# High Performance Urdu and Arabic Video Text Recognition Using Convolutional Recurrent Neural Networks

Abdul Rehman[1](✉), Adnan Ul-Hasan[2], and Faisal Shafait[1,2]

[1] School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST), Islamabad, Pakistan
{arehman.bese17seecs,faisal.shafait}@seecs.edu.pk
[2] Deep Learning Laboratory, National Center of Artificial Intelligence,
Lahore, Pakistan
adnan.ulhassan@seecs.edu.pk

**Abstract.** Text extraction from videos is an emerging research field in the document analysis community. We propose a simple Convolutional Recurrent Neural Network to perform text recognition on both Arabic and Urdu scripts. We use a large variety of data augmentation techniques to generalize the model and prevent over-fitting. We also use a slightly improved loss function that helps the model converge faster. Using the proposed method we achieved 99.73% CRR, 88.37% WRR and 89.92% LRR on the Urdu Ticker Text dataset and 96.82% CRR, 90.41% WRR and 76.78% LRR on the AcTiVComp20 dataset. The proposed method has significantly outperformed Google Vision API on both of the datasets.

**Keywords:** Urdu · Arabic · Video text recognition · CRNN

## 1 Introduction

Text recognition in its pure form - recognizing text from scanned documents - has been transformed in modern times. Smart phones have introduced a new capturing mechanism with additional challenges like page warps and view translations. Moreover, scene text recognition is a relatively new domain in text recognition that deals with text recognition in natural scenes and images. The text may occur in any shape, style or orientation, thereby increasing the text recognition challenge multi folds. An extension of scene text recognition is the emerging field of video text recognition. A video frame has additional challenges such as frame rate estimation, unique text determination, etc. in addition to the challenges of natural scenes.

Latin-based scripts receive most of the attention in both academia and industry. Remarkable results in Latin text recognition for both printed as well as handwritten text have been achieved. There has been very little or non-existent research carried out for Arabic scripts, including Arabic, Urdu, Persian, etc.


E. H. Barney Smith and U. Pal (Eds.): ICDAR 2021 Workshops, LNCS 12916, pp. 336–352, 2021.
https://doi.org/10.1007/978-3-030-86198-8_24

Arabic and the derived scripts pose several additional challenges for reliable recognition such as joined character, contextual shape change, diagonal writing flow (in case of Urdu Nastaleeq script) are some of these challenges.

The text recognition in video sequences has several important applications in the real world. News classification (sports, entertainment, current affairs, etc.) and coverage, Analytic dashboards highlighting the insights gained from entity extraction and news monitoring (identification of banned content) are some of the widely used scenarios of video text recognition. News channels use these applications to not only monitor their own transmission but also to keep an eye on their competition.

In contrast to text recognition from a scanned document, video text recognition poses several complex challenges. We need to extract individual frames from the video stream and apply text detection and recognition methods to get the text content. In video streams, the transition from one frame to another can cause some distortion that can be troublesome for text detection and recognition. We face this issue in NEWS channels. Static tickers fade in and out of the frame and in-between these transition models do not perform well. Distortion in scrolling tickers can cause a problem in text recognition. In video streams, we also need to extract unique instances of the text. We do so by keeping the track of position and duration of a text shown in the video stream. Text is video streams might also be in a different font, size or style. So in general, video text recognition is more challenging than documents.

The Urdu language is the national language of Pakistan. It is an Indo-Aryan language and shares similar phonology and syntax with the Hindi language; however, it is written in Arbic-like script. It has more than 170 million speakers. Urdu is mostly written in Nastaleeq script. Nastaleeq script was developed during the Persian region in the $14^{th}$ and $15^{th}$ centuries. The Urdu language has 37 unique alphabets. Alphabets are not written individually but are joined together to form ligatures.

The Arabic language is a universal language and the official language of 25 countries. It has over 300 million speakers. There are different writing styles for Arabic scripts. But it is mostly written in Naskh script. It first emerged in the $1^{st}$ to $4^{th}$ centuries CE. Similar to Urdu language, the Arabic alphabets are not written individually but are joined together to form ligatures.

In this paper, we focus on video text recognition in Arabic and Urdu that applies to natural images, videos and live streams. We pose the video text recognition as a sequence recognition problem. The problem can be described as converting an input image to a sequence of characters that represent text written in that image. It is also known as Image-based sequence recognition. Convolutional Neural Networks are excellent tools for extracting visual features from images. These visual features may represent different structures and shapes. In our case, these could be parts of letters, numbers or special characters. But these visual features alone are not enough to recognize the text written inside the image. We need to translate the sequence of visual features into character probability scores. Recurrent Neural Networks (RNNs) are excellent in sequence-to-sequence

translation. So using CNNs as visual feature extractors and RNNs as sequence-to-sequence translation we can effectively model our text recognition technique. We trained an end-to-end Convolutional Recurrent Neural Network (CRNN) to recognize Arabic and Urdu video text.

