Sign_Language_Detector_v2

May 18, 2020

Reference: https://www.kaggle.com/rafaeletereo/convolutional-model

0.0.1 Importing Libraries

[1]:

```
import numpy as np
import time
import random
# data visualization and plotting imports
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

import os
from sklearn.utils.multiclass import unique_labels
import cv2
import matplotlib.pyplot as plt
import seaborn as sn

# deep learning imports
# import keras
import tensorflow
layers = tensorflow.keras.layers
BatchNormalization = tensorflow.keras.layers.BatchNormalization
Conv2D = tensorflow.keras.layers.Conv2D
Flatten = tensorflow.keras.layers.Flatten
MaxPooling2D = tensorflow.keras.layers.MaxPooling2D
Dropout = tensorflow.keras.layers.Dropout
Dense = tensorflow.keras.layers.Dense
ImageDataGenerator = tensorflow.keras.preprocessing.image.ImageDataGenerator
Sequential = tensorflow.keras.Sequential

TensorBoard = tensorflow.keras.callbacks.TensorBoard
ModelCheckpoint = tensorflow.keras.callbacks.ModelCheckpoint
Adam = tensorflow.keras.optimizers.Adam
regularizers = tensorflow.keras.regularizers
categorical_crossentropy = tensorflow.keras.losses
K = tensorflow.keras.backend
plot_model = tensorflow.keras.utils.plot_model
```

```


# word library import
from nltk.corpus import words


os.environ['KMP_DUPLICATE_LIB_OK']='True'
```

0.0.2 Global Variables

[2]:

```
# setting up global variables
DATADIR = "../master_thesis_project1/asl-alphabet/asl_alphabet_train/"  ␣
 ↪#training data directory
CATEGORIES = ['A', 'B', 'C', 'D', 'del', 'E', 'F', 'G', 'H', 'I', 'J', 'K',␣
 ↪'L', 'M',
              'N', 'nothing', 'O', 'P', 'Q', 'R', 'S', 'space', 'T', 'U', 'V',␣
 ↪'W', 'X', 'Y', 'Z']
test_dir = "../master_thesis_procet1/asl-alphabet/asl_alphabet_test"    ␣
 ↪#testing data directory
#own_dir = "../input/ishaan/ishaan_pics/ishaan_pics"
```

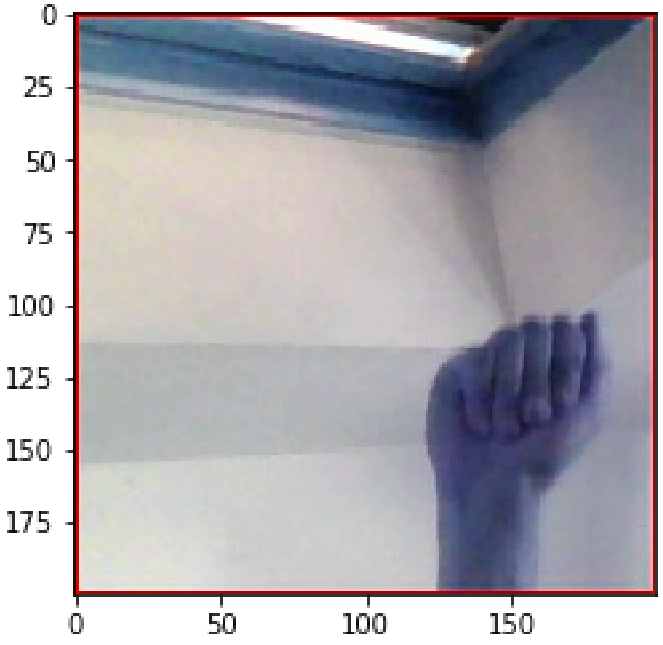
0.0.3 Getting Training Data

Testing experiment on the image array

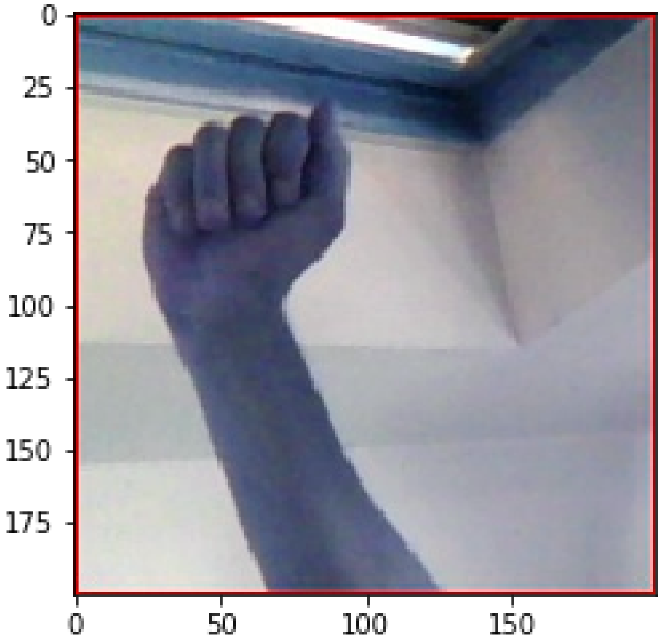
[3]:

```
from matplotlib import pyplot as plt
test_array = []
print('generating test_array data')
path = os.path.join(DATADIR, CATEGORIES[0])
print(path)
#sorted_files = sorted(os.listdir(path)[0:10])
for img_tst in sorted(os.listdir(path)[0:10]):
    img_tst_array = cv2.imread(os.path.join(path,img_tst), cv2.IMREAD_COLOR)
    print(img_tst)
    plt.imshow(img_tst_array)
    plt.show()
    test_array.append([img_tst_array, CATEGORIES.index('A')])
#print(test_array)
```

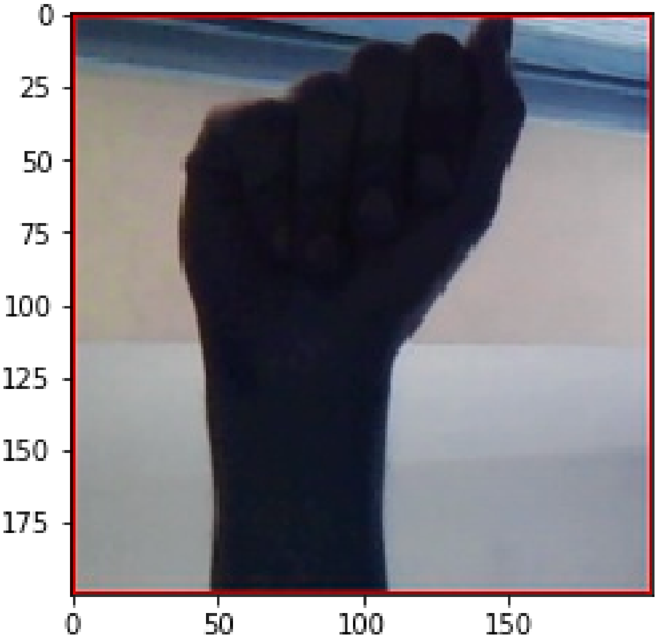
```
generating test_array data
../master_thesis_project1/asl-alphabet/asl_alphabet_train/A
A1382.jpg
```



