American Sign Language Detection

In this project, I have created a model that will predict the hand signs based on the American Sign Language(ASL) standards.

The dataset is taken from Kaggle and it has a total of **36 classes** including images of the numbers from 0-9 and all the English alphabets from A-Z. It has around **2515 images in total and around 70 images** in each class.
I have split the dataset into training and testing sets where there are 2012 images for training (55 images in each class) and 503 images for testing (14 images in each class).

Dataset Link: https://www.kaggle.com/datasets/ayuraj/asl-dataset

Importing the required libraries

In [1]:
```
import cv2 as cv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import os

from tensorflow.keras.layers import MaxPooling2D, Dense, Flatten, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.metrics import CategoricalAccuracy
from tensorflow.keras.models import load_model
```

Setting the path of the training and testing dataset

In [2]:
```
train_path = "dataset/train"
test_path = "dataset/test"
```

Performing data augmentation

Using ImageDataGenerator we rescale the images and and also artificially create different training and testing images through different ways of processing like shear and zoom. This introduces a sort of randomness in the dataset.

In [7]:
```
train_datagen = ImageDataGenerator(rescale = 1/255,
                                  shear_range=0.2,
                                   zoom_range=0.2)

test_datagen = ImageDataGenerator(rescale = 1/255,
```

```
                                   shear_range=0.2,
                                   zoom_range=0.2)
```

In [8]:

```
train_set = train_datagen.flow_from_directory(train_path,
                                              target_size = (224, 224),
                                              batch_size = 32,
                                              class_mode = 'categorical')
test_set = test_datagen.flow_from_directory(test_path,
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'categorical')
```

```
Found 2012 images belonging to 36 classes.
Found 503 images belonging to 36 classes.
```

In [9]:

```
y_train = train_set.classes
y_test = test_set.classes

train_set.class_indices
```

Out[9]:

```
{'0': 0,
 '1': 1,
 '2': 2,
 '3': 3,
 '4': 4,
 '5': 5,
 '6': 6,
 '7': 7,
 '8': 8,
 '9': 9,
 'a': 10,
 'b': 11,
 'c': 12,
 'd': 13,
 'e': 14,
 'f': 15,
 'g': 16,
 'h': 17,
 'i': 18,
 'j': 19,
 'k': 20,
 'l': 21,
 'm': 22,
 'n': 23,
 'o': 24,
 'p': 25,
 'q': 26,
 'r': 27,
 's': 28,
 't': 29,
 'u': 30,
 'v': 31,
 'w': 32,
 'x': 33,
 'y': 34,
 'z': 35}
```

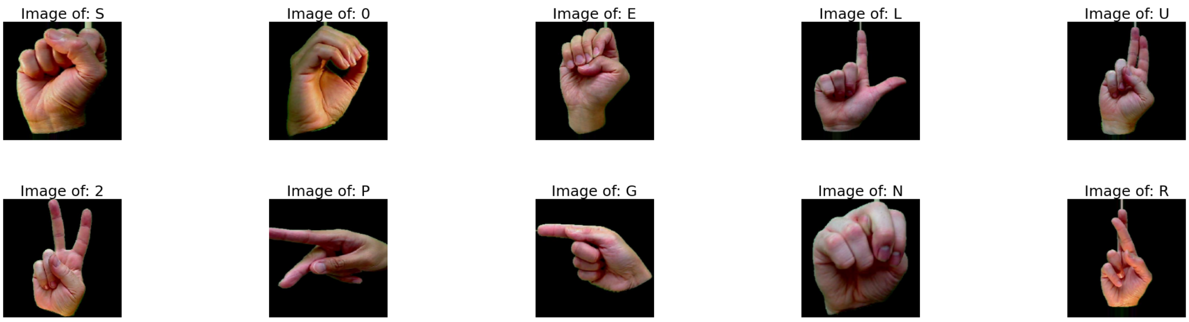
Plotting sample images from the training dataset

In [10]:

```
label_names = ['0', '1', '2', '3', '4', '5',
               '6', '7', '8', '9', 'A', 'B',
               'C', 'D', 'E', 'F', 'G', 'H',
               'I', 'J', 'K', 'L', 'M', 'N',
```

```
              'O', 'P', 'Q', 'R', 'S', 'T',
              'U', 'V', 'W', 'X', 'Y', 'Z']
```

