In [1]:
```
# For COLAB use only

from google.colab import drive
drive.mount('/content/drive')

import zipfile
import os

zip_ref = zipfile.ZipFile('/content/drive/MyDrive/Colab Notebooks/Mini Project/None
zip_ref.extractall('/content') # Extracts the files into the /tmp folder
zip_ref1 = zipfile.ZipFile('/content/drive/MyDrive/Colab Notebooks/Mini Project/tes
zip_ref1.extractall('/content') # Extracts the files into the /tmp folder
zip_ref.close()
zip_ref1.close()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call driv
e.mount("/content/drive", force_remount=True).
```

In [2]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
```

In [3]:
```
# define the image size and batch size
img_size = (40, 40)
batch_size = 32

# Path to your datasets folder
data_dir = "/content/None_set"
test_dir = "/content/test_set"
```

In [4]:
```
train_ds = tf.keras.preprocessing.image_dataset_from_directory( # training set
    data_dir,
    labels= "inferred",
    color_mode='grayscale',
    validation_split=0.2,
    subset='training',
    seed=123,
    image_size=img_size,
    batch_size=batch_size)

val_ds = tf.keras.preprocessing.image_dataset_from_directory( # testing set
    data_dir,
    labels= "inferred",
    color_mode='grayscale',
    validation_split=0.2,
    subset='validation',
    seed=123,
    image_size=img_size,
    batch_size=batch_size)
```

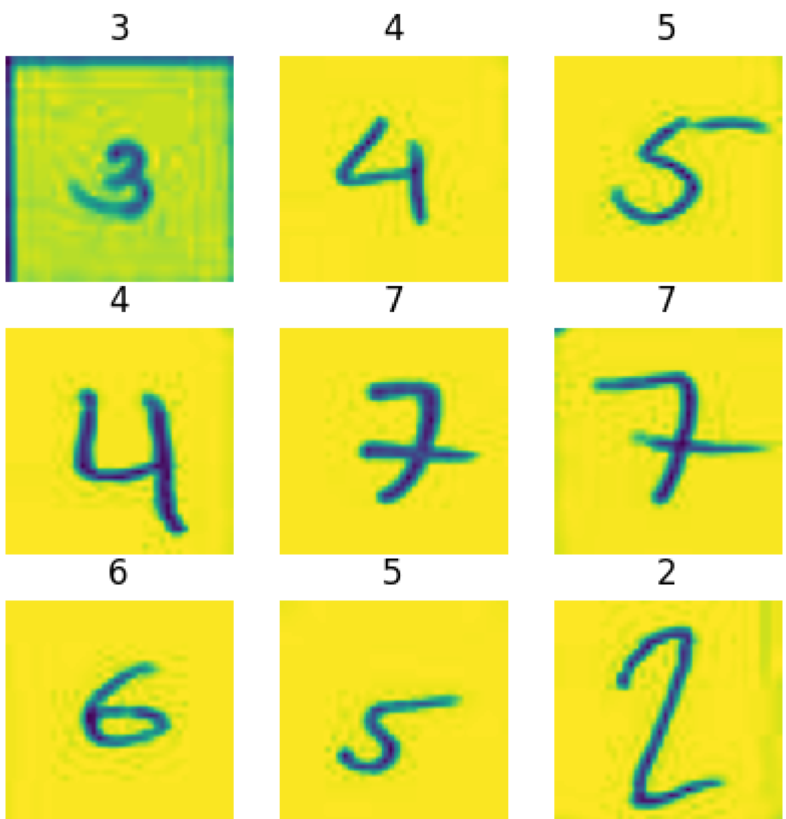
```
Found 21275 files belonging to 9 classes.
Using 17020 files for training.
Found 21275 files belonging to 9 classes.
Using 4255 files for validation.
```

In [5]:
```python
# inspect the class names and number of classes
class_names = train_ds.class_names
num_classes = len(class_names)
print('Class names:', class_names)
print('Number of classes:', num_classes)
```

