Exploring CNN Architectures: A Comparative Study on Arabic Character Classification

1st Eng. Ahmad Abbas
dept. Computer Engineering
Birzeit University
Ramallah, Palestine
ahmad.abbas176@yahoo.com

2nd Eng. Loor Sawalhi
dept. Computer Engineering
Birzeit University
Ramallah, Palestine
loorwael@yahoo.com

Dr. Aziz Qaroush
dept. Computer Engineering
Birzeit University
Ramallah, Palestine
aqaroush@birzeit.edu

***Abstract*—Handwriting recognition is a computer vision problem that includes having a computer recognise handwritten text and convert it into a format that can be read by machines. Many researches aimed to build suitable models for these task in different languages including Arabic language. This paper aims on studying different architectures aimed to solve this task beside well known architectures. Moreover, we will explore the effect of data augmentation as well as the function of hyper-parameter tuning in optimizing model performance. This is done by a comparative study using loss functions, optimization, data augmentation and transfer learning.**

***Index Terms*—CNN, Epoch, Kernel, KNN, k-Nearest Neighbors, NN, ReLU.**

I. Introduction

Automatic handwriting recognition is an important part of computer vision and is about recognizing human handwriting from different sources like paper, pictures, touch-screens, or other devices. This can be done either by scanning (offline) or using a pen on a device (online) [1]. Recognizing Arabic handwriting has become more important today because Arabic characters are detailed and Arabic is used more in areas like business, education, and government [2]. This technology is used for turning paper documents into digital files, typing on mobiles, and turning handwritten notes into typed text [3].

The challenge in this field comes from how different each person's handwriting is and the special features of Arabic writing. But, recent progress with machine learning and deep neural networks has really improved Arabic handwriting recognition, making it a field that is growing and showing a lot of promise [4]. Even though a lot of research has been done on handwriting recognition in languages like English, French, and Chinese, there is a growing need and potential for more work in Arabic handwriting recognition. This highlights the importance of continuing to research and come up with new ideas in this area [5].

Handwriting recognition has been studied using different methods like support vector machines (SVMs), K-nearest neighbors (KNNs), neural networks (NNs), and recently, convolutional neural networks (CNNs) [6]–[8]. Most of this research, however, has focused on handwriting in languages like English [9]. Arabic, spoken by around 300 million people and one of the top five most spoken languages in the world, presents unique challenges in handwriting recognition [10]. Arabic is written from right to left, and its alphabet has 28 letters that can change shape depending on their position in a word. Arabic also uses diacritical marks for short vowels and sounds, and it has ligatures, which are combinations of letters like alif-laam.

Handwriting recognition research often targets number recognition more than letter recognition and usually focuses on adult handwriting. Recognizing children's handwriting is becoming important for applications as children increasingly use technology like smartphones and tablets. Handwriting with a finger or stylus is a common way for children to interact with these devices. To date, there appears to be limited research specifically addressing children's handwriting [11], [12].

In this paper, our goal is to compare three different convolutional neural network (CNN) designs for recognizing Arabic handwritten characters. These designs are from the studies of Alsayed et al. (2023) [13], Alwagdani and Jaha (2023) [14], and Altwaijry and Al-Turaiki (2021) [5]. We will use the same dataset for all to make sure our comparison is fair. We also plan to test well-known architectures like AlexNet [15], ResNet18 [16], and VGG [17], which were originally trained on the ImageNet dataset [18]. Plus, we want to see how models pre-trained on the EMNIST dataset, which is a version of the MNIST dataset for English letters and numbers [19], do at recognizing Arabic handwritten characters. Our study aims to fill in gaps in the current research by giving a thorough look at different CNN designs and how well they work for Arabic character recognition.

II. Background

A. Neural Networks

Artificial Neural Networks, a part of machine learning that forms the core of deep learning algorithms. Their structure

are derived from the human brain, where they are composed from nodes or neurons. These neurons make up the network layers, which are composed of an output layer, an input layer, and one or more hidden layers. For each neuron, weight and threshold are associated to it. For this threshold, if the output is above the threshold, the neuron is activated and the data is sent to the next layer of the network [20].

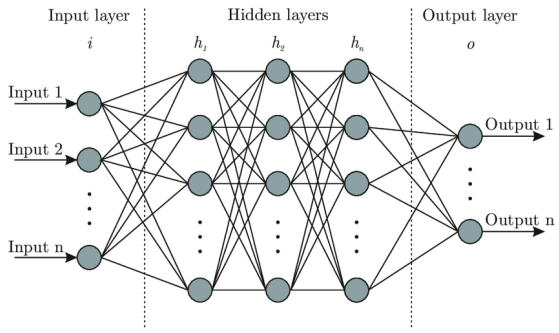


Fig. 1: Artificial Neural Network [21].

Figure 2 shows the structure of artificial neurons, where each input to the transfer function is associated with its weight, the output of the transfer function is then compared to a certain threshold to whether activate the neuron or not.

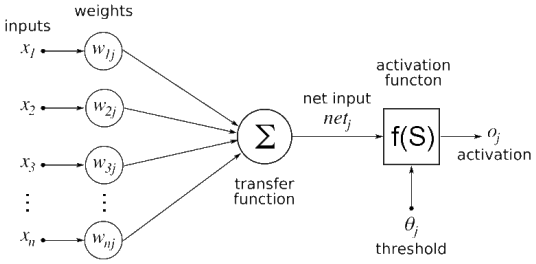


Fig. 2: Artificial Neuron [22].

B. Convolutional Neural Networks

Convolutional Neural Networks (CNN), is a deep learning model that consists of three main components, convolutional layer, pooling layer, and fully-connected layer. It is mainly designed for processing visual data, such as images and videos. They have influenced computer vision and made incredible progress possible in areas such as object detection, image segmentation, and image recognition.

Figure 3 shows the basic structure of CNNs, where the first part is feature extraction that is done by the convolutional and pooling layers, followed by the classification part done by the fully connected layers.

1) Convolutional Layer: The convolutional layer is the core building block in the CNN, it basically contains most of the computations that is done for extracting features. Essentially,

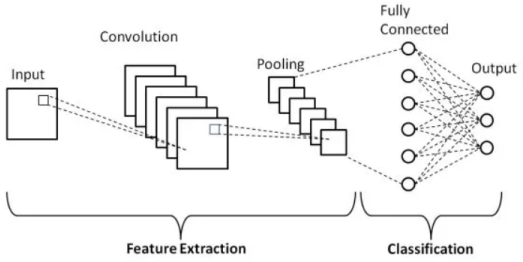


Fig. 3: Basic Structure of Convolutional Neural Networks [22].

it takes an input image and runs it through a certain number of filters, each of which has a given kernel size that is identical in depth but much less in width and height than the input. The kernel is shifted with a shifting amount known as the **Stride** across the width and height of the input image. A dot product is then calculated between them, and the resultant output is referred to as the feature map, which provides the kernel's response at each spatial position of the image [23].

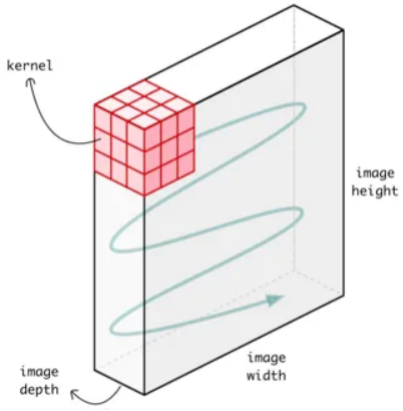


Fig. 4: Illustration of Convolution Operation [24].

