```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import os
from tensorflow.keras.models import Model
import matplotlib.pyplot as plt

# --- 1. Dataset Preparation ---

# Set the directories for training and testing datasets
train_dir = 'dataset/training_set'
test_dir = 'dataset/test_set'

# Set image dimensions, batch size, and number of epochs
img_width, img_height = 128, 128
batch_size = 128
epochs = 15

# Create an ImageDataGenerator for training data with augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,          # Normalize pixel values to [0, 1]
    shear_range=0.2,         # Randomly apply shearing transformations
    zoom_range=0.2,          # Randomly zoom into images
    horizontal_flip=True     # Randomly flip images horizontally
)

# Create an ImageDataGenerator for test data (only rescaling)
test_datagen = ImageDataGenerator(rescale=1./255)

# Load and preprocess training images from directory, apply
augmentations
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_width, img_height),  # Resize images to target
size
    batch_size=batch_size,                # Number of images per batch
    class_mode='binary'                   # Binary labels (cats vs
dogs)
)

# Load and preprocess test images from directory (no augmentation)
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(img_width, img_height),  # Resize images to target
size
    batch_size=batch_size,                # Number of images per batch
    class_mode='binary'                    # Binary labels (cats vs
dogs)
)
```

```
Found 8000 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
```

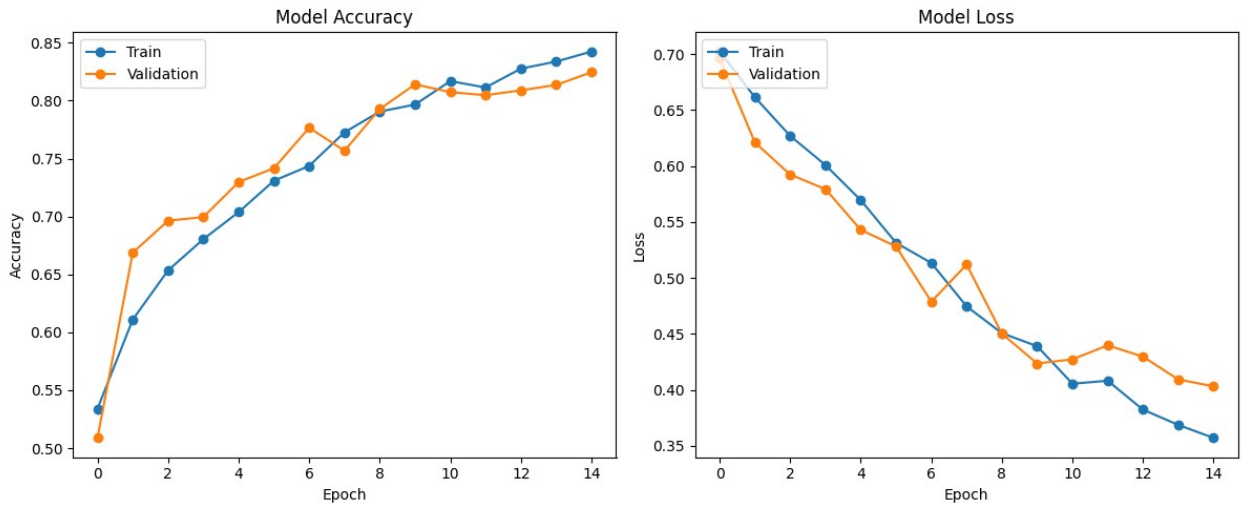
```
# --- 2. Build the CNN Model ---
# Define the CNN architecture in an easy-to-read structure
model = Sequential([
    # 1st Convolutional Layer
    Conv2D(32, (4, 4), activation='relu', input_shape=(img_width,
img_height, 3), name='conv2d'),
    # 1st Max Pooling Layer
    MaxPooling2D((2, 2)),
    # 2nd Convolutional Layer
    Conv2D(64, (4, 4), activation='relu', name='conv2d_1'),
    # 2nd Max Pooling Layer
    MaxPooling2D((2, 2)),
    # 3rd Convolutional Layer
    Conv2D(128, (4, 4), activation='relu', name='conv2d_2'),
    # 3rd Max Pooling Layer
    MaxPooling2D((2, 2)),

    # Flatten Layer
    Flatten(),

    # 1st Dense Layer
    Dense(512, activation='relu'),
    # 1st Dropout Layer
    Dropout(0.5),
    # 2nd Dense Layer
    Dense(256, activation='relu'),
    # 2nd Dropout Layer
    Dropout(0.3),

    # Output Layer
    Dense(1, activation='sigmoid')
])

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

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

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 125, 125, 32)      1568      
                                                                 
 max_pooling2d_30 (MaxPoolin  (None, 62, 62, 32)       0         
 g2D)                                                            
                                                                 
 conv2d_1 (Conv2D)           (None, 59, 59, 64)        32832     
                                                                 
 max_pooling2d_31 (MaxPoolin  (None, 29, 29, 64)       0         
 g2D)                                                            
                                                                 
 conv2d_2 (Conv2D)           (None, 26, 26, 128)       131200    
                                                                 
 max_pooling2d_32 (MaxPoolin  (None, 13, 13, 128)      0         
 g2D)                                                            
                                                                 
 flatten_10 (Flatten)        (None, 21632)             0         
                                                                 
 dense_29 (Dense)            (None, 512)               11076096  
                                                                 
 dropout_18 (Dropout)        (None, 512)               0         
                                                                 
 dense_30 (Dense)            (None, 256)               131328    
                                                                 
 dropout_19 (Dropout)        (None, 256)               0         
                                                                 
 dense_31 (Dense)            (None, 1)                 257       
                                                                 
=================================================================
Total params: 11,373,281
Trainable params: 11,373,281
Non-trainable params: 0
_________________________________________________________________

# --- 3. Train the Model ---
if os.path.exists(os.path.join(train_dir, 'dogs')) and \
os.path.exists(os.path.join(train_dir, 'cats')):
    try:
        history = model.fit(
            train_generator,
            steps_per_epoch=train_generator.samples // batch_size,
            epochs=epochs,
            validation_data=test_generator,
            validation_steps=test_generator.samples // batch_size
        )
    except Exception as e:
        print(f"Error during training: {e}")
        history = None
else:
    print("Skipping model training because dataset directories are
empty or missing real images.")
    history = None
```

