**MACHINE LEARNING – SIGN LANGUAGE PROJECT REPORT**

**-BY MADHUSUDAN PANWAR(20250509) AND SNEHAL DESHMUKH (20250404)**

* **Overview:** Our project is majorly based on CNN, deep learning. It is on predicting the meaning of each sign(symbol) to its corresponding alphabet. It can be very useful in the real world to recognise the sign language easily for a naïve individual who doesn’t understand it.

We have fetched the dataset from Kaggle and then learnt about CNN-Lenet to proceed ahead.

We have implemented the project in Python using various libraries and functions which would be described later.

* **Importing the data:**

To import the large amount of image data, we have used load\_img function from keras.preprocessing module of keras library.

training\_set = train\_datagen.flow\_from\_directory('C:\\Users\\sneha\\OneDrive\\Desktop\\archive\\asl\_alphabet\_train\\train',

target\_size = (64, 64),

batch\_size = 32,

class\_mode = 'sparse')

test\_image = image.load\_img('C:\\Users\\sneha\\OneDrive\\Desktop\\archive\\asl\_alphabet\_test\\test\\W\_test.jpg',

target\_size = (64, 64))

* **Data definition-**

**model:** Our model(sequential model), which we are generating as we proceed, is saved in this variable.

**train\_datagen:** It contains the information about how training data needs to be fetched and stored(it’s format).

**test\_datagen:** It contains the information about how test data needs to be fetched and stored(it’s format).

**training\_set:** Storing the training data in the above specified format.

**test\_set:** Storing the test data in the above specified format.

**modelasl.h5:** It is the whole generated model as specified.

**test\_image:** It has the image that we are passing to test or predict the letter we want to generate(i.e., the output).

**result:** After the prediction, storing the output as an array.

* **Libraries(packages) and layers/models/dataset/functions used in Python-**

**library(tensorflow) :** It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of [TensorFlow](https://machinelearningmastery.com/tensorflow-tutorial-deep-learning-with-tf-keras/).

**library(keras) :** It provides a python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

**SEQUENTIAL MODEL:** Sequential model is the simplest model in Keras, which is built by stacking layers sequentially.

**CONV2D:** A convolution layer extracts features from a source image by “scanning” the image with a filter. A 2D convolution layer means that the input of the convolution operation is three-dimensional.

Conv2d() is a TensorFlow function that is used to build a 2D convolutional layer as part of a CNN architecture.

**MaxPooling2D:** A pooling layer reduces the dimensionality of each feature to focus on the most important elements. This is done to in part to help over-fitting by providing an abstracted form of the representation.

MaxPooling2D() is used to max pool the value from the given size matrix.

**Dense:** Dense is used to make this a fully connected model and is the hidden layer.

**Flatten:** Flattenis used to flatten the dimensions of the image obtained after convolving it.

**ImageDataGenerator**:Rescales the image, applies shear in some range, zooms the image and does horizontal flipping with the image. This ImageDataGenerator includes all possible orientation of the image.

* **Algorithms referred/used-**

**CNN:** A CNN is a neural network algorithm used to recognize patterns in data.

In machine learning, a classifier assigns a class label to a data point. For example, an image classifier produces a class label (e.g, balloons, coffee, lifeboat etc) for what objects exist within an image.

In short, we can say, CNN(Convolutional Neural Network) is a type of classifier which does this.

A CNN has three main building blocks-

1. **A convolutional layer:** The convolutional layer is the very first layer where we extract features from the images in our datasets.

Convolution is basically filtering the image with a smaller pixel filter to decrease the size of the image without losing the relationship between pixels.