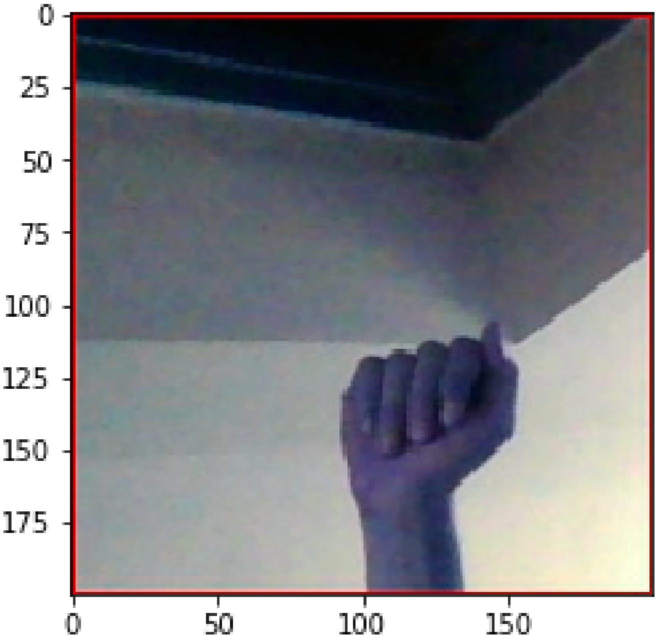
A1539.jpg



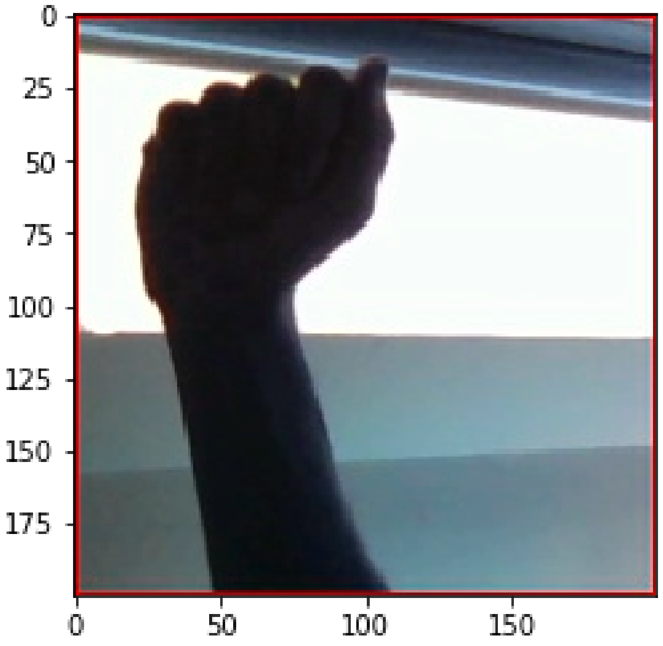
A166.jpg



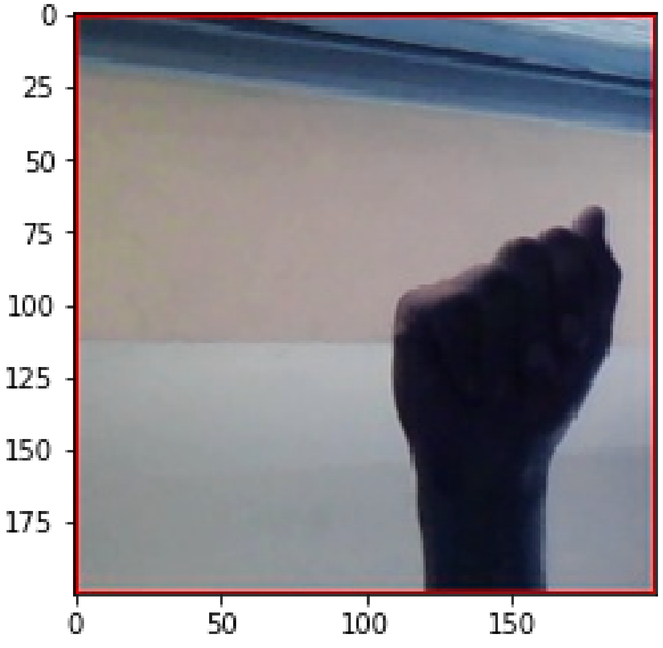
A2172.jpg



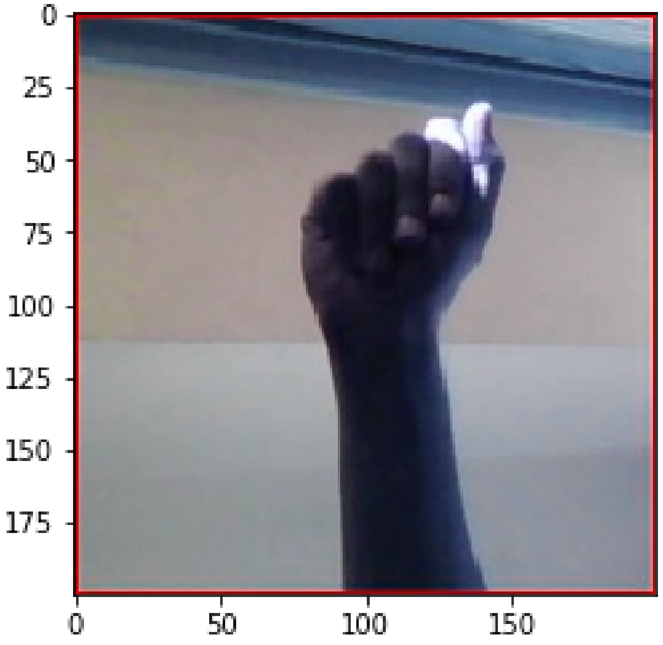
A2502.jpg



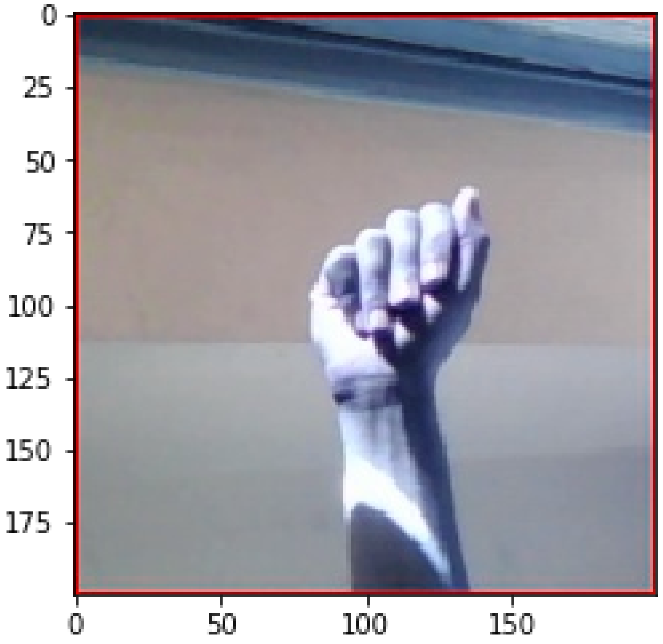
A412.jpg



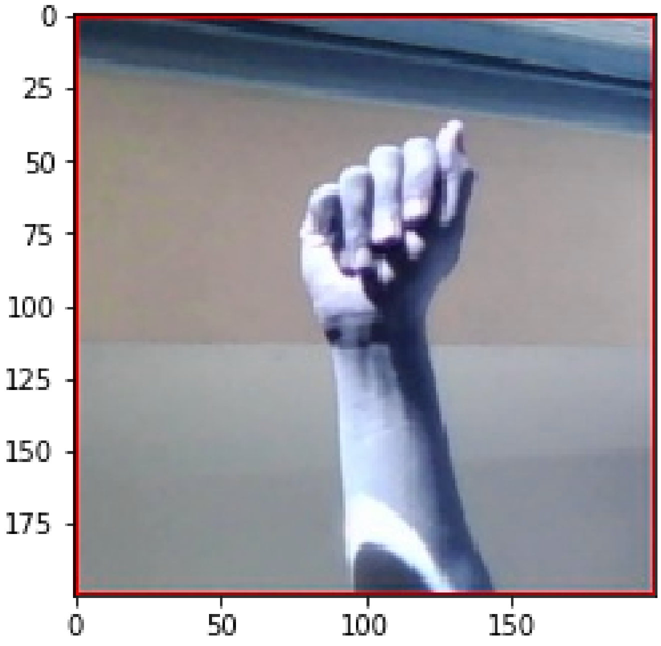
A469.jpg



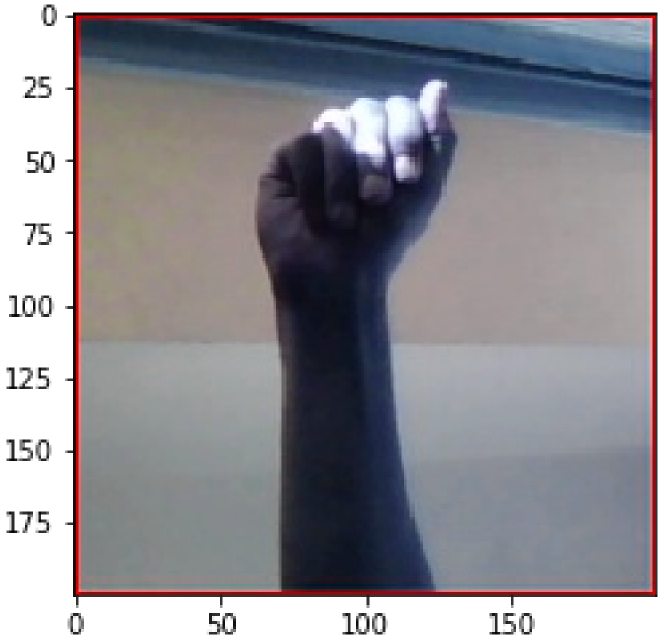
A505.jpg



A535.jpg



```
A679.jpg
```



[4]:
```
def create_training_data(modeltype):
    '''This function can run for each model in order to get the trainin data␣
 ↪from the filepath
        and convert it into array format'''
    training_data = []
    print('generating training data')
    if(modeltype == 'cnn'):
        for category in CATEGORIES:
            path = os.path.join(DATADIR, category)   #path to alphabets. e.g. ..
 ↪/../../asl_alphabet_train/A/
            class_num = CATEGORIES.index(category)
            for img in os.listdir(path):
                try:
                    img_array = cv2.imread(os.path.join(path,img), cv2.
 ↪IMREAD_COLOR)
                    new_array = cv2.resize(img_array, (64, 64))
                    final_img = cv2.cvtColor(new_array, cv2.COLOR_BGR2RGB)
                    training_data.append([final_img, class_num])
                except Exception as e:
                    pass
    else:
        for category in CATEGORIES:
            path = os.path.join(DATADIR, category)   #path to alphabets
            class_num = CATEGORIES.index(category)
            for img in os.listdir(path):
                try:
                    img_array = cv2.imread(os.path.join(path,img), cv2.
 ↪IMREAD_GRAYSCALE)
                    new_array = cv2.resize(img_array, (64, 64))
                    training_data.append([new_array, class_num])
                except Exception as e:
                    pass
    return training_data
```

0.0.4 Pre-processing Training Data

[5]:

```
def make_data(modeltype, training_data):
    '''This function formats the training data into the proper format and␣
 ↪passes it through an generator
    so that it can be augmented and fed into the model'''
    X = []
    y = []
    for features, label in training_data:
        X.append(features)
        y.append(label)
    if(modeltype == "cnn"):
        X = np.array(X).reshape(-1, 64, 64, 3)           #reshaping the array␣
 ↪into the 4-D.
        X = X.astype('float32')/255.0                    #to normalize data
        y = tensorflow.keras.utils.to_categorical(y)     #one-hot encoding
        y = np.array(y)
        datagen = ImageDataGenerator(
                                     validation_split = 0.1,
                                     rotation_range = 20,
                                     width_shift_range = 0.2,
                                     height_shift_range = 0.2,
                                     horizontal_flip = True)
        train_data = datagen.flow(X, y, batch_size=64, shuffle=True,␣
 ↪subset='training')
        val_data = datagen.flow(X, y, batch_size=64, shuffle=True,␣
 ↪subset='validation')
        return (train_data, val_data, X, y)
    else:
        X = np.array(X).flatten().reshape(-1, 4096)
        X = X.astype('float32')/255.0
        y = tensorflow.keras.utils.to_categorical(y)
        y = np.array(y)
        return (X, y)
```

