```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
import os
import cv2

import warnings
warnings.filterwarnings('ignore')

from PIL import Image
import tensorflow as tf

from sklearn.model_selection import train_test_split
from skimage.transform import resize
from sklearn.metrics import accuracy_score
from tensorflow.keras import models, layers
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
import keras.utils
from keras import utils as np_utils
```

```
data = []
labels = []
classes = 43
cur_path = '../input/gtsrb-german-traffic-sign/Train'

for i in os.listdir(cur_path):
    dir = cur_path + '/' + i
    for j in os.listdir(dir):
        img_path = dir+'/'+j
        img = cv2.imread(img_path,-1)
        img = cv2.resize(img, (30,30), interpolation =
cv2.INTER_NEAREST)
        data.append(img)
        labels.append(i)

data = np.array(data)
labels = np.array(labels)
print(data.shape, labels.shape)
```

```
(39209, 30, 30, 3) (39209,)
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define data generator with augmentation parameters
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=False)

# Create an iterator for the original dataset
it = datagen.flow(data, labels, batch_size=len(data), shuffle=False)

# Generate augmented images
aug_images, aug_labels = next(it)

# Combine original and augmented data
data_aug = np.concatenate([data, aug_images])
labels_aug = np.concatenate([labels, aug_labels])

# Shuffle the augmented data
shuffle_indexes = np.arange(data_aug.shape[0])
np.random.shuffle(shuffle_indexes)
data_aug = data_aug[shuffle_indexes]
labels_aug = labels_aug[shuffle_indexes]
```

```
X_train, X_val, y_train, y_val = train_test_split(data_aug,
labels_aug, test_size=0.2, random_state=42)
print(X_train.shape, X_val.shape, y_train.shape, y_val.shape)
```

```
(62734, 30, 30, 3) (15684, 30, 30, 3) (62734,) (15684,)
```

```
y_train = to_categorical(y_train, 43)
y_val = to_categorical(y_val, 43)
```

```
from numpy.random import seed
seed(508)
np.random.seed(508)
```

```
model = tf.keras.Sequential()

# Add convolutional and pooling layers
model.add(Conv2D(filters=32, kernel_size=(3,3), activation="relu",
input_shape= (30,30,3)))
model.add(Conv2D(filters=64, kernel_size=(3,3), activation="relu",
padding='same'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(layers.BatchNormalization())
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=128,kernel_size=(3,3),activation="relu",paddi
ng='same'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(layers.BatchNormalization())
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=512,kernel_size=(3,3),activation="relu",paddi
ng='same'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(layers.BatchNormalization())
model.add(Dropout(rate=0.25))

# Flatten the output from the convolutional layers
model.add(Flatten(input_shape=(4, 4, 512)))

# Add dense layers
model.add(Dense(4000, activation='relu'))
model.add(Dense(3000, activation='relu'))
model.add(Dense(1000, activation='relu'))
model.add(Dense(43, activation="softmax"))

# Compile the model
model.compile(loss = 'categorical_crossentropy', optimizer ='adam',
metrics =['accuracy'])

# Print the model summary
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 28, 28, 32)        896

 conv2d_5 (Conv2D)           (None, 28, 28, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 batch_normalization_3 (Batc  (None, 14, 14, 64)       256
 hNormalization)

 dropout_3 (Dropout)         (None, 14, 14, 64)        0

 conv2d_6 (Conv2D)           (None, 14, 14, 128)       73856

 max_pooling2d_4 (MaxPooling  (None, 7, 7, 128)        0
 2D)

 batch_normalization_4 (Batc  (None, 7, 7, 128)        512
 hNormalization)

 dropout_4 (Dropout)         (None, 7, 7, 128)         0

 conv2d_7 (Conv2D)           (None, 7, 7, 512)         590336

 max_pooling2d_5 (MaxPooling  (None, 3, 3, 512)        0
 2D)

 batch_normalization_5 (Batc  (None, 3, 3, 512)        2048
 hNormalization)

 dropout_5 (Dropout)         (None, 3, 3, 512)         0

 flatten_1 (Flatten)         (None, 4608)              0

 dense_4 (Dense)             (None, 4000)              18436000

 dense_5 (Dense)             (None, 3000)              12003000

 dense_6 (Dense)             (None, 1000)              3001000

 dense_7 (Dense)             (None, 43)                43043

=================================================================
Total params: 34,169,443
Trainable params: 34,168,035
Non-trainable params: 1,408
_________________________________________________________________
```

```
from tensorflow.keras.optimizers import Adam
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(learning_rate=0.001),
              metrics=['accuracy'])

from tensorflow.keras.callbacks import EarlyStopping

# Create the EarlyStopping callback
early_stop = EarlyStopping(monitor='val_loss', patience=3)

# Train the model with EarlyStopping callback
history = model.fit(X_train, y_train,
                    batch_size=32,
                    epochs=20,
                    verbose=1,
                    validation_data=(X_val, y_val),
                    callbacks=[early_stop])
##callbacks=[early_stop]
```

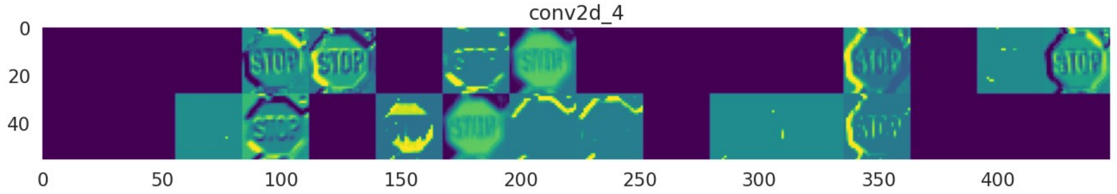
```
Epoch 1/20

2023-05-04 13:48:29.915761: E
tensorflow/core/grappler/optimizers/meta_optimizer.cc:954] layout
failed: INVALID_ARGUMENT: Size of values 0 does not match size of
permutation 4 @ fanin shape insequential_1/dropout_3/dropout/SelectV2-
2-TransposeNHWCToNCHW-LayoutOptimizer
```

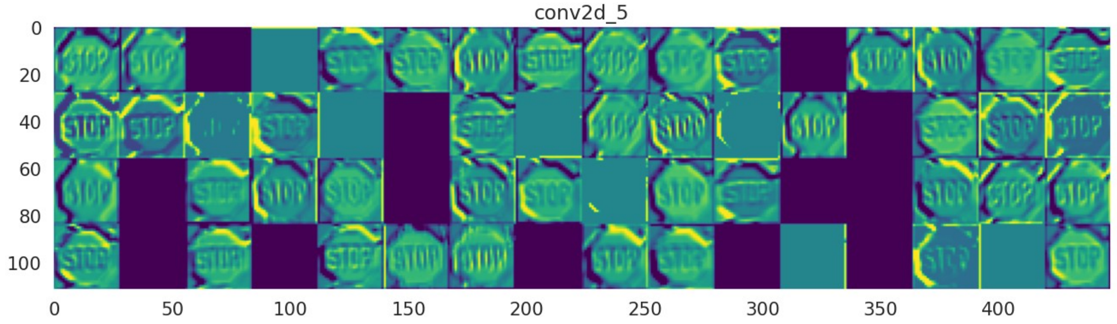
```
1961/1961 [==============================] - 25s 11ms/step - loss:
0.7913 - accuracy: 0.7726 - val_loss: 0.3468 - val_accuracy: 0.8988
Epoch 2/20
1961/1961 [==============================] - 22s 11ms/step - loss:
0.1973 - accuracy: 0.9472 - val_loss: 0.0813 - val_accuracy: 0.9791
Epoch 3/20
1961/1961 [==============================] - 22s 11ms/step - loss:
0.1683 - accuracy: 0.9588 - val_loss: 0.0677 - val_accuracy: 0.9848
Epoch 4/20
1961/1961 [==============================] - 22s 11ms/step - loss:
0.1329 - accuracy: 0.9683 - val_loss: 0.0953 - val_accuracy: 0.9788
Epoch 5/20
1961/1961 [==============================] - 21s 11ms/step - loss:
0.1411 - accuracy: 0.9690 - val_loss: 0.0606 - val_accuracy: 0.9846
Epoch 6/20
1961/1961 [==============================] - 22s 11ms/step - loss:
0.1099 - accuracy: 0.9753 - val_loss: 0.4562 - val_accuracy: 0.9619
Epoch 7/20
1961/1961 [==============================] - 21s 11ms/step - loss:
0.0945 - accuracy: 0.9788 - val_loss: 0.0289 - val_accuracy: 0.9923
Epoch 8/20
1961/1961 [==============================] - 21s 11ms/step - loss:
0.0920 - accuracy: 0.9799 - val_loss: 0.0531 - val_accuracy: 0.9872
Epoch 9/20
1961/1961 [==============================] - 22s 11ms/step - loss:
0.0827 - accuracy: 0.9826 - val_loss: 0.0482 - val_accuracy: 0.9927
Epoch 10/20
1961/1961 [==============================] - 21s 11ms/step - loss:
0.0780 - accuracy: 0.9833 - val_loss: 0.0434 - val_accuracy: 0.9927
```

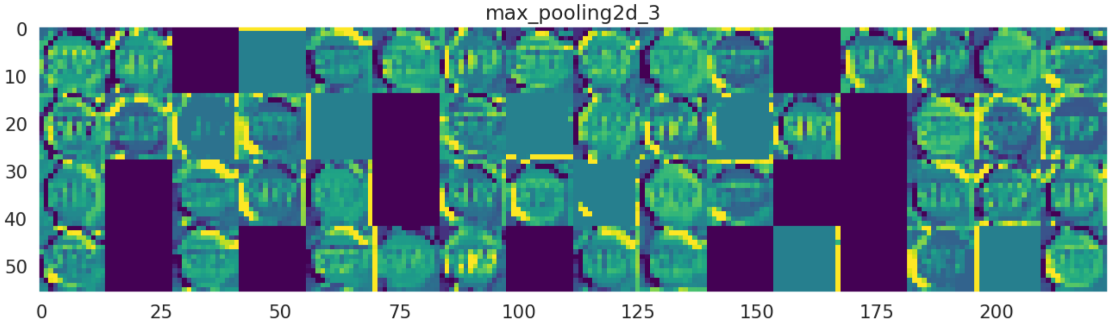
```python
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import Model

# Load an example image
img_path = '/kaggle/input/gtsrb-german-traffic-sign/Test/00093.png'
img = load_img(img_path, target_size=(30, 30))
img_array = img_to_array(img)
img_array = img_array.astype('float32') / 255.0

# Create a new model that outputs the activations of all layers
layer_outputs = [layer.output for layer in model.layers]
activation_model = Model(inputs=model.input, outputs=layer_outputs)

# Predict the class probabilities and get the layer activations
activations = activation_model.predict(img_array.reshape(1, 30, 30,
3))

# Visualize the activations
```

```
layer_names = [layer.name for layer in model.layers]
images_per_row = 16

for layer_name, layer_activation in zip(layer_names, activations):
    n_features = layer_activation.shape[-1]
    size = layer_activation.shape[1]
    n_cols = n_features // images_per_row
    display_grid = np.zeros((size * n_cols, images_per_row * size))
    for col in range(n_cols):
        for row in range(images_per_row):
            channel_image = layer_activation[0,:, :, col *
images_per_row + row]
            channel_image -= channel_image.mean()
            channel_image /= channel_image.std()
            channel_image *= 64
            channel_image += 128
            channel_image = np.clip(channel_image, 0,
255).astype('uint8')
            display_grid[col * size : (col + 1) * size,
                         row * size : (row + 1) * size] =
channel_image
    scale = 1. / size
    plt.figure(figsize=(scale * display_grid.shape[1],
                        scale * display_grid.shape[0]))
    plt.title(layer_name)
    plt.grid(False)
    plt.imshow(display_grid, aspect='auto', cmap='viridis')
    plt.show()  # added this line to display the plot
```

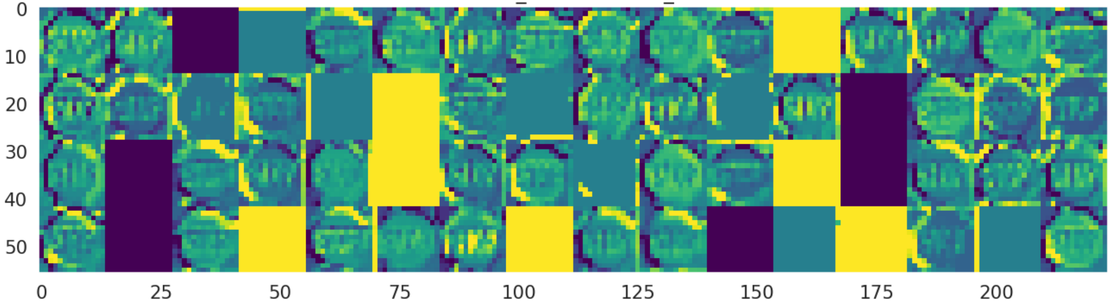
```
1/1 [==============================] - 0s 160ms/step
```