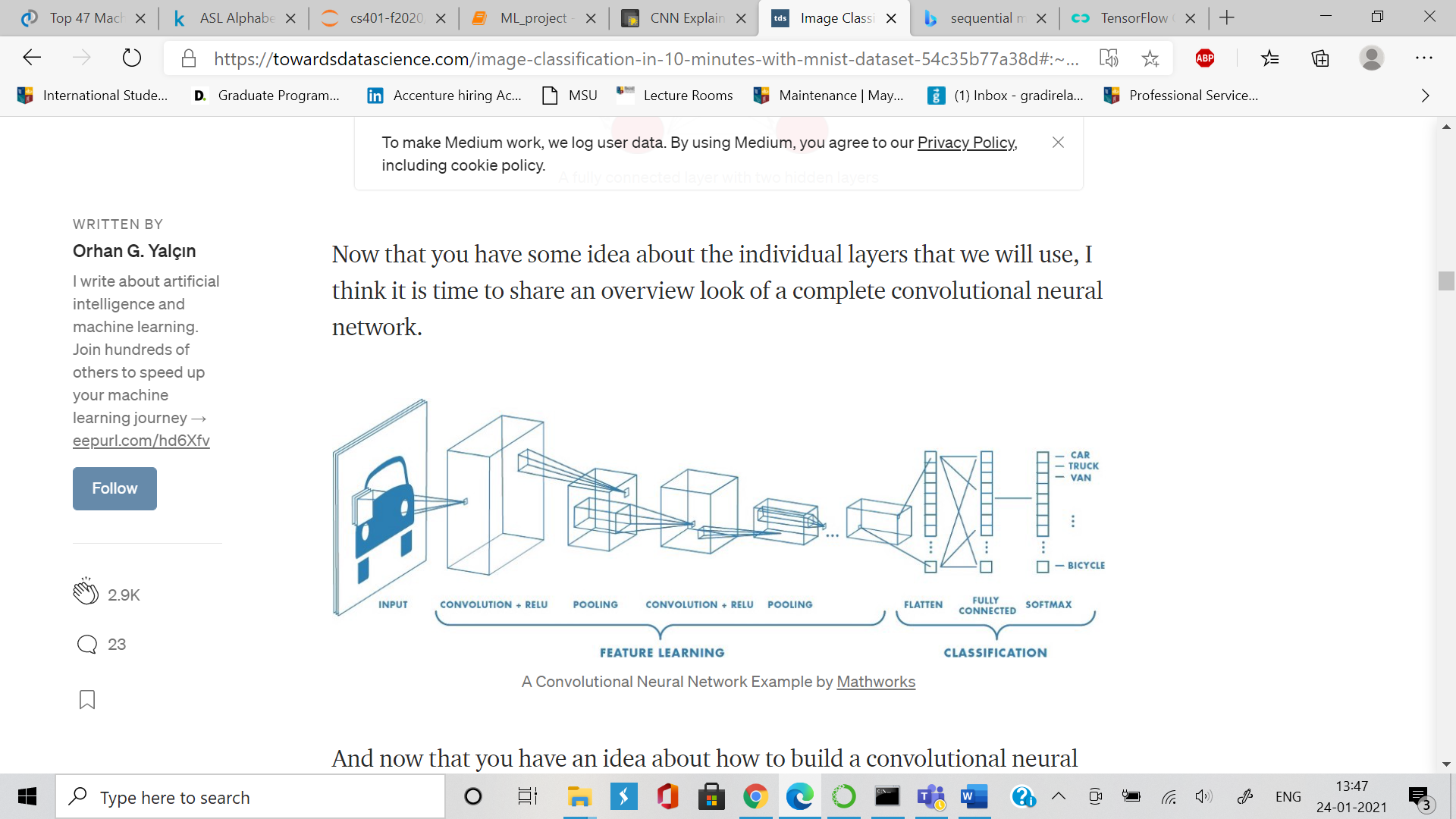
1. **A pooling layer or subsampling layer:** Pooling layers are inserted after each convolution layer in CNN which reduces the spatial size of the representation.

In addition, pooling layers also helps with the overfitting problem. Basically we select a pooling size to reduce the amount of the parameters by selecting the maximum, average, or sum values inside these pixels.

In short, this layer reduces the dimensionality of each feature to focus on most important elements.

1. **A set of fully connected layers:** It takes a flattened form of the features identified in the previous layers, and uses them to make a prediction about the image.

**A full CNN example(image)-**

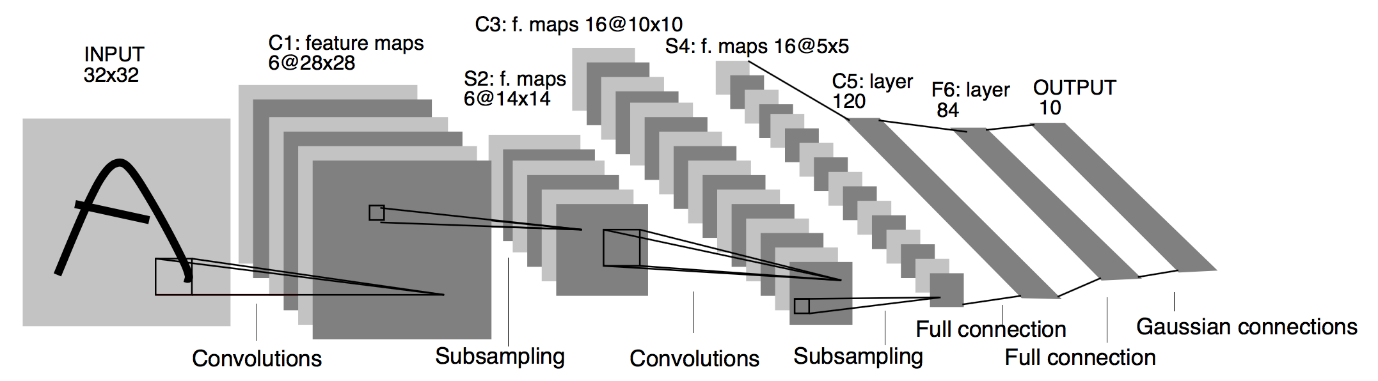


1. We tried to form our model using quite popular networks like **LeNET**, **AlexNET** and also thought of doing it using **VGG**. Let’s discuss each one of them one by one.

**LeNET**: LeNet-5, from the paper Gradient-Based Learning Applied to Document Recognition, is a very efficient convolutional neural network for handwritten character recognition (MNIST Dataset). LeNet5 is a small network, it contains the basic modules of deep learning: convolutional layer, pooling layer, and full link layer. It is the basis of other deep learning models.