0.1 The Model

[6]:

```
''' The author added a regularizer and BatchNormalization because, as you will␣
 ↪see below,
the model runs into problems with overfitting since all the training_data is␣
 ↪from one hand
and seems to be taken as a series of burst photos, which means that
it doesn't do well with data from other people's hands.
So the author added them in an attempt to reduce overfitting.'''

def build_model(modeltype):
    '''Builds the model based on the specified modeltype (either convolutional␣
 ↪or fully_connected)'''

    model = tensorflow.keras.Sequential()
    print('Building model', modeltype)

    if(modeltype == 'cnn'):
         ## CNN 4 layers
        model.add(Conv2D(64, kernel_size=(3, 3), activation='relu',␣
 ↪input_shape=(64, 64, 3)))
        model.add(BatchNormalization())
        model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
        model.add(BatchNormalization())
        model.add(Dropout(0.25))

        model.add(Conv2D(256, kernel_size=(3, 3), activation='relu'))
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        model.add(Flatten())

        model.add(Dense(256, activation='relu'))
        model.add(BatchNormalization())
        model.add(Dropout(0.25))

        model.add(Dense(29, activation='softmax'))


    else:
        model.add(layers.Conv2D(64, kernel_size=4, strides=1,␣
↪activation='relu', input_shape=(64,64,3)))
        model.add(layers.Conv2D(64, kernel_size=4, strides=2,␣
↪activation='relu'))
        model.add(Dropout(0.5))

        model.add(layers.Conv2D(128, kernel_size=4, strides=1,␣
↪activation='relu'))
        model.add(layers.Conv2D(128, kernel_size=4, strides=2,␣
↪activation='relu'))
        model.add(Dropout(0.5))

        model.add(Conv2D(256, kernel_size=4, strides=1, activation='relu'))
        model.add(Conv2D(256, kernel_size=4, strides=2, activation='relu'))

        model.add(BatchNormalization())

        model.add(Flatten())
        model.add(Dropout(0.5))
        model.add(Dense(512, activation='relu', kernel_regularizer =␣
↪regularizers.l2(0.001)))
        model.add(Dense(29, activation='softmax'))
    model.compile(optimizer = Adam(lr=0.0005), loss =␣
↪'categorical_crossentropy',
                  metrics = ["accuracy"]) # learning rate reduced to help␣
↪problems with overfitting
    return model
```

[7]:

```
def fit_fully_connected_model(X, y, model):
    '''fits the fully connected model'''

    filepath = "weights2.best.h5"

    # saving model weights with lowest validation loss to reduce overfitting
    checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1,␣
↪save_best_only=True, save_weights_only=False, mode='auto', period=1)
    #tensorboard
    tensorboard_callback = TensorBoard("logs")
    model.fit(X, y, epochs = 10, validation_split = 0.1, callbacks =␣
↪[checkpoint, tensorboard_callback])
```

[8]:

```
def fit_CNN_model(train_data, val_data, model):
    '''fits the CNN model'''

    filepath = "weights.mest.h5"

    # saving model weights with lowest validation loss to reduce overfitting
    checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1,␣
↪save_best_only=True, save_weights_only=False,
                                mode='auto', period=1)
    # tensorboard
    tensorboard_callback = TensorBoard("logs")

    # fitting model
    model.fit_generator(train_data, epochs=10, steps_per_epoch =1360,␣
↪validation_data = val_data,
                       validation_steps = len(val_data), callbacks =␣
↪[checkpoint, tensorboard_callback])
```

0.1.1 Data Visualization and Evaluation

[9]:

```
def show_classification_report(X, y, input_shape, model):
    '''This function prints a classification report for the validation data'''
    start_time = time.time()
    validation = [X[i] for i in range(int(0.1 * len(X)))]
    validation_labels = [np.argmax(y[i]) for i in range(int(0.1 * len(y)))]
    validation_preds = []
    labels = [i for i in range(29)]
    for img in validation:
        img = img.reshape((1,) + input_shape)
        pred = model.predict_classes(img)
        validation_preds.append(pred[0])
    print(classification_report(validation_labels, validation_preds, labels,␣
 ↪target_names = CATEGORIES))
    print("\n Evaluating the model took {:.0f} seconds".format(time.
 ↪time()-start_time))
    return (validation_labels, validation_preds)
```

[10]:

```
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting 'normalize=True'
    """

    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print("Confusion matrix, without normalization")


    # print(cm)

    fig, ax = plt.subplots(figsize=(20, 10))
    im = ax.imshow(cm, interpolation = 'nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",␣
 ↪rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax
np.set_printoptions(precision=2)
```

[30]:

```
def rotate_image(img):
    '''This function will be applied to the given test data to see how rotating␣
 ↪the data effects prediction accuracy.
       It rotates it in a way such that no part of the image is lost'''
    (h, w) = img.shape[:2]

    # calculate the center of the image
    center = (w/2, h/2)

    angle90 = 90
    angle180=180
    angle270=270

    scale = 1.0

    # Perform the counter clockwise rotation holding at the center
    # 90 degrees
    M = cv2.getRotationMatrix2D(center, angle90, scale)
    rotated90 = cv2.warpAffine(img, M, (h, w))

    # 180 degrees
    M = cv2.getRotationMatrix2D(center, angle180, scale)
    rotated180 = cv2.warpAffine(img, M, (h, w))

    # 270 degrees
    M = cv2.getRotationMatrix2D(center, angle270, scale)
    rotated270 = cv2.warpAffine(img, M, (h, w))
    return(rotated90, rotated180, rotated270)
```

0.1.2 Testing data and predictions

[32]:

```
def create_testing_data(path, input_shape, modeltype):
    '''This function will get and format the testing data from the dataset
       It works in almost the exact same way as training_data except it returns␣
 ↪image names to evaluate predictions'''
    testing_data = []
    names = []
    for img in os.listdir(path):
        if(modeltype == 'cnn'):
            img_array = cv2.imread(os.path.join(path,img), cv2.IMREAD_COLOR)
            rotated_90, rotated_180, rotated_270 = rotate_image(img_array)  #␣
 ↪in order to test predictions for rotated data
            imgs = [img_array, rotated_90, rotated_180, rotated_270]
            final_imgs = []
            for image in imgs:
                new_array = cv2.resize(image, (64, 64))
                final_img = cv2.cvtColor(new_array, cv2.COLOR_BGR2RGB)
                final_imgs.append(final_img)
        else:
            img_array = cv2.imread(os.path.join(path,img), cv2.IMREAD_GRAYSCALE)
            rotated_90, rotated_180, rotated_270 = rotate_image(img_array)
            imgs = [img_array, rotated_90, rotated_180, rotated_270]
            final_imgs = []
            for image in imgs:
                final_img = cv2.resize(image, (64, 64))
                final_imgs.append(final_img)
        # print(len(final_imgs))
        for final_img in final_imgs:
            testing_data.append(final_img)
            names.append(img)

    if modeltype == 'cnn':
        new_testing_data = np.array(testing_data).reshape((-1,) + input_shape)
    else:
        new_testing_data = np.array(testing_data).flatten().reshape((-1,) +␣
 ↪input_shape)
    new_testing_data = new_testing_data.astype('float32')/255.0
    return (testing_data, new_testing_data, names)


def prediction_generator(testing_data, input_shape, model):
    '''This function generates predictions for sets of testing data'''
    predictions = []
    for img in testing_data:
        img = img.reshape((1,) + input_shape)
        pred = model.predict_classes(img)
        predictions.append(pred[0])
    predictions = np.array(predictions)
    return predictions
```

