Faculty of Engineering & Technology – Electrical & Computer Engineering Department

First Semester 2023 – 2024

Computer Vision – ENCS5343

Arabic Handwritten Character Recognition Using Convolutional Neural Networks

---

Prepared By:
Ahmaide Awawda – Ameer Bazzar
1190823 – 1191015

Instructor:
Mr. Aziz Qaroush

Date: January 2024

Abstract

The aim of this project is to build a conventional neural network (CNN) that recognizes all 28 handwritten Arabic characters and gives them the right labeling and classification given a data set that contains both training and validation data.

This project will include building a CNN with all the types of its layers (convolutional layer, pooling layers, activation functions, and fully connected layers). alongside optimizations, loss functions, training, evaluation, visualization, expanding the data set using data augmentation, alongside implementing the system on other pre-trained models.

The used programming language for this project is python with the help of Google's Collaboratory, as all the needed codes for this project can be found in the attached ipnyb file.

❖ Table of Content

List of Figures

1. Background

1.1. CNN Architecture

CNNs are a subset of deep learning architectures that are specifically engineered to process data in a grid pattern, such as images, automatically and efficiently.

The proficiency of CNN is in identifying hierarchical patterns, encompassing intricate shapes and basic edges, rendering them perfect for the analysis of visual data. Figure 1-1 shows how the architecture of the CNN looks like.

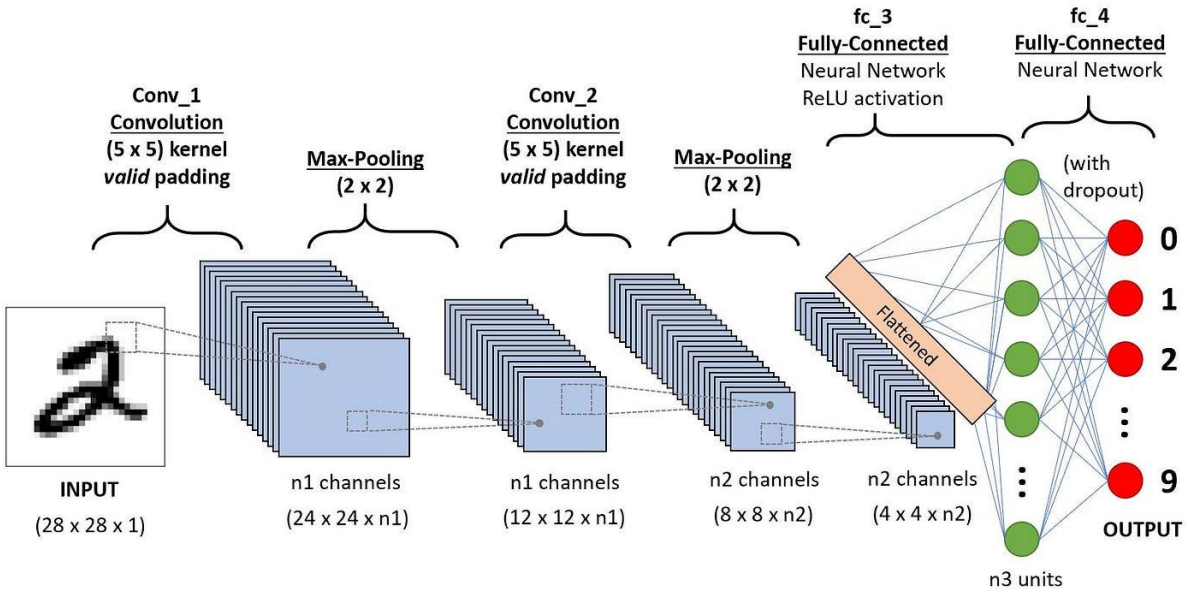

Figure 1-1: CNN Architecture [1]

1.2. CNN Layers

The fundamental components of a CNN, known as convolutional layers, oversee feature detection. They extract significant features from the input, like edges, corners, and textures, by applying different filters. These characteristics are necessary to comprehend the intricate picture structures.

Convolutional layers are followed by pooling layers, which lower the computational load and overfitting risk by shrinking the spatial dimensions of the feature maps. These extracted features are used by the fully connected layers at the end of the CNN to classify the input into various categories based on the patterns that are learned.

The convolutional layers are basically used for the image's features extraction (like edge detections for example) alongside the pooling layers, on the other hand the fully commented layers are used for the image's classification as shown in figure 1-2.

