Final Report of Internship Program 2021

On

"SIGN LANGUAGE RECOGNITION"

MEDTOUREASY, NEW DELHI

30th January 2021

Submitted by-

Shashank Shukla

ACKNOWLDEGMENTS

The internship opportunity that I had with MedTourEasy was a great change for learning and understanding the intricacies of the subject of Machine Learning; and also, for personal as well as professional development. I am very obliged for having a chance to interact with so many professionals who guided me throughout the internship project and made it a great learning curve for me.

Firstly, I express my deepest gratitude and special thanks to the Training Head of MedTourEasy, Mr. Ankit Hasija who gave me an opportunity to carry out my internship at their esteemed organization. Also, I express my thanks to him for making me understand the details of the Machine Learning profile and training me in the same so that I can carry out the project properly and with maximum client satisfaction and also for spearing his valuable time in spite of his busy schedule.

I would also like to thank the team of MedTourEasy and my colleagues who made the working environment productive and very conducive.

TABLE OF CONTENTS

ABSTRACT

The Survey of Income and Program Participation (SIPP) is one of a few national surveys that regularly collects data identifying the American population of persons with hearing loss or deafness. Estimates from the SIPP indicate that fewer than 1 in 20 Americans are currently deaf or hard of hearing. In round numbers, nearly 10,000,000 persons are hard of hearing and close to 1,000,000 are functionally deaf. More than half of all persons with hearing loss or deafness are 65 years or older and less than 4% are under 18 years of age.

However, these findings are limited to those who report difficulty hearing "normal conversation" and do not include the larger population of persons with hearing loss for which only hearing outside the range and circumstances of normal conversation is affected. Policy makers, communications technology manufacturers, health and education service providers, researchers, and advocacy organizations have an interest in these results.

According to WHO reports, Over 5% of the world's population – or 466 million people – has disabling hearing loss (432 million adults and 34 million children). It is estimated that by 2050 over 900 million people – or one in every ten people – will have disabling hearing loss. Approximately one third of people over 65 years of age are affected by disabling hearing loss. The prevalence in this age group is greatest in South Asia, Asia Pacific and sub-Saharan Africa.

In India, Four in every 1000 children suffer from severe to profound hearing loss. With over 100,000 babies that are born with hearing deficiency every year. The estimated prevalence of adult-onset deafness in India was found to be 7.6% and childhood onset deafness to be 2%. The National Sample Survey 58th round (2002) surveyed disability in Indian households and found that hearing disability was the 2nd most common cause of disability and top most cause of sensory deficit.

To stand against such problem, American Sign Language (ASL) is developed which is the primary language used by many deaf individuals in NorthAmerica, and it is also used by hard-of-hearing and hearing individuals. The language is as rich as spoken languages and employs signs made with the hand, along with facial gestures and bodily postures.

Therefore, this project aims at collecting and analyzing ASL dataset for 3 letters and training convolutional neural network on this data. This model can be used in various applications predicting sign languages to help individuals.

1.1 About the Company

MedTourEasy, a global healthcare company, provides you the informational resources needed to evaluate your global options. It helps you find the right healthcare solution based on specific health needs, affordable care while meeting the quality standards that you expect to have in healthcare.
MedTourEasy improves access to healthcare for people everywhere. It is an easy to use platform and service that helps patients to get medical second opinions and to schedule affordable, high-quality medical treatment abroad.

1.2 About the Project

According to WHO reports, Over 5% of the world's population – or 466 million people – has disabling hearing loss (432 million adults and 34 million children). It is estimated that by 2050 over 900 million people – or one in every ten people – will have disabling hearing loss. Approximately one third of people over 65 years of age are affected by disabling hearing loss. The prevalence in this age group is greatest in South Asia, Asia Pacific and sub-Saharan Africa.

In India, Four in every 1000 children suffer from severe to profound hearing loss. With over 100,000 babies that are born with hearing deficiency every year. The estimated prevalence of adult-onset deafness in India was found to be 7.6% and childhood onset deafness to be 2%. The National Sample Survey 58th round (2002) surveyed disability in Indian households and found that hearing disability was the 2nd most common cause of disability and top most cause of sensory deficit.

In that reference, it is extremely crucial to create such technologies which can predict ASL language signs indicated by deaf individuals. ASL is a language completely separate and distinct from English. It contains all the fundamental features of language, with its own rules for pronunciation, word formation, and word order. While every language has ways of signaling different functions, such as asking a question rather than making a statement, languages differ in how this is done.

Hence, this project aims at developing a convolutional neural network model which can predict ASL signs with good accuracy. The project is majorly divided into 9 subsections, as below,

- *American Sign Language (ASL)*
- *Visualize the training data*
- *Examine the dataset*
- *One-hot encode the data*
- *Define the model*
- *Compile the model*
- *Train the model*
- *Test the model*
- *Visualize mistakes*

Each of the above sub-section has been implemented using python language on Jupyter Notebook IDE and python packages like keras, numpy etc. These sections use a wide array of functions and packages in python to create intuitive and reliable models, which can then be used by any application as backened and could integrate this model to predict sign languages accurately.