We test our proposed solution on two different datasets. The first dataset is the our own developed NUST-Urdu Ticker Text (NUST-UTT) dataset. It contains images of static and scrolling ticker text from different Urdu channels. The second dataset is the AcTiVComp20 dataset [1]. It also consists of cropped ticker text from different Arabic channels. We calculated Character Recognition Rate (CRR), Word Recognition Rate (WRR) and Line Recognition Rate (LRR) on each dataset, with and without spaces. We achieved CRR, WRR and LRR of 99.73%, 88.37% and 89.92% on the Ticker Text dataset and 96.82%, 90.41% and 76.78% on the AcTiVComp20 dataset.

The remaining of the paper is organized as follow. The Sect. 2 provides an overview of related work in the field of video text recognition, Sect. 3 provides the details of the proposed approach, Sect. 4 describes the dataset and outline the performed experiments. Section 5 details the results obtained and Sect. 6 concludes the paper with future directions.

## 2 Related Work

Text recognition is an old and well-researched field. But it is mainly researched in English or Chinese [15]. The field is wide-open for other languages. One of the main hurdles is the availability of data. But it is possible to generate printed text synthetically using different techniques. Researchers have tried different approaches for text recognition. These methods can be divided into segmentation-based techniques and non-segmentation-based techniques.

Older methods use a segmentation-based approach. These methods focus on isolated character or word recognition and recognizing them individually, which is possible in English and Chinese [11,20]. This sounds logical but it may become difficult when the characters are overlapping especially in handwriting recognition. Also, it is not easy to isolate each character in the languages Urdu as well as Arabic text. Another approach is to isolate ligatures instead of individual characters [3,4]. This approach works well for Urdu and Arabic text [13]. But this requires a lot of data. The technique becomes ineffective when the ligature count becomes very large. To classify each ligature there should be enough data samples in the dataset. In the NUST-UTT dataset alone there are 5,450 unique ligatures. Some other techniques also try to achieve word-level recognition but suffer the same problem [12]. The total number of classes grows so much, it becomes difficult to classify. Techniques that perform an isolated character, ligature, or word recognition are not effective for Urdu and Arabic scripts.

Modern techniques use a non-segmentation-based approach. We use a neural network to recognize the whole sequence instead of recognizing individual characters or ligatures. Non-segmentation approach uses Hidden Markov Models and recognizes the entire sequence [6,10]. Convolutional Neural Networks (CNNs) are

excellent at understanding visual features. Models like AlexNet and ResNet are exceptionally good at recognition objects [7,9]. But CNNs perform poorly in the sequence recognition tasks. RNNs are another family of Neural Networks which are pretty good at sequence-to-sequence tasks [16,19]. So, instead of recognizing isolated characters or words, researchers worked on sequence recognition. In sequence recognition, we recognize the entire line instead of recognizing individual parts. Recurrent Neural Networks are context-aware and can recognize patterns occurring in time series [19]. But visual features are troublesome for RNNs to learn. So a Convolutional Recurrent Neural architecture was proposed [14,18]. It uses a Convolutional Neural Network as a feature extractor and RNNs for sequence recognition. It takes advantage of both architectures and performs exceptionally well.

In ICDAR2017 Competition on Arabic Text Detection and Recognition in Multi-resolution Video Frames, the THDL-Rec technique used GRUs and achieved second best scores in evaluation [1,21]. GRU layers usually perform better than RNNs and LSTMs [5,8]. The most recent work is of OCR framework for detecting and recognizing Urdu in News channels [17]. It also uses a similar approach. It uses Convolutions Neural Networks for extracting visual features and Bidirectional LSTM layers for sequence recognition. We have further improved the method mentioned above by introducing a new model architecture, an improvement in loss function and an aggressive data augmentation for generalization.

## 3 Methodology

Since we only focus on Text Recognition in images, videos or live streams, we assume that we already have a Text Detection module to extract bounding boxes where the text is present. And we also need dataset containing those cropped images and their corresponding labels (in form of machine-readable text). In our case, we have NUST-UTT and the AcTiVComp20 dataset. We evaluate our method on both datasets.

To easily load datasets for training and inference we propose a formal method for storing them. This reduces CPU bottleneck and we do not need to write separate training and evaluation scripts for each dataset. We compute necessary hyper-parameters for each dataset during this step. This is further discussed in the Experimental Evaluation section (Sect. 4).

Once the dataset is ready, we train our model. In the training loop, we fetch images and labels from the dataset, perform data augmentation and preprocessing, feed it through the network, compute loss and back-propagate to adjust weights. We save the weights after each epoch if the validation loss improves. After training the model, we compute all five metrics (individual for each sample as well as collective) on the test set. We analyze the weaknesses of the model and adjust hyper-parameters to improve results.

