Project 2: Neural Networks

# Introduction

Neural networks are a powerful tool for data scientists to classify complex relationships within data. In this report, I will explore binary, multi-class classification, and regression using three different datasets: a heart disease dataset from IEEE Dataport, an iris dataset from sklearn, and a music dataset from Kaggle. The first dataset aims to predict whether or not someone has had a stroke based on diagnostic measurements, the second dataset seeks to identify the type of plant based on physical attributes, and the third dataset aims to predict the song's popularity level based on its music metrics.

In the Background section of this report, I will provide a clear definition of neural networks and explain how they are implemented in Keras. The Methods section will detail the steps I took to get the results. Finally, in the Results section, I will present the results of the experiments, including how changing the model’s parameters affected the classification accuracy.

# Background

**Neural Networks**

Neural Networks are a type of deep learning algorithm. They are widely used to identify relationships between complex and nonlinear datasets. Neural networks are capable of training themselves to recognize patterns within data and then predict outputs for new sets of data.

A basic neural network consists of an input layer, hidden layers, and an output layer. The input layer receives the input data, the hidden layers perform most of the computation in the network, and the output layer produces the final prediction. These layers are made up of nodes that are connected to each other to form an interconnected network.

Each layer serves a specific function and acts as a filter for the data being passed to it. The nodes within the layers read the data from the previous layer and determine the strongest relationships. Once this is determined, they pass the information to the nodes connected to the next layer.

The connections between the nodes of different layers hold weights, which are multiplied by their given inputs, and the sum is sent as inputs to the nodes in the hidden layer. Each node within the hidden layers has a bias value, which is added to the input sum and passed to an activation function. This function decides whether a neuron should be activated or not. The result is transmitted to nodes in the next layer.

When the output layer is reached, the node with the highest value determines the prediction output. This output is cross-referenced with the actual output to calculate the magnitude of error. The magnitude of error determines how inaccurate the model is. This information is then sent back through the network, and the weights are adjusted to minimize the error until the model performs back-propagation.

Backpropagation is a process where the model learns from its mistakes and adjusts its weights to make correct predictions. As the complexity of the model increases, so does the number of layers within the network, making it more effective in identifying complex relationships within datasets.

**Binary Classification**

Binary classification in neural networks predicts one of two outcomes based on input data. The network is trained on labeled data, and during training, it minimizes the difference between its predicted output and the true label using binary cross-entropy.

**Multiclass Classification**

Multiclass classification in neural networks categorizes inputs into several possible categories. The network is trained on labeled data and learns to map the input to the corresponding category using activation functions and other parameters. During training, the network minimizes the difference between its predicted output and the true label using categorical cross-entropy.

