Deep Learning Multi-Class Classification for Customer Segmentation

An automobile company aims to predict customer segments for new potential customers. This project utilizes deep learning and Keras to create multi-class classification models. The goal is to classify customers into four segments (A, B, C, D) to enhance targeted marketing efforts.

Credits & Mentor : Arup Das

Dataset Preparation and Preprocessing

1

Data

Loading the datasets

2

Handling Missing values

Handling Missing values by imputing numerical and categorical variables in test.csv and train.csv with suitable imputation techniques without dropping any of them, ensuring consistent preprocessing of train.csv with test.csv.

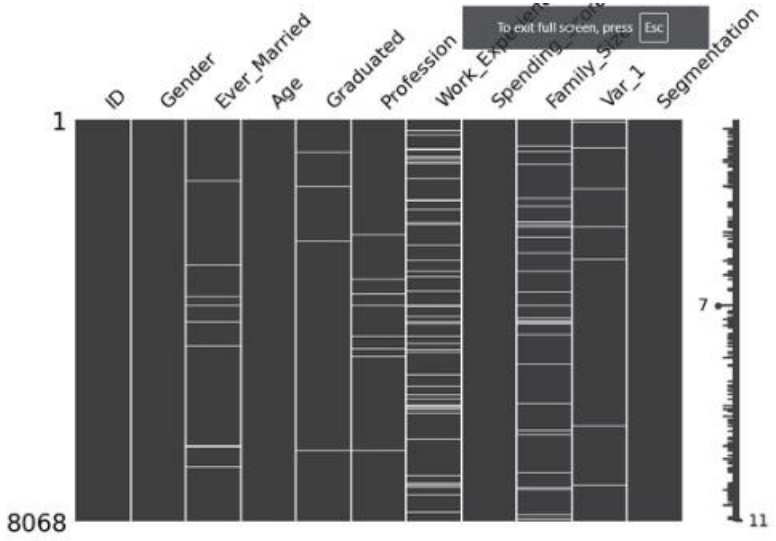
3

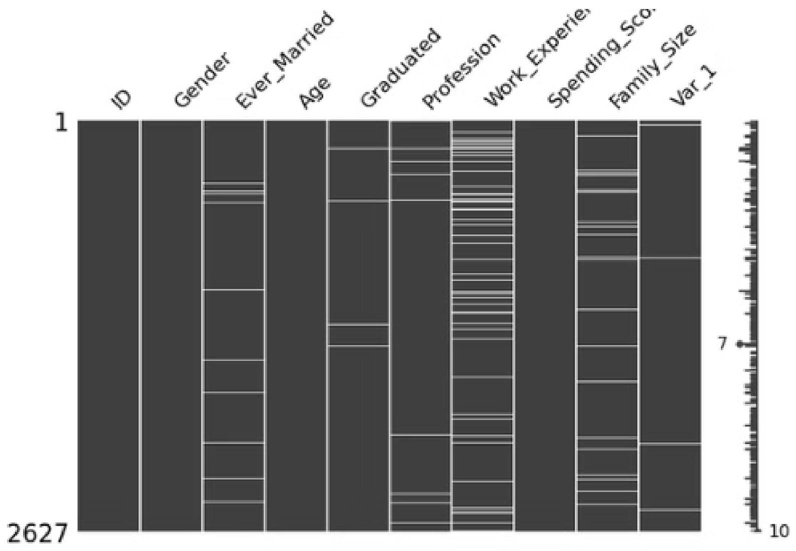
Scaling and Encoding

Apply necessary scaling and encoding techniques to both datasets. Used one-hot encoding for categorical variables.

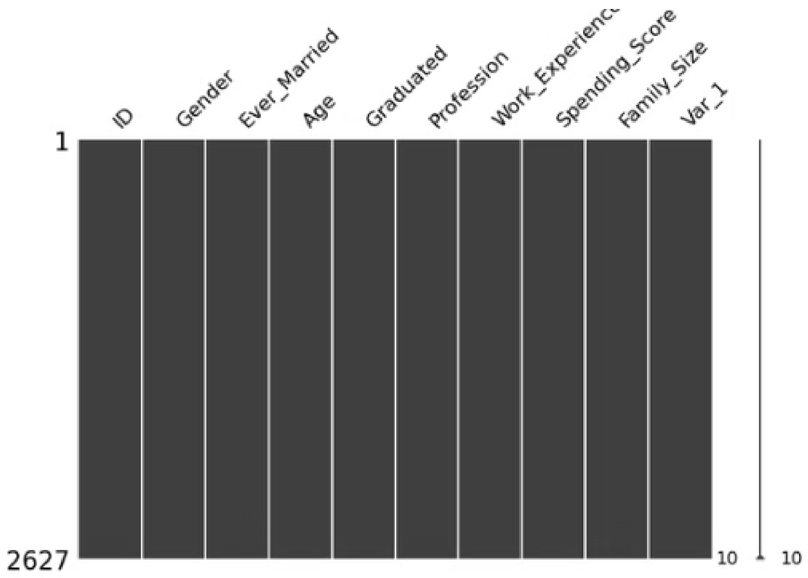
Missingo Representation of Missing values in train and test datasets before and after handling the missing values

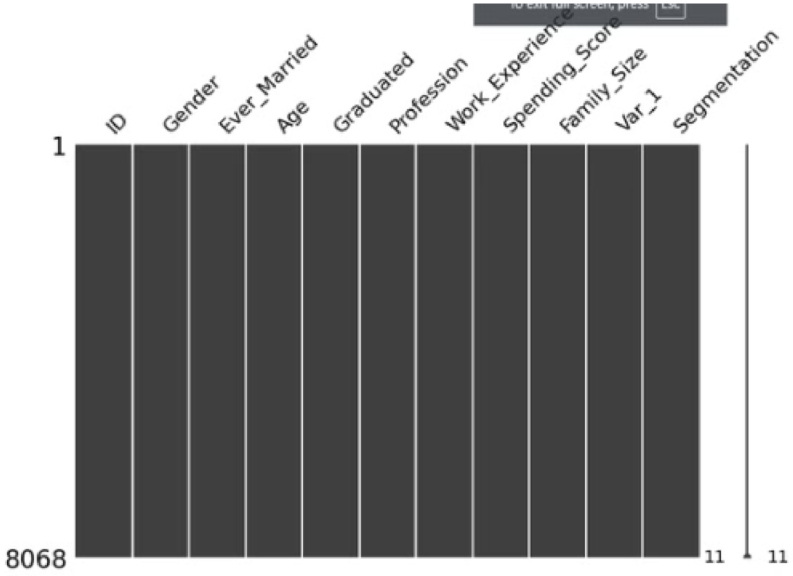
Before handling the missing values:





After Handling the missing values:





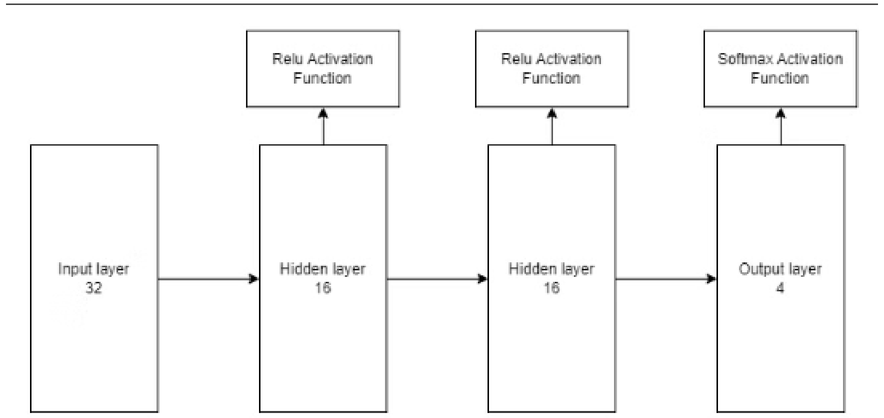
Defining Model Architecture

Model 1: Best Practices

```
model_1 = Sequential()


model_1.add(Dense(units=32, activation='relu', input_dim=X_train_transformed.shape[1]))#input_layer
model_1.add(Dense(units=16, activation='relu'))#hidden_layer-1
model_1.add(Dense(units=8, activation='relu'))#hidden-layer-2
model_1.add(Dense(units=4, activation='softmax'))#Output_layer

#compiling the model
model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```



Model 2: Keras Tuner

model using Keras Tuner for hyperparameter tuning.

```python
import keras_tuner as kt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# Define a function that builds the model
def build_model(hp):
    model_2 = Sequential()

    # Define the input layer based on the shape of X_train
    input_shape = X_train_transformed.shape[1]

    # Tune the number of hidden layers from 1 to 10
    for i in range(hp.Int('num_layers', 1, 10)):
        # Tune the number of units in each layer
        model_2.add(Dense(units=hp.Int(f'units_{i}', min_value=32, max_value=256, step=32),
                          activation='relu'))  # We are only using 'relu' as specified
```

```python
    # Output layer for multi-class classification
    model_2.add(Dense(4, activation='softmax'))  # Assuming 4 classes

    # Tune the learning rate
    learning_rate = hp.Choice('learning_rate', [1e-2, 1e-3, 1e-4])

    # Compile the model
    model_2.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
                    loss='categorical_crossentropy',
                    metrics=['accuracy'])

    return model_2
```

Training the Model

Epochs

Train for 50 epochs with a batch size of 32.

Early Stopping

Implement early stopping to prevent overfitting, monitoring validation loss.

Validation Split

Use 20% of training data as a validation set during training.

Model-1

```
[13] # Define early stopping to monitor validation loss
     early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

     # Train the model
     history = model_1.fit(
         X_train_transformed,
         y_train_transformed,
         epochs=50,
         batch_size=32,
         validation_split=0.2,
         callbacks=[early_stopping],
         verbose=1  # Display training progress
     )
```

Model-2

```
# Initialize the RandomSearch tuner
tuner = kt.RandomSearch(
        build_model,
    objective='val_accuracy',
    max_trials=5,  # Experiment with 5 trials
    executions_per_trial=3,  # Run each trial multiple times to ensure reliability
    directory='keras_tuner_dir',
    project_name='hyperparameter_tuning'
)
```

```
# Define early stopping to monitor validation loss
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

tuner.search(X_train_transformed, y_train_transformed,
             epochs=50,
             validation_split=0.2,
             callbacks=[early_stopping])
```

```
Trial 5 Complete [00h 01m 02s]
val_accuracy: 0.524989664554596

Best val_accuracy So Far: 0.531391978263855
Total elapsed time: 00h 04m 08s
```

Tuning and Training the Model With best Hyperparameters

```
[18] # Get the optimal hyperparameters
     best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]

     # Build the model with the best hyperparameters
     best_model = tuner.hypermodel.build(best_hps)

     # Training the best model on the training data
     history_best = best_model.fit(X_train_transformed, y_train_transformed,
                                   epochs=50,
                                   batch_size=32,
                                   validation_split=0.2,
                                   callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)])
```

Summary of Both the Models:

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 32)	960
dense_1 (Dense)	(None, 16)	528
dense_2 (Dense)	(None, 8)	136
dense_3 (Dense)	(None, 4)	36

Total params: 4,982 (19.46 KB)
Trainable params: 1,660 (6.48 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 3,322 (12.98 KB)

```
best_model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_8 (Dense)	(None, 160)	4,800
dense_9 (Dense)	(None, 96)	15,456
dense_10 (Dense)	(None, 224)	21,728
dense_11 (Dense)	(None, 4)	900

Total params: 128,654 (502.56 KB)
Trainable params: 42,884 (167.52 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 85,770 (335.04 KB)

Evaluating Model Performances

1

Loss Analysis

Analyze training and validation loss trends to identify overfitting.

```
253/253 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5638 - loss: 1.0109
253/253 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5707 - loss: 0.9890
Model 1 Training Accuracy: 0.5545364618301392
Model 1 Training Loss: 1.0200445652008057
Model 2 Training Accuracy: 0.5576351284980774
Model 2 Training Loss: 1.0098611116409302
```

2

Test Data Predictions

Make predictions on test data to assess model generalization

```
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Predicted lables for model_1: [0 0 0 ... 0 1 3]
Predicted Segments for model_1 :
 ['A', 'A', 'A', 'C', 'D', 'B', 'A', 'C', 'C', 'D', 'D', 'D', 'C',
Predicted lables for model_2: [0 3 0 ... 0 2 3]
Predicted Segments for model_2 :
 ['A', 'D', 'A', 'C', 'D', 'A', 'A', 'C', 'C', 'D', 'D', 'D', 'C',
```

.

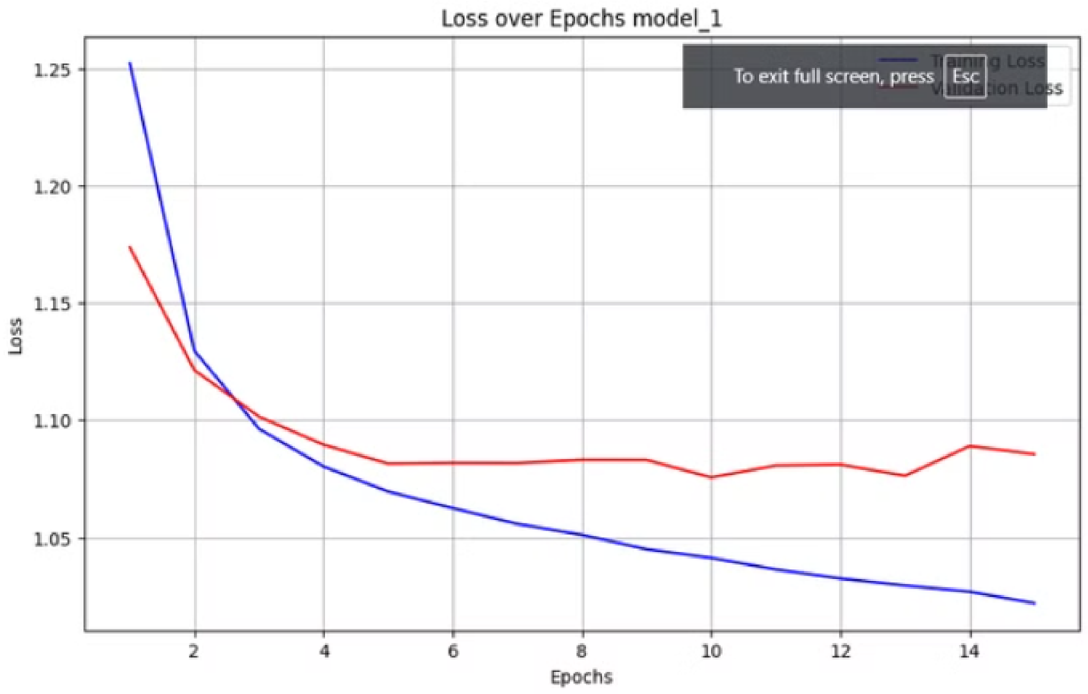
3

Performance Metrics

Evaluate using classification reports and confusion matrices for both models.

Loss over Epochs Graph for Both Models:

Looking at the divergence in between validation loss and training loss. Clearly, The models are overfitting.



Model-1

Model-2

Classification Reports:

Classification Report of best practices model (model_1):

	precision	recall	f1-score	support
0	0.45	0.57	0.51	1972
1	0.45	0.31	0.37	1858
2	0.61	0.57	0.59	1970
3	0.67	0.73	0.70	2268
accuracy			0.55	8068
macro avg	0.55	0.54	0.54	8068
weighted avg	0.55	0.55	0.55	8068

Classification Report of keras tuner model (model_2):

	precision	recall	f1-score	support
0	0.46	0.53	0.49	1972
1	0.44	0.33	0.38	1858
2	0.63	0.54	0.58	1970
3	0.65	0.79	0.71	2268
accuracy			0.56	8068
macro avg	0.55	0.55	0.54	8068
weighted avg	0.55	0.56	0.55	8068

Metric	Model 1 (Best Practices Model)	Model 2 (Keras Tuner Model)
Overall Accuracy	0.55	0.56
Weighted Avg Precision	0.55	0.55
Weighted Avg Recall	0.56	0.56
Weighted Avg F1-Score	0.55	0.55
Best Predicted Segment	Segment '3'	Segment '3'
Weakest Segment (Low F1-Score)	Segment '1'	Segment '1'
Analysis	Similar performance, slight improvement by model_2 in accuracy	Slight improvement in accuracy, but overall similar to model_1

Model 1: Best Practices Model

- Overall Accuracy: 0.55
- Precision: The model achieved a weighted average precision of 0.55, indicating that 55% of the positive predictions were correct across all classes.
- Recall: The weighted average recall is 0.56, meaning the model correctly identified 56% of the true class labels across all classes.
- F1-Score: The weighted average F1-score is 0.55, representing the balance between precision and recall.

Model 2: Keras Tuner Model

- Overall Accuracy: 0.56
- Precision: The Keras Tuner model also has a weighted average precision of 0.55, similar to the best practices model.
- Recall: The weighted average recall is slightly better at 0.56.
- F1-Score: The weighted average F1-score for the Keras Tuner model is also 0.55.

Analysis Based on Classification Reports:

- Both models performed similarly, with `model_2` showing a slight improvement in accuracy (0.56 vs. 0.55).
- Segment '3' is the best predicted by both models, achieving the highest precision, recall, and F1-score, suggesting this class is the easiest to distinguish.
- There is room for improvement, especially for class '1,' where the F1-scores are relatively low in both models.

Improving Model Accuracy

Hyperparameter Tuning

Expand hyperparameter ranges and experiment with various configurations.