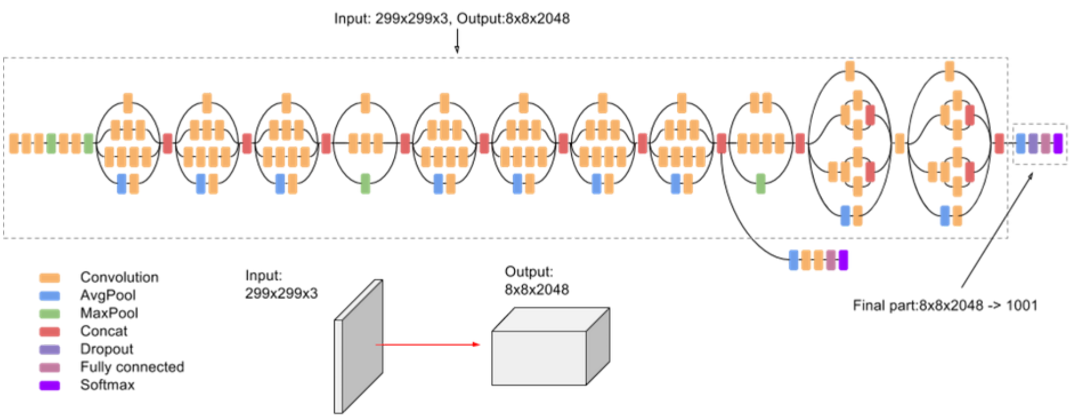
In [11]:
```
imgs, labels = next(iter(train_set))
counter = 1
for img, label in zip(imgs, labels):
    plt.subplot(5,5,counter)
    plt.subplots_adjust(right=5, top=5, wspace=0.5, hspace=0.5)
    value=np.argmax(label)
    labelname=label_names[value]
    plt.imshow(img)
    plt.title("Image of: "+labelname, fontdict={'fontsize': 25})
    counter+=1
    plt.axis("off")
    if(counter>10):
        break

plt.show()
```



Creating the model

InceptionV3 transfer learning

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a different task. So an already trained model on some other dataset is used and modified to fit the new task.
Inception v3 is an image recognition model that has been shown to attain greater than 78.1% accuracy on the ImageNet dataset. The model is the culmination of many ideas developed by multiple researchers over the years.



Loading inceptionV3 as the base model

In [12]:
```
base_model = InceptionV3(input_shape=(224,224,3),
                         include_top=False,
                         weights = "imagenet")
```

In [13]:
```
base_model.trainable = False
```

Adding the base model and a few layers to our model

In [14]:
```
model = Sequential([
    base_model,
    MaxPooling2D(),
    Flatten(),
    Dense(36, activation="softmax")])
```

In [15]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 inception_v3 (Functional)   (None, 5, 5, 2048)        21802784  
                                                                 
 max_pooling2d_4 (MaxPooling  (None, 2, 2, 2048)       0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 8192)              0         
                                                                 
 dense (Dense)               (None, 36)                294948    
                                                                 
=================================================================
Total params: 22,097,732
Trainable params: 294,948
Non-trainable params: 21,802,784
_________________________________________________________________
```

Compiling and fitting the model on the training dataset

In [16]:
```
model.compile(optimizer=Adam(learning_rate = 0.01),
              loss = CategoricalCrossentropy(),
              metrics = [CategoricalAccuracy()])
```

In [21]:
```
model.fit(train_set,
         validation_data = test_set,
         steps_per_epoch = 32,
         epochs = 32)
```