The output of the convolutional layer is effected by several hyperparameters which are [25]:

1) Kernel Size: is the sliding window size, is preferred to be an odd number that is much smaller than the input size. 3 and 5 are preferable.
2) Stride: the number of pixels the kernel window will move during the convolution step. It is usually set to 1 and can be raised if decreasing the input size is the goal.
3) Padding: Is the process of adding zeros to an image's border . The kernel can thoroughly filter each point in an input image via padding, guaranteeing that even the edges are handled correctly.
4) Number of Filters: Number of filters determines how many patterns or features the layer will search for.

The output height 1 and width 2 is as the following:

$$H_{\text{out}} = 1 + \frac{H_{\text{in}} + (2 \cdot \text{pad}) - K_{\text{height}}}{s} \tag{1}$$

$$W_{\text{out}} = 1 + \frac{W_{\text{in}} + (2 \cdot \text{pad}) - K_{\text{width}}}{s} \tag{2}$$

2) Pooling Layer: Pooling layer, or the downsampling layer, is the layer that works in reducing the dimensionality of the input. It is similar to the convolutional layer in the general concept, where a filter is passed across the input. However, the pooling filter has no weights, the filter creates the output array by applying an aggregation function to the data in its receptive field. To be noted that the size of the filter in the pooling is preferred to be even [25]. The aggregation function can be out of two:

1) Max Pooling: It picks the pixel to send to the output array with the highest value.
2) Average pooling: Determines the average value to send to the output array from the receptive field.

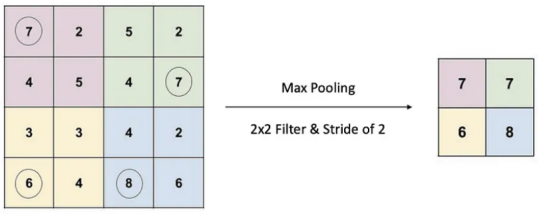


Fig. 5: Illustration of Pooling Operation [25].

Assuming a pooling kernel of spatial size F, a stride of S, and an activation map of size W x W x D, the output volume is calculated as:

$$W_{\text{out}} = \frac{W - F}{S} + 1 \tag{3}$$

The result will be an output that is W_{out} x W_{out} x D.

3) Fully Connected Layer: The fully connected layers of a CNN, tend to be towards the end. Each neuron in these layers is connected to every other neuron in the layer before, forming a dense network of connections.These layers are responsible for combining the high-level features learned by the convolutional layers to make final predictions or classifications. In image classification tasks, probability scores for various classes are often generated by feeding the output of the final fully connected layer into a softmax activation function [25].

C. Activation Functions

Activation functions is a transfer function that is used to determine the output of a neural network. The Activation Functions can be divided linear and non-linear functions.

1) Sigmoid: The sigmoid is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

The plot of the sigmoid is shown in Figure 6.

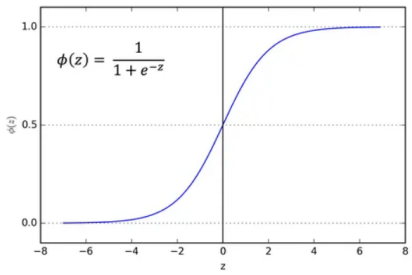


Fig. 6: Sigmoid Activation Function [26].

2) ReLU: Rectified Linear Unit or ReLU is defined as:

$$\text{ReLU}(x) = \max(0, x) \tag{5}$$

The plot of the ReLU is shown in Figure 7.

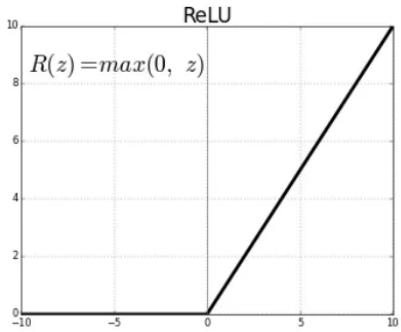


Fig. 7: ReLU Activation Function [26].

D. Loss Functions

the loss function is a way of evaluating how will a machine learning algorithm works in terms of predicting the expected outcome. Note that loss functions for classification problems differ than those for regression problems [27].

1) Binary Cross-Entropy Loss: A common loss function in classification, The value decreases when the expected probability approaches the true label. A probability value between 0 and 1 is the expected output of the classification model [27].

For binary classification the loss is calculated as:

$$L = -\frac{1}{m} \sum_{i=1}^{m} (y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)) \tag{6}$$

While for multi-class classification:

$$L = -\frac{1}{m} \sum_{i=1}^{m} y_i \cdot \log(\hat{y}_i) \tag{7}$$

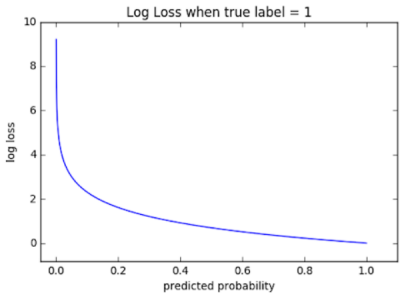


Fig. 8: Cross Entropy Loss for True Label = 1 [27].

2) Mean Square Error: A common regression loss function , it can be defined as the mean squared difference between the expected and actual value [27]. The loss function can be expressed as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2 \tag{8}$$

E. Hyper-parameters

1) Learning Rate: Learning rate defines the rate of which an algorithm converges to a solution. It is one of the most important hyper-parameters in machine learning, and choosing the correct value affects the learning process [28].

If using gradient descent, the learning rate forms the step size, such that small learning rate will slow the learning process, a larger one will diverge and never meet the optimal solution. Even-though, the learning rate can vary through the learning process, where it can be changes based on the learning status [28].

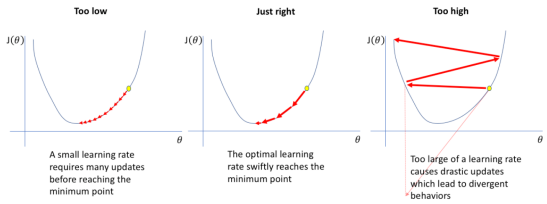


Fig. 9: Learning VS Learning Rate [29].

2) Batch Size: Batch size is the number of samples used in each batch in the learning process. There are three types of batches:

1) Batch gradient descent: uses all the samples at once.
2) Stochastic gradient descent: for each epoch, one sample is used.
3) Mini-batch gradient descent: a small subset of the training is used in each epoch.

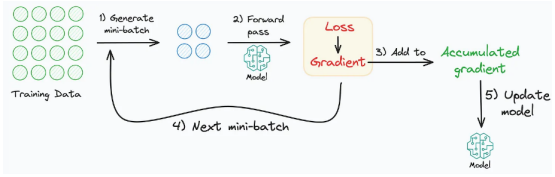


Fig. 10: Example for training using a mini-batch [30].

F. Regularization Techniques

In machine learning, regularisation is a collection of techniques designed for reducing . After training, the majority of models show excellent performance on training sets but struggle with generalisation. Regularisation techniques seek to minimise training error while also reducing overfitting this results in simpler models.

