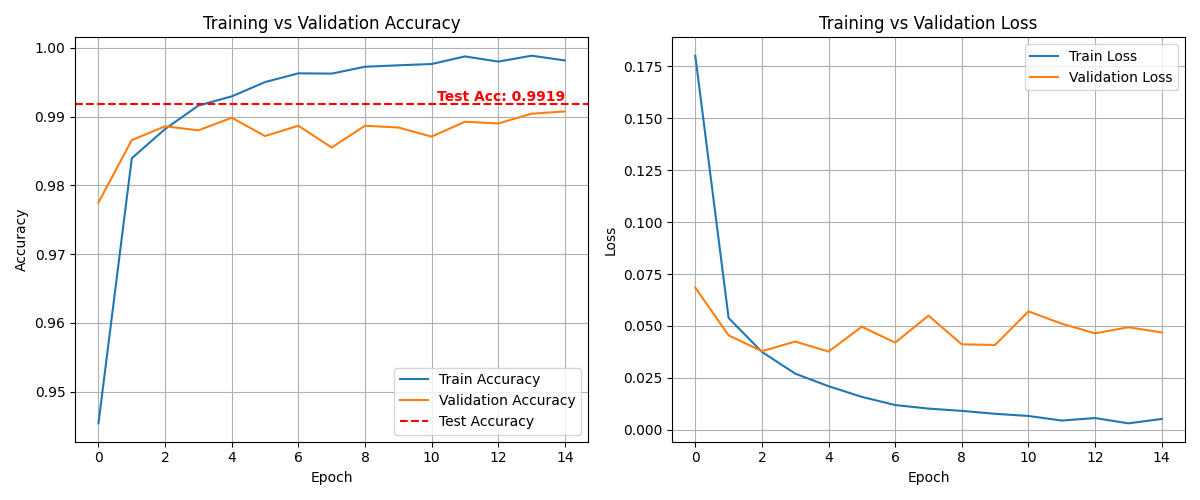
**Report of Week 1 Task**

**Given Tasks:**

1. Implement image digit recognition task using MNIST dataset (suggested to use entire data) with train, validation and test split.
2. For the given entire data i.e., without changing the train, validation and test split, please modify the architecture such that it will overfit.
3. Now, without changing the architecture, train the model to overcome the overfitting problem for the same data

**Task 1:**  
  
Basic architecture:

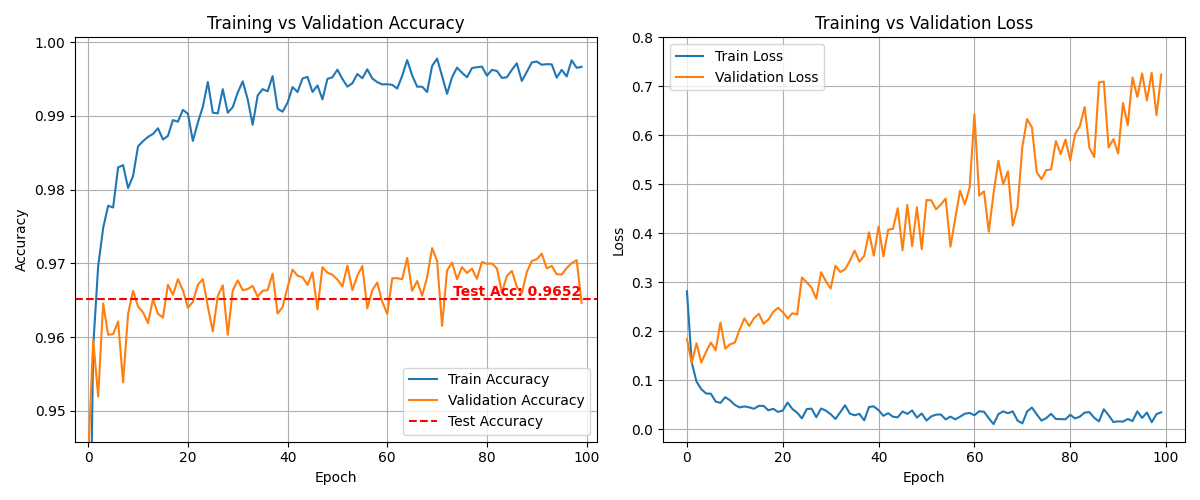
* Input layer with shape (28, 28, 1) for grayscale MNIST images.
* 1st Conv2D layer with 32 filters, 3×3 kernel size, and ReLU activation.
* 1st MaxPooling2D layer with 2×2 pool size.
* 2nd Conv2D layer with 64 filters, 3×3 kernel size, and ReLU activation.
* 2nd MaxPooling2D layer with 2×2 pool size.
* Flatten layer to convert 2D feature maps into a 1D vector.
* Dense layer with 128 neurons and ReLU activation for feature learning.
* Output Dense layer with 10 neurons and softmax activation for classification.
* Model compiled with Adam optimizer, sparse categorical cross entropy loss, and accuracy metric.
* Trained for 15 epochs with batch size 64 and 20% validation split.



**Task 2: Overfitting**

Changes made to architecture:

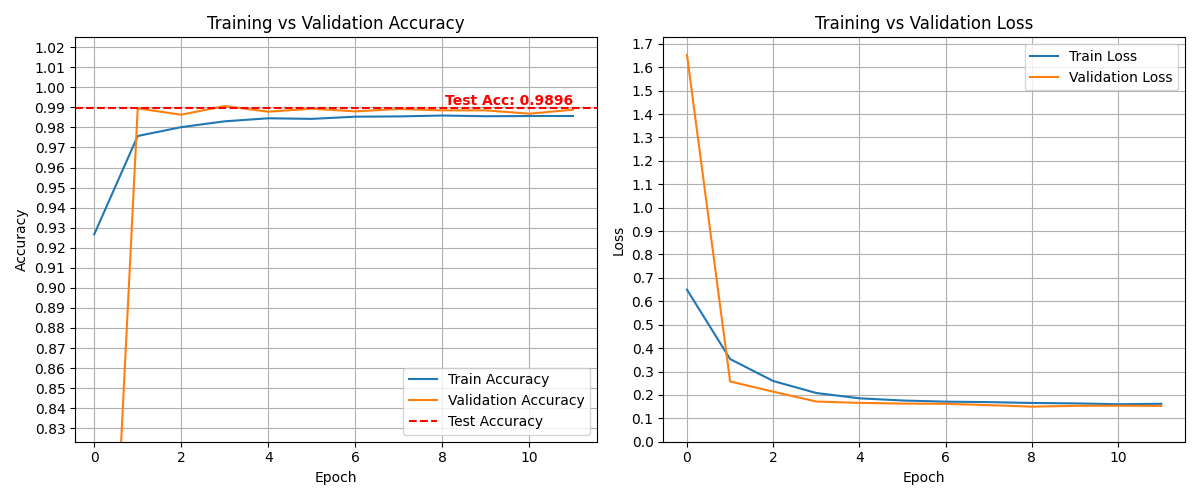
* convolution layer to bring the total to 2 convolution layers with 64, 128 filters.
* Increased the number of neurons in the 1st dense layer from 64 to 128.
* Increased number of epochs fromo 15 to 100.
* Used a learning rate of 0.001 with Adam optimizer.



**Task 3: Solving Overfitting**

Introduced the various Overfitting reduction techniques like:

* **L2 Regularization:** Applied regularizers.l2(0.001) to Conv2D and Dense.
* **Dropout:** Added Dropout(0.25) after convolutional blocks and Dropout(0.5) before final output.
* **Batch Normalization:** Added after Conv2D and Dense.
* **Early Stopping:** Training stops early with a patience of 3 epochs.
* Used a relatively high learning rate of 0.009 with Adam optimizer.
* Increased the batch size to 128.



**Theoretical concepts**

**Convolutional Neural Network (CNN)**

A Convolutional Neural Network is a class of deep learning models that is highly effective for image processing, pattern recognition, and spatial data analysis.

**Key Components of a Convolutional Neural Network:**

* [**Convolutional Layers**](https://www.geeksforgeeks.org/what-are-convolution-layers/)**:** These layers apply convolutional operations to input images using filters or kernels to detect features such as edges, textures and more complex patterns.
* [**Pooling Layers**](https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/)**:** They downsample the spatial dimensions of the input, reducing the computational complexity and the number of parameters in the network.
* [**Activation Functions**](https://www.geeksforgeeks.org/activation-functions/)**:** They introduce non-linearity to the model by allowing it to learn more complex relationships in the data.
* [**Fully Connected Layers**](https://www.geeksforgeeks.org/what-is-fully-connected-layer-in-deep-learning/)**:** These layers are responsible for making predictions based on the high-level features learned by the previous layers. They connect every neuron in one layer to every neuron in the next layer.
* [**Convolutional Layers**](https://www.geeksforgeeks.org/what-are-convolution-layers/)**(Conv2D)**
* A convolutional layer is the core building block of a Convolutional Neural Network that performs a mathematical operation called convolution to extract features from input data.
* detect local patterns such as edges, corners, textures, and more complex structures.
* It performs element-wise multiplication and sums the result to produce a feature map.

**Conv2D(64, (3, 3), activation='relu', input\_shape=(28, 28, 1))**

**Conv2D(128, (3, 3), activation='relu')**

* Parameters used:
  + **filters** – Specifies how many feature detectors (kernels) the layer learns, determining the output depth (eg : 32,64 or 128…).
  + **kernel\_size** – Defines the height and width of the sliding window (filter) applied to the input image.(eg : (3,3) or (5,5) …)
  + **activation** – Applies a non-linear function like ReLU to the output to help the network learn complex patterns.
  + **input\_shape** – Indicates the shape of input data to the first convolution layer (e.g (28, 28, 1) for grayscale).
* **Pooling Layers(Max Pooling)**
* Pooling layers reduce the spatial dimensions (width and height) of feature maps while preserving important information.
* It decreases computational complexity , helps prevent overfitting , provides spatial invariance.
* **Max Pooling**: Selects the maximum value from the window

**MaxPooling2D((2, 2))**

* **Pool Size = (2, 2)** → Takes a 2×2 region and returns only the max value.
* It reduces size from 28×28 → 14×14.
* Max Pooling follows first Conv layer helps to summarize low-level features like edges.
* Max Pooling follows second Conv layer helps to focus on more abstract features.
* [**Activation Functions**](https://www.geeksforgeeks.org/activation-functions/)**(ReLu)**
* Introduce non-linearity into the neural network, allowing it to learn complex patterns.
* Without activation functions, a neural network would behave like a linear model regardless of depth.
* **Common Activation Functions :**
  + ReLU (Rectified Linear Unit)
  + Sigmoid
  + Tanh
  + Softmax
* **ReLU**
  + Used in all hidden layers (Conv2D and Dense layers).
  + Used to extract and pass useful patterns forward while ignoring negative activations.
  + Helps the model learn non-linear patterns in image features.
  + Fast, efficient, and effective in deep networks.
* **Softmax**
  + Used in the **output layer** (Dense(10, activation='softmax')).
  + Used to predict the most likely digit by comparing class probabilities.
  + Converts the final output to **probabilities for each digit class (0–9)**.
  + Ensures the sum of output values equals 1 (i.e., valid probability distribution).
* [**Fully Connected Layers**](https://www.geeksforgeeks.org/what-is-fully-connected-layer-in-deep-learning/)
* Every neuron in a fully connected (Dense) layer is connected to every neuron in the previous layer.
* Combines features learned by convolutional and pooling layers to make final predictions.
* **Flatten Layer**
  + Converts multi-dimensional (2D/3D) feature maps into a 1D vector.
  + Acts as a bridge between convolutional/pooling layers and fully connected (Dense) layers.
  + It transforms the output of the last pooling layer to a flat vector before passing to Dense.
* **Dense Layer**
  + Fully connected neural network layer; each neuron receives input from all neurons in the previous layer.
  + Performs classification or regression based on extracted features.

**Dense(128, activation='relu')**

**Dense(10, activation='softmax')**

* + **Parameters**:
    - units: Number of neurons (e.g : Dense(128) has 128 neurons).
    - activation: Function like 'relu', 'softmax' to introduce non-linearity.

**Overfitting**

* Overfitting occurs when a machine learning model learns not only the underlying patterns in the training data but also the noise and random fluctuations.
* As a result, the model performs very well on training data but poorly on new, unseen test data, indicating poor generalization.
* This happens when the model is too complex, has too many parameters, or is trained too long without regularization.
* Symptoms include high training accuracy but low validation/test accuracy.
* **Common solutions include using**:
* **Dropout layers**
* **Regularization (L1/L2)**
* **Early stopping**
* **Batch Normalization**
* **Dropout layers**
* Dropout randomly disables a fraction of neurons during training, forcing the model to not rely on specific paths and thus promoting robustness.
* Helps prevent overfitting by introducing noise and reducing co-adaptations of neurons**.**

**Dropout(0.25)**

**Dropout(0.5)**

* After the first and second convolution + pooling blocks, Dropout(0.25) is applied — meaning 25% of neurons are ignored at each step.
* After dense layers, Dropout(0.5) is applied — 50% of neurons are turned off.
* Reduces overfitting by forcing the network to learn robust patterns that do not rely on specific neurons.
* **Regularization (L1/L2)**
* **L1 Regularization** adds the sum of absolute weights to the loss function, encouraging sparsity (many weights become zero).
* **L2 Regularization** adds the sum of squared weights to the loss function, discouraging large weights and promoting simpler models.
* Keeps weights small and generalizes better on unseen data.

**kernel\_regularizer=regularizers.l2(0.001)**

* L2 regularization adds a penalty to the loss function based on the squared value of the weights.
* Helps reduce overfitting by discouraging large weights.
* In your model, every Conv2D and Dense layer uses L2 regularization with a penalty factor of 0.001.
* **Early stopping**
* **EarlyStopping** monitors validation loss and stops training when it doesn't improve for a set number of epochs.
* Prevents overfitting by halting training before the model starts to memorize the training data.

**early\_stop = EarlyStopping(monitor='val\_loss', patience=3, restore\_best\_weights=True)**

* Early stopping stops training when the validation loss doesn’t improve for 3 consecutive epochs (patience=3).
* restore\_best\_weights=True ensures the model goes back to the best weights observed during training.
* Prevents the model from over-training and degrading performance on validation/test data
* **Batch Normalization**
* **Batch Normalization** normalizes the output of a layer by re-centering and re-scaling, which speeds up training and improves stability.
* It also has a slight regularizing effect and can reduce the need for dropout in some cases.
* Stabilizes training, allows for higher learning rates, and may reduce the need for Dropout.