**Layers of LeNET –**

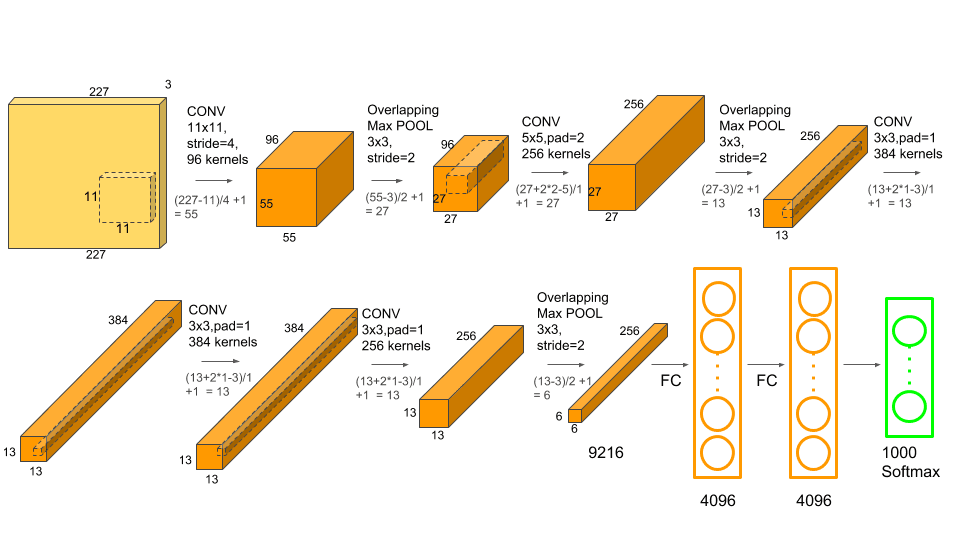
1. Convolution layer
2. Pooling layer (Down sampling layer)
3. Convolution layer
4. Pooling layer (Down sampling layer)
5. Convolution layer
6. Fully Connected layer
7. Output layer



**AlexNET:** AlexNet was designed by Hinton, winner of the 2012 ImageNet competition, and his student Alex Krizhevsky. **AlexNet** architecture consists of 5 convolutional layers, 3 max-pooling layers, 2 normalization layers, 2 fully connected layers, and 1 softmax layer.  Each convolutional layer consists of convolutional filters and a nonlinear activation function ReLU.

Layers of AlexNET –

1. 1st Convolution layer
2. Pooling layer
3. 2nd Convolution layer
4. Pooling layer
5. 3rd Convolution layer
6. 4th Convolution layer
7. 5th Convolution layer
8. Pooling layer
9. 1st Dense layer
10. 2nd Dense layer
11. 3rd Dense layer
12. Output layer



We executed our model using LeNET and got around 90% accuracy in 5 epochs. But due to computational limitation and so many number of parameters , we were unable to train our model using AlexNET and VGG. We trained it for around 6 hours using Alexnet and still it didn’t complete 1 epoch so we gave up on AlexNET and VGG.

**Output Model –** We saved our trained model in .h5 format which is easily deployable i.e. we can deploy this trained model on Heroku, Azure, GCP or AWS and can see it working in real time. We will surely require a little bit front end to see it working on cloud.

We can also see also the internal structure of model (i.e. layers, weights etc.) by uploading our model file here - <https://netron.app/>

**Overall Code:**

import tensorflow as tf

import keras

from keras.layers import Conv2D, MaxPooling2D

from keras.layers import Dense, Flatten

from keras.models import Sequential

from keras.preprocessing.image import ImageDataGenerator

**#creating model**

model = Sequential()

#Select 6 feature convolution kernels with a size of 5 \* 5 (without offset), and get 66 feature maps. The size of each feature map is 32−5 + 1 = 2832−5 + 1 = 28.

#That is, the number of neurons has been reduced from 10241024 to 28 ∗ 28 = 784 28 ∗ 28 = 784.

#Parameters between input layer and C1 layer: 6 ∗ (5 ∗ 5 + 1)

model.add(Conv2D(6, kernel\_size=(5, 5), activation='relu', input\_shape=(64, 64, 3)))

# The input of this layer is the output of the first layer, which is a 28 \* 28 \* 6 node matrix.

# The size of the filter used in this layer is 2 \* 2, and the step length and width are both 2, so the output matrix size of this layer is 14 \* 14 \* 6.

model.add(MaxPooling2D(pool\_size=(2, 2)))

# The input matrix size of this layer is 14 \* 14 \* 6, the filter size used is 5 \* 5, and the depth is 16. This layer does not use all 0 padding, and the step size is 1.

# The output matrix size of this layer is 10 \* 10 \* 16. This layer has 5 \* 5 \* 6 \* 16 + 16 = 2416 parameters

model.add(Conv2D(16, kernel\_size=(5, 5), activation='relu'))

# The input matrix size of this layer is 10 \* 10 \* 16. The size of the filter used in this layer is 2 \* 2, and the length and width steps are both 2, so the output matrix size of this layer is 5 \* 5 \* 16.

model.add(MaxPooling2D(pool\_size=(2, 2)))

# The input matrix size of this layer is 5 \* 5 \* 16. This layer is called a convolution layer in the LeNet-5 paper, but because the size of the filter is 5 \* 5, #

# So it is not different from the fully connected layer. If the nodes in the 5 \* 5 \* 16 matrix are pulled into a vector, then this layer is the same as the fully connected layer.

