CNN_Final

December 4, 2024

[1]:
```python
import xarray as xr
import pandas as pd
import numpy as np
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
```

[2]:
```python
names = ['ssp126',
         "ssp370",
        'ssp585',
         'hist-aer',
         'hist-GHG',
         'historical',
        ]
```

[3]:
```python
outputs={}
inputs={}
#load in our data from train_vals for the inputs and outputs
for name in names:
    if "hist" in name:
        inputs[f"{name}"] = xr.open_dataset(f"../train_val/inputs_{name}.nc")
        outputs_temp = xr.open_dataset(f"../train_val/outputs_{name}.nc").
 ↪mean(dim='member')
    else:
        inputs[f"{name}"] = xr.open_mfdataset(['../train_val/inputs_historical.
 ↪nc',
                                    f"../train_val/inputs_{name}.nc"]).compute()

        outputs_temp = xr.concat([xr.open_dataset("../train_val/
 ↪outputs_historical.nc").mean(dim='member'),
                               xr.open_dataset(f"../train_val/outputs_{name}.
 ↪nc").mean(dim='member')],
                               dim='time').compute()

    outputs_temp = outputs_temp.assign({"pr": outputs_temp.pr * 86400, "pr90":␣
 ↪outputs_temp.pr90 * 86400})
    outputs_temp = outputs_temp.rename({'lon':'longitude', 'lat': 'latitude'}).
 ↪transpose('time','latitude', 'longitude')
```

```
    outputs_temp = outputs_temp.drop_vars('quantile')
    outputs[f"{name}"] = outputs_temp
```

[4]:
```
len_historical_data = len(outputs['historical'].time)
```

[5]:
```
del outputs['historical']
del inputs['historical']
```

[6]:
```
mean_std_dict={}
input_vars = ['CO2', 'CH4', 'SO2', 'BC']
#takes the mean and standard deviation of all the input variables
for var in input_vars:
    once_historical = []

    for key in inputs.keys():
        #counts historical data only once, for ssp126
        if "ssp" in key and key!="ssp126":
            data=inputs[key][var].sel(time=slice(len_historical_data, None)).
 ↪data
            once_historical.append(data)
        else:
            data=inputs[key][var].data
            once_historical.append(data)
    once_historical_array = np.concatenate(once_historical)

    mean_std_dict[var] = (once_historical_array.mean(), once_historical_array.
 ↪std())
    print(var)
    print("Mean: ", mean_std_dict[var][0], "Standard Deviation: ",␣
 ↪mean_std_dict[var][1])
```

```
CO2
Mean:  1074.172303244536 Standard Deviation:  1755.690699230666
CH4
Mean:  0.1927369743762821 Standard Deviation:  0.18457590641432994
SO2
Mean:  2.5623359997066674e-12 Standard Deviation:  2.25011456678326e-11
BC
Mean:  1.4947905009818114e-13 Standard Deviation:  1.031334255483836e-12
```

[7]:
```
def normalize(data, var, meanstd_dict):
    mean = meanstd_dict[var][0]
    std = meanstd_dict[var][1]
    return (data - mean)/std

def unnormalize(data, var, meanstd_dict):
    mean = meanstd_dict[var][0]
```

```
    std = meanstd_dict[var][1]
    return data * std + mean
```

[8]:
```
X_train_norm = []
#Normalize the input variables
for key, data in inputs.items():
    for var in input_vars:
        var_dims = data[var].dims
        data=data.assign({var: (var_dims, normalize(data[var].data, var,␣
 ↪mean_std_dict))})
    X_train_norm.append(data)
```

[9]:
```
#temporal window to track trends over time
slider = 10
```

[10]:
```
#reshapes data for prediction with sliders
def training_input(X_train, skip_historical=False):
    X_train_np = X_train.to_array().transpose('time', 'latitude', 'longitude',␣
 ↪'variable').data

    time_length = X_train_np.shape[0]
    #If we skip historical, then start at the length of historical onwards␣
 ↪rather than from 1850
    if skip_historical:
         X_train_mod = np.array([X_train_np[i:i+slider] for i in␣
 ↪range(len_historical_data-slider+1, time_length-slider+1)])
    else:
        X_train_mod = np.array([X_train_np[i:i+slider] for i in range(0,␣
 ↪time_length-slider+1)])

    return X_train_mod

def training_output(Y_train, var, skip_historical=False):
    Y_train_np = Y_train[var].data

    time_length = Y_train_np.shape[0]
    #If we skip historical, then start at the length of historical onwards␣
 ↪rather than from 1850
    if skip_historical:
         Y_train_mod = np.array([[Y_train_np[i+slider-1]] for i in␣
 ↪range(len_historical_data-slider+1, time_length-slider+1)])
    else:
        Y_train_mod = np.array([[Y_train_np[i+slider-1]] for i in range(0,␣
 ↪time_length-slider+1)])

    return Y_train_mod
```

[11]:
```
predict_var='tas'
Y_train=list(outputs.values())
#Concatenates the training inputs and the outputs, making sure to only count␣
↪historical once
X_train_all = np.concatenate([training_input(X_train_norm[i],
                                             skip_historical=(i<2),
                                             ) for i in␣
↪range(len(outputs))], axis = 0)
Y_train_all = np.concatenate([training_output(Y_train[i],
                                              predict_var,
                                              skip_historical=(i<2),
                                             ) for i in␣
↪range(len(outputs))], axis=0)
print(X_train_all.shape, Y_train_all.shape)
```

```
(726, 10, 96, 144, 4) (726, 1, 96, 144)
```

[12]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Conv2D, Flatten, Input,␣
↪Reshape, AveragePooling2D, MaxPooling2D, Conv2DTranspose, TimeDistributed,␣
↪LSTM, GlobalAveragePooling2D, BatchNormalization, Dropout
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

import random
seed = 6
random.seed(seed)
np.random.seed(seed)
tf.random.set_seed(seed)
```

[13]:
```
keras.backend.clear_session()
cnn_model = None
```

```
WARNING:tensorflow:From C:\Users\charl\anaconda3\envs\B13\lib\site-
packages\keras\src\backend\common\global_state.py:82: The name
tf.reset_default_graph is deprecated. Please use
tf.compat.v1.reset_default_graph instead.
```

[14]:
```
cnn_model = Sequential()
cnn_model.add(Input(shape=(slider, 96, 144, 4)))
cnn_model.add(TimeDistributed(Conv2D(20, (3, 3), padding='same',␣
↪activation='relu')))
cnn_model.add(TimeDistributed(AveragePooling2D(2)))
cnn_model.add(TimeDistributed(GlobalAveragePooling2D()))
```

```
cnn_model.add(LSTM(25, activation='relu'))
cnn_model.add(Dense(1*96*144))
cnn_model.add(Activation('linear'))
cnn_model.add(Reshape((1, 96, 144)))
```

[15]:
```
cnn_model.summary()
```

```
Model: "sequential"



 Layer (type)                    Output Shape                                ␣
↪Param #

 time_distributed (TimeDistributed)     (None, 10, 96, 144, 20)              ␣
↪740

 time_distributed_1 (TimeDistributed)  (None, 10, 48, 72, 20)                ␣
↪  0

 time_distributed_2 (TimeDistributed)  (None, 10, 20)                        ␣
↪  0

 lstm (LSTM)                     (None, 25)                                  ␣
↪4,600

 dense (Dense)                   (None, 13824)                               ␣
↪359,424

 activation (Activation)         (None, 13824)                               ␣
↪  0

 reshape (Reshape)               (None, 1, 96, 144)                          ␣
↪  0



 Total params: 364,764 (1.39 MB)

