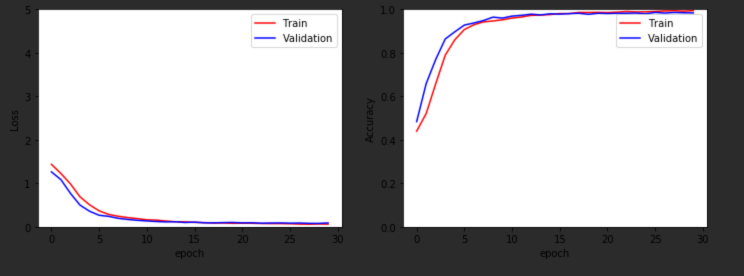
Thoughts:

* Epoch
* Batch size – too low(overfit) too high(underfit)
* <https://stackoverflow.com/questions/35050753/how-big-should-batch-size-and-number-of-epochs-be-when-fitting-a-model-in-keras>
* Low batch underfits when moving the image around. Validation stays at 50%
* Learning rate of 0.1 is too big
* Basic CNN – 30 epoch

**There is a huge amount of things affecting the speed of convergence of the model.**

Among others:

* The architecture of the network
  + The number of layers
  + The number of parameters in each layer
  + Activation functions used
  + Other architectural details
* The dataset and the complexity of the problem
* Learning algorithm
* Hyperparameters, such as
  + Learning rate
  + Dropout rate
  + Weight decay
* Loss function
* Weight initialization
  + Random
  + Pre-trained model

Looking at the training epochs, it seems to me you set a patience parameter that is too short. Please consider removing early stopping at all, for a model trained on 1500 observations only. Early stopping comes useful for particularly heavy models, but in this you shouldn't need it.

I think the size of each mini-batch is very small. That would make gradient descend very noisy, please consider increasing its size, or using full batch training as well.

Additionally, I think you have implemented a Network that is too big. Your input is very small, therefore you don't need to expand its signal on layers of size 64. There are too many nodes that are trying to "learn" not many things, IMHO. A good architecture could be:

model = Sequential()

model.add(Dense(6, input\_dim=6, activation='relu'))

model.add(Dense(6, activation='relu'))

model.add(Dense(1, activation=None))

That would make your model faster to train, and ensure that each node is learning relevant features of your data.

I would also change the output layer. Since you want to predict an outcome, you need an output node with no activation (i.e. linear activation). That is mandatory for regression tasks with unbounded output.

Additional things you can try are:

* change dropout levels (but for such a small network it might not be needed at all),
* try regularization techniques, such as Batchnorm, L1 - L2 regularization, different weights initialization... you name it,
* try alternative activation functions.

Decisions I made:

**General:**

* Because the image is small and greyscale there are only so many features that can be used, I decided to keep the features small because of this. This was backed up by my testing that adding extra convolutional layers with a large amount of filters had minimal impact on the accuracy and loss of the model, yet it decreased the performance of the model.

**Sign Shape**

* Not to zoom in on pictures, this is because the shape of the size could go out of the image

**Sign Type**