```
Epoch 1/32
32/32 [==============================] - 114s 4s/step - loss: 75.7752 - categorica
l_accuracy: 0.2549 - val_loss: 26.1806 - val_categorical_accuracy: 0.5229
Epoch 2/32
32/32 [==============================] - 132s 4s/step - loss: 11.3058 - categorica
l_accuracy: 0.6520 - val_loss: 6.3638 - val_categorical_accuracy: 0.7555
Epoch 3/32
32/32 [==============================] - 160s 5s/step - loss: 7.4541 - categorical
_accuracy: 0.7431 - val_loss: 6.3073 - val_categorical_accuracy: 0.8231
Epoch 4/32
32/32 [==============================] - 157s 5s/step - loss: 6.0897 - categorical
_accuracy: 0.8157 - val_loss: 8.4653 - val_categorical_accuracy: 0.7893
Epoch 5/32
32/32 [==============================] - 143s 4s/step - loss: 5.2440 - categorical
_accuracy: 0.8167 - val_loss: 5.5587 - val_categorical_accuracy: 0.8509
Epoch 6/32
32/32 [==============================] - 115s 4s/step - loss: 3.8357 - categorical
_accuracy: 0.8676 - val_loss: 7.1680 - val_categorical_accuracy: 0.8072
Epoch 7/32
32/32 [==============================] - 157s 5s/step - loss: 4.1102 - categorical
_accuracy: 0.8706 - val_loss: 3.3246 - val_categorical_accuracy: 0.8688
Epoch 8/32
32/32 [==============================] - 137s 4s/step - loss: 3.7903 - categorical
_accuracy: 0.8652 - val_loss: 8.0009 - val_categorical_accuracy: 0.8231
Epoch 9/32
32/32 [==============================] - 74s 2s/step - loss: 6.6158 - categorical_
accuracy: 0.8304 - val_loss: 6.0190 - val_categorical_accuracy: 0.8310
Epoch 10/32
32/32 [==============================] - 69s 2s/step - loss: 4.5418 - categorical_
accuracy: 0.8510 - val_loss: 6.0803 - val_categorical_accuracy: 0.8290
Epoch 11/32
32/32 [==============================] - 85s 3s/step - loss: 3.6250 - categorical_
accuracy: 0.8794 - val_loss: 3.3195 - val_categorical_accuracy: 0.8887
Epoch 12/32
32/32 [==============================] - 76s 2s/step - loss: 4.2262 - categorical_
accuracy: 0.8828 - val_loss: 5.1528 - val_categorical_accuracy: 0.8867
Epoch 13/32
32/32 [==============================] - 80s 3s/step - loss: 7.0744 - categorical_
accuracy: 0.8828 - val_loss: 6.3674 - val_categorical_accuracy: 0.8569
Epoch 14/32
32/32 [==============================] - 83s 3s/step - loss: 4.2262 - categorical_
accuracy: 0.8922 - val_loss: 5.5608 - val_categorical_accuracy: 0.8410
Epoch 15/32
32/32 [==============================] - 79s 2s/step - loss: 3.4522 - categorical_
accuracy: 0.8980 - val_loss: 3.6311 - val_categorical_accuracy: 0.9125
Epoch 16/32
32/32 [==============================] - 78s 2s/step - loss: 3.1487 - categorical_
accuracy: 0.9118 - val_loss: 2.8936 - val_categorical_accuracy: 0.9185
Epoch 17/32
32/32 [==============================] - 84s 3s/step - loss: 2.8909 - categorical_
accuracy: 0.9196 - val_loss: 4.0013 - val_categorical_accuracy: 0.9185
Epoch 18/32
32/32 [==============================] - 81s 3s/step - loss: 3.5171 - categorical_
accuracy: 0.9121 - val_loss: 4.5746 - val_categorical_accuracy: 0.8867
Epoch 19/32
32/32 [==============================] - 83s 3s/step - loss: 2.7683 - categorical_
accuracy: 0.9225 - val_loss: 3.3338 - val_categorical_accuracy: 0.9145
Epoch 20/32
32/32 [==============================] - 92s 3s/step - loss: 2.7902 - categorical_
accuracy: 0.9108 - val_loss: 3.8350 - val_categorical_accuracy: 0.9125
Epoch 21/32
32/32 [==============================] - 85s 3s/step - loss: 2.3011 - categorical_
accuracy: 0.9343 - val_loss: 2.6748 - val_categorical_accuracy: 0.9225
Epoch 22/32
```