1.3 Objectives and Deliverables

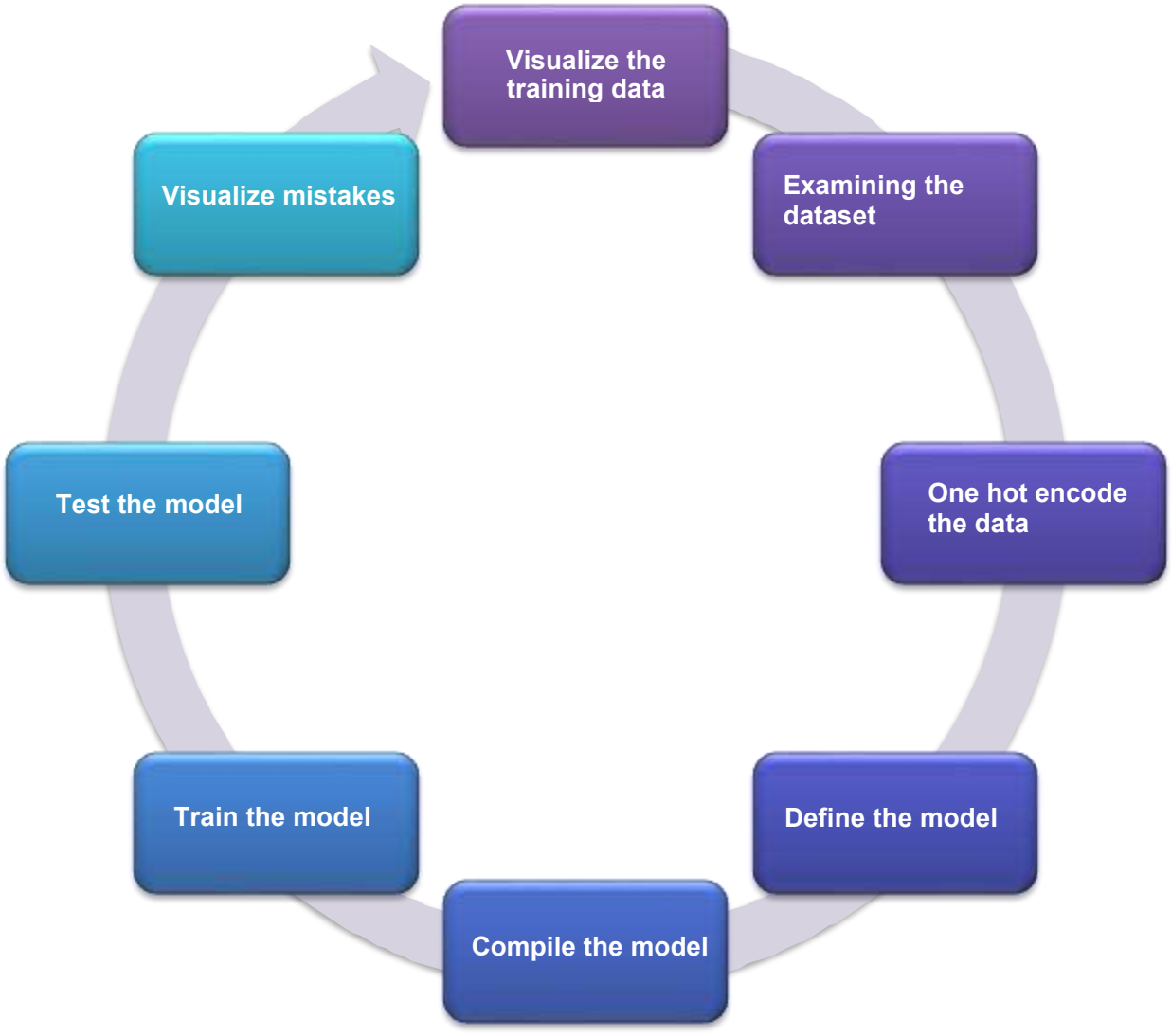
This project focuses on creating easily understandable, and accurate neural network model by training data on special kind of neural network architecture suitable for image data. Using the coding language python and packages like numpy, keras, matplotlib, and other Python Packages to create convolutional neural network model which will help to predict and detect sign languages accurately.

The project consists of the delieverable that use the training data and the label data to train the model for two epochs. It must attain a final validation accuracy of at least 80%. (Note that it has to set aside 20% of the training data to be used as validation data)

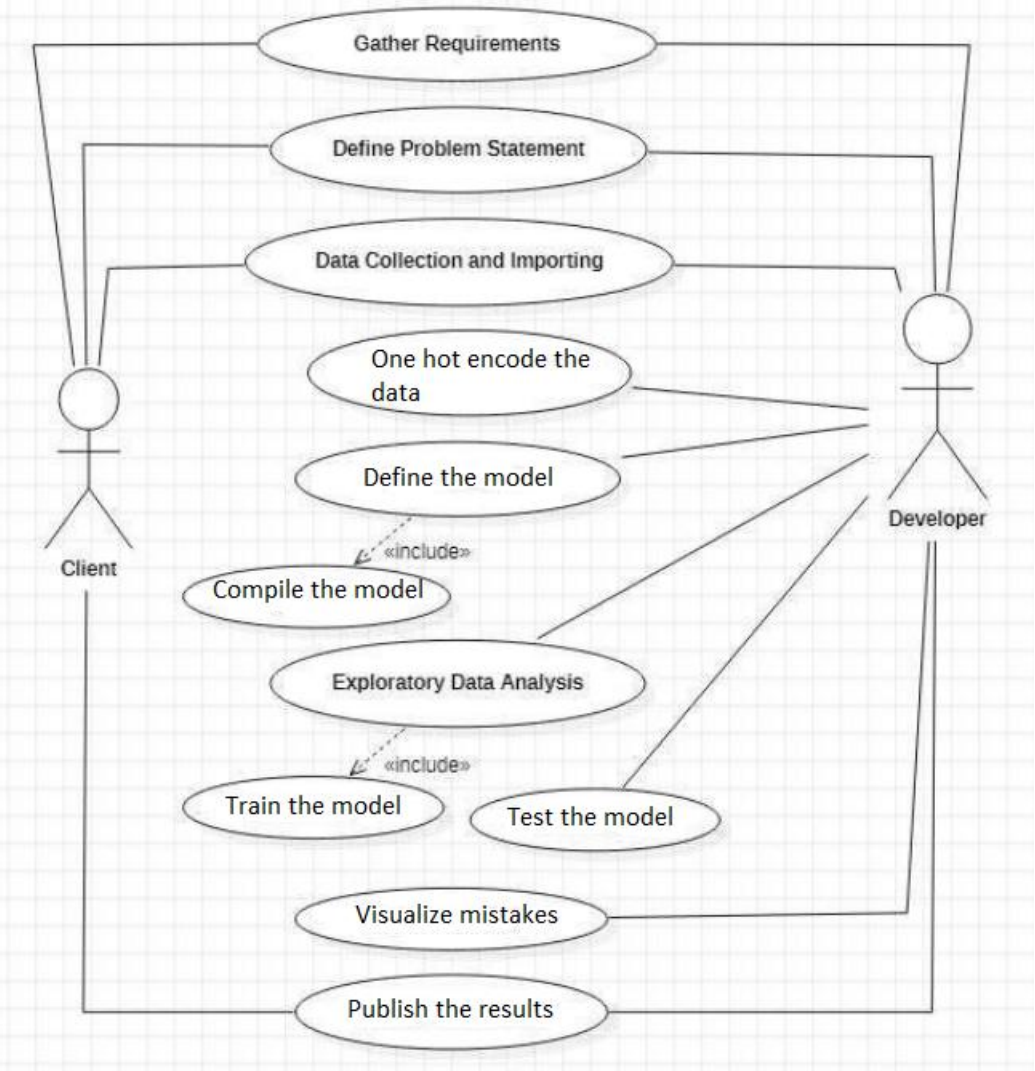
I. METHODOLOGY

2.1 Flow of the Project

The project followed the following steps to accomplish the desired objectives and deliverables. Each step has been explained in detail in the following section.