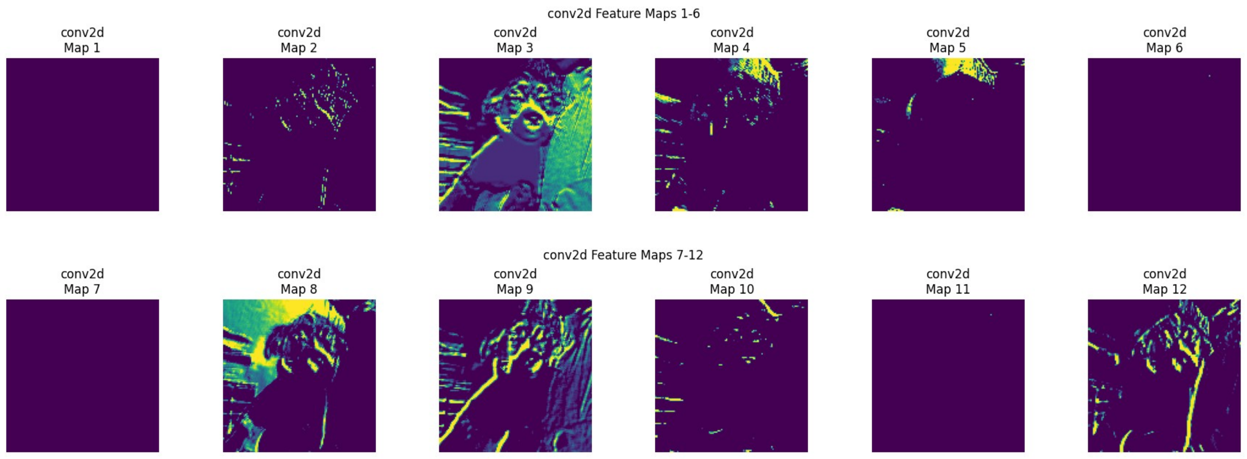
```
Epoch 1/15
62/62 [==============================] - 142s 2s/step - loss: 0.7023 -
accuracy: 0.5340 - val_loss: 0.6962 - val_accuracy: 0.5089
Epoch 2/15
62/62 [==============================] - 29s 459ms/step - loss: 0.6614
- accuracy: 0.6110 - val_loss: 0.6210 - val_accuracy: 0.6687
Epoch 3/15
62/62 [==============================] - 29s 463ms/step - loss: 0.6270
- accuracy: 0.6535 - val_loss: 0.5924 - val_accuracy: 0.6964
Epoch 4/15
62/62 [==============================] - 29s 464ms/step - loss: 0.6009
- accuracy: 0.6804 - val_loss: 0.5794 - val_accuracy: 0.6995
Epoch 5/15
62/62 [==============================] - 29s 458ms/step - loss: 0.5695
- accuracy: 0.7038 - val_loss: 0.5430 - val_accuracy: 0.7297
Epoch 6/15
62/62 [==============================] - 29s 460ms/step - loss: 0.5314
- accuracy: 0.7308 - val_loss: 0.5282 - val_accuracy: 0.7417
Epoch 7/15
62/62 [==============================] - 29s 459ms/step - loss: 0.5134
- accuracy: 0.7436 - val_loss: 0.4788 - val_accuracy: 0.7766
Epoch 8/15
62/62 [==============================] - 29s 460ms/step - loss: 0.4747
- accuracy: 0.7726 - val_loss: 0.5120 - val_accuracy: 0.7568
Epoch 9/15
62/62 [==============================] - 29s 461ms/step - loss: 0.4507
- accuracy: 0.7905 - val_loss: 0.4503 - val_accuracy: 0.7927
Epoch 10/15
62/62 [==============================] - 29s 461ms/step - loss: 0.4391
- accuracy: 0.7966 - val_loss: 0.4236 - val_accuracy: 0.8141
Epoch 11/15
62/62 [==============================] - 29s 460ms/step - loss: 0.4056
- accuracy: 0.8168 - val_loss: 0.4273 - val_accuracy: 0.8073
Epoch 12/15
62/62 [==============================] - 29s 459ms/step - loss: 0.4082
- accuracy: 0.8114 - val_loss: 0.4398 - val_accuracy: 0.8047
Epoch 13/15
62/62 [==============================] - 29s 463ms/step - loss: 0.3823
- accuracy: 0.8276 - val_loss: 0.4297 - val_accuracy: 0.8089
Epoch 14/15
62/62 [==============================] - 29s 460ms/step - loss: 0.3687
- accuracy: 0.8337 - val_loss: 0.4094 - val_accuracy: 0.8135
Epoch 15/15
```

```
62/62 [==============================] - 29s 460ms/step - loss: 0.3571
- accuracy: 0.8424 - val_loss: 0.4032 - val_accuracy: 0.8245

# --- 4. Improved Helper Functions for Visualization ---

def plot_training_history(history):
    if history is None:
        print("No training history to plot.")
        return
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], marker='o')
    plt.plot(history.history['val_accuracy'], marker='o')
    plt.title('Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], marker='o')
    plt.plot(history.history['val_loss'], marker='o')
    plt.title('Model Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')
    plt.tight_layout()
    plt.show()

def visualize_feature_maps(model, img_paths, layer_names=None,
group_size=6):
    """
    Visualize all feature maps for one or more images at selected
layers, grouped for clarity.
    Args:
        model: Trained Keras model.
        img_paths: List of image paths (or a single path as string).
        layer_names: List of layer names to visualize. If None, all
Conv2D layers are used.
        group_size: Number of feature maps to show per figure
(default: 4).
    """
    if isinstance(img_paths, str):
        img_paths = [img_paths]
    if layer_names is None:
        layer_names = [layer.name for layer in model.layers if
isinstance(layer, Conv2D)]
        if not layer_names:
            print("No Conv2D layers found in the model.")
            return
    outputs = [model.get_layer(name).output for name in layer_names]
    activation_model = Model(inputs=model.input, outputs=outputs)
    for img_path in img_paths:
        if not os.path.exists(img_path):
            print(f"Image not found at {img_path}.")
