NNDL_LAB4_2348045

In [ ]:

```
#1
```

In [4]:

```
import warnings
warnings.filterwarnings("ignore")
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras import Input, Model  # Import Input and Model

input_dim = 100  # Example input dimension
output_dim = 10  # Example output dimension (for classification with 10 classe

# Sequential API
model_seq = models.Sequential([
    layers.Dense(512, activation='relu', name='H-Layer-1', input_shape=(input_
                 kernel_initializer='he_normal', bias_initializer='zeros', ke
    layers.Dense(512, activation='relu', name='H-Layer-2',
                 kernel_initializer='he_normal', bias_initializer='zeros', ke
    layers.Dense(1024, activation='relu', name='H-Layer-3',
                 kernel_initializer='he_normal', bias_initializer='zeros', ke
    layers.Dense(output_dim, activation='softmax', name='O-Layer',
                 kernel_initializer='he_normal', bias_initializer='zeros', ke
])

model_seq.summary()
```

```
Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━
┃ Layer (type)                         ┃ Output Shape                ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━
│ H-Layer-1 (Dense)                    │ (None, 512)                 │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ H-Layer-2 (Dense)                    │ (None, 512)                 │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ H-Layer-3 (Dense)                    │ (None, 1024)                │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ O-Layer (Dense)                      │ (None, 10)                  │
└──────────────────────────────────────┴─────────────────────────────┴───────

 Total params: 849,930 (3.24 MB)

 Trainable params: 849,930 (3.24 MB)

 Non-trainable params: 0 (0.00 B)
```

Sequential API Model The Sequential API model in the notebook is constructed using TensorFlow's Keras library. It consists of three hidden dense layers (H-Layer-1, H-Layer-2, H-Layer-3) with ReLU activation functions, which help introduce non-linearity into the model. The output layer (O-Layer) utilizes a softmax activation function, suitable for multi-class classification problems like the MNIST dataset. The model summary indicates it has a total of 849,930 trainable parameters. This architecture ensures that the model has sufficient capacity to learn the underlying patterns in the data while maintaining simplicity in its structure.

In [5]:

```
# Functional API
inputs = Input(shape=(input_dim,))
x = layers.Dense(512, activation='relu', name='H-Layer-1',
                 kernel_initializer='he_normal', bias_initializer='zeros', ke
x = layers.Dense(512, activation='relu', name='H-Layer-2',
                 kernel_initializer='he_normal', bias_initializer='zeros', ke
x = layers.Dense(1024, activation='relu', name='H-Layer-3',
                 kernel_initializer='he_normal', bias_initializer='zeros', ke
outputs = layers.Dense(output_dim, activation='softmax', name='O-Layer',
                       kernel_initializer='he_normal', bias_initializer='zero

model_func = Model(inputs, outputs)

model_func.summary()
```

```
Model: "functional_2"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━
┃ Layer (type)                         ┃ Output Shape                 ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━
│ input_layer_1 (InputLayer)           │ (None, 100)                  │
├──────────────────────────────────────┼──────────────────────────────┼──────
│ H-Layer-1 (Dense)                    │ (None, 512)                  │
├──────────────────────────────────────┼──────────────────────────────┼──────
│ H-Layer-2 (Dense)                    │ (None, 512)                  │
├──────────────────────────────────────┼──────────────────────────────┼──────
│ H-Layer-3 (Dense)                    │ (None, 1024)                 │
├──────────────────────────────────────┼──────────────────────────────┼──────
│ O-Layer (Dense)                      │ (None, 10)                   │
└──────────────────────────────────────┴──────────────────────────────┴──────

 Total params: 849,930 (3.24 MB)

 Trainable params: 849,930 (3.24 MB)

 Non-trainable params: 0 (0.00 B)
```

Functional API Model The Functional API model follows a similar architecture to the Sequential API model but uses TensorFlow's Input, Model, and layers.Dense classes to define the model more flexibly. This approach allows for more complex architectures where layers can be reused or connected in non-linear ways. Despite this flexibility, the model maintains the same structure: three dense layers with ReLU activations and an output layer with a softmax activation. The summary of this model also reveals 849,930 trainable parameters, indicating

In [ ]:

```
#2
```

In [10]:

```
import numpy as np
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


# Load data
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(-1, 784).astype('float32') / 255.0
x_test = x_test.reshape(-1, 784).astype('float32') / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

In [11]:

```
# Simple Dense Layers
model1 = Sequential([
    Dense(512, activation='relu', input_shape=(784,)),
    Dense(512, activation='relu'),
    Dense(10, activation='softmax')
])

model1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['a
history1 = model1.fit(x_train, y_train, validation_split=0.2, epochs=10, batc

test_loss1, test_acc1 = model1.evaluate(x_test, y_test)
print(f'Architecture 1 Test accuracy: {test_acc1}')
```