# The number of output nodes in this layer is 120, with a total of 5 \* 5 \* 16 \* 120 + 120 = 48120 parameters.

model.add(Flatten())

model.add(Dense(120, activation='relu'))

# The number of input nodes in this layer is 120 and the number of output nodes is 84. The total parameter is 120 \* 84 + 84 = 10164 (w + b)

model.add(Dense(84, activation='relu'))

# The number of input nodes in this layer is 84 and the number of output nodes is 10. The total parameter is 84 \* 10 + 10 = 850

model.add(Dense(26, activation='softmax'))

model.compile(loss=keras.metrics.sparse\_categorical\_crossentropy, optimizer=keras.optimizers.Adam(), metrics=['accuracy'])

**#training and testing**

train\_datagen = ImageDataGenerator(rescale = 1./255,

shear\_range = 0.2,

zoom\_range = 0.2,

horizontal\_flip = True)

test\_datagen = ImageDataGenerator(rescale = 1./255)

training\_set = train\_datagen.flow\_from\_directory('C:\\Users\\sneha\\OneDrive\\Desktop\\archive\\asl\_alphabet\_train\\train',

target\_size = (64, 64),

batch\_size = 32,

class\_mode = 'sparse')

test\_set = test\_datagen.flow\_from\_directory('C:\\Users\\sneha\\OneDrive\\Desktop\\archive\\asl\_alphabet\_test\\test',

target\_size = (64, 64),

batch\_size = 32,

class\_mode = 'sparse')

modelasl = model.fit\_generator(training\_set,

steps\_per\_epoch = 2000,

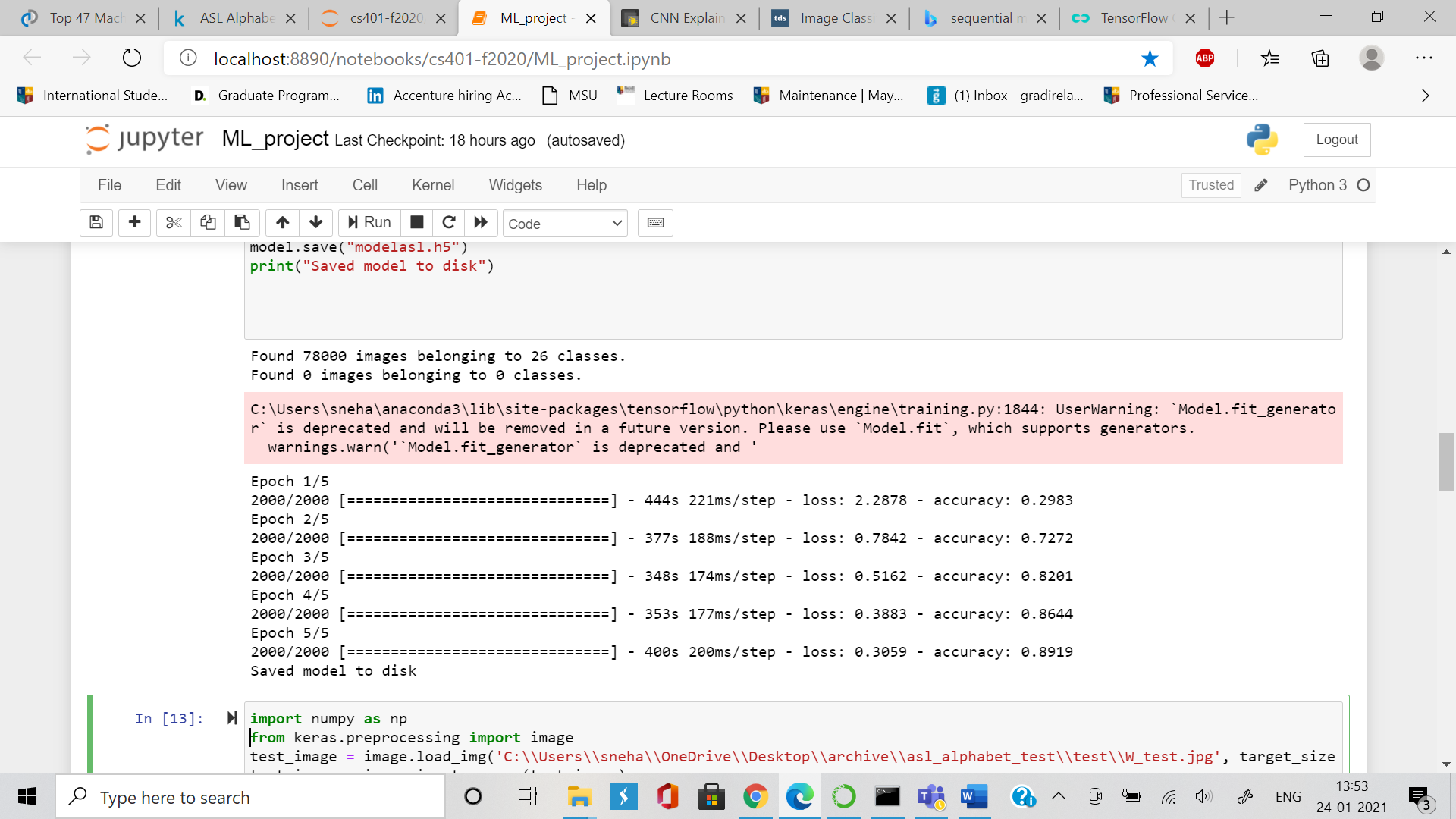
epochs = 5,

validation\_data = test\_set,

validation\_steps = 500)

model.save("modelasl.h5")

print("Saved model to disk")