            continue
        img = tf.keras.preprocessing.image.load_img(img_path,
target_size=(img_width, img_height))
        img_tensor = tf.keras.preprocessing.image.img_to_array(img)
        img_tensor = np.expand_dims(img_tensor, axis=0)
        img_tensor /= 255.
        try:
            activations = activation_model.predict(img_tensor)
        except Exception as e:
            print(f"Error predicting activations: {e}")
            continue
        print(f"\nFeature maps for image:
{os.path.basename(img_path)}")
        plt.figure(figsize=(2,2))

plt.imshow(tf.keras.preprocessing.image.array_to_img(img_tensor[0]))
        plt.title("Original Image")
        plt.axis('off')
        plt.show()
        for layer_name, layer_activation in zip(layer_names,
activations):
            if len(layer_activation.shape) == 4:
                n_features = layer_activation.shape[-1]
                size = layer_activation.shape[1]
                n_groups = (n_features + group_size - 1) // group_size
                for g in range(n_groups):
                    start = g * group_size
                    end = min(start + group_size, n_features)
                    fig, axes = plt.subplots(1, end - start,
figsize=(3 * (end - start), 3))
                    if end - start == 1:
                        axes = [axes]
                    for idx, col in enumerate(range(start, end)):
                        channel_image = layer_activation[0, :, :, col]
                        channel_image -= channel_image.mean()
                        channel_image /= (channel_image.std() + 1e-5)
                        channel_image *= 64
                        channel_image += 128
                        channel_image = np.clip(channel_image, 0,
255).astype('uint8')
                        axes[idx].imshow(channel_image,
cmap='viridis')
                        axes[idx].set_title(f"{layer_name}\nMap
{col+1}")
                        axes[idx].axis('off')
                    plt.suptitle(f"{layer_name} Feature Maps
{start+1}-{end}")
                    plt.tight_layout()
                    plt.show()
            else:
                print(f"Skipping {layer_name}, not a 2D feature map.")

def visualize_filters(model, layer_name, num_filters_to_show=8):
    """
    Visualize learned filters of a convolutional layer.
    """
    try:
        layer = model.get_layer(name=layer_name)
    except ValueError:
        print(f"Layer '{layer_name}' not found.")
        return
    if not isinstance(layer, Conv2D):
        print(f"Layer '{layer_name}' is not a Conv2D layer.")
        return
    filters, biases = layer.get_weights()
    f_min, f_max = filters.min(), filters.max()
    filters = (filters - f_min) / (f_max - f_min)
    n_filters = min(num_filters_to_show, filters.shape[3])
    n_channels = filters.shape[2]
    fig, axes = plt.subplots(n_channels, n_filters, figsize=(n_filters
* 2, n_channels * 2))
    fig.suptitle(f'Learned Filters for {layer_name}', fontsize=16)
    for i in range(n_filters):
        f = filters[:, :, :, i]
        for j in range(n_channels):
            ax = axes[j, i] if n_channels > 1 else axes[i]
            ax.imshow(f[:, :, j], cmap='gray')
            ax.set_xticks([])
            ax.set_yticks([])
            if j == 0:
                ax.set_title(f'Filter {i+1}')
            if i == 0:
                ax.set_ylabel(f'Channel {j+1}')
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()

# Example usage:
sample_images = [
    'dataset/test_set/dogs/dog.4010.jpg',
    'dataset/test_set/cats/cat.4010.jpg'
]
if history is not None:
    plot_training_history(history)
    visualize_feature_maps(model, sample_images,
layer_names=['conv2d', 'conv2d_1', 'conv2d_2'])
```

