# Supplemental information

We generally assume the multivariate predictor vector is $\boldsymbol{x}$ and the binary predictand is $\boldsymbol{y}$.

## The Shallow models

### Lasso regression (LASSO)

In Lasso regression, the log-likelihood function of $\boldsymbol{y}$ given $\boldsymbol{\beta}$, $\log \ell(\boldsymbol{y};\boldsymbol{\beta})$, is maximized with an additional penalty term $\lambda$ for the size of the parameters:

$$\sum \log \ell(\boldsymbol{y};\boldsymbol{\beta}) - \lambda |\beta|_1 \tag{S1}$$

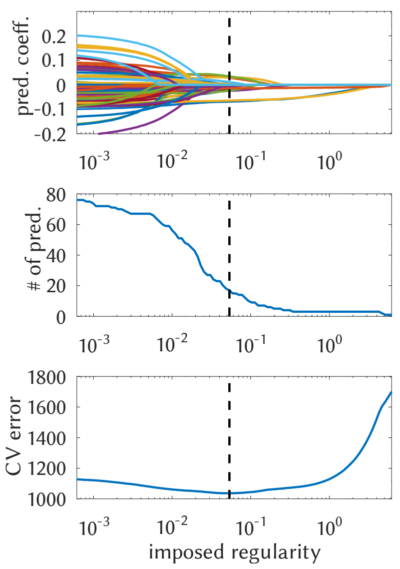

**Figure S1. Result of Lasso regression. The panels show the dependency of three regression indicators on the imposed regularity (often denoted λ), the active predictors (top), the number of predictors (middle), and the corresponding cross-validation error.**

With $\lambda \longrightarrow 0$ one obtains the classical maximum-likelihood estimate, and a growing $\lambda$ squeezes the coefficients $\boldsymbol{\beta}$ towards zero; the optimum $\lambda$ is found using cross-validation (but limited to the calibration set). We use the *penalized* toolbox (McIlhagga, 2016). A typical result is shown in Figure S1. It shows the successive disappearance of predictors with increasing regularity and how the cross-validated model error changes with that; its minimum is achieved with 18 predictors.

### Random Forest (TREE)

Random forests are obtained from *bootstrapped averaging* ('bagging') of ensembles of decision trees, where the individual trees are obtained from resampling the data; each decision tree is grown following the 'M5 method'. For details and the corresponding software see the M5PrimeLab package, cf. http://www.cs.rtu.lv/jekabsons/regression.html. Here we use the M5PrimeLab toolbox of Gints Jēkabsons, cf. http://www.cs.rtu.lv/jekabsons/regression.html, with 50 trees to build (instead of the default 100). A typical tree for a case with 81 principal components as predictors is shown in Figure S2.

### Neural net (NNET)

This is a simple feed-forward backpropagation net with merely 2 layers, one with 7 and one with 3 nodes (neurons); the numbers are based on very basic testing, cf. https://octave.sourceforge.io/nnet.

### Logistic regression (NLS)

To generally map $\boldsymbol{x}$ to a probability we combine a normal linear mapping, given by a parameter vector $\boldsymbol{\beta}$, with the sigmoid function

$$\boldsymbol{x} \quad \mapsto \quad \frac{1}{1+e^{-\beta x}} \tag{S2}$$

Following ordinary regression, the following expression needs to be minimized:

$$\sum \left( \frac{1}{1+e^{-\beta x}} - y \right)^2 \tag{S3}$$

for which a classical optimization procedure can be applied (here: Levenberg-Marquart).