Architecture Optimization

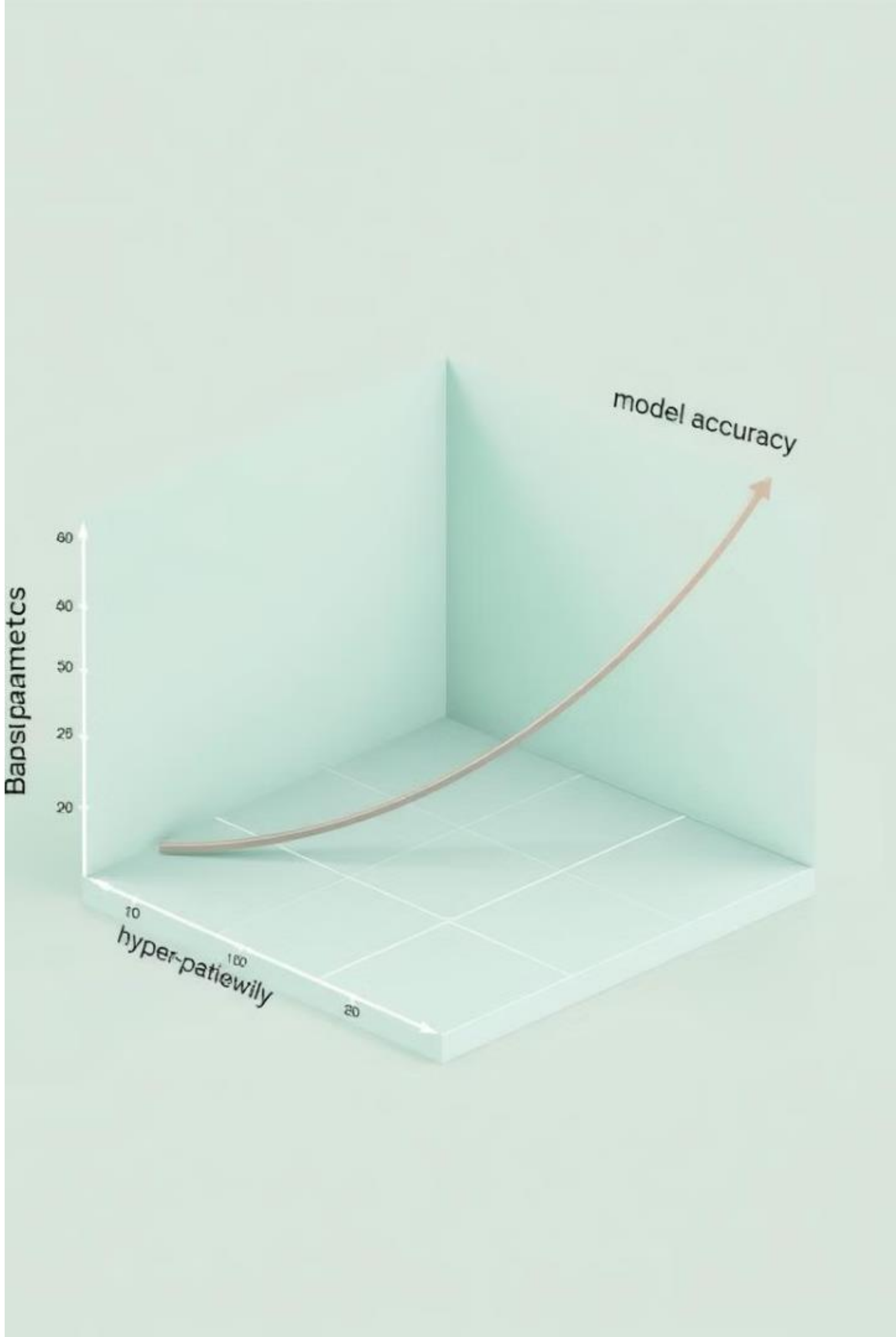
Explore deeper neural networks while considering overfitting risks.

Data Enhancement

Train on more diverse data to reduce biases and improve generalization.

Regularization

Implement L2 regularization and dropout to prevent overfitting.



Business Recommendations

Segment	Characteristics	Key Recommendations
A: Budget-Conscious	Price-sensitive	Offer discounts, highlight affordability
B: Occasional Shoppers	Infrequent buyers	Personalized reminders, seasonal offers
C: Brand-Loyal	Repeat buyers	VIP program, collect feedback
D: Premium/High-Value	High spenders	Exclusive perks, premium experiences

Thank You!