1) Early Stopping: Early stopping is an easy technique that entails stopping the neural network's training at an earlier epoch. This is carried out in order to avoid overfitting the training set in the case that the training error becomes too low. The early stopping will keep it from approaching zero. This can be achieved by keeping an eye on changes in metrics like validation accuracy and error as well as weight vector changes [31].

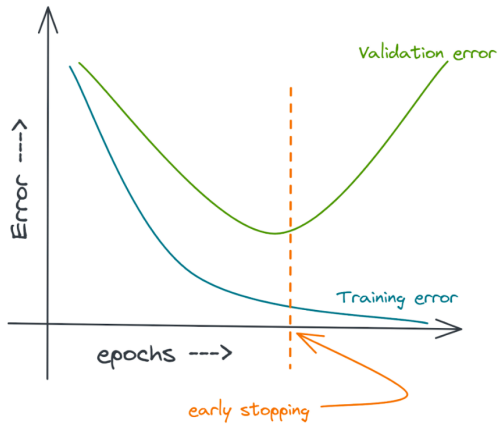


Fig. 11: Early stopping for the case of having the validation error not decreasing [32].

2) L1 Regularization: L1 regularisation adds the absolute value of the weights to the loss function. This basically does feature selection by decreasing some weights to become exactly zero [33].

$$L1(\mathbf{w}) = \lambda \sum_{i} |w_i| \tag{9}$$

3) L2 Regularization: L2 regularisation adds the adds the square of the weights to the loss function. This helps in

distributing the importance to all features, it also reduces the weights so they can't grow too large [33].

$$L2(\mathbf{w}) = \lambda \sum_i w_i^2 \tag{10}$$

4) Dropout: Dropout, a noise injection technique that is used as a regularization one. This is done by ignoring the outputs of randomly picked layers at the training temporarily. As a result, during training, every layer update is carried out using an individual "view" of the preset layer,it's similar to training many neural networks with various architectures simultaneously [31].

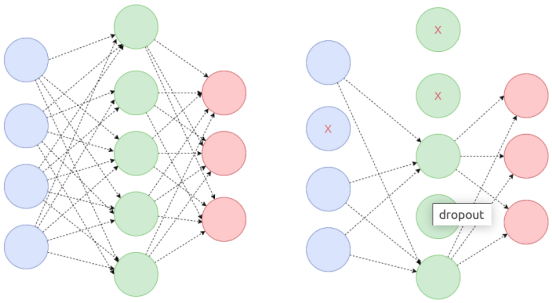


Fig. 12: Illustration of dropout [31].

5) Data Augmentation: Data augmentation is useful when there isn't enough data to train a neural network, as it improves network generalisation in situations where deep neural networks require big training datasets. To accomplish this, random transformations are applied to the data, such as cropping, rotating, or flipping images, to create new training examples [33].

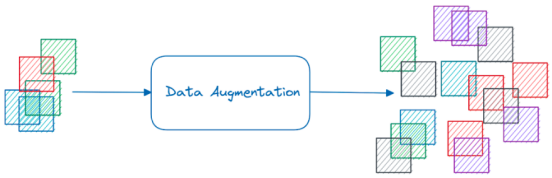


Fig. 13: Data Augmentation [32].

G. Optimizers

Optimizers are mathematical functions that are used to minimize the error in order to enhance the system efficiency. They mainly are effected by model's learnable parameters i.e Weights & Biases. They aid in reducing losses by understanding how to alter the neural network's weights and learning rate.

1) Gradient Descent: Gradient descent is an optimisation algorithm that minimises a given function to its local minimum by iteratively adjusting its parameters based on a convex function. Gradient Descent moves in the opposite direction of the sharpest climb, reducing a loss function iteratively. To discover minima, it depends on the derivatives of the loss function [29]. It is computer as:

$$W_{\text{new}} = W_{\text{old}} - \alpha \frac{\partial \text{Loss}}{\partial W_{\text{old}}} \tag{11}$$

Advantages

1) Easy to understand.
2) Easy to implement.

Disadvantages

1) Very slow.
2) Requires large memory.

2) Adam Optimizer: A popular optimizer computes adaptive learning rates for each parameter. When working with complex problems requiring several data points or parameters, the approach is incredibly effective. It is effective and uses little memory. It makes logical sense to combine the "gradient descent with momentum" and "RMSP" algorithms [29].

The update rule for the parameters using Adam is given by:

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{S_{dw_t}} - \epsilon} * V_{dw_t} \tag{12}$$

$$b_t = b_{t-1} - \frac{\eta}{\sqrt{S_{db_t}} - \epsilon} * V_{db_t} \tag{13}$$

These equations show the update rules for the weight w_t and bias b_t at time step t.

Advantages

1) Easy to implement
2) Computationally efficient.
3) Little memory requirements.

III. Related Work

A variety of datasets have been developed for Arabic character recognition. The IFN/ENIT database, created by the Institut für Nachrichtentechnik and the Ecole Nationale d'Ingénieurs de Tunis, includes 26,549 images of Tunisian town names, contributed by 411 writers [34]. Another resource, the Arabic Handwriting Database (AHDB), offers images depicting numerical words commonly found in checks [35]. Additionally, the Center for Pattern Recognition and Machine Intelligence (CENPARMI) provides several Arabic character databases, including those for Arabic cheque recognition [36] and offline handwriting recognition [37].

Significant progress has been made in Arabic handwriting recognition. For numeral recognition, Das et al. developed an 88-feature model and utilized a multi-layer perceptron (MLP)

classifier trained with back propagation, achieving 94.93% accuracy on the CMATERDB 3.3.1 dataset [38]. Ashiquzzaman et al. enhanced MLP performance by incorporating the ReLU activation function and the softmax classification layer and compared it with a CNN model, also trained using back-propagation on the same dataset. The CNN outperformed both MLP versions, reaching 97.4% accuracy, while the MLPs both achieved 93.8% [39]. Furthermore, the use of a Restricted Boltzmann Machine (RBM) in conjunction with a CNN, applied to the same dataset, yielded an even higher accuracy of 98.59% [40].

In 2018, Latif et al. proposed a new CNN architecture for recognizing mixed numerals in multiple languages, including Eastern Arabic, Persian, Devanagari, Urdu, and Western Arabic. This architecture consisted of an input layer matching the 28x28 pixel image size, followed by two hidden convolution layers using a 5x5 kernel, and subsequent maxpool layers with a 2x2 kernel. The model demonstrated a remarkable overall accuracy of 99.26%, with individual language accuracies averaging at 99.322% [41].

In our research we are going through three main papers with our introduced baseline, which is introduced just in terms of the project requirements. We will go through them with discussing their architectures.

The paper "Arabic Handwritten Character Recognition Using Convolutional Neural Networks" [13] focuses on developing a new model, CNN-14, for recognizing handwritten Arabic characters. This model is distinctive due to its 14-layer architecture, including convolutional, max-pooling, and fully connected layers. The authors used two datasets for training and testing: the Arabic Handwritten Character Dataset (AHCD) [42] and the Hijja dataset [5]. They achieved high accuracy rates of 99.36% on AHCD and 94.35% on the Hijja dataset.