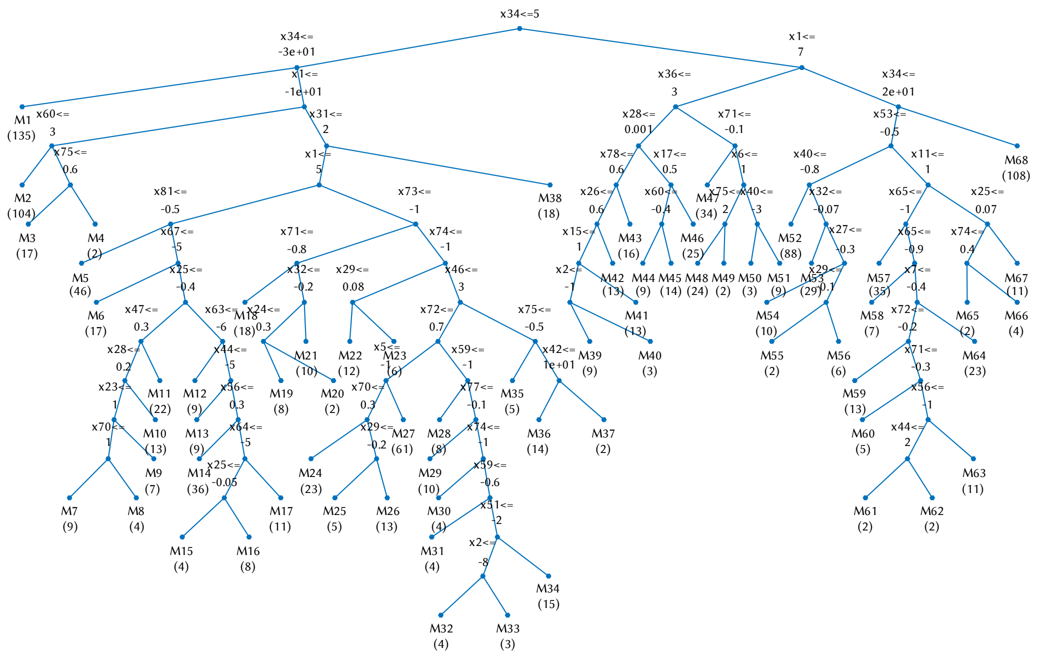

**Figure S2. One sample model tree with 81 predictors (principal components), taken from an ensemble of 50 random trees.**

## Training CNNs with Caffe

Caffe is a framework for the training of CNNs, developed at the Berkeley AI Research (BAIR) and Berkeley Vision and Learning Center (BVLC), cf. https://github.com/BVLC/caffe. Because of its Matlab/Octave interface it was easily implemented in our programming framework at https://gitlab.dkrz.de/b324017/carlofff; all that was needed was two text files, a solver and a network file, containing a few crucial parameters that had to be adapted to the new context. The main body of their content could be left intact. Table S1 lists the hyperparameters (cf. https://github.com/BVLC/caffe/blob/master/src/caffe/proto/caffe.proto#L164-L165). All default settings can be inspected under https://gitlab.dkrz.de/b324017/carlofff/-/tree/master/models, with the model specific settings in the respective directory. No extra data split was done for the hyperparameter tuning, so the validation data are not fully independent. However, because the sole tuning criterion was not model skill but learning convergence, this should be negligible.

**Table S1. Tuned hyperparameters for the networks, as modified from the original sources.**

| | max_iter | power | batch_size | lr_policy |
|---|---|---|---|---|
| LeNet-5 | 1000 | 3 | 100 | poly |
| AlexNet | 500 | 3 | 50 | poly |
| CIFAR-10 | 1000 | 3 | 10 | poly |
| ALL-CNN | 1000 | 0.5 | 100 | poly |
| GoogLeNet | 500 | 3 | 10 | poly |
| ResNet | 300 | 3 | 100 | poly |
| DenseNet | 500 | 3 | 10 | poly |
| Simple | 1000 | 3 | 100 | poly |
| Logreg | 1000 | 3 | 100 | poly |