2.2 Use Case Diagram



Above figure shows the use case of the project. There are two main actors in the same: The Client and Developer. The developer will first gather requirements and define the problem statement then collecting the required data and importing it. Then the developer will do one hot encode the data. Next step is to define the model with required parameters. Next, model is being compiled with correct accuracy, loss function and optimizer. Then the model is trained on training data and tested on validation data. Finally, model is developed and analyzed to publish the results to the client.

2.3 Language and Platform Used

2.3.1 Language: Python

Python is an interpreted, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python was created in the late 1980s, and first released in 1991, by Guido van Rossum as a successor to the ABC programming language. Python 2.0, released in 2000. The important features of python are:

- Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc.
- Python language is freely available at the official website, it is free and open source.
- One of the key features of python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, objects encapsulation, etc.
- Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk in python..
- Python has a large standard library which provides a rich set of module and functions so you do not have to write your own code for every single thing.

2.3.2 IDE: Jupyter Notwbook

The Jupyter Notebook is a living online notebook, letting faculty and students weave together computational information (code, data, statistics) with narrative, multimedia, and graphs. Faculty can use it to set up interactive textbooks, full of explanations and examples which students can test out right from their browsers.Students can use it to explain their reasoning, show their work, and draw connections between their classwork and the world outside. Scientists, journalists, and researchers can use it to open up their data, share the stories behind their computations, and enable future collaboration and innovation. Major features are:

- One major feature of the Jupyter notebook is the ability to display plots that are the output of running code cells.
- The IPython kernel is designed to work seamlessly with the matplotlib plotting library to provide this functionality. Specific plotting library integration is a feature of the kernel.

  The Jupyter wiki has a long list of Kernels for other languages. They usually come with instructions on how to make the kernel available in the notebook.
- The notebook server verifies this signature when a notebook is opened.

2.3.3 Package: Numpy

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.

2.3.4 Package: Matplotlib

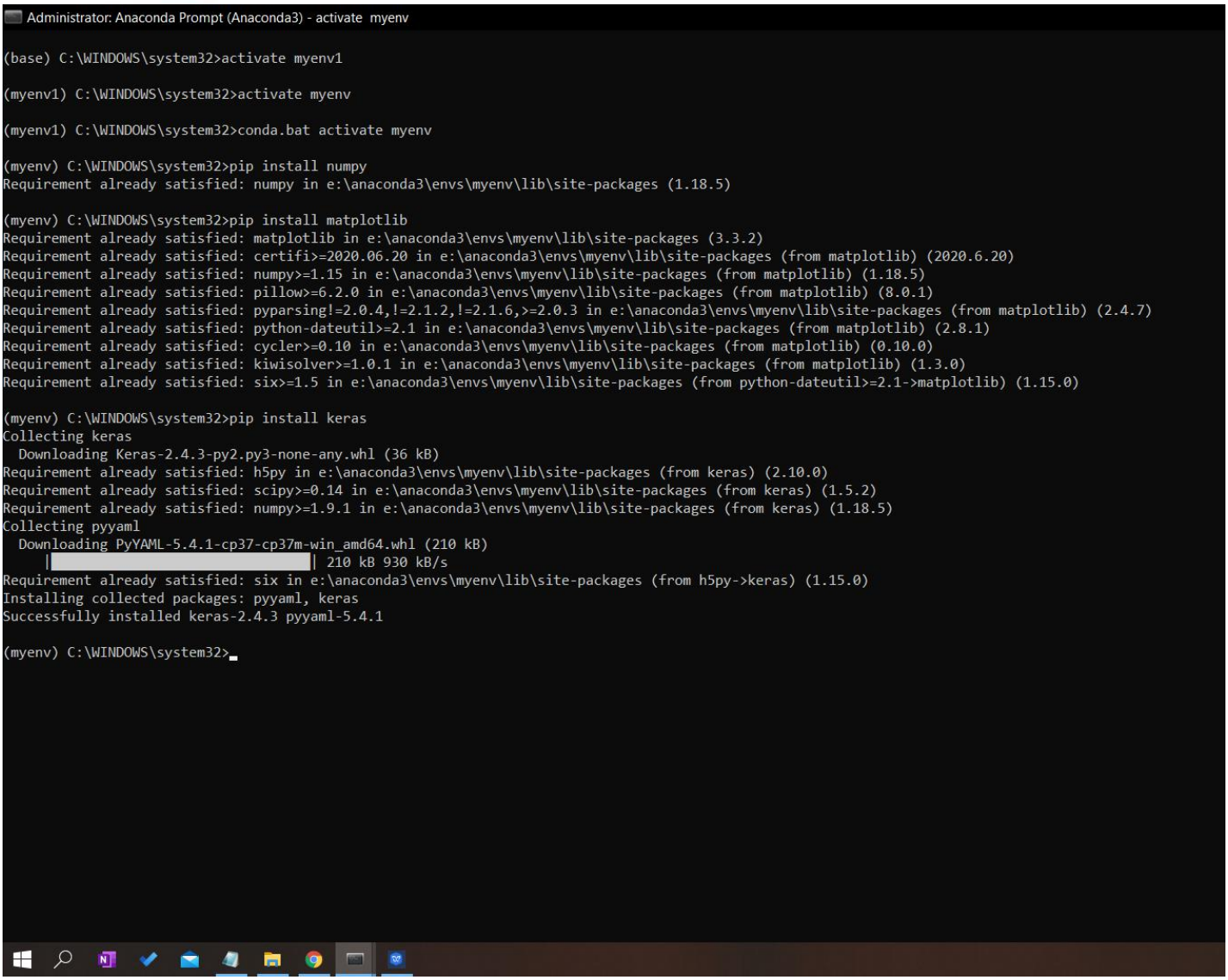
Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged.[3] SciPy makes use of Matplotlib.