import numpy as np

from keras.preprocessing import image

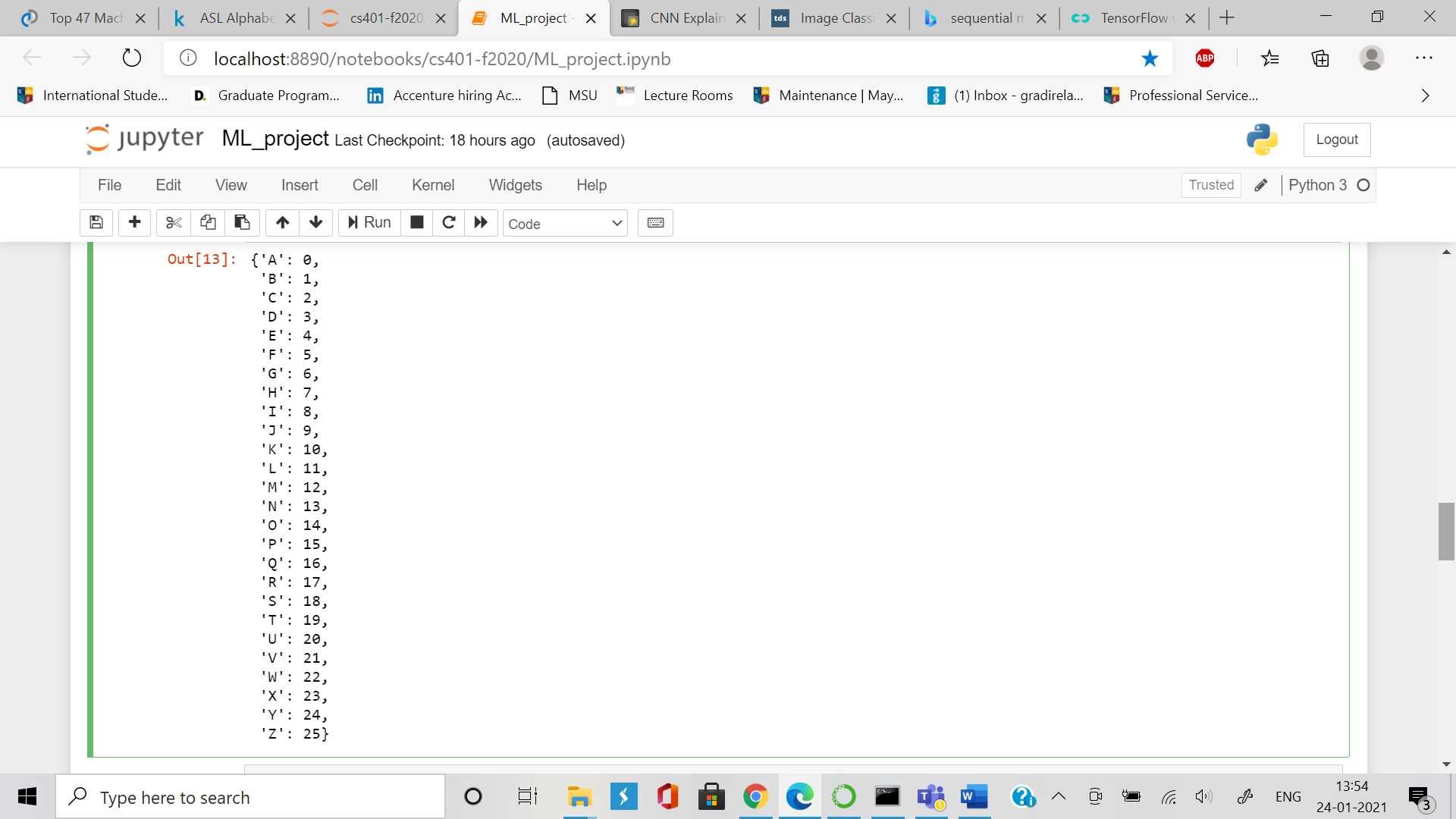
test\_image = image.load\_img('C:\\Users\\sneha\\OneDrive\\Desktop\\archive\\asl\_alphabet\_test\\test\\W\_test.jpg', target\_size = (64, 64))

test\_image = image.img\_to\_array(test\_image)