```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 15s 8ms/step - accuracy: 0.8958 - loss: 0.348
2 - val_accuracy: 0.9601 - val_loss: 0.1356
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - accuracy: 0.9724 - loss: 0.090
2 - val_accuracy: 0.9711 - val_loss: 0.0949
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - accuracy: 0.9828 - loss: 0.058
6 - val_accuracy: 0.9686 - val_loss: 0.1096
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 8ms/step - accuracy: 0.9865 - loss: 0.043
5 - val_accuracy: 0.9717 - val_loss: 0.1034
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - accuracy: 0.9898 - loss: 0.031
6 - val_accuracy: 0.9743 - val_loss: 0.1001
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9904 - loss: 0.027
6 - val_accuracy: 0.9753 - val_loss: 0.1069
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9933 - loss: 0.021
0 - val_accuracy: 0.9775 - val_loss: 0.1053
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9943 - loss: 0.018
4 - val_accuracy: 0.9752 - val_loss: 0.1253
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 14s 9ms/step - accuracy: 0.9942 - loss: 0.017
5 - val_accuracy: 0.9768 - val_loss: 0.1096
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 13s 9ms/step - accuracy: 0.9939 - loss: 0.016
3 - val_accuracy: 0.9797 - val_loss: 0.1114
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9732 - loss: 0.1217
Architecture 1 Test accuracy: 0.9782999753952026
```

In [12]:

```
from tensorflow.keras.layers import Dropout

# Adding Dropout
model2 = Sequential([
    Dense(512, activation='relu', input_shape=(784,)),
    Dropout(0.2),
    Dense(512, activation='relu'),
    Dropout(0.2),
    Dense(10, activation='softmax')
])

model2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['a
history2 = model2.fit(x_train, y_train, validation_split=0.2, epochs=10, batc

test_loss2, test_acc2 = model2.evaluate(x_test, y_test)
print(f'Architecture 2 Test accuracy: {test_acc2}')
```

```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 17s 8ms/step - accuracy: 0.8855 - loss: 0.377
8 - val_accuracy: 0.9654 - val_loss: 0.1162
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9670 - loss: 0.107
7 - val_accuracy: 0.9694 - val_loss: 0.0989
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9731 - loss: 0.082
6 - val_accuracy: 0.9737 - val_loss: 0.0855
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9805 - loss: 0.063
6 - val_accuracy: 0.9759 - val_loss: 0.0833
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9842 - loss: 0.049
2 - val_accuracy: 0.9767 - val_loss: 0.0846
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9842 - loss: 0.049
9 - val_accuracy: 0.9739 - val_loss: 0.1091
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9856 - loss: 0.046
2 - val_accuracy: 0.9779 - val_loss: 0.0906
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9870 - loss: 0.040
8 - val_accuracy: 0.9778 - val_loss: 0.0977
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9885 - loss: 0.035
3 - val_accuracy: 0.9771 - val_loss: 0.1095
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9886 - loss: 0.038
0 - val_accuracy: 0.9799 - val_loss: 0.0920
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9739 - loss: 0.1180
Architecture 2 Test accuracy: 0.9790999889373779
```

```
In [13]: from tensorflow.keras.layers import BatchNormalization

         # Adding Batch Normalization
         model3 = Sequential([
             Dense(512, activation='relu', input_shape=(784,)),
             BatchNormalization(),
             Dense(512, activation='relu'),
             BatchNormalization(),
             Dense(10, activation='softmax')
         ])

         model3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['a
         history3 = model3.fit(x_train, y_train, validation_split=0.2, epochs=10, batc

         test_loss3, test_acc3 = model3.evaluate(x_test, y_test)
         print(f'Architecture 3 Test accuracy: {test_acc3}')
```

```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 17s 8ms/step - accuracy: 0.8970 - loss: 0.
3470 - val_accuracy: 0.9625 - val_loss: 0.1259
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9649 - loss: 0.
1097 - val_accuracy: 0.9719 - val_loss: 0.0937
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9732 - loss: 0.
0855 - val_accuracy: 0.9694 - val_loss: 0.1014
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9770 - loss: 0.
0719 - val_accuracy: 0.9735 - val_loss: 0.0936
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9816 - loss: 0.
0577 - val_accuracy: 0.9747 - val_loss: 0.0862
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9854 - loss: 0.
0468 - val_accuracy: 0.9762 - val_loss: 0.0889
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9860 - loss: 0.
```

Simple Dense Layers Model:

This model, without any additional regularization techniques, achieves a test accuracy of approximately 97.83%. It serves as a baseline to compare the effects of dropout and batch normalization.

Model with Dropout:

Dropout layers are added after each dense layer to prevent overfitting by randomly dropping units during training. This model achieves a slightly improved test accuracy of around 97.91%, indicating that dropout effectively enhances generalization.

Model with Batch Normalization:

Batch normalization layers are added after each dense layer to stabilize and speed up training by normalizing the output of the previous activation layer. This model achieves a test accuracy of approximately 97.90%, demonstrating that batch normalization also contributes to improved performance.

In [ ]:
```
#3
```

In [14]:
```
# Common model
def create_model():
    model = Sequential([
        Dense(512, activation='relu', input_shape=(784,)),
        Dense(512, activation='relu'),
        Dense(10, activation='softmax')
    ])
    return model
```

In [16]:
```
# Compile with adam optimizer and categorical_crossentropy loss
model_adam = create_model()
model_adam.compile(optimizer='adam', loss='categorical_crossentropy', metrics
history_adam = model_adam.fit(x_train, y_train, validation_split=0.2, epochs=
```

```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 14s 8ms/step - accuracy: 0.8953 - loss: 0.335
5 - val_accuracy: 0.9649 - val_loss: 0.1124
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 8ms/step - accuracy: 0.9744 - loss: 0.082
2 - val_accuracy: 0.9682 - val_loss: 0.1063
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - accuracy: 0.9831 - loss: 0.053
6 - val_accuracy: 0.9722 - val_loss: 0.0977
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - accuracy: 0.9864 - loss: 0.042
1 - val_accuracy: 0.9758 - val_loss: 0.0878
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 8ms/step - accuracy: 0.9909 - loss: 0.028
3 - val_accuracy: 0.9773 - val_loss: 0.0825
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 8ms/step - accuracy: 0.9919 - loss: 0.022
8 - val_accuracy: 0.9735 - val_loss: 0.1119
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 8ms/step - accuracy: 0.9919 - loss: 0.025
3 - val_accuracy: 0.9770 - val_loss: 0.1034
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9949 - loss: 0.016
5 - val_accuracy: 0.9652 - val_loss: 0.1639
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9946 - loss: 0.018
9 - val_accuracy: 0.9743 - val_loss: 0.1321
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9928 - loss: 0.020
8 - val_accuracy: 0.9797 - val_loss: 0.1049
```

In [17]:
```python
from tensorflow.keras.metrics import Precision

# Compile with sgd optimizer and binary_crossentropy loss
model_sgd = create_model()
model_sgd.compile(optimizer='sgd', loss='binary_crossentropy', metrics=[Preci
history_sgd = model_sgd.fit(x_train, y_train, validation_split=0.2, epochs=10
```

```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - loss: 0.3635 - precision: 0.635
6 - val_loss: 0.2153 - val_precision: 0.9939
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 0.1966 - precision: 0.978
1 - val_loss: 0.1427 - val_precision: 0.9539
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 0.1391 - precision: 0.943
8 - val_loss: 0.1126 - val_precision: 0.9364
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 0.1128 - precision: 0.930
4 - val_loss: 0.0961 - val_precision: 0.9328
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 0.0985 - precision: 0.926
1 - val_loss: 0.0855 - val_precision: 0.9314
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 0.0888 - precision: 0.924
2 - val_loss: 0.0783 - val_precision: 0.9315
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - loss: 0.0811 - precision: 0.924
4 - val_loss: 0.0730 - val_precision: 0.9323
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 0.0759 - precision: 0.924
9 - val_loss: 0.0689 - val_precision: 0.9334
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 0.0731 - precision: 0.925
6 - val_loss: 0.0655 - val_precision: 0.9345
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 0.0688 - precision: 0.928
5 - val_loss: 0.0628 - val_precision: 0.9350
```

In [18]:

```
from tensorflow.keras.metrics import Recall

# Compile with rmsprop optimizer and mean_squared_error loss
model_rmsprop = create_model()
model_rmsprop.compile(optimizer='rmsprop', loss='mean_squared_error', metrics
history_rmsprop = model_rmsprop.fit(x_train, y_train, validation_split=0.2, e
```

```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - loss: 0.0220 - recall: 0.7772
- val_loss: 0.0063 - val_recall: 0.9517
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - loss: 0.0061 - recall: 0.9529
- val_loss: 0.0053 - val_recall: 0.9598
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - loss: 0.0038 - recall: 0.9724
- val_loss: 0.0042 - val_recall: 0.9682
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - loss: 0.0028 - recall: 0.9804
- val_loss: 0.0039 - val_recall: 0.9711
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - loss: 0.0023 - recall: 0.9848
- val_loss: 0.0040 - val_recall: 0.9711
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 8ms/step - loss: 0.0017 - recall: 0.9884
- val_loss: 0.0037 - val_recall: 0.9730
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - loss: 0.0014 - recall: 0.9908
- val_loss: 0.0033 - val_recall: 0.9775
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - loss: 0.0012 - recall: 0.9928
- val_loss: 0.0037 - val_recall: 0.9751
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - loss: 9.3744e-04 - recall: 0.9
940 - val_loss: 0.0035 - val_recall: 0.9762
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 7ms/step - loss: 7.7890e-04 - recall: 0.9
952 - val_loss: 0.0031 - val_recall: 0.9792
```