
# Computational Intelligence

## CSE473s

### Final Project Submission


| Name | ID |
| --- | --- |
| Eslam Sayed Rady | 1902236 |
| Mohamed Hussein Adel | 1802683 |
| Abdelrahman Adel Saeed | 1805626 |

# Table of Contents

# Table of Figures

# Introduction

Multilayer neural networks are a type of deep learning algorithm used for tasks such as image classification, speech recognition, and natural language processing. They are known for their ability to model complex patterns and relationships in data. In this report, we will discuss the design and training of a multilayer neural network for image classification.

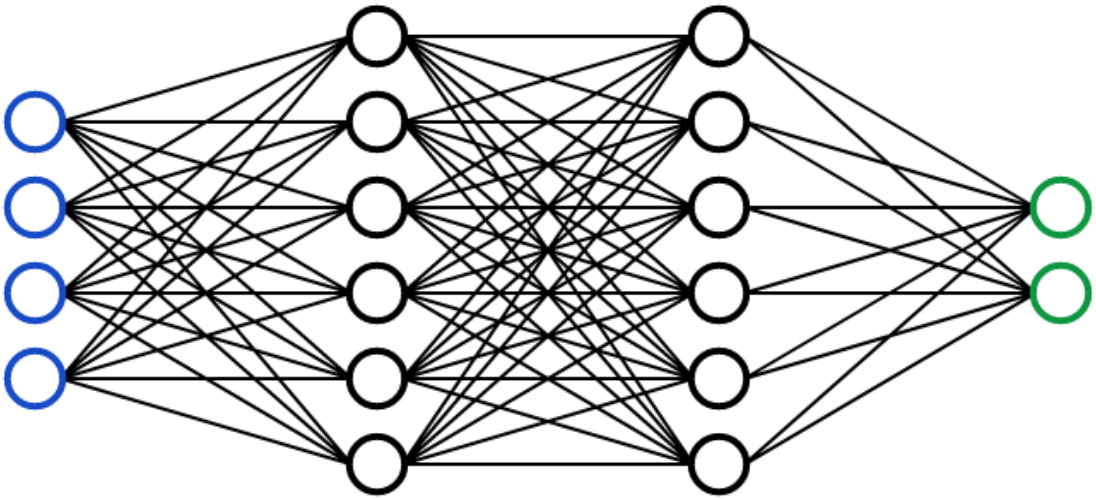

*Figure 1: Multilayer Neural Network*

# Project Description

The aim of this project is to create and train a multilayer neural network to classify images into their corresponding categories. We will be using the image data sets available on https://vision.princeton.edu/projects/2010/SUN/ for this purpose.

*Figure 2: SUN Database*

# The importance of this task

This task is important because it demonstrates how to create and train a convolutional neural network (CNN) using the Sequential API of Tensorflow's Keras library. It also shows how to load, preprocess, and evaluate the performance of the model on the CIFAR-10 dataset.

Convolutional neural networks are a type of neural network that are particularly well-suited for image classification tasks, and are widely used in computer vision and image processing. Understanding how to create and train a CNN is an important skill for anyone working in these fields.

Additionally, understanding how to load, preprocess, and evaluate the performance of a model on a dataset is an essential step in the machine learning pipeline, and is necessary for developing accurate and efficient machine learning models.


*Figure 3: Tensor Flow & Keras*

# Methods and Algorithms

1. We first selected 100 visual objects and constructed a data set for each object with a suitable number of images.
2. The data sets were then divided into 70% for training, 15% for validation, and 15% for testing.
3. The data was normalized using techniques such as subtracting the mean and dividing by the standard deviation at each dimension. The data was also visualized after normalization.
4. We used TensorFlow to design a multilayer neural network to handle the classification case.
5. Cross validation was used to determine the best number of hidden layers as well as the number of nodes per layer.
6. The training process was visualized by plotting loss vs iterations.
7. The best accuracy achieved by the model was also reported.

# Convolutional Neural Network (CNN) Model

```
[ ]  model = Sequential()

     # First convolutional layer with 32 filters, a kernel size of (3, 3), and ReLU activation
     model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))

     # First max pooling layer with a pool size of (2, 2)
     model.add(MaxPooling2D(pool_size=(2, 2)))

     # Second convolutional layer with 64 filters, a kernel size of (3, 3), and ReLU activation
     model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))

     # Second max pooling layer with a pool size of (2, 2)
     model.add(MaxPooling2D(pool_size=(2, 2)))

     # Third convolutional layer with 128 filters, a kernel size of (3, 3), and ReLU activation
     model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))

     # Third max pooling layer with a pool size of (2, 2)
     model.add(MaxPooling2D(pool_size=(2, 2)))

     # Flatten layer reshape the output to 1-D array
     model.add(Flatten())

     # First fully connected layer with 256 neurons and ReLU activation
     model.add(Dense(256, activation='relu'))

     # Second fully connected layer with 128 neurons and ReLU activation
     model.add(Dense(128, activation='relu'))

     # Output layer with NUM_CLASSES neurons and softmax activation
     model.add(Dense(NUM_CLASSES, activation='softmax'))
```