The paper introduces a new 14-layer neural network called CNN-14, specifically designed for recognizing Arabic characters. This model is built with a mix of 8 convolutional layers, 4 max pooling layers, and 2 flattening layers, see Figure 14. The process begins with preparing the images through steps like resizing, rotating, and adjusting zoom. These images are then fed into the model in a 32x32 pixel format in grayscale (as shown in Table I). The network includes two hidden layers: the first with 32 filters and the second with 64 filters, both using a 3x3 kernel size. Padding is applied to ensure the entire image is processed, and the 'relu' activation function is used in these layers. The final part of the model classifies the output into 28 classes for the AHCD dataset and 29 classes for the Hijja dataset, using the softmax function for each class representing an Arabic letter. The model's learning is optimized using the Adam optimizer with a learning rate of 0.001 with total 13,191,644 trainable parameters.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 32, 32]	640
Conv2d-2	[-1, 64, 32, 32]	36,928
MaxPool2d-3	[-1, 64, 16, 16]	0
Conv2d-4	[-1, 128, 16, 16]	73,856
Conv2d-5	[-1, 128, 16, 16]	147,584
MaxPool2d-6	[-1, 128, 8, 8]	0
Conv2d-7	[-1, 256, 8, 8]	295,168
Conv2d-8	[-1, 256, 8, 8]	590,080
MaxPool2d-9	[-1, 256, 4, 4]	0
Conv2d-10	[-1, 512, 4, 4]	1,180,160
Conv2d-11	[-1, 512, 4, 4]	2,359,808
MaxPool2d-12	[-1, 512, 2, 2]	0
Linear-13	[-1, 4096]	8,392,704
Dropout-14	[-1, 4096]	0
Linear-15	[-1, 28]	114,716

TABLE I: Neural Network Architecture

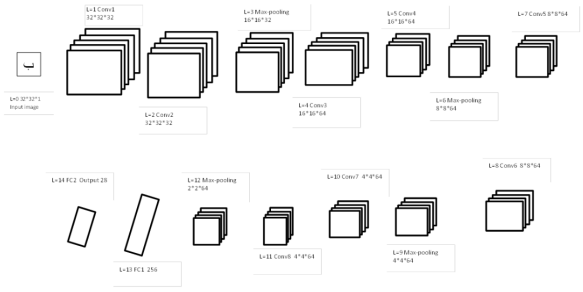


Fig. 14: CNN-14 Architecture

Alwagdani et al. [14] at their work proposed unique architecture featuring a dropout rate of 0.2 after all convolution and max-pooling layers, enhancing the model's ability to generalize and reduce overfitting. Notably, the architecture comprises two dense layers positioned at the end of the network. The first of these dense layers contains 512 neurons and utilizes a ReLU activation function, providing the necessary non-linearity for complex feature extraction. In contrast, the second dense layer bifurcates to address two distinct tasks: character recognition and writer-group classification. For character recognition, the layer consists of 28 neurons, while for writer-group classification, it includes two neurons. Both tasks employ a Softmax activation function for the final classification. Morover, the authors introduced a higher dropout rate of 0.4 after the first dense layer, further reinforcing the robustness of their model against overfitting. See Table II for details.

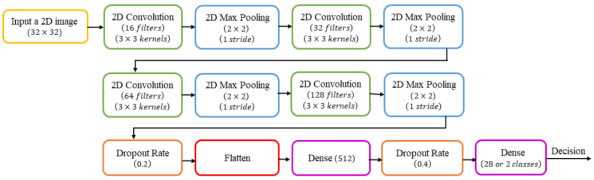


Fig. 15: Custom Architecture by Alwagdani et al. [14]

The model was trained with a batch size of 80 and for 100 epochs. They also used a technique called ReduceonLRP-Plateau in the Keras library to gradually decrease the learning rate. This decrease started from 0.001 and went down to 0.00001, reducing by a factor of 0.1 each time. Similarly, for the Feed-Forward Neural Network (FFNN) model, they used the same batch size and number of epochs. Here, they used the Nadam optimizer [43] and a binary cross-entropy loss function [33]. For this architecture, it have 927,004 trainable parameters.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 32, 32]	320
MaxPool2d-2	[-1, 32, 16, 16]	0
Conv2d-3	[-1, 64, 16, 16]	18,496
MaxPool2d-4	[-1, 64, 8, 8]	0
Conv2d-5	[-1, 128, 8, 8]	73,856
MaxPool2d-6	[-1, 128, 4, 4]	0
Conv2d-7	[-1, 256, 4, 4]	295,168
MaxPool2d-8	[-1, 256, 2, 2]	0
Linear-9	[-1, 512]	524,800
Linear-10	[-1, 28]	14,364

TABLE II: Alwagdani et al. [14] Detailed Architecture.

The last paper on our research is "Arabic handwriting recognition system using convolutional neural network." [5]. The have introduced ArabicCharCNN, which is an advanced convolutional neural network designed for Arabic character classification. Its architecture features three convolutional layers with increasing output channels (64, 128, 256), each followed by batch normalization and max pooling to enhance feature extraction and reduce dimensions. The network includes four fully connected layers, transitioning from 4096 to 28 output units, corresponding to the character classes. Notably, it employs a high dropout rate (0.8) after each dense layer (except the final one) to prevent overfitting. The model is sizable, with a total of 21,886,364 trainable parameters.

The last thing to talk about is the used pre-trained models, we have used the well known architectures Alexnet, Resnet18, VGG that trained on EMNIST dataset. We have got the files from this github repository [44].

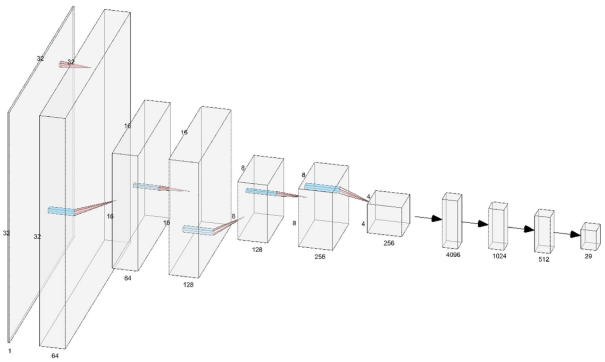


Fig. 16: ArabicCharCNN Architecture by Altwaijry et al. [5]

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 32, 32]	640
BatchNorm2d-2	[-1, 64, 32, 32]	128
MaxPool2d-3	[-1, 64, 16, 16]	0
Conv2d-4	[-1, 128, 16, 16]	73,856
BatchNorm2d-5	[-1, 128, 16, 16]	256
MaxPool2d-6	[-1, 128, 8, 8]	0
Conv2d-7	[-1, 256, 8, 8]	295,168
BatchNorm2d-8	[-1, 256, 8, 8]	512
MaxPool2d-9	[-1, 256, 4, 4]	0
Linear-10	[-1, 4096]	16,781,312
Dropout-11	[-1, 4096]	0
Linear-12	[-1, 1024]	4,195,328
Dropout-13	[-1, 1024]	0
Linear-14	[-1, 512]	524,800
Dropout-15	[-1, 512]	0
Linear-16	[-1, 28]	14,364

TABLE III: ArabicCharCNN Architecture

IV. Experiment

A. Experimental setup

All the models were tuned using the same set of hyperparameters, shown in Table IV, with using an early stop condition if there is no improvement on the models' performance for 5 epochs. The total number of epochs depends on the proposed architectures from the literature. All the models were built using Pytorch framework and trained on an NVIDIA GTX 1060 GPU.

