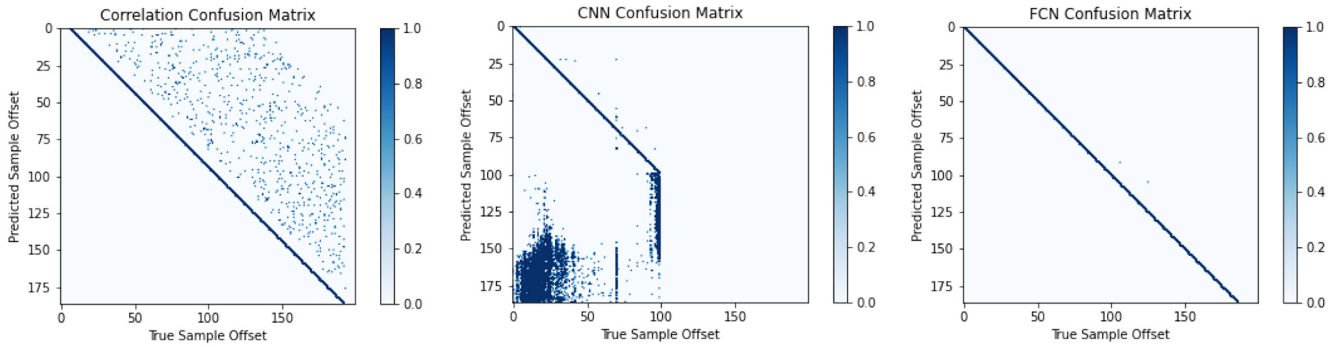
Parameter	FCN	CNN
Optimizer	ADAM	ADAM
Loss function	MSE	Cross entropy
Num. training examples	8192	32768
Num. validation examples	512	512
Batch size	32	32
Num. epochs	30	30
Learning rate (α)	0.001	0.001
Regularization Factor (λ)	0.01	0.01
Training SNRs (dB) for 8, 16, 32 bit preambles	10, 5, -5	10, 5, -5

Over time large DNNs can start memorizing the training set, including the noise of each sample, which results in a loss of ability to generalize to unseen inputs. To combat this, validation loss monitoring and early stopping are often used to improve model robustness. Typically an additional loss metric is evaluated on a separate validation set (which is not used to update the weights of the DNN) and training is stopped once the validation loss stops improving, as shown in Figure 6.

C. TRAINING PARAMETERS

Every FCN network was trained on 8k training examples, and every CNN on 32k. The CNN had to be trained on 4 times as much data to match the performance of the FCN, because it requires every classs instance (or offset) to be represented in the dataset in order to be able to make predictions on the training dataset window of 200 samples. This is one of the downsides of using a classification method such as a CNN, rather than a filter-like regression.

The training parameters for both networks are summarized in Table 3. For both networks the ADAM [25] optimizer was used in PyTorch [26]. The training batch size was 32, learning rate $\alpha = 1e^{-3}$, regularization factor $\lambda = 1e^{-2}$, and training lasted a maximum of 30 epochs, with early stopping employed to avoid overfitting by monitoring the validation



(a) Correlation (b) CNN (c) FCN

FIGURE 7. Confusion matrices of the different frame synchronization models. a) shows the classifications for the baseline correlation method, where failures are only seen where preamble repeats in the payload. b) the CNN performs well in the range of offsets it was trained for, but because it has never seen past $\tau = 100$ it cannot generalize to longer offsets. c) the FCN was trained on the same dataset as the CNN, but because it acts as a filter, using a sliding window approach, it can still generalize to new offsets. It also outperforms correlation, because it is less likely to misclassify a preamble found inside the payload, resulting in a cleaner diagonal.

loss. For 8, 16 and 32-bit preamble lengths the training SNR levels were 10dB, 5dB and −5dB respectively – requiring roughly 5dB less for each additional byte in the preamble sequence.

D. DATASET CONSTRAINTS OF CNNS

One issue that is very commonly encountered in classifiers, such as the CNN, is that sometimes classes in a dataset can be under-represented, stifling the training and causing poor test accuracy. We often see this in other fields as a class imbalance problem [27]. In the field of wireless communications, we have rich software libraries that allow us to generate large quantities of data, so this issue does not manifest as a class imbalance. However, for the case of frame synchronization being approached as a classification problem, especially when long input sizes are considered, this can cause complications in training and deployment from a scaling perspective. For long input frames we will require more data to represent all possible offsets within a captured waveform, which adds to complexity, required network size and training time. The FCN architecture does not have this limitation, because it acts as a filter and can generalize to longer sequences having been trained on much smaller inputs.