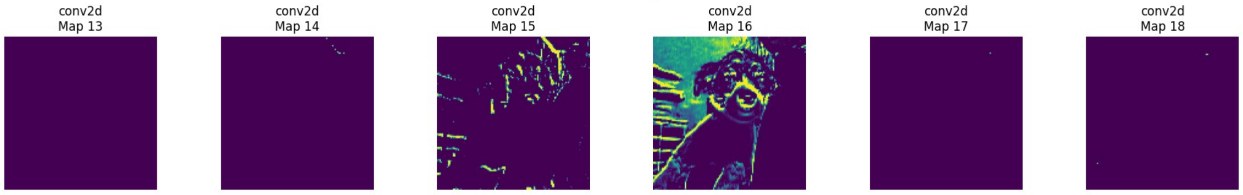
```
    visualize_filters(model, 'conv2d', num_filters_to_show=8)
    visualize_filters(model, 'conv2d_1', num_filters_to_show=8)
    visualize_filters(model, 'conv2d_2', num_filters_to_show=8)
```



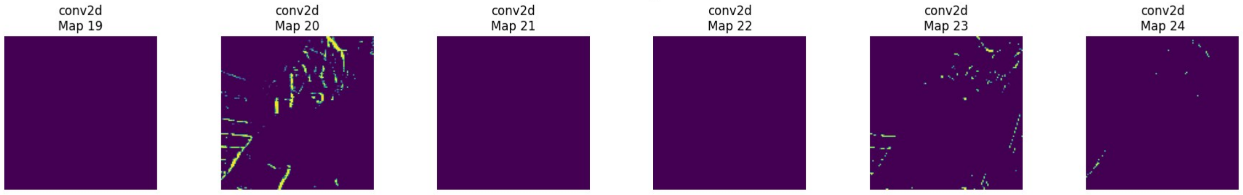
```
1/1 [==============================] - 0s 52ms/step

Feature maps for image: dog.4010.jpg
```