[34]:
```
def plot_predictions(testing_data, predictions, names):
    '''This function plots the testing data predictions along with the actual␣
 ↪letter they represent
       so we can see the accuracy of the model.'''
    fig = plt.figure(figsize = (100, 100))
    fig.subplots_adjust(hspace = 0.8, wspace = 0.5)

    index = 0
    for i in range(1, len(testing_data)):
        y = fig.add_subplot(12, np.ceil(len(testing_data)/float(12)), i)

        str_label = CATEGORIES[predictions[index]]
        y.imshow(testing_data[index], cmap = 'gray')
        if(index%4==0):
            title = "prediction = {}\n {}\n unrotated".format(str_label,␣
 ↪names[index])
        else:
            title = "prediction = {}\n {}".format(str_label,names[index])
        y.set_title(title, fontsize = 60)
        y.axes.get_xaxis().set_visible(False)
        y.axes.get_yaxis().set_visible(False)
        index+=1
```

[14]:
```
def calculate_loss(names, predictions):
    y_true = K.variable(np.array([CATEGORIES.index(name[0].upper()) for name in␣
 ↪names]))
    y_pred = K.variable(np.array(predictions))
    print(y_true)
    print(y_pred)
    error = K.eval(categorical_crossentropy(y_true, y_pred))
    print('Loss:', error)
```

0.1.3 TensorBoard

[15]:
```
%load_ext tensorboard
%tensorboard --logdir logs
```

```
<IPython.core.display.HTML object>
```

0.2 Convolutional Neural Network

[16]:
```
modeltype2 = "cnn"
input_shape2 = 64, 64, 3

# getting training data
training_data2 = create_training_data(modeltype2)
random.shuffle(training_data2)
```

```
generating training data
```

[17]:
```
# building model
model2 = build_model(modeltype2)

# formatting data
train_data2, val_data2, X2, y2 = make_data(modeltype2, training_data2)

# fitting model
fit_CNN_model(train_data2, val_data2, model2)
model2.load_weights("weights.best.h5")
graph2 = plot_model(model2, to_file="my_model2.png", show_shapes=True)
```

```
Building model cnn
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to
specify the frequency in number of samples seen.
Epoch 1/10
1359/1360 [============================>.] - ETA: 5s - loss: 1.7498 - accuracy:
0.4704
Epoch 00001: val_loss improved from inf to 1.56687, saving model to
weights.mest.h5
1360/1360 [==============================] - 7219s 5s/step - loss: 1.7490 -
accuracy: 0.4707 - val_loss: 1.5669 - val_accuracy: 0.5638
Epoch 2/10
1359/1360 [============================>.] - ETA: 5s - loss: 0.5224 - accuracy:
0.8294
Epoch 00002: val_loss improved from 1.56687 to 0.52231, saving model to
weights.mest.h5
1360/1360 [==============================] - 7286s 5s/step - loss: 0.5223 -
accuracy: 0.8295 - val_loss: 0.5223 - val_accuracy: 0.8262
Epoch 3/10
1359/1360 [============================>.] - ETA: 5s - loss: 0.2693 - accuracy:
0.9114
Epoch 00003: val_loss did not improve from 0.52231
1360/1360 [==============================] - 7286s 5s/step - loss: 0.2692 -
accuracy: 0.9115 - val_loss: 0.7338 - val_accuracy: 0.7993
Epoch 4/10
1359/1360 [============================>.] - ETA: 5s - loss: 0.1857 - accuracy:
```

```
0.9389
Epoch 00004: val_loss improved from 0.52231 to 0.38350, saving model to
weights.mest.h5
1360/1360 [==============================] - 7426s 5s/step - loss: 0.1857 -
accuracy: 0.9389 - val_loss: 0.3835 - val_accuracy: 0.8733
Epoch 5/10
1359/1360 [============================>.] - ETA: 5s - loss: 0.1431 - accuracy:
0.9530
Epoch 00005: val_loss improved from 0.38350 to 0.08836, saving model to
weights.mest.h5
1360/1360 [==============================] - 7428s 5s/step - loss: 0.1431 -
accuracy: 0.9530 - val_loss: 0.0884 - val_accuracy: 0.9701
Epoch 6/10
1359/1360 [============================>.] - ETA: 5s - loss: 0.1192 - accuracy:
0.9603
Epoch 00006: val_loss did not improve from 0.08836
1360/1360 [==============================] - 7401s 5s/step - loss: 0.1192 -
accuracy: 0.9603 - val_loss: 0.1572 - val_accuracy: 0.9471
Epoch 7/10
1359/1360 [============================>.] - ETA: 5s - loss: 0.1041 - accuracy:
0.9645
Epoch 00007: val_loss did not improve from 0.08836
1360/1360 [==============================] - 7373s 5s/step - loss: 0.1041 -
accuracy: 0.9645 - val_loss: 0.1752 - val_accuracy: 0.9456
Epoch 8/10
1359/1360 [============================>.] - ETA: 5s - loss: 0.0889 - accuracy:
0.9702
Epoch 00008: val_loss did not improve from 0.08836
1360/1360 [==============================] - 7351s 5s/step - loss: 0.0890 -
accuracy: 0.9702 - val_loss: 0.0886 - val_accuracy: 0.9697
Epoch 9/10
1359/1360 [============================>.] - ETA: 5s - loss: 0.0782 - accuracy:
0.9741
Epoch 00009: val_loss did not improve from 0.08836
1360/1360 [==============================] - 7485s 6s/step - loss: 0.0782 -
accuracy: 0.9741 - val_loss: 0.7596 - val_accuracy: 0.8209
Epoch 10/10
1359/1360 [============================>.] - ETA: 5s - loss: 0.0745 - accuracy:
0.9747
Epoch 00010: val_loss did not improve from 0.08836
1360/1360 [==============================] - 7380s 5s/step - loss: 0.0745 -
accuracy: 0.9747 - val_loss: 0.2741 - val_accuracy: 0.9208
```

```
       ␣
 ↪---------------------------------------------------------------------------
```

```
       OSError                                   Traceback (most recent call␣
↪last)

       <ipython-input-17-4a12eba82e62> in <module>
         7 # fitting model
         8 fit_CNN_model(train_data2, val_data2, model2)
   ----> 9 model2.load_weights("weights.best.h5")
        10 graph2 = plot_model(model2, to_file="my_model2.png",␣
↪show_shapes=True)


       ~/anaconda3/lib/python3.7/site-packages/tensorflow_core/python/keras/
↪engine/training.py in load_weights(self, filepath, by_name)
       179         raise ValueError('Load weights is not yet supported with␣
↪TPUStrategy '
       180                          'with steps_per_run greater than 1.')
   --> 181     return super(Model, self).load_weights(filepath, by_name)
       182
       183   @trackable.no_automatic_dependency_tracking


       ~/anaconda3/lib/python3.7/site-packages/tensorflow_core/python/keras/
↪engine/network.py in load_weights(self, filepath, by_name)
      1169           'first, then load the weights.')
      1170     self._assert_weights_created()
   -> 1171     with h5py.File(filepath, 'r') as f:
      1172       if 'layer_names' not in f.attrs and 'model_weights' in f:
      1173         f = f['model_weights']


       ~/anaconda3/lib/python3.7/site-packages/h5py/_hl/files.py in␣
↪__init__(self, name, mode, driver, libver, userblock_size, swmr, **kwds)
       310             with phil:
       311                 fapl = make_fapl(driver, libver, **kwds)
   --> 312                 fid = make_fid(name, mode, userblock_size, fapl,␣
↪swmr=swmr)
       313
       314                 if swmr_support:


       ~/anaconda3/lib/python3.7/site-packages/h5py/_hl/files.py in␣
↪make_fid(name, mode, userblock_size, fapl, fcpl, swmr)
       140         if swmr and swmr_support:
       141             flags |= h5f.ACC_SWMR_READ
   --> 142         fid = h5f.open(name, flags, fapl=fapl)
       143     elif mode == 'r+':
       144         fid = h5f.open(name, h5f.ACC_RDWR, fapl=fapl)
```

```
        h5py/_objects.pyx in h5py._objects.with_phil.wrapper()


        h5py/_objects.pyx in h5py._objects.with_phil.wrapper()


        h5py/h5f.pyx in h5py.h5f.open()


        OSError: Unable to open file (unable to open file: name = 'weights.best.
↪h5', errno = 2, error message = 'No such file or directory', flags = 0,␣
↪o_flags = 0)
```