```
Class names: ['0', '1', '2', '3', '4', '5', '6', '7', 'None']
Number of classes: 9
```

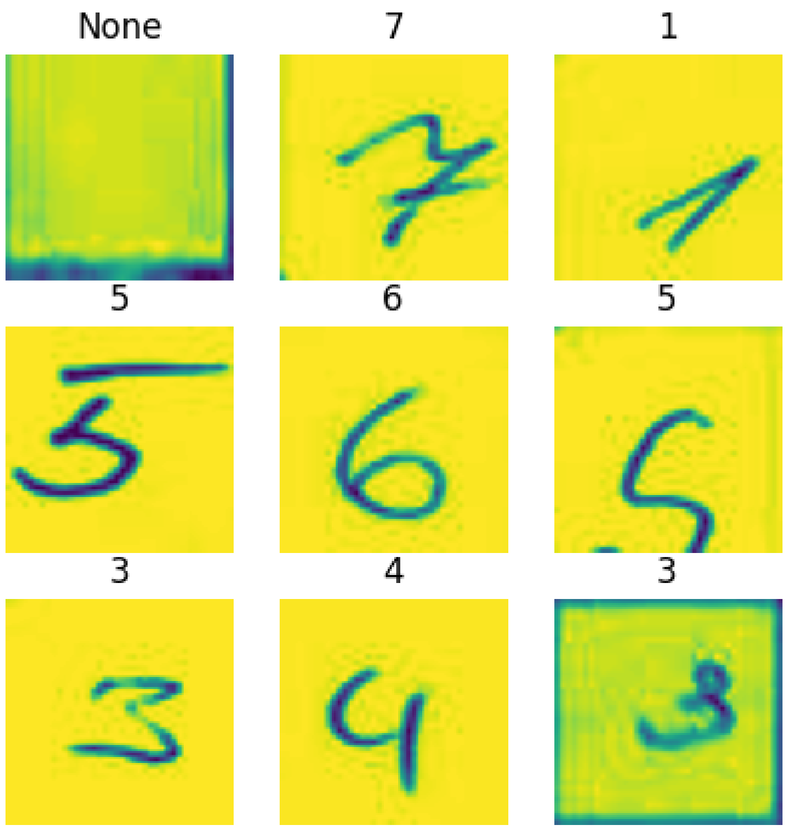
In [6]:
```python
print("Training Set")
plt.figure(figsize=(5, 5))
for images, labels in train_ds.take(2):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

```
Training Set
```



In [7]:
```python
print("Validation Set")
plt.figure(figsize=(5, 5))
for images, labels in val_ds.take(2):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

```
Validation Set
```



In [8]:
```
# create the neural network
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(16, 3, activation='relu', input_shape=(40, 40, 1)),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, 4, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])

# The neural network architecture consists of sequential layers, where the output o

# The first layer is a convolutional layer with 16 filters of size 3x3, followed by
# The input shape of the layer is (40, 40, 1), which means that the layer expects i

# The second layer is a max pooling layer that reduces the spatial dimensions of th

# The third layer is another convolutional layer with 32 filters of size 3x3, follo

# The fourth layer is another max pooling layer.

# The fifth layer is a flatten layer that flattens the output from the previous lay

# The sixth layer is a dense layer with 64 units and a ReLU activation function.

# The seventh and final layer is a dense layer with num_classes units and a softmax
```

In [9]:
```
model.compile(optimizer='adam',
```

```
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy']) # compiling the model
```

In [10]:
```
history = model.fit(train_ds, validation_data=val_ds, epochs=30) # Fitting the mode
```

```
Epoch 1/30
```

```
/usr/local/lib/python3.10/dist-packages/keras/backend.py:5612: UserWarning: "`spar
se_categorical_crossentropy` received `from_logits=True`, but the `output` argumen
t was produced by a Softmax activation and thus does not represent logits. Was thi
s intended?
  output, from_logits = _get_logits(
```

