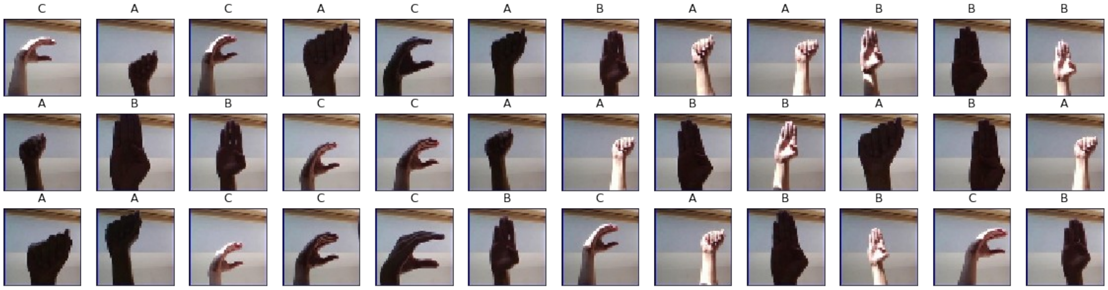
1. American Sign Language (ASL)

```
# Import packages and set numpy random seed
import numpy as np
np.random.seed(5)
import tensorflow as tf
tf.set_random_seed(2)
from datasets import sign_language
import matplotlib.pyplot as plt
%matplotlib inline

# Load pre-shuffled training and test datasets
(x_train, y_train), (x_test, y_test) = sign_language.load_data()
```

```
Using TensorFlow backend.
```

2. Visualize the training data

```
# Store labels of dataset
labels = ['A', 'B', 'C']

# Print the first several training images, along with the labels
fig = plt.figure(figsize=(20,5))
for i in range(36):
    ax = fig.add_subplot(3, 12, i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_train[i]))
    ax.set_title("{}".format(labels[y_train[i]]))
plt.show()
```



3. Examine the dataset

```
# Number of A's in the training dataset
num_A_train = sum(y_train==0)
# Number of B's in the training dataset
num_B_train = sum(y_train==1)
# Number of C's in the training dataset
num_C_train = sum(y_train==2)

# Number of A's in the test dataset
num_A_test = sum(y_test==0)
# Number of B's in the test dataset
num_B_test = sum(y_test==1)
```

```
# Number of C's in the test dataset
num_C_test = sum(y_test==2)

# Print statistics about the dataset
print("Training set:")
print("\tA: {}, B: {}, C: {}".format(num_A_train, num_B_train,
num_C_train))
print("Test set:")
print("\tA: {}, B: {}, C: {}".format(num_A_test, num_B_test,
num_C_test))
```

```
Training set:
      A: 540, B: 528, C: 532
Test set:
      A: 118, B: 144, C: 138
```

4. One-hot encode the data

```
from keras.utils import np_utils

# One-hot encode the training labels
y_train_OH = np_utils.to_categorical(y_train)

# One-hot encode the test labels
y_test_OH = np_utils.to_categorical(y_test)
```

5. Define the model

```
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Flatten, Dense
from keras.models import Sequential

model = Sequential()
# First convolutional layer accepts image input
model.add(Conv2D(filters=5, kernel_size=5, padding='same',
activation='relu',
                        input_shape=(50, 50, 3)))
# Add a max pooling layer
model.add(MaxPooling2D((4, 4)))
# Add a convolutional layer
model.add(Conv2D(filters=15, kernel_size=5, padding='same',
activation='relu'))
# Add another max pooling layer
model.add(MaxPooling2D((4, 4)))
# Flatten and feed to output layer
model.add(Flatten())
model.add(Dense(3, activation='softmax'))

# Summarize the model
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 50, 50, 5)         380       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 5)         0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 15)        1890      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 15)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 135)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 408       
=================================================================
Total params: 2,678
Trainable params: 2,678
Non-trainable params: 0
_________________________________________________________________
```

6. Compile the model

```
# Compile the model
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

7. Train the model

```
# Train the model
hist = model.fit(x=x_train, y=y_train_OH, epochs=2,
validation_split=0.2, batch_size=32)
```

```
Train on 1280 samples, validate on 320 samples
Epoch 1/2
1280/1280 [==============================] - 2s 2ms/step - loss:
0.9560 - acc: 0.6164 - val_loss: 0.7627 - val_acc: 0.8500
Epoch 2/2
1280/1280 [==============================] - 2s 2ms/step - loss:
0.6086 - acc: 0.8875 - val_loss: 0.4636 - val_acc: 0.9187
```

8. Test the model

```
# Obtain accuracy on test set
score = model.evaluate(x=x_test,
                       y=y_test_OH,
                       verbose=0)
print('Test accuracy:', score[1])
```

```
Test accuracy: 0.94
```

9. Visualize mistakes

```
# Get predicted probabilities for test dataset
y_probs = model.predict(x_test)

# Get predicted labels for test dataset
y_preds = np.argmax(y_probs, axis=1)

# Indices corresponding to test images which were mislabeled
bad_test_idxs = np.arange(len(y_test))[y_test != y_preds]

# Print mislabeled examples
fig = plt.figure(figsize=(25,4))
for i, idx in enumerate(bad_test_idxs):
    ax = fig.add_subplot(2, np.ceil(len(bad_test_idxs)/2), i + 1,
xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_test[idx]))
    ax.set_title("{} (pred: {})".format(labels[y_test[idx]],
labels[y_preds[idx]]))
```