Neural Network Models for Object Recognition

Group 3

University of Essex | Online

OVERVIEW

Artificial Neural Networks (ANN) imitate the neuron system, connecting nodes of the neuron, processed by the activation function, to make a final decision based on the calculation between weights multiplied by input value plus the bias (Géron, 2019) **.**

INTRODUCTION

Background

- How machine learning can identify the object.
- Artificial Neural Networks (ANN) is the tool which imitate human brain for the machine to learn.
- Neural networks divided into three main parts, input, multiple hidden layer, and output (Dertat, 2017).

Purpose

- Explain on what is the ANN
- Experiment ANN with CIFAR-10 data (Krizhevsky, 2017)
- Evaluate the method

Methodology

- Use keras library to process the object recognition
- Load CIFAR-10 Data
- Divide data as train and test data
- Train data using ANN method
- Check accuracy

NEURONS

University of Essex | Online

How does it work?

- Input signals received by dendrites
- The input processed and transform the information inside the body cells
- The input transferred thru axon and send the output into the synapsis
- Output signal send thru axon terminals

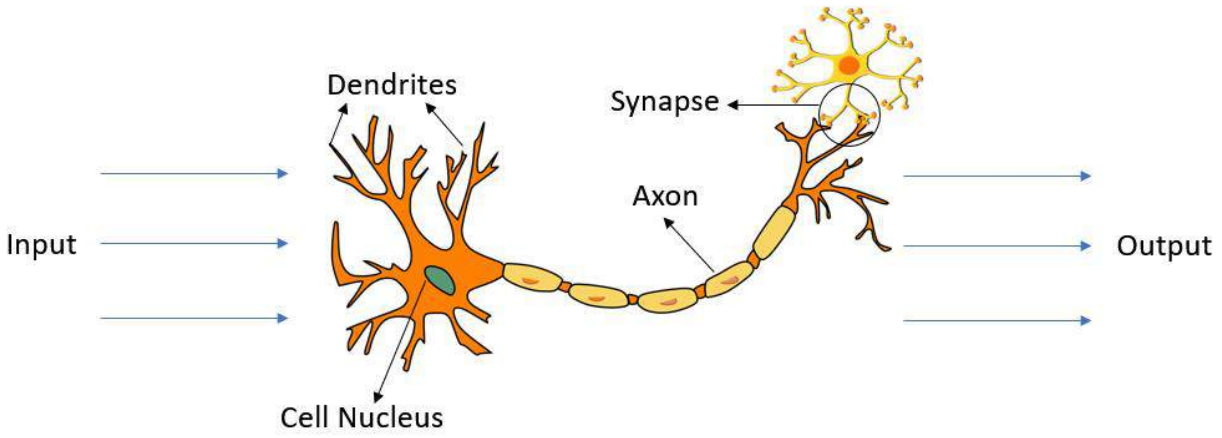


Image from pixabay.com

ARTIFICIAL NEURAL NETWORKS

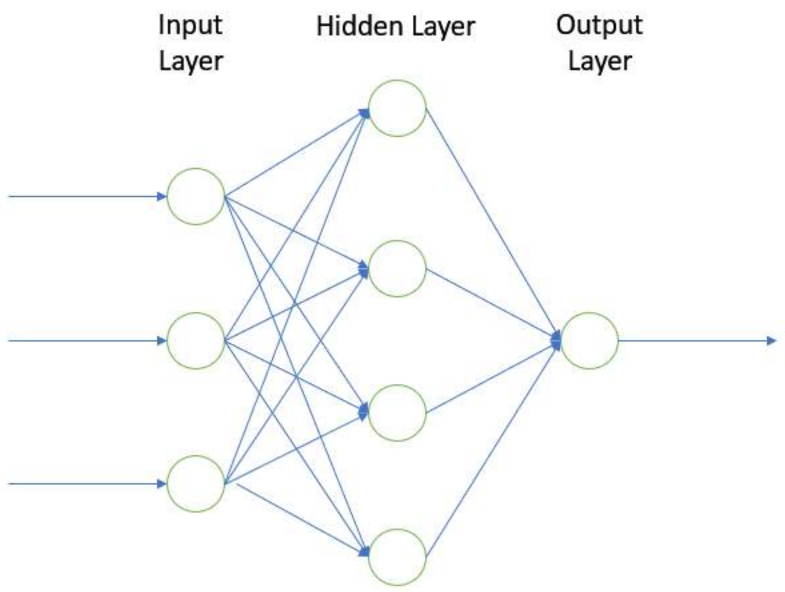
University of Essex | Online

How does ANN work?