### 3.1 Preprocessing

Urdu and Arabic Text is written from Right to Left. The data in computers (such as in arrays) are stored from Left to Right. So our labels (sequences of characters) are stored in reverse order. We either need to invert our label or the image itself to compute loss. We decided to invert the image (flip it on the x-axis). The color range of a grey-scale image is 0 to 255. So we normalize the image to have a range of $-1$ to $+1$. Normalizing input helps in gradient flow and the model converges faster. At this point, we have images that have fixed height but variable width. We need to have fixed height and width to create batches. So we pad zeros to images to make its width equal to max-width (computed in Data Preparation step). For labels, we simply convert each character to its corresponding index in characters plus 1 and pad the rest of the sequence with zeros (blank class) to make length equal to max label size.

### 3.2 Model Architecture

The network architecture is similar to CRNN model [14]. Original CRNN paper takes input image of height 32px while we input image of 64px. We also use fewer Convolution filters and fewer hidden units in RNN layers as compared to CRNN paper. Lastly, we use Bidirectional GRU layers instead of vanilla RNN layers. The architecture is further discussed below.

Our model has three parts. A CNN features an extractor, RNN layers for sequence recognition and a classification layer to classify feature vectors to characters. Initializing the model requires two hyper-parameters. Max Width $W$ and Total Characters $C$. These two parameters differ from dataset to dataset and are computed during the Data Preparation step. It is important to notice Total Characters also includes a blank character which is used by the CTC Loss function. For NUST-UTT, dataset we used $W = 1300$ and $C = 91$. For AcTiVComp20 dataset, we used $W = 1600$ and $C = 83$.

**CNN Feature Extractor.** The main purpose of the CNN feature extractor is to extract visual features from the image. RNN layers can be directly applied to input images but RNN layers are not very good at extracting visual features. CNN feature makes it easier for RNNs to understand sequence. Most of the noise and augmentations are filtered out by these CNN layers. This part consists of Convolution and Max Pooling layers. Each Convolution layer is followed by a Batch Normalization layer and Leaky ReLU activation function. We also use Dropout layers for regularization. Max pooling operations reduce the dimensions of the features. We only apply max pooling on the x-axis twice, reducing the width by a factor of 4. Applying further max-pooling can increase receptive field on CNNs but the operation results in poor results as multiple characters start to overlap on a single feature vector. The max-pooling operations along with the final Convolution layer squash the height dimension. We simply squeeze the height dimension.

**Table 1.** CNN feature extractor architecture

| Layer | Kernel size | Padding | Output shape | Parameters |
|---|---|---|---|---|
| Input | | | (1, 64, W) | 0 |
| Conv2d (16) | $3 \times 3$ | $1 \times 1$ | (16, 64, W) | 160 |
| BN (16) + LReLU | | | (16, 64, W) | 32 |
| MaxPool2d | $2 \times 2$ | | (16, 32, W//2) | 0 |
| Conv2d (32) | $3 \times 3$ | $1 \times 1$ | (32, 32, W//2) | 4,640 |
| BN (32) + LReLU | | | (32, 32, W//2) | 64 |
| MaxPool2d | $2 \times 2$ | | (32, 16, W//4) | 0 |
| Conv2d (48) | $3 \times 3$ | $1 \times 1$ | (48, 16, W//4) | 13,872 |
| BN (48) + LReLU | | | (48, 16, W//4) | 96 |
| Conv2d (64) | $3 \times 3$ | $1 \times 1$ | (64, 16, W//4) | 27,712 |
| BN (64) + LReLU | | | (64, 16, W//4) | 128 |
| MaxPool2d | $2 \times 1$ | | (64, 8, W//4) | 0 |
| Dropout (0.2) | | | (64, 8, W//4) | 0 |
| Conv2d (96) | $3 \times 3$ | $1 \times 1$ | (96, 8, W//4) | 55,392 |
| BN (96) + LReLU | | | (96, 8, W//4) | 192 |
| Conv2d (128) | $3 \times 3$ | $1 \times 1$ | (128, 8, W//4) | 110,720 |
| BN (128) + LReLU | | | (128, 8, W//4) | 256 |
| MaxPool2d | $2 \times 1$ | | (128, 4, W//4) | 0 |
| Dropout (0.2) | | | (128, 4, W//4) | 0 |
| Conv2d (256) | $4 \times 4$ | | (256, 1, W//4 − 3) | 524,544 |
| BN (256) + LReLU | | | (256, 1, W//4 − 3) | 512 |
| Reshape | | | (256, W//4 − 3) | 0 |
| Total parameters | | | | 738,320 |

**Table 2.** RNN layers architecture

| Layer | Output shape | Parameters |
|---|---|---|
| Input | (256, W//4 − 3) | 0 |
| Bidirectional-GRU (256) | (512, W//4 − 3) | 789,504 |
| BatchNorm (512) + LReLU | (512, W//4 − 3) | 1024 |
| Bidirectional-GRU (512) | (1024, W//4 − 3) | 3,151,872 |
| BatchNorm (1024) + LReLU | (1024, W//4 − 3) | 2048 |
| Total parameters | | 3,944,448 |

**RNN Layers.** RNN layers are used for sequence-to-sequence translation. In our case, we need to translate visual features into feature vectors that can be easily classified into characters. In short, we are performing sequence recognition. In this section, we use specifically two Bidirectional GRU layers, each one followed