 Trainable params: 364,764 (1.39 MB)

 Non-trainable params: 0 (0.00 B)
```

[16]:
```
X_train_all.shape, Y_train_all.shape
```

[16]: ((726, 10, 96, 144, 4), (726, 1, 96, 144))

[17]:
```
cnn_model.compile(optimizer='rmsprop', loss='mse', metrics=['mse'])
```

[18]:
```
#Constructrs data generator for tensorflow 2.18, comment out if using␣
↪Tensorflow 2.10 and use below
hist = cnn_model.fit(X_train_all,
                     Y_train_all,
                     batch_size=16,
                     epochs=30,
                     verbose=1,
                    )

#hist = cnn_model.fit(X_train_all,
#                     Y_train_all,
#                     use_multiprocessing=True,
#                     #workers=5,
#                     batch_size=16,
#                     epochs=30,
#                     verbose=1)
#
```

```
Epoch 1/30
46/46                 22s 452ms/step -
loss: 2.7271 - mse: 2.7271
Epoch 2/30
46/46                 21s 451ms/step -
loss: 0.4963 - mse: 0.4963
Epoch 3/30
46/46                 21s 454ms/step -
loss: 0.3967 - mse: 0.3967
Epoch 4/30
46/46                 21s 452ms/step -
loss: 0.3884 - mse: 0.3884
Epoch 5/30
46/46                 21s 452ms/step -
loss: 0.3823 - mse: 0.3823
Epoch 6/30
46/46                 21s 453ms/step -
loss: 0.3765 - mse: 0.3765
Epoch 7/30
46/46                 21s 452ms/step -
loss: 0.3702 - mse: 0.3702
Epoch 8/30
46/46                 21s 458ms/step -
loss: 0.3524 - mse: 0.3524
Epoch 9/30
46/46                 21s 455ms/step -
loss: 0.3285 - mse: 0.3285
Epoch 10/30
```

```
46/46                21s 452ms/step -
loss: 0.3198 - mse: 0.3198
Epoch 11/30
46/46                21s 453ms/step -
loss: 0.3158 - mse: 0.3158
Epoch 12/30
46/46                21s 454ms/step -
loss: 0.3130 - mse: 0.3130
Epoch 13/30
46/46                21s 453ms/step -
loss: 0.3108 - mse: 0.3108
Epoch 14/30
46/46                21s 453ms/step -
loss: 0.3090 - mse: 0.3090
Epoch 15/30
46/46                21s 453ms/step -
loss: 0.3075 - mse: 0.3075
Epoch 16/30
46/46                21s 453ms/step -
loss: 0.3062 - mse: 0.3062
Epoch 17/30
46/46                21s 453ms/step -
loss: 0.3052 - mse: 0.3052
Epoch 18/30
46/46                21s 454ms/step -
loss: 0.3043 - mse: 0.3043
Epoch 19/30
46/46                21s 454ms/step -
loss: 0.3035 - mse: 0.3035
Epoch 20/30
46/46                21s 452ms/step -
loss: 0.3029 - mse: 0.3029
Epoch 21/30
46/46                21s 453ms/step -
loss: 0.3023 - mse: 0.3023
Epoch 22/30
46/46                21s 453ms/step -
loss: 0.3016 - mse: 0.3016
Epoch 23/30
46/46                21s 457ms/step -
loss: 0.3010 - mse: 0.3010
Epoch 24/30
46/46                21s 453ms/step -
loss: 0.3004 - mse: 0.3004
Epoch 25/30
46/46                21s 453ms/step -
loss: 0.2998 - mse: 0.2998
Epoch 26/30
```

```
46/46               21s 456ms/step -
loss: 0.2992 - mse: 0.2992
Epoch 27/30
46/46               21s 455ms/step -
loss: 0.2986 - mse: 0.2986
Epoch 28/30
46/46               21s 455ms/step -
loss: 0.2980 - mse: 0.2980
Epoch 29/30
46/46               21s 455ms/step -
loss: 0.2973 - mse: 0.2973
Epoch 30/30
46/46               21s 455ms/step -
loss: 0.2967 - mse: 0.2967
```

[19]:
```
X_test = xr.open_dataset(f"../test/inputs_ssp245.nc")
Y_test = xr.open_dataset(f"../test/outputs_ssp245.nc").mean(dim='member')
```

[20]:
```
X_test_norm = xr.Dataset()
#Normalizes Test Data
for var in input_vars:
    dims = X_test[var].dims
    X_test_norm = X_test_norm.assign({var: (dims, normalize(X_test[var].data,␣
 ↪var, mean_std_dict))} )
```

[21]:
```
X_test_all  = training_input(X_test_norm)
```

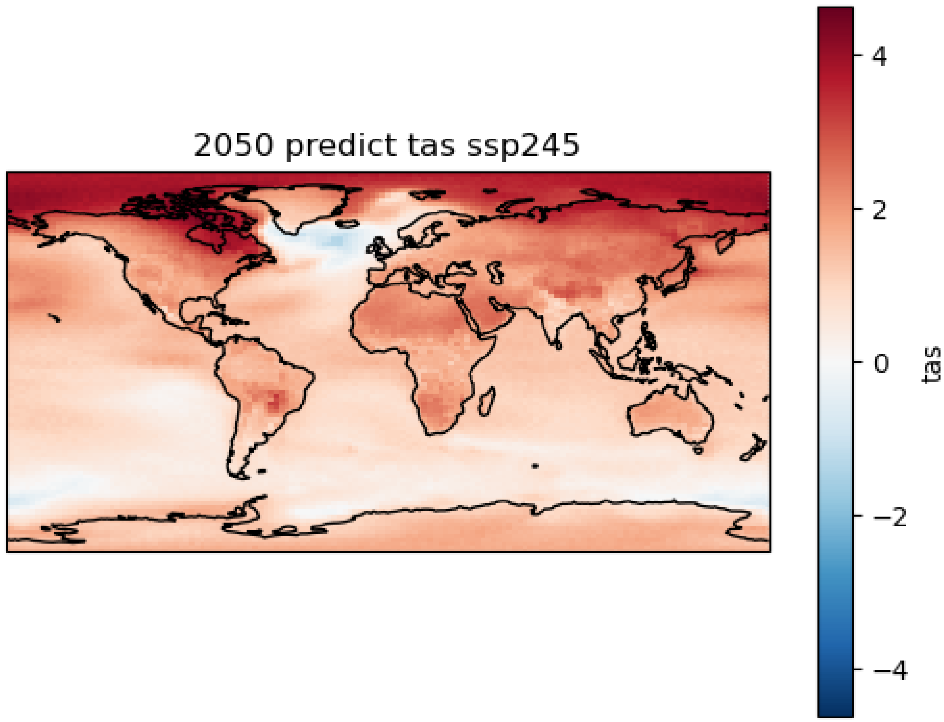
[22]:
```
#Predicts off test data
m_pred = cnn_model.predict(X_test_all)
m_pred = m_pred.reshape(m_pred.shape[0], m_pred.shape[2], m_pred.shape[3])
m_pred = xr.DataArray(m_pred, dims=['time', 'lat', 'lon'], coords=[X_test.time.
 ↪data[slider-1:], X_test.latitude.data, X_test.longitude.data])
m_pred = m_pred.transpose('time', 'lat', 'lon').sel(time=slice(2015, 2101)).
 ↪to_dataset(name=predict_var)
m_pred
```

```
3/3              1s 157ms/step
```

[22]:
```
<xarray.Dataset> Size: 4MB
Dimensions:  (time: 77, lat: 96, lon: 144)
Coordinates:
  * time     (time) int64 616B 2024 2025 2026 2027 2028 … 2097 2098 2099 2100
  * lat      (lat) float64 768B -90.0 -88.11 -86.21 -84.32 … 86.21 88.11 90.0
  * lon      (lon) float64 1kB 0.0 2.5 5.0 7.5 10.0 … 350.0 352.5 355.0 357.5
Data variables:
    tas      (time, lat, lon) float32 4MB 1.077 1.056 1.098 … 5.497 5.476
```

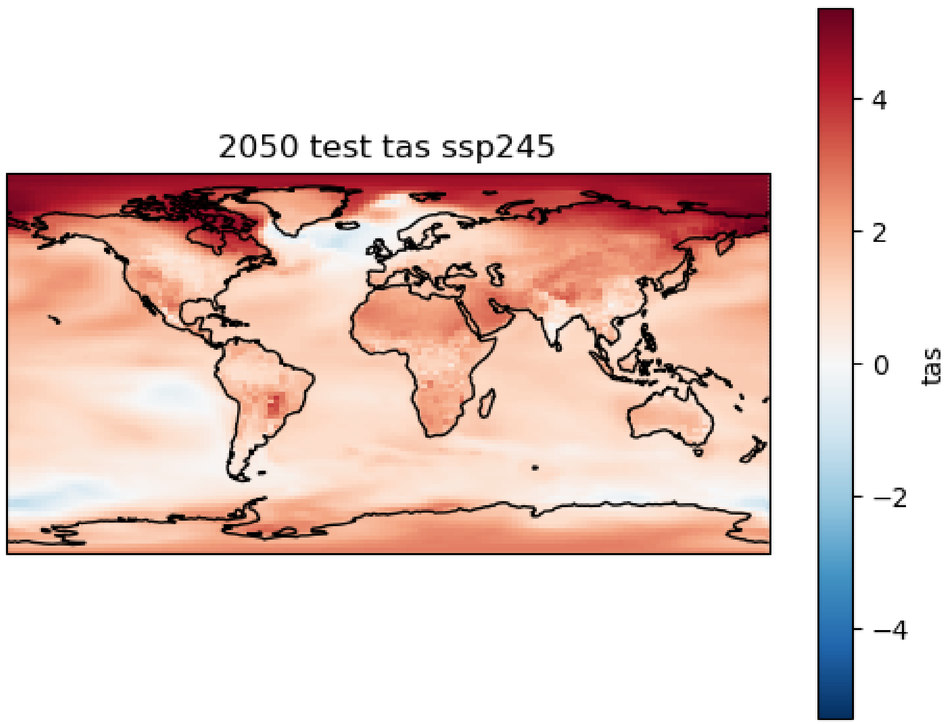
[24]:
```
#Visualizes the model's predicted tas for 2050
alpha = m_pred['tas'].sel(time=2050).plot(transform=ccrs.PlateCarree(),
                                                  subplot_kws={'projection':␣
 ↪ccrs.PlateCarree()})
plt.gca().set_title('2050 predict tas ssp245')
plt.gca().coastlines()
```

[24]: <cartopy.mpl.feature_artist.FeatureArtist at 0x22e72002080>



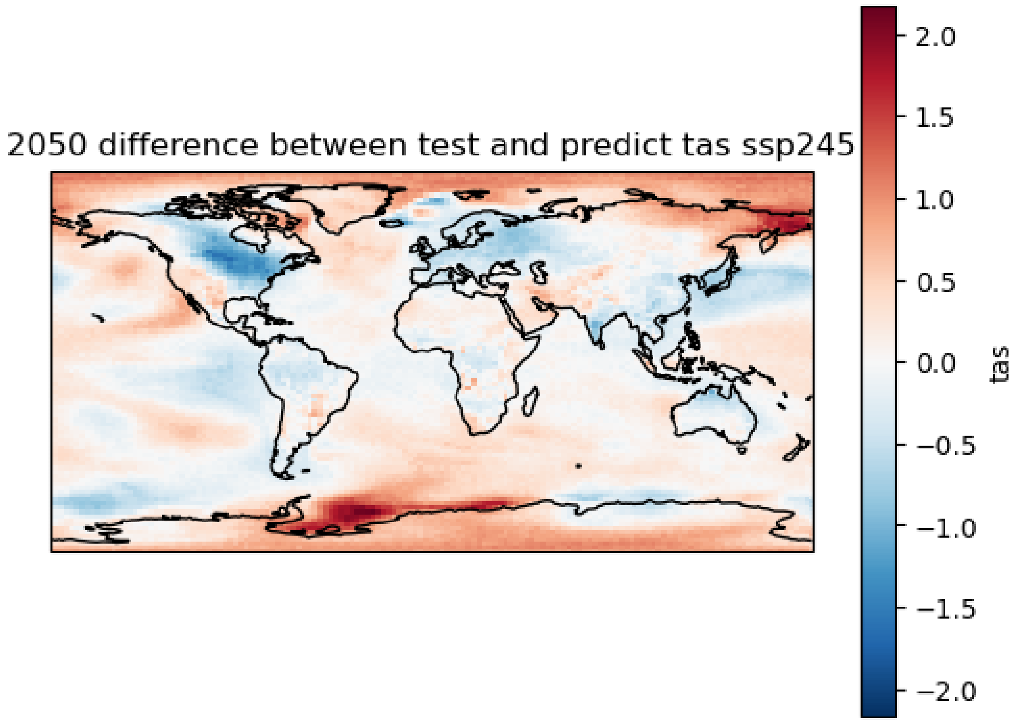
[25]:
```
#Visualizes the actual tas for 2050
beta = Y_test['tas'].sel(time=2050).plot(transform=ccrs.PlateCarree(),
                                                  subplot_kws={'projection':␣
 ↪ccrs.PlateCarree()})
plt.gca().set_title('2050 test tas ssp245')
plt.gca().coastlines()
```

[25]: <cartopy.mpl.feature_artist.FeatureArtist at 0x22ecfbf9870>



[26]:
```
#Visualizes the difference between actual and predicted tas for 2050
(Y_test['tas']-m_pred['tas']).sel(time=2050).plot(transform=ccrs.PlateCarree(),
                                                  subplot_kws={'projection':␣
 ↪ccrs.PlateCarree()})
plt.gca().set_title('2050 difference between test and predict tas ssp245')
plt.gca().coastlines()
```

[26]: <cartopy.mpl.feature_artist.FeatureArtist at 0x22ecfc54250>



[27]:
```
def get_rmse(truth, pred):
    weights = np.cos(np.deg2rad(truth.lat))
    return np.sqrt(((truth-pred)**2).weighted(weights).mean(['lat', 'lon'])).
↪data.mean()
```

[28]:
```
print(f"RMSE at 2050: {get_rmse(Y_test[predict_var][35], m_pred[predict_var].
↪sel(time=2050))}")
print(f"RMSE at 2100: {get_rmse(Y_test[predict_var][85], m_pred[predict_var].
↪sel(time=2100))}")
print(f"RMSE 2045-2055: {get_rmse(Y_test[predict_var][30:41],␣
↪m_pred[predict_var].sel(time=slice(2045, 2055)))}")
print(f"RMSE 2090-2100: {get_rmse(Y_test[predict_var][75:], m_pred[predict_var].
↪sel(time=slice(2090, 2100)))}")
print(f"RMSE 2050-2100: {get_rmse(Y_test[predict_var][35:], m_pred[predict_var].
↪sel(time=slice(2050, 2100)))}")

print(f"RMSE average last 20y: {get_rmse(Y_test[predict_var][-20:].
↪mean(dim='time'),  m_pred[predict_var][-20:].mean(dim='time'))}")
```

```
RMSE at 2050: 0.36620983361418086
RMSE at 2100: 0.3862124353265415
RMSE 2045-2055: 0.39816862816306087
RMSE 2090-2100: 0.4313775029239601
```

```
RMSE 2050-2100: 0.4331768989470518
RMSE average last 20y: 0.29905970325923464
```

[29]:
```
datapath="./results/"
```

[30]:
```
m_pred.to_netcdf(datapath + 'outputs_ssp245_predict_{}.nc'.format(predict_var),␣
 ↪'w')
```

[31]:
```
m_pred.close()
```

0.0.1 Repeat the process above in one cell for all Prediction Variables rather than just 'tas' and saves the output in results

[32]:
```
vars_to_predict = [ 'diurnal_temperature_range', 'pr', 'pr90']
```

[35]:
```
for predict_var in vars_to_predict:
    #Takes the xtrains
    Y_train=list(outputs.values())
    X_train_all = np.concatenate([training_input(X_train_norm[i],
                                              skip_historical=(i<2),
                                              ) for i in␣
 ↪range(len(outputs))], axis = 0)
    Y_train_all = np.concatenate([training_output(Y_train[i],
                                               predict_var,
                                               skip_historical=(i<2),
                                              ) for i in␣
 ↪range(len(outputs))], axis=0)

    keras.backend.clear_session()
    cnn_model = None

    #Establishes the model
    cnn_model = Sequential()
    cnn_model.add(Input(shape=(slider, 96, 144, 4)))
    cnn_model.add(TimeDistributed(Conv2D(20, (3, 3), padding='same',␣
 ↪activation='relu')))
    cnn_model.add(TimeDistributed(AveragePooling2D(2)))
    cnn_model.add(TimeDistributed(GlobalAveragePooling2D()))
    cnn_model.add(LSTM(25, activation='relu'))
    cnn_model.add(Dense(1*96*144))
    cnn_model.add(Activation('linear'))
    cnn_model.add(Reshape((1, 96, 144)))

    cnn_model.compile(optimizer='rmsprop', loss='mse', metrics=['mse'])
    #Constructrs data generator for tensorflow 2.18, comment out if using␣
 ↪Tensorflow 2.10 and use below
    hist = cnn_model.fit(X_train_all,
```

```
                    Y_train_all,
                    batch_size=16,
                    epochs=30,
                   )

    #hist = cnn_model.fit(X_train_all,
    #                     Y_train_all,
    #                     use_multiprocessing=True,
    #                     #workers=5,
    #                     batch_size=16,
    #                     epochs=30,
    #                     callbacks=[early_stopping],
    #                      )

    #Predicts off the model
    m_pred = cnn_model.predict(X_test_all)
    m_pred = m_pred.reshape(m_pred.shape[0], m_pred.shape[2], m_pred.shape[3])
    m_pred = xr.DataArray(m_pred, dims=['time', 'lat', 'lon'], coords=[X_test.
↪time.data[slider-1:], X_test.latitude.data, X_test.longitude.data])
    m_pred = m_pred.transpose('time', 'lat', 'lon').sel(time=slice(2015, 2101)).
↪to_dataset(name=predict_var)
    m_pred

    print(f"RMSE at 2050: {get_rmse(Y_test[predict_var][35],␣
↪m_pred[predict_var].sel(time=2050))}")
    print(f"RMSE at 2100: {get_rmse(Y_test[predict_var][85],␣
↪m_pred[predict_var].sel(time=2100))}")
    print(f"RMSE 2045-2055: {get_rmse(Y_test[predict_var][30:41],␣
↪m_pred[predict_var].sel(time=slice(2045, 2055)))}")
    print(f"RMSE 2090-2100: {get_rmse(Y_test[predict_var][75:],␣
↪m_pred[predict_var].sel(time=slice(2090, 2100)))}")
    print(f"RMSE 2050-2100: {get_rmse(Y_test[predict_var][35:],␣
↪m_pred[predict_var].sel(time=slice(2050, 2100)))}")

    print(f"RMSE average last 20y: {get_rmse(Y_test[predict_var][-20:].
↪mean(dim='time'),  m_pred[predict_var][-20:].mean(dim='time'))}")

    #Visual for the predicted variable
    plt.figure()
    m_pred[predict_var].sel(time=2050).plot(transform=ccrs.PlateCarree(),
                                            subplot_kws={'projection':␣
↪ccrs.PlateCarree()})
    plt.gca().set_title(f'2050 predict {predict_var} ssp245')
    plt.gca().coastlines()

    #Visual for the actual variable
```

```
    plt.figure()
    beta = Y_test[predict_var].sel(time=2050).plot(transform=ccrs.PlateCarree(),
                                                    subplot_kws={'projection':␣
↪ccrs.PlateCarree()})
    plt.gca().set_title(f'2050 test {predict_var} ssp245')
    plt.gca().coastlines()

    #Visual for the difference between actual and predicted variable
    plt.figure()
    (Y_test[predict_var]-m_pred[predict_var]).sel(time=2050).
↪plot(transform=ccrs.PlateCarree(),
                                                    subplot_kws={'projection':␣
↪ccrs.PlateCarree()})
    plt.gca().set_title(f'2050 difference between test and predict␣
↪{predict_var} ssp245')
    plt.gca().coastlines()

    if predict_var=="pr90" or predict_var=="pr":
        m_pred = m_pred.assign({predict_var: m_pred[predict_var] / 86400})

    # Save test predictions as .nc
    if predict_var == 'diurnal_temperature_range':
        m_pred.to_netcdf(datapath + 'outputs_ssp245_predict_dtr.nc', 'w')
    else:
        m_pred.to_netcdf(datapath + 'outputs_ssp245_predict_{}.nc'.
↪format(predict_var), 'w')
    m_pred.close()
```

```
Epoch 1/30
46/46            22s 453ms/step -
loss: 0.0763 - mse: 0.0763
Epoch 2/30
46/46            21s 454ms/step -
loss: 0.0729 - mse: 0.0729
Epoch 3/30
46/46            21s 451ms/step -
loss: 0.0472 - mse: 0.0472
Epoch 4/30
46/46            21s 454ms/step -
loss: 0.0415 - mse: 0.0415
Epoch 5/30
46/46            21s 455ms/step -
loss: 0.0393 - mse: 0.0393
Epoch 6/30
46/46            21s 452ms/step -
loss: 0.0387 - mse: 0.0387
Epoch 7/30
```

```
46/46             21s 452ms/step -
loss: 0.0383 - mse: 0.0383
Epoch 8/30
46/46             21s 454ms/step -
loss: 0.0379 - mse: 0.0379
Epoch 9/30
46/46             21s 452ms/step -
loss: 0.0376 - mse: 0.0376
Epoch 10/30
46/46             21s 452ms/step -
loss: 0.0373 - mse: 0.0373
Epoch 11/30
46/46             21s 453ms/step -
loss: 0.0370 - mse: 0.0370
Epoch 12/30
46/46             21s 452ms/step -
loss: 0.0367 - mse: 0.0367
Epoch 13/30
46/46             21s 452ms/step -
loss: 0.0364 - mse: 0.0364
Epoch 14/30
46/46             21s 453ms/step -
loss: 0.0362 - mse: 0.0362
Epoch 15/30
46/46             21s 452ms/step -
loss: 0.0360 - mse: 0.0360
Epoch 16/30
46/46             21s 454ms/step -
loss: 0.0359 - mse: 0.0359
Epoch 17/30
46/46             21s 456ms/step -
loss: 0.0358 - mse: 0.0358
Epoch 18/30
46/46             21s 453ms/step -
loss: 0.0357 - mse: 0.0357
Epoch 19/30
46/46             21s 452ms/step -
loss: 0.0356 - mse: 0.0356
Epoch 20/30
46/46             21s 456ms/step -
loss: 0.0355 - mse: 0.0355
Epoch 21/30
46/46             21s 452ms/step -
loss: 0.0354 - mse: 0.0354
Epoch 22/30
46/46             21s 453ms/step -
loss: 0.0353 - mse: 0.0353
Epoch 23/30
```

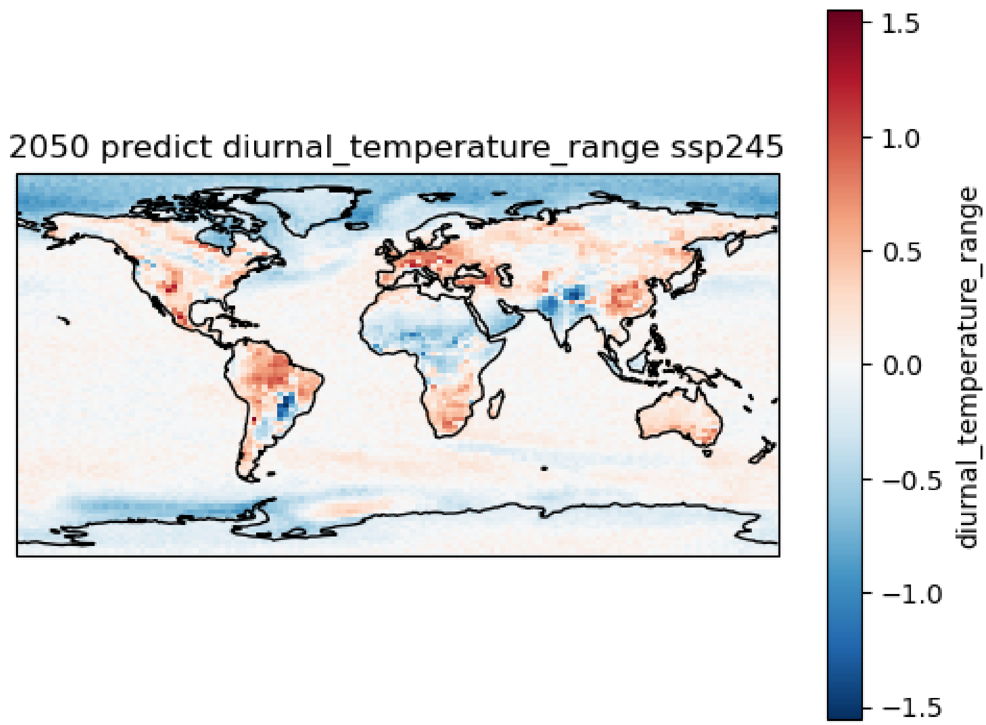
```
46/46              21s 453ms/step -
loss: 0.0353 - mse: 0.0353
Epoch 24/30
46/46              21s 452ms/step -
loss: 0.0352 - mse: 0.0352
Epoch 25/30
46/46              21s 452ms/step -
loss: 0.0351 - mse: 0.0351
Epoch 26/30
46/46              21s 455ms/step -
loss: 0.0349 - mse: 0.0349
Epoch 27/30
46/46              21s 452ms/step -
loss: 0.0348 - mse: 0.0348
Epoch 28/30
46/46              21s 452ms/step -
loss: 0.0347 - mse: 0.0347
Epoch 29/30
46/46              21s 453ms/step -
loss: 0.0346 - mse: 0.0346
Epoch 30/30
46/46              21s 451ms/step -
loss: 0.0344 - mse: 0.0344
3/3             1s 150ms/step
RMSE at 2050: 0.17777889224929544
RMSE at 2100: 0.15018804661310853
RMSE 2045-2055: 0.17997979728423158
RMSE 2090-2100: 0.16606559554829825
RMSE 2050-2100: 0.16934062967987562
RMSE average last 20y: 0.11272139686851651
Epoch 1/30
46/46              22s 452ms/step -
loss: 0.3305 - mse: 0.3305
Epoch 2/30
46/46              21s 453ms/step -
loss: 0.3283 - mse: 0.3283
Epoch 3/30
46/46              21s 451ms/step -
loss: 0.3020 - mse: 0.3020
Epoch 4/30
46/46              21s 453ms/step -
loss: 0.2935 - mse: 0.2935
Epoch 5/30
46/46              21s 453ms/step -
loss: 0.2902 - mse: 0.2902
Epoch 6/30
46/46              21s 451ms/step -
loss: 0.2877 - mse: 0.2877
```

```
Epoch 7/30
46/46             21s 452ms/step -
loss: 0.2869 - mse: 0.2869
Epoch 8/30
46/46             21s 452ms/step -
loss: 0.2864 - mse: 0.2864
Epoch 9/30
46/46             23s 492ms/step -
loss: 0.2860 - mse: 0.2860
Epoch 10/30
46/46             21s 453ms/step -
loss: 0.2857 - mse: 0.2857
Epoch 11/30
46/46             21s 452ms/step -
loss: 0.2853 - mse: 0.2853
Epoch 12/30
46/46             21s 451ms/step -
loss: 0.2850 - mse: 0.2850
Epoch 13/30
46/46             21s 453ms/step -
loss: 0.2846 - mse: 0.2846
Epoch 14/30
46/46             21s 452ms/step -
loss: 0.2842 - mse: 0.2842
Epoch 15/30
46/46             21s 451ms/step -
loss: 0.2836 - mse: 0.2836
Epoch 16/30
46/46             21s 452ms/step -
loss: 0.2828 - mse: 0.2828
Epoch 17/30
46/46             21s 452ms/step -
loss: 0.2817 - mse: 0.2817
Epoch 18/30
46/46             21s 452ms/step -
loss: 0.2804 - mse: 0.2804
Epoch 19/30
46/46             21s 453ms/step -
loss: 0.2791 - mse: 0.2791
Epoch 20/30
46/46             21s 452ms/step -
loss: 0.2779 - mse: 0.2779
Epoch 21/30
46/46             21s 453ms/step -
loss: 0.2770 - mse: 0.2770
Epoch 22/30
46/46             21s 453ms/step -
loss: 0.2762 - mse: 0.2762
```

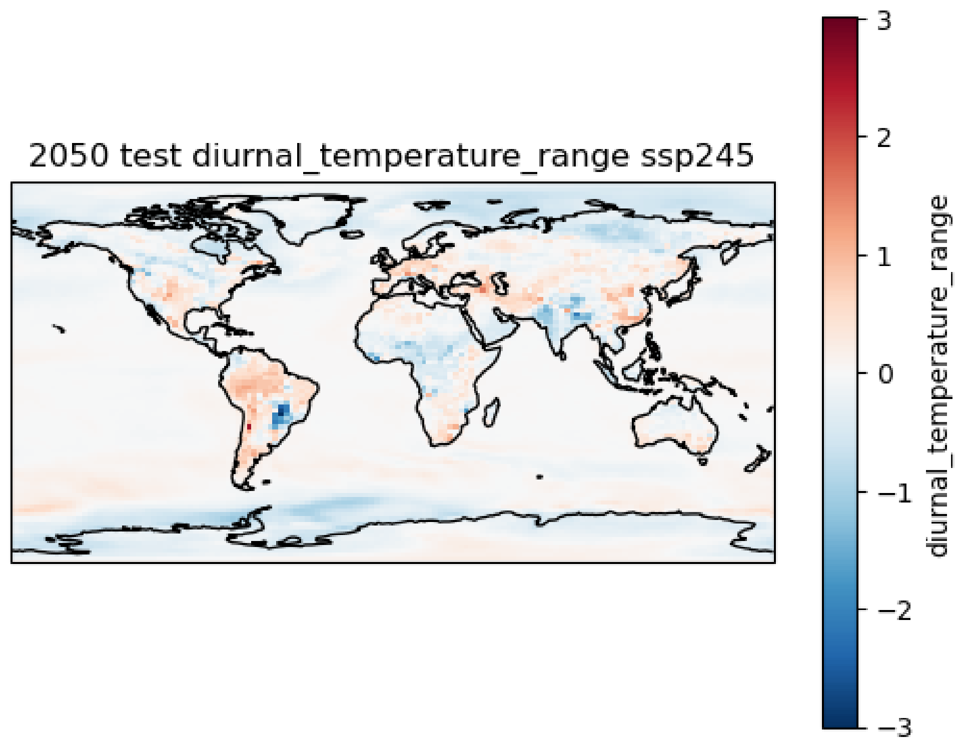
```
Epoch 23/30
46/46            21s 452ms/step -
loss: 0.2756 - mse: 0.2756
Epoch 24/30
46/46            21s 452ms/step -
loss: 0.2751 - mse: 0.2751
Epoch 25/30
46/46            21s 453ms/step -
loss: 0.2747 - mse: 0.2747
Epoch 26/30
46/46            21s 452ms/step -
loss: 0.2745 - mse: 0.2745
Epoch 27/30
46/46            21s 452ms/step -
loss: 0.2743 - mse: 0.2743
Epoch 28/30
46/46            21s 457ms/step -
loss: 0.2741 - mse: 0.2741
Epoch 29/30
46/46            21s 457ms/step -
loss: 0.2739 - mse: 0.2739
Epoch 30/30
46/46            21s 457ms/step -
loss: 0.2738 - mse: 0.2738
3/3           1s 157ms/step
RMSE at 2050: 0.3836859608404659
RMSE at 2100: 0.5115423635507769
RMSE 2045-2055: 0.3837151698223513
RMSE 2090-2100: 0.5038189957529605
RMSE 2050-2100: 0.4543299171185843
RMSE average last 20y: 0.49465478723256573
Epoch 1/30
46/46            22s 457ms/step -
loss: 2.8006 - mse: 2.8006
Epoch 2/30
46/46            21s 459ms/step -
loss: 2.5382 - mse: 2.5382
Epoch 3/30
46/46            21s 463ms/step -
loss: 2.4862 - mse: 2.4862
Epoch 4/30
46/46            21s 458ms/step -
loss: 2.4682 - mse: 2.4682
Epoch 5/30
46/46            21s 465ms/step -
loss: 2.4584 - mse: 2.4584
Epoch 6/30
46/46            22s 469ms/step -
```

```
loss: 2.4461 - mse: 2.4461
Epoch 7/30
46/46            22s 468ms/step -
loss: 2.4333 - mse: 2.4333
Epoch 8/30
46/46            21s 459ms/step -
loss: 2.4172 - mse: 2.4172
Epoch 9/30
46/46            21s 454ms/step -
loss: 2.4017 - mse: 2.4017
Epoch 10/30
46/46            21s 456ms/step -
loss: 2.3898 - mse: 2.3898
Epoch 11/30
46/46            21s 457ms/step -
loss: 2.3816 - mse: 2.3816
Epoch 12/30
46/46            21s 456ms/step -
loss: 2.3757 - mse: 2.3757
Epoch 13/30
46/46            21s 454ms/step -
loss: 2.3711 - mse: 2.3711
Epoch 14/30
46/46            21s 457ms/step -
loss: 2.3675 - mse: 2.3675
Epoch 15/30
46/46            21s 461ms/step -
loss: 2.3645 - mse: 2.3645
Epoch 16/30
46/46            22s 473ms/step -
loss: 2.3620 - mse: 2.3620
Epoch 17/30
46/46            22s 469ms/step -
loss: 2.3598 - mse: 2.3598
Epoch 18/30
46/46            21s 460ms/step -
loss: 2.3578 - mse: 2.3578
Epoch 19/30
46/46            21s 461ms/step -
loss: 2.3561 - mse: 2.3561
Epoch 20/30
46/46            21s 458ms/step -
loss: 2.3543 - mse: 2.3543
Epoch 21/30
46/46            21s 460ms/step -
loss: 2.3526 - mse: 2.3526
Epoch 22/30
46/46            21s 460ms/step -
```

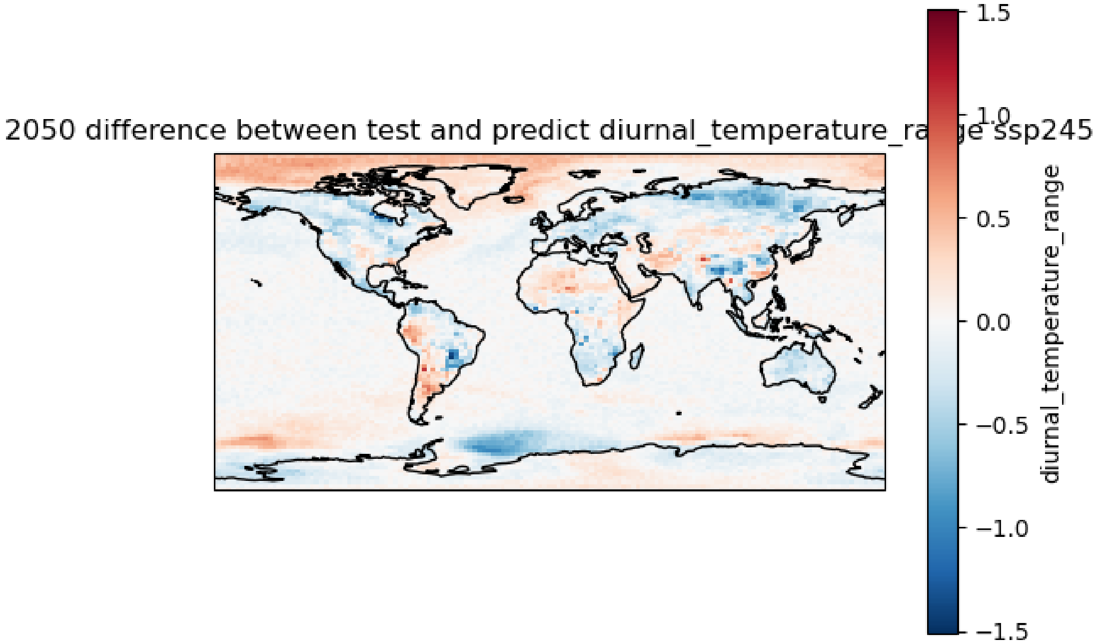
```
loss: 2.3511 - mse: 2.3511
Epoch 23/30
46/46            21s 460ms/step -
loss: 2.3493 - mse: 2.3493
Epoch 24/30
46/46            21s 456ms/step -
loss: 2.3475 - mse: 2.3475
Epoch 25/30
46/46            21s 453ms/step -
loss: 2.3458 - mse: 2.3458
Epoch 26/30
46/46            21s 452ms/step -
loss: 2.3451 - mse: 2.3451
Epoch 27/30
46/46            21s 451ms/step -
loss: 2.3437 - mse: 2.3437
Epoch 28/30
46/46            21s 453ms/step -
loss: 2.3433 - mse: 2.3433
Epoch 29/30
46/46            21s 450ms/step -
loss: 2.3432 - mse: 2.3432
Epoch 30/30
46/46            21s 455ms/step -
loss: 2.3433 - mse: 2.3433
3/3            1s 157ms/step
RMSE at 2050: 1.0534298728244667
RMSE at 2100: 1.6638128757448276
RMSE 2045-2055: 1.0555962002945691
RMSE 2090-2100: 1.6077949158754128
RMSE 2050-2100: 1.3307765903662618
RMSE average last 20y: 1.5445240256222896
```

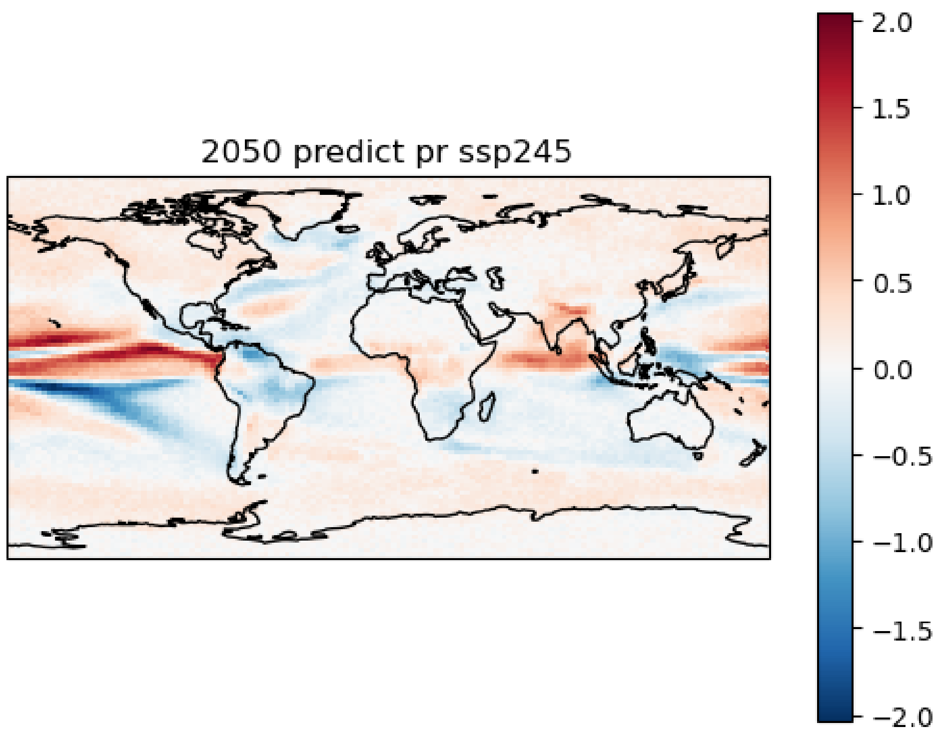


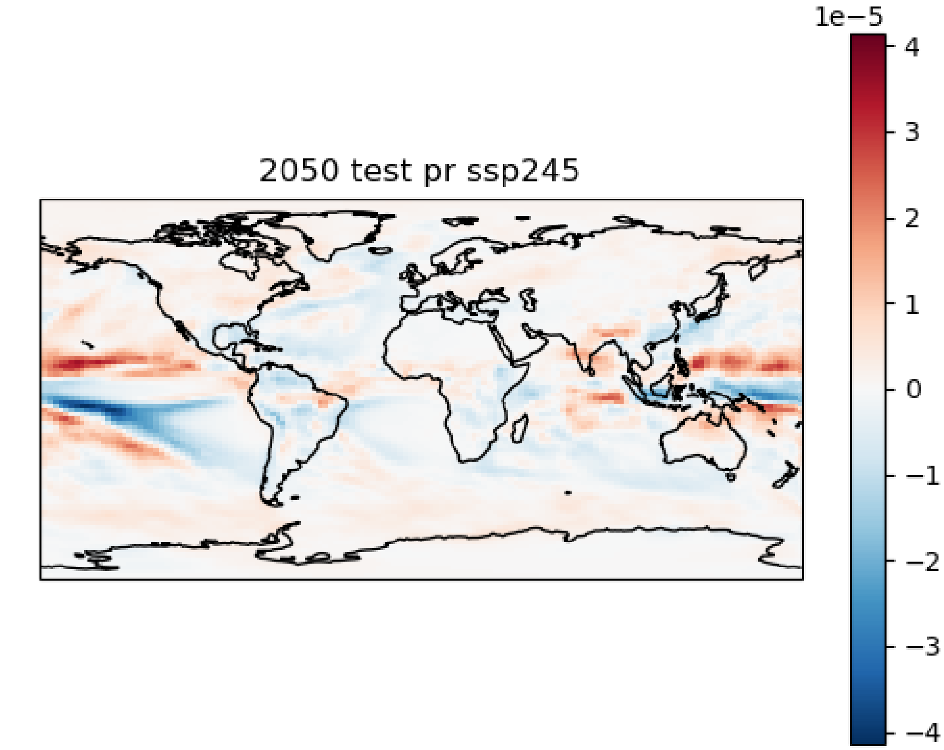
2050 predict diurnal_temperature_range ssp245
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
diurnal_temperature_range

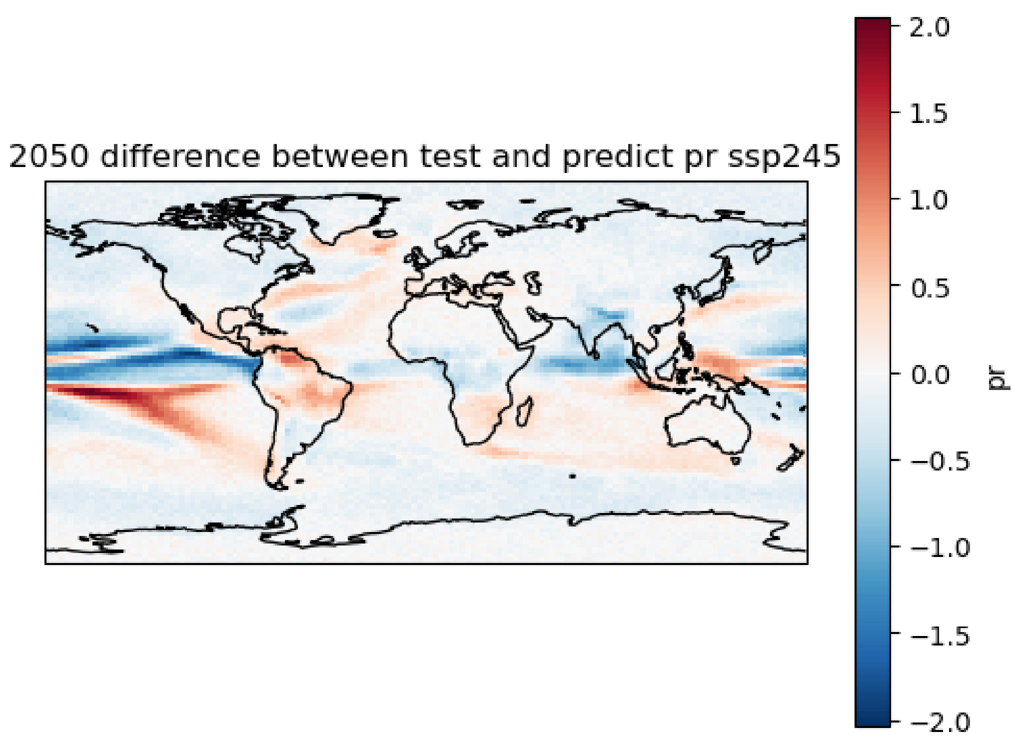


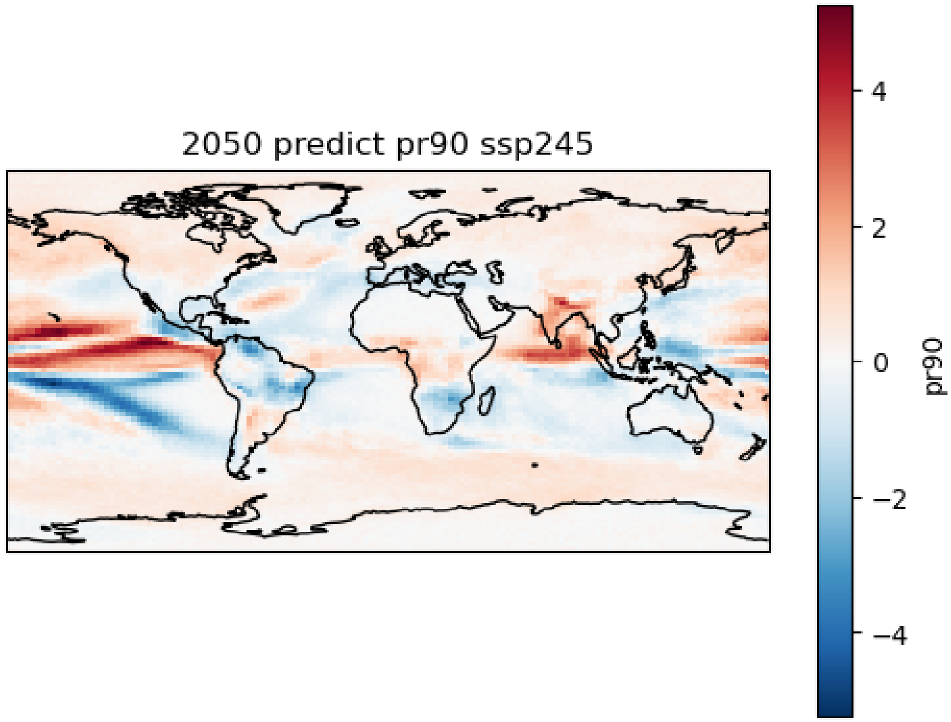
2050 test diurnal_temperature_range ssp245
3
2
1
0
−1
−2
−3
diurnal_temperature_range

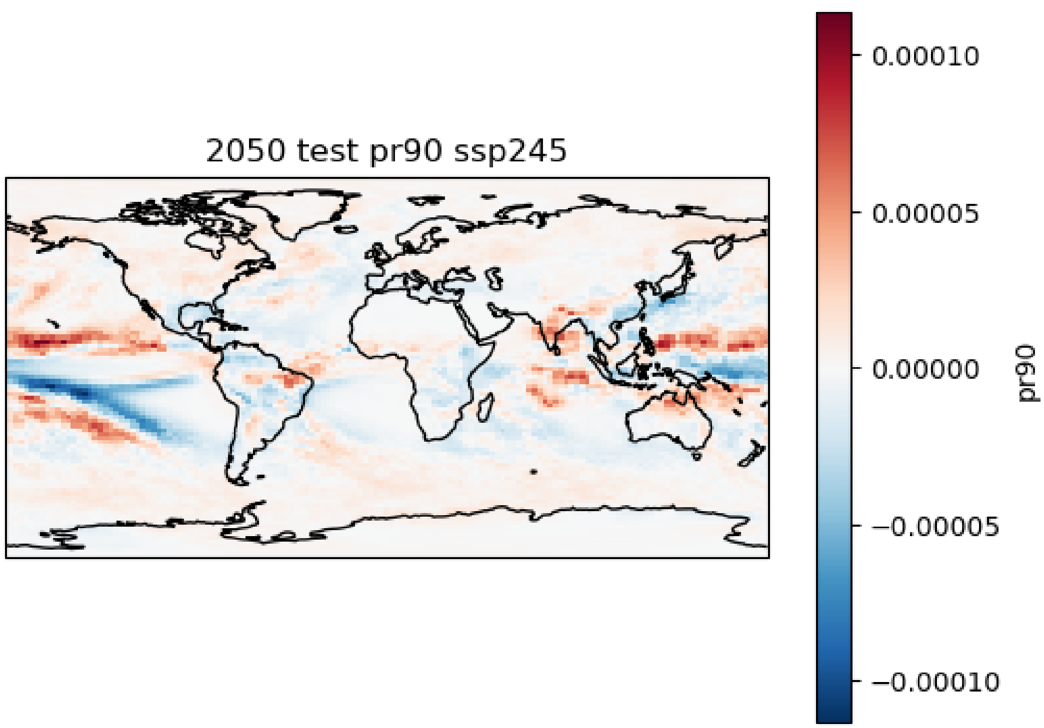

2050 difference between test and predict diurnal_temperature_range ssp245
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
diurnal_temperature_range

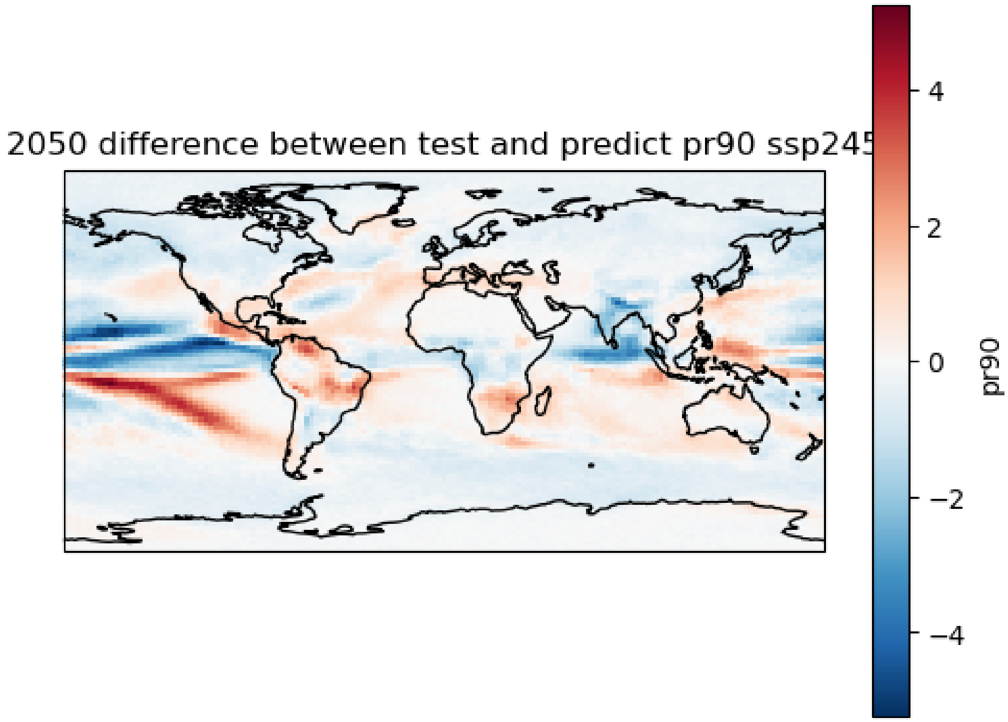

2050 predict pr ssp245
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
pr


2050 test pr ssp245
1e−5
4
3
2
1
0
−1
−2
−3
−4
pr


2050 difference between test and predict pr ssp245
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
pr


2050 predict pr90 ssp245
4
2
0
−2
−4
pr90


2050 test pr90 ssp245
0.00010
0.00005
0.00000
−0.00005
−0.00010
pr90


2050 difference between test and predict pr90 ssp245
4
2
0
−2
−4
pr90

[ ]: