# Constructing a sign language images classifier

*Elba Fernández López, María Loureiro Casalderrey*

University Carlos III de Madrid

## Abstract

The communication of sign language speakers is difficulted by the lack of population who masters this language. The existence of a translator that helps non-speakers understand might be a very useful tool and greatly improve their quality of life of hearing-impaired people. This paper presents the development and analysis of an algorithm which classifies real signed letter images as written letters. Three datasets have been used to successfully develop the algorithm. Two of them already existed and another one was created for this project.

**Key words**: ASL, Sign Language, Resnet, CNN, datasets, Data Augmentation, PyTorch

## 1. Introduction

According to the OMS, 5% of the world's population suffers from some kind of hearing loss [1]. In Spain, approximately 13.300 people communicate using sign language. These are the people that have trouble communicating with the rest of the population, who do not use dactylology. For this reason, it would be interesting to develop a digital translator to translate from sign language to written language in order to allow communication between these two groups of people. A first approach would consist in developing an alphabet sign language translator.

Nowadays there are simple alphabetic translators on the internet, which mainly focus on translating from written language to sign language by using drawn representations of each letter. However, there are not in the opposite direction, translating from real hand gestures to written text.

Our aim for this project was to develop an algorithm which receives as an input images with the dactylological alphabet and returns the corresponding written language letter.

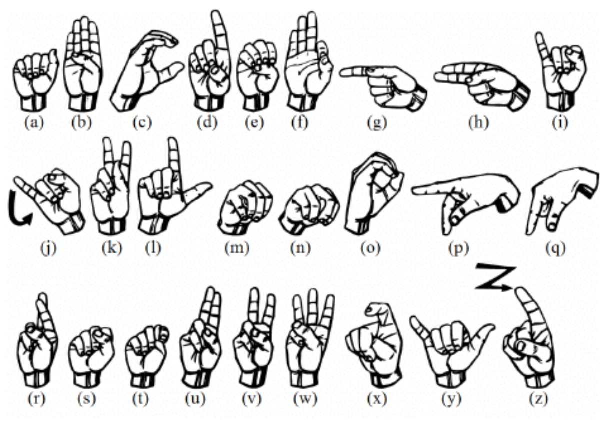

*Figure 1. Sign Language Alphabet*

The idea proposed to solve this was to implement an image classifier. Nowadays, there are two principal approaches: design and training of a Convolutional Neural Network (CNN) or finetuning the weights of a pre-existing net using a specific dataset from the problem one wants to solve.

The first approaches were done using already existing datasets and models trained to solve the said problem proposed, but several troubles were found along the testing of the algorithms.

Finally, we opted for augmenting one of the aforementioned datasets with our own data and finetune a pre-existing model.

## 2. Datasets and methods

As mentioned on the introduction section, we began the process by using already existing datasets. The most relevant ones were Sign Language MINST [2], ASL Alphabet [3] and Significant (ASL) Sign Language Alphabet Dataset [4]. For these datasets, there were several algorithms developed on Kaggle [5] that served our purpose. Additionally, we created our own test dataset following each of the dataset's criteria to check if said algorithms could generalize well for a different type of images. We found several issues that made these algorithms fail when classifying the images from our dataset, all of that explained below.

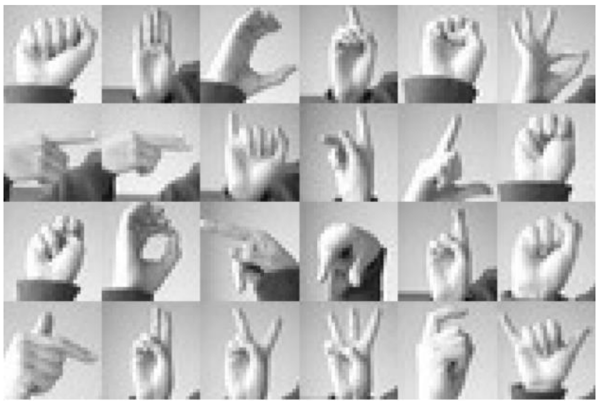

*Figure 2: SignLanguage MINST dataset*

### 2.1 Sign Language MNIST

This first dataset's format is patterned to match with the classic MNIST dataset. Each training and test case represents a label from 0 to 25 as a one-to-one map for each of the letters of the alphabet, with the exception of 'J' and 'Z', since they require motion.

The training data is formed by 27455 cases, while the test data is formed by 7172 cases. Each of the images is square shaped with a 28x28 pixel resolution and grayscale values between 0 and 255. The pictures represent users repeating gestures against different backgrounds.

The data comes from extending 1704 of the color images included as not cropped around the region of interest. To create new data, an image pipeline was used that included cropping to hands-only, gray-scaling, resizing and creating 50 variations to enlarge quantity. This strategy also counted with applying certain filters ('Mitchell', 'Robidoux', etc) along with 5% random pixelation, +/- 15% brightness and contrast and 3 degrees of rotation.

For this first experiment we used an algorithm from an user in Kaggle that proved to have a 100% accuracy. We decided to retrain the neural network to check the results finding that it was in fact true.

We tested this network using a test dataset of our own after adjusting the size and resolution of the images to fit the algorithm. We obtained an accuracy of approximately 20%, meaning that this network could not generalize for our kind of images.

We tested again this algorithm by creating a new dataset taking into account the images from the training set and adjusting the cropping to fit the hand in the maximum size possible as observed in the training dataset. We obtained again the same accuracy. Our intuition is that it fails because the training images have very specific illumination, that the resolution of the images is very low and, since they are processed to be grayscale images, they can not benefit from the color information.

So, the reason why the user obtained a 100% accuracy for the test set relies on the fact that the images were very similar to the ones in the training set.

We concluded that, with this specific database, no algorithm could be trained to detect images with different lightning and with hands locations at different distances from the objective.

## 2.2 ASL Alphabet

The ASL (American Signed Language) Alphabet is a data set collection of images from the American Sign Language, which are separated in 29 folders according to the different letters classes.

The training data set contains 87000 images which are 200x200 pixels. 26 of them are for the letters from A to Z, while there are 3 extra classes for SPACE, DELETE and NOTHING. This last three classes are very helpful when it comes to real-time applications and classification. The test dataset is formed by only 29 images, encouraging the use of new different test images.

We used again an algorithm developed by a Kaggle user [7], which, to test, had used a very small test set that contained one image per class.

This algorithm reached a 99.68% of accuracy for the test set. As before, we retrained the model to check the results. However, again, when introducing our own test dataset, the results were considerably worse, not reaching 30% of accuracy. After analyzing the contents of both test and training set, we found out that the bad results for our test dataset compared with the users' dataset were due to the fact that both sets had similar if not the same images. Also, it is clearly seen that the lightning in both of sets is very poor, which makes the shape of the hands easily depicted and does not allow the correct classification of the well illuminated hand images.

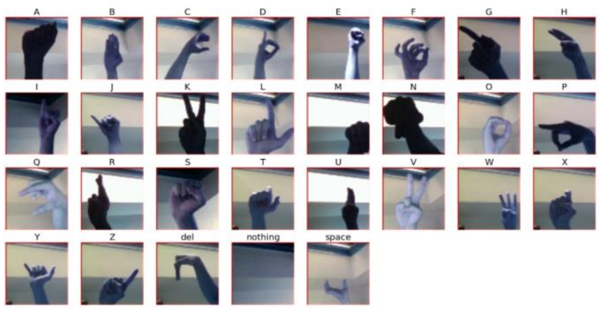

*Figure 3: Training set for the ASL Alphabet dataset*

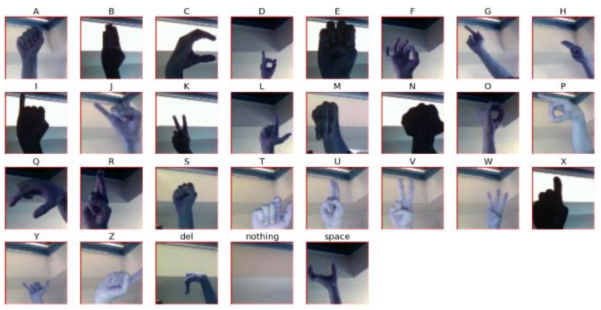

*Figure 4: Test dataset for the ASL Alphabet dataset*

## 2.3 Significant (ASL) Sign Language Alphabet Dataset

This dataset is formed by only a training set, which is composed of 27 classes, one for each letter of the alphabet and one extra for the class SPACE.

This dataset has approximately 3000 differently shaped images of each letter so, to better test the accuracy, we added the corresponding number of images to each folder in order to have the same number of images for each one of the classes.

We thoroughly analyzed the content of the dataset and found out that it was the most diverse that we could find up to that point. Seeing that there was not an specific algorithm developed for this dataset, we decided to finetune a preexisting model, further explained below.

Since it was lacking a validation set, we decided to use an already existing one, created based on the ASL Alphabet dataset mentioned above. This was called ASL Alphabet Test [8], and contains 30 images per symbol, which makes a total of 870 pictures, shaped 200x200. This dataset was originally formed by 29 different classes, one for each letter of the alphabet, as well as the three aforementioned classes SPACE, DEL and NOTHING. We removed both DEL and NOTHING, since they were not considered in the training set. We also removed the letters 'J' and 'Z' from all of the datasets, since they require motion and may lead to misclassifications.

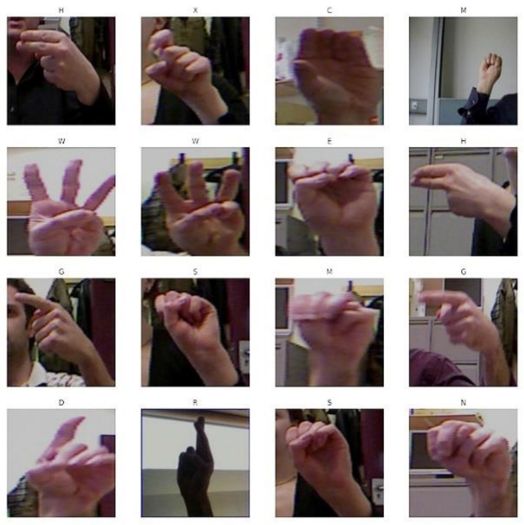

Figure 5: Training set

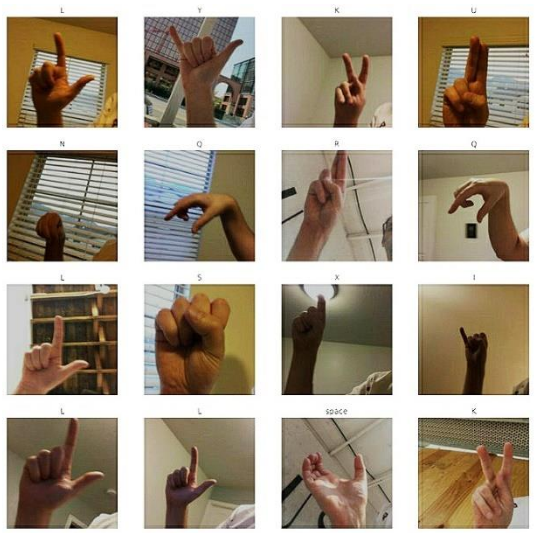

Figure 6: Validation set

As for the test set, we decided to develop our own, using pictures taken by us. We considered different backgrounds, distances and lightning. For each letter, except 'J' and 'Z', we collected ten images captured in square shape to avoid later processing. We also considered hands from different people.

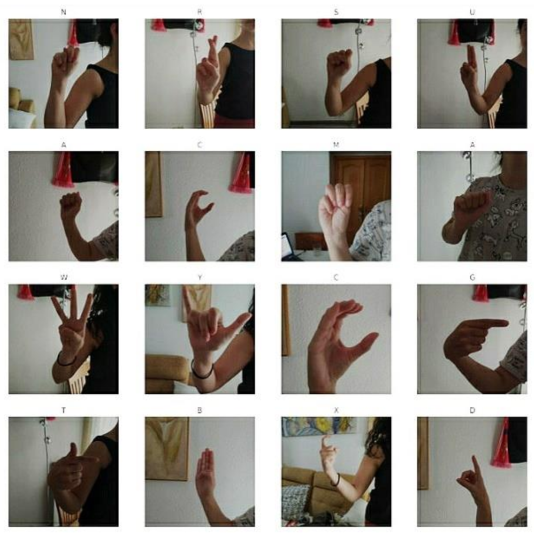

Figure 7: Test set

Once a balanced dataset was built, the next step was to develop an algorithm that would generalize for any kind of images where the main focus is a hand signaling a letter in the ASL Alphabet. In order to do this, we opted for finetuning a pre-existing model using *torchvision* [9].

We opted to use *resnet18*, which has 18 layers, with early stopping. All of the pre-trained models expect input images to be normalized in a specific way. So, in this case, images had to be 3-channel RGB images [3, H, W], where H (height) and W (width) are expected to be at least 224. All of the inputs need to have the same characteristics. We had to consider the fact that our training set had different shapes, so we preprocessed them in order to square them. This step was not necessary for the validation and test set, since they already had an adequate shape.

Since our dataset has a limited amount of images, we decided to implement data augmentation. Data augmentation is a very relevant step in the training of a model, so that the machine learning model has a bigger diversity of data. The bigger the variability in the pictures the more complex is the task that our model has to perform so, to make the model as robust as possible we needed to perform this data augmentation.

To prepare the data in order to fit the *resnet18* model, we used *PyTorch,* since it provides the tools to simplify the data loading preprocess, as well as the transformations needed for the data augmentation. So, we unified both this steps to simplify the algorithm.

The tunning of the parameters and transformations is further explained in section 3.

## 3. Experiments and analysis of the results

We trained *resnet18* with the following settings:

- Optimizer: Stochastic Gradient Descent (SGD).
  - Learning Rate: 1e-3
  - Momentum: 0.9
  - Learning Rate Scheduler:
    - Step Size: 8
    - Gamma: 0.1
- Loss function: Cross Entropy Loss.
- Early stopping: Done by saving the weights at the iteration that reached a smaller validation loss than the previous iterations.

For data preprocessing and augmentation we tested several combinations using *torchvision.transforms*.

To begin, we decided to do a study on how the different transformations affected the images, to fix some of the parameters to specific values and to analyze the results obtained when modifying others. To use correctly these types of transformations, it is really important to check the resultant images, to see if they could be real images from the test set and not too different from the actual test dataset.

The following transformations are applied only to the training set, while for the test and validation we only applied resizing and normalizing.

We decided to use in every case *RandomHorizontalFlip*and after some experiments we fixed the value for *RandomRotation* to 30. Then, we tested the influence that the transformation *ColorJitter* had on the images. After some adjustments, we decided to leave its values to 0.3 (for brightness, contrast and saturation) while the value for hue was set to 0. The other fixed parameters were those in the *normalize* transform since they are characteristic from the *resnet18* model. Its values were: ([ 0.485, 0.456, 0.406], [0.229, 0.224, 0.255]), corresponding the first vector to the mean, and the second one to the standard deviation.

The first training process of the model was done adding the transformation *resize* with the values (224, 224) in order to have all of the images with the same size.

The values obtained for each of the accuracies are: training accuracy of 99,85%, validation accuracy of 54,38% and testing accuracy of 67,41%.

For the second training we decided to change the hue value to test its effect, so we fixed it to 0.05. In this new model we obtained accuracies of 99,56% for training, 60,94% for validation and 70,54% for testing.

In the next training process, we added to the fixed parameters the *Grayscale* transform with a number of outputs of 3, so that it would fit the characteristics of the network. We did that because we believed that the model might be wrongly associating some of the colors to certain letters hampering the correct classification. So, we applied this transformation to all of the datasets to have all of them in grayscale. The accuracies in this case are: 97.07% for the training accuracy, 46,09% for the validation accuracy and test accuracy of 61,61%.

Next, we removed the *Grayscale* transformation and decided to try to train the network using a function that implemented PCA color augmentation. However, the training was unsuccessful due to problems in its implementation while working with the rest of the torch transformations.

After that, we realized the resizing in our training set was deforming the images, so we decided to change its value to 224 only for the training set, so that it would maintain its proportionality. Then, we decided to add the transformation *CenterCrop* with value 224, so that we would obtain a squared image without deformations. This approach had the disadvantage that, in certain occasions, part or even the whole hand was left out of the resultant image. In this case, the accuracies obtained were: 94,82% for the training, 54,06% for the validation and 66,52% for the testing.

Afterwards, we opted for changing the *CenterCrop* transformation to a *RandomCrop* with the same value. In this occasion the same problem as before occurred. This approach was different than the previous one in the sense that, for some of the images, while center cropping could result in a hand being always left out, random cropping could leave out the hand on some occasions, while in others it would include it. The accuracies obtained in this case were: 93,90% for training, 48,12% for validation and 67,41% for testing.

It is clearly seen that the best option was the second training attempt, which managed to get a 70,54% accuracy for the test set. These results show that the colors of the images carry important information for the classification of the letters, since adding the grayscale transformation worsens the accuracy previously obtained. In addition to this, we can estate that, although keeping the whole image translates in deformations due to resizing, it its preferred to cropping, which can result on losing part of the hand and discarding relevant information.

In the case of cropping, better results were obtained with random cropping, due to the fact that it manages to provide the information from the whole image to the network even when on some occasions it provides the parts of the image that are not relevant for the classifier.

Finally, adding a bit of color variation with the hue parameter results on an improvement of the results due to the fact that it adds a variation that looks realistic to the training set, resulting in a better generalization of the algorithm for new inputs.

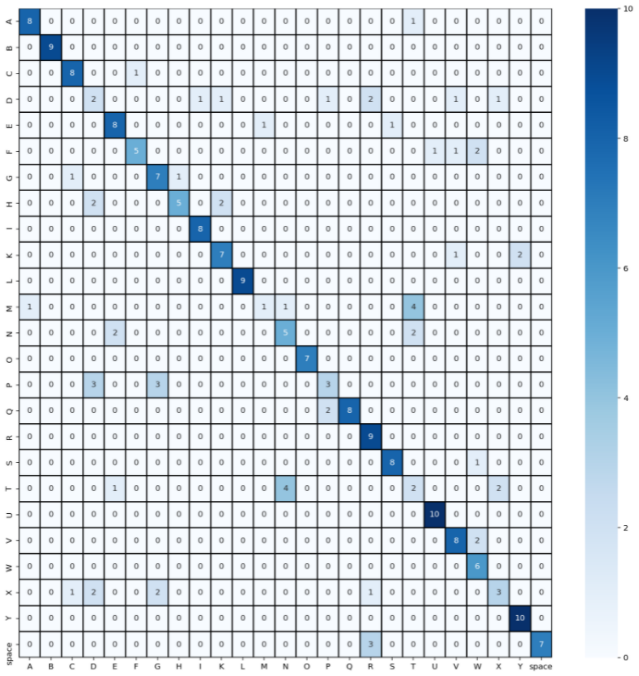

Figure 8: Confusion matrix for the best model

From the confusion matrix we can conclude that the algorithm has trouble discerning those signs that are represented by a fist with the small change of the position of one finger. This is clearly seen in the case of letter 'M', which has been misclassified six times as an 'A', 'N' and 'T'. A good example of this is also letter 'D', which is misclassified eight times as letters 'I', 'K', 'P', 'R', 'V' and 'X'. These letters are represented by one or two raised fingers, which, depending on the angle of the picture as well as the lightning are easily mistakable. It also affects the random rotation of 30 degrees applied in the data augmentation, particularly in the case of letter 'P', which has the same shape with a different orientation. It also important to notice, that letter 'P' is mistaken for letter 'G', as they both have a signaling finger following a horizontal direction.

Letter 'X' is misclassified six times with letters that also have a concave shape, like 'C', 'D' and 'G'.

Finally, those letters which are very characteristic and different from the rest, are perfectly classified. Among these letters are 'Y', 'W', 'L', 'B', 'R', and 'U'.

Table 1 collects the statistical information about the results obtained for our best model.

| | **precision** | **recall** | **f1-score** | **support** |
|---|---|---|---|---|
| **A** | 0.89 | 0.89 | 0.89 | 9 |
| **B** | 1.00 | 1.00 | 1.00 | 9 |
| **C** | 0.80 | 0.89 | 0.84 | 9 |
| **D** | 0.22 | 0.22 | 0.22 | 9 |
| **E** | 0.73 | 0.80 | 0.76 | 10 |
| **F** | 0.83 | 0.56 | 0.67 | 9 |
| **G** | 0.58 | 0.78 | 0.67 | 9 |
| **H** | 0.83 | 0.56 | 0.67 | 9 |
| **I** | 0.89 | 1.00 | 0.94 | 8 |
| **K** | 0.70 | 0.70 | 0.70 | 10 |
| **L** | 1.00 | 1.00 | 1.00 | 9 |
| **M** | 0.50 | 0.14 | 0.22 | 7 |
| **N** | 0.50 | 0.56 | 0.53 | 9 |
| **O** | 1.00 | 1.00 | 1.00 | 7 |
| **P** | 0.50 | 0.33 | 0.40 | 9 |
| **Q** | 1.00 | 0.80 | 0.89 | 10 |
| **R** | 0.60 | 1.00 | 0.75 | 9 |
| **S** | 0.89 | 0.89 | 0.89 | 9 |
| **T** | 0.22 | 0.22 | 0.22 | 9 |
| **U** | 0.91 | 1.00 | 0.95 | 10 |
| **V** | 0.73 | 0.80 | 0.76 | 10 |
| **W** | 0.55 | 1.00 | 0.71 | 6 |
| **X** | 0.50 | 0.33 | 0.40 | 9 |
| **Y** | 0.83 | 1.00 | 0.91 | 10 |
| **space** | 1.00 | 0.70 | 0.82 | 10 |
| | | | | |
| **accuracy** | | | 0.73 | 224 |
| **macro avg** | 0.73 | 0.73 | 0.71 | 224 |
| **weightedavg** | 0.73 | 0.73 | 0.72 | 224 |

Table1: Metrics from the best model

## 4. Conclusion

The main task of this project was to develop an algorithm that could receive as an input images of hands using ASL and identify which letter was represented. To do that, we finetuned a pre-existing model (*resnet18*), which would calculate the probability of the input items belonging to each one of the 25 classes defined to later classify them. This classes correspond to each one of the letters from the alphabet, except 'J' and 'Z', which need motion to be represented, and an additional class called 'SPACE'.

To solve this problem, several approaches were taken. We began by studying different already existing datasets to later complement the one we chose to use with our own test dataset. After that, we tested different parameter combinations for our data augmentation and introduced our test data into the algorithm developed to check the performance for each combination. After testing each of the changes, we concluded that the best result obtained comes from adding some random rotation; horizontal flipping; brightness, contrast, saturation and color variation; and performing a resizing into a squared shape.

From the results we can extrapolate that the classifier works fine for those signs that are very characteristic, while it has difficulties discerning those which are pretty similar.

## 5. Limitations and future work

The main limitation of our project was related to the training dataset. For future works it is recommended the construction of a training dataset with images of the same dimensions. Furthermore, the more variety and quantity of images, the more robust the algorithm can be, so we highly recommend amplifying the quantity elements for each class.

Another limitation of our work was the equipment available, since our personal computers were not able to fine tune models with a larger number of layers. Using better equipment could lead to finding better results with more complex models.

Time was also a limitation since the training of the neural networks took a lot of hours and it made it impossible to try other approaches (the time needed for training one model varied between 5 and 9 hours, depending on the use of *Google Colab* or our own hardware). Besides trying other pretrained classification networks, we encourage the test of object detection or segmentation models that could lead to an improvement of the results.

Another improvement for the problem of alphabet classification would be to include the recognition of the letters 'J' and 'Z' including a tracking algorithm that would take into account the movement of the hands.

In addition to this, successfully using PCA color augmentation would probably improve significantly the results.

Finally, we consider that it would be relevant to include hands with different skin tones so that the algorithm would be more robust and work for any kind of test set.

## 6. References

[1]«OMS | 10 datos sobre la sordera». https://www.who.int/features/factfiles/deafness/facts/es/

[2] «Sign Language MNIST». https://kaggle.com/datamunge/sign-language-mnist

[3] «ASL Alphabet». https://kaggle.com/grassknoted/asl-alphabet

[4]«Significant (ASL) Sign Language Alphabet Dataset». https://kaggle.com/kuzivakwashe/significant-asl-sign-language-alphabet-dataset

[5]«Kaggle: Your Home for Data Science». https://www.kaggle.com/

[6] «CNN using Keras(100% Accuracy)». https://kaggle.com/madz2000/cnn-using-keras-100-accuracy

[7]«ASL Alphabet classification| 99.68%». https://kaggle.com/shivam2811/asl-alphabet-classification-99-68

[8] «ASL Alphabet Test». https://kaggle.com/danrasband/asl-alphabet-test

[9] «Finetuning Torchvision Models — PyTorch Tutorials 1.2.0 documentation». https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html