2.3.5 Library: Keras

Keras is a minimalist Python library for deep learning that can run on top of Theano or TensorFlow. It was developed to make implementing deep learning models as fast and easy as possible for research and development. It runs on Python 2.7 or 3.5 and can seamlessly execute on GPUs and CPUs given the underlying frameworks. It is released under the permissive MIT license. Keras was developed and maintained by François Chollet, a Google engineer.

Installation:

```
Pip install numpy

Pip install matplotlib

Pip install Keras

Conda install jupyter-notebook
```



This shows the installation of required packages and library in anaconda terminal.

II. IMPLEMENTATION

3.1 American Sign Language (ASL)

What is American Sign Language?

American Sign Language (ASL) is a complete, natural language that has the same linguistic properties as spoken languages, with grammar that differs from English. ASL is expressed by movements of the hands and face. It is the primary language of many North Americans who are deaf and hard of hearing, and is used by many hearing people as well.

Is sign language the same in other countries?

There is no universal sign language. Different sign languages are used in different countries or regions. For example, British Sign Language (BSL) is a different language from ASL, and Americans who know ASL may not understand BSL. Some countries adopt features of ASL in their sign languages.

Where did ASL originate?

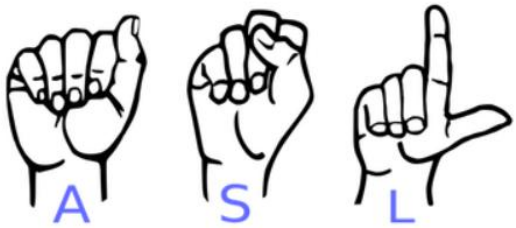
No person or committee invented ASL. The exact beginnings of ASL are not clear, but some suggest that it arose more than 200 years ago from the intermixing of local sign languages and French Sign Language (LSF, or Langue des Signes Française). Today's ASL includes some elements of LSF plus the original local sign languages; over time, these have melded and changed into a rich, complex, and mature language. Modern ASL and modern LSF are distinct languages. While they still contain some similar signs, they can no longer be understood by each other's users.

How does ASL compare with spoken language?

ASL is a language completely separate and distinct from English. It contains all the fundamental features of language, with its own rules for pronunciation, word formation, and word order. While every language has ways of signaling different functions, such as asking a question rather than making a statement, languages differ in how this is done. For example, English speakers may ask a question by raising the pitch of their voices and by adjusting word order; ASL users ask a question by raising their eyebrows, widening their eyes, and tilting their bodies forward.

1. American Sign Language (ASL)

American Sign Language (ASL) is the primary language used by many deaf individuals in North America, and it is also used by hard-of-hearing and hearing individuals. The language is as rich as spoken languages and employs signs made with the hand, along with facial gestures and bodily postures.



A lot of recent progress has been made towards developing computer vision systems that translate sign language to spoken language. This technology often relies on complex neural network architectures that can detect subtle patterns in streaming video. However, as a first step, towards understanding how to build a translation system, we can reduce the size of the problem by translating individual letters, instead of sentences.

In this notebook, we will train a convolutional neural network to classify images of American Sign Language (ASL) letters. After loading, examining, and preprocessing the data, we will train the network and test its performance.

In the code cell below, we load the training and test data.

- `x_train` and `x_test` are arrays of image data with shape `(num_samples, 3, 50, 50)`, corresponding to the training and test datasets, respectively.
- `y_train` and `y_test` are arrays of category labels with shape `(num_samples,)`, corresponding to the training and test datasets, respectively.

3.2 Data Collection and Importing

Data collection is a systematic approach for gathering and measuring information from a variety of sources in order to obtain a complete and accurate picture of an interest area. It helps an individual or organization to address specific questions, determine outcomes and forecast future probabilities and patterns.

Sample data-

A-

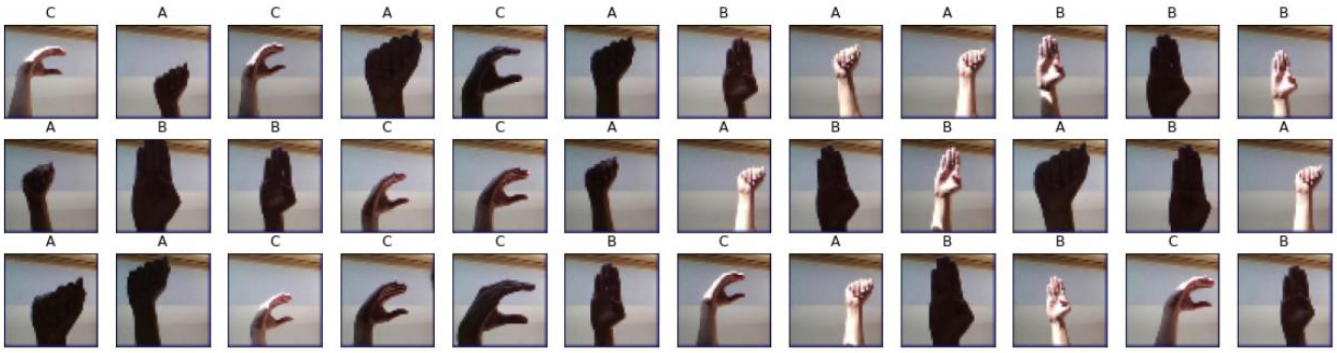
B-

C-

2. Visualize the training data

Now we'll begin by creating a list of string-valued labels containing the letters that appear in the dataset. Then, we visualize the first several images in the training data, along with their corresponding labels.

In [2]:
```
# 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

Let's examine how many images of each letter can be found in the dataset.

Remember that dataset has already been split into training and test sets for you, where `x_train` and `x_test` contain the images, and `y_train` and `y_test` contain their corresponding labels.

Each entry in `y_train` and `y_test` is one of `0`, `1`, or `2`, corresponding to the letters `'A'`, `'B'`, and `'C'`, respectively.

We will use the arrays `y_train` and `y_test` to verify that both the training and test sets each have roughly equal proportions of each letter.

In [3]:
```
# 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
```

Sample Code:

```
import random
import numpy as np
from keras.utils import np_utils, to_categorical
from keras.preprocessing import image
from os import listdir
from os.path import isdir, join


def load_data(container_path='datasets', folders=['A', 'B', 'C'],
              size=2000, test_split=0.2, seed=0):
    """
    Loads sign language dataset.
    """

    filenames, labels = [], []

    for label, folder in enumerate(folders):
        folder_path = join(container_path, folder)
        images = [join(folder_path, d)
                  for d in sorted(listdir(folder_path))]
        labels.extend(len(images) * [label])
        filenames.extend(images)

    random.seed(seed)
    data = list(zip(filenames, labels))
    random.shuffle(data)
    data = data[:size]
    filenames, labels = zip(*data)


    # Get the images
    x = paths_to_tensor(filenames).astype('float32')/255
    # Store the one-hot targets
    y = np.array(labels)

    x_train = np.array(x[:int(len(x) * (1 - test_split))])
    y_train = np.array(y[:int(len(x) * (1 - test_split))])
    x_test = np.array(x[int(len(x) * (1 - test_split)):])
    y_test = np.array(y[int(len(x) * (1 - test_split)):])

    return (x_train, y_train), (x_test, y_test)


def path_to_tensor(img_path, size):
    # loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size=(size, size))
    # convert PIL.Image.Image type to 3D tensor
    x = image.img_to_array(img)
    # convert 3D tensor to 4D tensor
    return np.expand_dims(x, axis=0)

def paths_to_tensor(img_paths, size=50):
    list_of_tensors = [path_to_tensor(img_path, size) for img_path in
img_paths]
    return np.vstack(list_of_tensors)


"""
    num_types = len(data['target_names'])
    targets = np_utils.to_categorical(np.array(data['target']), num_types)
"""
```

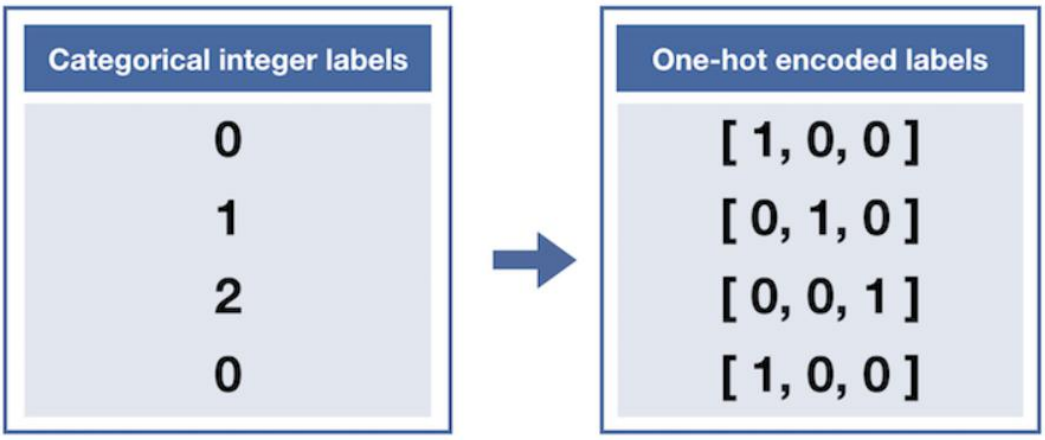
3.3 One-hot encode the data

In this function the built-in Keras function to_categorical is used to one-hot encode the data. Class vector y_train is used to assign the variable y_train_OH to the one-hot training labels and class vector y_test is used to assign the variable y_test_OH to the one-hot test labels.

4. One-hot encode the data

Currently, our labels for each of the letters are encoded as categorical integers, where `'A'`, `'B'` and `'C'` are encoded as `0`, `1`, and `2`, respectively. However, recall that Keras models do not accept labels in this format, and we must first one-hot encode the labels before supplying them to a Keras model.

This conversion will turn the one-dimensional array of labels into a two-dimensional array.



Each row in the two-dimensional array of one-hot encoded labels corresponds to a different image. The row has a `1` in the column that corresponds to the correct label, and `0` elsewhere.

For instance,

- `0` is encoded as `[1, 0, 0]`,
- `1` is encoded as `[0, 1, 0]`, and
- `2` is encoded as `[0, 0, 1]`.

Screenshot-

In [101]:
```
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)

print(y_train)
print(y_train_OH)
```

```
[2 0 2 ... 0 2 2]
[[0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]
 ...
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 0. 1.]]
```

3.4 Define the model

Specified a convolutional neural network in Keras.

• The first convolutional layer in the network contains a filter of size (5,5), padding as same, activation function as relu and input shape of images as (50,50,3) which denotes that each image is of size 50x50 and 3 represents that each image is a RGB.

• Added a max pooling layer (pooling over windows of size 4x4).

• Added another convolutional layer (15 filters, kernel size of 5, same padding, relu activation).

• Added another max pooling layer (pooling over windows of size 4x4)

After these layers, flatten layer is added to flatten all output of convolution layers to feed it into dense layer which expects flatten input. Then Dense layer is added with 3 as input and softmax as activation function. It shows that output will range between 3 values I.e., 0, 1 or 2 but they are hot encoded.

Functions Used:

Sequential: A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor. We can create a Sequential model by passing a list of layers to the Sequential constructor.

MaxPooling2D: Downsamples the input representation by taking the maximum value over the window defined by pool_size for each dimension along the features axis. The window is shifted by strides in each dimension. The resulting output when using "valid" padding option has a shape(number of rows or columns) of: output_shape = (input_shape - pool_size + 1) / strides)

Conv2D: This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If use_bias is True, a bias vector is created and added to the outputs. Finally, if activation is not None, it is applied to the outputs as well.

Flatten: Flattens the input. So, that it could be inputted to Dense layer. Converts all n-dimensional vectors into 1 dimensional vector.

Dense: Dense implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer

Softmax: Softmax is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector.

Relu: The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero.

Filter: In Convolutional Neural Networks, Filters detect spatial patterns such as edges in an image by detecting the changes in intensity values of the image.

Padding: One of "valid" or "same". "valid" means no padding. "same" results in padding evenly to the left/right or up/down of the input such that output has the same height/width dimension as the input.

Screenshot-

5. Define the model

Now it's time to define a convolutional neural network to classify the data.

This network accepts an image of an American Sign Language letter as input. The output layer returns the network's predicted probabilities that the image belongs in each category.

In [98]:
```
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))
# 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))
# Flatten and feed to output layer
model.add(Flatten())
model.add(Dense(3, activation='softmax'))

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

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 50, 50, 5)         380
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 12, 12, 5)         0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 12, 12, 15)        1890
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 3, 3, 15)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 135)               0
_________________________________________________________________
dense_4 (Dense)              (None, 3)                 408
=================================================================
Total params: 2,678
Trainable params: 2,678
Non-trainable params: 0
```

3.4 Compile the model

Specified the optimizer and loss function, along with a metric that will be tracked during training. Compiled the model with the 'rmsprop' optimizer, 'categorical_crossentropy' as the loss function, and 'accuracy' as a metric.

Functions Used:

Optimizer: Optimizers are algorithms or methods used to change the attributes of the neural network such as weights and learning rate to reduce the losses.

rmsprop: It is an optimization algorithm whose implementation used plain mountain. Maintains a moving average of the gradients.

Loss: The purpose of loss functions is to compute the quantity that a model should seek to minimize during training.

categorical_crossentropy: Use this crossentropy loss function when there are two or more label classes. It expects labels to be provided in a one_hot representation.

Metrics: A metric is a function that is used to judge the performance of your model. Metric functions are similar to loss functions, except that the results from evaluating a metric are not used when training the model.

accuracy: It calculates how often predictions equal labels.

Screenshot-

6. Compile the model ¶

After we have defined a neural network in Keras, the next step is to compile it!

In [99]:
```
# Compile the model
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

3.5 Train the model

Used the model's .fit() method of keras to train the model. The training data in x_train and the label data in y_train_OH was used to train the model for two epochs. Goal was to must attain a final validation accuracy of at least 80%. 20% of training data is used as validation data and batch_size is kept as 32. After running the model for 2 epochs, model achieved accuracy of 91% on testing data which is quite decent.

Functions Used:

Model.fit(): It trains the model for a fixed number of epochs (iterations on a dataset). Its like passing training dataset over and over again to set correct weights.

Epoch: Number of epochs to train the model. An epoch is an iteration over the entire x and y data provided.

Batch_size: It is the number of samples per gradient update. If unspecified, batch_size will default to 32. Gradient means error or value by which weights need to be updated.

Screenshot-

7. Train the model

Once we have compiled the model, we're ready to fit it to the training data.

In [100]:
```
# Train the model
hist = model.fit(x_train, y_train_OH, validation_data=(x_test, y_test_OH), epochs=2, batch_size=32)
```

```
Train on 1600 samples, validate on 400 samples
Epoch 1/2
1600/1600 [==============================] - 1s 922us/step - loss: 0.8863 - accuracy: 0.6394 - val_loss: 0.6804 - val_accuracy:
0.7225
Epoch 2/2
1600/1600 [==============================] - 0s 175us/step - loss: 0.5239 - accuracy: 0.8094 - val_loss: 0.4001 - val_accuracy:
0.9150
```

3.6 Test the model

Used the model's .evaluate() method to evaluate the model.. Assigned x to the test data in x_test, and assigned y to the test labels in y_test_OH.. be drawn.

Functions Used:

Model.evaluate(): It evaluates the model on the test data via evaluate().

Verbose: It shows how process will be shown. Possible values are 0, 1, or 2. Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per epoch.

x and y: Variables taking feature values and label values correspondingly.

Screenshot-

8. Test the model

To evaluate the model, we'll use the test dataset. This will tell us how the network performs when classifying images it has never seen before!

If the classification accuracy on the test dataset is similar to the training dataset, this is a good sign that the model did not overfit to the training data.

In [102]:
```
# Obtain accuracy on test set
score = model.evaluate(x=x_test,
                       y=y_test_OH,
                       verbose=0)
print('Test accuracy:', score[1])
```

```
Test accuracy: 0.9150000214576721
```

3.7 Visualize mistakes

Visualized images that were incorrectly classified by the model. using the model's.predict() method. Assigned y_probs to a numpy array with shape (600,3) containing the model's predicted probabilities that each image belongs to each imageclass. Assigned y_preds to the model's predicted labels for each image in the testing dataset. y_preds is a numpy array with shape (600,), where each entry is one of 0, 1,or 2, corresponding to 'A', 'B', and 'C', respectively. Used the ground truth labels for the testing dataset (y_test) and the model's predicted labels(y_preds) to determine which images were misclassified. Assigned the variable bad_test_idxs to a one-dimensional numpy array containing all indices corresponding to images that were incorrectly classified by the model.

Functions Used:

Model.predict(): This function predicts labels of test dataset containing features. In this program, it gives predicted output as one-hot encoded labels as training data contains same specifications.

Argmax: The numpy.argmax() function returns indices of the max element of the array in a particular axis. Axis=0 for maximum in column, 1 for rows and no value gives maximum in whole array.

Where: The numpy.where() function returns the indices of elements in an input array where the given condition is satisfied.

Figure: Matplotlib is a library in Python and it is numerical – mathematical extension for NumPy library. The figure() function in pyplot module of matplotlib library is used to create a new figure.

Add subplot: The add_subplot() method figure module of matplotlib library is used to add an Axes to the figure as part of a subplot arrangement.

Squeeze: numpy.squeeze() function is used when we want to remove single-dimensional entries from the shape of an array.

Figsize: figsize(float, float): These parameter are the width, height in inches.

Screenshot-

9. Visualize mistakes

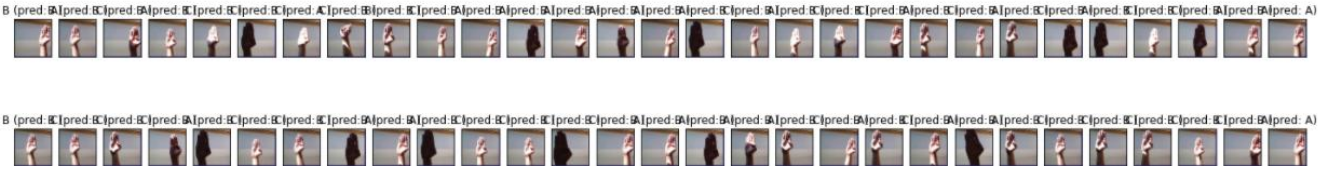
Hooray! Our network gets very high accuracy on the test set!