```
532/532 [==============================] - 9s 11ms/step - loss: 1.5254 - accuracy:
0.6226 - val_loss: 0.5211 - val_accuracy: 0.8468
Epoch 2/30
532/532 [==============================] - 4s 7ms/step - loss: 0.3725 - accuracy:
0.8882 - val_loss: 0.3750 - val_accuracy: 0.8893
Epoch 3/30
532/532 [==============================] - 5s 10ms/step - loss: 0.2348 - accuracy:
0.9294 - val_loss: 0.2778 - val_accuracy: 0.9231
Epoch 4/30
532/532 [==============================] - 6s 12ms/step - loss: 0.1562 - accuracy:
0.9519 - val_loss: 0.2484 - val_accuracy: 0.9405
Epoch 5/30
532/532 [==============================] - 3s 6ms/step - loss: 0.1114 - accuracy:
0.9665 - val_loss: 0.3154 - val_accuracy: 0.9222
Epoch 6/30
532/532 [==============================] - 3s 6ms/step - loss: 0.0895 - accuracy:
0.9733 - val_loss: 0.2423 - val_accuracy: 0.9387
Epoch 7/30
532/532 [==============================] - 4s 8ms/step - loss: 0.0790 - accuracy:
0.9763 - val_loss: 0.2359 - val_accuracy: 0.9483
Epoch 8/30
532/532 [==============================] - 4s 8ms/step - loss: 0.0765 - accuracy:
0.9771 - val_loss: 0.2797 - val_accuracy: 0.9342
Epoch 9/30
532/532 [==============================] - 3s 6ms/step - loss: 0.0870 - accuracy:
0.9747 - val_loss: 0.2412 - val_accuracy: 0.9509
Epoch 10/30
532/532 [==============================] - 3s 6ms/step - loss: 0.0539 - accuracy:
0.9845 - val_loss: 0.2579 - val_accuracy: 0.9546
Epoch 11/30
532/532 [==============================] - 5s 10ms/step - loss: 0.0384 - accuracy:
0.9877 - val_loss: 0.2189 - val_accuracy: 0.9629
Epoch 12/30
532/532 [==============================] - 3s 6ms/step - loss: 0.0538 - accuracy:
0.9845 - val_loss: 0.2534 - val_accuracy: 0.9553
Epoch 13/30
532/532 [==============================] - 3s 6ms/step - loss: 0.0592 - accuracy:
0.9823 - val_loss: 0.3846 - val_accuracy: 0.9356
Epoch 14/30
532/532 [==============================] - 4s 8ms/step - loss: 0.0543 - accuracy:
0.9860 - val_loss: 0.2755 - val_accuracy: 0.9577
Epoch 15/30
532/532 [==============================] - 5s 9ms/step - loss: 0.0395 - accuracy:
0.9888 - val_loss: 0.2572 - val_accuracy: 0.9626
Epoch 16/30
532/532 [==============================] - 3s 6ms/step - loss: 0.0191 - accuracy:
0.9938 - val_loss: 0.2631 - val_accuracy: 0.9586
Epoch 17/30
532/532 [==============================] - 3s 7ms/step - loss: 0.0951 - accuracy:
0.9753 - val_loss: 0.3539 - val_accuracy: 0.9518
Epoch 18/30
532/532 [==============================] - 4s 8ms/step - loss: 0.0414 - accuracy:
0.9892 - val_loss: 0.3019 - val_accuracy: 0.9622
Epoch 19/30
532/532 [==============================] - 4s 8ms/step - loss: 0.0343 - accuracy:
0.9894 - val_loss: 0.2922 - val_accuracy: 0.9600
```

```
Epoch 20/30
532/532 [==============================] - 3s 6ms/step - loss: 0.0346 - accuracy: 0.9902 - val_loss: 0.2837 - val_accuracy: 0.9664
Epoch 21/30
532/532 [==============================] - 3s 6ms/step - loss: 0.0243 - accuracy: 0.9937 - val_loss: 0.2823 - val_accuracy: 0.9638
Epoch 22/30
532/532 [==============================] - 4s 8ms/step - loss: 0.0138 - accuracy: 0.9958 - val_loss: 0.3022 - val_accuracy: 0.9633
Epoch 23/30
532/532 [==============================] - 3s 6ms/step - loss: 0.0246 - accuracy: 0.9939 - val_loss: 0.3790 - val_accuracy: 0.9485
Epoch 24/30
532/532 [==============================] - 3s 6ms/step - loss: 0.0664 - accuracy: 0.9839 - val_loss: 0.3578 - val_accuracy: 0.9633
Epoch 25/30
532/532 [==============================] - 5s 10ms/step - loss: 0.0275 - accuracy: 0.9924 - val_loss: 0.3396 - val_accuracy: 0.9615
Epoch 26/30
532/532 [==============================] - 3s 6ms/step - loss: 0.0374 - accuracy: 0.9914 - val_loss: 0.4336 - val_accuracy: 0.9551
Epoch 27/30
532/532 [==============================] - 3s 6ms/step - loss: 0.0217 - accuracy: 0.9935 - val_loss: 0.3638 - val_accuracy: 0.9657
Epoch 28/30
532/532 [==============================] - 4s 7ms/step - loss: 0.0335 - accuracy: 0.9925 - val_loss: 0.4277 - val_accuracy: 0.9608
Epoch 29/30
532/532 [==============================] - 6s 11ms/step - loss: 0.0353 - accuracy: 0.9913 - val_loss: 0.3335 - val_accuracy: 0.9657
Epoch 30/30
532/532 [==============================] - 3s 6ms/step - loss: 0.0172 - accuracy: 0.9954 - val_loss: 0.3252 - val_accuracy: 0.9680
```

In [11]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 38, 38, 16)        160

 max_pooling2d (MaxPooling2D  (None, 19, 19, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 16, 16, 32)        8224

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 32)         0
 2D)

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 64)                131136

 dense_1 (Dense)             (None, 9)                 585

=================================================================
Total params: 140,105
Trainable params: 140,105
Non-trainable params: 0
_________________________________________________________________
```

