**Assignment 1**

ITRI 626 - 30010993

Marnus Steyn

2023

**Purpose of Document**

**Part 0: Technology stack discussion and reasoning:**

In the process of determining the technology to be used for this assignment, the decision was made to simply repurpose the functional neural network made for the exam assignment in July as this would save a lot of development time as opposed to starting from scratch. As such this document will be very similar to the one submitted with the aforementioned exam assignment, but will differ in a few areas.

Following from that, this assignment was done using the same stack, namely Node, Javascript and Tensorflow JS. This stack was once again chosen due to my own familiarity with Javascript and Node, and Tensorflow JS as it is simply a Javascript based version of Tensorflow, one of the most popular and widely used machine learning libraries out there.

**Part 1: Loading and preprocessing the data:**

To start the process, the data needs to be loaded. I decided to use the SVHN dataset, and as such I faced an issue. The file is 3gb even when zipped, and goes up to around 4.5gb unzipped. This would pose a problem for any sort of shipping or distribution of the program, as the size of the dataset far exceeds the size of the actual program. After having some difficulty related to Node with loading the dataset at runtime, I decided to use the dataset locally. It has been extracted and placed in the project folder submitted with this report.

After loading the dataset, the preprocessing is taken care of by Tensorflow functions, as shown in the method below:

A screen shot of a computer program

Description automatically generated

The data is normalized to pixel values between 0 and 1, and then returned.

**Part 2: Defining the Neural Network:**

In order to actually define the neural network, I asked for some suggestion from ChatGPT, and came up with the following architecture:

A black background with text

Description automatically generated

As can be seen, the model is not too complex. It consists of 6 layers:

1. Input layer: this expects an image with a shape of 32x32 pixels and with 3 channels for colour (the standard RGB)
2. A convolutional layer to produce feature maps the depend on the input and the parameters, uses the ReLu activation function
3. A max pooling layer using 2 strides, halving the input maps’ sizes
4. A flatten layer that transforms the inputs into a one-dimensional vector form a 2d map
5. A dense layer using ReLu activation with 128 units
6. The output dense layer, using softmax and 10 units

The final output of this model is expected to be a vector representing the predicted probabilities of each class.

**Part 5: Train and Validate model:**

The training function for this model is as below:



Asking ChatGPT how this would function, the following is given:

Prompt:

async function trainSVHNModel() {

const { trainImages, trainLabels, testImages, testLabels } = await loadAndPreprocessSVHN();

// Create and compile the SVHN model

const model = createSVHNModel();

model.compile({

optimizer: 'adam',

loss: 'categoricalCrossentropy',

metrics: ['accuracy'],

});

// Train the model

await model.fit(trainImages, trainLabels, {

epochs: 10,

batchSize: 32,

validationData: [testImages, testLabels],

callbacks: {

onEpochEnd: (epoch, logs) => {

console.log(

`Epoch ${epoch + 1} - loss: ${logs.loss.toFixed(4)}, accuracy: ${logs.acc.toFixed(4)}`

);

},

},

});

console.log('Training complete!');

}

Can you explain how this function would execute in the context of a convolutional neural network using the SVHN dataset?

Answer:

Certainly! The `trainSVHNModel` function is responsible for training a convolutional neural network (CNN) using the SVHN (Street View House Numbers) dataset. Let's break down how this function works:

1. `const { trainImages, trainLabels, testImages, testLabels } = await loadAndPreprocessSVHN();`

- This line asynchronously loads and preprocesses the SVHN dataset. It assumes that there is a function called `loadAndPreprocessSVHN` that returns an object containing training and testing images along with their corresponding labels. The data is loaded and preprocessed so that it can be used for training and evaluation.

2. `const model = createSVHNModel();`

- This line creates an instance of the SVHN model using the `createSVHNModel` function you provided earlier. This model defines the architecture and layers of the neural network.

3. Model Compilation:

- `model.compile({ optimizer: 'adam', loss: 'categoricalCrossentropy', metrics: ['accuracy'] });`

- Here, the model is compiled with the following settings:

- Optimizer: 'adam' - Adam optimizer is a commonly used optimization algorithm.

- Loss: 'categoricalCrossentropy' - This is the loss function used for categorical classification problems. It measures the difference between the predicted class probabilities and the actual labels.

- Metrics: ['accuracy'] - The model will track and display accuracy during training as one of the evaluation metrics.

4. Model Training:

- `await model.fit(trainImages, trainLabels, { epochs: 10, batchSize: 32, validationData: [testImages, testLabels], callbacks: { onEpochEnd: (epoch, logs) => { console.log(`Epoch ${epoch + 1} - loss: ${logs.loss.toFixed(4)}, accuracy: ${logs.acc.toFixed(4)}`); }, }, });`

- This code trains the model using the training data and labels. Here's what each parameter does:

- `trainImages` and `trainLabels` are the training data and labels, respectively.

- `epochs: 10` - The training process will run for 10 epochs, meaning the entire training dataset will be processed 10 times.

- `batchSize: 32` - Training is performed in batches of 32 samples at a time. This helps with memory efficiency and can improve convergence.

- `validationData: [testImages, testLabels]` - During training, the model's performance will be evaluated on the test dataset to monitor its progress and prevent overfitting.

- `callbacks` allows you to define a callback function that is executed at the end of each training epoch. In this case, it logs the loss and accuracy for each epoch.

5. `console.log('Training complete!');`

- After training is complete, this line prints a message indicating that the training has finished.

Overall, the `trainSVHNModel` function loads the dataset, creates and compiles the CNN model, trains the model on the training data, and monitors its performance during training. The function provides insights into the model's accuracy and loss for each epoch, making it easier to assess its training progress.

**Part 6: Developer Notes:**

Attached in Submission (Must install Nodejs to use npm install http-server. Run http-server and follow the ip addresses given to view the app).

**Part 7: Extras:**

Image generated using the prompt: an ai learning from a dataset using a neural network



As an avid data science student, I have enjoyed learning about the inner workings of AI and related technologies. The way that simple principles can be used together to create complex behaviors and structures is always fascinating.