Implementation

An appendix to the Mathematical Exploration of Janav Nagapatla and Aiden Lim
Contains: main.py, model.py, layer.py, helpers.py, image.py

This code is made publicly available at: https://github.com/jnagapatla-x-aidenlims/implementation

Part 2 of a series of 2 programs

Take note that these programs require NumPy AND Pillow to run

main.py (1 of 2)

```python
# Import Python libraries
from PIL import Image

# Import structures from other files
from model import Model
from image import greyscale, black, resample, arrayed

# Print manifest
print("\033c", end="", flush=True)
print("Year 4 Mathematical Exploration 2024: Implementation", end="\033[K\n\a")
print("Janav Nagapatla and Aiden Lim", end="\033[K\n")
print("All rights reserved", end="\033[K\n")

# Import model
print("", end="\033[K\n")
print("Importing model", end="\033[K\n")

model: Model = Model(input("> Path to your model file (.networkconfig): \a\033[K"))

print(f"    > Model name: {model.name}", end="\033[K\n")
print(f"    > Model program: {model.program}", end="\033[K\n")
print(f"    > Model authors: {model.authors}", end="\033[K\n")
print(f"    > Model layers: {len(model.layers)}", end="\033[K\n")

print("> Successfully imported model", end="\033[K\n")

# Process image
print("", end="\033[K\n")
print("Processing image", end="\033[K\n")

image: Image = (
    resample(
        greyscale(
            black(
                greyscale(
                    Image.open(input("> Path to your image file (any type/resolution): \a\033[K")))))))
```

```python
if input("> Would you like to view the image in Preview.app or MS Paint (yes/no): \a\033[K").lower()[0] == "y":
    image.show()

print("> Successfully imported image file", end="\033[K\n")

# Predicting result
print("", end="\033[K\n")
print("Predicting value of image", end="\033[K\n")

prediction, confidence = model.predict(arrayed(image))

print(f"> {model.name} predicts that the image is a {prediction} with {confidence:.2%} confidence", end="\033[K\n")
```

model.py (1 of 2)

```python
# Import Python libraries
import numpy as np

# Import structures from other files
from layer import Layer
from helpers import sigmoid, tanh, softmax


class Model:
    """
    Using a model file from a trainer, one can predict images using this class
    """

    def __init__(self,
                 model_path: str) -> None:
        """
        Imports the weights of a trained model into the program
        """

        self.layers: list[Layer] = []

        with open(model_path, "r") as model_file:
            self.name: str = model_file.readline()[7:-1]
            self.program: str = model_file.readline()[9:-1]
            self.authors: str = model_file.readline()[9:-1]
            model_file.readline()

            while True:
                match model_file.readline():
                    case "> New Layer\n":
                        input_size: int = int(model_file.readline()[22:-1])
                        output_size: int = int(model_file.readline()[23:-1])
                        activation: str = model_file.readline()[27:-1]
                        model_file.readline()
                        weights: np.ndarray = np.array([[float(model_file.readline()[10:-1])
                                                         for _ in range(input_size)]
                                                        for _ in range(output_size)])
                        model_file.readline()
```

model.py (2 of 2)

```python
                        biases: np.ndarray = np.array([[float(model_file.readline()[10:-1])]
                                                       for _ in range(output_size)])

                        match activation:
                            case "sigmoid":
                                self.layers.append(Layer(weights, biases, sigmoid))
                            case "tanh":
                                self.layers.append(Layer(weights, biases, tanh))
                    case "--- End Network Configuration ---":
                        break

    def predict(self,
                image: np.ndarray) -> tuple[int, float]:
        """
        Feeds the image array through the program
        """

        evaluation: np.ndarray = image

        for layer in self.layers[0:-1]:
            evaluation = layer.forward(evaluation)

        probability = softmax(self.layers[-1].nonactivated(evaluation))

        evaluation = self.layers[-1].forward(evaluation)

        return int(np.argmax(evaluation)), probability


if __name__ == "__main__":
    exit("This script cannot be run on its own.\033[K")
```

```python
# Import Python libraries
import numpy as np
from typing import Callable


class Layer:
    """
    A representation of a connected layer
    Requires the number of input and output neurones of the layer and its desired activation function + derivative
    """

    def __init__(self,
                 weights: np.ndarray,
                 biases: np.ndarray,
                 activation: Callable[[np.ndarray], np.ndarray]) -> None:
        """
        Generates random weights and biases (as a starting point) or the layer
        """

        self.weights: np.ndarray = weights
        self.biases: np.ndarray = biases

        self.activation: Callable[[np.ndarray], np.ndarray] = activation

    def forward(self,
                previous: np.ndarray) -> np.ndarray:
        """
        Conducts forward propagation and returns the output neurones
        """

        return self.activation(np.dot(self.weights, previous) + self.biases)

    def nonactivated(self,
                     previous: np.ndarray) -> np.ndarray:
        """
        Conducts forward propagation without activation and returns the output neurones
        """
        return np.dot(self.weights, previous) + self.biases


if __name__ == "__main__":
    exit("This script cannot be run on its own.\033[K")
```

helpers.py (1 of 1)

```python
# Import Python libraries
import numpy as np


def sigmoid(value: np.ndarray) -> np.ndarray:
    """
    Returns logistic sigmoid at value
    """

    return 1.0 / (1.0 + np.exp(-value))


def tanh(value: np.ndarray) -> np.ndarray:
    """
    Returns tanh at value
    """

    return np.tanh(value)


def softmax(value: np.ndarray) -> float:
    """
    Returns softmax at value
    """

    return np.exp(value.max()) / np.sum(np.exp(value))


if __name__ == "__main__":
    exit("This script cannot be run on its own.\033[K")
```

image.py (1 of 2)

```python
# Import Python libraries
import numpy as np
from PIL import Image


def threshold(x: int) -> int:
    """
    Returns 255 if x is more than or equal to the cutoff (else 0)
    """

    return 255 if x >= 75 else 0


def greyscale(image: Image) -> Image:
    """
    Makes the image greyscale from coloured
    """

    return image.convert('L')


def black(image: Image) -> Image:
    """
    Makes the image black and white from coloured
    """

    return image.point(threshold, mode='1')


def resample(image: Image) -> Image:
    """
    Resizes and resamples via LANCZOS the image
    """

    result = Image.new("L", (28, 28), 255)
    result.paste(image.resize((20, 20), Image.Resampling.BICUBIC), (4, 4))
    return result
```

```python
def arrayed(image: Image) -> np.ndarray:
    """
    Converts the image into input neurones
    """

    return 1 - np.array(image).reshape((784, 1)) / 255


if __name__ == "__main__":
    exit("This script cannot be run on its own.\033[K")
```

To God Be The Glory
The Best Is Yet To Be