B. Evaluation Metrics

1) Accuracy: The accuracy metric provides an overall assessment of the model's ability to correctly predict the output on the complete dataset. Each individual in the dataset

Hyperparameter	Values	Description
Learning Rate (lr)	0.001, 0.0005	The step size for updating network weights during training.
Weight Decay	1e-4, 1e-5, 0	Regularization parameter to prevent overfitting by penalizing large weights.
Validation Ratio (val_ratio)	0.2	The proportion of the dataset to be used for validation.

TABLE IV: Hyperparameters and Their Descriptions

contributes equally to the accuracy score, as every unit holds the same weight [45].

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{14}$$

2) Precision: Precision refers to the number of correct positive results divided by the number of all positive results.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{15}$$

3) Recall: Recall refers to the number of correct positive results divided by the number of positive results that should have been returned.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{16}$$

4) F1-Score: The F1-score can be seen as a weighted average of Precision and Recall. Precision and Recall hold equal importance in determining the F1-score, and the harmonic mean helps find the optimal balance between these two metrics [45].

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{17}$$

C. Dataset

The dataset was firstly proposed by El-Sawy et al. [46] on their paper. The dataset were provided by our instructor Dr. Aziz Qaroush.

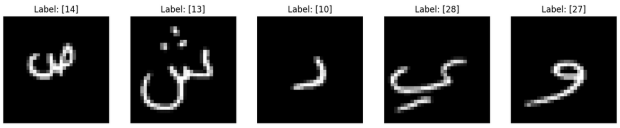


Fig. 17: Samples from the dataset

D. Results

All models were tested with the same dataset under the same setup. The code is provided on this github repositoy. We have tested the three models represented on the papers using and without using data augmentation. Moreover, tested our baseline model. We will introduce the results as the tasks order in the project description.

1) Task 1: This task is mainly training the models without any data augmentations. The BaselineCNN architecture starts with two convolutional layers (conv1 and conv2), each defined with 32 and 64 filters respectively, kernel size 3, stride 1, and padding 1. After each convolutional layer, a max-pooling layer (pool) with a 2x2 window and stride of 2 reduces the spatial size, condensing the features.

The network employs a dropout layer (dropout) with a rate of 0.5 to mitigate overfitting by randomly deactivating neurons during training, thereby forcing the network to learn more robust features. This is particularly useful in preventing the model from relying too heavily on any single feature. The best parameters for this model were $ls = 0.0005$ with no weight decay and batch size of 32 batches. After that, we have trained the model on the resulted parameters on 20 epochs and got 91.31% accuracy with the following curves on Figure 18. The figure shows training on 20 epochs in order to view the effect of over training the model. After the epoch 15 we can see that the testing accuracy decreases while the training accuracy increases which indicate the overfitting behavior of the model. Moreover, looking to the loss we can see how the validation loss was nearly constant after the epoch number 15. For the first paper, we have tested it under the

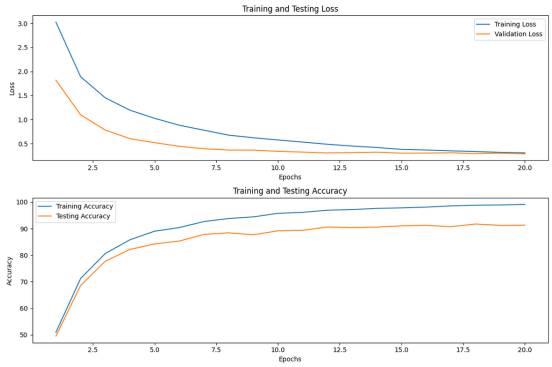


Fig. 18: Loss and Accuracy Curves of Baseline

best parameters after hyperparameter tuning, which was lr = 0.0005 and $weight\_decay = 1e-05$. We have got 96.01% accuracy, which is better than our baseline. On Figure 19, we can see that the model stopped on the epoch number 16 with an early stopping condition. Which indicates faster convergence with better results due to the feature extraction abilities by this model.

For CustomCNN model, the best parameters were lr = 0.001 and $weight\_decay = 1e-05$. It got 95.29% accuracy on the

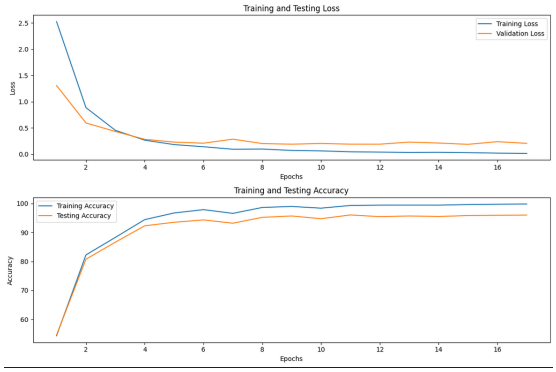


Fig. 19: Loss and Accuracy Curves of CNN14

testing data stopping at the 16^{th} epoch which is clear on Figure 20.

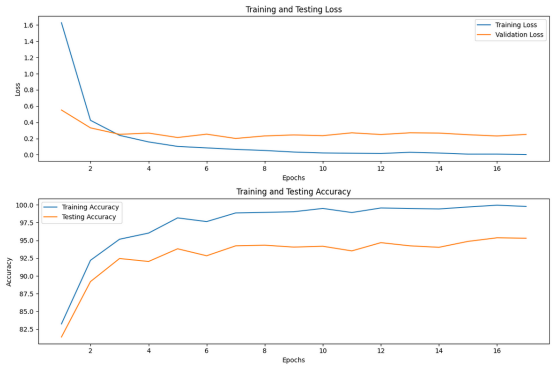


Fig. 20: Loss and Accuracy Curves of CustomCNN

For the last model, ArabicCharCNN, the best parmeters were lr = 0.001 and without weight decay. The model needed more epochs to converge, this is due to its deep architecture. As Figure 21 shows, we can see that it needed about 55 epochs to converge, which is more than previous approaches. It got accuracy of 96.27%.

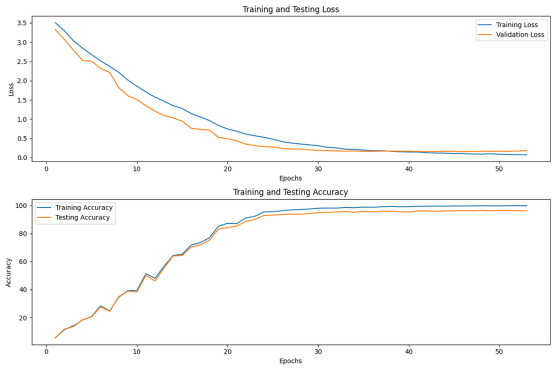


Fig. 21: Loss and Accuracy Curves of ArabicCharCNN

2) Task 2: For this task, we have used three different data augmentation techniques. Which were: Random Rotation by 5 degrees, translation, and random scaling. We have added the new data to our existing dataset. For each image we have applied only random type of augmentation. All the models were trained using the same parameters obtained from the hyperparameter tuning on the last step. The Baseline model got 94.91% accuracy, which is much better than before. As shown in Figure 22, we can see that the model needed more epochs than before as the number of data was increased. But with better performance for both training and testing.