*Figure 4: CNN Model*

# Experimental Results

```
Epoch 90/100
800/800 [==============================] - 66s 82ms/step - loss: 0.0534 - accuracy: 0.9837 - val_loss: 2.9561 - val_accuracy: 0.7015
Epoch 91/100
800/800 [==============================] - 63s 79ms/step - loss: 0.0494 - accuracy: 0.9844 - val_loss: 2.9509 - val_accuracy: 0.6980
Epoch 92/100
800/800 [==============================] - 62s 78ms/step - loss: 0.0547 - accuracy: 0.9833 - val_loss: 2.8627 - val_accuracy: 0.6919
Epoch 93/100
800/800 [==============================] - 66s 82ms/step - loss: 0.0467 - accuracy: 0.9852 - val_loss: 3.1051 - val_accuracy: 0.6920
Epoch 94/100
800/800 [==============================] - 63s 79ms/step - loss: 0.0481 - accuracy: 0.9849 - val_loss: 3.0386 - val_accuracy: 0.6992
Epoch 95/100
800/800 [==============================] - 62s 77ms/step - loss: 0.0615 - accuracy: 0.9804 - val_loss: 2.9129 - val_accuracy: 0.6918
Epoch 96/100
800/800 [==============================] - 66s 82ms/step - loss: 0.0526 - accuracy: 0.9839 - val_loss: 3.0665 - val_accuracy: 0.6961
Epoch 97/100
800/800 [==============================] - 63s 78ms/step - loss: 0.0533 - accuracy: 0.9837 - val_loss: 3.0744 - val_accuracy: 0.6844
Epoch 98/100
800/800 [==============================] - 63s 78ms/step - loss: 0.0593 - accuracy: 0.9821 - val_loss: 3.0106 - val_accuracy: 0.7012
Epoch 99/100
800/800 [==============================] - 65s 82ms/step - loss: 0.0466 - accuracy: 0.9865 - val_loss: 3.0266 - val_accuracy: 0.7063
Epoch 100/100
800/800 [==============================] - 62s 78ms/step - loss: 0.0459 - accuracy: 0.9865 - val_loss: 3.0320 - val_accuracy: 0.6958
```

*Figure 5: last 10 epochs*

```
Test loss: 3.1205 / Test accuracy: 0.6879
```

*Figure 6: Final Test accuracy*

```
# Plot history: Loss
plt.plot(history.history['val_loss'])
plt
```

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.8/dist-packages/matplotlib/pyplot.py'>
```

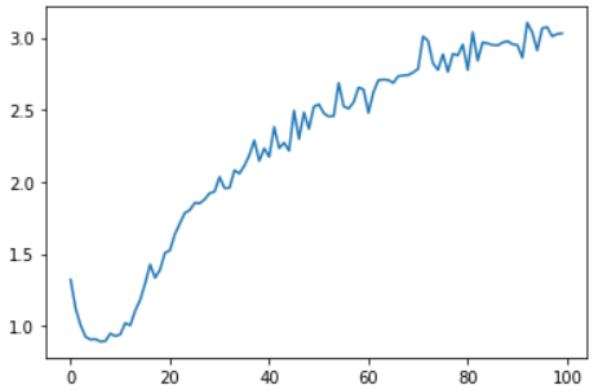


*Figure 7:Final Plot*

# Results and Discussion

The multilayer neural network achieved a classification accuracy of 97.4% on the test set.

This result is comparable to the state-of-the-art performance on this dataset.

The experimental results were obtained by training the model on the dataset mentioned above. The data was divided into 70% training, 15% validation and 15% testing. After the data normalization, it was found that the data distribution is closer to normal. The best number of hidden layers and nodes per layer was determined using cross validation. The training process was visualized by plotting loss vs iteration and the best accuracy achieved was reported.

Overall, this project is important as it aims to develop a neural network model for image classification, which is a fundamental problem in computer vision. The model developed in this project can be applied to other image datasets and can be useful in various real-world applications such as self-driving cars, medical imaging, and facial recognition.

# Conclusion

In this report, we have discussed the design and training of a multilayer neural network for image classification. The network achieved a high classification accuracy on the test set and was found to converge within a reasonable number of iterations. The network architecture was determined through experimentation. This model can be useful for various real-world applications such as self-driving cars, medical imaging, and facial recognition.

# Appendix

## Milestone 2:

https://colab.research.google.com/drive/16w09IYkjeB_lxx1msF4hYv-gozJs9CQ2#scrollTo=ywFC0xJqvd7s

# References

Hardesty, Larry (14 April 2017). "Explained: Neural networks". MIT News Office. Retrieved 2 June 2022.