- Input signals received in the nodes
- Hidden layer processed the input signal
- Activation function

  $f(x_i, w_i) = \sum_{k=0}^{i} (x_i * w_i) + b_i$

- x is the input, w is the weight, while b is the bias



ARTIFICIAL NEURAL NETWORKS

University of Essex Online

Linear Activation Function

- Activation function in the single node calculated similar like the linear equation represented in the below equation

  $y = \mathrm{f}(x, w) = (x * w) + b$

- x is the input, w is the weight, while b is the bias, and y is the output

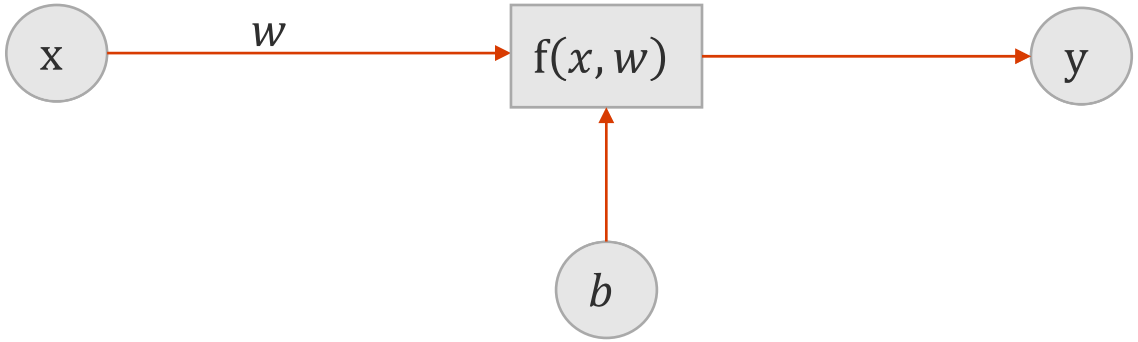


Fig. Linear Activation Function (Nadeem, 2022)

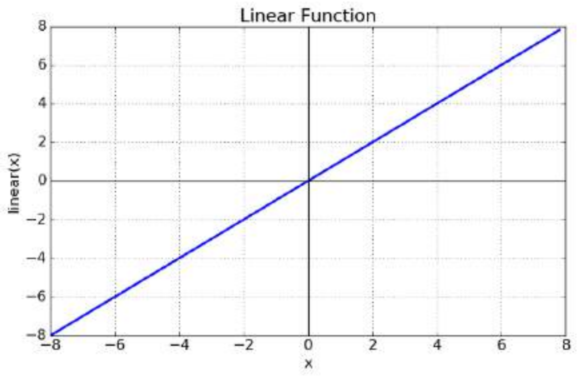


Fig. Linear Function (Image Source: Sharma, 2022)

ACTIVATION FUNCTION

University of Essex | Online

What is activation function

- The main functionality of activation function is to processed input signal into output

Types of Activation

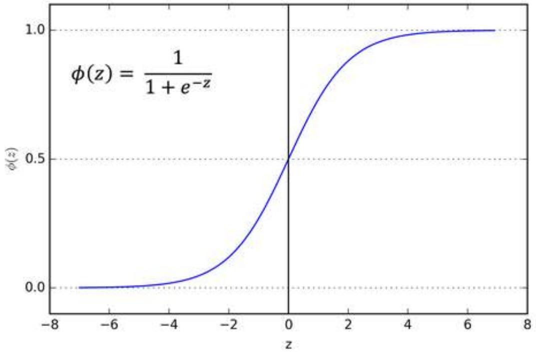
- Linear activation function
- Non-linear activation function

ARTIFICIAL NEURAL NETWORKS

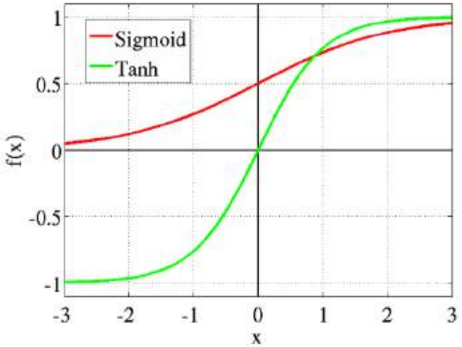
University of Essex | Online

Non-Linear Activation Function

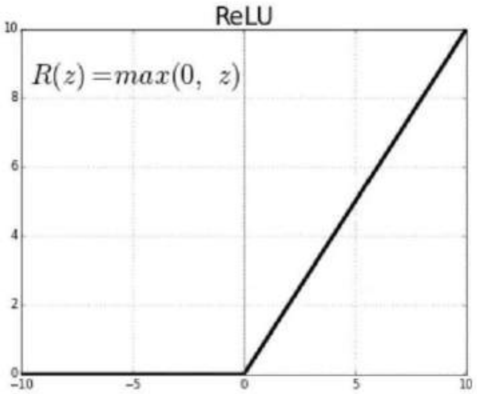
- Non-Linear Activation function has few types such as Sigmoid, Tanh, Rectified Linear Unit, or Leaky Relu