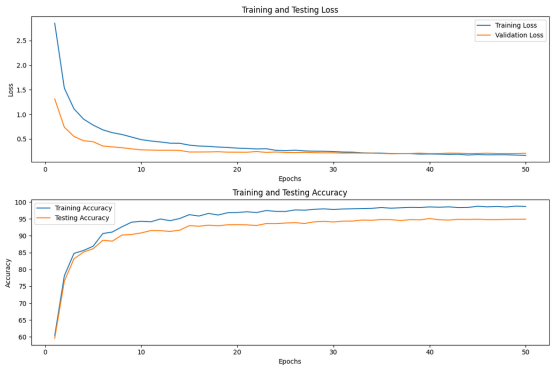


Fig. 22: Loss and Accuracy Curves of Baseline with data augmentation

For the first paper, the results were slightly better of 96.66% accuracy with 22 epochs. As Figure 23 suggests, the curves are very similar to the one without data augmentation but with more epochs.

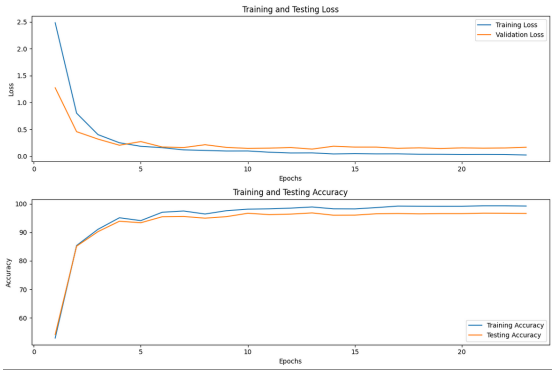


Fig. 23: Loss and Accuracy Curves of CNN14 with data augmentation

The second paper got 96.84%, which was the highest improvement among the three approaches. This tends to the simple architecture with the regularization techniques. Meanwhile, the third paper got 94.82% accuracy, this tends to the complex architecture and the large number of fully connected layers, that tends to over fit the data, especially after data augmentation. See Figures 24 and 25

3) Task 3: For this task, we have used Alexnet, Resnet18, Resnet152 using both non-trained and pre-trained models on

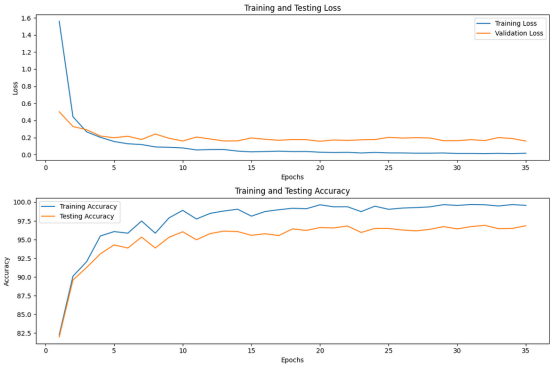


Fig. 24: Loss and Accuracy Curves of CustomCNN with data augmentation

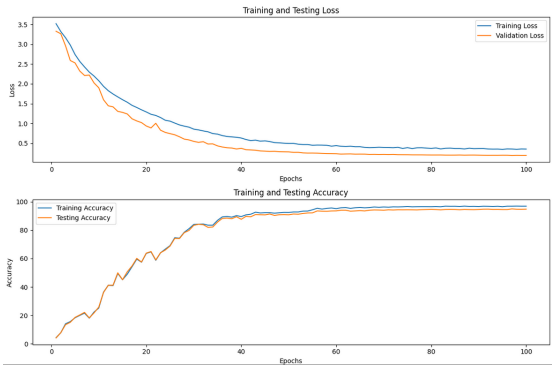


Fig. 25: Loss and Accuracy Curves of ArabicCharCNN with data augmentation

imagenet to test their performance. Surly we have changed the last layer to output 28 class. For all model, we hace used learning rate = 0.0001 and weight decay = 0.0001. On AlexNet, it got 91.27% accuracy and the pre-trained amazingly got 3.57%, which is getting the same output for all classes, this is due to the bias of the weights toward a far feature extraction outputs. Figure 26 shows that the model needed 18 epochs to converge.

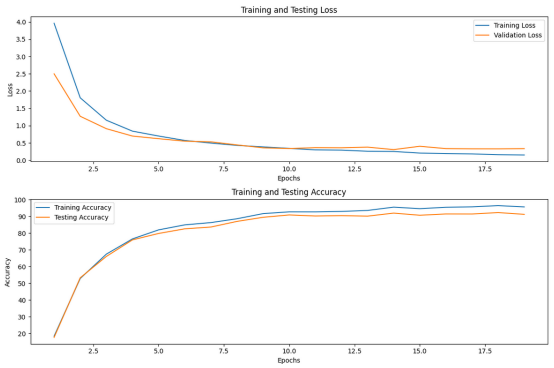
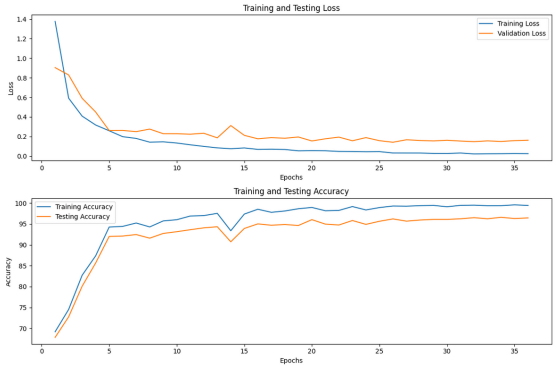


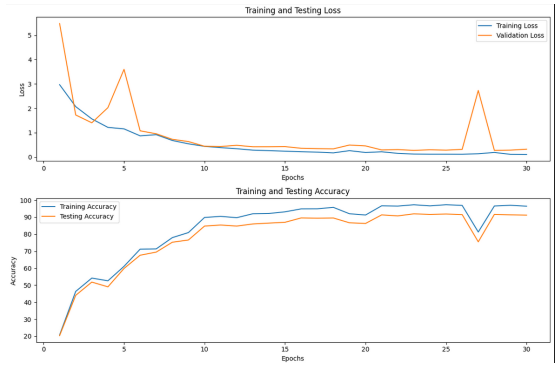
Fig. 26: Loss and Accuracy Curves of Alexnet with data augmentation

For Resnet18 and Resnet152, the model got 96.45%, 91.19% respectively. We can see that the Resnet18 outperformed Resnet152 because the dataset doesn't need that number of layers because it is not that complex. The two Figures 27a and 27b below show the curves of the two models. We can see that Resnet152 struggles on learning due to its complex arcitecture while Resnet18 has more stable learning. Using pre-trained models, Resnet18 got 95.7% accuracy.

Fig. 27: Loss and Accuracy Curves with Data Augmentation



(a) Resnet18



(b) Resnet152

4) Task 4: On this task, we have fine-tuned models that were trained on EMNIST dataset. Alexnet got 97.5% accuracy, which is much better than previous two setups for it. While Resnet got 99.07% for the pre-trained model. Figure 28 shows that alexnet got this accuracy with more number of epochs and more stable learning.

E. Discussion

The results of our experiments present insights into the performance of different convolutional neural network (CNN) architectures and the impact of data augmentation and pre-training on model accuracy.

1) Task 1: Training without Data Augmentation: Our BaselineCNN model, featuring a simple architecture with two convolutional layers followed by max-pooling and dropout

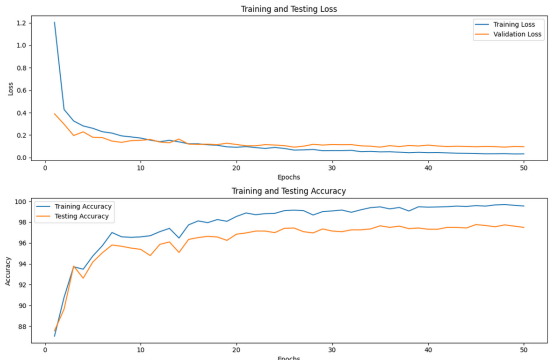


Fig. 28: Loss and Accuracy Curves of Alexnet with data augmentation using pre-trained model

layers, achieved a respectable accuracy of 91.3%. However, the model exhibited overfitting beyond 15 epochs, as indicated by the divergence of training and testing accuracy and the stop of validation loss. This behavior underscores the model's limited capacity to generalize beyond the training data without over-relying on specific features.

In comparison, the model from the first paper outperformed our baseline by achieving 96.01% accuracy. Its early stopping at epoch 16 suggests a more efficient learning process, likely due to superior feature extraction and generalization capabilities.

The CustomCNN and ArabicCharCNN models also outperformed the baseline but required different epochs for convergence, reflecting their varying architectural complexities. The deep architecture of ArabicCharCNN, in particular, necessitated more epochs (55) for convergence, indicating a potentially more intricate learning process but resulting in the highest accuracy of 96.27% among the models tested without data augmentation.

2) Task 2: Training with Data Augmentation: Introducing data augmentation improved the performance of all models, demonstrating the effectiveness of augmentation techniques in enhancing the generalizability and robustness of CNNs. The Baseline model's accuracy increased to 94.91%, benefiting from the augmented dataset's diversity, though it required more epochs to train due to the increased data volume.

The first paper's model and CustomCNN showed marginal improvements in accuracy with data augmentation, which might be due to their already high baseline performance. In contrast, the ArabicCharCNN model's performance slightly decreased, possibly due to its complex architecture and the added variability from data augmentation leading to overfitting.

3) Task 3: Testing with Pre-trained and Non-trained Models: The use of pre-trained and non-trained AlexNet, Resnet18, and Resnet152 models on our dataset yielded varied results. AlexNet's performance was moderate, but its pre-trained version performed poorly, likely due to pre-existing biases in the weights that were not suitable for our dataset's characteristics.

Resnet18 showed an excellent balance of complexity and performance, achieving higher accuracy than Resnet152. This suggests that the latter's deeper architecture was unnecessary and possibly negative for our dataset, which didn't require such a level of feature abstraction. The pre-trained Resnet18 further confirmed the utility of transfer learning, albeit with a slight decrease in performance compared to its non-trained counterpart.

4) Task 4: Fine-tuning Pre-trained Models: Fine-tuning models pre-trained on the EMNIST dataset yielded the highest accuracies in our experiments. AlexNet's performance improved significantly, demonstrating the impact of relevant pre-training. Resnet, when pre-trained, achieved an impressive accuracy of 99.07%, showcasing the power of transfer learning when pre-training and fine-tuning are aligned with the characteristics of the target dataset.

In summary, our experiments highlight the importance of model architecture, data augmentation, and pre-training in developing effective CNNs for specific tasks. While simpler models can achieve good performance, advanced models with appropriate training strategies can significantly outperform them. Moreover, data augmentation and pre-training are crucial tools for enhancing model robustness and generalizability, especially in the context of limited or specific datasets.

V. Conclusion

In this study, we did a comprehensive comparative analysis of different convolutional neural network (CNN) designs focusing on Arabic character recognition. Our study had a range of configurations, including various CNN architectures and the application of techniques such as transfer learning and data augmentation. The results showed the effectiveness of CNN architecture in image recognition tasks. Our BaselineCNN, which is very simple, was outperformed by literature models like ArabicCharCNN and CNN14. Clearly, ArabicCharCNN, despite its longer convergence time due to its deeper design, achieved the highest accuracy without data augmentation. While the introduction of data augmentation enhanced the performance of all models, with our Baseline model showing a particularly notable improvement. However, this technique slightly reduced the efficacy of the ArabicCharCNN, suggesting a potential issue with overfitting.

Further analysis involved exploring the effects of employing pre-trained and untrained models, such as variations of Resnet

and AlexNet, on Arabic character recognition. This exploration revealed the limitations of applying pre-existing network biases to new datasets, as shown by the weaker performance of the pre-trained AlexNet. In contrast, Resnet18, although more complex, outperformed the deeper Resnet152, highlighting that increased complexity does not always provide better performance. The marginal performance differences between pre-trained and non-trained models underscored the value of transfer learning, though it also ensured on the need for careful selection of pre-training that aligns with the target dataset. The highest accuracy was achieved by optimizing pre-trained models on the EMNIST dataset, showing the effectiveness of fine-tuning pre-trained models to adapt to specific dataset characteristics.

REFERENCES

[1] A. S. Abdalkafor, "Survey for databases on arabic off-line handwritten characters recognition system," in *2018 1st International Conference on Computer Applications & Information Security (ICCAIS)*. IEEE, 2018, pp. 1–6.

[2] W. Albattah and S. Albahli, "Intelligent arabic handwriting recognition using different standalone and hybrid cnn architectures," *Applied Sciences*, vol. 12, no. 19, p. 10155, 2022.

[3] A. A. A. Ali, M. Suresha, and H. A. M. Ahmed, "A survey on arabic handwritten character recognition," *SN Computer Science*, vol. 1, no. 3, p. 152, 2020.

[4] H. M. Balaha, H. A. Ali, and M. Badawy, "Automatic recognition of handwritten arabic characters: a comprehensive review," *Neural Computing and Applications*, vol. 33, pp. 3011–3034, 2021.

[5] N. Altwaijry and I. Al-Turaiki, "Arabic handwriting recognition system using convolutional neural network," *Neural Computing and Applications*, vol. 33, no. 7, pp. 2249–2261, 2021.

[6] I. Ahmad and G. A. Fink, "Class-based contextual modeling for handwritten arabic text recognition," in *2016 15th international conference on frontiers in handwriting recognition (ICFHR)*. IEEE, 2016, pp. 554–559.

[7] A. Baldominos, Y. Saez, and P. Isasi, "A survey of handwritten character recognition with mnist and emnist," *Applied Sciences*, vol. 9, no. 15, p. 3169, 2019.

[8] M. Ramzan, H. U. Khan, S. M. Awan, W. Akhtar, M. Ilyas, A. Mahmood, and A. Zamir, "A survey on using neural network based algorithms for hand written digit recognition," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 9, 2018.

[9] A. Ashiquzzaman, A. K. Tushar, A. Rahman, and F. Mohsin, "An efficient recognition method for handwritten arabic numerals using cnn with data augmentation and dropout," in *Data Management, Analytics and Innovation: Proceedings of ICDMAI 2018, Volume 1*. Springer, 2019, pp. 299–309.

[10] R. Weipert, "The oxford handbook of arabic linguistics.(the oxford handbooks in linguistics)," 2014.

[11] R. Plamondon, D. P. Lopresti, L. R. Schomaker, and R. Srihari, "Online handwriting recognition," *Wiley encyclopedia of electrical and electronics engineering*, vol. 15, pp. 123–146, 1999.

[12] L. Schomaker, "Advances in writer identification and verification," in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, vol. 2. IEEE, 2007, pp. 1268–1273.

[13] A. Alsayed, C. Li, A. Fat'hAlalim, M. Abdalsalam, and Z. Obied, "Arabic handwritten character recognition using convolutional neural networks," *None*, 2023.

[14] M. S. Alwagdani and E. S. Jaha, "Deep learning-based child handwritten arabic character recognition and handwriting discrimination," *Sensors*, vol. 23, no. 15, p. 6774, 2023.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[19] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, "Emnist: Extending mnist to handwritten letters," in *2017 international joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 2921–2926.

[20] IBM, "What are neural networks?" accessed: date-of-access. [Online]. Available: https://www.ibm.com/topics/neural-networks

[21] F. Bre, J. M. Gimenez, and V. D. Fachinotti, "Prediction of wind pressure coefficients on building surfaces using artificial neural networks," *Energy and Buildings*, vol. 158, pp. 1429–1441, 2018.

[22] K. V. Artificial neural network, its inspiration and the working mechanism. Accessed: date-of-access. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/04/artificial-neural-network-its-inspiration-and-the-working-mechanism/

[23] M. Mishra. (2021, Dec.) Convolutional neural networks, explained. Accessed on [28-01-2024]. [Online]. Available: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

[24] K. K. Sahoo, R. Hazra, M. F. Ijaz, S. Kim, P. K. Singh, and M. Mahmud, "Mic_fuzzynet: Fuzzy integral based ensemble for automatic classification of musical instruments from audio signals," *IEEE Access*, vol. 10, pp. 100 797–100 811, 2022.

[25] A. Umer. (2023) Understanding convolutional neural networks: A beginner's journey into the architecture. [Online]. Available: https://medium.com/codex/understanding-convolutional-neural-networks-a-beginners-journey-into-the-architecture-aab30dface10

[26] S. Sharma, "Activation functions in neural networks - towards data science," 11 2022. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[27] S. Gupta, "The 7 most common machine learning loss functions," 6 2023. [Online]. Available: https://builtin.com/machine-learning/common-loss-functions

[28] E. Zvornicanin and E. Zvornicanin, "Relation between learning rate and batch size — Baeldung on Computer Science," 3 2023. [Online]. Available: https://www.baeldung.com/cs/learning-rate-batch-size

[29] J. Jordan, "Setting the learning rate of your neural network." 3 2023. [Online]. Available: https://www.jeremyjordan.me/nn-learning-rate/

[30] A. Chawla, "Gradient accumulation: Increase batch size without explicitly increasing batch size," 10 2023. [Online]. Available: https://www.blog.dailydoseofds.com/p/gradient-accumulation-increase-batch

[31] S. Karagiannakos, "Regularization techniques for training deep neural networks — AI Summer," 5 2021. [Online]. Available: https://theaisummer.com/regularization/

[32] B. P. C, "Regularization in neural networks," accessed on [29-01-2024]. [Online]. Available: https://www.pinecone.io/learn/regularization-in-neural-networks/

[33] U. Ruby and V. Yendapalli, "Binary cross entropy with deep learning technique for image classification," *Int. J. Adv. Trends Comput. Sci. Eng*, vol. 9, no. 10, 2020.

[34] G. Dimauro, S. Impedovo, R. Modugno, and G. Pirlo, "A new database for research on bank-check processing," in *Proceedings eighth international workshop on frontiers in handwriting recognition*. IEEE, 2002, pp. 524–528.

[35] S. Al-Ma'adeed, D. Elliman, and C. A. Higgins, "A data base for arabic handwritten text recognition research," in *Proceedings eighth international workshop on frontiers in handwriting recognition*. IEEE, 2002, pp. 485–489.

[36] S. Abdleazeem and E. El-Sherif, "Arabic handwritten digit recognition," *International Journal of Document Analysis and Recognition (IJDAR)*, vol. 11, pp. 127–141, 2008.

[37] A. Huda, J. Sadri, C. Suen, and N. Nobile, "A novel comprehensive database for arabic off-line handwriting recognition," in *Proceedings of 11th International Conference on Frontiers in Handwriting Recognition, ICFHR*, vol. 8, 2008, pp. 664–669.

[38] N. Das, A. F. Mollah, S. Saha, and S. S. Haque, "Handwritten arabic numeral recognition using a multi layer perceptron," *arXiv preprint arXiv:1003.1891*, 2010.

[39] A. Ashiquzzaman and A. K. Tushar, "Handwritten arabic numeral recognition using deep learning neural networks," in *2017 IEEE International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*. IEEE, 2017, pp. 1–4.

[40] A. A. Alani, "Arabic handwritten digit recognition based on restricted boltzmann machine and convolutional neural networks," *Information*, vol. 8, no. 4, p. 142, 2017.

[41] G. Latif, J. Alghazo, L. Alzubaidi, M. M. Naseer, and Y. Alghazo, "Deep convolutional neural network for recognition of unified multi-language handwritten numerals," in *2018 IEEE 2nd International workshop on Arabic and derived script analysis and recognition (ASAR)*. IEEE, 2018, pp. 90–95.

[42] M. Alheraki, R. Al-Matham, and H. Al-Khalifa, "Handwritten arabic character recognition for children writing using convolutional neural network and stroke identification," *Human-Centric Intelligent Systems*, pp. 1–13, 2023.

[43] A. Tato and R. Nkambou, "Improving adam optimizer," 2018.

[44] XavierSpycy, "Github - xavierspycy/emnist-classifier: Handwritten character recognition on emnist byclass using convolutional neural networks with pytorch." [Online]. Available: https://github.com/XavierSpycy/EMNIST-Classifier

[45] M. Grandini, E. Bagli, and G. Visani, "Metrics for multi-class classification: an overview," *arXiv preprint arXiv:2008.05756*, 2020.

[46] A. El-Sawy, M. Loey, and H. El-Bakry, "Arabic handwritten characters recognition using convolutional neural network," *WSEAS Transactions on Computer Research*, vol. 5, no. 1, pp. 11–19, 2017.

APPENDIX

	Model	Status	Accuracy	Avg_Precision	Avg_Recall	Avg_F1
1	BaselineCNN	No Augmentation	91.31	91.472	91.310	91.317
2	CNN14	No Augmentation	96.012	96.062	96.012	96.001
3	CustomCNN	No Augmentation	95.298	95.469	95.298	95.306
4	ArabicCharCNN	No Augmentation	96.270	96.298	96.012	96
5	BaselineCNN	Augmentation	94.911	94.984	94.911	94.918
6	CNN14	Augmentation	96.667	96.793	96.667	96.667
8	CustomCNN	Augmentation	96.845	96.888	96.845	96.838
9	ArabicCharCNN	Augmentation	94.821	95.073	94.821	94.808
10	AlexNet	Augmentation + Re-Train	91.27	91.012	91.298	91
11	ResNet18	Augmentation + Re-Train	96.45	96.888	96.012	96.12
12	AlexNet	Augmentation + Finetune Train On EMNIST	97.5	97.5	97.5	97.5
13	VGGnet	Augmentation + Finetune Train On EMNIST	97.827	97.863	97.827	97.828
14	Resnet	Augmentation + Finetune Train On EMNIST	99.07	98.888	99.012	98