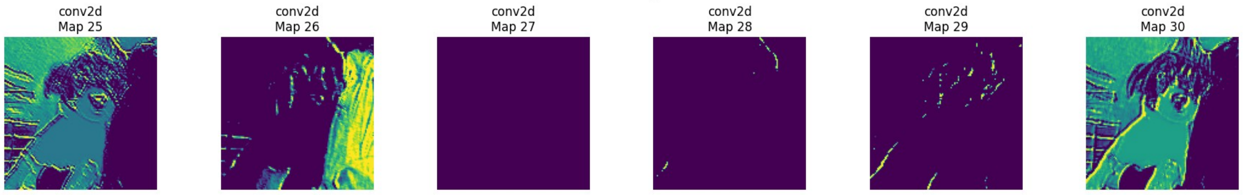
conv2d Feature Maps 13-18



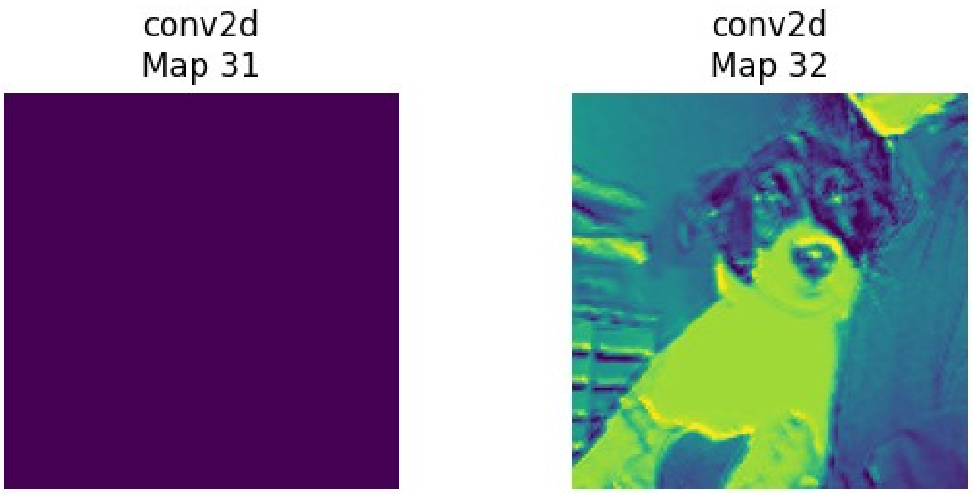
conv2d Feature Maps 19-24



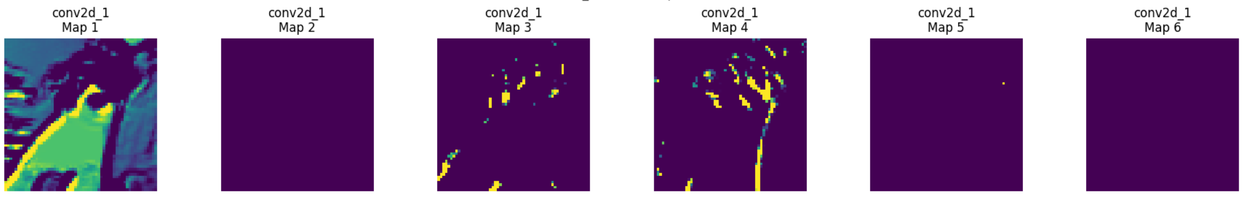
conv2d Feature Maps 25-30



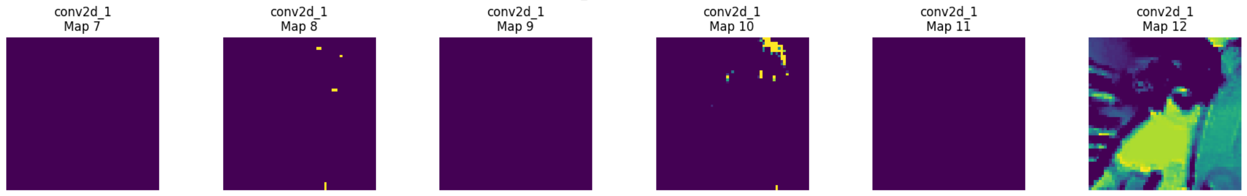
conv2d Feature Maps 31-32



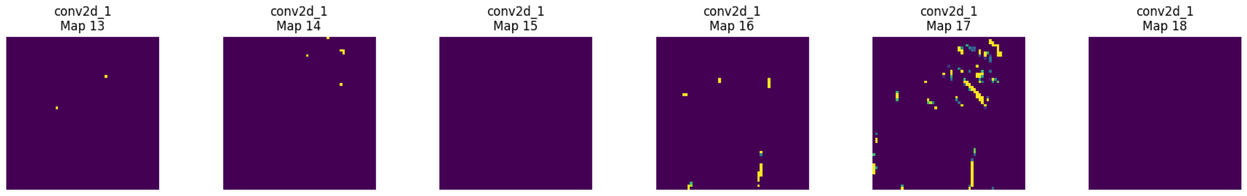
conv2d_1 Feature Maps 1-6



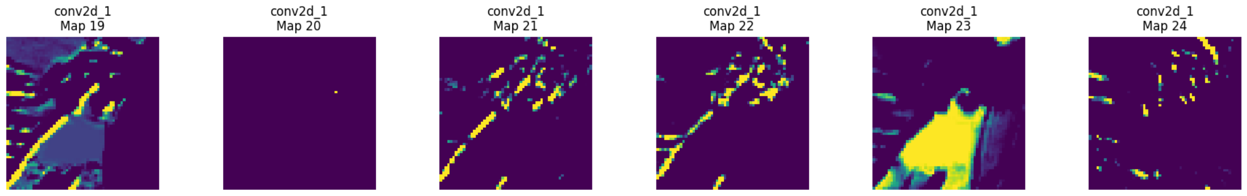
conv2d_1 Feature Maps 7-12



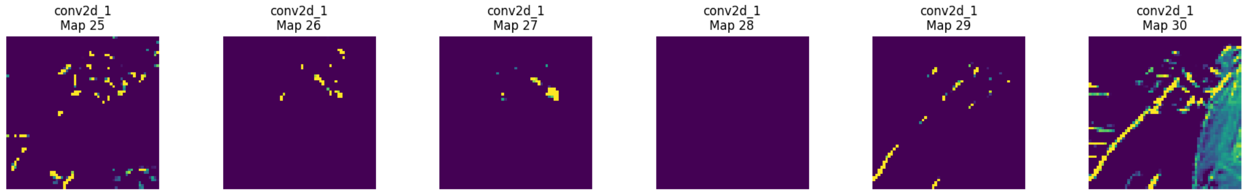
conv2d_1 Feature Maps 13-18