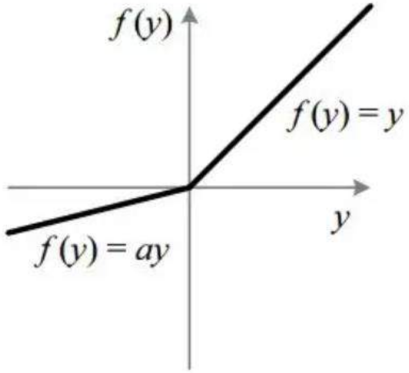
Sigmoid Function



Tanh Function



Relu



Leaky Relu

Image Source: Sharma, 2022

ARTIFICIAL NEURAL NETWORKS

Loss Function

- Loss function is used to see how well the predicted result against the target value (Yathish, 2022).
- There are few types of loss function, such as cross-entropy, binary cross-entropy, categorical cross-entropy, sparse categorical cross-entropy, or mean squared error.

Mean Squared Error

- One of the most popular lost function is Mean Squared Error
- Predicted and expected value should be real number
- Calculate the difference between predicted and target value, then squared it.
- The cost will be calculated as the average.

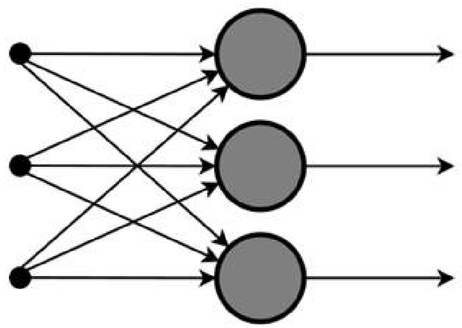
$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

TYPES OF ARTIFICIAL NEURAL NETWORKS

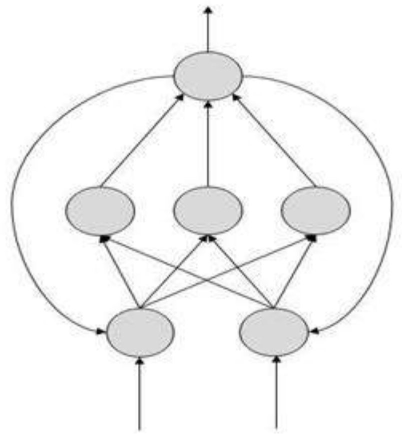
University of Essex | Online

Types of ANN

- Feed forward ANN
- Feedback ANN



Feed Forward ANN (Image Source: Ali, 2019)



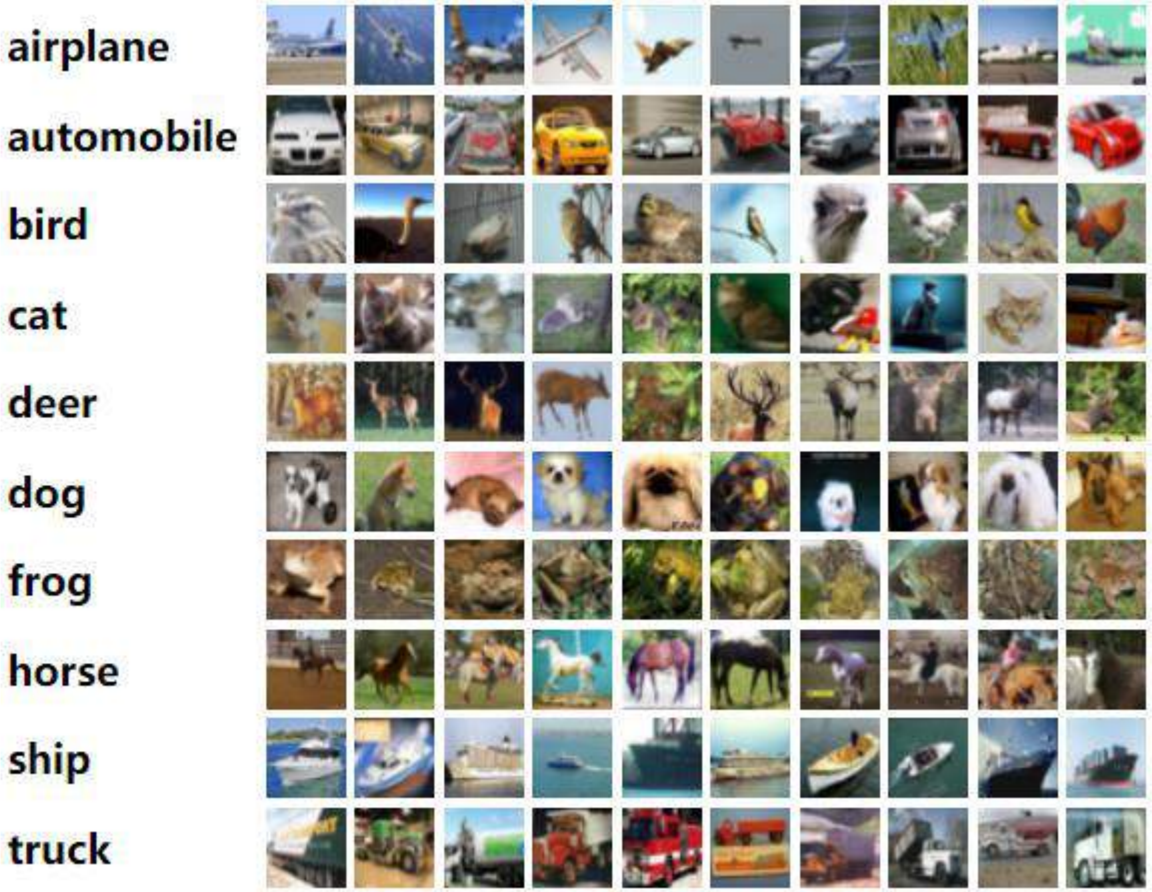
Feedback ANN (Image Source: Ali, 2019)