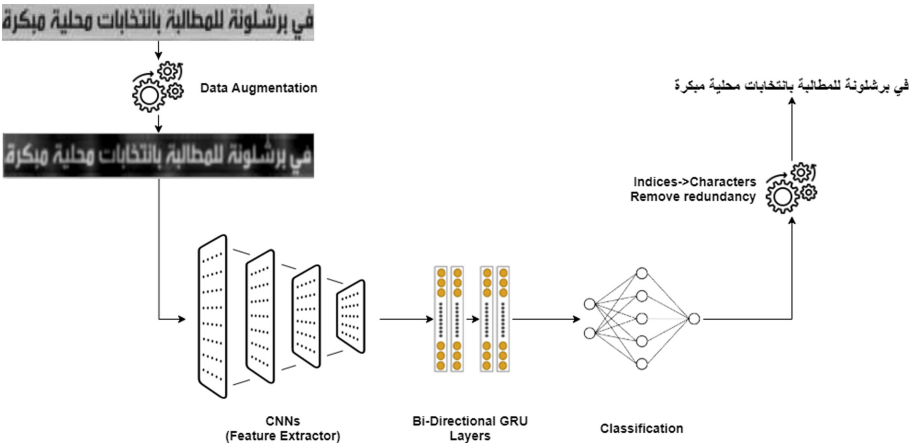



**Fig. 1.** A diagram summarizing the Model Architecture. The input sample is taken from AcTiVComp20 dataset [1]. The model consists of CNN Feature Extractor, two Bi-directional GRU layers and a Classification layer. Data augmentation is applied only during the training loop.

by a Batch Normalization layer and a Leaky ReLU activation function. We used GRU layers as these layers suffer least from gradient vanishing and exploding problems. GRU layers are faster and lightweight compared to LSTM layers.

**Classification Layer.** The classification layer consists of a single 1-d Convolution layer of kernel size of 1 followed by a LogSoftmax activation function. This layer classifies each feature vector of RNN Layer output into a character. The output channels of Convolution are equal to Total Characters C. Total number of parameters of the classification layer are $1025 * C$.

### 3.3 Model Training

To start training, we first load the target dataset containing train, validation and test sets. We initialize the model with default parameters suited for the target dataset. We initialize the AdamW optimizer to optimize the weights of the network.

The training loop takes a batch of images, passes them through the network and gets output activation of the model. After that, it computes loss between output activation and ground truths, and backpropagates through the network to compute gradients. The optimizer optimizes the model based on computed gradients. The same training loop is used to compute validation loss as well

but the loss is not backpropagated. The validation loss is used to identify if the model is converging or not. Model checkpoint is saved only when validation loss decreases compared to the previous epoch. This makes sure we do not save a checkpoint that is overfitted on the dataset. Once the training is finished we take the best checkpoint and evaluate it on the test set. It is important to notice that we do not use the test set during the whole training process. It is only used to compute metrics once the whole training process is finished (Table 1).

We train the models in two steps. In the first step, we train the model with a learning rate of 0.0003. Model mostly learns in the first step. We train the model for nearly 100 epochs or until it stops converging. In the second step, we further fine-tune the model with a learning rate of 0.0001 for nearly 50 epochs. In this step, the optimizer tweaks the parameters to slightly improve accuracy (Fig. 1).

### 3.4 Loss Function

Similar to the CRNN model we also use Connectionist Temporal Classification Loss (CTC Loss) [14]. It is used to predict as well as align the output with the ground truth. Since we pad images during Pre-processing, we also need to ignore the output activations for the padded region of the image. Max Width is the maximum possible width an image can have in the dataset. Most of the images will not even reach close to the maximum possible width. The padded region will not contribute to the output of the model, making it difficult for the model to align text from such a large region. We can make this job easier for the model by simply ignoring the activations of the padded region. The shape of the activation region for any image with width W can be calculated using the formula (C, $W//4 - 3$) given in Table 2. So, We can only consider the first $W//4 - 3$ vectors.

We also need to consider receptive fields of CNNs and RNNs. To make the model fail-safe we add another 16 vectors after the $W//4 - 3$ vectors. We call these vectors CTC pad. So total number of output feature vectors we use for loss function are $W//4 - 3 + 16$. But the term may exceed the total output feature vectors returned by the model. So, the fail-safe formula is $min(Width//4 - 3 + 16, MaxWidth//4 - 3)$.

Figure 2 shows an input image and the output activations of the model. The input image is pre-processed. In the pre-processing step, we invert the image and pad it to make it fixed-sized. In the output, we can see scores of each class or character. The Image Activations classify the target character while the CTC pad section has the highest score for the blank class. The Padding Activations are random. These are ignored in both training and inference.

