PROBLEM FORMULATION

For most image classification problems, the best approach is to develop a Convolutional Neural Network (CNN) model wherein the convolution layers perform the task of feature extraction from the images, and the neural network uses the extracted features for the image classification task. A CNN model is preferred over a Deep Neural Network model as it can learn the filters on its own, is not sensitive to the object's location in an image and involves much lesser weight computations.

As the dataset contains many classes, we use multiple convolutional layers, each with a different number of filters for better feature identification and hence, better classification accuracy. We also use other parameters like stride and padding to include corner pixels of the images in the feature extraction phase, thereby improving the performance of the convolution layers in the model. A maxpooling operation performed after each convolution layer helps in reducing the number of computations in the feature extraction phase. Lastly, the dropout parameter helps with regularization in the image classification Neural Network.

PREPROCESSING

Some preprocessing was already done on the train and test dataset provided for the problem. All the images were downsized to 28 x 28 pixels. This was done so that the model trains on images of the same size. Another preprocessing step was converting images to grayscale to reduce the channels from 3 (RGB) to 1. As the image data for this problem is primarily black and white hand sketches independent of color, converting the images to grayscale will reduce the model training time.

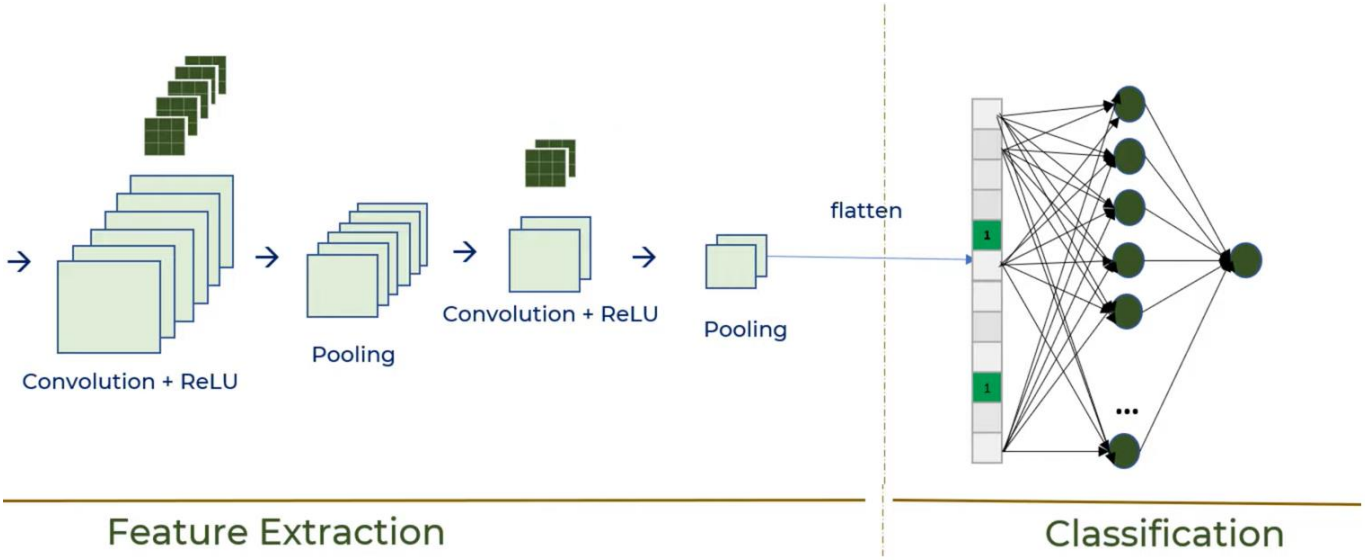
Apart from the already applied preprocessing, another preprocessing step taken was the normalization of train and test data by dividing all the values by 255, as the pixel values range from 0-255. Post normalization, the numbers become smaller, and computation becomes easier. Lastly, the train and test data were reshaped to include the image size in the dimensions of the dataset. This was done to avoid issues converting numpy array to tensor during the model training phase.

```
#Normalizing train and test data
train_data = train_data / 255
test_data = test_data / 255
```

```
#Reshaping train and test data to include image size
train_data = np.array([np.array(val) for val in train_data])
train_target = np.array([np.array(val) for val in train_target])
test_data = np.array([np.array(val) for val in test_data])
```

MODEL DESIGN

The model developed here is a convolutional neural network with 2 layers of convolution in the feature extraction phase and 2 dense layers in the image classification phase. The central assumption in this problem is that all the images in the training dataset have been correctly classified, as the accuracy of the test data is highly dependent on it.



(*Illustration of implemented model)*

Hyperparameter tuning using the random search functionality of keras tuner was done and the following parameters were used to create the final CNN model.

```
tuner = RandomSearch(build_model, objective='val_accuracy', max_trials = 25)

tuner.search(train_data, train_target, epochs=3, validation_data = (train_data, train_target))
```

```
INFO:tensorflow:Reloading Oracle from existing project .\untitled_project\oracle.json
INFO:tensorflow:Reloading Tuner from .\untitled_project\tuner0.json

Search: Running Trial #21

Value             |Best Value So Far |Hyperparameter
32                |128               |conv_1_filter
5                 |5                 |conv_1_kernel
32                |48                |conv_2_filter
4                 |2                 |conv_2_kernel
250               |400               |dense_1_units
0.1               |0.001             |learning_rate

Epoch 1/3
1597/3125 [==============>..............] - ETA: 10s - loss: 5.2054 - accuracy: 0.0099
```

1st convolution layer

Filters = 128, Stride = 5, Padding = Same, Activation = ReLU

2nd convolution layer

Filters = 48, Stride = 2, Padding = Same, Activation = ReLU

The 1st convolution layer has more filters and a bigger stride to detect many basic features that make up the different parts of the images. The 2nd convolution layer has lesser filters and a smaller stride so that it can detect complex features within images that are a combination of several basic features.

Both layers use the 'same' padding function that allows the feature map obtained after the convolution operation to have the exact dimensions as that of the original image. Both layers also use the rectified linear unit (ReLU) as their activation function.

The 1st convolution layer is followed by a maxpooling operation with a window size of 3x3, and a maxpooling operation with a window size of 2x2 follows the 2nd convolution layer. The

maxpooling operation reduces the size of the feature map, which helps the number of computations involved in the feature extraction phase.

The convolution layers are followed by a flatten operation to convert the 2D/3D feature map to a 1D array. A dropout value of 0.15 is specified, which helps with regularization. The second to last and the last dense layers in the image classification phase have 400 and 100 neurons with activation functions of 'relu' and 'softmax', respectively. The last layer has 100 neurons, as there are 100 classes in the dataset. The final output layer uses the softmax function, as this is a multilabel classification problem.

The 'adam' optimizer was used to compile the model with a learning rate of 0.001. 'Adam' optimizer generally provides better results than other optimization algorithms and has faster computation times with fewer parameters to tune. A lower learning rate allows the model to learn the problem better. The sparse categorical cross-entropy loss function was used as the model's output is a single number representing the class. 10 epochs were used while building the model using the training set.

```
cnn = models.Sequential([
    layers.Conv2D(filters = 128, kernel_size = 5, padding = 'same', activation = 'relu', input_shape = (28,28,1)),
    layers.MaxPooling2D((3,3)),
    layers.Conv2D(filters = 48, kernel_size = 2, padding = 'same', activation = 'relu', input_shape = (28,28,1)),
    layers.MaxPooling2D((2,2)),
    layers.Dropout(0.15),
    layers.Flatten(),
    layers.Dense(400, activation = 'relu'),
    layers.Dense(100, activation = 'softmax')])

optimizer = keras.optimizers.Adam(learning_rate = 0.001)

cnn.compile(optimizer = optimizer,
            loss = 'sparse_categorical_crossentropy',
            metrics = ['accuracy'])
```

RESULTS

The model was trained on the provided training set. 10% of the training set was kept as a validation set to verify accuracy on unknown examples. 10 epochs were used for training.

```
cnn.fit(x_train, y_train, epochs = 10)
```

```
Epoch 1/10
14063/14063 [==============================] - 281s 20ms/step - loss: 1.7368 - accuracy: 0.5625
Epoch 2/10
14063/14063 [==============================] - 272s 19ms/step - loss: 1.3613 - accuracy: 0.6473
Epoch 3/10
14063/14063 [==============================] - 267s 19ms/step - loss: 1.2702 - accuracy: 0.6683
Epoch 4/10
14063/14063 [==============================] - 282s 20ms/step - loss: 1.2236 - accuracy: 0.6789
Epoch 5/10
14063/14063 [==============================] - 281s 20ms/step - loss: 1.1907 - accuracy: 0.6859
Epoch 6/10
14063/14063 [==============================] - 306s 22ms/step - loss: 1.1671 - accuracy: 0.6918
Epoch 7/10
14063/14063 [==============================] - 286s 20ms/step - loss: 1.1510 - accuracy: 0.6950
Epoch 8/10
14063/14063 [==============================] - 260s 18ms/step - loss: 1.1372 - accuracy: 0.6982
Epoch 9/10
14063/14063 [==============================] - 277s 20ms/step - loss: 1.1251 - accuracy: 0.7001
Epoch 10/10
14063/14063 [==============================] - 285s 20ms/step - loss: 1.1165 - accuracy: 0.7025
```

The model achieves an accuracy of 70.25% on the training set with a loss of 1.11. Since the loss is low and the training accuracy is high, the model has performed well on the given image classification problem.

```
cnn.evaluate(x_test, y_test)
```

```
1563/1563 [==============================] - 8s 5ms/step - loss: 1.1731 - accuracy: 0.6970

[1.1731359958648682, 0.6970199942588806]
```

The model achieves an accuracy of 69.7% on the validation set. As the validation accuracy is almost the same as the training accuracy, we can say that there is no overfitting and the model is able to generalize well.

IMPROVING ACCURACY

Hyperparameter tuning is a common approach used for improving the accuracy of any machine learning model. We can use the same approach here to enhance the accuracy by testing over different values of filters and kernel in the convolution layers of the feature extraction phase and testing over different values for dropout and neurons in the image classification phase. Testing different learning rates in the optimizer algorithm can also help improve the model's accuracy.

Using more epochs while building the model on the training dataset can also help in achieving higher accuracy on the test dataset.

Apart from these 2 approaches, we can also use transfer learning from different pre-trained models for image classification problems. Some of the popular pre-trained models are VGG-16, ResNet50, YOLO and EfficientNet. Multi-model approaches like combining convolution layers with random forests for image classification can also be used to improve this problem's accuracy.