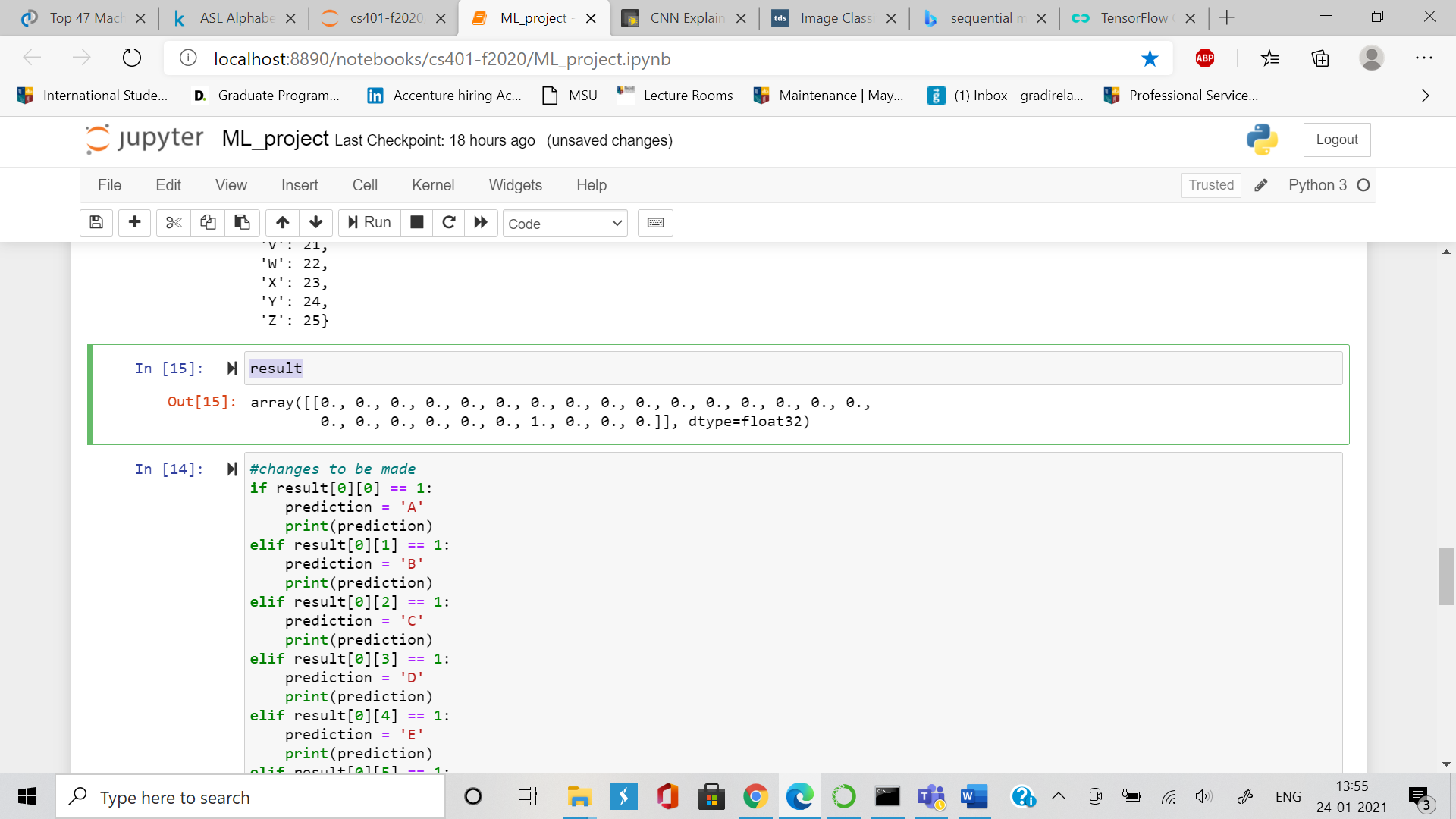
test\_image = np.expand\_dims(test\_image, axis = 0)

result = model.predict(test\_image)

training\_set.class\_indices



result



if result[0][0] == 1:

prediction = 'A'

print(prediction)

elif result[0][1] == 1:

prediction = 'B'

print(prediction)

elif result[0][2] == 1:

prediction = 'C'

print(prediction)

elif result[0][3] == 1:

prediction = 'D'

print(prediction)

elif result[0][4] == 1:

prediction = 'E'

print(prediction)

elif result[0][5] == 1:

prediction = 'F'

print(prediction)

elif result[0][6] == 1:

prediction = 'G'

print(prediction)

elif result[0][7] == 1:

prediction = 'H'

print(prediction)

elif result[0][8] == 1:

prediction = 'I'

print(prediction)

elif result[0][9] == 1:

prediction = 'J'

print(prediction)

elif result[0][10] == 1:

prediction = 'K'

print(prediction)

elif result[0][11] == 1:

prediction = 'L'

print(prediction)

elif result[0][12] == 1:

prediction = 'M'

print(prediction)

elif result[0][13] == 1:

prediction = 'N'

print(prediction)

elif result[0][14] == 1:

prediction = 'O'

print(prediction)

elif result[0][15] == 1:

prediction = 'P'

print(prediction)

elif result[0][16] == 1:

prediction = 'Q'

print(prediction)

elif result[0][17] == 1:

prediction = 'R'

print(prediction)

elif result[0][18] == 1:

prediction = 'S'

print(prediction)

elif result[0][19] == 1:

prediction = 'T'

print(prediction)

elif result[0][20] == 1:

prediction = 'U'

print(prediction)

elif result[0][21] == 1:

prediction = 'V'

print(prediction)

elif result[0][22] == 1:

prediction = 'W'

print(prediction)

elif result[0][23] == 1:

prediction = 'X'

print(prediction)

elif result[0][24] == 1:

prediction = 'Y'