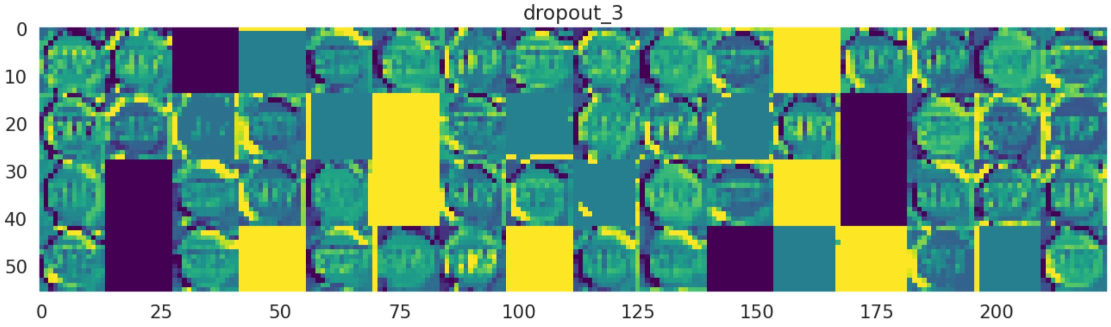




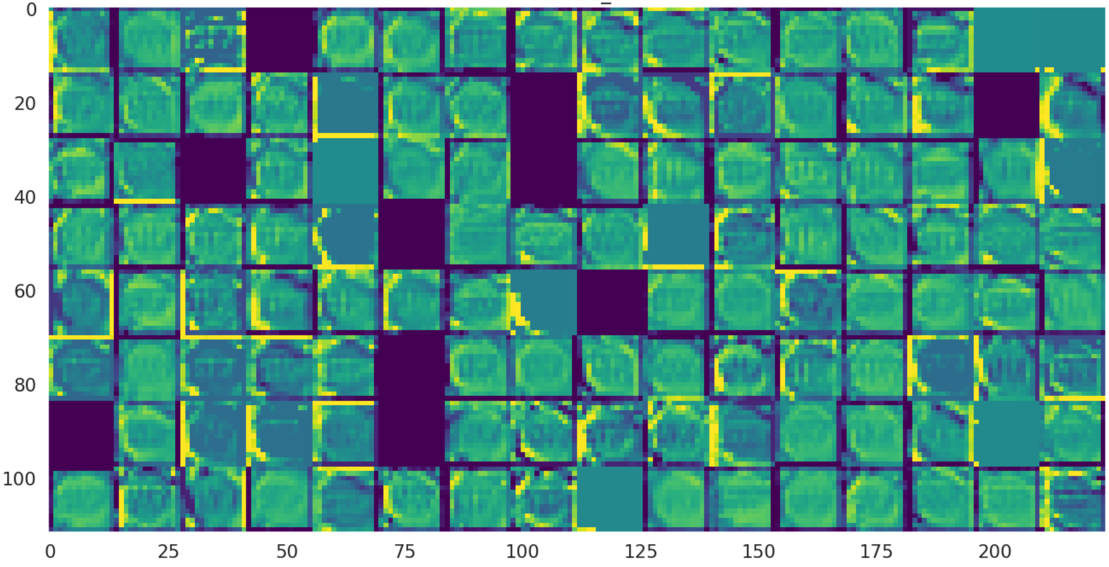
max_pooling2d_3



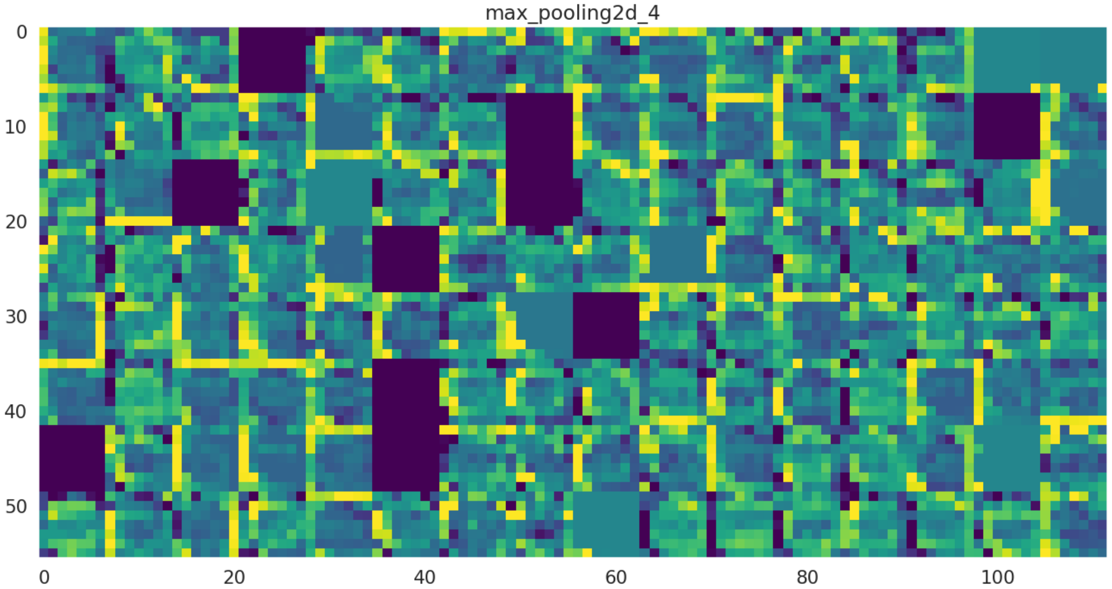
batch_normalization_3

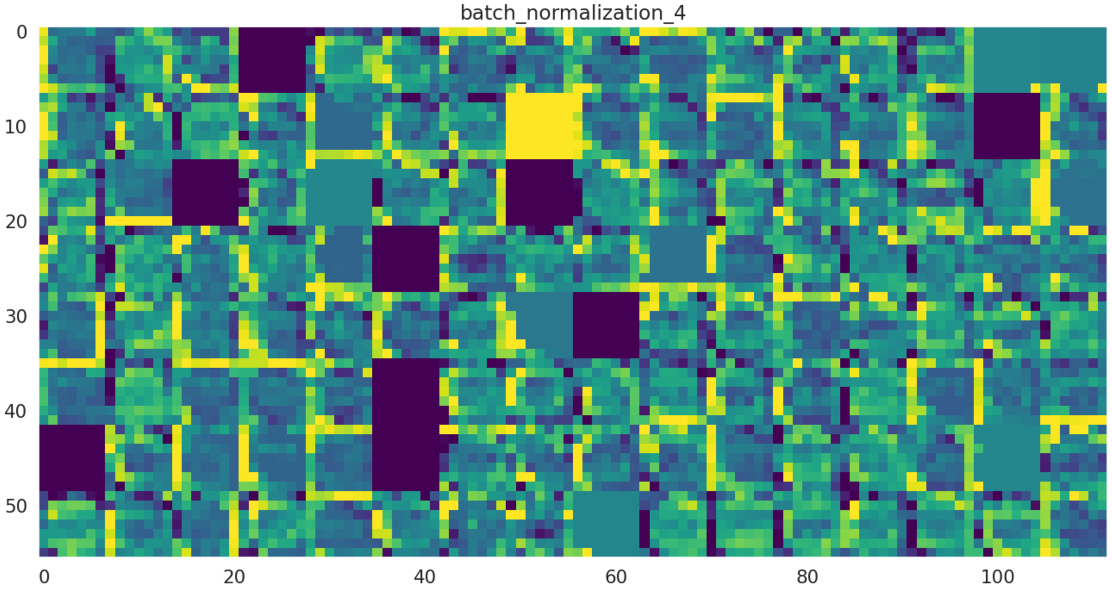


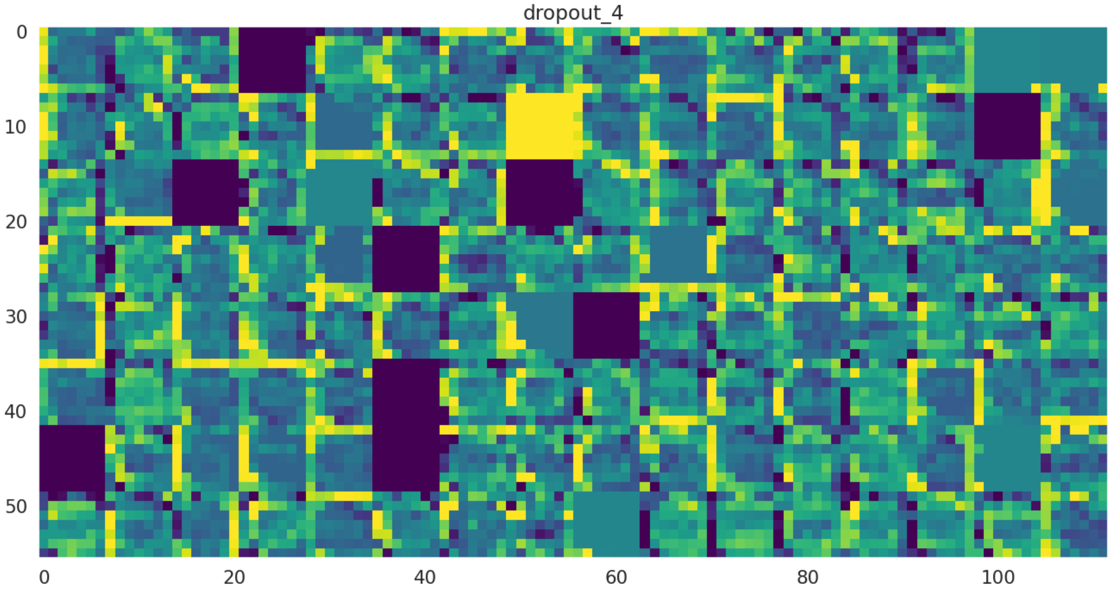
dropout_3

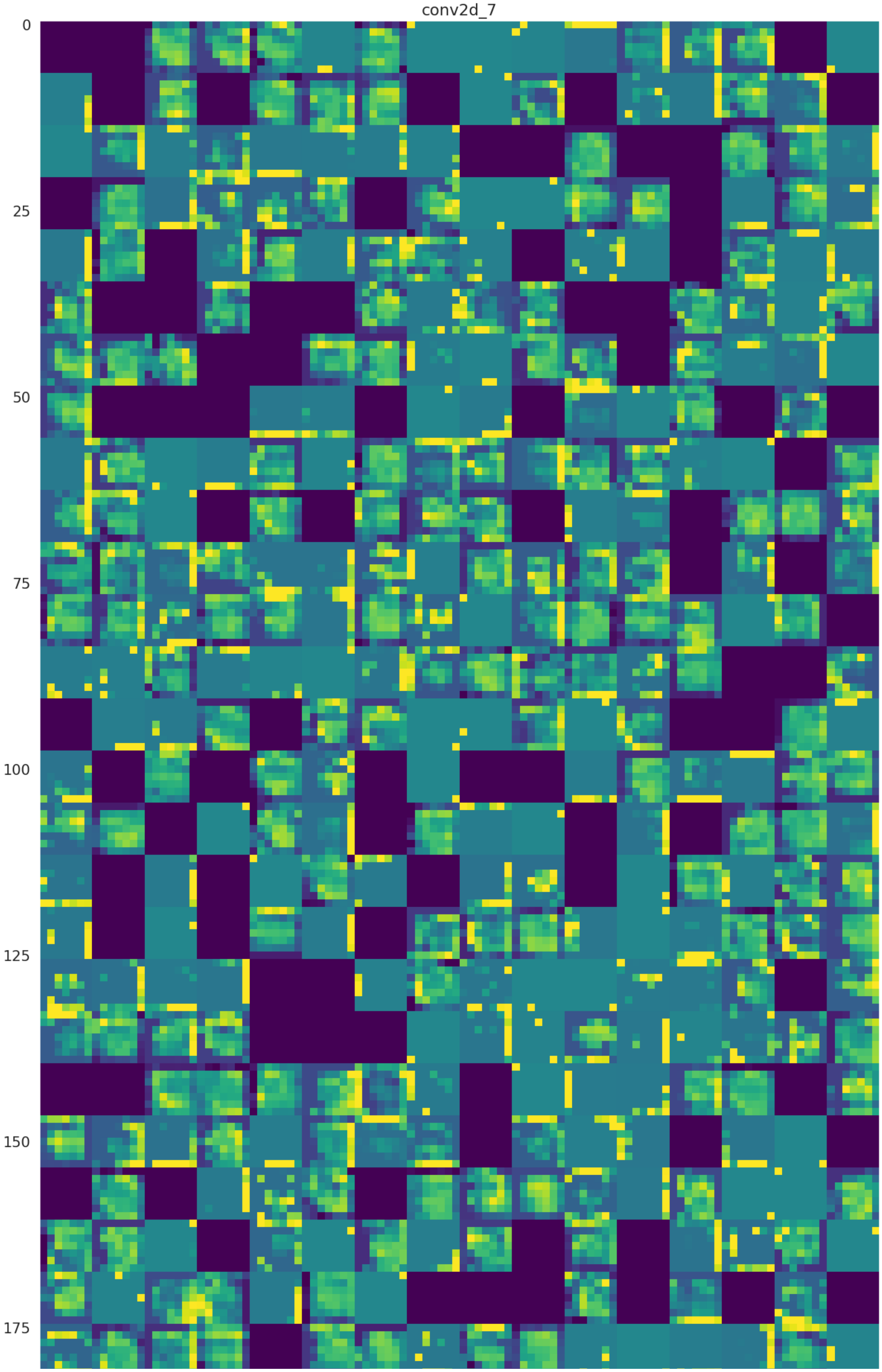


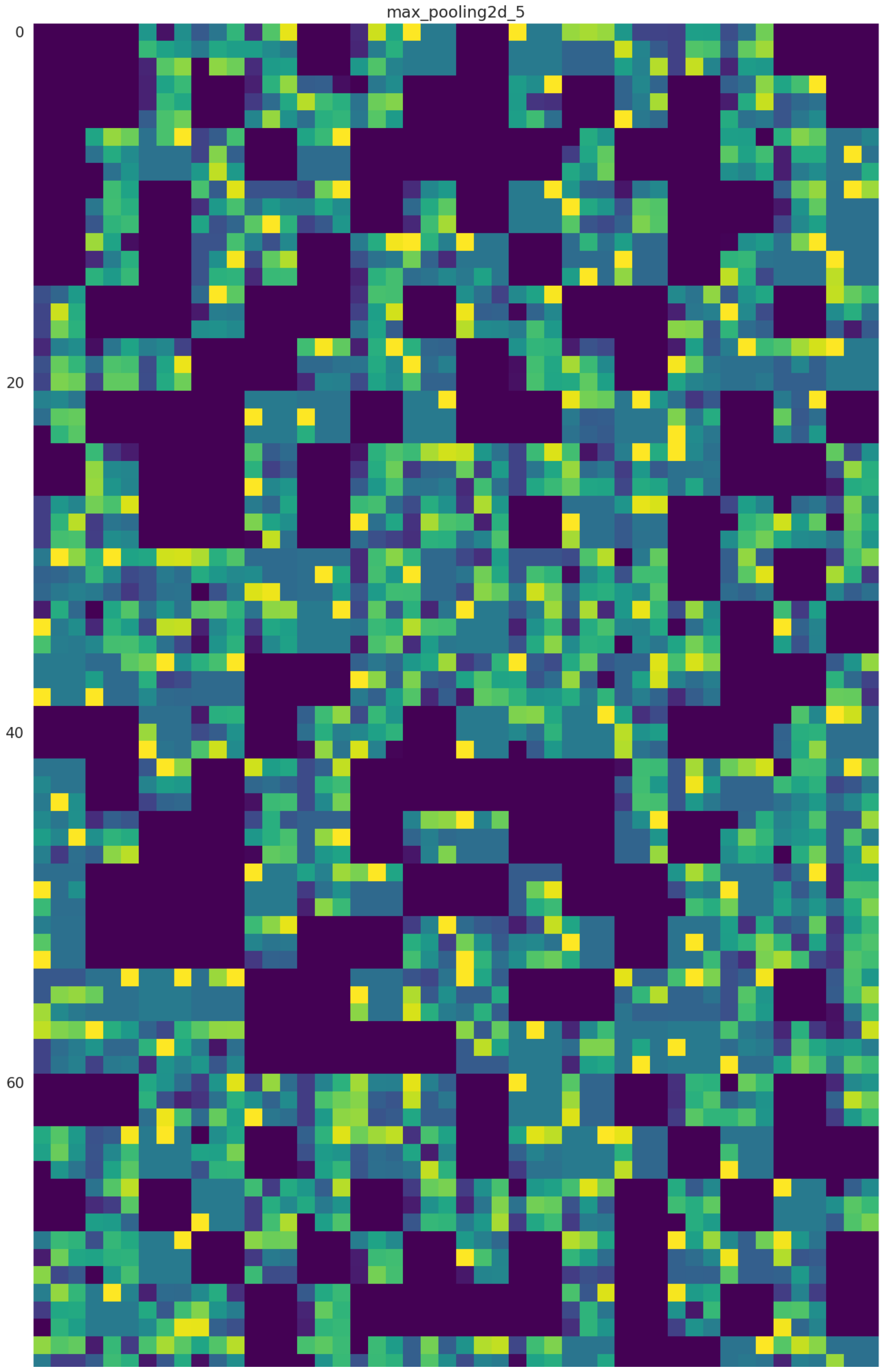
conv2d_6

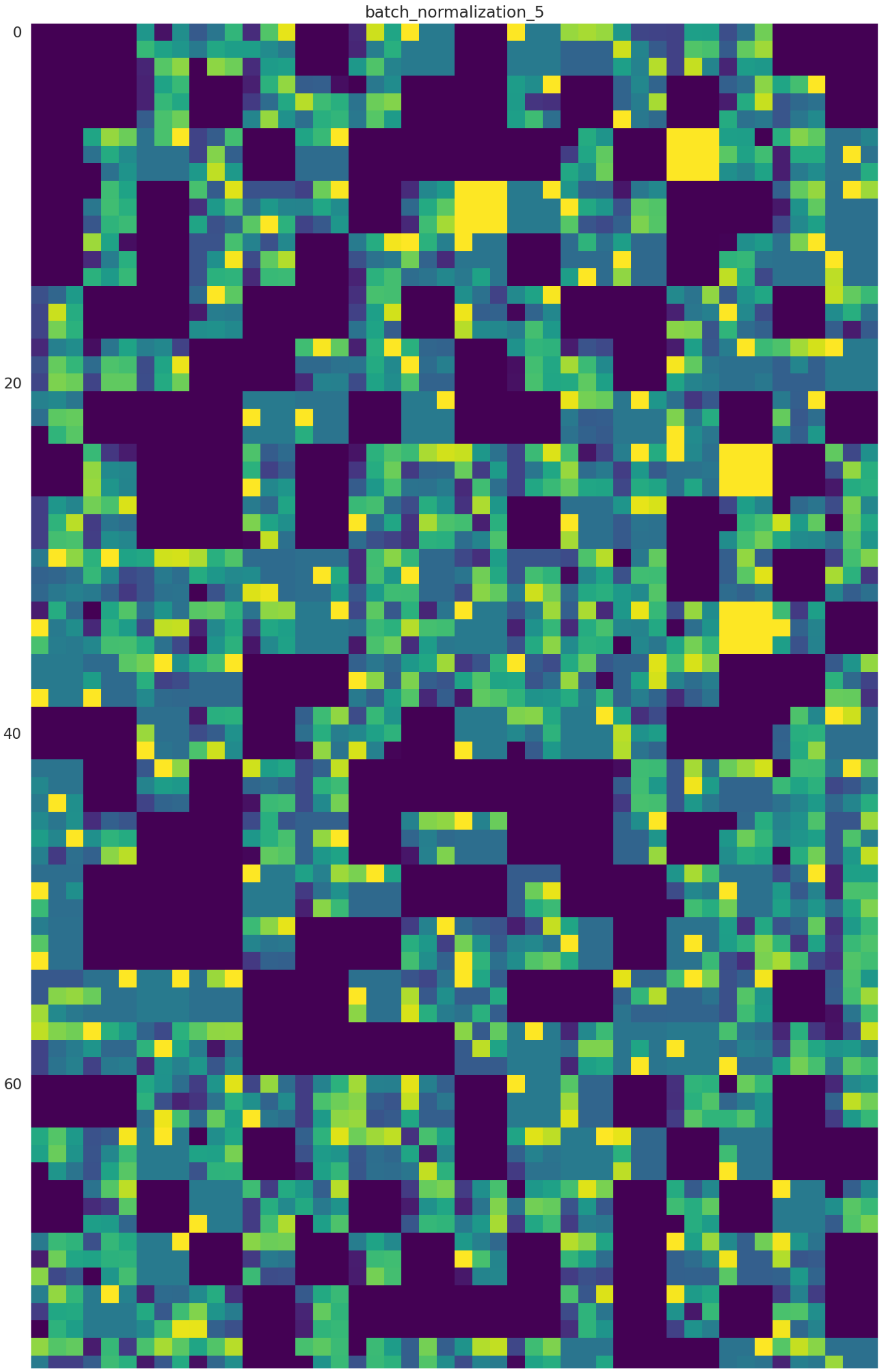

max_pooling2d_4
0
10
20
30
40
50
0
20
40
60
80
100

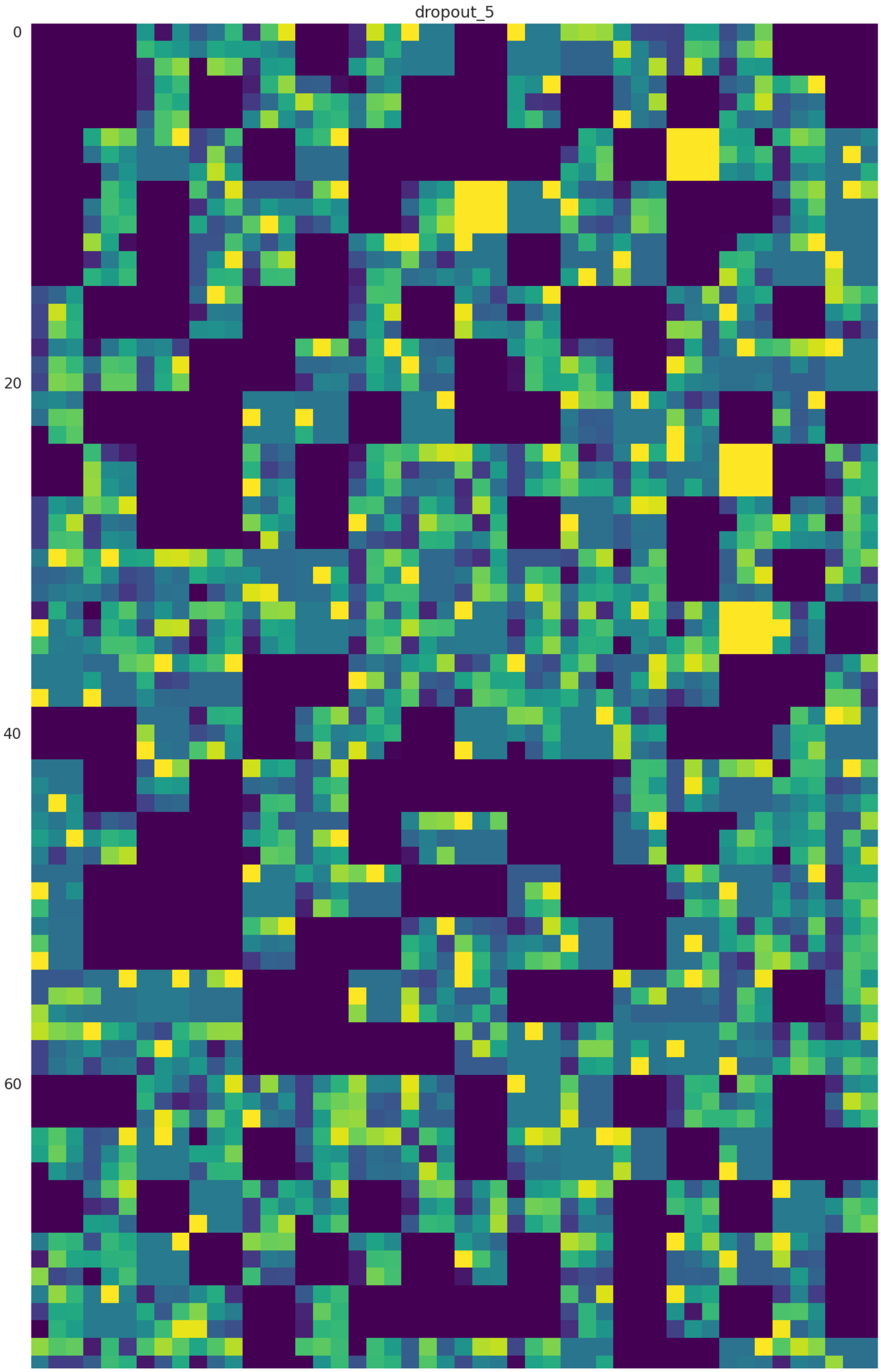

batch_normalization_4
0
10
20
30
40
50
0
20
40
60
80
100


dropout_4
0
10
20
30
40
50
0
20
40
60
80
100


conv2d_7
0
25
50
75
100
125
150
175


max_pooling2d_5
0
20
40
60

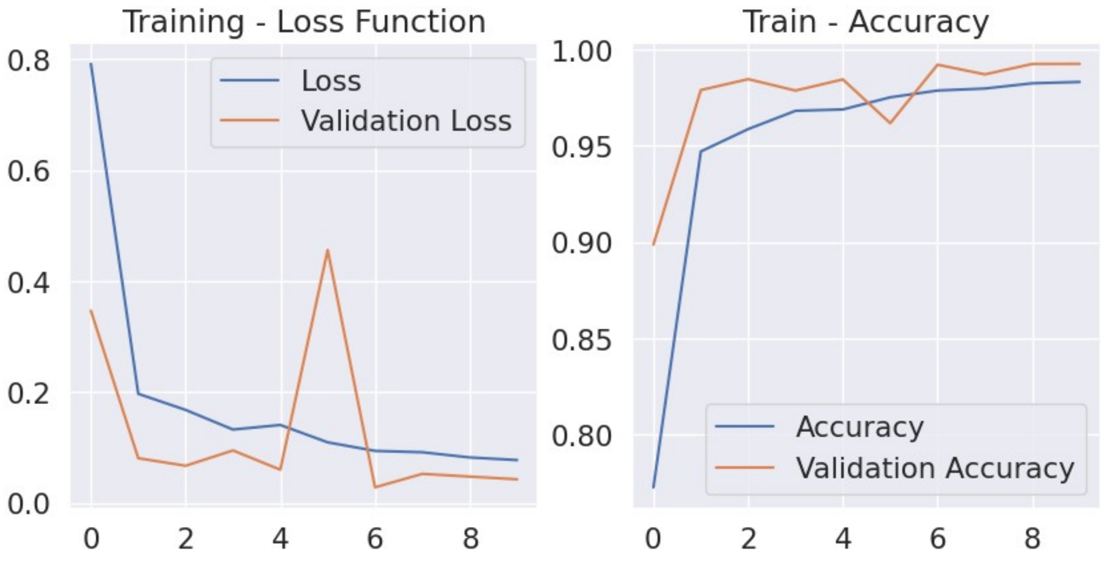

batch_normalization_5
0
20
40
60


dropout_5
0
20
40
60

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call
last)
/tmp/ipykernel_24/3907228436.py in <module>
     28     for col in range(n_cols):
     29         for row in range(images_per_row):
---> 30             channel_image = layer_activation[0,:, :, col *
images_per_row + row]
     31             channel_image -= channel_image.mean()
     32             channel_image /= channel_image.std()

IndexError: too many indices for array: array is 2-dimensional, but 4
were indexed
```

```
#Plot Training Model
plt.figure(figsize=(10, 10))

plt.subplot(2, 2, 1)
plt.plot(history.history['loss'], label='Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title('Training - Loss Function')

plt.subplot(2, 2, 2)
plt.plot(history.history['accuracy'], label='Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.title('Train - Accuracy')
```

```
Text(0.5, 1.0, 'Train - Accuracy')
```



```
from sklearn.metrics import accuracy_score

test = pd.read_csv("../input/gtsrb-german-traffic-sign/Test.csv")
test_labels = test['ClassId'].values
test_img_path = "../input/gtsrb-german-traffic-sign"
test_imgs = test['Path'].values

test_data = []
test_labels = []

for img in test_imgs:
    im = Image.open(test_img_path + '/' + img)
    im = im.resize((30,30))
    im = np.array(im)
    test_data.append(im)

test_data = np.array(test_data)
print(test_data.shape)

import warnings
warnings.filterwarnings("ignore")
test_labels = test['ClassId'].values
test_labels
```

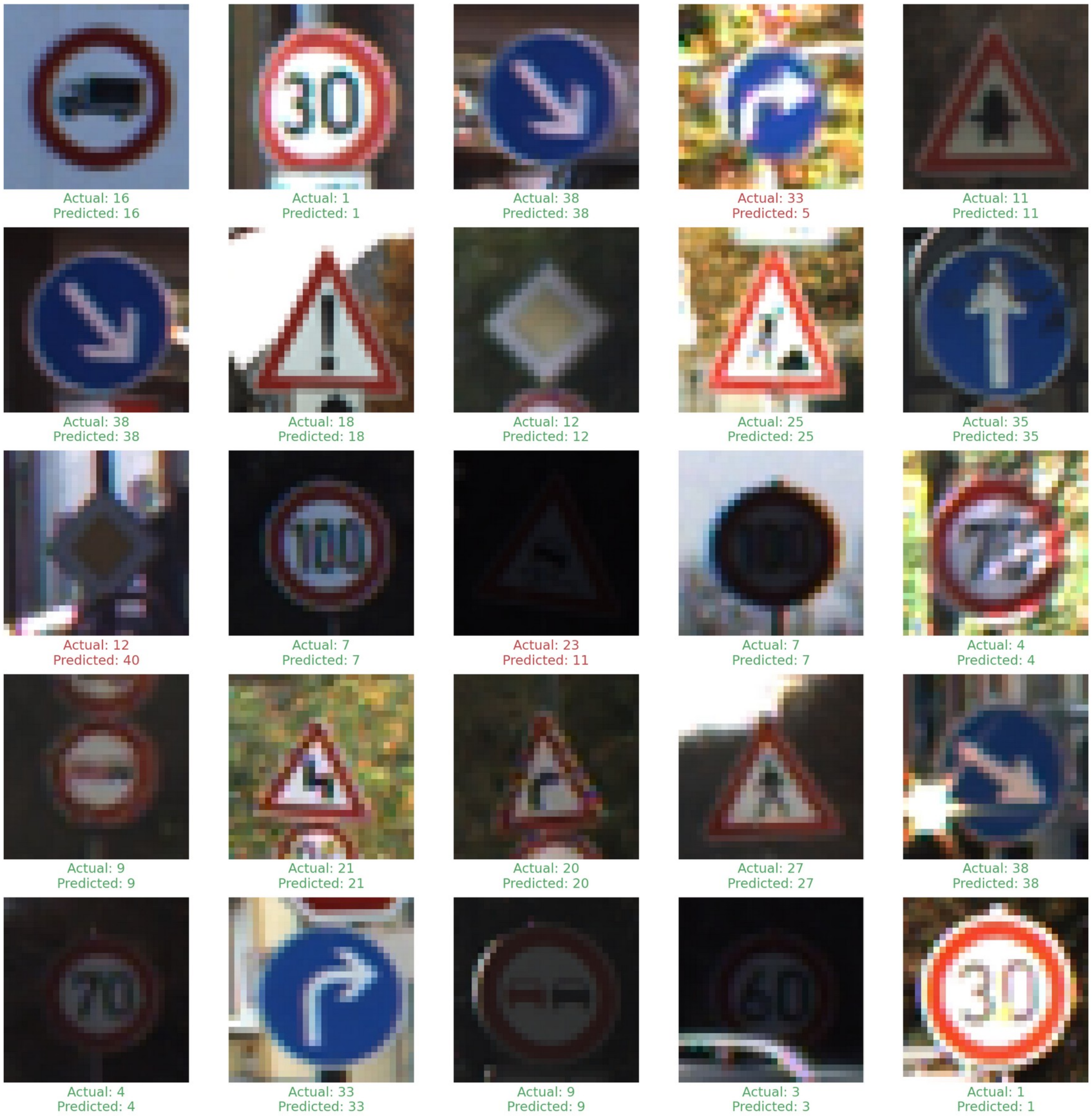
```
(12630, 30, 30, 3)
```

```
array([16,  1, 38, ...,  6,  7, 10])
```

```
#predictions = model.predict_classes(test_data)
pred = np.argmax(model.predict(test_data),axis=1)
print("accuracy: ", accuracy_score(test_labels, pred))
```

```
395/395 [==============================] - 1s 2ms/step
accuracy:  0.8697545526524149
```

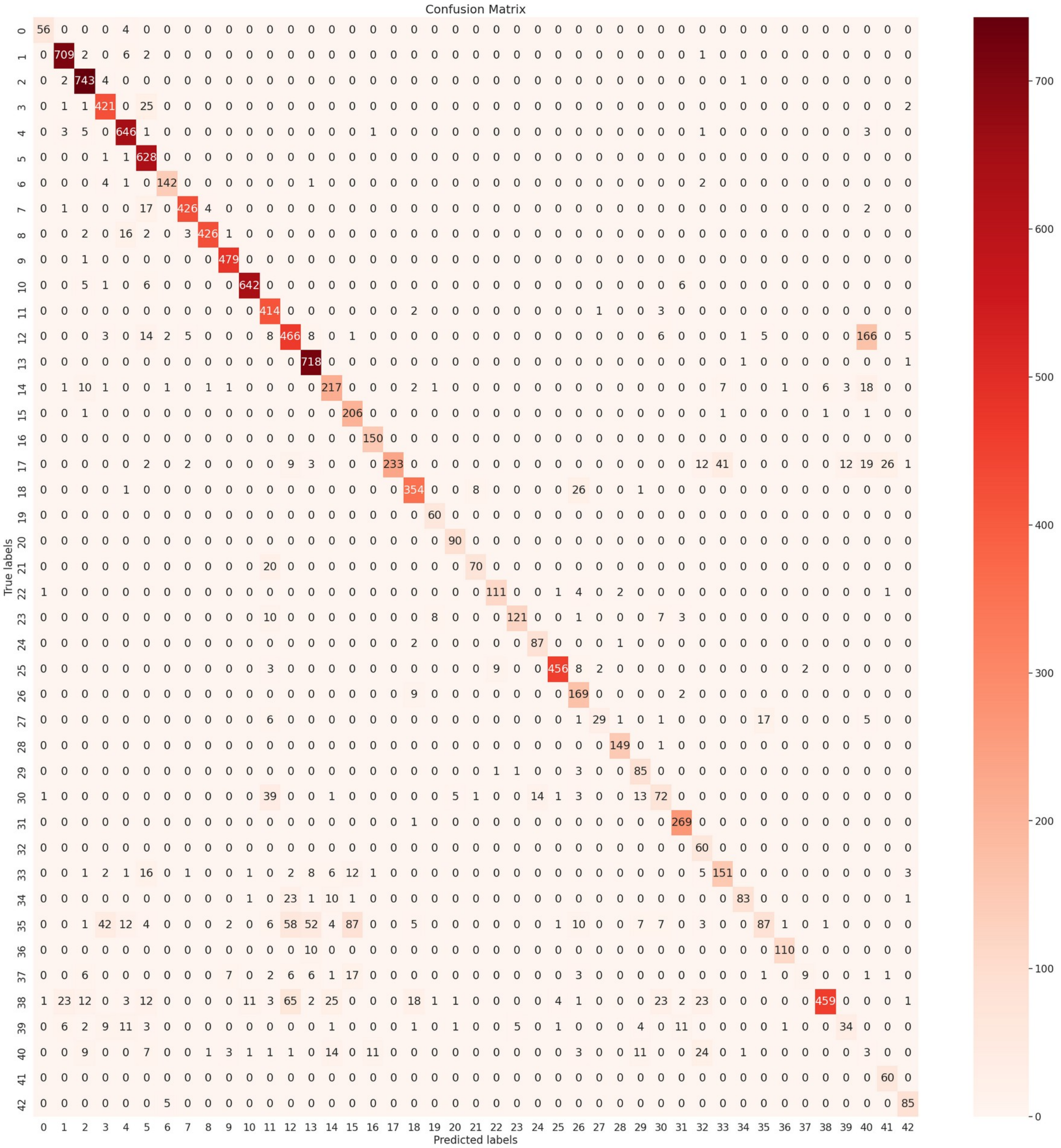
```
plt.figure(figsize=(25, 25))
for i in range(25):
    plt.subplot(5, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(test_data[i])
    if pred[i] == test_labels[i]:
        color = 'g'
    else:
        color = 'r'
    plt.xlabel("Actual: {}\nPredicted: {}".format(test_labels[i],
pred[i]), color=color)
plt.show()
```



```python
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Generate confusion matrix
cm = confusion_matrix(test_labels, pred)

# Set figure size and font size
fig, ax = plt.subplots(figsize=(30, 30))
sns.set(font_scale=1.4)

# Plot confusion matrix
sns.heatmap(cm, annot=True, fmt='g', cmap='Reds', ax=ax)

# Set axis labels and title
```

```
ax.set_xlabel('Predicted labels', fontsize=16)
ax.set_ylabel('True labels', fontsize=16)
ax.set_title('Confusion Matrix', fontsize=18)
```

```
Text(0.5, 1.0, 'Confusion Matrix')
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	56	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	709	2	0	6	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
2	0	2	743	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
3	0	1	1	421	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
4	0	3	5	0	646	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	3	0	0
5	0	0	0	1	1	628	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	4	1	0	142	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0
7	0	1	0	0	0	17	0	426	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0
8	0	0	2	0	16	2	0	3	426	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	1	0	0	0	0	0	0	479	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	5	1	0	6	0	0	0	0	642	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	414	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	1	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	3	0	14	2	5	0	0	0	8	466	8	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	1	5	0	0	0	0	166	0	5
13	0	0	0	0	0	0	0	0	0	0	0	0	0	718	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
14	0	1	10	1	0	0	1	0	1	1	0	0	0	0	217	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	1	0	6	3	18	0	0
15	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	206	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	150	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	2	0	2	0	0	0	0	9	3	0	0	0	233	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	41	0	0	0	0	0	12	19	26	1
18	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	354	0	0	8	0	0	0	0	26	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	90	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	70	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	111	0	0	1	4	0	2	0	0	0	0	0	0	0	0	0	0	0	0	1	0
23	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	8	0	0	0	121	0	0	1	0	0	0	7	3	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	87	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	9	0	0	456	8	2	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	169	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	29	1	0	1	0	0	0	0	17	0	0	0	0	5	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	149	0	1	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	3	0	0	85	0	0	0	0	0	0	0	0	0	0	0	0	0
30	1	0	0	0	0	0	0	0	0	0	0	39	0	0	1	0	0	0	0	0	5	1	0	0	14	1	3	0	0	13	72	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	269	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0
33	0	0	1	2	1	16	0	1	0	0	1	0	2	8	6	12	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	151	0	0	0	0	0	0	0	0	3
34	0	0	0	0	0	0	0	0	0	0	1	0	23	1	10	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	83	0	0	0	0	0	0	0	1
35	0	0	1	42	12	4	0	0	0	2	0	6	58	52	4	87	0	0	5	0	0	0	0	0	0	1	10	0	0	7	7	0	3	0	0	87	1	0	1	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	110	0	0	0	0	0	0
37	0	0	6	0	0	0	0	0	0	7	0	2	6	6	1	17	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	1	0	9	0	0	1	1	0
38	1	23	12	0	3	12	0	0	0	0	11	3	65	2	25	0	0	0	18	1	1	0	0	0	0	4	1	0	0	0	23	2	23	0	0	0	0	0	459	0	0	0	1
39	0	6	2	9	11	3	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	5	0	1	0	0	0	4	0	11	0	0	0	0	1	0	0	34	0	0	0
40	0	0	9	0	0	7	0	0	1	3	1	1	1	0	14	0	11	0	0	0	0	0	0	0	0	0	3	0	0	11	0	0	24	0	1	0	0	0	0	0	3	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0
42	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	85