[18]:
```
model2.save('tfcnnmodel2')
```

```
WARNING:tensorflow:From /home/samir/anaconda3/lib/python3.7/site-
packages/tensorflow_core/python/ops/resource_variable_ops.py:1781: calling
BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops)
with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
INFO:tensorflow:Assets written to: tfcnnmodel2/assets
```

[19]:
```
model2_json = model2.to_json()
with open("model2.json", "w") as json_file:
    json_file.write(model2_json)
# serialize weights to HDF5
model2.save_weights("model2.h5")
print("Saved model2 to disk")
```

```
Saved model2 to disk
```

[21]:
```
model2.save("cnnmodel2.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

[22]:
```
# evaluating validation data
validation_labels2, validation_preds2 = show_classification_report(X2, y2,␣
↪input_shape2, model2)
```

```
              precision    recall  f1-score   support
```

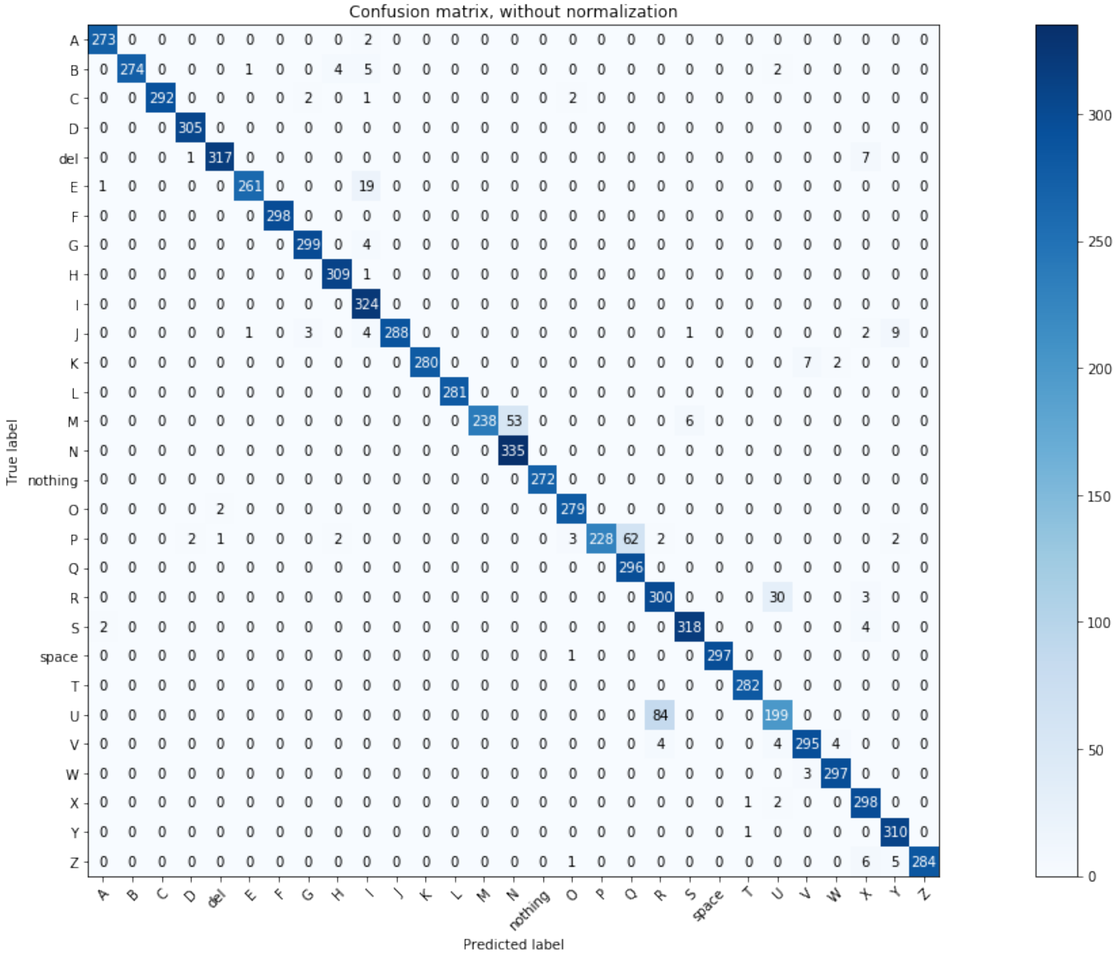
```
           A       0.99      0.99      0.99       275
           B       1.00      0.96      0.98       286
           C       1.00      0.98      0.99       297
           D       0.99      1.00      1.00       305
         del       0.99      0.98      0.98       325
           E       0.99      0.93      0.96       281
           F       1.00      1.00      1.00       298
           G       0.98      0.99      0.99       303
           H       0.98      1.00      0.99       310
           I       0.90      1.00      0.95       324
           J       1.00      0.94      0.97       308
           K       1.00      0.97      0.98       289
           L       1.00      1.00      1.00       281
           M       1.00      0.80      0.89       297
           N       0.86      1.00      0.93       335
     nothing       1.00      1.00      1.00       272
           O       0.98      0.99      0.98       281
           P       1.00      0.75      0.86       302
           Q       0.83      1.00      0.91       296
           R       0.77      0.90      0.83       333
           S       0.98      0.98      0.98       324
       space       1.00      1.00      1.00       298
           T       0.99      1.00      1.00       282
           U       0.84      0.70      0.77       283
           V       0.97      0.96      0.96       307
           W       0.98      0.99      0.99       300
           X       0.93      0.99      0.96       301
           Y       0.95      1.00      0.97       311
           Z       1.00      0.96      0.98       296

    accuracy                           0.96      8700
   macro avg       0.96      0.96      0.96      8700
weighted avg       0.96      0.96      0.96      8700
```