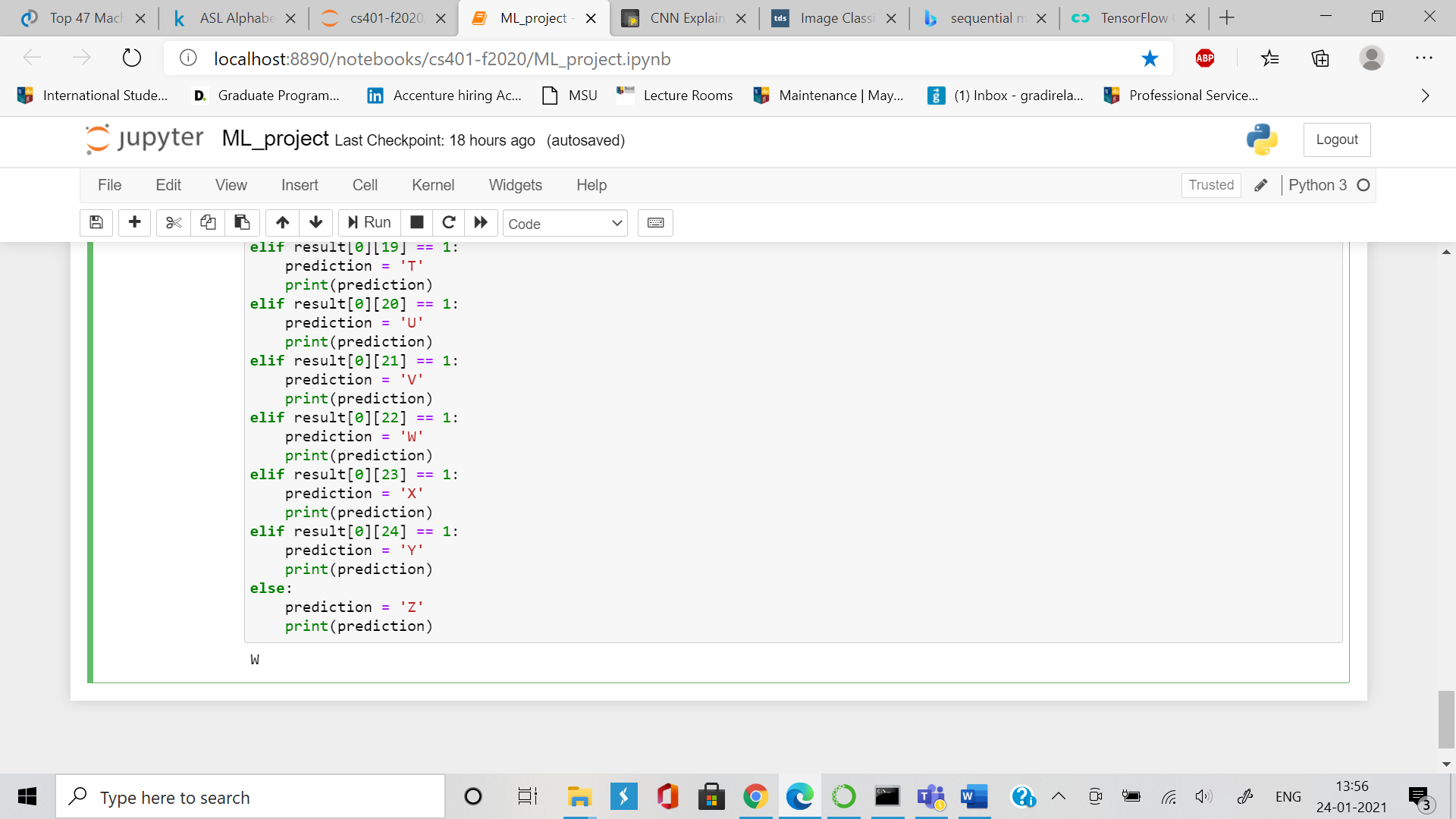
print(prediction)

else:

prediction = 'Z'

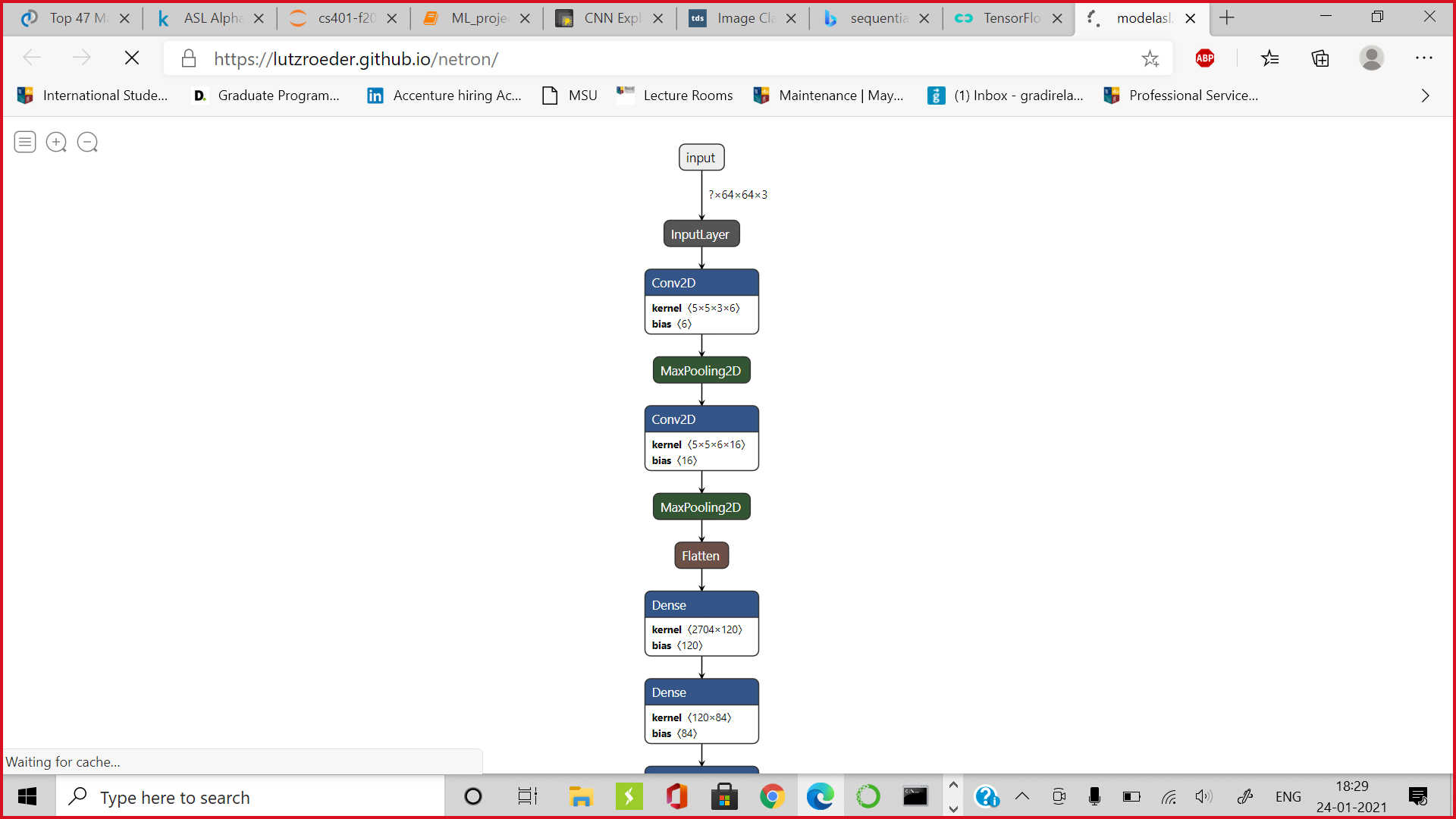
print(prediction)

