Experiment No.: 5

Title: Convolutional Neural Network

Batch: HO–DL 1 **Roll No.: 16010422234** **Experiment No. 05**

Aim: To implement Convolutional Neural Network.

Theory:

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs. They have three main types of layers as shown in figure 1, which are:

1. Convolutional layer
2. Pooling layer
3. Fully connected (FC) layer

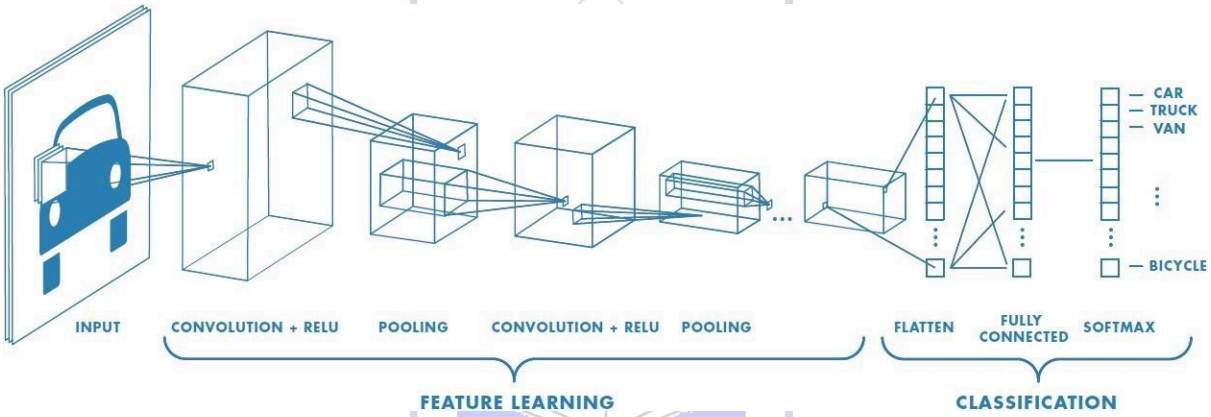


Figure 1: Architecture of CNN

The convolutional layer is the first layer of a convolutional network. While convolutional layers can be followed by additional convolutional layers or pooling layers, the fully connected layer is the final layer. With each layer, the CNN increases in its complexity, identifying greater portions of the image. Earlier layers focus on simple features, such as colors and edges. As the image data progresses through the layers of the CNN, it starts to recognize larger elements or shapes of the object until it finally identifies the intended object.

1. **Convolutional Layer**

The convolutional layer is the core building block of a CNN, and it is where the majority of computation occurs. It requires a few components, which are input data, a filter, and a feature map. Let's assume that the input will be a color image, which is made up of a matrix of pixels in 3D. This means that the input will have three dimensions—a height, width, and depth—which correspond to RGB in an image. We also have a feature detector, also known as a kernel or a filter, which will move across the receptive fields of the image, checking if the feature is present. This process is known as a convolution.

The feature detector is a two-dimensional (2-D) array of weights, which represents part of the image. While they can vary in size, the filter size is typically a 3x3 matrix; this also determines the size of the receptive field. The filter is then applied to an area of the image, and a dot product is calculated between the input pixels and the filter. This dot product is then fed into an output array. Afterwards, the filter shifts by a stride, repeating the process until the kernel has swept across the entire image. The final output from the series of dot products from the input and the filter is known as a feature map, activation map, or a convolved feature.

2. Pooling Layer

Pooling layers, also known as down sampling, conducts dimensionality reduction, reducing the number of parameters in the input. Similar to the convolutional layer, the pooling operation sweeps a filter across the entire input, but the difference is that this filter does not have any weights. Instead, the kernel applies an aggregation function to the values within the receptive field, populating the output array. There are two main types of pooling:

- **Max pooling**: As the filter moves across the input, it selects the pixel with the maximum value to send to the output array. As an aside, this approach tends to be used more often compared to average pooling.
- **Average pooling**: As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.