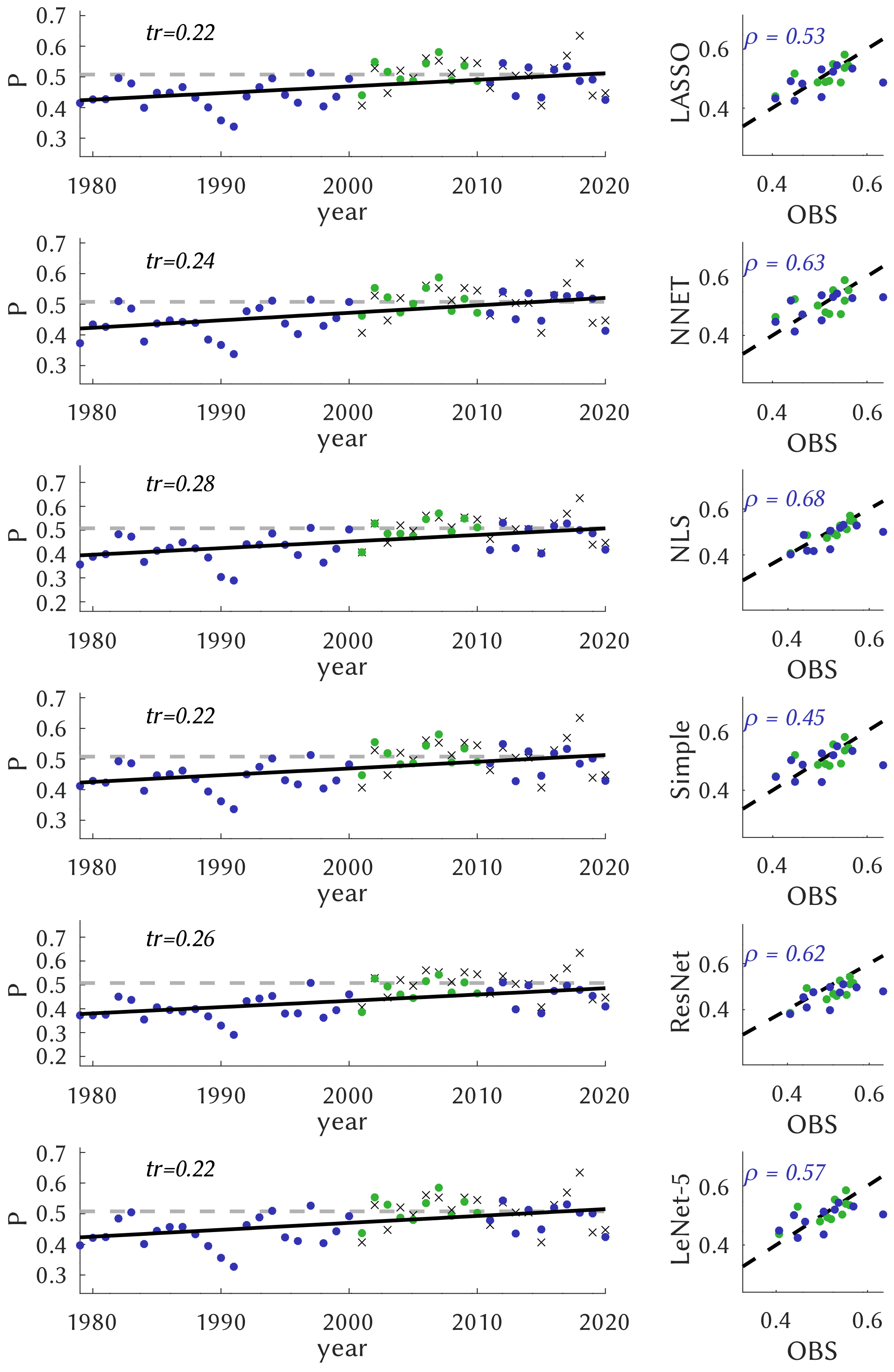

**Figure S3. Similar to Fig. 6, for the remaining