In [11]:
```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

In [12]:
```
# Step 1: Load the Data
df = pd.read_csv('./a4-data/train.csv')
df.head()
```

Out[12]:

	Feature_1	Feature_2	Feature_3	Feature_4	Feature_5	Feature_6	Feature_7	Fea
0	3289	22	19	240	93	1708	205	
1	2963	21	18	134	27	1243	206	
2	3037	185	9	127	10	6462	222	
3	3113	203	13	190	22	2125	213	
4	3128	346	9	120	36	552	203	

5 rows × 55 columns

In [13]:
```
# Step 2: Split the Data into Features (X) and Target (y)
X = df.drop('Target', axis=1)
y = df['Target'] - 1

# Scale features
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

Rationale Behind Separating Features from the Target in Machine Learning

The aim is to build a model that can make accurate predictions based on a set of input features. The target (or label) is what we want to predict. By separating the features (inputs) from the target (output), we can train our model to learn the underlying relationship between the features and the target.

In [14]:
```
# Step 3: Data Splitting
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0
```

Importance of Having Training and Test Datasets in Model Evaluation

Splitting the data into training and test datasets is crucial for evaluating the performance of our model. The training set is used to train the model, while the test set is used to evaluate how well the model generalizes to unseen data. This helps us ensure that our model is not just memorizing the training data (overfitting), but is actually learning to generalize from it.

In [15]:
```
# Build model
model = Sequential()
model.add(Dense(500, input_dim=X_train.shape[1], activation='relu'))  # Inpu
model.add(Dense(400, activation='relu'))  # Hidden layers
model.add(Dense(300, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(40, activation='relu'))
model.add(Dense(15, activation='relu'))
model.add(Dense(7, activation='softmax'))  # Output layer

# Compile model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metr

# Train model
model.fit(X_train, y_train, epochs=10, batch_size=32)

# Step 6: Model Evaluation
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Loss: {loss}, Accuracy: {accuracy}')
```

```
Epoch 1/10
11621/11621 [==============================] - 27s 2ms/step - loss: 0.5844 -
accuracy: 0.7485
Epoch 2/10
11621/11621 [==============================] - 27s 2ms/step - loss: 0.4250 -
accuracy: 0.8189
Epoch 3/10
11621/11621 [==============================] - 28s 2ms/step - loss: 0.3550 -
accuracy: 0.8512
Epoch 4/10
11621/11621 [==============================] - 27s 2ms/step - loss: 0.3144 -
accuracy: 0.8690
Epoch 5/10
11621/11621 [==============================] - 27s 2ms/step - loss: 0.2865 -
accuracy: 0.8811
Epoch 6/10
11621/11621 [==============================] - 27s 2ms/step - loss: 0.2657 -
accuracy: 0.8903
Epoch 7/10
11621/11621 [==============================] - 27s 2ms/step - loss: 0.2497 -
accuracy: 0.8977
Epoch 8/10
11621/11621 [==============================] - 26s 2ms/step - loss: 0.2365 -
accuracy: 0.9030
Epoch 9/10
11621/11621 [==============================] - 26s 2ms/step - loss: 0.2245 -
accuracy: 0.9079
Epoch 10/10
11621/11621 [==============================] - 28s 2ms/step - loss: 0.2161 -
accuracy: 0.9120
2906/2906 [==============================] - 2s 795us/step - loss: 0.2119 -
accuracy: 0.9153
Loss: 0.2119128704071045, Accuracy: 0.9152771830558777
```

Model's Architecture, Layer Types, Sizes, and Activation Functions

The architecture of a model includes the number of layers, the type of each layer (dense, convolutional, recurrent, etc.), the size of each layer (number of neurons or units), and the activation function used in each layer. The choice of these parameters depends on the specific task at hand.

Activation functions introduce non-linearity into the model, allowing it to learn more complex patterns. Common choices include ReLU (Rectified Linear Unit), sigmoid, and tanh. The choice of activation function can significantly impact the performance of the model. For example, ReLU is often used in the hidden layers of a neural network because

it helps mitigate the vanishing gradient problem.

Model Training Process, Optimizer, and Loss Function

The model training process involves feeding our training data through the model (forward pass), using a loss function to measure how well the model's predictions match the actual targets, and then adjusting the model's weights using an optimization algorithm to minimize the loss.

The choice of optimizer and loss function can significantly impact the performance and training speed of the model. Common optimizers include Adam etc, and as explained in class, performs better than general gradient descent.

The loss function depends on the task - for example, Mean Squared Error for regression tasks, and Cross-Entropy for classification tasks.

In this scenario, since the range of Target feature ranged from 1 to 7, we can say it was a multi-class classification problem. Hence, we use 'Softmax' Activation function along with 'Spare_categorical_crossentropy' as the loss function, as it is a classification problem and the Data in the first 10 features seems to be sparsely populated.

We also made use of ReLu as the activation function in the first two layers since the data is positive and does not have any negative values. Therefore, ReLu seems as the right choice!

Even the performance of the model is decent, with an Accuracy of 92%. This shows our choice of Activation, Loss and Optimizer functions is fair.

In [18]:
```
# Step 7: Make Predictions
test_data = pd.read_csv('./a4-data/test.csv')
test_data.head()
```

Out[18]:

	Feature_1	Feature_2	Feature_3	Feature_4	Feature_5	Feature_6	Feature_7	Fea
0	3351	206	27	726	124	3813	192	
1	2732	129	7	212	1	1082	231	
2	2572	24	9	201	25	957	216	
3	2824	69	13	417	39	3223	233	
4	2529	84	5	120	9	1092	227	

5 rows × 54 columns

In [19]:
```
X_test_scaled = scaler.transform(test_data)

# Predict class labels for the scaled test features
predictions = model.predict(X_test_scaled)

# Get the predicted class labels
predicted_labels = predictions.argmax(axis=1)

# Add 1 to the predicted labels to match your original target labels
predicted_labels += 1

print(predicted_labels)
```

```
3632/3632 [==============================] - 2s 652us/step
[1 2 2 ... 2 2 7]
```

Model Evaluation and Performance Metrics

Model evaluation involves using our test set to assess how well our model generalizes to unseen data. Common metrics include accuracy for classification tasks, and Mean Squared Error for regression tasks. It's important to choose a metric that aligns with our specific goals for the model.

Making Predictions with a Trained Model

Once the model is trained, we can use it to make predictions on new, unseen data. This involves feeding the new data through the model (forward pass), and then interpreting the output of the model as our prediction. The exact process can vary depending on the type of model and the specific task.

Here, the 'Target' feature is being predicted by the model created, which is missing in

the 'test.csv' file and creating a submission file below to update the predictions.

In [20]:
```
# Step 8: Generate Submission File
submission = pd.DataFrame({'ID': test_data.index, 'Target': predicted_labels
submission.to_csv('submission.csv', index=False)
```

In [ ]:
```

```