HW-Topic-4

Data Acquisition, Modeling and Analysis: Big Data Analytics

Submitted By – **Sudhanshu Kakkar**
CWID – **20036779**

Deep Learning

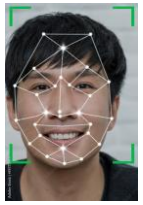
WHAT IS IT?

Deep learning is a subfield of **machine learning** that uses **artificial neural networks** with multiple layers (hence "deep") to analyze data, learn patterns, and make decisions or predictions.

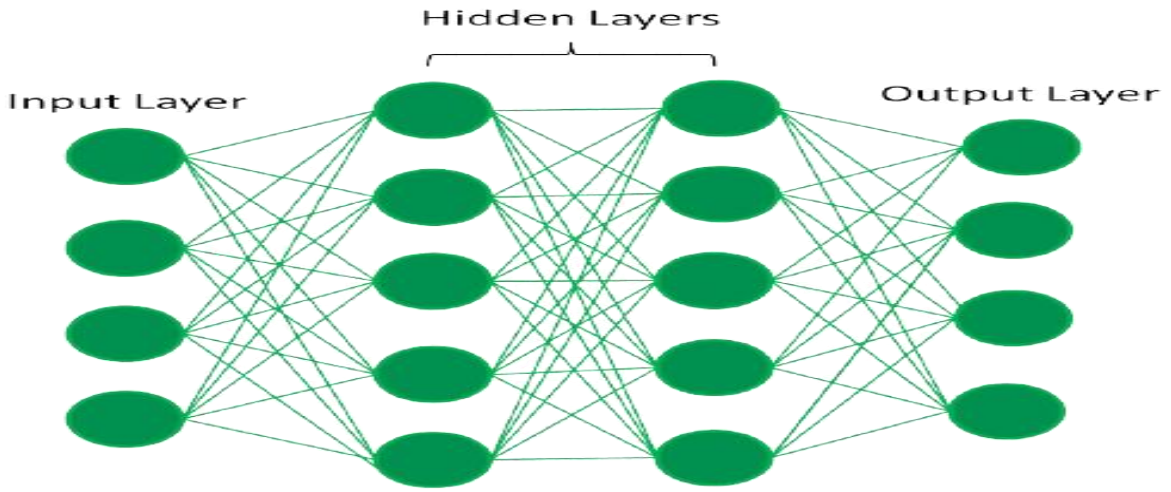
KEY CONCEPTS

- **Neural Networks:** Artificial neural networks with multiple hidden layers (deep) form the computational core, inspired by the human brain.
- **Feature Extraction:** Deep learning automatically learns the most important features directly from raw data, eliminating manual feature engineering.

APPLICATIONS



- Self Driving Cars
- Natural Language Processing
- Movie Recommendation Systems
- Natural Language Processing
- Computer Vision
- Financial Fraud Detection
- Healthcare



THE PROCESS

1. **Forward Propagation:** The input data is processed through the network's layers, where initial weights are applied to make the model's first prediction (the "guess").

2. **Measure Loss:** A mathematical function calculates the difference (the "loss") between the model's prediction and the actual correct answer, measuring how wrong the guess was.

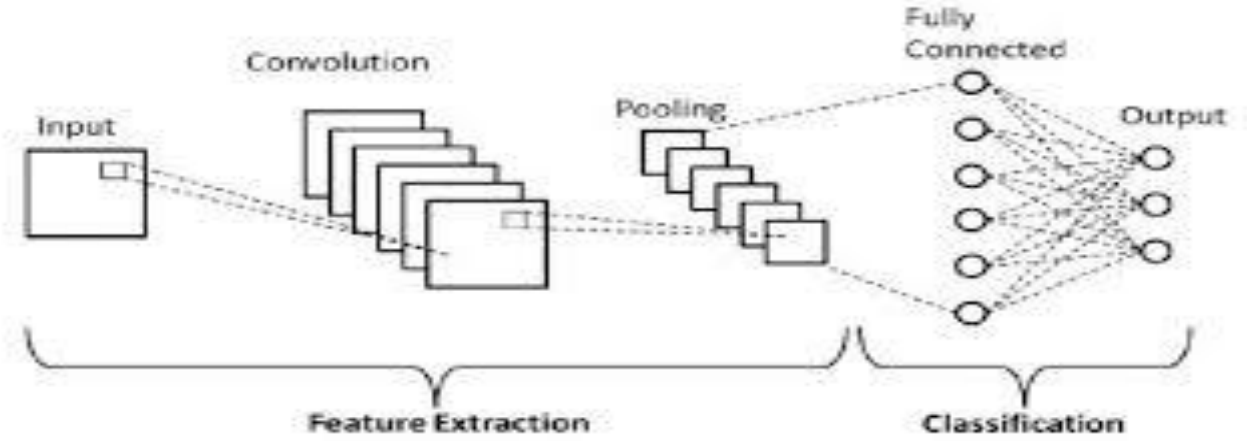
3. **Backpropagation:** The error signal is sent backward through the network layers, revealing how much each internal connection (weight) contributed to the total error.

4. **Weight Adjustment:** Using the error feedback, an optimizer (Gradient Descent) makes small, iterative adjustments to the weights and biases to ensure the next guess is more accurate.

Convolution Neural Network

WHAT IS IT?

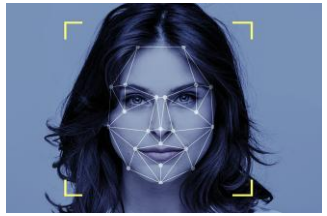
A **Convolutional Neural Network (CNN)** is a type of deep learning network specifically designed to process data with a known grid-like structure, such as **images**. Its process is specialized in automatically extracting visual features.



KEY CONCEPTS

- **Convolution:** Filter slides over input to extract features (edges, textures).
- **Filter / Kernel**: A small learnable weight matrix that detects one specific feature
- **Parameter Sharing**: A single filter's weights are reused across the entire image.
- **Pooling (Max):** Down samples feature map size, adding robustness to shifts.
- **Flatten**: Converts 3D features into a 1D vector for final classification

APPLICATIONS



- Facial Recognition
- Autonomous Vehicles
- Image Classification
- OCR / Handwriting recognition
- Product recommendation engine

THE PROCESS

1. **Convolution:** Extracts features (edges, textures, patterns) by applying a filter (kernel) across the input.
2. **Activation (ReLU):** Introduces non-linearity to the extracted features, enabling the model to learn complex data maps.
3. **Pooling:** Reduces the spatial dimensions (width and height) of the feature data.
4. **Flattening:** Transforms the 2D/3D feature maps into a single 1D vector.
5. **Fully Connected:** Learns high-level global patterns from the flattened feature vector.
6. **Output (Softmax):** Calculates the final probability for each potential class label.

Recurrent Neural Network

WHAT IS IT?

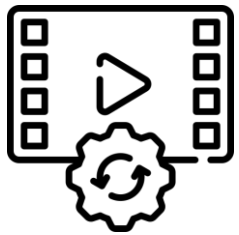
A **Recurrent Neural Network (RNN)** is a type of deep learning model specifically designed to process **sequential data** or data where the order matters. Unlike a traditional neural network, an RNN has an internal memory that allows it to remember information from previous steps in the sequence.

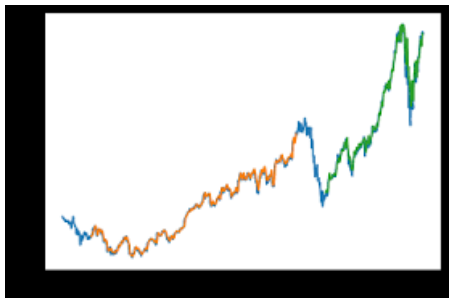
KEY CONCEPTS

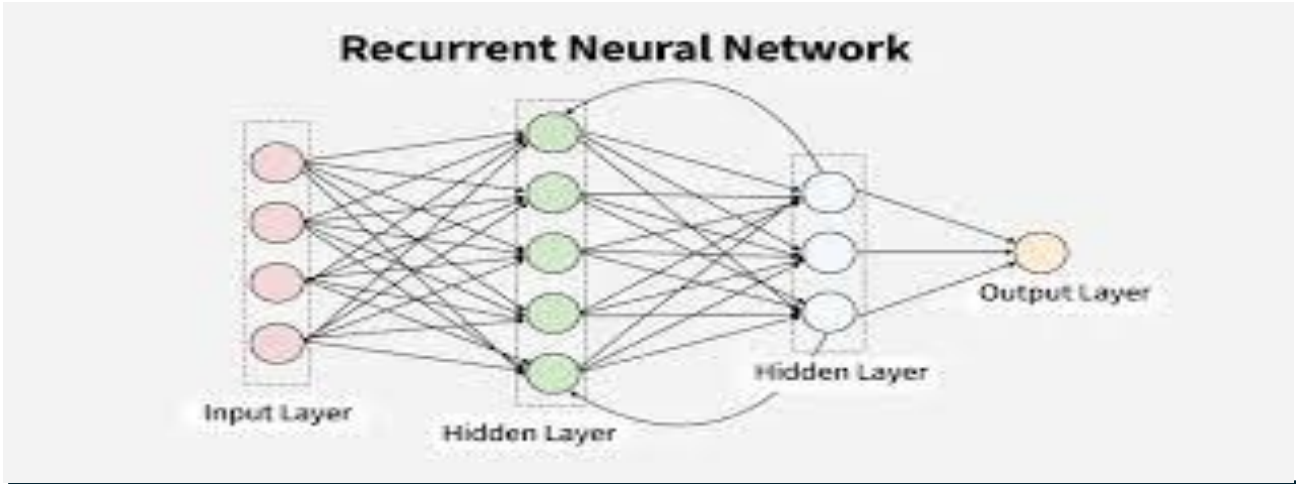
- **Contextual Understanding:** Uses internal memory to process current data based on its entire history.
- **Variable Length Input:** Naturally handles sequences of any length (e.g., short or long sentences).
- **Sequence Mapping:** Great at mapping one sequence to another (like translation) or to a single label.
- **Long-Term Memory (LSTM):** Advanced versions use gates to remember critical info over extended periods.

APPLICATIONS

- Sentiment Analysis
- Video Processing
- Stock Prices Prediction
- Text Prediction







THE PROCESS

1. **Input Embedding:** Converts the sequential element (like a word) into a **dense numerical vector** for the network to understand.
2. **Initial Hidden State (Old Memory):** Sets the starting **internal memory** (context) to an empty state.
3. **Recurrence (Calculation):** Takes the **old memory** and the **current input** to compute the **new, updated memory**.
4. **Weight Sharing: Reuses the same set of weights** (rules) for every single time step in the sequence.
5. **Final Hidden State (Full Context):** The memory vector that holds the **complete understanding** of the entire sequence.
6. **Output (Prediction):** Uses the memory (from a specific step or the final one) to make a final **prediction** (e.g., translation, sentiment).

CNN Code Demo (python)

Training and Model Building Step

```
# initializing the image generators each for training and validation set

#using 20% of train images for validation and rest 80% for training

train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_generator = train_datagen.flow_from_directory(
    base_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    subset='training',
    shuffle=True
)

validation_generator = train_datagen.flow_from_directory(
    base_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    subset='validation',
    shuffle=False
)
```

In the above code snippet, we are dividing the data set into 20% validation and rest 80% for training.

```
# building the model here
#as per the approach 3 I am using 3 convolution layers and respective maxpooling layers

model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)), #1st con layer
    MaxPooling2D(2,2),
    Conv2D(64, (3,3), activation='relu'), #2nd conv layer
    MaxPooling2D(2,2),
    Conv2D(128, (3,3), activation='relu'), #3rd conv layer
    MaxPooling2D(2,2),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(1)
])

model.compile(
    optimizer=Adam(learning_rate=1e-5, clipnorm=1.0),
    loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
    metrics=['accuracy']
)
```

In the above code snippet, we are building the model with 3 convolution layers.

Testing Step

```python
from tensorflow.keras.models import load_model
import numpy as np
from tensorflow.keras.preprocessing import image

#loading the model for image label testing
model = load_model('./cat_dog_model.h5')

def predict_image(model, img_path):
    # processing the image to make it ready to feed into the model for prediction
    img = image.load_img(img_path, target_size=(150, 150))
    img_array = image.img_to_array(img)
    img_array = img_array / 255.0  # normalize
    img_array = np.expand_dims(img_array, axis=0)  # add batch dimension

    prediction = model.predict(img_array)[0][0]  # get prediction value

    if prediction > 0.5:
        print(f"Prediction: dog ({prediction:.2f})")
    else:
        print(f"Prediction: Cat ({1 - prediction:.2f})")

imgPath = "./cat.14.jpg"  # input image 1
predict_image(model, imgPath);

imgPath = "./dog.5.jpg" # image image 2
predict_image(model, imgPath);
```

In the above code snippet, we are testing the trained model with images of cat and dog.

Input – Dog image

Output –

```
Prediction: dog (0.9101)
```