```
 Evaluating the model took 736 seconds
```

[23]:

```python
# confusion matrix for validation data
plot_confusion_matrix(validation_labels2, validation_preds2, classes=CATEGORIES,
                      title='Confusion matrix, without normalization')
plt.show()
```

```
Confusion matrix, without normalization
```



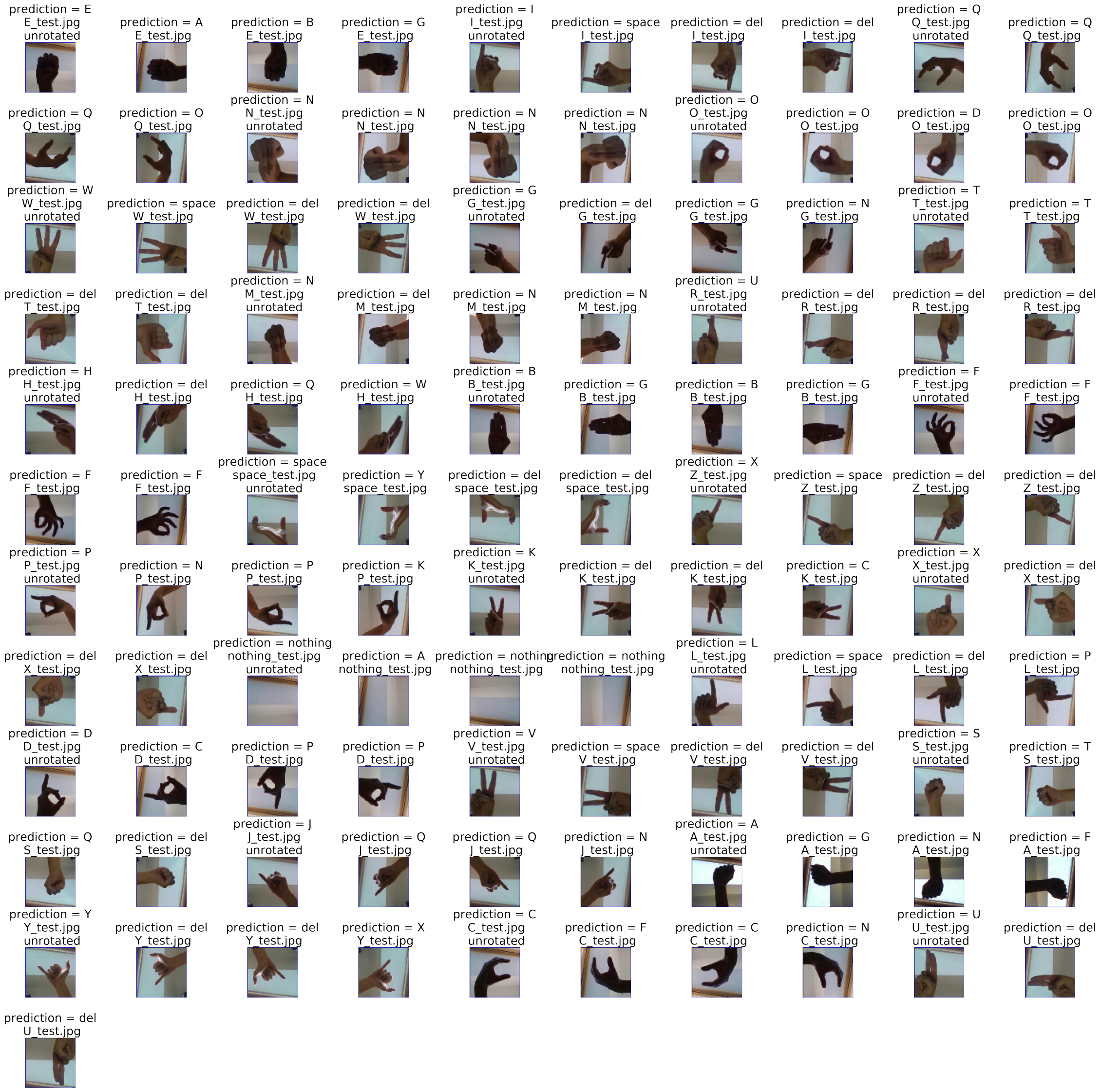
	A	B	C	D	del	E	F	G	H	I	J	K	L	M	N	nothing	O	P	Q	R	S	space	T	U	V	W	X	Y	Z
A	273	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	274	0	0	0	1	0	0	4	5	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
C	0	0	292	0	0	0	0	2	0	1	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	305	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
del	0	0	0	1	317	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0
E	1	0	0	0	0	261	0	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	298	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	299	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	309	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	324	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	1	0	3	0	4	288	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	2	9	0
K	0	0	0	0	0	0	0	0	0	0	0	280	0	0	0	0	0	0	0	0	0	0	0	0	7	2	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	0	281	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0	0	238	53	0	0	0	0	0	6	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	335	0	0	0	0	0	0	0	0	0	0	0	0	0	0
nothing	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	272	0	0	0	0	0	0	0	0	0	0	0	0	0
O	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	279	0	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	2	1	0	0	0	2	0	0	0	0	0	0	0	3	228	62	2	0	0	0	0	0	0	0	2	0
Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	296	0	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	300	0	0	0	30	0	0	3	0	0
S	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	318	0	0	0	0	0	4	0	0
space	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	297	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	282	0	0	0	0	0	0
U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	84	0	0	0	199	0	0	0	0	0
V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	4	295	4	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	297	0	0	0
X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	298	0	0
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	310	0
Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	6	5	284

```
[35]: # database testing data and predictions
      test_dir = "../master_thesis_project1/asl-alphabet/asl_alphabet_test/"
      testing_data2, new_testing_data2, names2 = create_testing_data(test_dir,␣
       ↪input_shape2, modeltype2)
      predictions2 = prediction_generator(new_testing_data2, input_shape2, model2)
      plot_predictions(testing_data2, predictions2, names2)
      calculate_loss(names2, predictions2)
```

```
<tf.Variable 'Variable:0' shape=(112,) dtype=float32, numpy=
array([ 5.,  5.,  5.,  5.,  9.,  9.,  9.,  9., 18., 18., 18., 18., 14.,
       14., 14., 14., 16., 16., 16., 16., 25., 25., 25., 25.,  7.,  7.,
        7.,  7., 22., 22., 22., 22., 13., 13., 13., 13., 19., 19., 19.,
       19.,  8.,  8.,  8.,  8.,  1.,  1.,  1.,  1.,  6.,  6.,  6.,  6.,
       20., 20., 20., 20., 28., 28., 28., 28., 17., 17., 17., 17., 11.,
       11., 11., 11., 26., 26., 26., 26., 14., 14., 14., 14., 12., 12.,
       12., 12.,  3.,  3.,  3.,  3., 24., 24., 24., 24., 20., 20., 20.,
       20., 10., 10., 10., 10.,  0.,  0.,  0.,  0., 27., 27., 27., 27.,
        2.,  2.,  2.,  2., 23., 23., 23., 23.], dtype=float32)>
```

```
<tf.Variable 'Variable:0' shape=(112,) dtype=float32, numpy=
array([ 5.,  0.,  1.,  7.,  9., 21.,  4.,  4., 18., 18., 18., 16., 14.,
       14., 14., 14., 16., 16.,  3., 16., 25., 21.,  4.,  4.,  7.,  4.,
        7., 14., 22., 22.,  4.,  4., 14.,  4., 14., 14., 23.,  4.,  4.,
        4.,  8.,  4., 18., 25.,  1.,  7.,  1.,  7.,  6.,  6.,  6.,  6.,
       21., 27.,  4.,  4., 26., 21.,  4.,  4., 17., 14., 17., 11., 11.,
        4.,  4.,  2., 26.,  4.,  4.,  4., 15.,  0., 15., 15., 12., 21.,
        4., 17.,  3.,  2., 17., 17., 24., 21.,  4.,  4., 20., 22., 18.,
        4., 10., 18., 18., 14.,  0.,  7., 14.,  6., 27.,  4.,  4., 26.,
        2.,  6.,  2., 14., 23.,  4.,  4.,  4.], dtype=float32)>
```

```
       ␣
 ↪---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call␣
 ↪last)

        <ipython-input-35-cbe7d08c1762> in <module>
          4 predictions2 = prediction_generator(new_testing_data2, input_shape2,␣
 ↪model2)
          5 plot_predictions(testing_data2, predictions2, names2)
    ----> 6 calculate_loss(names2, predictions2)


        <ipython-input-14-8fd6d4977a6b> in calculate_loss(names, predictions)
          4     print(y_true)
          5     print(y_pred)
    ----> 6     error = K.eval(categorical_crossentropy(y_true, y_pred))
          7     print('Loss:', error)


        TypeError: 'module' object is not callable
```


prediction = E
E_test.jpg
unrotated
prediction = A
E_test.jpg
prediction = B
E_test.jpg
prediction = G
E_test.jpg
prediction = I
I_test.jpg
unrotated
prediction = space
I_test.jpg
prediction = del
I_test.jpg
prediction = del
I_test.jpg
prediction = Q
Q_test.jpg
unrotated
prediction = Q
Q_test.jpg
prediction = Q
Q_test.jpg
prediction = O
Q_test.jpg
prediction = N
N_test.jpg
unrotated
prediction = N
N_test.jpg
prediction = N
N_test.jpg
prediction = N
N_test.jpg
prediction = O
O_test.jpg
unrotated
prediction = O
O_test.jpg
prediction = D
O_test.jpg
prediction = O
O_test.jpg
prediction = W
W_test.jpg
unrotated
prediction = space
W_test.jpg
prediction = del
W_test.jpg
prediction = del
W_test.jpg
prediction = G
G_test.jpg
unrotated
prediction = del
G_test.jpg
prediction = G
G_test.jpg
prediction = N
G_test.jpg
prediction = T
T_test.jpg
unrotated
prediction = T
T_test.jpg
prediction = del
T_test.jpg
prediction = del
T_test.jpg
prediction = N
M_test.jpg
unrotated
prediction = del
M_test.jpg
prediction = N
M_test.jpg
prediction = N
M_test.jpg
prediction = U
R_test.jpg
unrotated
prediction = del
R_test.jpg
prediction = del
R_test.jpg
prediction = del
R_test.jpg
prediction = H
H_test.jpg
unrotated
prediction = del
H_test.jpg
prediction = Q
H_test.jpg
prediction = W
H_test.jpg
prediction = B
B_test.jpg
unrotated
prediction = G
B_test.jpg
prediction = B
B_test.jpg
prediction = G
B_test.jpg
prediction = F
F_test.jpg
unrotated
prediction = F
F_test.jpg
prediction = F
F_test.jpg
prediction = F
F_test.jpg
prediction = space
space_test.jpg
unrotated
prediction = Y
space_test.jpg
prediction = del
space_test.jpg
prediction = del
space_test.jpg
prediction = X
Z_test.jpg
unrotated
prediction = space
Z_test.jpg
prediction = del
Z_test.jpg
prediction = del
Z_test.jpg
prediction = P
P_test.jpg
unrotated
prediction = N
P_test.jpg
prediction = P
P_test.jpg
prediction = K
P_test.jpg
prediction = K
K_test.jpg
unrotated
prediction = del
K_test.jpg
prediction = del
K_test.jpg
prediction = C
K_test.jpg
prediction = X
X_test.jpg
unrotated
prediction = del
X_test.jpg
prediction = del
X_test.jpg
prediction = del
X_test.jpg
prediction = nothing
nothing_test.jpg
unrotated
prediction = A
nothing_test.jpg
prediction = nothing
nothing_test.jpg
prediction = nothing
nothing_test.jpg
prediction = L
L_test.jpg
unrotated
prediction = space
L_test.jpg
prediction = del
L_test.jpg
prediction = P
L_test.jpg
prediction = D
D_test.jpg
unrotated
prediction = C
D_test.jpg
prediction = P
D_test.jpg
prediction = P
D_test.jpg
prediction = V
V_test.jpg
unrotated
prediction = space
V_test.jpg
prediction = del
V_test.jpg
prediction = del
V_test.jpg
prediction = S
S_test.jpg
unrotated
prediction = T
S_test.jpg
prediction = Q
S_test.jpg
prediction = del
S_test.jpg
prediction = J
J_test.jpg
unrotated
prediction = Q
J_test.jpg
prediction = Q
J_test.jpg
prediction = N
J_test.jpg
prediction = A
A_test.jpg
unrotated
prediction = G
A_test.jpg
prediction = N
A_test.jpg
prediction = F
A_test.jpg
prediction = Y
Y_test.jpg
unrotated
prediction = del
Y_test.jpg
prediction = del
Y_test.jpg
prediction = X
Y_test.jpg
prediction = C
C_test.jpg
unrotated
prediction = F
C_test.jpg
prediction = C
C_test.jpg
prediction = N
C_test.jpg
prediction = U
U_test.jpg
unrotated
prediction = del
U_test.jpg
prediction = del
U_test.jpg

[ ]: