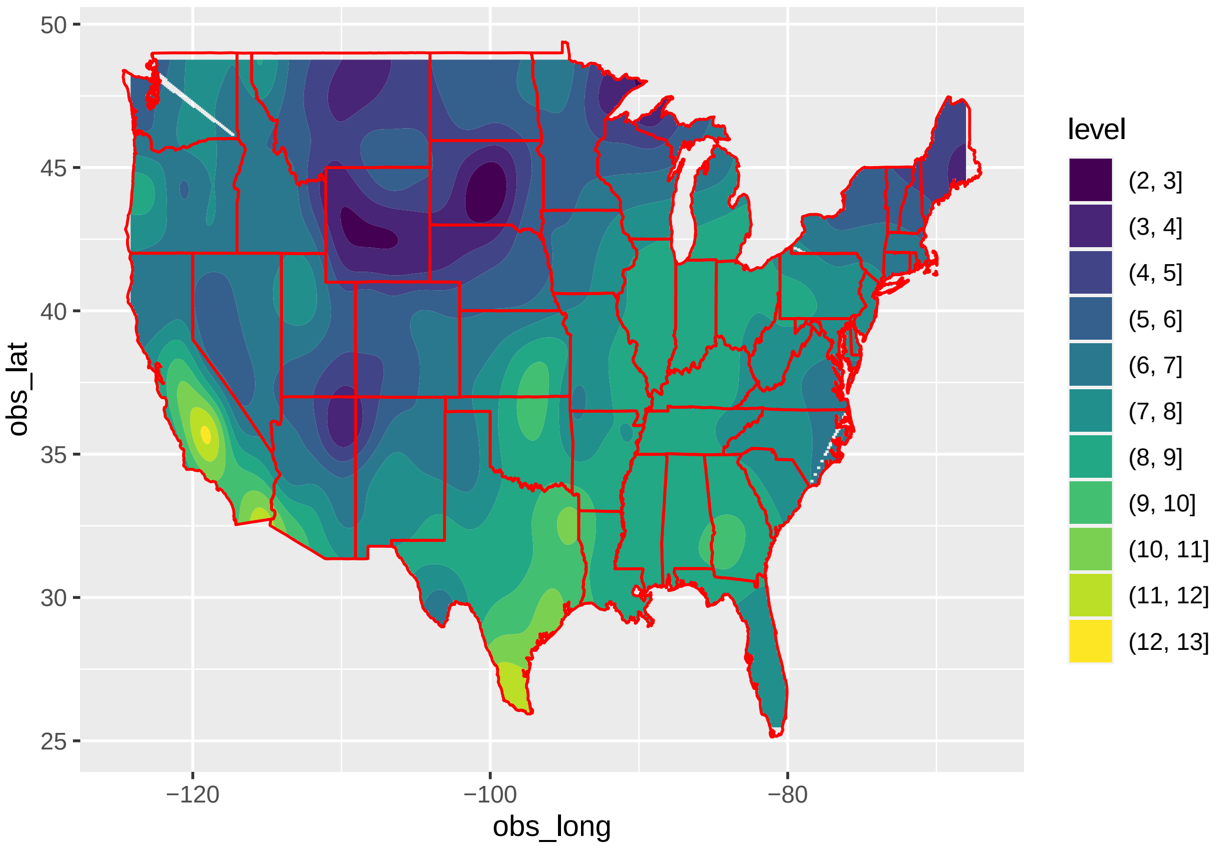
pm_large

Qi Wang

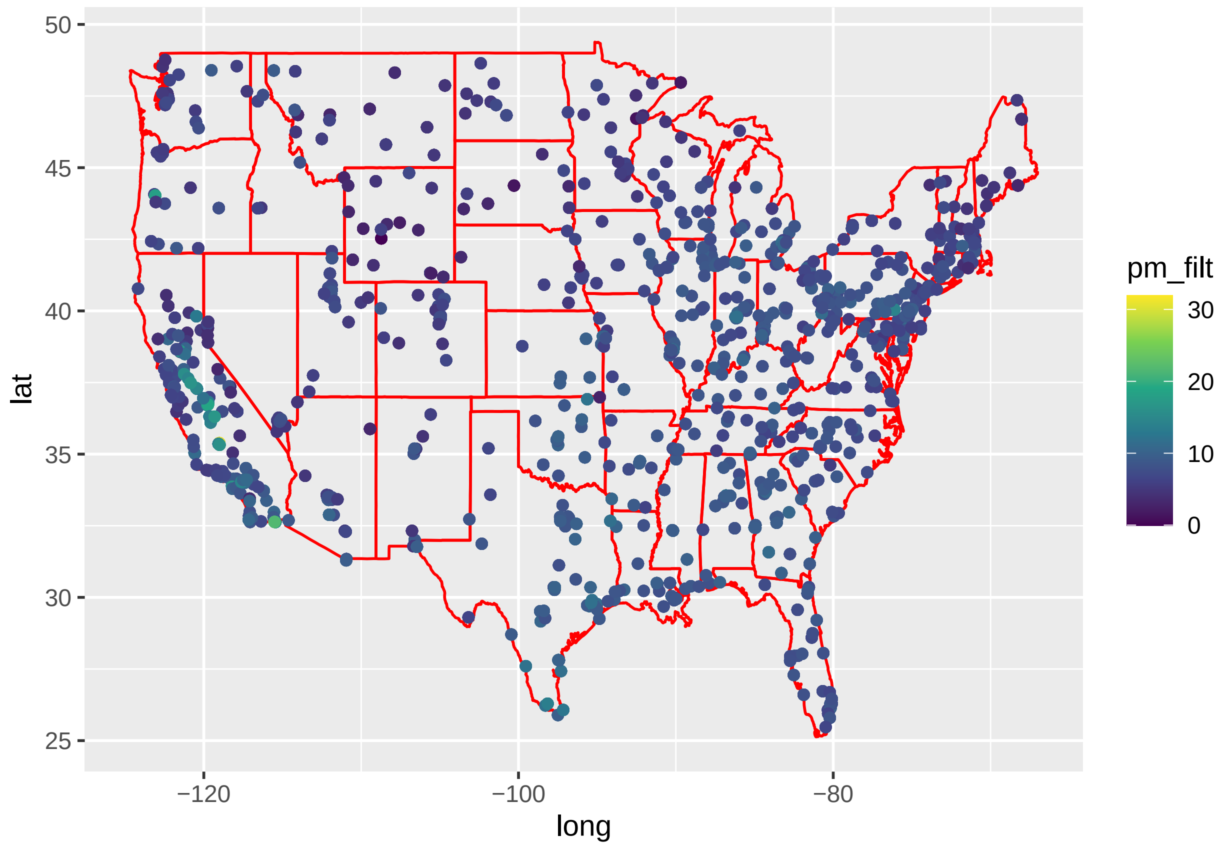
2023/4/4

```r
us_map <- map_data("state", region=c("alabama", "arizona", "arkansas", "california", "colorado", "conne
                                     "delaware", "florida", "georgia", "idaho", "illinois", "indiana",
                                     "iowa", "kansas", "kentucky", "louisiana", "maine", "maryland",
                                     "massachusetts", "michigan", "minnesota", "mississippi", "missouri
                                     "montana", "nebraska", "nevada", "new hampshire", "new jersey",
                                     "new mexico", "new york", "north carolina", "north dakota", "ohio"
                                     "oklahoma", "oregon", "pennsylvania", "rhode island", "south caroli
                                     "south dakota", "tennessee", "texas", "utah", "vermont", "virginia
                                     "washington", "west virginia", "wisconsin", "wyoming"))

states = c("Alabama", "Arizona", "Arkansas", "California", "Colorado", "Connecticut",
           "Delaware", "Florida", "Georgia", "Idaho", "Illinois", "Indiana",
           "Iowa", "Kansas", "Kentucky", "Louisiana", "Maine", "Maryland",
           "Massachusetts", "Michigan", "Minnesota", "Mississippi", "Missouri",
           "Montana", "Nebraska", "Nevada", "New Hampshire", "New Jersey",
           "New Mexico", "New York", "North Carolina", "North Dakota", "Ohio",
           "Oklahoma", "Oregon", "Pennsylvania", "Rhode Island", "South Carolina",
           "South Dakota", "Tennessee", "Texas", "Utah", "Vermont", "Virginia",
            "West Virginia", "Wisconsin", "Wyoming", "Washington")



pm_large_all <- read.csv(here::here("annual_conc_by_monitor_2022.csv"))

pm_large_all <- pm_large_all[which(pm_large_all$Parameter.Name == "PM2.5 - Local Conditions"),]

point_aba <- unique(c(which(pm_large_all$Longitude <= -130), which(pm_large_all$Latitude <=20 ) ))

pm_large_all <- pm_large_all[-point_aba,]

long <- pm_large_all$Longitude + rnorm( length(pm_large_all$Longitude), 0 , sd = 0.001)
lat <- pm_large_all$Latitude + rnorm( length(pm_large_all$Latitude), 0 , sd = 0.001)
pm_filt <- pm_large_all$Arithmetic.Mean



pm_large <- as.data.frame(cbind(long, lat, pm_filt))

coordinates(pm_large) <- ~ long + lat
coords <- cbind(long, lat)

x.res <- 500
```

```
y.res <- 500
surf <- mba.surf(cbind(coords, pm_filt), no.X = x.res, no.Y = y.res, h = 5, m = 2, extend = TRUE)$xyz.es

all_inside <- NULL
exp_grid <- expand.grid(surf$x, surf$y)
for (i in 1:length(states)) {
  tem = spBayes::pointsInPoly(as.matrix(map_data("state", region = states[i])[,1:2]),as.matrix(exp_grid)
  all_inside <- unique(c(all_inside, tem))

}
obs_pm <- surf$z[all_inside]
obs_long <-  as.matrix(exp_grid)[all_inside,1]
obs_lat <-  as.matrix(exp_grid)[all_inside,2]
p1 <-
ggplot() +
  geom_contour_filled(aes(x = obs_long, y = obs_lat, z = obs_pm))+
   geom_path(data = us_map, aes(x = long, y = lat, group = group), color = "red")
p2 <-
ggplot() +
  geom_path(data = us_map, aes(x = long, y = lat, group = group), color = "red") +
  geom_point(aes(x = long, y = lat, color = pm_filt)) +
  scale_color_viridis_c()

p1
```



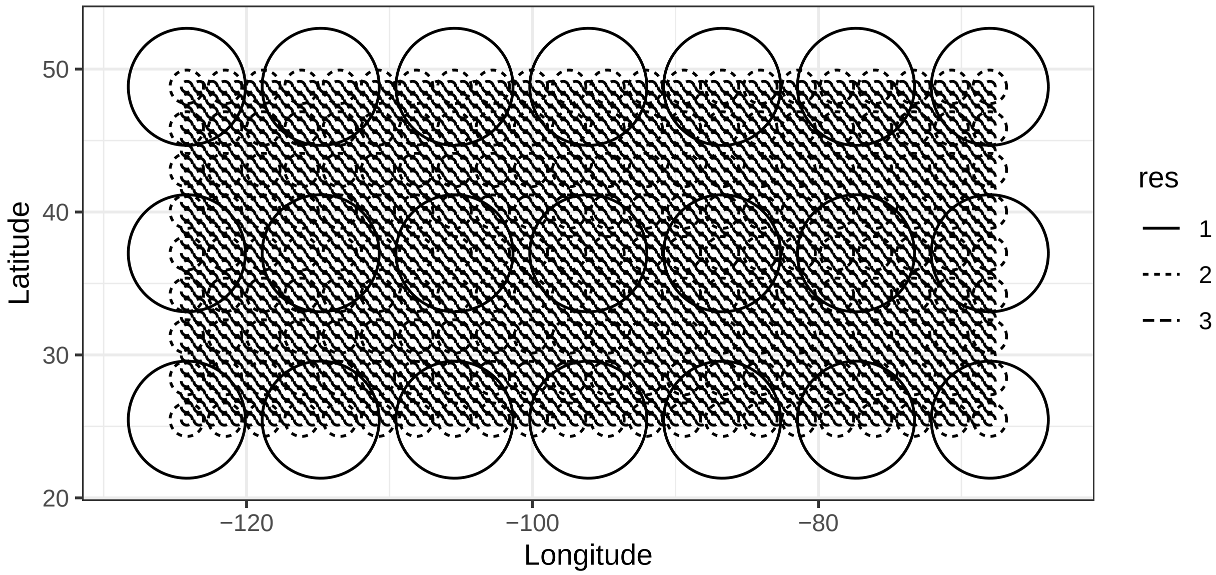
```
p2
```



Basis Function Generation

```
# pm_large <- as.matrix(pm_large)

gridbasis1 <- auto_basis(mainfold = plane(), data = pm_large, nres = 1, type = "Gaussian", regular = 1)
gridbasis2 <- auto_basis(mainfold = plane(), data = pm_large, nres = 2, type = "Gaussian", regular = 1)
gridbasis3 <- auto_basis(mainfold = plane(), data = pm_large, nres = 3, type = "Gaussian", regular = 1)

show_basis(gridbasis3) +
  coord_fixed() +
  xlab("Longitude") +
  ylab("Latitude")
```

```
## Note: show_basis assumes spherical distance functions when plotting
```




```
basis_1 <- matrix(NA, nrow = nrow(pm_large), ncol = length(gridbasis1@fn))
for (i in 1:length(gridbasis1@fn)) {
  basis_1[,i] <- gridbasis1@fn[[i]](coordinates(pm_large))
}

basis_2 <- matrix(NA, nrow = nrow(pm_large), ncol = length(gridbasis2@fn))
for (i in 1:length(gridbasis2@fn)) {
  basis_2[,i] <- gridbasis2@fn[[i]](coordinates(pm_large))
}

basis_3 <- matrix(NA, nrow = nrow(pm_large), ncol = length(gridbasis3@fn))
for (i in 1:length(gridbasis3@fn)) {
  basis_3[,i] <- gridbasis3@fn[[i]](coordinates(pm_large))
}


basis_use <- basis_3[,-(1:ncol(basis_2))]

depth <- 3
shape_row <- length(table(gridbasis3@df[which(gridbasis3@df$res == depth) , 2 ]))
shape_col <- length(table(gridbasis3@df[which(gridbasis3@df$res == depth) , 1 ]))
basis_arr <- array(NA, dim = c(nrow(pm_large), shape_row, shape_col))
for (i in 1:nrow(pm_large)) {
  basis_arr[i,,] <- matrix(basis_use[i,], nrow = shape_row, ncol = shape_col, byrow = T)
}
```

Deep Kriging Model

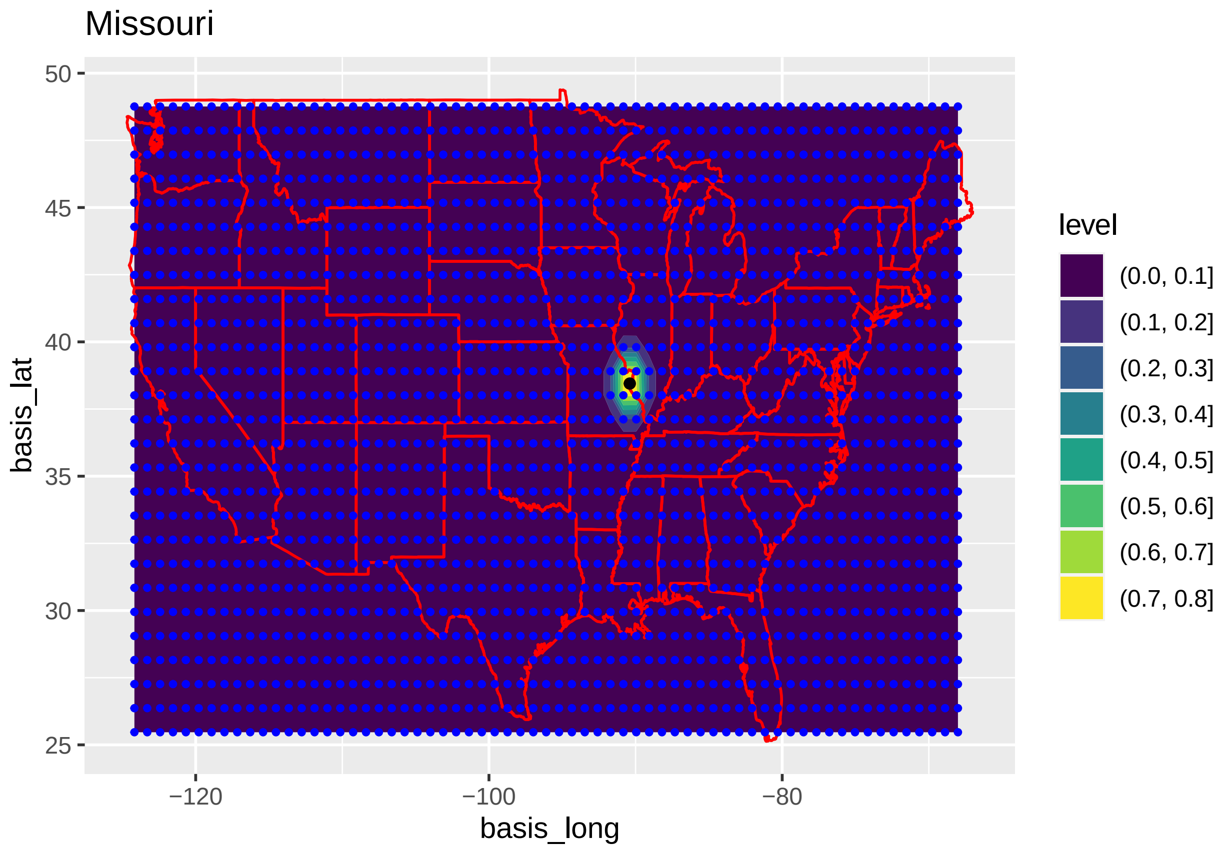
```
set.seed(0)
train_index <- sample(1:nrow(pm_large), 7000, replace = FALSE)
```

```
basis_tr <- basis_arr[train_index,,]
basis_te <- basis_arr[-train_index,,]

state_tr <- pm_large_all$State.Name[train_index]
long_tr <- pm_large_all$Longitude[train_index]
lat_tr <- pm_large_all$Latitude[train_index]

x_tr <- array_reshape(basis_tr, c(nrow(basis_tr), shape_row*shape_col))  # So we want to reshape each o
x_te <- array_reshape(basis_te, c(nrow(basis_te), shape_row*shape_col))  # Same as prervious step

pm_tr <- pm_filt[train_index]
pm_te <- pm_filt[-train_index]


model_dk <- keras_model_sequential()

model_dk %>%
  layer_dense(units = 256, activation = 'relu', input_shape = c(ncol(x_tr))) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 1, activation = 'linear')

# Compile the model

model_dk %>% compile(
  loss = "mse",
  optimizer = optimizer_adam(),
  metrics = list("mse")
)

mod_train_dk <- model_dk %>%
  fit(x = x_tr, y = pm_tr, epochs = 30, batch_size = 128,
      validation_split = 0.2)

loss_dk <- model_dk %>%
          evaluate(x_te, pm_te)
```

Convolutional Kriging Model

```
# Define a few parameters to be used in the CNN model
batch_size <- 128
epochs <- 50
```

```r
basis_long <- rep(NA, shape_col*shape_row)
basis_lat <- rep(NA, shape_col*shape_row)

for (i in 1:(shape_row*shape_col)) {
  basis_long[i] <- gridbasis3@pars[[ ncol(basis_2)+i ]]$loc[1]
  basis_lat[i] <- gridbasis3@pars[[ ncol(basis_2)+i ]]$loc[2]
}



# Input image dimensions
img_rows <- shape_row
img_cols <- shape_col

index_image = 1000 ## change this index to see different image. For now, we see the 1000th picture
input_matrix <- matrix( as.vector(basis_tr[index_image,,]), nrow = shape_row, ncol = shape_col)


#exp_grid <- expand.grid(x = 1:img_cols, y = 1:img_rows)

ggplot() +
  #geom_contour_filled(aes(x = exp_grid$x, y = exp_grid$y, z = as.vector(input_matrix))) +
  geom_contour_filled(aes(x = basis_long, y = basis_lat, z = as.vector(t(input_matrix)))) +
  labs(title = state_tr[index_image]) +
  geom_path(data = us_map, aes(x = long, y = lat, group = group), color = "red") +
  geom_point(aes( x = basis_long, y = basis_lat), color = "blue", size = 0.8) +
  geom_point(aes( x = long_tr[index_image], y = lat_tr[index_image]), color = "black")
```




```r
x_tr <- array_reshape(basis_tr, c(nrow(basis_tr), img_rows, img_cols, 1))
x_te <- array_reshape(basis_te, c(nrow(basis_te), img_rows, img_cols, 1))
input_shape <- c(img_rows, img_cols, 1)



model_ck <- keras_model_sequential() %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu', input_shape = input_shape) %>%
  #layer_conv_2d(filters = 32, kernel_size = c(2,2), activation = 'relu') %>%
  #layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_dropout(rate = 0.3) %>%
  layer_flatten() %>%
  layer_dense(units = 256, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 1, activation = 'linear')


model_ck %>% compile(
  loss = "mse",
  optimizer = optimizer_adam(),
  metrics = list("mse")
)
```

```r
mod_train_ck <- model_ck %>%
  fit(x = x_tr, y = pm_tr, epochs = 30, batch_size = 128,
      validation_split = 0.2)

loss_ck <- model_ck %>%
          evaluate(x_te, pm_te)
```

```r
rbind(loss_dk, loss_ck)
```

```
##             loss      mse
## loss_dk 1.543647 1.543647
## loss_ck 1.359268 1.359268
```