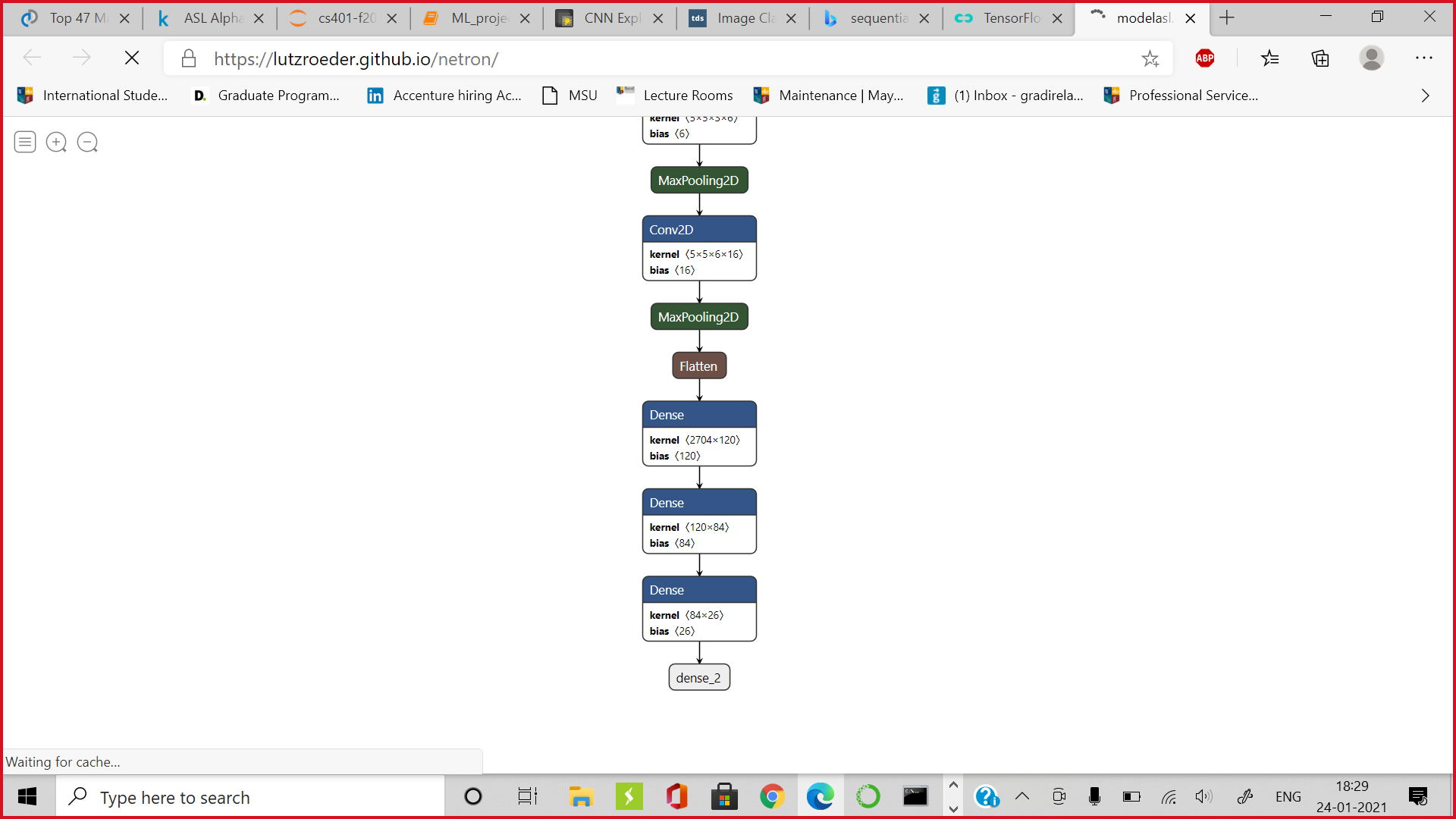
**Output:-**



Hence, we see that the image has predicted that the specified sign image is a ‘W’.

* **Generated Model:-**





* **Contributions-**

1. **Madhusudan Panwar: Developed model using LeNet.**
2. **Snehal Deshmukh: Tried AlexNet and made report.**