models (first half).**

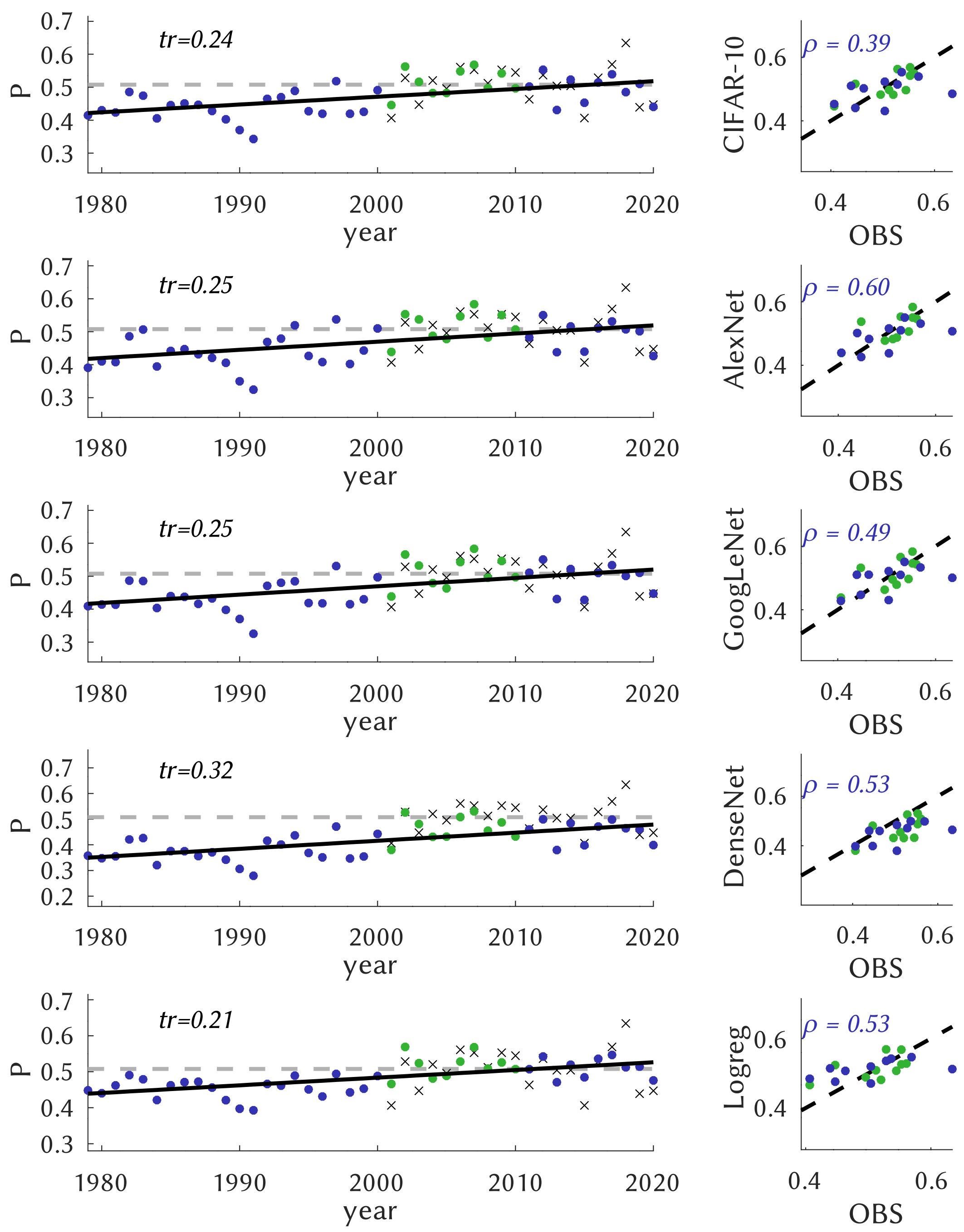

**Figure S4. Fig. S3 continued.