In [12]:

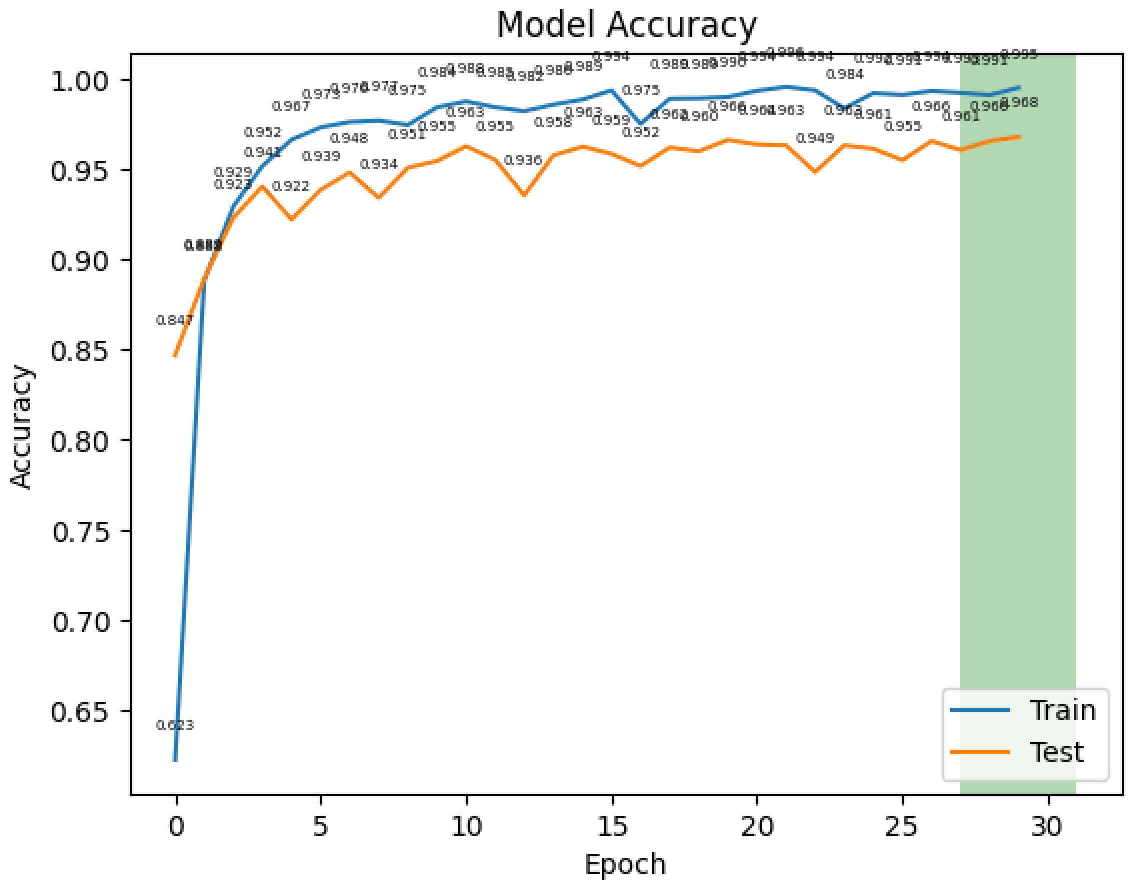
```
_, train_acc = model.evaluate(train_ds, verbose=0)
_, val_acc = model.evaluate(val_ds, verbose=0)

print(f'Training Accuracy: {(train_acc * 100):.2f}%\nValidation Accuracy: {(val_acc
```

```
Training Accuracy: 99.79%
Validation Accuracy: 96.80%
```

In [13]:

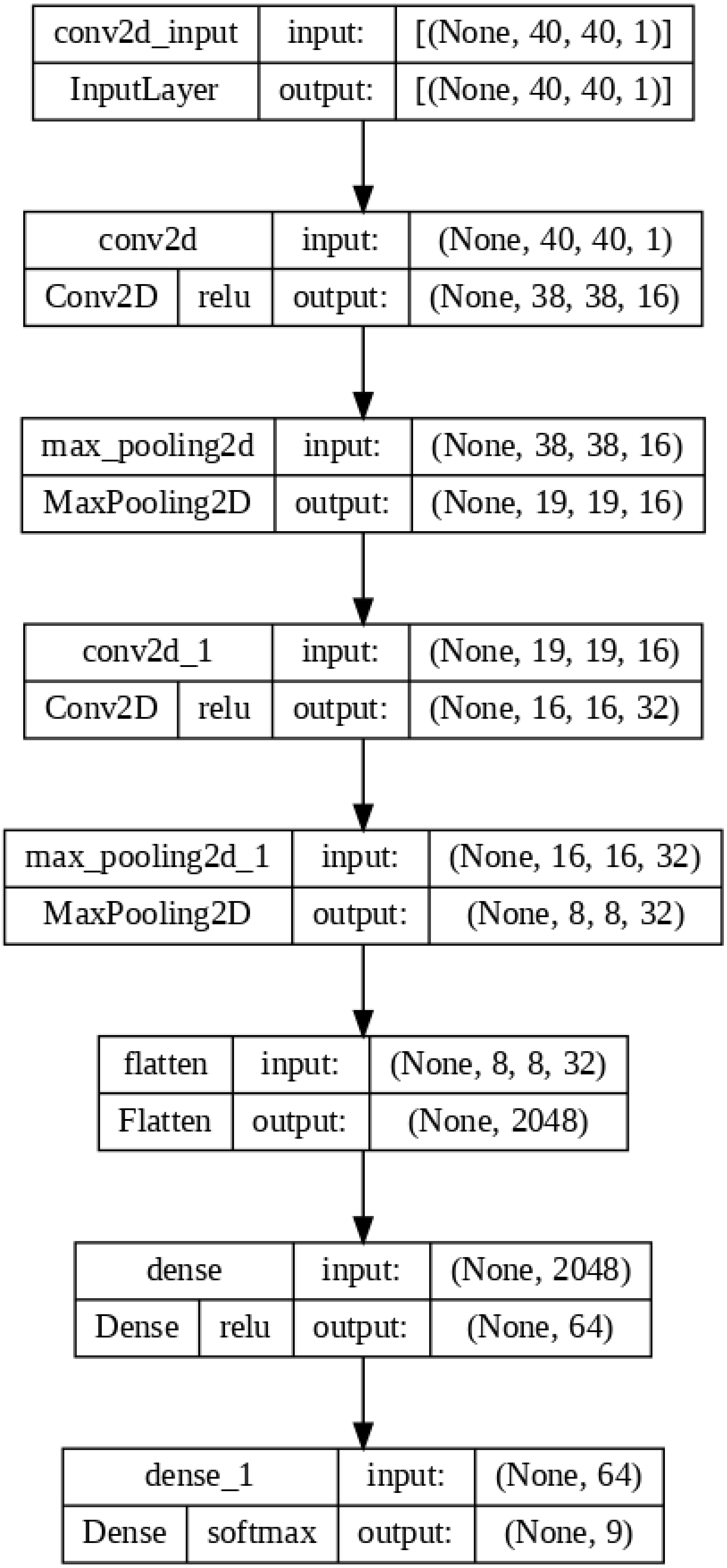
```
import matplotlib.pyplot as plt
import numpy as np

# Plot the accuracy over epochs
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='lower right')

# Add annotations to display y-axis values at the top of each point
for i, acc in enumerate(history.history['accuracy']):
    plt.annotate('{:.3f}'.format(acc), (i, acc), xytext=(0, 10), textcoords='offset
for i, val_acc in enumerate(history.history['val_accuracy']):
    plt.annotate('{:.3f}'.format(val_acc), (i, val_acc), xytext=(0, 10), textcoords

# Find epoch with highest validation accuracy
best_epoch = np.argmax(history.history['val_accuracy'])

# Highlight region around best epoch
plt.axvspan(best_epoch - 2, best_epoch + 2, facecolor='green', alpha=0.3)
```

```
plt.show()
```



In [14]:
```
# For visualizing the model

from keras.utils.vis_utils import plot_model
plot_model(model,
           to_file='model.png',
           show_shapes=True,
           show_dtype=False,
           show_layer_names=True,
           rankdir='TB',
           expand_nested=False,
           dpi=100,
           layer_range=None,
           show_layer_activations=True
          )
```

Out[14]:



In [15]:
```
model.save('MP_Latest_Model.h5')
```

Testing the model using the `model` variable in the above code, and displaying metrics like

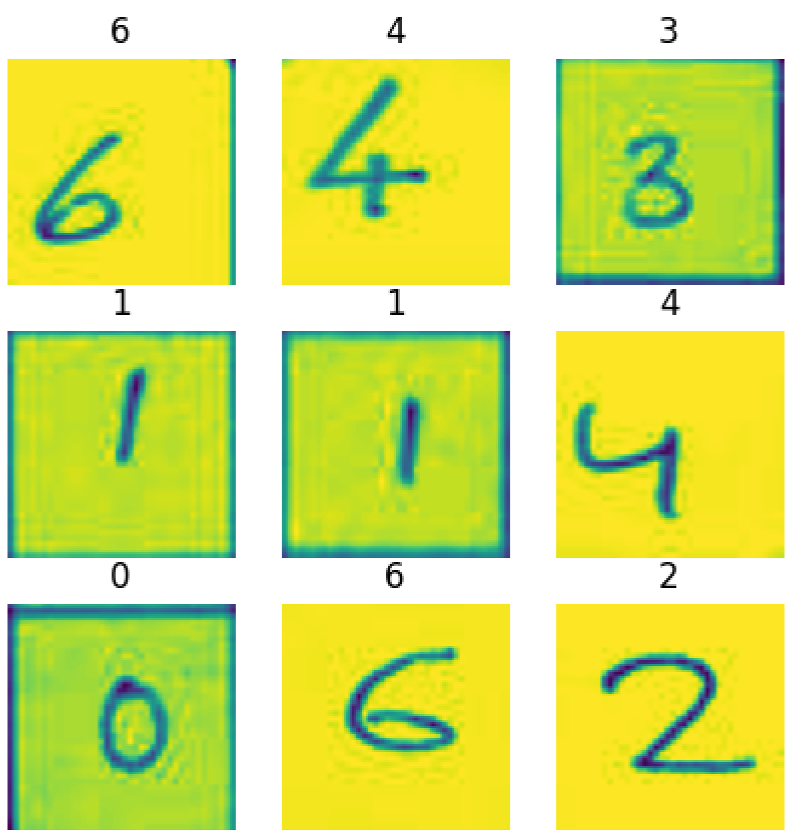
`Confusion Matrix` and `Classification Report` of the same. Thus training, validation & testing in the same code

In [16]:
```
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    test_dir,
    labels="inferred",
    color_mode='grayscale',
    validation_split=0.99,
    subset='validation',  # Use 'validation' as the subset for the test dataset
    seed=123,
    image_size=img_size,
    batch_size=batch_size
)
```

```
Found 1800 files belonging to 9 classes.
Using 1782 files for validation.
```

In [17]:
```
import matplotlib.pyplot as plt

print("Testing Set")
plt.figure(figsize=(5, 5))
for images, labels in test_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

```
Testing Set
```



In [18]:
```
# Change path to dataset here ↓
```

```
img_dirs = [f"/content/test_set/{i}" for i in range(9)]
img_dir_0, img_dir_1, img_dir_2, img_dir_3, img_dir_4, img_dir_5, img_dir_6, img_di

# Declaring images in each folder and their ground truth as a list
image_list_0, image_list_1, image_list_2, image_list_3, image_list_4, image_list_5,

gnd_trt_0, gnd_trt_1, gnd_trt_2, gnd_trt_3, gnd_trt_4, gnd_trt_5, gnd_trt_6, gnd_tr
```

In [19]:
```
def prediction(image_folder, image_list):
    pred = []

    # Loop through all the images in the folder and classify them
    for image_name in image_list:

        test_image = tf.keras.preprocessing.image.load_img(os.path.join(image_folde
        test_image = np.expand_dims(test_image, axis=0) # Expand the dimensions of

        result = model.predict(test_image)

        predicted_value = np.argmax(result[0])

        pred.append(predicted_value)

#          pred = ['None' if x == 8 else x for x in pred]

    return pred
```

In [20]:
```
def accuracy(gnd_trt, image_list, preds): # Accuracy calculation
    true_labels = [gnd_trt] * len(image_list)
    correct_predictions = sum(preds[i] == true_labels[i] for i in range(len(image_l
    accuracy_percent = (correct_predictions / len(true_labels)) * 100
    return accuracy_percent
```

In [21]:
```
preds_0 = prediction(img_dir_0, image_list_0)
preds_1 = prediction(img_dir_1, image_list_1)
preds_2 = prediction(img_dir_2, image_list_2)
preds_3 = prediction(img_dir_3, image_list_3)
preds_4 = prediction(img_dir_4, image_list_4)
preds_5 = prediction(img_dir_5, image_list_5)
preds_6 = prediction(img_dir_6, image_list_6)
preds_7 = prediction(img_dir_7, image_list_7)
preds_8 = prediction(img_dir_8, image_list_8)
```

```
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
```

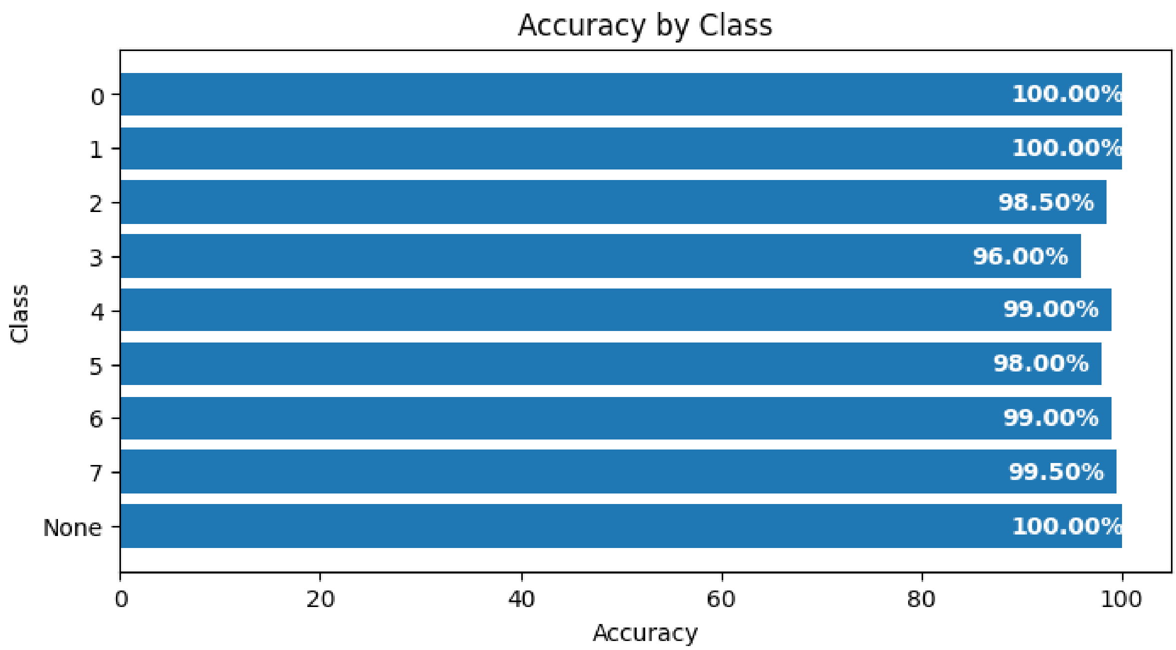
In [22]:

```
for i in range(9): # Hard coded, as we know the no of classes
    print(f"Total images in {i} : {len(locals()[f'image_list_{i}'])}")
    # locals() method returns a dictionary with all the local variables for the cur
```

```
Total images in 0 : 200
Total images in 1 : 200
Total images in 2 : 200
Total images in 3 : 200
Total images in 4 : 200
Total images in 5 : 200
Total images in 6 : 200
Total images in 7 : 200
Total images in 8 : 200
```

In [23]:

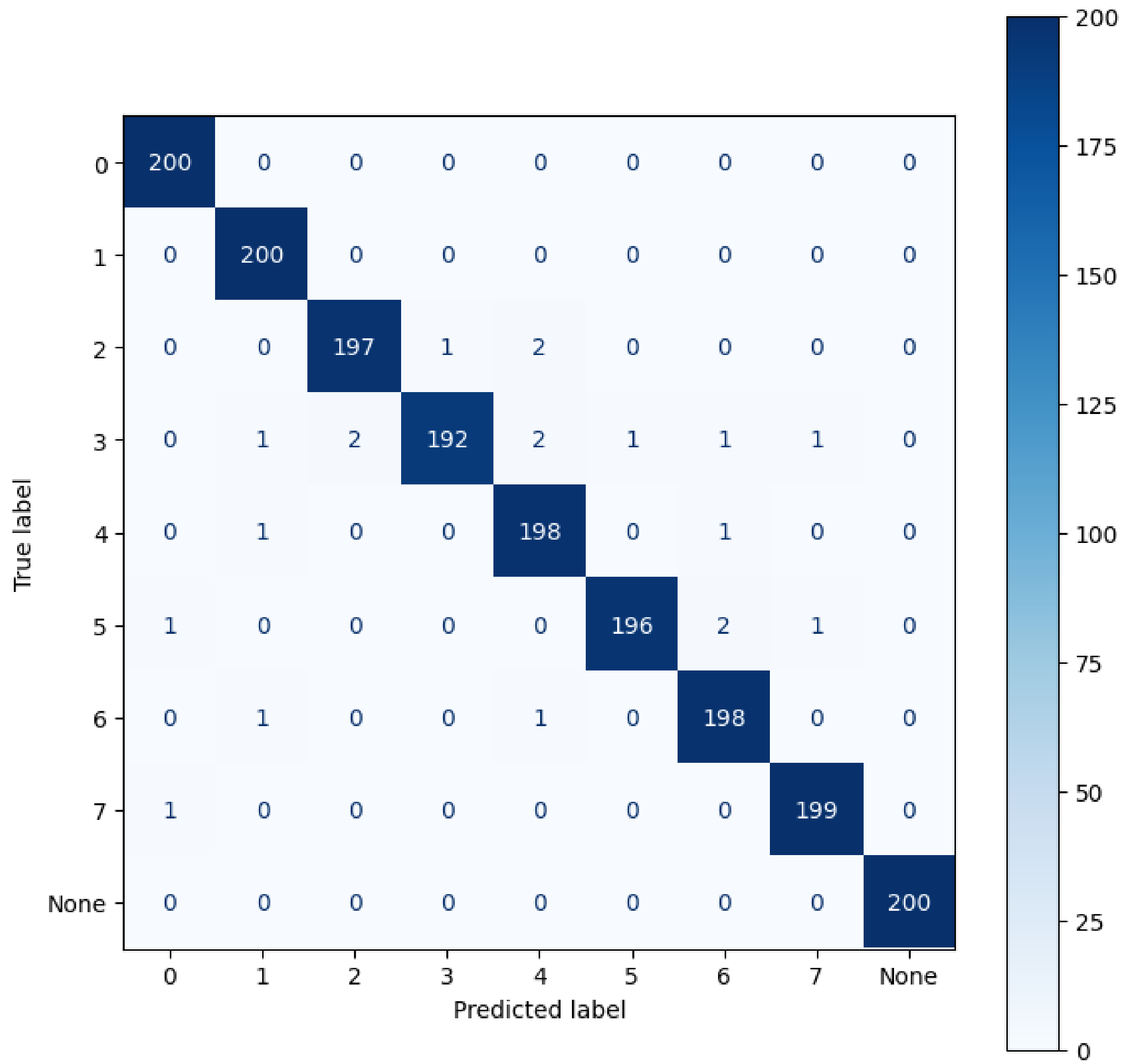
```
acc=[
accuracy(gnd_trt_0, image_list_0, preds_0),
accuracy(gnd_trt_1, image_list_1, preds_1),
accuracy(gnd_trt_2, image_list_2, preds_2),
accuracy(gnd_trt_3, image_list_3, preds_3),
accuracy(gnd_trt_4, image_list_4, preds_4),
accuracy(gnd_trt_5, image_list_5, preds_5),
accuracy(gnd_trt_6, image_list_6, preds_6),
accuracy(gnd_trt_7, image_list_7, preds_7),
accuracy(gnd_trt_8, image_list_8, preds_8),
]

import matplotlib.pyplot as plt

labels = [f'{i}' for i in range(9)]
labels[8] = 'None'
fig, ax = plt.subplots(figsize=(8,4))
ax.barh(labels, acc, align='center')

# Add axis labels and title
ax.set_xlabel('Accuracy')
ax.set_ylabel('Class')
ax.set_title('Accuracy by Class')

ax.invert_yaxis() # Invert y-axis to list classes from top to bottom

for i, acc in enumerate(acc):
    ax.text(acc -11, i, f'{acc:.2f}%', va='center',weight='bold',color='white')
plt.show()
```



In [29]:
```
pd = [[] for _ in range(9)]

for i in range(9):
    pred_len = len(locals()[f'preds_{i}'])
    for j in range(pred_len):
        pd[i].append(i)
```

In [26]:
```
y_true = np.concatenate([pd[i] for i in range(9)])
y_pred = np.concatenate([preds_0,preds_1,preds_2,preds_3,preds_4,preds_5,preds_6,pr
```

In [27]:
```
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

class_names = ['0', '1', '2', '3', '4', '5', '6', '7', 'None']

cm = confusion_matrix(y_true, y_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
disp.plot(cmap=plt.cm.Blues)
fig = disp.ax_.get_figure()
fig.set_figwidth(8)
fig.set_figheight(8)
plt.show()
```



In [28]:
```
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       200
           1       0.99      1.00      0.99       200
           2       0.99      0.98      0.99       200
           3       0.99      0.96      0.98       200
           4       0.98      0.99      0.98       200
           5       0.99      0.98      0.99       200
           6       0.98      0.99      0.99       200
           7       0.99      0.99      0.99       200
           8       1.00      1.00      1.00       200

    accuracy                           0.99      1800
   macro avg       0.99      0.99      0.99      1800
weighted avg       0.99      0.99      0.99      1800
```