```
32/32 [==============================] - 84s 3s/step - loss: 1.3872 - categorical_
accuracy: 0.9490 - val_loss: 3.8232 - val_categorical_accuracy: 0.9046
Epoch 23/32
32/32 [==============================] - 76s 2s/step - loss: 2.5412 - categorical_
accuracy: 0.9235 - val_loss: 4.6172 - val_categorical_accuracy: 0.8946
Epoch 24/32
32/32 [==============================] - 93s 3s/step - loss: 4.7006 - categorical_
accuracy: 0.8971 - val_loss: 4.9880 - val_categorical_accuracy: 0.9284
Epoch 25/32
32/32 [==============================] - 82s 3s/step - loss: 2.3069 - categorical_
accuracy: 0.9363 - val_loss: 3.4355 - val_categorical_accuracy: 0.9105
Epoch 26/32
32/32 [==============================] - 85s 3s/step - loss: 2.7154 - categorical_
accuracy: 0.9238 - val_loss: 3.0825 - val_categorical_accuracy: 0.9145
Epoch 27/32
32/32 [==============================] - 93s 3s/step - loss: 2.5790 - categorical_
accuracy: 0.9297 - val_loss: 4.6318 - val_categorical_accuracy: 0.9026
Epoch 28/32
32/32 [==============================] - 112s 4s/step - loss: 1.6941 - categorical
_accuracy: 0.9434 - val_loss: 3.8219 - val_categorical_accuracy: 0.9105
Epoch 29/32
32/32 [==============================] - 85s 3s/step - loss: 4.2054 - categorical_
accuracy: 0.9059 - val_loss: 6.7358 - val_categorical_accuracy: 0.8946
Epoch 30/32
32/32 [==============================] - 85s 3s/step - loss: 5.1555 - categorical_
accuracy: 0.9258 - val_loss: 9.4819 - val_categorical_accuracy: 0.8628
Epoch 31/32
32/32 [==============================] - 78s 2s/step - loss: 2.7753 - categorical_
accuracy: 0.9414 - val_loss: 1.8054 - val_categorical_accuracy: 0.9483
Epoch 32/32
32/32 [==============================] - 87s 3s/step - loss: 3.2423 - categorical_
accuracy: 0.9392 - val_loss: 3.2049 - val_categorical_accuracy: 0.9205
```

Out[21]:

```
<keras.callbacks.History at 0x252edd2c100>
```

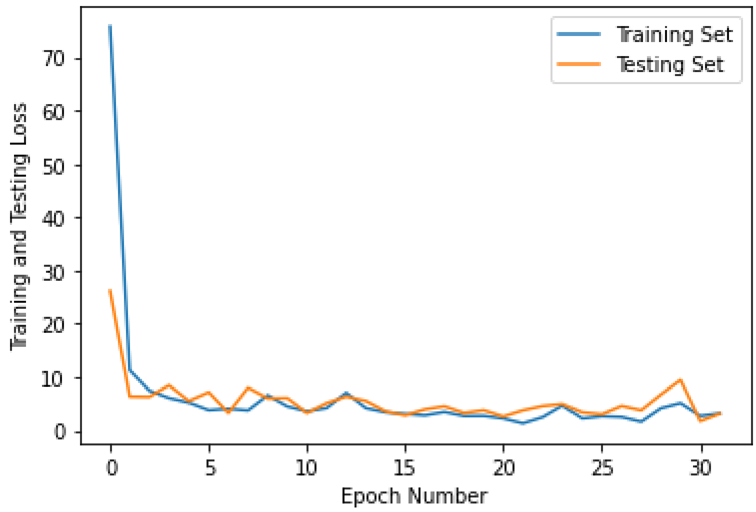
Plotting the Loss and Accuracy graphs

In [22]:

```python
plt.xlabel('Epoch Number')
plt.ylabel('Training and Testing Loss')
plt.plot(model.history.history['loss'], label='Training Set')
plt.plot(model.history.history['val_loss'], label='Testing Set')
plt.legend()
```

Out[22]:

```
<matplotlib.legend.Legend at 0x252efb985b0>
```

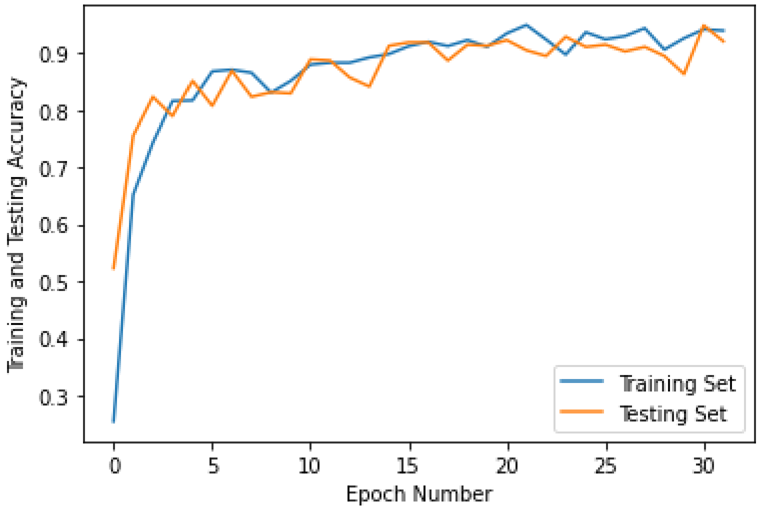


In [23]:

```python
plt.xlabel('Epoch Number')
```

```
plt.ylabel('Training and Testing Accuracy')
plt.plot(model.history.history['categorical_accuracy'], label='Training Set')
plt.plot(model.history.history['val_categorical_accuracy'], label='Testing Set')
plt.legend()
```

Out[23]: <matplotlib.legend.Legend at 0x252ef7cf430>



Saving the model

In [24]:
```
model_name = 'SignLanguage_recognition_inceptionv3.h5'
model.save(model_name, save_format='h5')
```

In [25]:
```
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
```

Testing the model's accuracy on the testing dataset

In [17]:
```
test_set = test_datagen.flow_from_directory(test_path,
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'categorical',
                                            shuffle=False)
```

```
Found 503 images belonging to 36 classes.
```

In [30]:
```
predictions = model.predict(test_set)
```

```
16/16 [==============================] - 36s 2s/step
```

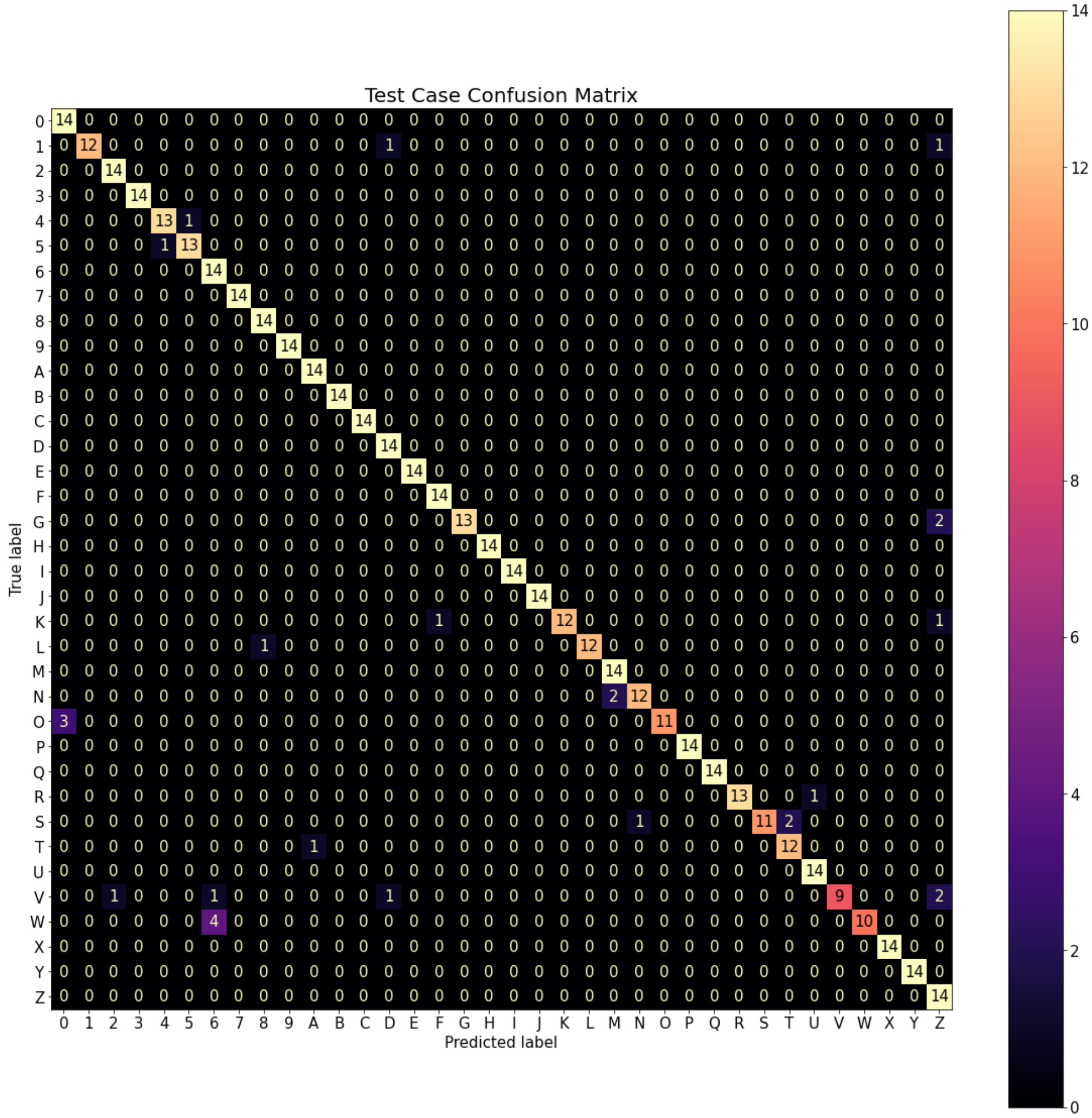
In [24]:
```
y_pred = [np.argmax(p) for p in predictions]
y_true = test_set.classes

print("False predictions are: ")
for i in range(len(y_pred)):
    if(y_true[i]!=y_pred[i]):
        print('index = {0:2d}, True class => {1}, {2} <= Predicted class'.format(i
```

```
False predictions are:
index = 22, True class => 1, D <= Predicted class
index = 26, True class => 1, Z <= Predicted class
index = 65, True class => 4, 5 <= Predicted class
index = 75, True class => 5, 4 <= Predicted class
index = 224, True class => G, Z <= Predicted class
index = 230, True class => G, Z <= Predicted class
index = 293, True class => K, Z <= Predicted class
index = 294, True class => K, F <= Predicted class
index = 298, True class => L, 8 <= Predicted class
index = 332, True class => N, M <= Predicted class
index = 334, True class => N, M <= Predicted class
index = 338, True class => O, 0 <= Predicted class
index = 342, True class => O, 0 <= Predicted class
index = 348, True class => O, 0 <= Predicted class
index = 390, True class => R, U <= Predicted class
index = 392, True class => S, T <= Predicted class
index = 403, True class => S, T <= Predicted class
index = 405, True class => S, N <= Predicted class
index = 415, True class => T, A <= Predicted class
index = 434, True class => V, 2 <= Predicted class
index = 435, True class => V, 6 <= Predicted class
index = 436, True class => V, Z <= Predicted class
index = 441, True class => V, Z <= Predicted class
index = 445, True class => V, D <= Predicted class
index = 449, True class => W, 6 <= Predicted class
index = 451, True class => W, 6 <= Predicted class
index = 458, True class => W, 6 <= Predicted class
index = 460, True class => W, 6 <= Predicted class
```

Plotting the Confusion Matrix

In [22]:

```
%matplotlib inline

cm = confusion_matrix(y_true, y_pred)
plt.rcParams['figure.figsize'] = (20,20)
plt.rcParams['font.size'] = 15
display_cm = ConfusionMatrixDisplay(cm, display_labels=label_names)

display_cm.plot(cmap='magma')

plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.ylabel('True label')
plt.xlabel('Predicted label')

plt.title('Test Case Confusion Matrix', fontsize=20)

plt.show()
```



Evaluating the model

In [23]:
```
model.evaluate(test_set)
```

```
16/16 [==============================] - 60s 4s/step - loss: 3.9102 - categorical_
accuracy: 0.9264
```

Out[23]:
```
[3.9102275371551514, 0.9264413714408875]
```

Thus we have created a model that recognizes the hand signs based on the ASL with an accuracy of 92.64% and a loss of 3.91