**

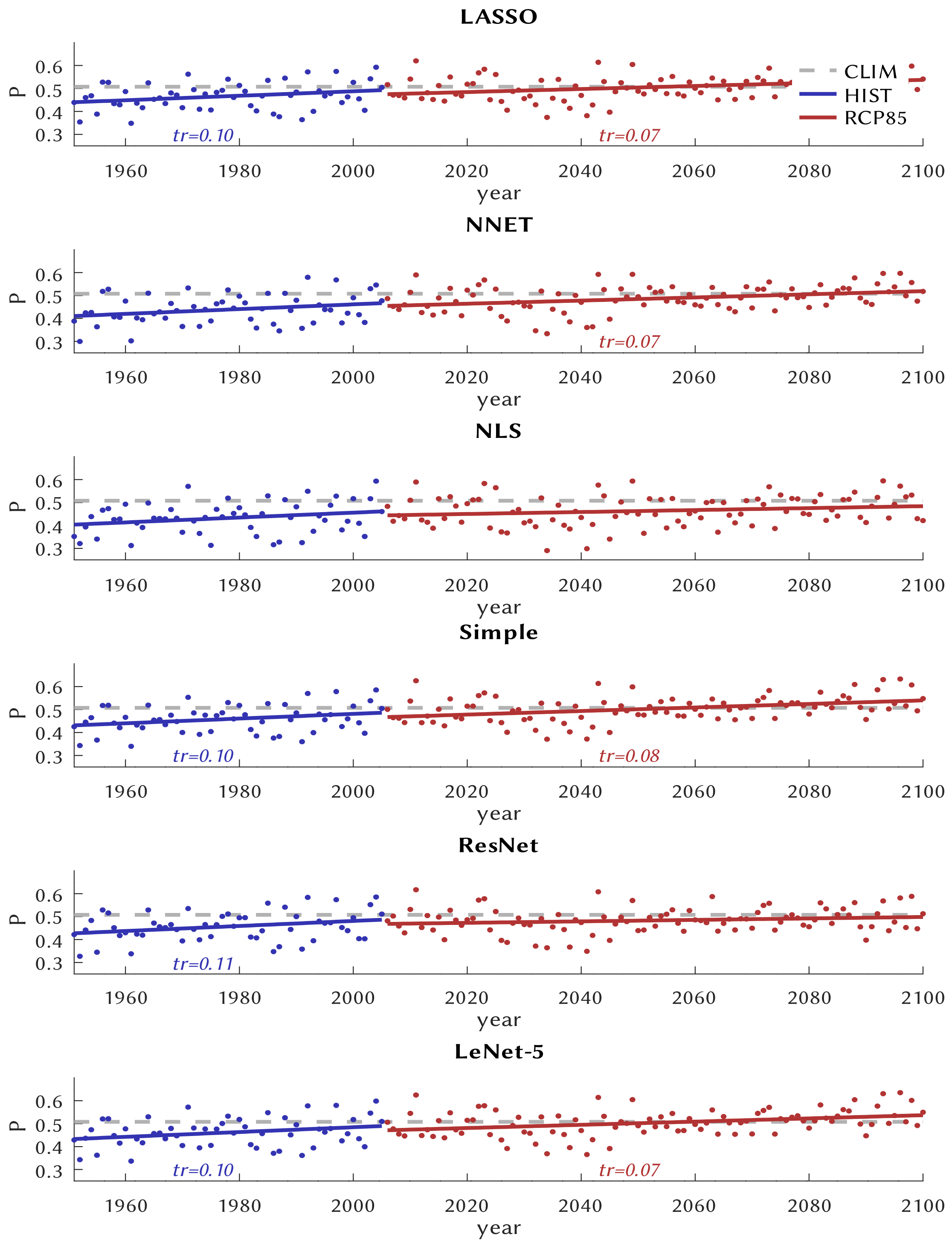

**Figure S5. Like Figure 7, for the remaining models.**

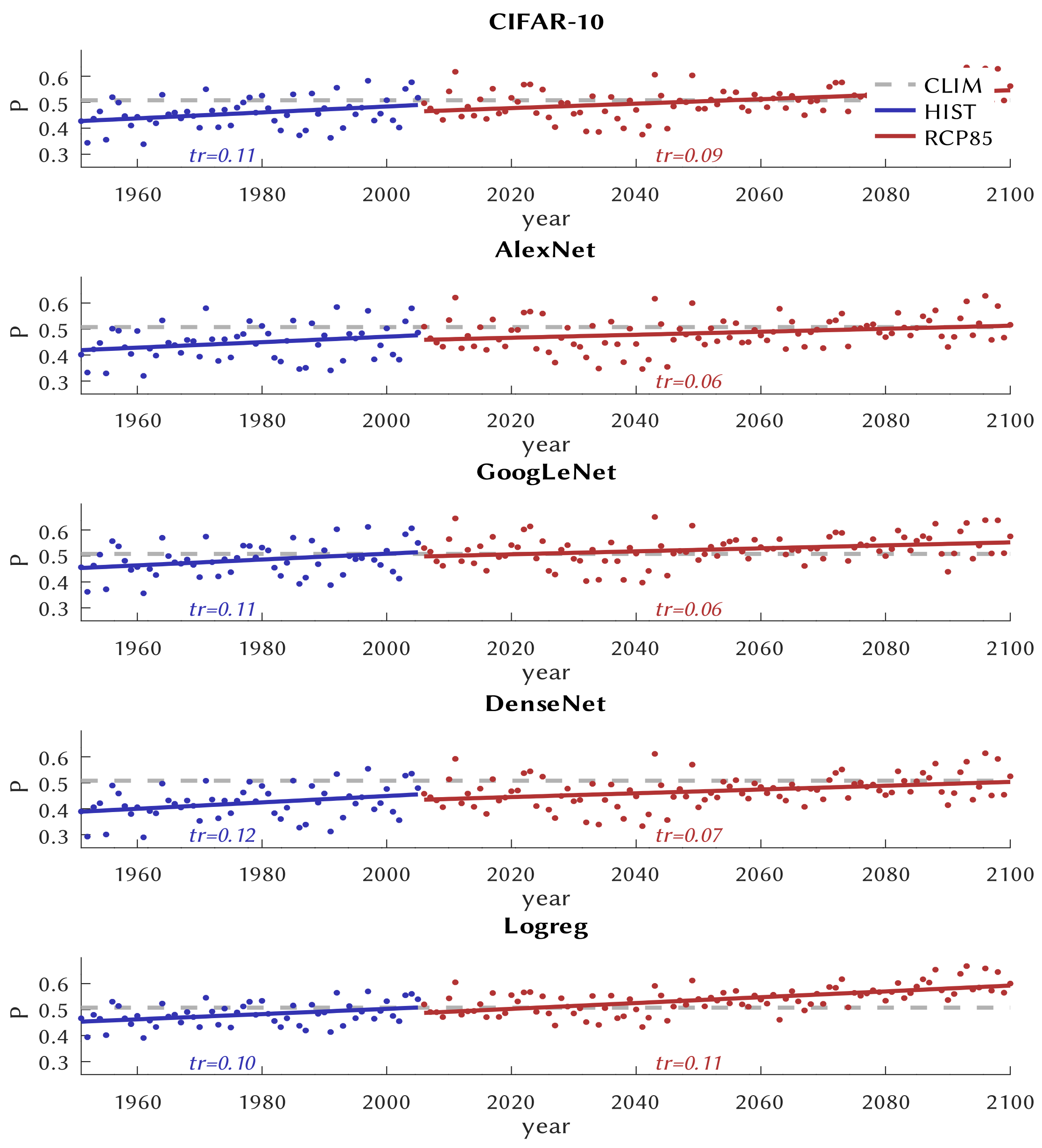

**Figure S6. Fig. S5 continued.**