Handwriting Recognition

The objective of this project was to create a Deep Learning network capable of classifying images of handwriting by their character or word contents. As there are multiple approaches to this computer vision task, it was up to my decision on whether to train the model on word-level labels or character-level labels. Based on the scope of dataset I was planning to use; it made more sense to do character-level recognition. A word-level recognition model would require a large dataset with enough unique classes to represent an entire dictionary or language full of words. My next important decision was to plan on which kind of network to use. I decided a Convolutional Neural Network (CNN) would be a good choice as image classification is a common task solved by similar models.

A good dataset was needed to properly train our model on human handwriting recognition. Extensive planning for the future would have helped me greatly in the project as I made the mistake of trying to take a dataset which contained images of entire names and find a way to train the model at a character-level. Although this issue was not impossible to resolve, I admit it was a mistake for me to take such a dataset and attempt to model it at the character-level. An approach to take such images and teach them at a character-level could possibly have been solved with the 'Sliding Window' method. The Sliding Window method basically slides a small window over the target image and examines the area covered by the window for learnable features. It is commonly used for images too large to extract features from at one time. It could have been used in this case to extract individual characters from our images, however this would still lead to more issues. How would I be able to correctly label every window segment? What would the model predict in cases where perhaps multiple characters are in view of the window? I did not have any answers to the issues, and so I decided to switch to a more traditional dataset consisting of individual characters in each image instead of trying to find a way to break down each name into smaller pieces. The original dataset I was working with was this one: https://www.kaggle.com/datasets/landlord/handwriting-recognition/.

I progressed the project with a different dataset consisting of 3410 total inputs and 62 unique labels. The labels consisted of the Latin alphabet (A-Z, a-z) and the numbers 0 through 9. There was an equal amount of data for each label (55 data entries per label). This dataset was much easier to work with and made my course of action very clear. I was able to refurbish much of my original model plotting methods to fit this new dataset instead of starting from scratch again, and after loading the data my task was mostly just writing the loops to convert the labels into numerical values and one-hot encoding them to give to the model. Previously, I just wrote a Python Dictionary which contained 29 unique labels as my previous dataset contained only A-Z and some punctuation. Since we were working with over double the labels now, I figured it would be best to write the loops to automate this task. The new dataset and the one the model is trained on can be found here: https://www.kaggle.com/datasets/dhruvildave/english-handwritten-characters-dataset.

The project was written on a Python Notebook within the Jupyter environment. The entire project uses multiple imported libraries and functions to assist in the data preprocessing, model creation, training, and statistic plotting of our model:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as img
import tensorflow as tf
from tensorflow import nn
from tensorflow.keras import regularizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.layers import Activation, Conv2D, Dense, Dropout, Flatten, MaxPooling2D, Reshape
import cv2
import random
from sklearn import metrics
import seaborn as sns
```

After finishing the imports, we begin loading and preprocessing the data. Preprocessing included creating our RGB array from the images, resizing images to 128x128, scaling the RGB arrays to contain values between 0 and 1, generating a numerical Python Dictionary based on image labels, and then one-hot encoding our data. After data preprocessing, the model was created. The model consists of three convolutional layers consisting of 32 filters of size 3x3 and taking strides of 1. The data is then flattened and put through a Dense layer of 128 units before being put through our final output layer. The activation function used in every layer except the output was ReLU activation. Our output layer used Softmax activation. L1L2 regularization, Dropout, and Learning Rate adjustment were used to help improve model generalization.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 32)      320

 max_pooling2d (MaxPooling2  (None, 63, 63, 32)        0
 D)

 dropout (Dropout)           (None, 63, 63, 32)        0

 conv2d_1 (Conv2D)           (None, 61, 61, 32)        9248

 activation (Activation)     (None, 61, 61, 32)        0

 max_pooling2d_1 (MaxPoolin  (None, 30, 30, 32)        0
 g2D)

 dropout_1 (Dropout)         (None, 30, 30, 32)        0

 conv2d_2 (Conv2D)           (None, 28, 28, 32)        9248

 max_pooling2d_2 (MaxPoolin  (None, 14, 14, 32)        0
 g2D)

 dropout_2 (Dropout)         (None, 14, 14, 32)        0

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 128)               802944

 dropout_3 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 62)                7998

=================================================================
Total params: 829758 (3.17 MB)
Trainable params: 829758 (3.17 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

The model was able to learn the training set very well as it would sometimes hit 98% training accuracy, however even with all the regularization techniques, only a validation accuracy of 67% could be achieved as a maximum. The means the model was quite clearly overfitting. Some further actions were taken to try to improve this number such as increasing the size of the validation set and simplifying the model, however it would seem the issue stemmed from the size of the dataset. 3410 examples split between 62 labels is not a large amount of data to share between both the training and validation sets. The model may perform better given a larger dataset. One solution that could have been implemented would be to augment our existing data to generate more examples. Some techniques of data augmentation are rotation, flipping, filtering, cropping, etc. By applying these methods, we could increase the amount of data in our dataset while also giving the model new images to be trained on.

In conclusion, the model was able to learn from the training set, however its ability to generalize is suspect and would likely be improved by increasing the amount of data in our dataset. If I were to add anything more to the project, it would definitely be data augmentation to attempt to solve this issue.