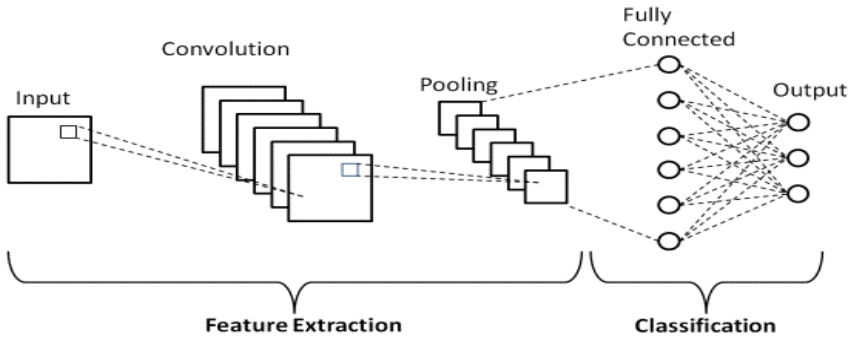

Figure 1-2: CNN's Layers [3]

1.3. Applications of CNNs

CNNs have revolutionized various fields including image and video recognition, image classification, medical image analysis, and natural language processing. Their capacity to identify complex patterns in pictures makes them especially helpful in domains like autonomous driving, facial recognition, and the analysis of visual imagery from different sources.

When it comes to handwritten character recognition CNNs can play a vital role as their capacity to handle the complexity and variability of handwriting. Despite the subtle differences in handwriting styles, CNNs aid in the accurate recognition and differentiation of many languages characters due to their distinct positional forms and dots. [4]

1.4. Methodology of CNNs in Arabic Handwritten Character Recognition

Usually, the process entails feeding the handwritten images into a CNN after preprocessing them using techniques like binarization and normalization. To correctly recognize the text, the network learns to distinguish characteristics of Arabic characters, accounting for their varied forms and connecting patterns. [5]

1.5. Data Augmentation in CNNs

Data augmentation uses transformations like rotation, scaling, and translation to artificially increase the diversity of the training dataset. This is important for handwritten recognition because it increases the model's accuracy and robustness by allowing it to learn from a variety of handwriting styles.

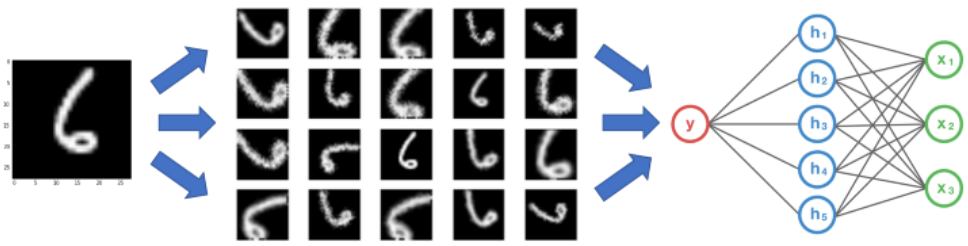

Figure 1-3: Data Augmentation [6]

1.6. Transfer Learning & Tuning

Transfer learning is the process of optimizing a previously trained CNN for a particular task by using it on a sizable dataset. This method can greatly shorten training times and increase accuracy for this project, particularly when there is a small amount of dataset available. Figure 1-3 shows the implementation of transfer learning.

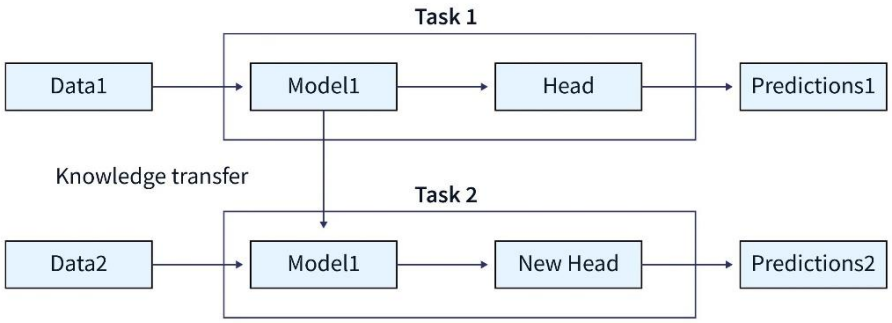

Figure 1-4: Transfer Learning [7]

It is essential to fine-tune parameters like learning rate, filter size, and layer count. It can have a significant effect on the model's performance, with different optimal settings depending on the features being used. [5]

1.7. Visualization Techniques in CNNs

CNNs' activation maps and filters can be seen to better understand how the network handles images. Higher-level filters capture more intricate patterns, while lower-level filters frequently identify basic features like edges. By visualizing these, one can ascertain which characteristics have the greatest influence on the network's decision-making.

Blocking portions of an image is known as image occlusion, and it is used to determine which areas are critical to the model's predictions. Saliency maps, on the other hand, provide information about the areas of a picture that the model is most interested in by using gradients to highlight those areas. [8]

2. Experimental Setup & Results

2.1. Used Libraries

For this handwritten character recognition project, many libraries were included to form the dataset, implement deep learning methods, dealing with complex mathematical issues, data preprocessing, and optimization, the used libraries were as follows:

- Pandas: used for reading the dataset values
- Numpy: used to apply mathematical operation on arrays and matrices.
- Tensorflow.eras: This library is used for creating, importing, optimizing, categorizing saving, loading, and plotting CNN models.models.
- MatPlotLib: This library is basically responsible for plotting graphs.
- Scippy: The use of this library is just to zoom into the images to increase the number of pixels.

2.2. Dataset

The given dataset contains 32x32 grey colored (0-255) images of handwritten Arabic characters, meaning that it has 28 labels (same number as the Arabic alphabetical characters).

Now of course the data set is split into training and testing images, as the number of testing images is 13440 images while the number of the training images is 3360 images, with equal number of images from each Arabic character (label), figure 2-1 presents how the image in the dataset looks like.

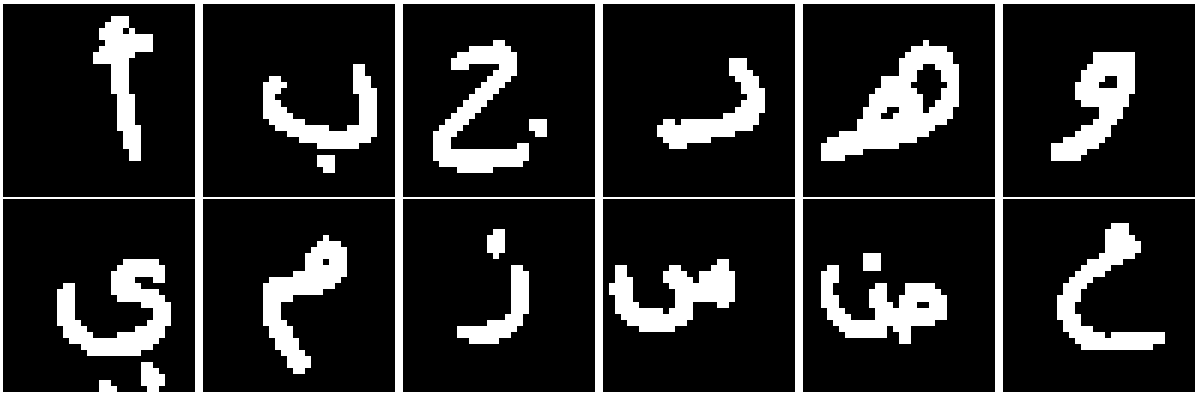

Figure 2-1: Dataset Format

The pixel values for the dataset are normalized from values between (0, 255) to be values between (0, 1).

2.3. Task 1: Building a Custom CNN

In this task the handwritten character recognition convolutional neural network is built with the setup of its architecture, layers, and hyperparameters, parameters.

2.3.1. Related Work

To start implementing CNN, first there must be some related resources that can help in building the needed model.

First there was a paper that talked about implementing a CNN-14 model using AHCL and Hijja datasets where the accuracy for the first dataset has accessed 99% with 100 epochs with 0.14 loss, while the second dataset reached 94% accuracy with 0.06 loss. [9]

Another paper implemented a deep learning-based child's HACR CNN alongside HOG-based features and statistical-based features reaching up to over 98% accuracy with 0.0.1 validation loss implemented with 100 epochs. [10]

Another paper from Neural Comput & Applic talked about the implementation of a deep leaning CNN for children's HACR yet this paaper also included the Arabic characters on different parts of the word which made the classification more complex. [11]

Checking those HACR CNN's help give a better understanding of the most suitable attributes for the system's architecture with layers and parameters, which helped building the current model.

2.3.2. Designing the CNN

Given that the CNN takes a 32x32 grey level image with an ending layer of 28 classifiers, the CNN was built as following:

First of all, three convolutional layers were added each one followed by a pooling layer (max pooling) where the first convolutional layer included 16 3x3 filters while the second one included 32 3x3 filters and the third one including 64 5x5 filters each one having the relu as its activation function, and as for the pooling layer the first one with one stride and the second and third one with two strides.

Those six layers are then followed by a dropout layer with 40% rate before flatting the data to be inserted into the fully connected layers where the first dense layer included 128 neurons with L2 regularization (to reduce the overfitting chances), with a learning rate of 0.0001 then a batch normalization is added with a 32 batch size , followed by another dropout layer with 40% rate and finally a dense layer that uses the softmax function to classify the outcome into the final 28 classifications.

The Adam optimizer with the learning rate of 0.0001 is implemented to the model at last before it is ready, Figure 2-2 shows a diagram of the CNN.

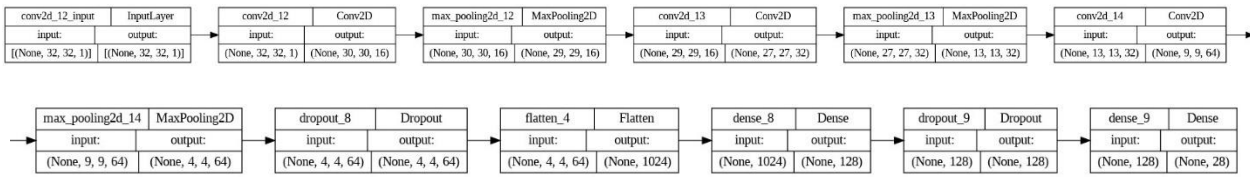

Figure 2-2: CNN Architecture Plot

The number of epochs for this CNN architecture is set to 100, as the image is trained with the data set 100 different times.

2.3.3. Results

After running the 100 epochs, the total parameters where 190876 parameters as they all are considered trainable, figure 2-3 shows the two plots for the accuracy and the loss for both the training and testing.

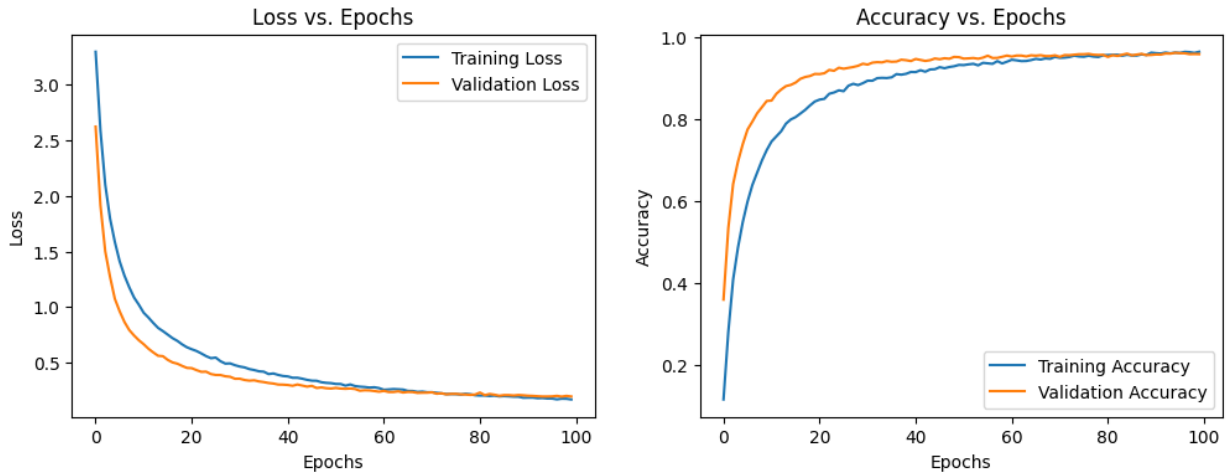

Figure 2-3: Custom CNN Results

As shown in figure 2-3, by approaching the 100th epoch the training and validations accuracy become close as the training accuracy exceeds the validation accuracy while the training loss goes beyond the validation loss, it is concluded that increasing the number of epochs will lead to an observing overfitting, so the current number of epochs is considered fair.

The results values were as follows:

Best Training Accuracy: 0.965, Best Training Loss: 0.17, Best Test Accuracy: 0.962, Best Test Loss: 0.196.

And from the best model from training:

Training Accuracy: 0.996, Training Loss: 0.196, Test Accuracy: 0.963, Test Loss: 0.197.

2.4. Task 2: Adding the Data Augmentation

In this task using the CNN from the previous part, the given dataset is expanded using the data augmentation aiming to get better results from the previous parts with more data being involved.

2.4.1. Image Transformation Generation

First, as the dataset had some changes, the number of epochs was increased to 300 aiming to achieve higher accuracy for both training and validation and less loss.

The data augmentation included adding a 5^{o} angle rotation alongside both height and width shift ranges with a value of 0.1 and a zoom range which is also 0.1, and the filling mode depended on the nearest pixels.

2.4.2. Results

After running the 300 epochs, the total parameters same as the previous part as they all are considered trainable, figure 2-4 shows the two plots for the accuracy and the loss for both the training and testing.

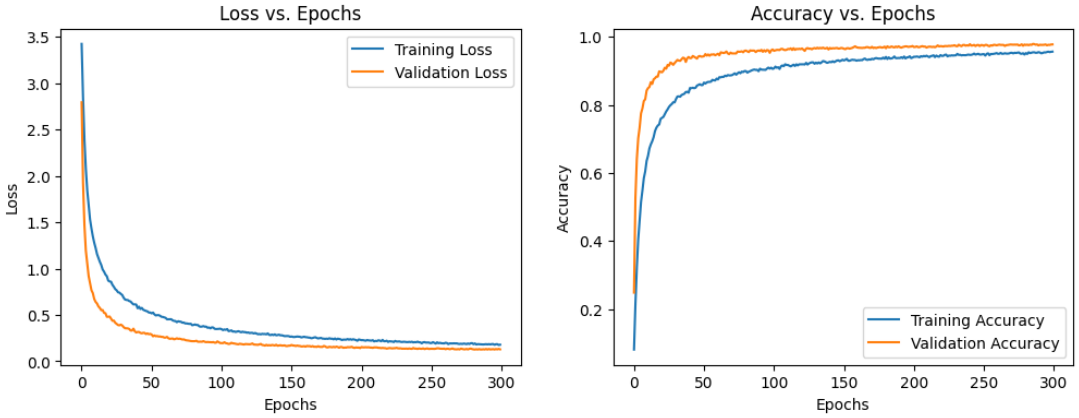

Figure 2-4: Custom CNN with Data Augmentation Results

As shown in figure 2-4, by reaching the 300^{th} epoch the training and validations accuracy settle down where the chances of having an overfitting here are most likely impossible, so it can be said that the number of epochs is good. Comparing it with the previous part, it can be said that the data augmentation has enhanced the accuracy and prevented overfitting.

The results values were as follows:

Best Training Accuracy: 0.957, Best Training Loss: 0.176, Best Test Accuracy: 0.979, Best Test Loss: 0.126.

And from the best model from training:

Training Accuracy: 0.99, Training Loss: 0.072, Test Accuracy: 0.979, Test Loss: 0.127.

2.5. Task 3: Using Well Known Network (MobileNet)

For this part the MobileNet CNN is used as a well-known published CNN where the focus in this task is on the tradeoff between accuracy and network complicity with respect to the given dataset.

2.5.1. MobileNet CNN

Google created MobileNet, a simplified convolutional neural network (CNN) architecture, in 2017. It balances accuracy and efficiency with a design tailored for mobile and embedded devices. Using depthwise separable convolutions, which lowers the model size and computational cost significantly when compared to standard convolutional networks, is what makes MobileNet unique. [12]

Figure 2-5 shows the architecture of the MobileNet CNN.

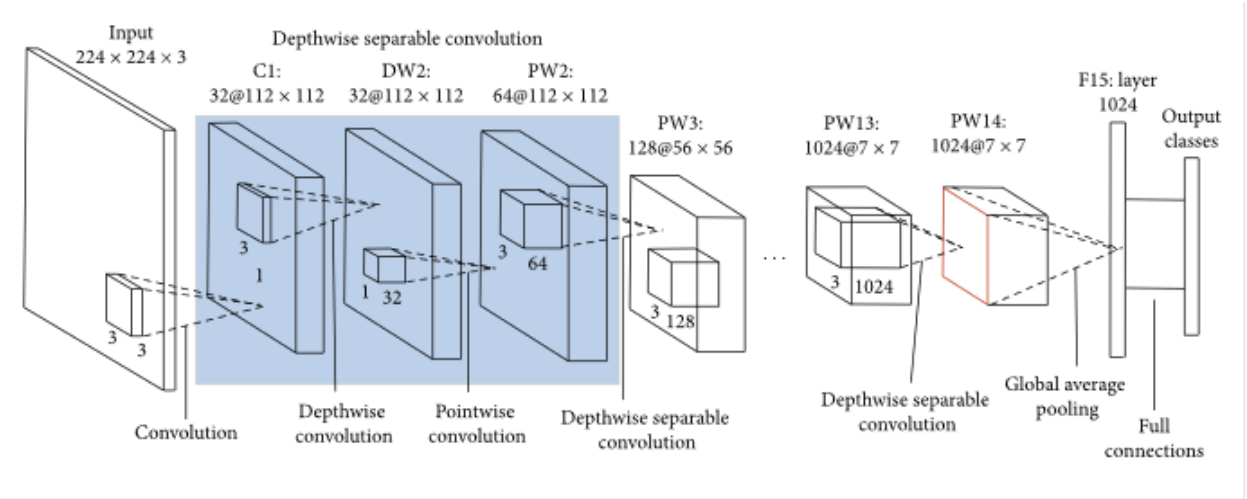

Figure 2-5: MobileNet CNN Architecture [13]

2.5.2. Added Layers & Training Hyperparameters

First this model was imported using the TenserFlow Keras Library extension. Now as known for the first layer that it shall expect a 32x32 grey color image where it has only one channel, so that needed to be set up. For the weights they were set based on the ImageNet dataset.

In this task the fully connected layers from the MobileNet CNN aren't taken, as after the taking the lower part (Convolutional parts with max pooling) a flatten layer is added followed by a 256 neurons dense layer with L2 regularization with a learning rate of 0.001, a 32-batch normalization is added before having a 0.2 dropout layer, and finally the classification layer which consists of 28 neurons. For the number of epochs in this task it is set to 50 with same data augmentation from the previous part. Figure 2-6 shows the layers of the CNN used in this task.

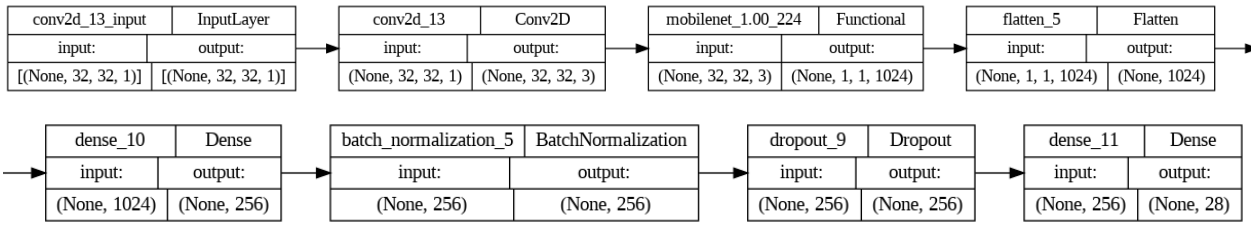

Figure 2-6: MobileNet Architecture Plot

2.5.3. Results

After running the 50 epochs, the total parameters were 3499514 where 3477114 were trainable and 22400 were not, figure 2-7 shows the two plots for the accuracy and the loss for both the training and testing.

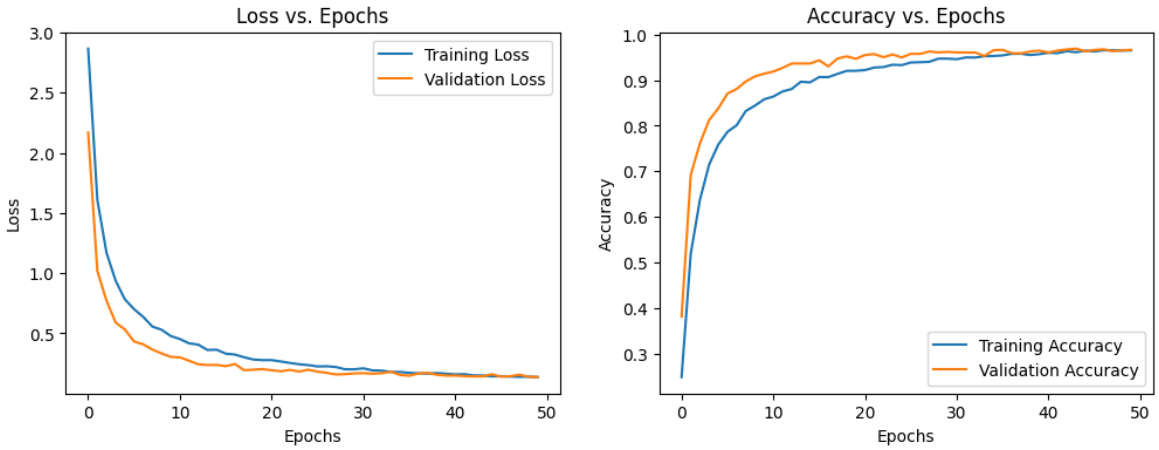

Figure 2-7: MobileNet Results

As shown in figure 2-7 by reaching the 50th epoch the training and validation accuracy and loss become nearly the same which can lead for an absolute overfitting with more epochs, although it needed less epochs to get to higher accuracies which is a big difference from the two previous parts using the custom CNN yet this one also had the fastest reaching for overfitting if more number of epochs were included.

The use of MobileNet did shows its effectiveness as it gave a fast-growing accuracy, with less complexity building the system as the layers were ready to observe, yet still as t was built to work on different datasets made its way to cause overfitting way shorter as at the end both accuracies were very close.

The results values were as follows:

Best Training Accuracy: 0.966, Best Training Loss: 0.139, Best Test Accuracy: 0.988, Best Test Loss: 0.128.

And from the best model from training:

Training Accuracy: 0.988, Training Loss: 0.077, Test Accuracy: 0.971, Test Loss: 0.128.

2.6. Task 4: Using Pre-Trained Network

For this task a pre-trained CNN that was used for a similar task to the task in this project (HACR) using the appropriate transfer learning method.

2.6.1. The Used CNN

The used CNN for this task was a pre-trained CNN for Handwritten hiragana recognition. Hiragana is basically a Japanese syllabary, part of the Japanese writing system. [14]

The used CNN comes very close to this project (both are handwritten character recognition) and having hiragana including dots in its characters makes the two datasets very similar and has a high chance of giving good results for this project.

The architecture of the CNN was made to take 64x64 grey color (one channel) images, and give out 71 classifications, resulting in 99% accuracy.

2.6.2. Added Layers, Training Hyperparameters & Tuning

First as the model takes 64x64 images, instead of editing the first layer of the CNN, the images in the dataset were resized to fit in as 64x64 images using the zoom function in the SciPy library.

For the CNN network layers since now the first layer matches the input images in the data set, the only layer that was removed is the last layer that contained 71 neurons (the number of hiragana characters), in order to enhance the performance of this CSS the fine parameter turning was implemented as the last layers in the fully connected part were extended with two layers as the first one has 1024 neurons and the second layer with 128 neurons followed by a 0.2 dropout and two dense layers where one had 64 neurons and the final one had the number of labels which is 28.

This part used the same data augmentation of the previous parts, with 150 epochs and a batch size of 32. Figure 2-8 shows the layers of the CNN used in this task.

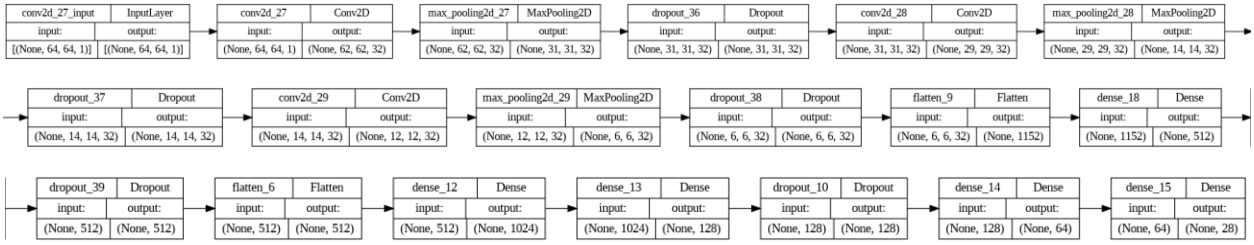

Figure 2-8: Pre-Trained CNN Architecture Plot

As shown in figure 2-8 due to the size of the images and the tuning layers the resulted network had 19 layers.

2.6.3. Results

After running the 150 epochs, the total parameters were 1275740 as they all were considered trainable, figure 2-9 shows the two plots for the accuracy and the loss for both the training and testing.

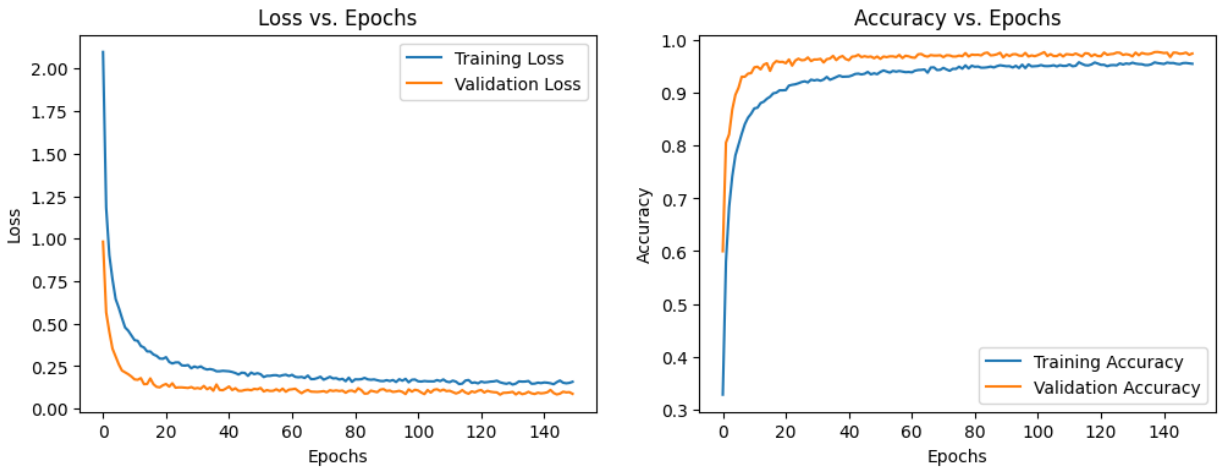

Figure 2-9: Using Pre-Trained Model Results

As shown in figure 2-9 by reaching the 150th epoch the training and validation accuracy and loss stop increasing which means there will be no further overfitting if the number of epochs has increased as the testing or validation accuracy is still higher than the validation accuracy.

The use of a pre-trained model based on its results it can be said that it is the best solution so far as it gave high accuracy for both training and validation and a low data lose for both of them too which gave it an advantage over both tasks 1 & 3, and for it to settle and have its highest values it needed 150 epochs which half of what task 2 needed,

Adding On using a pre-trained network gives flexibility as of not starting off at the zero point which was helpful in implementing the needed CNN model.

The results values were as follows:

Best Training Accuracy: 0.958, Best Training Loss: 0.141, Best Test Accuracy: 0.978, Best Test Loss: 0.082.

And from the best model from training:

Training Accuracy: 0.989, Training Loss: 0.037, Test Accuracy: 0.978, Test Loss: 0.094

3. Conclusion

In conclusion this project shows the power of convolutional neural networks in handwritten character recognition and how their layers architecture and parameters can be implemented as for the layers to be not fully connected can result in better results for the inserted image, and how accurate the results can be.

This project introduced the role of data augmentation in expanding the dataset that is fed to the network alongside the parameter tuning, visualization for error detection, overfitting avoidance methodologies, and finally the tradeoff between accuracy and network complexity.

Another thing that was concluded is that in the world of deep learning, not everything has to start from the baseline as that was shown in using a pretrained network with the implementation of the activation function as using a similar network that was trained for something else instead of starting off from the baseline resulted in less effort and better results.

4. References

[1]. Saha, S. (2022, November 16). *A comprehensive guide to Convolutional Neural Networks - the eli5 way*. Medium. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 Accessed on January 28th, 2024.

[2]. Keita, Z. (2023, November 14). *An introduction to Convolutional Neural Networks: A comprehensive guide to CNNS in Deep learning*. DataCamp. https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns Accessed on January 28th, 2024.

[3]. Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network | upGrad blog. (n.d.). *Basic CNN architecture: Explaining 5 layers of Convolutional Neural Network*. upGrad blog. https://www.upgrad.com/blog/basic-cnn-architecture/ Accessed on January 28th, 2024.

[4]. Zvornicanin, W. by: E. (2023, April 14). *How to design deep convolutional neural networks?* Baeldung on Computer Science. https://www.baeldung.com/cs/deep-cnn-design Accessed on January 29th, 2024.

[5]. Zvornicanin, W. by: E. (2023a, April 14). *How to design deep convolutional neural networks?*. Baeldung on Computer Science. https://www.baeldung.com/cs/deep-cnn-design Accessed on January 29th, 2024.

[6]. Gandhi, A. (2023, September 11). *Data augmentation: How to use deep learning when you have limited data*. Nanonets Intelligent Automation, and Business Process AI Blog. https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/ Accessed on January 28th, 2024.

[7]. Malingan, N. (2023, March 23). *Transfer learning*. Scaler Topics. https://www.scaler.com/topics/deep-learning/transfer-learning/ Accessed on January 28th, 2024.

[8]. Brownlee, J. (2019, July 5). *How to visualize filters and feature maps in Convolutional Neural Networks*. MachineLearningMastery.com. https://machinelearningmastery.com/how-to-visualize-filters-and-feature-maps-in-convolutional-neural-networks/ Accessed on January 28th, 2024.

[9]. Alsayed, Alhag & Li, Chunlin & Ahamed, & Hazim, Mohammed & Obied, Zainab. (2023). Arabic Handwritten Character Recognition Using Convolutional Neural Networks. 10.21203/rs.3.rs-3141935/v1 Accessed on January 23rd, 2024.

[10]. Alwagdani, M.S.; Jaha, E.S. Deep Learning-Based Child Handwritten Arabic Character Recognition and Handwriting Discrimination. Sensors 2023, 23, 6774. https://doi.org/10.3390/s23156774. Accessed on January 23rd, 2024.

[11]. Altwaijry, N., Al-Turaiki, I. Arabic handwriting recognition system using convolutional neural network. Neural Comput & Applic 33, 2249–2261 (2021). Accessed on January 24th, 2024.

[12]. Wang, W., Li, Y., Zou, T., Wang, X., You, J., & Luo, Y. (2020, January 6). *A novel image classification approach via dense-mobilenet models*. Mobile Information Systems. https://www.hindawi.com/journals/misy/2020/7602384/ Accessed on January 24th, 2024.

[13]. *K_04 understanding of mobilenet - en*. 위키독스. (n.d.). https://wikidocs.net/165429 Accessed on January 24th, 2024.

[14]. Kotbenek. (n.d.). *Kotbenek/hiragana-recognition: Handwritten hiragana recognition software*. GitHub. https://github.com/Kotbenek/Hiragana-recognition Accessed on January 24th, 2024.