**Regression Analysis**

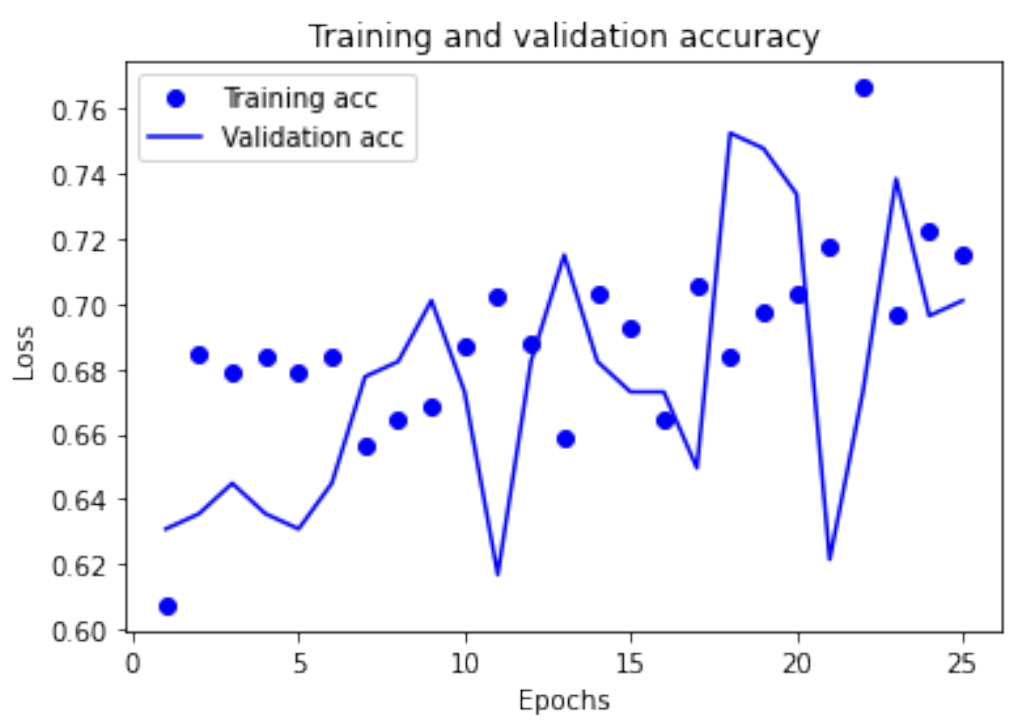
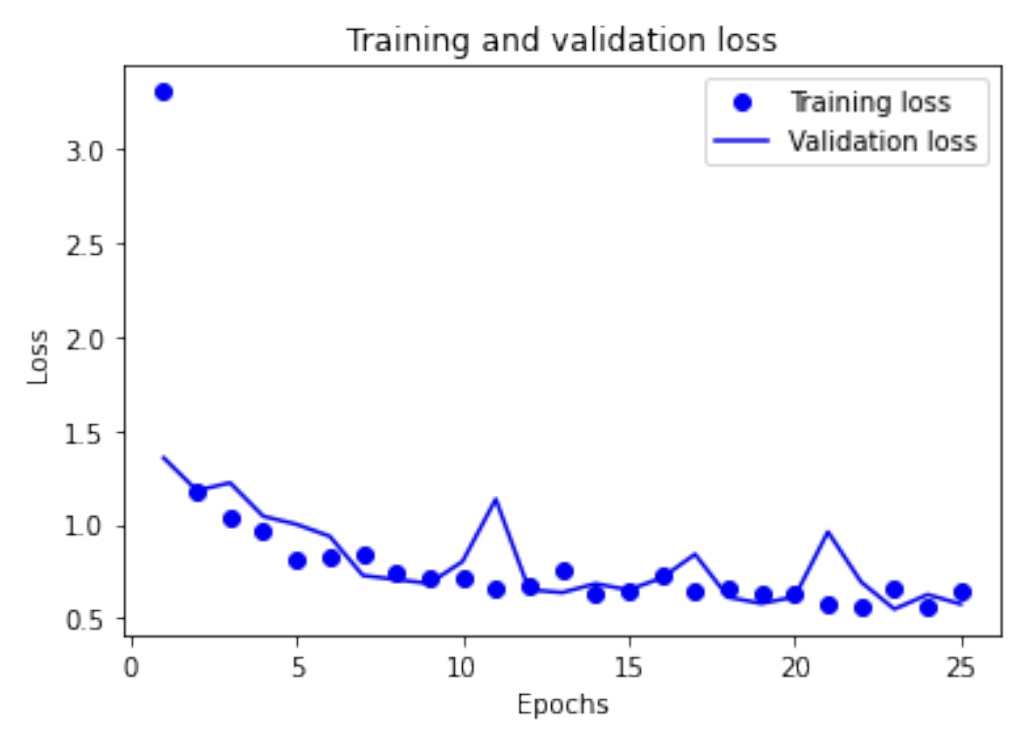
Regression in neural networks predicts a continuous numerical value or set of values based on an input. The network is trained on a labeled dataset, where each input is associated with a numerical value indicating the desired output. During training, the network tries to minimize the difference between its predicted output and the true label using a loss function.

# Method

Keras is a powerful Python library that enables data scientists to develop and evaluate deep learning models. It allows users to define and train neural network models, making it an essential tool in the development of machine learning algorithms. Keras has predefined layers that are commonly used and can be extended, making it easy to prototype and learn neural network development.

**Binary Classification**

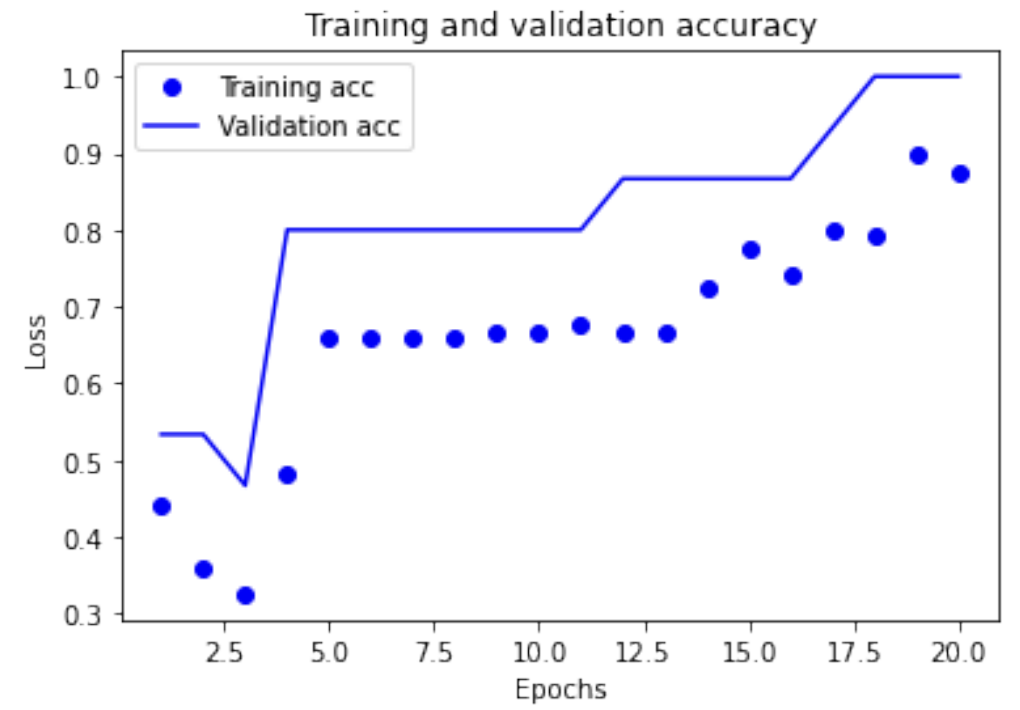
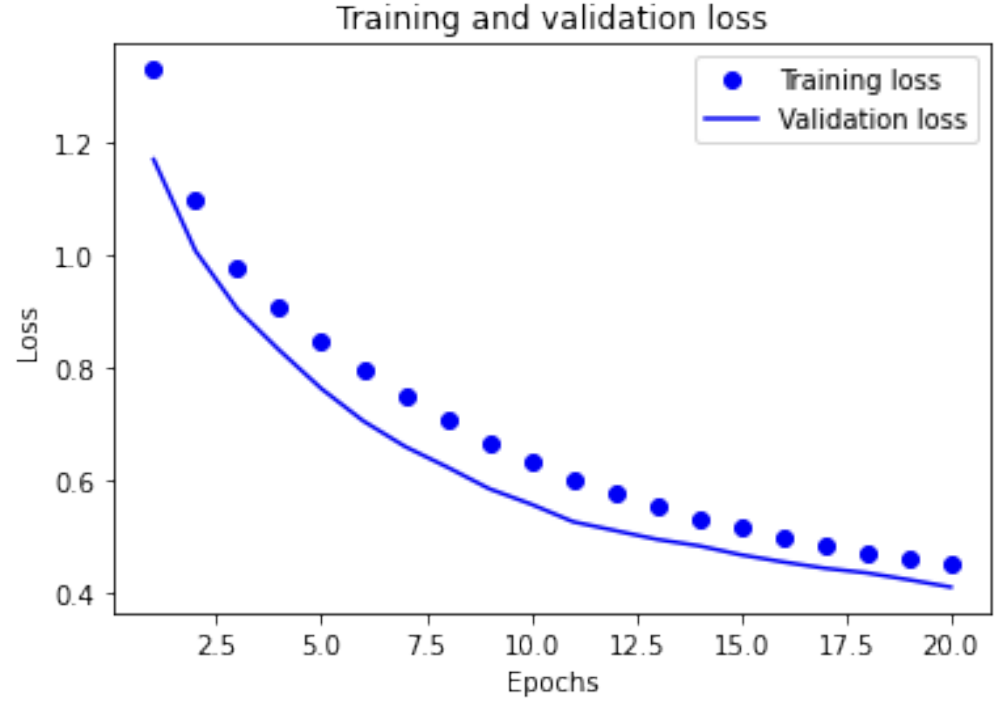
In this report, I implemented three models using Keras. The first model was a binary classification model used on the heart stroke dataset from Kaggle, which aims to predict whether a patient has had a stroke based on diagnostic measurements. The first step was to import the necessary libraries and data, then split the data into input and output variables. I performed a Train-Test split that included a set for validation. Next, I defined the Keras model and added layers specifying different input units, input dimensions, and activation functions. I compiled the model with a loss function, an optimizer, and an accuracy metric. The model was then fit to the training data and evaluated for accuracy on the testing data. I evaluated the model's accuracy for the testing and validation data sets. I then used Matplotlib to plot the training and validation accuracy.



As illustrated in the above images, the loss of the training data decreased with each epoch, while the accuracy of the training data increased accordingly. This trend is consistent with the expected behavior of the gradient descent optimization algorithm, which aims to minimize the objective function with each iteration. Notably, the model achieved the highest accuracy for both training and validation data at the 21st epoch. After 21 epochs the model begins to overfit.

**Multiclass Classification**

The second model was a multi-class classification model used on the iris dataset, which attempts to determine which of 3 species of plants the plant is based of physical features. I performed the same procedure as I did for the binary classification model. I broke up the data between feature variables and the target variable and performed one-hot encoding to change the categorical class of the species of plants. I split the data into train, testing, and validation data, then compiled, trained, and evaluated the model. I evaluated the model's performance using the accuracy of the model with the test and validation data sets. I then displayed its loss and accuracy curves



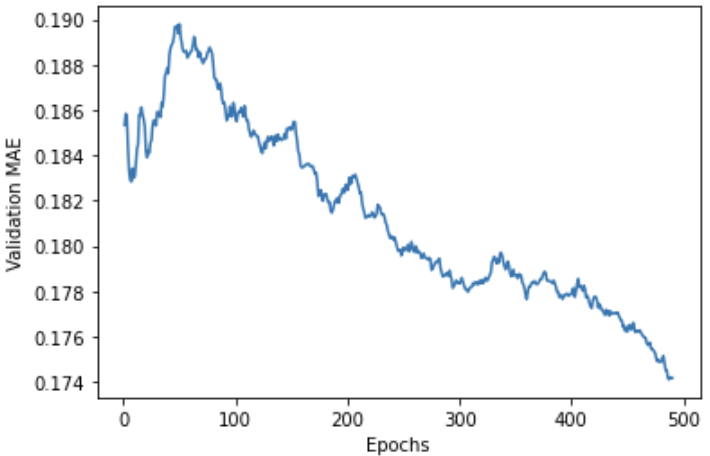
As depicted in the graph above, the model starts to exhibit signs of overfitting after 19 epochs.

**Regression Analysis**

The third model was a regression model used on the song dataset, which attempts to predict the popularity level of a song based on music metrics. I first define the neural network model using the Keras library. the model has three layers, with two layers of 64 neurons with a 'relu' activation function and an output layer. The model is compiled with a 'rmsprop' optimizer, a 'mse' (mean squared error) loss function, and evaluating with 'mae' (mean absolute error).

I then set up a k-fold cross-validation. Within each fold, the data is divided into training and validation sets, and the model is built and trained on the training set. The performance of the model is then evaluated on the validation set, and the mean absolute error is saved for each fold.

Next, I plot the mean absolute error against the number of epochs using the matplotlib library. I also added a smoothing function to make the plot easier to read.



As indicated by the image above, the validation mean absolute error continues to improve considerably until roughly 500 epochs.

# Results

Neural networks are widely used in the field of data science. In this project, I have demonstrated their effectiveness in binary, multi-class classification, and regression tasks using the Keras library. In the experiments, I tested various model parameters such as the number of hidden layers, number of hidden units, loss function, and the activation function to see how they affected the model's accuracy.

**Binary Classification**

In the binary classification dataset, the initial model consisted of 2 hidden layers with 16 nodes, yielding an accuracy score of 78%. To optimize the model's performance, I experimented with varying the number of hidden layers and nodes. The results revealed that increasing or decreasing the hidden layers resulted in a lower accuracy score.

Further experimentation with the number of nodes per layer indicated that increasing the number of layers did not result in a higher accuracy score.

In addition, I investigated the effect of changing the loss function from binary\_crossentropy to mean squared error (MSE) and discovered that MSE was more effective for this classification, as the model's accuracy increased to 79% when using the MSE loss function.

Finally, I explored the impact of changing the activation function from relu to tanh, and found that the relu model outperformed the tanh model by 3% in accuracy.

**Multiclass Classification**

While working with the Iris dataset, the initial model, comprised of 2 hidden layers with 16 nodes per layer, achieved an accuracy score of 80%. I then tested how changing the parameters would affect the model.

By increasing the number of nodes, I discovered that the model's accuracy increased. Specifically, when the number of nodes was increased to 32, the accuracy score surged to 93%. However, when I further increased the number of nodes, the model's accuracy plateaued, and there was no significant improvement.

Next, I experimented with varying the number of hidden layers. I found that adding more layers had an adverse effect on the model's accuracy, reducing its accuracy to 60%. Removing an additional layer had little effect on the model's accuracy.

**Regression**

For the regression test, I performed k-fold cross-validation to evaluate a neural network model's performance on the regression problem. The code trained the model for 500 epochs during each fold and stores the validation mean absolute error (MAE) for each fold in a list. The training data is split into a validation set and a partial training set for each fold, and the model is trained on the partial training data. The validation MAE history is then extracted and appended to a list. Finally, the mean of all scores obtained from the k-fold cross-validation is computed using the np.mean() function. The results of the regression test displayed that the model’s output of song popularity is off by 17 on average.

# Conclusion

In conclusion, neural networks are a powerful tool for data scientists, and with careful parameter tuning, they can be used to classify complex datasets accurately. However, finding the best model may require extensive trial and error, making it time-consuming and computationally expensive.