The final step is to take a look at the images that were incorrectly classified by the model. Do any of the mislabeled images look relatively difficult to classify, even to the human eye?

Sometimes, it's possible to review the images to discover special characteristics that are confusing to the model. However, it is also often the case that it's hard to interpret what the model had in mind!

In [93]:
```
# 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.where(y_preds != y_test)[0]

# 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]]))
```

```
E:\Anaconda3\envs\myenv1\lib\site-packages\ipykernel_launcher.py:13: MatplotlibDeprecationWarning: Passing non-integers as thre
e-element position specification is deprecated since 3.3 and will be removed two minor releases later.
  del sys.path[0]
```



In [94]:
```
l=[]
for i in range(4):
    l.append(bad_test_idxs[i])
l=np.array(l)

fig = plt.figure(figsize=(25,4))
for i, idx in enumerate(l):
    ax = fig.add_subplot(2, np.ceil(len(l)/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]]))
```

```
E:\Anaconda3\envs\myenv1\lib\site-packages\ipykernel_launcher.py:8: MatplotlibDeprecationWarning: Passing non-integers as three
-element position specification is deprecated since 3.3 and will be removed two minor releases later.
```

III. SAMPLE SCREENSHOTS AND OBSERVATIONS

jupyter notebook Last Checkpoint: 7 hours ago (autosaved)

Logout

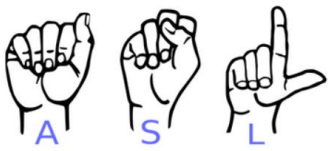
File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

1. American Sign Language (ASL)

American Sign Language (ASL) is the primary language used by many deaf individuals in North America, and it is also used by hard-of-hearing and hearing individuals. The language is as rich as spoken languages and employs signs made with the hand, along with facial gestures and bodily postures.



A lot of recent progress has been made towards developing computer vision systems that translate sign language to spoken language. This technology often relies on complex neural network architectures that can detect subtle patterns in streaming video. However, as a first step, towards understanding how to build a translation system, we can reduce the size of the problem by translating individual letters, instead of sentences.

In this notebook, we will train a convolutional neural network to classify images of American Sign Language (ASL) letters. After loading, examining, and preprocessing the data, we will train the network and test its performance.

In the code cell below, we load the training and test data.

- `x_train` and `x_test` are arrays of image data with shape `(num_samples, 3, 50, 50)`, corresponding to the training and test datasets, respectively.
- `y_train` and `y_test` are arrays of category labels with shape `(num_samples,)`, corresponding to the training and test datasets, respectively.

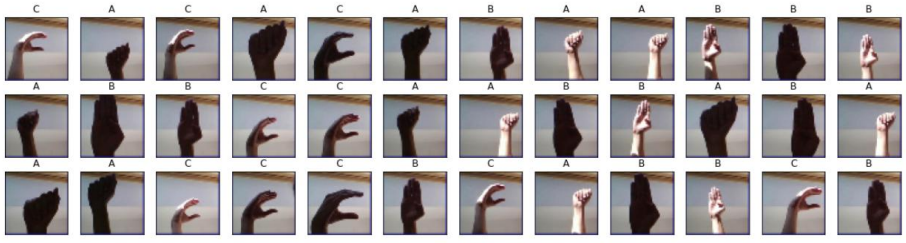
In [1]:
```
# Import packages and set numpy random seed
import numpy as np
np.random.seed(5)
import tensorflow as tf
# tf.set_random_seed(2)
from datasets import sign_language
import matplotlib.pyplot as plt
%matplotlib inline

# 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

Now we'll begin by creating a list of string-valued labels containing the letters that appear in the dataset. Then, we visualize the first several images in the training data, along with their corresponding labels.

In [2]:
```
# 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

Let's examine how many images of each letter can be found in the dataset.

Remember that dataset has already been split into training and test sets for you, where `x_train` and `x_test` contain the images, and `y_train` and `y_test` contain their corresponding labels.

Each entry in `y_train` and `y_test` is one of `0`, `1`, or `2`, corresponding to the letters `'A'`, `'B'`, and `'C'`, respectively.

We will use the arrays `y_train` and `y_test` to verify that both the training and test sets each have roughly equal proportions of each letter.

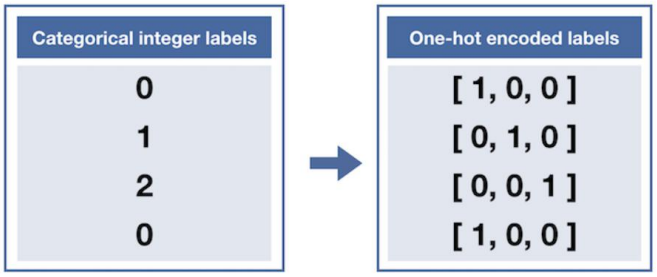
In [3]:
```
# Number of A's in the training dataset
num_A_train = sum(y_train==0)
# Number of B's in the training dataset
num_B_train = sum(y_train==1)
# Number of C's in the training dataset
num_C_train = sum(y_train==2)

# 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

Currently, our labels for each of the letters are encoded as categorical integers, where `'A'`, `'B'` and `'C'` are encoded as `0`, `1`, and `2`, respectively. However, recall that Keras models do not accept labels in this format, and we must first one-hot encode the labels before supplying them to a Keras model.

This conversion will turn the one-dimensional array of labels into a two-dimensional array.



Each row in the two-dimensional array of one-hot encoded labels corresponds to a different image. The row has a `1` in the column that corresponds to the correct label, and `0` elsewhere.

For instance,

- `0` is encoded as `[1, 0, 0]`,
- `1` is encoded as `[0, 1, 0]`, and
- `2` is encoded as `[0, 0, 1]`.

In [101]:
```
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
```

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

print(y_train)
print(y_train_OH)
```

```
[2 0 2 ... 0 2 2]
[[0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]
 ...
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 0. 1.]]
```

5. Define the model

Now it's time to define a convolutional neural network to classify the data.

This network accepts an image of an American Sign Language letter as input. The output layer returns the network's predicted probabilities that the image belongs in each category.

In [98]:
```
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))
# 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))
# Flatten and feed to output layer
model.add(Flatten())
model.add(Dense(3, activation='softmax'))

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

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 50, 50, 5)         380
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 12, 12, 5)         0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 12, 12, 15)        1890
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 3, 3, 15)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 135)               0
_________________________________________________________________
dense_4 (Dense)              (None, 3)                 408
=================================================================
Total params: 2,678
Trainable params: 2,678
Non-trainable params: 0
_________________________________________________________________
```

6. Compile the model

After we have defined a neural network in Keras, the next step is to compile it!

In [99]:
```
# Compile the model
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

7. Train the model

Once we have compiled the model, we're ready to fit it to the training data.

In [100]:
```
# Train the model
hist = model.fit(x_train, y_train_OH, validation_data=(x_test, y_test_OH), epochs=2, batch_size=32)
```

```
Train on 1600 samples, validate on 400 samples
Epoch 1/2
1600/1600 [==============================] - 1s 922us/step - loss: 0.8863 - accuracy: 0.6394 - val_loss: 0.6804 - val_accuracy:
```

7. Train the model

Once we have compiled the model, we're ready to fit it to the training data.

In [100]:
```
# Train the model
hist = model.fit(x_train, y_train_OH, validation_data=(x_test, y_test_OH), epochs=2, batch_size=32)
```

```
Train on 1600 samples, validate on 400 samples
Epoch 1/2
1600/1600 [==============================] - 1s 922us/step - loss: 0.8863 - accuracy: 0.6394 - val_loss: 0.6804 - val_accuracy:
0.7225
Epoch 2/2
1600/1600 [==============================] - 0s 175us/step - loss: 0.5239 - accuracy: 0.8094 - val_loss: 0.4001 - val_accuracy:
0.9150
```

8. Test the model

To evaluate the model, we'll use the test dataset. This will tell us how the network performs when classifying images it has never seen before!

If the classification accuracy on the test dataset is similar to the training dataset, this is a good sign that the model did not overfit to the training data.

In [102]:
```
# Obtain accuracy on test set
score = model.evaluate(x=x_test,
                       y=y_test_OH,
                       verbose=0)
print('Test accuracy:', score[1])
```

```
Test accuracy: 0.9150000214576721
```

9. Visualize mistakes

Hooray! Our network gets very high accuracy on the test set!

The final step is to take a look at the images that were incorrectly classified by the model. Do any of the mislabeled images look relatively difficult to classify, even to the human eye?

Sometimes, it's possible to review the images to discover special characteristics that are confusing to the model. However, it is also often the case that it's hard to interpret what the model had in mind!

In [93]:
```
# 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.where(y_preds != y_test)[0]

# 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]]))
```

```
E:\Anaconda3\envs\myenv1\lib\site-packages\ipykernel_launcher.py:13: MatplotlibDeprecationWarning: Passing non-integers as thre
e-element position specification is deprecated since 3.3 and will be removed two minor releases later.
  del sys.path[0]
```

In [94]:
```
l=[]
for i in range(4):
```

jupyter notebook Last Checkpoint: 7 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help

Trusted | Python 3

Run Code

In [94]:
```
l=[]
for i in range(4):
    l.append(bad_test_idxs[i])
l=np.array(l)

fig = plt.figure(figsize=(25,4))
for i, idx in enumerate(l):
    ax = fig.add_subplot(2, np.ceil(len(l)/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]]))
```

```
E:\Anaconda3\envs\myenv1\lib\site-packages\ipykernel_launcher.py:8: MatplotlibDeprecationWarning: Passing non-integers as three
-element position specification is deprecated since 3.3 and will be removed two minor releases later.
```

In [ ]:

IV. CONCLUSION AND FUTURE SCOPE

In India, Four in every thousand children suffer from severe to profound hearing loss. With over 100,000 babies that are born with hearing deficiency every year. The estimated prevalence of adult-onset deafness in India was found to be 7.6% and childhood onset deafness to be 2%. The National Sample Survey 58th round (2002) surveyed disability in Indian households and found that hearing disability was the 2nd most common cause of disability and top most cause of sensory deficit.

In that reference, it is extremely crucial to create such technologies which can predict ASL language signs indicated by deaf individuals. ASL is a language completely separate and distinct from English. It contains all the fundamental features of language, with its own rules for pronunciation, word formation, and word order. While every language has ways of signaling different functions, such as asking a question rather than making a statement, languages differ in how this is done.

In this project, a technique to predict signs of ASL language is explored harnessing the power of neural networks. The architecture explored is convolutional neural network which are specifically designed for image data. The proposed model achieves an accuracy of 91% just with 2 epochs. If trained further on training dataset, accuracy might be rise by a significant percentage. There are a lot of future works involved with this project. Different preprocessing techniques can be used to enhance results. Data augmentation and batch normalization can also be used to increase accuracy or decrease loss on testing data. The model proposed in this project can be deployed and used as backened for any application helping deaf individuals in real life by predicting asl signs.

V. REFERENCES

The following websites have been referred for python programming and machine learning tutorials and other statistics:

a. https://machinelearningmastery.com/introduction-python-deep-learning-library-keras

b. https://matplotlib.org/devdocs/api/_as_gen/matplotlib.pyplot.figure.html

c. https://www.w3schools.com/python/numpy_intro.asp

d. https://jupyter-notebook.readthedocs.io/en/stable/notebook.html

e. https://blendedlearning.blogs.brynmawr.edu/what-are-jupyter-notebooks-why-would-i-want-to-use-them/

f. https://www.geeksforgeeks.org/python-features/

g. https://www.python.org/doc/

h. https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss

i. https://www.nidcd.nih.gov/health/american-sign-language

j. https://numpy.org/doc/stable/reference/generated/numpy.squeeze.html

k. https://matplotlib.org/devdocs/api/_as_gen/matplotlib.pyplot.subplots.html

l. https://keras.io/guides/sequential_model/

m. https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/