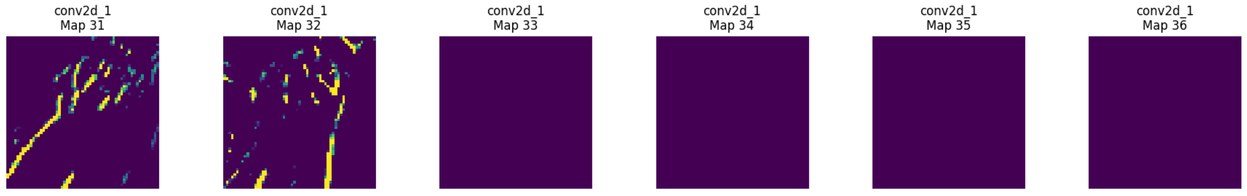
conv2d_1 Feature Maps 19-24



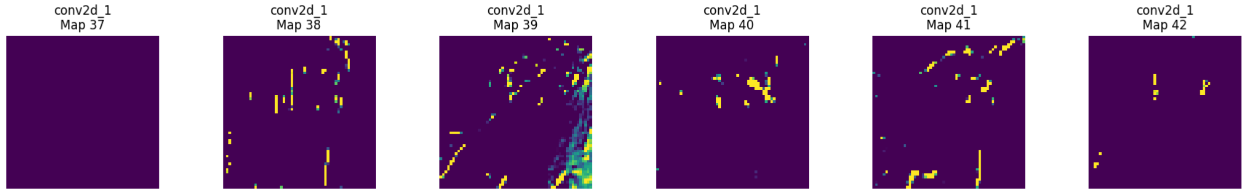
conv2d_1 Feature Maps 25-30



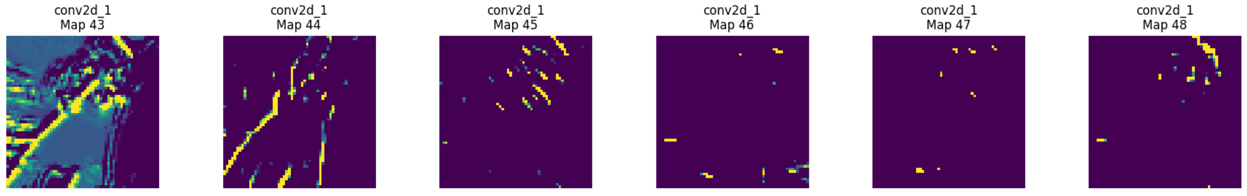
conv2d_1 Feature Maps 31-36



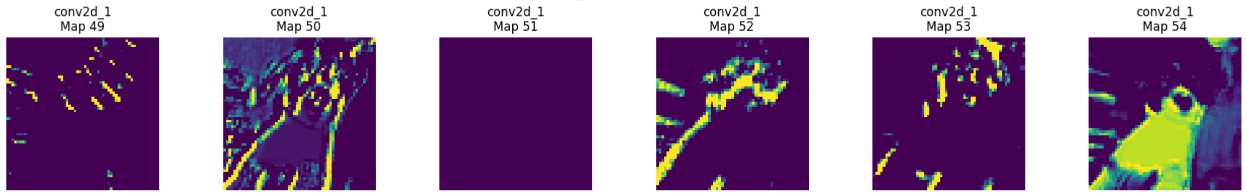
conv2d_1 Feature Maps 37-42



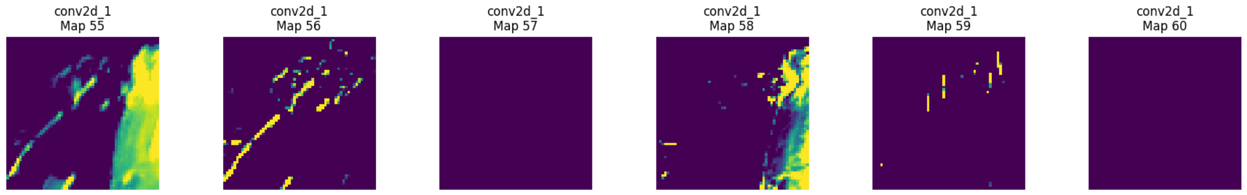
conv2d_1 Feature Maps 43-48



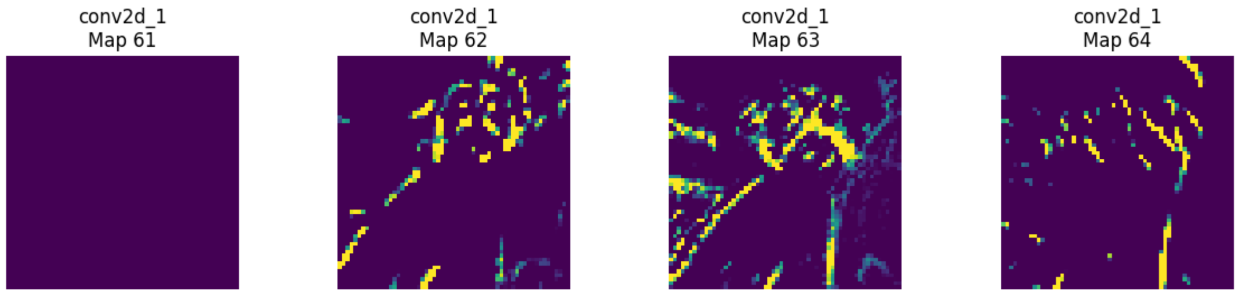
conv2d_1 Feature Maps 49-54



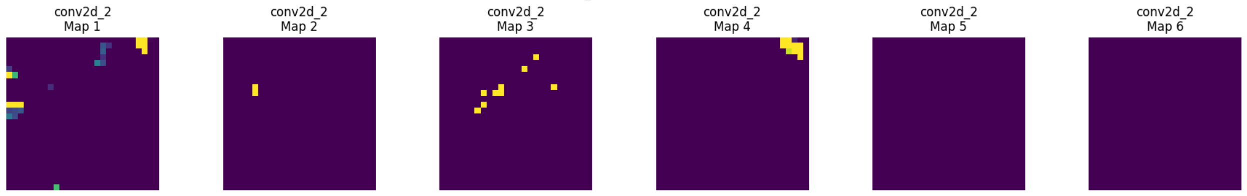
conv2d_1 Feature Maps 55-60



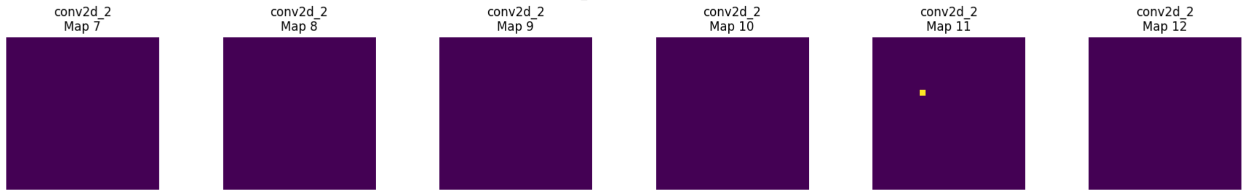
conv2d_1 Feature Maps 61-64



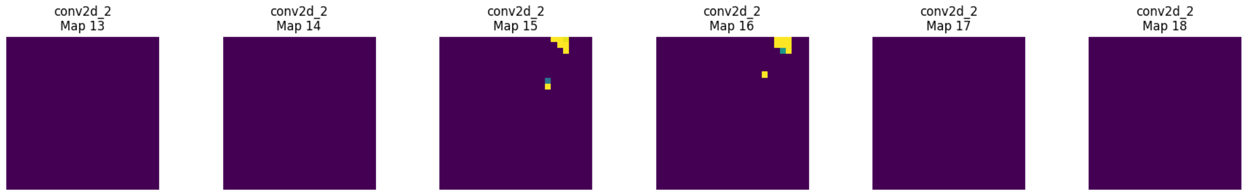
conv2d_2 Feature Maps 1-6



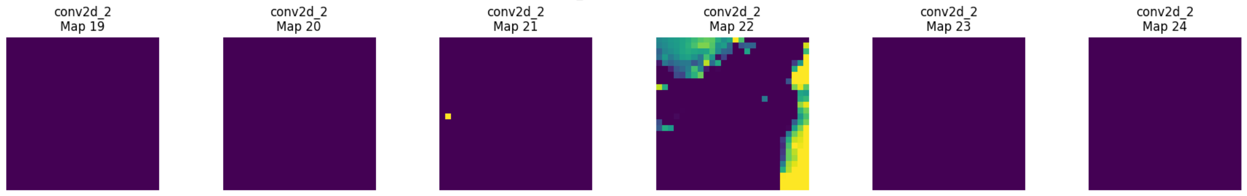
conv2d_2 Feature Maps 7-12



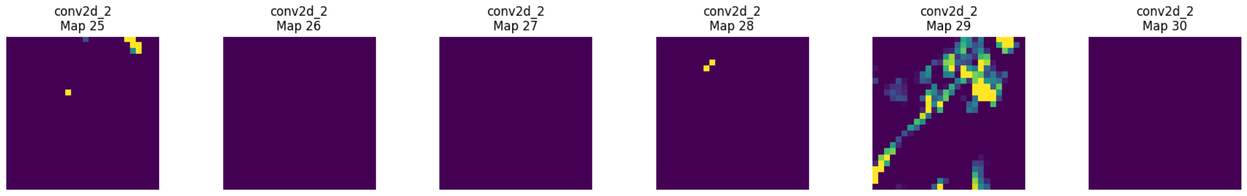
conv2d_2 Feature Maps 13-18



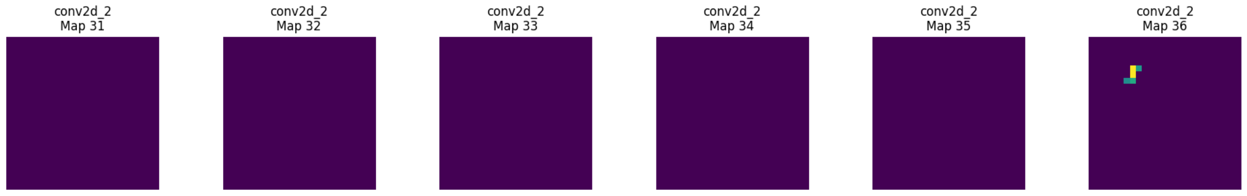
conv2d_2 Feature Maps 19-24



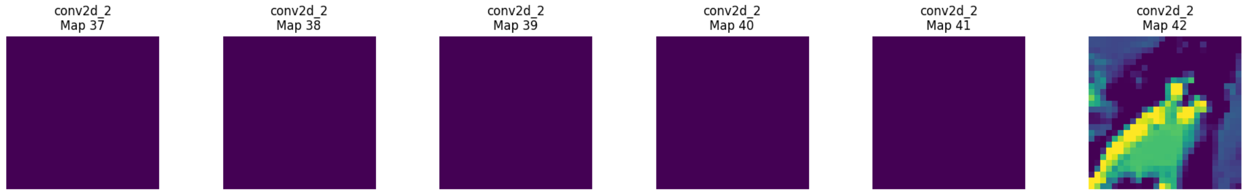
conv2d_2 Feature Maps 25-30



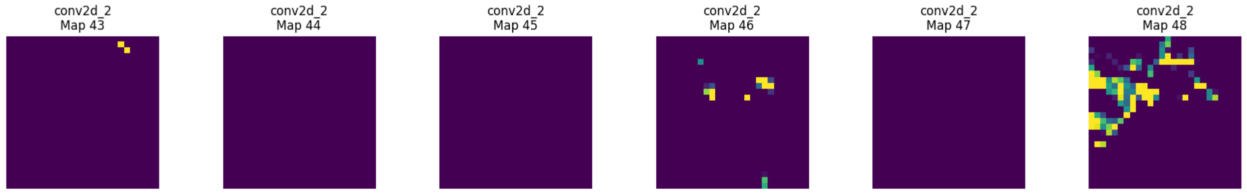
conv2d_2 Feature Maps 31-36