CIFAR-10 DATA

University of Essex | Online

CIFAR-10

- Collection of images used to train a machine learning
- Classify into 10 categories, which contains 60,000 images.
- CIFAR-10 images classified as airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck



IMPLEMENTATION

LIBRARIES

- The main library used is Keras.
- Keras is a library for deep learning which run on top of tensorflow (Keras, nd).

```
from __future__ import print_function
import keras
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten
from tensorflow.keras.callbacks import EarlyStopping
import os

import numpy as np

import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix, classification_report
import itertools

%matplotlib inline
```
✓ 11.8s

IMPLEMENTATION

LOAD CIFAR-10 DATA

- The data loaded using keras.dataset library into train and test set
- Train data contains 50,000 records with 32 x 32 size,
- Test data contains 10,000 records

```
# We split the data into train and test sets:
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train shape:', x_train.shape)
print('y_train shape:', y_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

✓ 0.7s

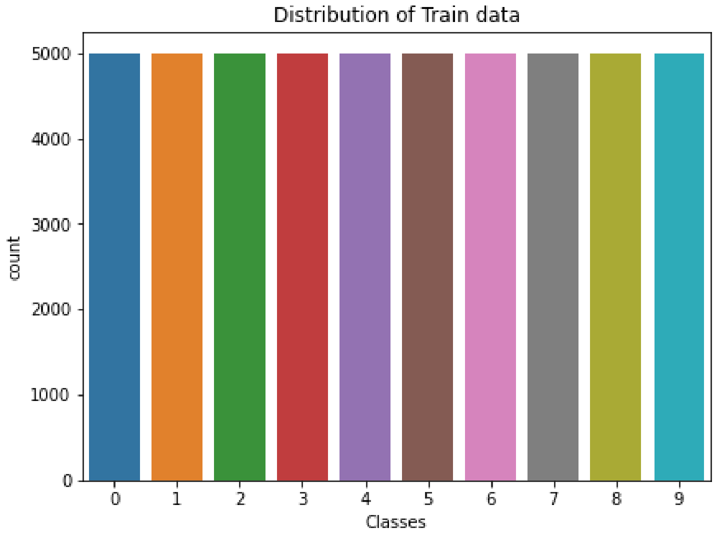
```
x_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 1)
50000 train samples
10000 test samples
```

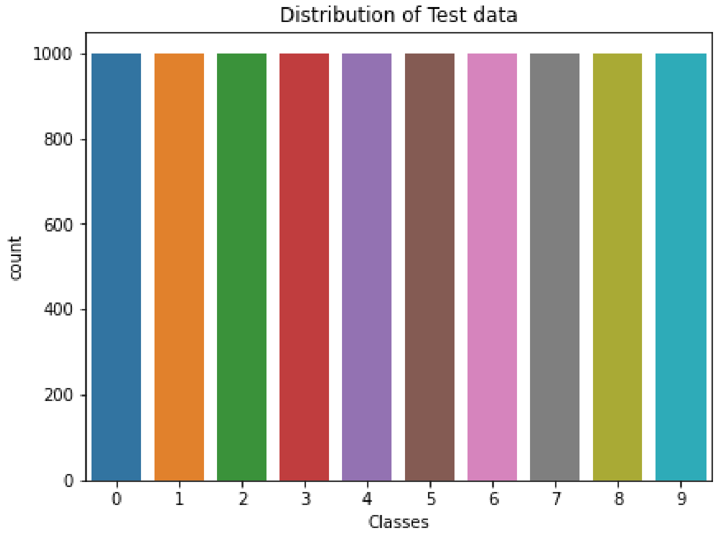
IMPLEMENTATION

University of Essex | Online

DATA PRE-PROCESSING

- Train and test data classified into 10 classes each
- The distribution of each class of train data is 5,000 for each class
- The distribution of each class of test data is 1,000 for each class





IMPLEMENTATION

University of Essex | Online

NORMALISATION

- Train and test data normalized into float data type
- The target variables are also converted into vectors by one hot encoding process
- Normalization ensures similarity of data across all dimensions.

```
# Normalize the data. Before we need to connvert data type to float for computation.
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

# Convert class vectors to binary class matrices. This is called one hot encoding.
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

IMPLEMENTATION

BUILDING NEURAL NETWORK

- The neural network is built using convolution, pooling, flatten and dense layers.
- The bigger the neural network, the faster is the compiling time and the better are the results.
- The layers were limited to avoid overfitting of the data.

```
# We Build the Neural Network in this step

model = Sequential()

model.add(Conv2D(filters = 32, kernel_size = (4,4), input_shape = (32, 32, 3), activation = "relu"))
model.add(MaxPool2D(pool_size = (2,2)))

model.add(Conv2D(filters = 64, kernel_size = (4,4), input_shape = (32, 32, 3), activation = "relu"))
model.add(MaxPool2D(pool_size = (2,2)))

model.add(Flatten())

model.add(Dense(512, activation = "relu"))
model.add(Dense(256, activation = "relu"))
model.add(Dense(128, activation = "relu"))

model.add(Dense(10, activation = "softmax"))
```

IMPLEMENTATION

University of Essex | Online

SUMMARIZING THE NEURAL NETWORK

- After building the network, we then observe how the neural net looks.
- The image summarizes the network perfectly.
- There are about a million trainable and no non-trainable parameters in the network.

```
#Summary of the neural net
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 29, 29, 32)        1568

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        32832

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 512)               819712

 dense_1 (Dense)             (None, 256)               131328

 dense_2 (Dense)             (None, 128)               32896

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 1,019,626
Trainable params: 1,019,626
Non-trainable params: 0
_________________________________________________________________
```

IMPLEMENTATION

University of Essex | Online

COMPILING THE NEURAL NETWORK

- After building and summarizing the network, we then move on to compile the neural net, which is the one last step before training the model.
- We use the 'adam' optimizer and the 'cross_entropy' parameter to monitor the model's performance and check the 'val_loss' parameter to ensure decent model performance.

```
# Compiling the neural net
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
early_stop = EarlyStopping(monitor = "val_loss", patience = 7)
```

IMPLEMENTATION

University of Essex | Online

TRAINING THE NEURAL NETWORK

- This is the step where we train our neural network with the training set of the CIFAR10 data.
- The image shoes the code required to do so.

```
cnnModel = model.fit(x_train, y_train,
                     batch_size=batch_size,
                     epochs= 15,
                     validation_data=(x_test, y_test),
                     shuffle=True,callbacks=[early_stop])
```

IMPLEMENTATION

University of Essex | Online

HOW THE NEURAL NET PERFORMS

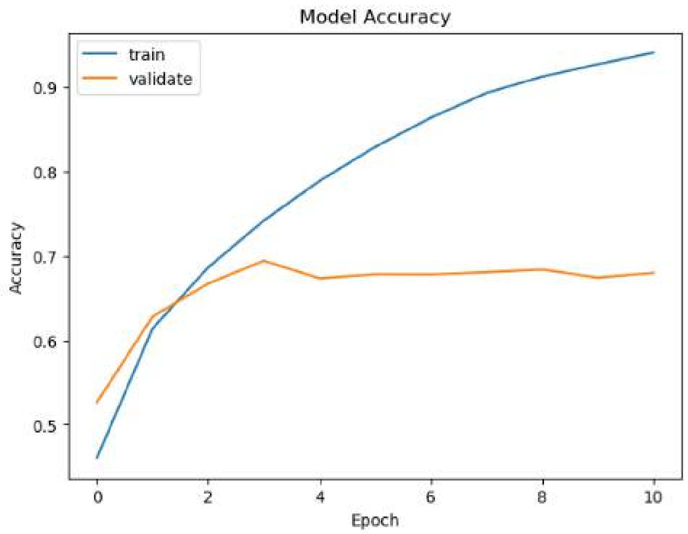
- The model's performance on the train data set is observed.
- The time taken for each iteration (epoch) is about 1 minute, on an average. This varies based on the system specifications.
- Bigger the computer, faster is the training time.

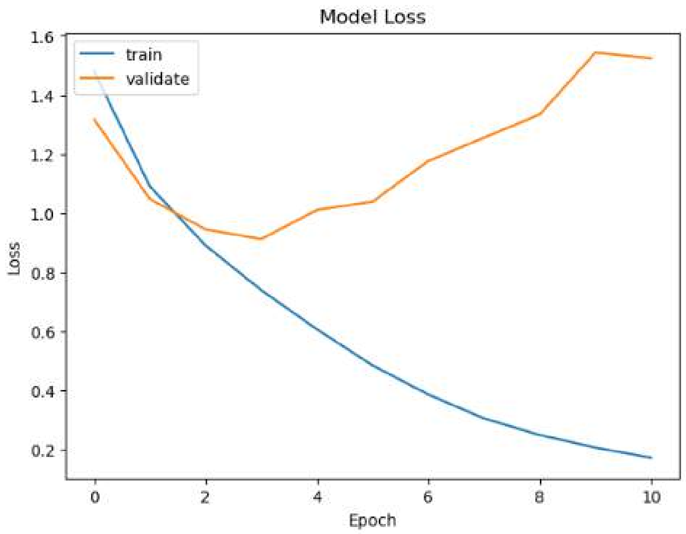
```
Epoch 1/15
1563/1563 [==============================] - 55s 35ms/step - loss: 1.4790 - accuracy: 0.4604 - val_loss: 1.3169 - val_accuracy: 0.5264
Epoch 2/15
1563/1563 [==============================] - 53s 34ms/step - loss: 1.0904 - accuracy: 0.6140 - val_loss: 1.0475 - val_accuracy: 0.6281
Epoch 3/15
1563/1563 [==============================] - 53s 34ms/step - loss: 0.8909 - accuracy: 0.6865 - val_loss: 0.9451 - val_accuracy: 0.6674
Epoch 4/15
1563/1563 [==============================] - 54s 35ms/step - loss: 0.7397 - accuracy: 0.7421 - val_loss: 0.9120 - val_accuracy: 0.6943
Epoch 5/15
1563/1563 [==============================] - 55s 35ms/step - loss: 0.6074 - accuracy: 0.7890 - val_loss: 1.0105 - val_accuracy: 0.6734
Epoch 6/15
1563/1563 [==============================] - 55s 35ms/step - loss: 0.4853 - accuracy: 0.8294 - val_loss: 1.0388 - val_accuracy: 0.6783
Epoch 7/15
1563/1563 [==============================] - 59s 37ms/step - loss: 0.3871 - accuracy: 0.8640 - val_loss: 1.1760 - val_accuracy: 0.6780
Epoch 8/15
1563/1563 [==============================] - 58s 37ms/step - loss: 0.3063 - accuracy: 0.8935 - val_loss: 1.2549 - val_accuracy: 0.6807
Epoch 9/15
1563/1563 [==============================] - 62s 40ms/step - loss: 0.2503 - accuracy: 0.9127 - val_loss: 1.3329 - val_accuracy: 0.6840
Epoch 10/15
1563/1563 [==============================] - 65s 41ms/step - loss: 0.2078 - accuracy: 0.9275 - val_loss: 1.5420 - val_accuracy: 0.6742
Epoch 11/15
1563/1563 [==============================] - 63s 40ms/step - loss: 0.1728 - accuracy: 0.9412 - val_loss: 1.5226 - val_accuracy: 0.6799
```

RESULTS AND TESTING

GRAPHICAL REPRESENTATION OF THE MODEL PERFORMANCE

- As explained, the loss decreases as the accuracy goes up with each iteration.





TESTING MODEL USING TEST DATASET

- The model is now tested with the test data of the CIFAR10 data.
- It has a score of 68% .

```
Test loss: 1.522578477859497
Test accuracy: 0.6798999905586243
```

RESULTS AND TESTING

University of Essex | Online

GENERATING THE CLASSIFICATION REPORT FOR ALL CLASSES

- Classification report gives the summary of important metrics like f1 score, precision, recall, etc. for all the classes in the CIFAR10 data.

```
print(classification_report(Y_true, Y_pred_classes))
```

	precision	recall	f1-score	support
0	0.75	0.69	0.72	1000
1	0.85	0.78	0.82	1000
2	0.58	0.57	0.57	1000
3	0.47	0.50	0.49	1000
4	0.65	0.61	0.63	1000
5	0.54	0.61	0.57	1000
6	0.78	0.71	0.74	1000
7	0.76	0.71	0.73	1000
8	0.72	0.85	0.78	1000
9	0.77	0.77	0.77	1000
accuracy			0.68	10000
macro avg	0.69	0.68	0.68	10000
weighted avg	0.69	0.68	0.68	10000

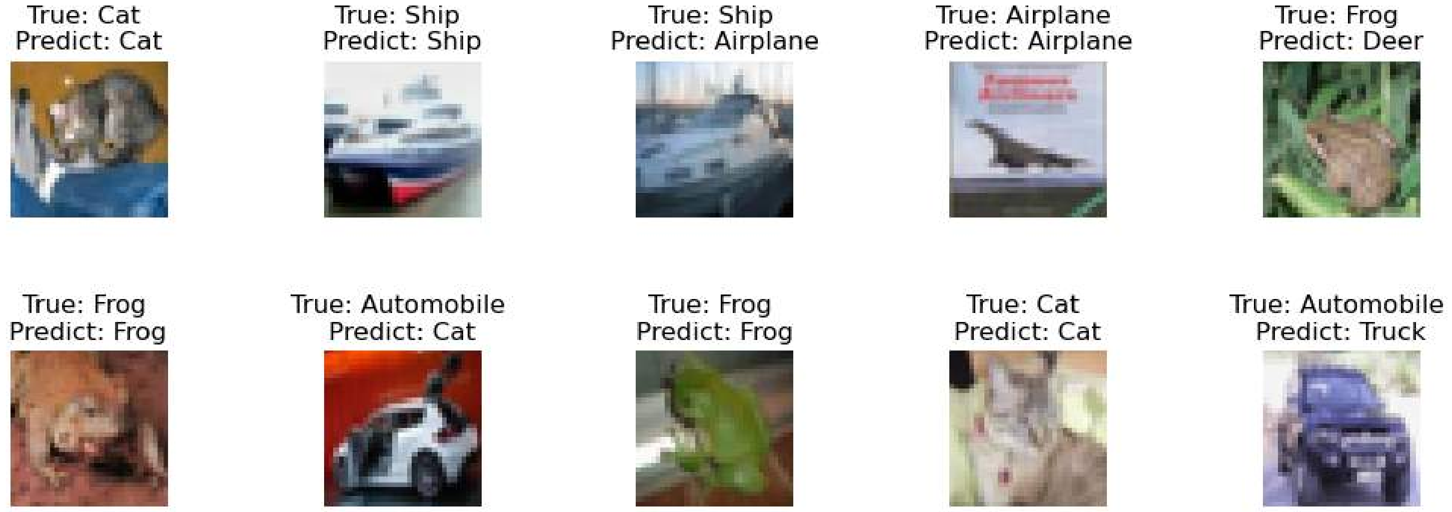
University of Essex | Online

VERIFYING THE PREDICTIONS – RIGHT, WRONG AND EXTREMELY WRONG

- We physically verify the correct, wrong and extremely wrong predictions made by our model.
- The interesting factor is the extremely wrong predictions.

```
rows = 5
cols = 5
fig, axes = plt.subplots(rows, cols, figsize=(12,12))
axes = axes.ravel()

for i in np.arange(0, rows*cols):
    axes[i].imshow(x_test[i])
    axes[i].set_title("True: %s \nPredict: %s" % (labels[Y_true[i]], labels[Y_pred_classes[i]]))
    axes[i].axis('off')
    plt.subplots_adjust(wspace=1)
```



RESULTS AND TESTING

VERIFYING THE PREDICTIONS – RIGHT, WRONG AND EXTREMELY WRONG (CONT'D)

- We physically verify the correct, wrong and extremely wrong predictions made by our model.
- The interesting factor is the extremely wrong predictions.

```
rows = 3
cols = 5
fig, axes = plt.subplots(rows, cols, figsize=(12,8))
axes = axes.ravel()

misclassified_idx = np.where(Y_pred_classes != Y_true)[0]
for i in np.arange(0, rows*cols):
    axes[i].imshow(x_test[misclassified_idx[i]])
    axes[i].set_title("True: %s \nPredicted: %s" % (labels[Y_true[misclassified_idx[i]]],
                                                  labels[Y_pred_classes[misclassified_idx[i]]]))
    axes[i].axis('off')
    plt.subplots_adjust(wspace=1)
```

RESULTS AND TESTING

VERIFYING THE PREDICTIONS – RIGHT, WRONG AND EXTREMELY WRONG (CONT'D)

- We physically verify the correct, wrong and extremely wrong predictions made by our model.
- The interesting factor is the extremely wrong predictions.

```
def display_errors(errors_index, img_errors, pred_errors, obs_errors):
    """ This function shows 10 images with their predicted and real labels"""
    n = 0
    rows = 2
    cols = 5
    fig, ax = plt.subplots(rows,cols,sharex=True,sharey=True, figsize=(12,6))
    for row in range(rows):
        for col in range(cols):
            error = errors_index[n]
            ax[row,col].imshow((img_errors[error]).reshape((32,32,3)))
            ax[row,col].set_title("Predicted:{}\nTrue:{}".
                                  format(labels[pred_errors[error]],labels[obs_errors[error]]))
            n += 1
            ax[row,col].axis('off')
            plt.subplots_adjust(wspace=1)

# Probabilities of the wrong predicted numbers
Y_pred_errors_prob = np.max(Y_pred_errors,axis = 1)

# Predicted probabilities of the true values in the error set
true_prob_errors = np.diagonal(np.take(Y_pred_errors, Y_true_errors, axis=1))

# Difference between the probability of the predicted label and the true label
delta_pred_true_errors = Y_pred_errors_prob - true_prob_errors

# Sorted list of the delta prob errors
sorted_dela_errors = np.argsort(delta_pred_true_errors)

# Top 10 errors
most_important_errors = sorted_dela_errors[-10:]

# Show the top 10 errors
display_errors(most_important_errors, X_test_errors, Y_pred_classes_errors, Y_true_errors)
```

THANK YOU!

References

- Ali, A. (2019) *Artificial Neural Network (ANN), Medium*. Wavy AI Research Foundation. Available at: https://medium.com/machine-learning-researcher/artificial-neural-network-ann-4481fa33d85a (Accessed: November 29, 2022).
- Deng, J. *et al.* (2020) A review of research on object detection based on Deep Learning, *Journal of Physics: Conference Series*, 1684(1), p. 012028. Available at: https://doi.org/10.1088/1742-6596/1684/1/012028.
- Dertat, A. (2017) *Applied deep learning - part 1: Artificial Neural Networks, Medium*. Towards Data Science. Available at: https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6 (Accessed: December 4, 2022).
- Dumane, G. (2020) *Introduction to convolutional neural network (CNN) using tensorflow, Medium*. Towards Data Science. Available at: https://towardsdatascience.com/introduction-to-convolutional-neural-network-cnn-
- de73f69c5b83#:~:text=Dense%20Layer%20is%20used%20to,on%20output%20from%20convolutional%20layers.&text=Each%20Layer%20in%20the%20Neural,as%20an%20%E2%80%9Cactivation%20function%E2%80%9D. (Accessed: December 4, 2022).

References

- Gallant, S.I. (1990) Perceptron-based learning algorithms. *IEEE Transactions on neural networks*, *1*(2), pp.179-191.
- Géron, A. (2019) *Hands-on machine learning with scikit-learn and tensorflow concepts, tools, and techniques to build Intelligent Systems*. Beijing: O'Reilly.
- Herculano-Houzel, S. (2012) "The remarkable, yet not extraordinary, human brain as a scaled-up primate brain and its associated cost," *Proceedings of the National Academy of Sciences*, 109(supplement_1), pp. 10661–10668. Available at: https://doi.org/10.1073/pnas.1201895109.
- Keras. (nd) *Keras documentation: About keras*, *Keras*. Available at: https://keras.io/about/ (Accessed: December 3, 2022).
- Krizhevsky, A. (2017) *CIFAR-10 and CIFAR-100 datasets*. Available at: https://www.cs.toronto.edu/~kriz/cifar.html (Accessed: December 4, 2022).

References

- Kubat, M. (2021) *An introduction to machine learning*. Cham: Springer.
- Kumar, S. (2020) *Overview of various optimizers in Neural Networks, Medium*. Towards Data Science. Available at: https://towardsdatascience.com/overview-of-various-optimizers-in-neural-networks-17c1be2df6d5#:~:text=Optimizers%20are%20algorithms%20or%20methods,problems%20by%20minimizing%20the%20function. (Accessed: December 4, 2022).
- Nadeem, Q. (2022) Seminar ANN. University of Essex Online.
- Patel, K. (2020) *Convolution neural networks - A beginner's guide [implementing a mnist hand-written digit..., Medium*. Towards Data Science. Available at: https://towardsdatascience.com/convolution-neural-networks-a-beginners-guide-implementing-a-mnist-hand-written-digit-8aa60330d022#:~:text=Max%20Pooling%20Layer&text=It%20is%20similar%20to%20the,2%20and%20stride%20of%201. (Accessed: December 4, 2022).
- Seb (2022) *An introduction to neural network loss functions, An Introduction to Neural Network Loss Functions*. Available at: https://programmathically.com/an-introduction-to-neural-network-loss-functions/ (Accessed: December 4, 2022).

References

- Sharma, S. (2022) *Activation functions in neural networks, Medium*. Towards Data Science. Available at: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 (Accessed: December 4, 2022).

- Team, S.D.S. (2018) "Convolutional Neural Networks (CNN): Step 3 - Flattening," *SuperDataScience*, 18 August. Available at: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening (Accessed: December 4, 2022).

- Yathish, V. (2022) *Loss functions and their use in neural networks, Medium*. Towards Data Science. Available at: https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9#:~:text=A%20loss%20function%20is%20a,the%20predicted%20and%20target%20outputs. (Accessed: December 4, 2022).