Model1:
For model1, we start with a sequential model to layer operations linearly.
2. Use convolutional layers with ReLU activation for initial feature extraction.
3. Apply batch normalization to stabilize activations across layers.
4. Reduce feature map sizes with max pooling to emphasize important features.
5. Integrate dropout to mitigate overfitting by randomly omitting units.
6. Flatten convolutional output to a 1D vector for dense layer processing.
7. Add a dense layer with many neurons for complex pattern recognition.
8. Compile the model focusing on accuracy, using Adam optimizer and categorical loss.

We used 40k as train data and the first 10k(the true label) as test data. The accuracy was 0.472 and runtime was 821.61s. We decreased the batch size and also changed the learning rate which made the runtime longer than model2.

Model2: Model2 contains 3 parts.
Part one: 1.Implement `ReduceLROnPlateau` callback to reduce learning rate when validation loss stops improving.
2. Use `ModelCheckpoint` to save the model with the best validation accuracy during training.
3. Construct a feature extractor with convolutional, batch normalization, and max pooling layers.
4. Assemble the main model, integrating feature extraction with processing of noisy label inputs.
5. Compile the model, specifying the optimizer, loss function, and metrics for performance evaluation.
6. Employ K-Fold cross-validation to train the model systematically on different data subsets.
7. Fit the model using both clean images and noisy labels, applying specified callbacks.

Part two: 1. Predict probabilities for noisy images using both image features and noisy labels.
2. Convert prediction probabilities to class labels by selecting the class with highest probability.
3. Apply a confidence threshold of 0.9 to filter predictions with high certainty.
4. Identify indices of predictions exceeding the confidence threshold for reliable pseudo-label generation.
5. Generate pseudo labels for high-confidence predictions to use in further model training/enhancement.

Part three:1. Apply two convolutional layers with 24 filters each for initial feature extraction.
2. Use max pooling after convolutional layers to reduce dimensionality and emphasize features.
3. Add more convolutional layers with increased filters (48) for deeper feature extraction.
4. Flatten the output from convolutional layers to prepare for LSTM processing.
5. Reshape flattened data for compatibility with LSTM, treating image features as sequences.
6. Integrate an LSTM layer with 64 units to analyze image features as temporal sequences.
7. Include dense layers for high-level reasoning and a dropout layer to prevent overfitting.
8. Output layer utilizes softmax activation to classify images into one of ten categories.

9. Compile the model with Adam optimizer and sparse categorical crossentropy for multi-class classification.
10. Train the model using combined clean and pseudo-labeled images for robust learning.
11. Apply K-Fold cross-validation to ensure model generalizes well across different data splits.
12. Utilize callbacks for dynamic learning rate adjustments and to save the best-performing model.

We designed model2 in versions. One is without k-fold and with LSTM layers. The train accuracy was 0.8424 and runtime was 73.98s. One is with k-fold and with LSTM layers. The train accuracy was 0.9184 and runtime was 382.77s. We used 50k to train the model and 0.2 validation. We also used the 49k data to train the model and rest 1k to test(evaluation part). The weight average accuracy for each label was 0.6148.

```
[61]: classification_report(clean_labels[:1000]  ,model2_label,zero_division=0, output_dict=True)

[61]: {'0': {'precision': 0.7088607594936709,
         'recall': 0.5490196078431373,
         'f1-score': 0.6187845303867404,
         'support': 102.0},
        '1': {'precision': 0.8043478260869565,
         'recall': 0.6607142857142857,
         'f1-score': 0.7254901960784315,
         'support': 112.0},
        '2': {'precision': 0.55,
         'recall': 0.4444444444444444,
         'f1-score': 0.4916201117318436,
         'support': 99.0},
        '3': {'precision': 0.3815789473684211,
         'recall': 0.31521739130434784,
         'f1-score': 0.34523809523809523,
         'support': 92.0},
        '4': {'precision': 0.5436893203883495,
         'recall': 0.5656565656565656,
         'f1-score': 0.5544554455445545,
         'support': 99.0},
        '5': {'precision': 0.4935064935064935,
         'recall': 0.4470588235294118,
         'f1-score': 0.4691358024691358,
         'support': 85.0},
        '6': {'precision': 0.5923076923076923,
         'recall': 0.719626168224299,
         'f1-score': 0.649789029535865,
         'support': 107.0},
        '7': {'precision': 0.693069306930693,
         'recall': 0.6862745098039216,
         'f1-score': 0.6896551724137931,
         'support': 102.0},
        '8': {'precision': 0.7073170731707317,
         'recall': 0.8787878787878788,
         'f1-score': 0.7837837837837839,
         'support': 99.0},
        '9': {'precision': 0.6115107913669064,
         'recall': 0.8252427184466019,
         'f1-score': 0.7024793388429752,
         'support': 103.0},
        'accuracy': 0.616,
        'macro avg': {'precision': 0.6086188210619914,
         'recall': 0.6092042393754894,
         'f1-score': 0.6030431506025218,
         'support': 1000.0},
        'weighted avg': {'precision': 0.6147993059530343,
         'recall': 0.616,
         'f1-score': 0.609393072444342,
         'support': 1000.0}}
```