An example of the effects missing class labels can have on model performance is shown in Figure 7, where the frame synchronization method confusion matrices are plotted. The confusion matrix is a useful tool for evaluating classification models by giving insight into which classes are getting misclassified, by helping to identify specific cases, rather than just looking at overall accuracy or DER. Ideally the result should be a perfect diagonal, meaning that each prediction corresponds to the correct label. In this example we trained both the CNN and FCN on a dataset where the waveform input lengths were 200 samples, and only had packet time offsets represented in the range of $\tau \in \{0, 100\}$, meaning that there were no examples where a packet preamble was intercepted past a time offset of $\tau = 100$. The confusion matrices were obtained by testing on the entire range of possible time offsets $\tau \in \{0, 184\}$ where the full preamble can be found. Looking at the CNN confusion matrix in Figure 7b, we can observe that it cannot generalize to offsets that have not been represented in the training set. Shown in Figure 7c, the FCN acts in a very similar way to standard correlation (Figure 7a), even though it was subjected to the same constrained dataset as the CNN. In fact, as mentioned earlier, the CNN had to be trained on 4 times as much data (32k waveforms) to match the performance of the FCN (8k waveforms) in this narrow range of offsets.

IV. SIMULATION RESULTS

FCN models for each preamble length were trained offline using simulated data, then compared against analytical methods in terms of Detection Error Rate (DER) under different SNRs with random carrier phase offset and frequency offset. In each simulation, 128-bit payload packets prepended with 8, 16 and 32-bit preambles were used for frame detection. Robustness to Carrier Frequency Offset (CFO) was evaluated by comparing both DL models to the correlation based baseline approaches, showing the maximum tolerance of each method.

The FCNs showed improved performance over correlation based methods in terms of DER, especially when detecting shorter preambles, and generalized well to longer capture lengths without having to be retrained. FCNs and CNNs showed similar tolerance to CFO, as long as the CNN was constrained to the τ distribution that was used in the training set, but both heavily outperformed the baseline models. The FCN was also evaluated under different flat and multipath fading channels, and showed consistently better results for the shorter preambles, however for the 32-bit preamble we saw little to no improvement.

In most of the DER experiments we saw a high error floor for the 8-bit preamble sequences of the correlation based methods, similarly, but not as extremely for the 16-bit preambles. This can be explained by the fact that for very short preambles, the likelihood of them appearing in the payload is

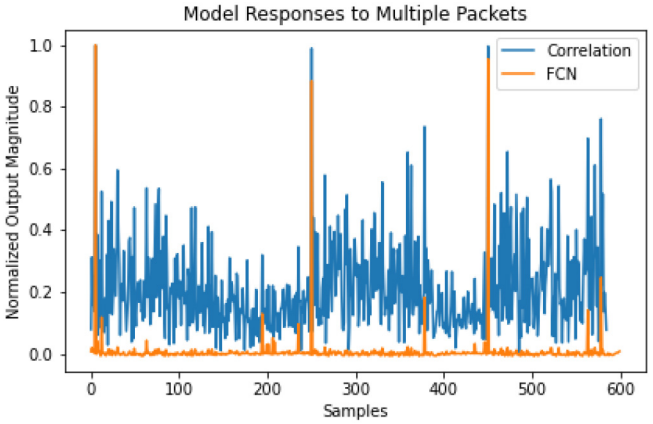


FIGURE 8. Correlation and FCN inference on a 600 sample input containing multiple packets.

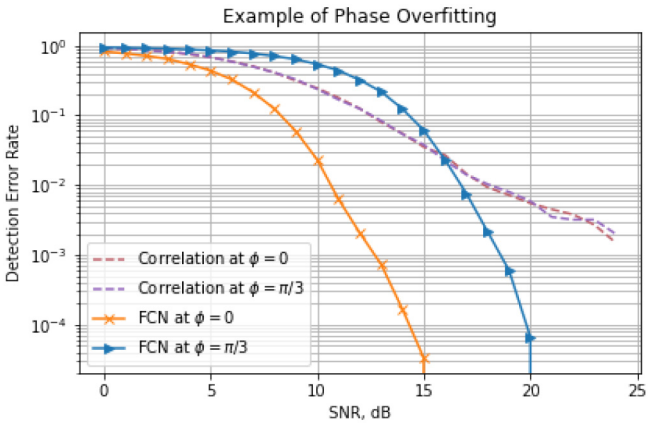


FIGURE 9. DER comparison under different phase offsets of an FCN trained on carrier phase offset $\phi = 0$.

quite high, resulting in false positives from the correlation output. In the following experiments we will see that the FCN approach does not have the same error floor because it learned to use more features than just the correlation to perform the detection.

A. ROBUSTNESS TO DIFFERENT CAPTURE LENGTHS

One of the main strengths of using FCNs instead of other neural network architectures is the ability to deploy these models to data of arbitrary length L_{test}, even after training on relatively short waveforms of length L_{train}, where $L_{train} < L_{test}$. In Figure 8 we compare the FCN output against the output of a correlation detector on waveforms containing multiple packets. The test waveform contains three 128-bit packets of BPSK modulated data, with a 16-bit preamble – a total capture length of $L_{test} = 600$ samples.

Even though the FCN was trained on waveforms containing only a single packet with captures of $L_{train} = 200$ samples, it is capable of generalizing and making predictions on extended data configurations, previously not seen in the training set. In Figure 8, we observe that the neural network performs a significant amount of denoising when compared to a simple correlation output, which is helpful when minimizing the number of false positives and finding good detection threshold values.

B. AWGN AND PHASE OFFSET

It is important to consider random phase offsets when obtaining these results to ensure that the trained FCN is not overfitting to a particular carrier phase. DNN models are highly susceptible to overfitting (this is why regularization, as discussed in Section III, is important), however no amount of regularization can replace a good training dataset representative of the rich array of channel conditions that the model may encounter.

In Figure 9 an FCN trained on a dataset with no phase offsets is evaluated at offset $\phi = 0$ and $\phi = \pi/3$. Because noncoherent correlation is not affected by phase offsets

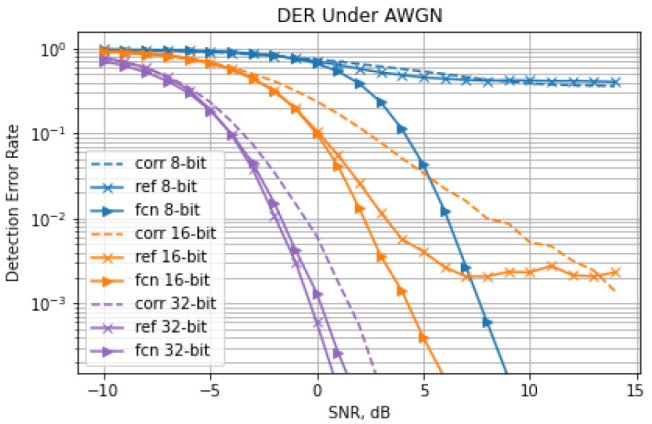


FIGURE 10. DER under AWGN, with random phase offsets, for preamble lengths of 8, 16 and 32 bits.

it is only evaluated at a single phase. We see that the network performance at a fixed $\phi = 0$ is stellar, and actually outperforms the baseline. However, if the carrier phase changes, which is an unavoidable real world occurence, the performance of the FCN deteriorates greatly when introduced to a phase it has not been trained on. For this reason, all networks in this paper have been trained on random distributions of phase offsets.

For the AWGN DER results, the FCN models are tested at an SNR range of $\{-10,15\}$, with a phase offset ϕ randomly sampled from the range $\{-\pi, \pi\}$. The results are compared with baselines calculated by the standard correlation method and the method described in [20], and these baselines are denoted in Figures 10-12 as 'ref'.

It is evident that the correlation methods have an error floor no matter how high the SNR. This is most clearly seen in the 8 bit sequence correlation results. The FCN performs much better in this scenario because it has learned to not only perform correlation, but also to undertake other useful tasks, such as energy detection, for extracting a variety of features which assist in the detection of a packet.

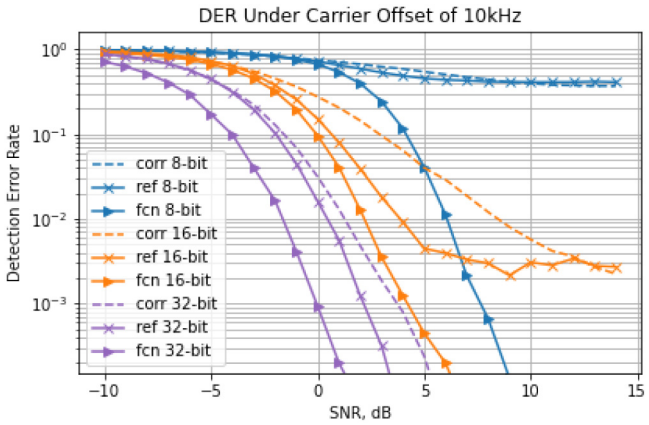


FIGURE 11. Detection error rates with carrier frequency offset CFO = 10kHz, for preamble lengths of 8, 16 and 32 bits.

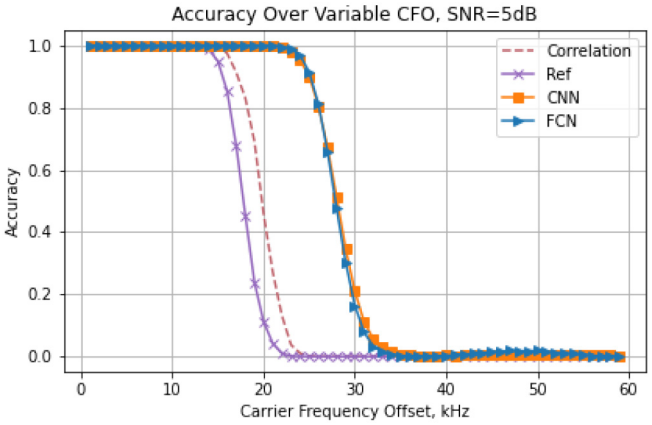


FIGURE 12. Packet detection accuracy for a 32-bit preamble as CFO increases.

We can see a general improvement over correlation methods, but the gains of using an FCN diminish as the preamble length increases – we observe little to no advantage at 32 bits. However AWGN is a relatively simple channel to overcome. The advantage of using DL is when faced with more challenging channel impairments.

C. DETECTION UNDER CARRIER FREQUENCY OFFSET

While carrier phase offset is very important for training DNNs for FS. It has little to no effect on the performance of correlation methods, as shown in Figure 9. CFO does affect correlation methods, which typically would necessitate more synchronization steps. Frequency offsets are also typically unavoidable in realistic communications channels, which is why it is necessary for the FCN model to be able to adapt to these types of impairments.

For CFO DER tests, the FCN models are tested at an SNR range of $\{-10{,}15\}$, with a phase offset ϕ randomly sampled from the range $\{-\pi, \pi\}$, as well as an added Carrier Frequency Offset (CFO) of 10kHz (which corresponds to the highest offset in the training set), with a sample rate of 1MHz. We observe that the FCNs show robustness to phase and frequency offsets, and the performance is especially improved when compared to the analytical methods for the shorter preamble lengths.

While an 8-bit preamble may not be enough for detection using correlation based methods, the FCN model shows similar performance to that which would be achieved using a 16-bit preamble using standard correlation.

D. MODEL SENSITIVITY TO CFO

In order to further evaluate how robust the FCN approach is to CFO, both baseline models and neural network models are compared over a range of CFOs. For this experiment we choose the FCN trained on data containing 32-bit preambles, and the same baseline correlation methods presented earlier.

To better visualize the FCN's robustness to CFO, the accuracy of each model is computed at frequency intervals of 1kHz. In this case the accuracy is simply the inverse of the DER. The results are shown in Figure 12.

It is clear that both deep learning approaches (CNN and FCN) maintain a 100% detection accuracy rate from no CFO to 24kHz, whereas the traditional approaches start losing accuracy at a CFO of 15kHz. This shows that any DNN should be able to cope with CFO if the training data includes examples with this impairment. While the FCN does not outperform the CNN in this instance, it is still a good result, because the FCN was trained on a much smaller dataset and is a more flexible architecture due to its filter-like nature.

E. FADING CHANNEL

We further analyze the model performance with a flat fading channel following a Rayleigh distribution, modelling a non-line-of-sight (NLOS) link. In Figure 13, similarly to the previous results, we see that for the shortest 8-bit preamble configuration the DER improvement is the greatest, and at very high SNR even exceeds the 16-bit correlation performance, whereas the correlation method seems to reach an error floor. Similarly, the FCN performance on the 16-bit preambles also shows improvement at high SNR, when compared to the reference methods. However the performance on the 32-bit preamble is comparable across all methods.

It is important to note, however, that the FCNs were not explicitly trained on this type of fading channel and are generalizing purely based on the training data that contained carrier phase and frequency offset impairments.

F. MULTIPATH FADING

Finally, we investigate an even more challenging, multipath channel. This channel follows a Rician distribution with relative time offsets of [0, 5e-6, 10e-6] seconds and average channel gains [0dB, −3dB, −6dB], with a sample rate of 1MHz. Consistently with the previous tests, based on the results in Figure 14, we can see that the most marked performance improvement is in the shorter preamble lengths. Interestingly, in this instance the 32-bit FCN DER performance was actually worse than the reference methods. Another observation we can make from these results

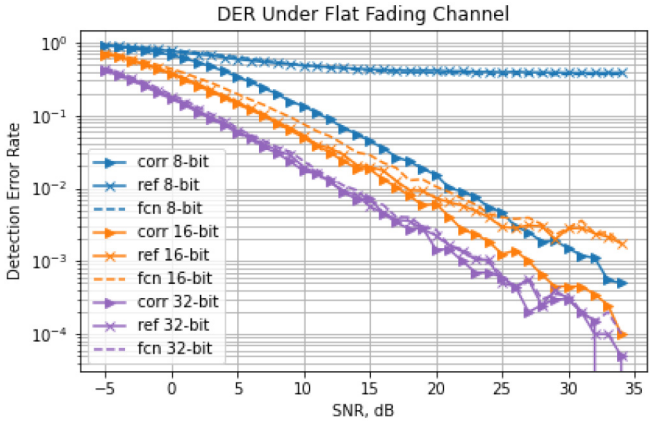


FIGURE 13. Detection error rates under a Rayleigh fading channel.

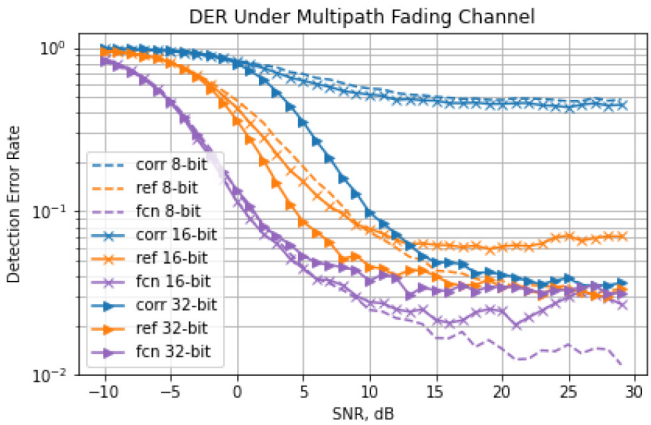


FIGURE 14. Detection error rates under a multipath fading channel.

is that all 3 FCN results converge to the same error floor as SNR increases. These unexpected results can likely be explained by the fact that the FCNs are unaccustomed to multipath components at high SNR because they have not encountered such effects in the training set.

Note that the DER values are above $10e^{-2}$, which is not practical in most communications systems. However, even though the DERs are high, the FCN approach still shows a considerable improvement over the correlation methods at lower preamble lengths. This is a promising result for future exploration of FCNs in more challenging multipath channels.

V. COMPLEXITY ANALYSIS

Model complexity, especially in deep learning, is difficult to quantify because of a variety of available hardware implementations and optimizations that can be incorporated. However, some aspects can be evaluated with certainty, such as the number of parameters required to represent the neural network architecture. In order to implement a neural network on hardware, the computational complexity and memory requirements of such a model must be considered. Depending on the application, the chosen architecture can consume a large amount of on-chip memory just to store all of the parameters (or weights).

In this section we compare the proposed FCN architecture presented in this paper, with a comparable CNN architecture similar to that seen in the literature [14], and a traditional matched filter consisting of 32 weights. Each method is evaluated at input lengths of 200 and 600 samples. For correlation and FCN, the evaluations may simply be re-run for the new input length, since they are implemented as filters. However it is important to note that for any approach containing fully connected layers, such as the CNN, the network must be re-designed to a new input-output sizing and re-trained.

Performing a fair comparison between model architectures is a challenging problem, because optimization steps and loss functions can differ between approaches. In our analysis best efforts were made to produce a CNN model that matches the FCN performance as closely as possible for target lengths, with the minimum number of weights. However, it cannot be guaranteed that either is the optimal solution.

Neural network Computational Complexity (CC) in flops (floating point operations) for the main layer types – convolutional and fully connected – can be determined using equations (8) and (9). The complexity of a convolutional layer is defined by

$$CC_{conv} = (2 * (c * l * h)) * k * W * H, \tag{8}$$

where CC_{conv} is the computational complexity of a convolutional layer in flops, c, l and h, are the convolutional layer filter parameters of input channels, kernel length and height respectively, k represents the number of filters, and W and H are the width and height of the input signal, or feature map. The complexity of a fully connected layer CC_{fc} is calculated using

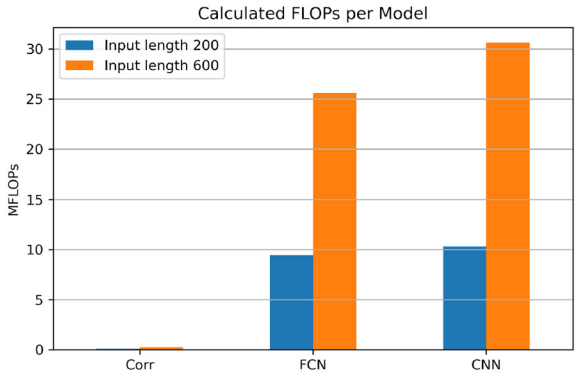
$$CC_{fc} = 2 * w_{in} * w_{out}, \tag{9}$$

where the flops of a fully connected layer are the multiplication of input vector length w_{in} and number of output neurons w_{out}. Note that this does not take into account other parts of the architectures, such as activation functions. The flops associated with correlation can simply be calculated by

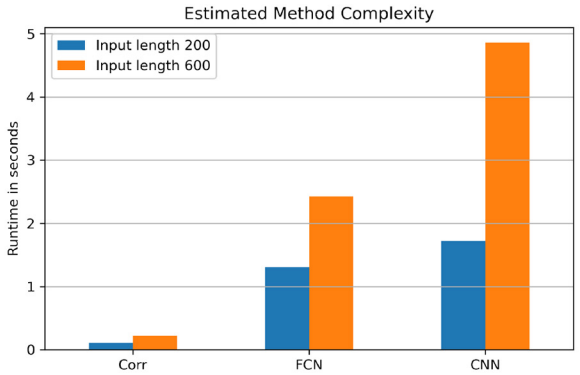
$$CC_{corr} = 2 * c * l * W * H, \tag{10}$$

where c, l are the number of channels and length of the filter, and W and H are the width and height of the input signal. The matched filter only needs to be evaluated once, since there are no layers. Substituting the parameters of the evaluated models from Tables 1 and 2 into the equations, we report the calculated flops in Figure 13(a). While this does not perfectly reflect the realities of hardware, it provides a good approximation of the computational costs for these types of architectures.

Furthermore, we estimate the computational complexity of each neural network method, as well as the traditional correlation for a baseline, in Figure 13(b), by evaluating them on a single core of an AMD Ryzen 5 CPU, and recording the runtime to execute inference on 10,000 waveforms. Obviously,



(a) Calculated CC in flops for each FS method at input lengths of 200 and 600 samples (excluding activation functions).



(b) Estimated computational complexity for each FS method based on CPU runtime.

FIGURE 15. Complexity analysis.

both DNNs are significantly more costly than the simple correlation implementation, as they involve multiple convolutions and matrix multiplications. The more interesting comparison is between the two neural network architectures. In terms of calculated flops, the two are comparable for input lengths of 200 and 600 samples. However, comparing estimated complexity based on CPU runtime, the CNN appears to be nearly twice as costly when considering the longer 600 sample input lengths – this can partly be explained by additional activation functions, such as softmax, and the number of output neurons scaling in accordance with the input size.

The computational complexity is comparable between the two neural network models, because they both involve large convolutional operations, with an increase in computation for a larger input length. However, in terms of required parameter storage, as shown in Figure 16, the FCN is significantly less costly than the CNN – this is due to the CNN containing multiple fully connected layers, which usually result in very large parameter sizes. In a fully connected network each input value must be multiplied by each weight of the layer – there is no weight sharing in these layers, which is an important property of convolutional networks when considering implementation [30]. Furthermore, while the FCN is composed of only convolutional layers and is essentially a deep filter, increasing the input size to the network actually does not require any additional weights. The proposed FCN architecture, much like a matched filter, can be applied to arbitrary signal processing flowgraphs without any re-tuning or re-training.

Model complexity in its various forms is a very important metric to consider, especially when taking into account edge applications such as wireless receivers. In this analysis we have demonstrated that while both DNN approaches impose an implementation penalty when comparing to a standard matched filter, the FCN model is less computationally intensive than an equivalent CNN operating on the same input data. Furthermore, the FCN is a highly advantageous choice when considering storage requirements. Using a CNN

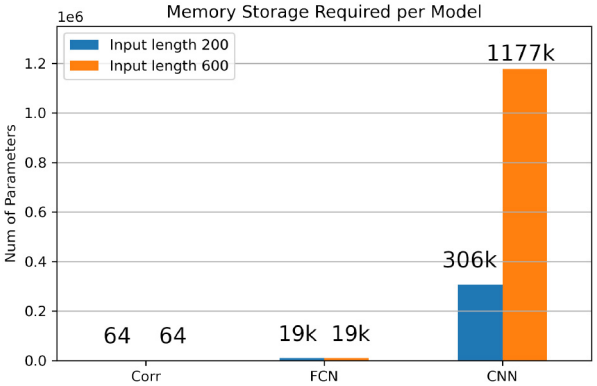


FIGURE 16. Number of parameters for each evaluated model, showing how the size of the network scales with fully connected layers vs. only convolutional layers.

the number of weights required to represent the network will scale with the size of the input it has to operate on. Regardless of the input size, an FCN, acting as a filter, will always require a fixed amount of memory to store its weights.

While we have shown that applying FCNs is advantageous over CNNs for the problem of frame synchronization, the computational cost is still orders of magnitude higher than the traditional correlation approaches. There are existing techniques like pruning and quantization for embedded DNNs that we did not consider in this work, which could significantly reduce the complexity here [31]. Also, since to the best of the authors' knowledge, this is the first paper demonstrating FCNs being applied to frame synchronization. Having proved the concept we anticipate that this paper may stimulate further implementation-based research. As a result, future implementations should become much more hardware efficient as the research in this field progresses.

VI. CONCLUSION

In this paper we have demonstrated the use of FCNs for radio physical layer frame synchronization. A key advantage of the

FCN model over other ML architectures is that this type of network can be applied as a deep filter on input sequences of arbitrary length, while not being explicitly trained on such inputs. Simulation results show that the FCN approach can drastically improve the DER of frames under CFO when compared to standard analytical models, such as the methods based on noncoherent correlation. The performance increase is especially prominent when short preambles of just a couple bytes are considered.

Reducing the preamble length can be beneficial in IoT applications, where transmissions can be short and bursty, with the transmitters of sensors having to carefully budget the transmission power. Allowing a more complex DL-based receiver would reduce the synchronization overheads, and as a result save energy on the transmitter side of the communication system. The reduction of overheads also reduces the usage of radio spectrum, which is becoming an increasingly precious natural resource.

VII. FUTURE WORK

Finding the correct optimization parameters for these networks is still an ongoing research area, even in the broader field of ML. In communications signal processing we have additional domain-specific concerns of SNR, phase offsets, channel effects, hardware induced nonlinearities, etc. Ideally a trained model would be robust to the many transformations that realistic communications signals are subject to – producing a good training dataset to achieve this task becomes an optimization problem of its own. We have shown the importance of producing a good training set by demonstrating the dataset constraints that CNNs have and the effects of phase offset overfitting while training the FCN.

ML for wireless communications, especially in the domain of frame synchronization, is still in its infancy with ripe opportunity for further exploration into training methodologies and model architectures. There are some interesting possibilities of using an FCN like this to also act as a feature extractor for other DNNs in the signal processing pipeline, because it can be one of the very first processing blocks of a receiver. This includes best training data selection, architecture optimization, as well as hyper parameter tuning (such as the learning rate, optimizer selection, loss functions, different regularization methods, etc.). While we trained the same model for different preamble sizes, it could be possible to reduce the memory footprint of the models even further by optimizing the architecture based on anticipated preamble length.

ACKNOWLEDGMENT

The authors would like to thank the MathWorks and AMD-Xilinx for their support on this work.

REFERENCES

[1] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001, doi: 10.1109/18.910578.

[2] M. Agiwal, A. Roy, and N. Saxena, "Next generation 5G wireless networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 3, pp. 1617–1655, 3rd Quart., 2016, doi: 10.1109/COMST.2016.2532458.

[3] F. A. Aoudia and J. Hoydis "Trimming the fat from OFDM: Pilot- and CP-less communication with end-to-end learning," 2021, *arXiv:2101.08213*.

[4] J. Massey, "Optimum frame synchronization," *IEEE Trans. Commun.*, vol. TCOM-20, no. 2, pp. 115–119, Apr. 1972, doi: 10.1109/TCOM.1972.1091127.

[5] B. Bloessl and F. Dressler, "mSync: Physical layer frame synchronization without preamble symbols," *IEEE Trans. Mobile Comput.*, vol. 17, no. 10, pp. 2321–2333, Oct. 2018, doi: 10.1109/TMC.2018.2808968.

[6] T. O'Shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Trans. Cogn. Commun. Netw.*, vol. 3, no. 4, pp. 563–575, Dec. 2017.

[7] H. Ye, G. Y. Li, and B.-H. Juang, "Power of deep learning for channel estimation and signal detection in OFDM systems," *IEEE Wireless Commun. Lett.*, vol. 7, no. 1, pp. 114–117, Feb. 2018.

[8] Y. Wang, M. Liu, J. Yang, and G. Gui, "Data-driven deep learning for automatic modulation recognition in cognitive radios," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 4074–4077, Apr. 2019, doi: 10.1109/TVT.2019.2900460.

[9] S. Dörner, S. Cammerer, J. Hoydis, and S. T. Brink, "Deep learning based communication over the air," *IEEE J. Sel. Topics Signal Process.*, vol. 12, no. 1, pp. 132–143, Feb. 2018.

[10] T. J. O'Shea, T. Roy, N. West, and B. C. Hilburn, "Physical layer communications system design over-the-air using adversarial networks," 2018, *arXiv:1803.03145*.

[11] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, 2017, pp. 1–12.

[12] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey of machine learning accelerators," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, 2020, pp. 1–12.

[13] C. Qing, W. Yu, B. Cai, J. Wang, and C. Huang, "ELM-based frame synchronization in burst-mode communication systems with nonlinear distortion," *IEEE Wireless Commun. Lett.*, vol. 9, no. 6, pp. 915–919, Jun. 2020, doi: 10.1109/LWC.2020.2975651.

[14] E.-R. Jeong, E.-S. Lee, J. Joung, and H. Oh, "Convolutional neural network (CNN)-based frame synchronization method," *Appl. Sci.*, vol. 10, no. 20, p. 7267, 2020. [Online]. Available: https://doi.org/10.3390/app10207267

[15] T. J. O'Shea, K. Karra, and T. C. Clancy, "Learning approximate neural estimators for wireless channel state information," 2018, *arXiv:1707.06260*.

[16] S. Kalade, L. Crockett, and R. W. Stewart, "Using sequence to sequence learning for digital BPSK and QPSK demodulation," in *Proc. IEEE 5G World Forum (5GWF)*, 2018, pp. 317–320, doi: 10.1109/5GWF.2018.8517049.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1904–1916, Sep. 2015, doi: 10.1109/TPAMI.2015.2389824.

[19] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017, doi: 10.1109/TPAMI.2016.2577031.

[20] M. Chiani, "Noncoherent frame synchronization," *IEEE Trans. Commun.*, vol. 58, no. 5, pp. 1536–1545, May 2010, doi: 10.1109/TCOMM.2010.05.090091.

[21] T. O'Shea and N. West, "Radio machine learning dataset generation with GNU radio," in *Proc. GNU Radio Conf.*, vol. 1, 2022, pp. 1–6.

[22] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Boston, MA, USA, 2015, pp. 3431–3440.

[23] D. George and E. A. Huerta, "Deep neural networks to enable real-time multimessenger astrophysics," *Phys. Rev. D*, vol. 97, no. 4, 2018, Art. no. 44039.

[24] E. Gordon-Rodriguez, G. Loaiza-Ganem, G. Pleiss, and J. P. Cunningham, "Uses and abuses of the cross-entropy loss: Case studies in modern deep learning," in *Proc. Int. Conf. Mach. Learn. Res.*, Dec. 2020, pp. 1–10.
[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[26] A. G. Paszke *et al.*, "Automatic differentiation in PyTorch," in *Proc. NIPS Workshop* 2017, pp. 1–4.
[27] J. M. Johnson and T. M. Khoshgoftaar, "Survey on deep learning with class imbalance," *J. Big Data*, vol. 6, p. 27, Mar. 2019. [Online]. Available: https://doi.org/10.1186/s40537-019-0192-5
[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org
[29] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Proc. 4th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, 1991, pp. 950–957.
[30] J. Garland and D. Gregg, "Low complexity multiply accumulate unit for weight-sharing convolutional neural networks," *IEEE Comput. Archit. Lett.*, vol. 16, no. 2, pp. 132–135, Jul./Dec. 2017, doi: 10.1109/LCA.2017.2656880.
[31] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, Oct. 2021.

SARUNAS KALADE received the B.Eng. degree (Hons.) in electronic and electrical engineering from the University of Strathclyde, in 2015, where he is currently a pursuing the Ph.D. degree with the Department of Electronic and Electrical Engineering.

He has completed several internships with the MathWorks and Xilinx, where he worked on hardware accelerated deep learning applications for computer vision and wireless communications. His primary research focus is on applying deep learning models on wireless communications datasets and implementing strategies of best data generation metholodogies in this field.

LOUISE H. CROCKETT received the master's and Ph.D. degrees in electronic and electrical engineering from the University of Strathclyde.

She is a Senior Teaching Fellow with the University of Strathclyde. She has core research interests in the hardware implementation of Digital Signal Processing systems, in particular for communications and SDR, and is the principal author of https://www.zynqbook.com/ "The Zynq Book." Her teaching focuses on digital systems design using hardware description language, Simulink block-based design, and FPGA/SoC technology, and builds practical skills to equip graduates for roles in industry.

ROBERT W. STEWART received the bachelor's and Ph.D. degrees from the University of Strathclyde.

He is a Professor of Signal Processing with the University of Strathclyde, where he served as the Head of the Department of Electronic and Electrical Engineering from 2014 to 2017. He manages a research group working on DSP, FPGAs, whitespace radio, and low-cost SDR implementation. He leads the Strathclyde cohort of the https://scotland5gcentre.org/Scotland 5G Centre. Building on the work of www.5GRuralFirst.org5G RuralFirst, the team has a focus on spectrum sharing, SDR eNodeB deployments, and developing innovative solutions to combat the connectivity issues faced in rural areas of the U.K.