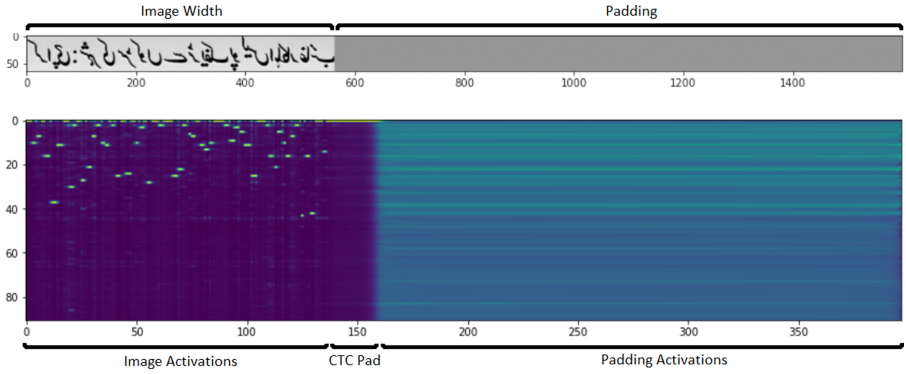



**Fig. 2.** The diagram shows the input image and the output activations of the model on the same image. The input can be divided into two parts, the image itself and the padding to make it fixed sized. The output activation maps show Image Activations, CTC Pad and Padding Activations. Image Activations are class scores of each character. CTC Pad consists of regions separating image activations from padding activations. Padding Activations are ignored in both training and inference.

## 4 Experimental Evaluation

In this section we will discuss about Data Preparation, Data Augmentation, Evaluation Metrics and Experiments performed on the datasets.

### 4.1 Datasets

We focus on two main dataset, AcTiVComp20 dataset and NUST-Urdu Ticker Text datasets. Both datasets contain ticker text data for News channels. Each dataset has its own set of alphabets. Both datasets have different characteristics which are mentioned below.

**AcTiVComp20 Dataset.** AcTiVComp20 dataset contains 7,943 training samples which have 36,593 words and 212,393 characters. It has total 83 unique characters. The dataset contains images and their corresponding XML files. XML files contain labels in both Arabic and Latin transcriptions. We only used Arabic transcription. The test set is given separately, so we only split the train set into train (90%) and validation (10%) sets while keeping the test set the same. We do not perform any data cleaning or correction for AcTiVComp20 dataset. It is important to notice that AcTiVComp20 dataset contains RGB images while our method operates at grey scale images. So we convert images to grey scale as mentioned in Algorithm 1.

**NUST-Urdu Ticker Text (NUST-UTT) Dataset.** The NUTS-Urdu Ticker Text (NUST-UTT) dataset[1] contains 19,437 data samples. But the dataset contains ligatures separated by spaces instead of words. It has total of 503,273 ligatures, 5,450 unique ligatures and 90 unique characters. We need to convert ligature into words. We cannot compute WRR without converting text to words. To solve this issue, we trained a simple 2 layer Bidirectional GRU model that predicts the positions where the spaces should be inserted in a space less text. We trained our model on a huge Urdu Corpus. Once the model was trained we corrected our ground truth with the trained model during Data Preparation step.

### 4.2 Data Preparation

The first step is to formalize the given dataset. In short, we convert it into an easy-to-use format. It reduces the CPU bottleneck for model training. We simply convert images to grey scale, resize all the images to a fixed height of 64px while keeping the image aspect ratio the same, store them in the "images" directory. Then we create three index files. Each one for train-set, validation-set and test-set. The index files contain a list of image-label pairs separated by a tab. We can quickly load any set by reading the index files. We also compute a character file which contains all unique characters that appear in the dataset. We also compute two hyper-parameters Max Image Width and Max Label Length which are later used in training process. We use the following algorithm to formalize the dataset.

The algorithm for each dataset is the same except for data loading and the data cleaning & data correction steps.

### 4.3 Data Augmentation

Data augmentation refers to the process of increasing data samples by transforming existing samples. It is very useful in image-based learning tasks. It prevents over-fitting. We use data augmentation during model training. Data augmentation is only performed on the train set. We have a total of 9 different augmentation functions. 4 of them are shape-based augmentation functions and the rest are color-based augmentations. Shape-based augmentations transform the image into different dimensions. So, after each shape-based augmentation, we resize the image to a height of 64px while keeping the aspect ratio the same.

We do not apply all the augmentations to every batch we fetch from the train set. We select K random augmentation functions for each image fetched from the train set. Applying all augmentation at the same time can distort the image to such an extent that it becomes unrecognizable. In other words, the distribution of the dataset becomes different. The resultant model does not perform well on the test-set. Applying all augmentation can also cause a CPU bottleneck because

[1] The Urdu Ticker Text Dataset will be available publicly at https://tukl.seecs.nust.edu.pk/downloads.html.

```
Data: List of image-label pairs
Result: Processed Dataset
data cleaning & correction;
compute maximum image width;
compute maximum label size;
save max image width and max label size;
create unique characters list;
foreach image_path, label ∈ pairs do
    add unseen characters from label to characters list;
    load image and convert it to grey scale;
    resize image to height 64px while keeping aspect ratio same;
    save image to new path and update path in pairs;
end
sort and save characters;
shuffle pairs;
if test set given then
    split images into training and validation sets;
else
    split images into training, validation and test sets;
end
foreach set ∈ {trainingset, validationset, testset} do
    create index file;
    foreach image, label ∈ set do
        append image path;
        append tab character;
        append image label ;
        append new line character;
    end
    close index file ;
end
```

**Algorithm 1:** Data preparation steps.

these augmentation functions are CPU-intensive tasks. After testing different values for K, we decided to use K = 3 for all datasets. Following is the list of all augmentation functions, their default hyper-parameters and the affect of each augmentation on the image given below.

**Invert Colors.** It is a color-based augmentation function that randomly inverts color. There is a 50% chance for image colors to be inverted.

**Pad Image.** It is a shape-based augmentation function that randomly pads each side of the image. It samples the number of pixels for each side (left, right, top and bottom) from a range of 0px to 20px and pads image accordingly.

**Brightness and Contrast.** It is a color-based augmentation function that randomly changes the brightness and contrast of the image. It samples $\alpha$ from a

range of 0.75 to 1.25 and $\beta$ from a range of $-0.25$ to 0.25. It computes new value $x'$ given original value as $x$ using formula $x' = x * \alpha + \beta$. The output value $x'$ may exceed the pixel intensity range of 0 to 255. So we clip values that fall out of this pixel intensity range.

**Cloudy Effect.** It is a color-based augmentation that adds random soft noise to the image. It creates a small image containing grain noise. Pixel intensity values of grain noise image range from $-255$ to 255. The grain noise image has dimensions of (W//DOWN_FACTOR, H//DOWN_FACTOR). The variable DOWN_FACTOR is sampled from a range of 8 to 16. Another variable intensity $i$ is sampled from a range of 0 to 1 that refers to the intensity at which the noise will be applied. To apply the effect, we resize the grain noise image to the dimensions of the input image using bilinear interpolation, multiply it with intensity $i$ and add it to the input image. Similar to "Brightness & Contrast" augmentation, the output image pixel intensity values may exceed the of 0 to 255. We clip values that fall outside of this range.

**Blur.** It is a color-based augmentation that applies box blur of a random kernel size to the image. The size of kernel, height and width is sampled from a range of 1 to 5.

**Squeeze.** It is a shape-based augmentation function that resizes the image with a random ratio. X & Y ratios are sampled from a range of 0.9 and 1.1, and the image is resized to new ratio.

**Degrade.** It is a color-based augmentation function that down-samples input image by a random factor and up-samples to original dimensions. It keeps aspect ratio same when down-sampling the image. Down-sample factor is sampled from a range of 1 to 2.

**Rotate.** It is a shape-based augmentation function that rotates image by a random angle. The angle is sampled from a range of $-0.5$ to 0.5.

**Stretch.** It is a shape-based augmentation function that stretches or compresses random chunks of the image. We divide image from left to right in chunks of random width ranging from 64px to 128px. Stretch or compression is only applied on x-axis (width). These chunks are then concatenate to get output image (Figs. 3 and 4).

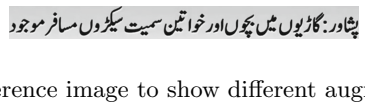
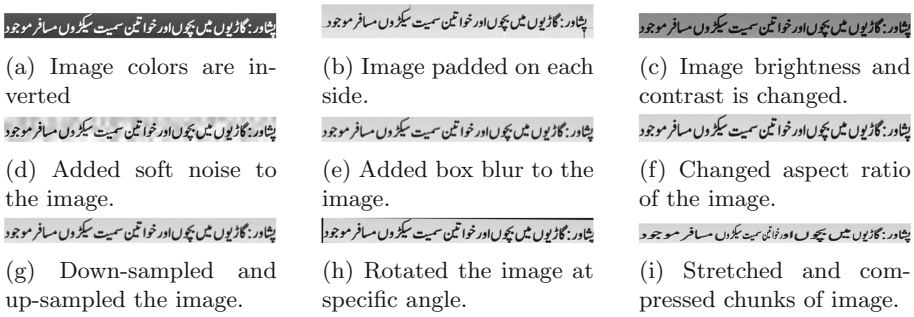



**Fig. 3.** Reference image to show different augmentations.




(a) Image colors are inverted (b) Image padded on each side. (c) Image brightness and contrast is changed.

(d) Added soft noise to the image. (e) Added box blur to the image. (f) Changed aspect ratio of the image.

(g) Down-sampled and up-sampled the image. (h) Rotated the image at specific angle. (i) Stretched and compressed chunks of image.

**Fig. 4.** Visualization of all augmentation functions respectively.

### 4.4 Evaluation Metrics

We trained and evaluated our model on both datasets. The evaluation is based on five different metrics. We compute character recognition space (CRR), word recognition rate (WRR) and line recognition rate (LRR) which are used to evaluate any text recognition model. We also compute CRR-WS and LRR-WS. WS referring to Without Spaces. In languages like Arabic or Urdu, some words can be understood without having a space between them. So space in such cases is optional. A space between two ligatures of a single word is optional as well. Word Recognition Rate (WRR) and Line Recognition Rate (LRR) are very sensitive on it. A single incorrectly predicted character can invalidate the entire line causing LRR to be zero. So also we compute CRR and LRR by ignoring spaces. We call them CRR-WS and LRR-WS.

Computing CRR and WRR without spaces are also important for the NUST-UTT dataset. The dataset originally contained ligatures separated by spaces. We fixed dataset by predicting spaces after words in a spaceless Urdu text using another model. The reconstructed dataset contains words separated by space. But the reconstruction is not perfect. So it is important to compute WRR-WS and LRR-WS metrics.

### 4.5 Hyper-parameter Selection

Before training the models, we performed a hyper-parameter search to look for the best parameters to train the model. We tested three optimizers with two different learning rates. We tested SGD, Adam and AdamW. For learning rate, we tested learning rates of 0.001 and 0.0003. Hyper-parameter search is a very time-consuming task, so we ran our simulations for 10 epochs and computed test metrics. This gives us a general idea about what parameters should we use.

### 4.6 Experiments Performed

We compared our model with Google Vision APIs as well [2]. Google Vision APIs provide Optical Character Recognition for both Urdu and Arabic language.

Urdu APIs are currently in experimental phase. We used Google Vision APIs to extract text from all the test images for both datasets. We compared the output of APIs with the ground-truth. We computed all five evaluation metrics for Google Vision APIs as well. We compared our model with the Google Vision APIs.

## 5 Results and Discussion

The hyper-parameter search gave us a general idea about parameters to choose for training. As Table 3 shows, the model optimized using AdamW optimizer with a learning rate of 0.0003 performed the best. SGD performed the worst. It could be because SGD takes much longer to optimize. SGD optimizer with a learning rate of 0.0003 did not learn anything in 10 epochs. After performing the hyper-parameter search we decided to use AdamW optimizer with a learning rate of 0.0003. We trained the models for 100 epochs and then further fine-tuned models with a learning rate of 0.0001 for 50 more epochs. Table 3 shows the final metrics of the model on each dataset.

**Table 3.** Results of the hyper-parameter search for Urdu Ticker Text dataset trained for 10 epochs. AdamW with a learning rate of 0.0003 performed the best.

| Optimizer | Learning rate | CRR | WRR | LRR | CRR-WS | LRR-WS |
|---|---|---|---|---|---|---|
| AdamW | 0.0003 | **98.52%** | **88.37%** | **48.92%** | **99.73%** | **89.92%** |
| AdamW | 0.001 | 98.39% | 87.6% | 46.4% | 99.65% | 86.57% |
| Adam | 0.0003 | 98.48% | 88.11% | 48.66% | 99.68% | 87.91% |
| Adam | 0.001 | 98.2% | 86.01% | 41.41% | 99.65% | 87.04% |
| SGD | 0.0003 | 0% | 0% | 0% | 0% | 0% |
| SGD | 0.001 | 42.16% | 5.74% | 0% | 34.28% | 0% |

Once the training finished, we evaluated both models on their respective datasets. We also evaluate results for Google Vision APIs on both datasets as well. We compare each sample prediction with its ground turth and take mean of all individual metric values. The Table 4 shows the final results of our models compared with Google APIs. [2]

In Urdu Ticker Text Dataset, metrics without spaces are much better than normal ones. The LRR is only 48.92% while LRR-WS is 89.92%. Its shows that space inconsistencies in Urdu Ticker Text Dataset are huge as compared to the AcTiVComp20 dataset. The results show that OCR is difficult to do in the Arabic language as compared to Urdu. But it could be due to the incorrect labeling of digits, as mentioned in the Failure Cases section.

Finally, we analyze the failure cases of our model. We created a table containing ground truth and model predictions of all data samples in the test set.

**Table 4.** Final results on both datasets compared with Google Vision APIs.

| Dataset | Method | CRR | WRR | LRR | CRR-WS | LRR-WS |
|---|---|---|---|---|---|---|
| NUST-UTT | Google Vision | 93.71% | 65.80% | 8% | 96.21% | 40.07% |
| NUST-UTT | Ours | **98.66%** | **89.32%** | **52.05%** | **99.76%** | **90.84%** |
| AcTiVComp20 | Google Vision | 91.69% | 76.62% | 64.61% | 92.95% | 71.25% |
| AcTiVComp20 | Ours | **96.82%** | **90.41%** | **76.78%** | **96.83%** | **79.38%** |

We also computed all of the five metrics for individual data samples. We were able to identify major problems with our approach.

In AcTiVComp20 dataset, the model mostly fails to recognize numeric digits or special character. It is because the digits in Arabic transcription are stored in reverse order but do visually appear correct as shown in Fig. 5. This makes harder for RNN layers to recognize digits in AcTiVComp20 dataset. This issue can be solved by inverting sequence of digits of all numbers in the dataset or simply using Latin transcription instead.


**Fig. 5.** The diagram shows the problem we faced with Arabic transcription of AcTiVComp20 dataset [1]. The order in which the characters are stored were incorrect for digits in Arabic transcription. Visually the text below aligns with the image but it is not read correctly by computers. The arrows at the bottom show the order in which the label is actually stored.

In NUST-UTT dataset, the failure cases are mostly related to space inconsistencies. This can be observed from results. There is huge difference in normal and WS metrics. The LRR improved from 52.05% to 90.84% when the spaces were ignored.

## 6 Conclusion

In this paper, we proposed a technique to efficiently recognize Urdu and Arabic text from video. This includes the different types of augmentation functions, how to apply them, and how it helps regularize the model. We also explained the model architecture, the purpose of each part and its total parameters. Lastly,

we explained the training and evaluation process for our model. In the training process, we explained how the CTC Loss function can be modified to achieve better results.

There is a lot of room for improvement. We only performed a hyper-parameter search for only 10 epochs. And the total search space was very limited. A better hyper-parameter search can help us improve the results. Results can be further improved by solving issues mentioned in the Failure Cases section. In the AcTiVComp20 dataset, we need to fix the order of digits for numbers. And for the NUST-UTT dataset, we need to fix the issue with space inconsistencies.

**Acknowledgement.** This work has been partially funded by the Higher Education Commission of Pakistan's grant for National Center of Artificial Intelligence (NCAI).

## References

1. Competition on superimposed text detection and recognition in Arabic news video frames. https://diuf.unifr.ch/main/diva/AcTiVComp/index.html. Accessed 23 Feb 2021
2. Detect text in images — cloud vision API — google cloud. https://cloud.google.com/vision/docs/ocr. Accessed 26 May 2021
3. Ahmad, I., Wang, X., Li, R., Ahmed, M., Ullah, R.: Line and ligature segmentation of Urdu Nastaleeq text. IEEE Access **5**, 1–17 (2017)
4. Al-Wzwazy, H.: Handwritten digit recognition using convolutional neural networks. Int. J. Innovative Res. Comput. Commun. Eng. **4**, 1101–1106 (2016)
5. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Netw. Publ. IEEE Neural Netw. Council **5**, 157–66 (1994)
6. Graves, A., Fernández, S., Gomez, F., Schmidhuber, J.: Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks, vol. 2006, pp. 369–376 (2006)
7. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition, vol. 7 (2015)
8. Informatik, F., Bengio, Y., Frasconi, P., Schmidhuber, J.: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In: A Field Guide to Dynamical Recurrent Neural Networks (2003)
9. Krizhevsky, A., Sutskever, I., Hinton, G.: ImageNet classification with deep convolutional neural networks. In: Neural Information Processing Systems, vol. 25 (2012). https://doi.org/10.1145/3065386
10. Melnikoff, S.J., Quigley, S.F., Russell, M.J.: Implementing a hidden Markov Model speech recognition system in programmable logic. In: Brebner, G., Woods, R. (eds.) FPL 2001. LNCS, vol. 2147, pp. 81–90. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44687-7_9
11. Mollah, A., Majumder, N., Basu, S., Nasipuri, M.: Design of an optical character recognition system for camera-based handheld devices. Int. J. Comput. Sci. Issues, vol. 8 (2011)
12. Rehman, A., Hussain, S.: Large scale font independent Urdu text recognition system (2020)

13. Sabbour, N., Shafait, F.: A segmentation free approach to Arabic and Urdu OCR. In: Proceedings of SPIE - The International Society for Optical Engineering, vol. 8658 (2013)
14. Shi, B., Bai, X., Yao, C.: An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. IEEE Trans. Pattern Anal. Mach. Intell. **39**(11), 2298–2304 (2016)
15. Ul-Hasan, A., Shafait, F., Breuel, T.: High-performance OCR for printed English and fraktur using lstm networks. In: Proceedings of the International Conference on Document Analysis and Recognition, ICDAR (2013)
16. Ul-Hasan, A., Ahmed, S., Rashid, S.F., Shafait, F., Breuel, T.: Offline printed Urdu Nastaleeq script recognition with bidirectional LSTM networks (2013)
17. Ur-Rehman, S., Tayyab, B., Naeem, M., Ul-Hasan, A., Shafait, F.: A multi-faceted OCR framework for artificial Urdu news ticker text recognition. In: 13th IAPR International Workshop on Document Analysis Systems, DAS 2018, Vienna, Austria, 24–27 April 2018, pp. 211–216. IEEE Computer Society (2018)
18. Xie, Z., Sun, Z., Jin, L., Feng, Z., Zhang, S.: Fully convolutional recurrent network for handwritten Chinese text recognition (2016)
19. Yanikoglu, B., Sandon, P.: Off-line cursive handwriting recognition using style parameters (1970)
20. Yuan, T.L., Zhu, Z., Xu, K., Li, C.J., Hu, S.M.: Chinese text in the wild (2018)
21. Zayene, O., Hennebert, J., Ingold, R., Essoukri Ben Amara, N.: ICDAR 2017 competition on Arabic text detection and recognition in multi-resolution video frames, pp. 1460–1465 (2017)