3. Fully Connected Layer

The name of the fully-connected layer aptly describes itself. As mentioned earlier, the pixel values of the input image are not directly connected to the output layer in partially connected layers. However, in the fully connected layer, each node in the output layer connects directly to a node in the previous layer. This layer performs the task of classification based on the features extracted through the previous layers and their different filters. While convolutional and pooling layers tend to use ReLu functions, FC layers usually leverage a softmax activation function to classify inputs appropriately, producing a probability from 0 to 1.

Activity:

1) Import requisite libraries using Tensorflow and Keras.
2) Load the selected dataset.
3) Visualize and display random images belonging to each class.
4) Develop the CNN model.
5) Print Model Summary and display architecture diagram.
6) Compile and fit the model on the train dataset.
7) Calculate training and the cross-validation accuracy.
8) Redefine the model by using appropriate regularization techniques to prevent overfitting.
9) Fit the data on the regularized model.
10) Calculate and plot loss function and accuracy using suitable loss function.

11) Display classification Report for un-regularized CNN model.
12) Display classification Report for regularized CNN model.
13) Comment on output.

Program:

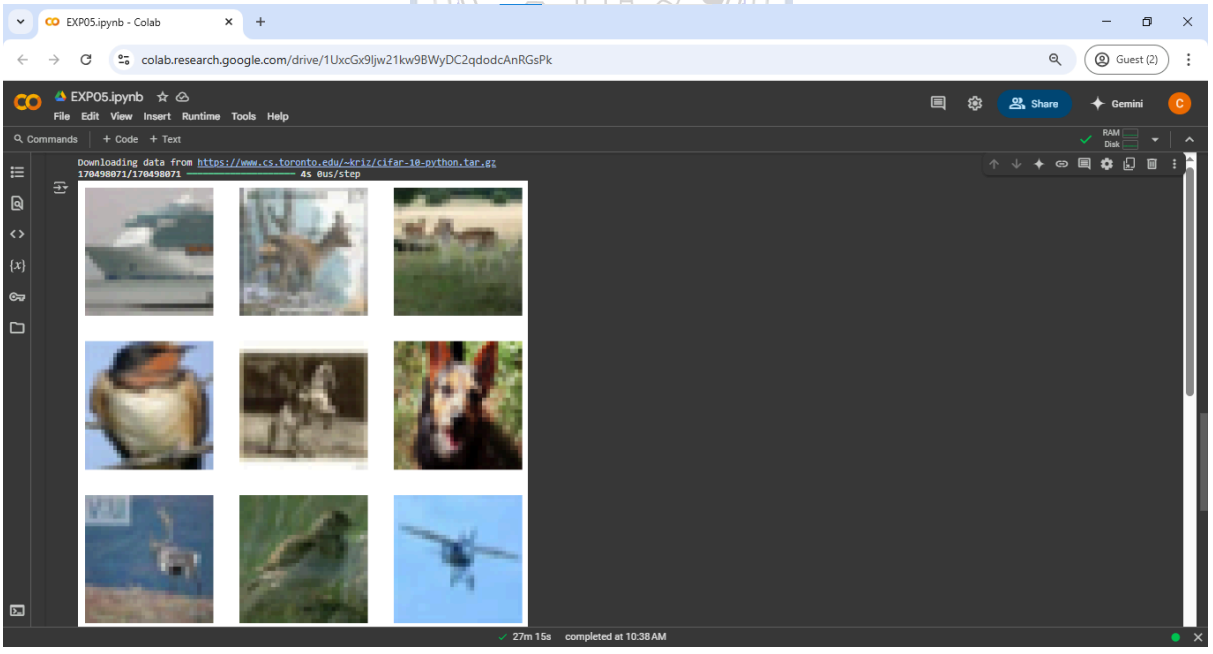
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from tensorflow.keras.datasets import cifar10

# Load the dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize the dataset
x_train, x_test = x_train / 255.0, x_test / 255.0

# Display random images from dataset
fig, axes = plt.subplots(3, 3, figsize=(8, 8))
class_names = ["airplane", "automobile", "bird", "cat", "deer", "dog",
"frog", "horse", "ship", "truck"]
for i in range(3):
    for j in range(3):
        index = np.random.randint(0, len(x_train))
        axes[i, j].imshow(x_train[index])
        axes[i, j].axis("off")
plt.show()

# Build the CNN model
model = keras.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32,
3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
```

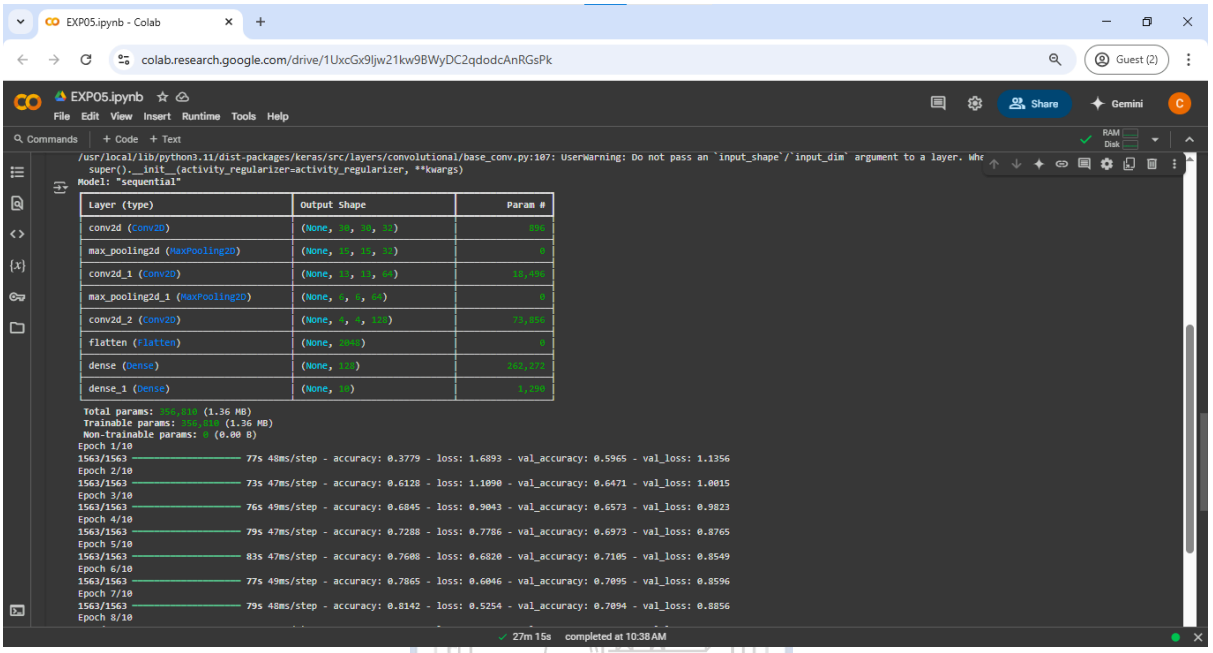
```
])

# Print model summary
model.summary()

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train, epochs=10,
validation_data=(x_test, y_test))

# Evaluate model performance
train_acc = history.history['accuracy'][-1]
val_acc = history.history['val_accuracy'][-1]
print(f"Training Accuracy: {train_acc:.4f}, Validation Accuracy:
{val_acc:.4f}")

# Apply Regularization
model_reg = keras.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32,
3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(128, activation='relu',
kernel_regularizer=keras.regularizers.l2(0.01)),
    layers.Dense(10, activation='softmax')
])

# Compile and train the regularized model
model_reg.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history_reg = model_reg.fit(x_train, y_train, epochs=10,
validation_data=(x_test, y_test))

# Plot loss and accuracy
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Unregularized Loss')
```

```
plt.plot(history_reg.history['loss'], label='Regularized Loss')
plt.legend()
plt.title('Loss Comparison')

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Unregularized Accuracy')
plt.plot(history_reg.history['accuracy'], label='Regularized Accuracy')
plt.legend()
plt.title('Accuracy Comparison')
plt.show()

# Classification report for unregularized model
y_pred = np.argmax(model.predict(x_test), axis=1)
print("Classification Report (Unregularized Model):")
print(classification_report(y_test, y_pred, target_names=class_names))

# Classification report for regularized model
y_pred_reg = np.argmax(model_reg.predict(x_test), axis=1)
print("Classification Report (Regularized Model):")
print(classification_report(y_test, y_pred_reg,
target_names=class_names))
```

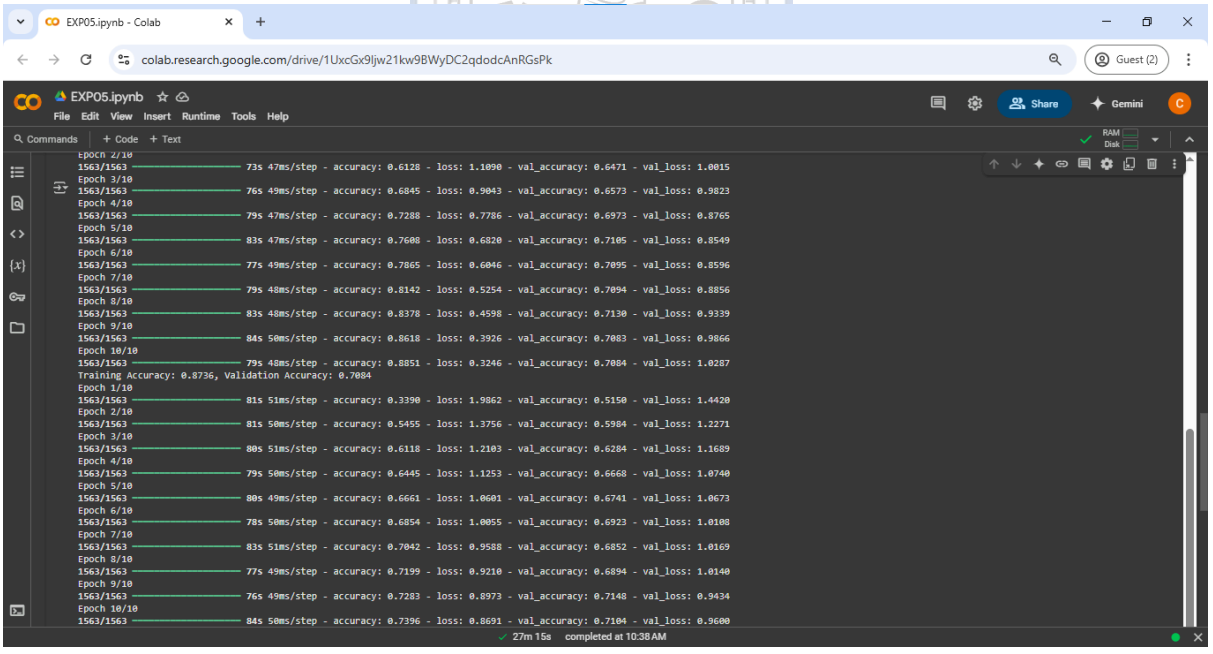
K. J. SOMAIYA COLLEGE OF ENGG.

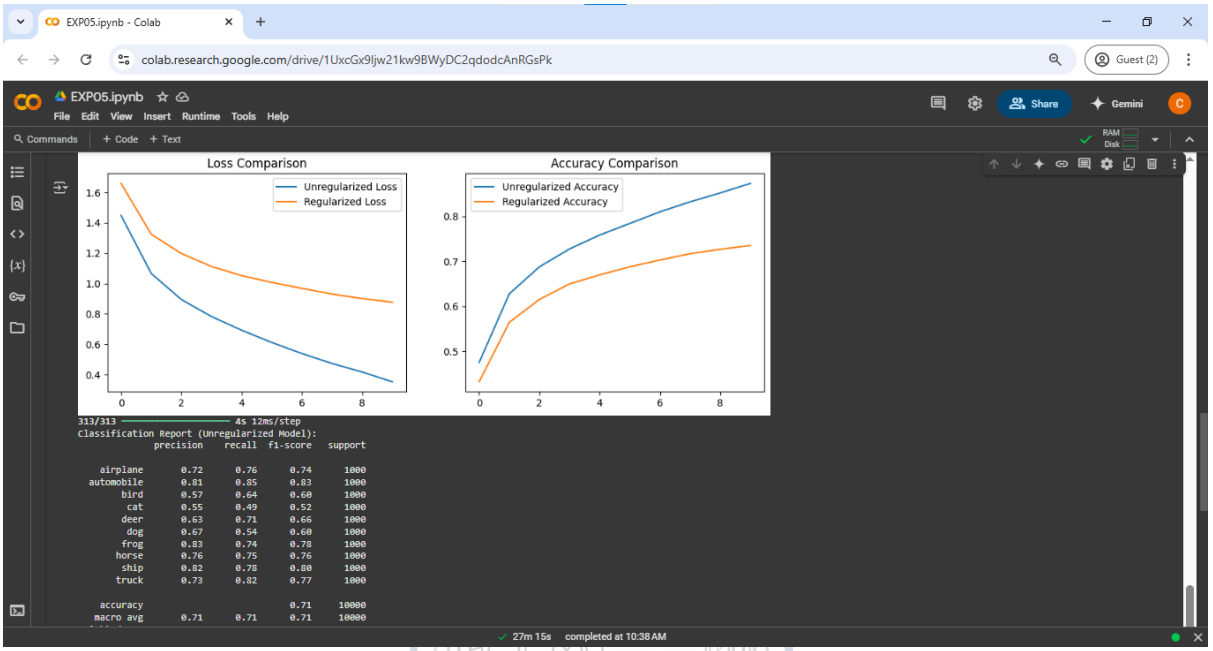
Output:

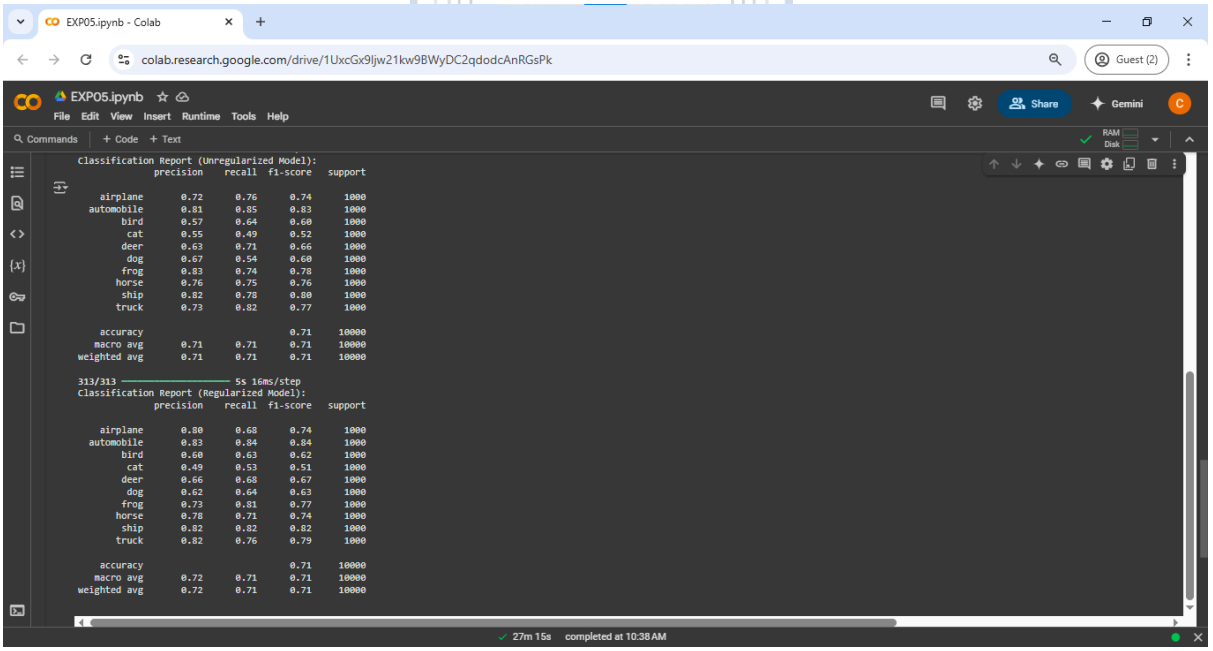




K. J. SOMAIYA COLLEGE OF ENGG.




EXP05.ipynb - Colab
colab.research.google.com/drive/1UxcGx9ljw21kw9BWyDC2qdodcAnRGsPk
EXP05.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Loss Comparison
Unregularized Loss
Regularized Loss
Accuracy Comparison
Unregularized Accuracy
Regularized Accuracy
313/313 4s 12ms/step
Classification Report (Unregularized Model):
precision recall f1-score support
airplane 0.72 0.76 0.74 1000
automobile 0.81 0.85 0.83 1000
bird 0.57 0.64 0.60 1000
cat 0.55 0.49 0.52 1000
deer 0.63 0.71 0.66 1000
dog 0.67 0.54 0.60 1000
frog 0.83 0.74 0.78 1000
horse 0.76 0.75 0.76 1000
ship 0.82 0.78 0.80 1000
truck 0.73 0.82 0.77 1000
accuracy 0.71 10000
macro avg 0.71 0.71 0.71 10000
27m 15s completed at 10:38 AM


EXP05.ipynb - Colab
colab.research.google.com/drive/1UxcGx9ljw21kw9BWyDC2qdodcAnRGsPk
EXP05.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Classification Report (Unregularized Model):
precision recall f1-score support
airplane 0.72 0.76 0.74 1000
automobile 0.81 0.85 0.83 1000
bird 0.57 0.64 0.60 1000
cat 0.55 0.49 0.52 1000
deer 0.63 0.71 0.66 1000
dog 0.67 0.54 0.60 1000
frog 0.83 0.74 0.78 1000
horse 0.76 0.75 0.76 1000
ship 0.82 0.78 0.80 1000
truck 0.73 0.82 0.77 1000
accuracy 0.71 10000
macro avg 0.71 0.71 0.71 10000
weighted avg 0.71 0.71 0.71 10000
313/313 5s 16ms/step
Classification Report (Regularized Model):
precision recall f1-score support
airplane 0.80 0.68 0.74 1000
automobile 0.83 0.84 0.84 1000
bird 0.60 0.63 0.62 1000
cat 0.49 0.53 0.51 1000
deer 0.66 0.68 0.67 1000
dog 0.62 0.64 0.63 1000
frog 0.73 0.81 0.77 1000
horse 0.78 0.71 0.74 1000
ship 0.82 0.82 0.82 1000
truck 0.82 0.76 0.79 1000
accuracy 0.71 10000
macro avg 0.72 0.71 0.71 10000
weighted avg 0.72 0.71 0.71 10000
27m 15s completed at 10:38 AM

CO–3: Assimilate fundamentals of Convolutional Neural Network.

Conclusion:

In this experiment, we implemented a Convolutional Neural Network (CNN) for image classification using TensorFlow and Keras. We built the model with convolutional, pooling, and fully connected layers, and evaluated its performance using accuracy and classification reports. Regularization techniques were applied to prevent overfitting, improving the model's generalization. The results demonstrated the effectiveness of CNNs in image classification tasks.

Grade: AA / AB / BB / BC / CC / CD / DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

1. Josh Patterson and Adam Gibson, "Deep Learning A Practitioner's Approach", O'Reilly Media, 2017

2. Nikhil Buduma, "Fundamentals of Deep Learning Designing Next-Generation Machine Intelligence Algorithms", O'Reilly Media 2017

3. Ian Goodfellow Yoshua Bengio Aaron Courville. "Deep Learning", MIT Press 2017