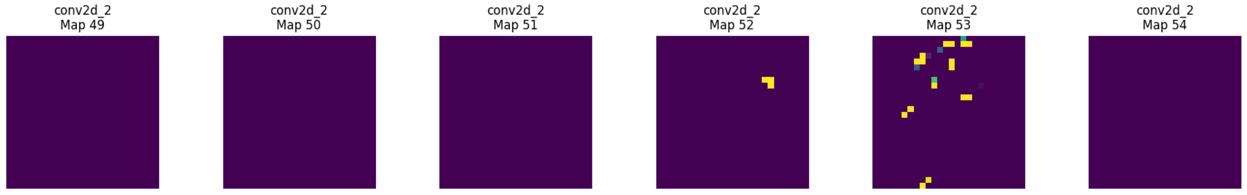
conv2d_2 Feature Maps 37-42



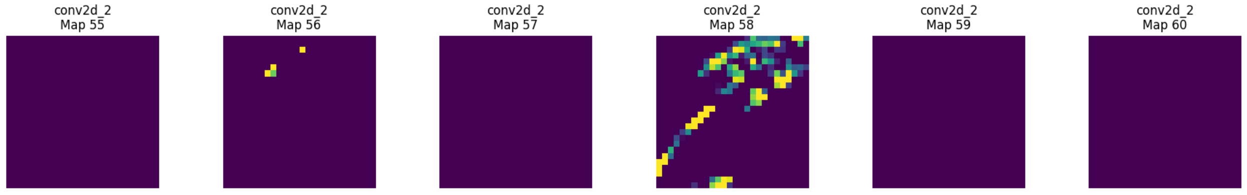
conv2d_2 Feature Maps 43-48



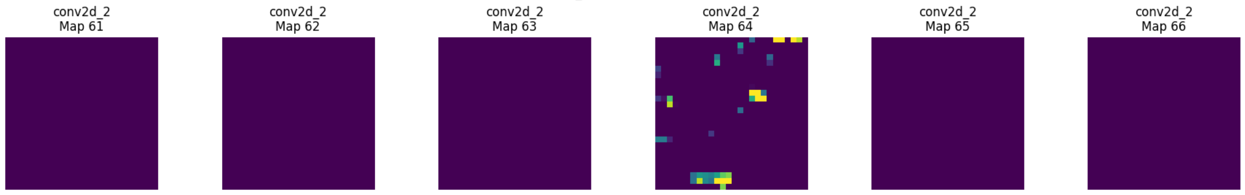
conv2d_2 Feature Maps 49-54



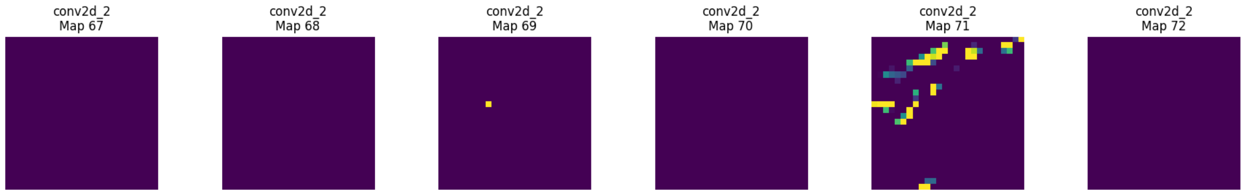
conv2d_2 Feature Maps 55-60



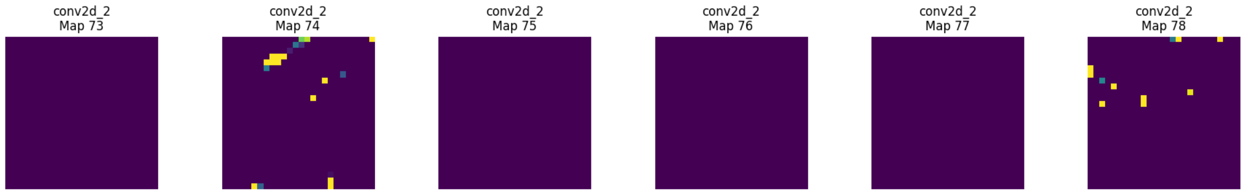
conv2d_2 Feature Maps 61-66



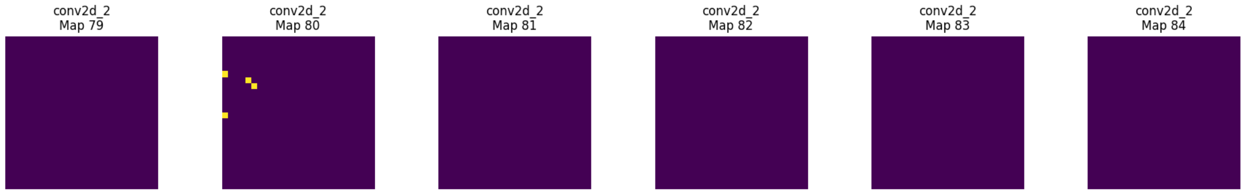
conv2d_2 Feature Maps 67-72



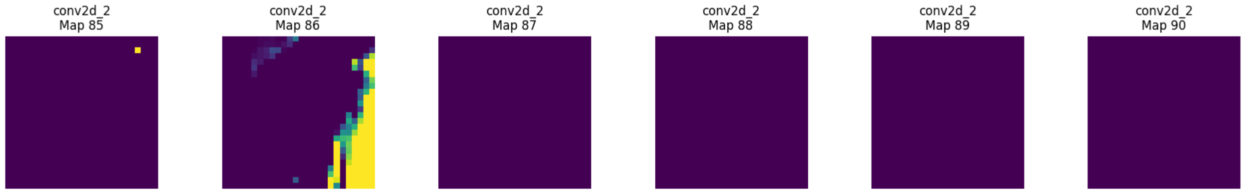
conv2d_2 Feature Maps 73-78



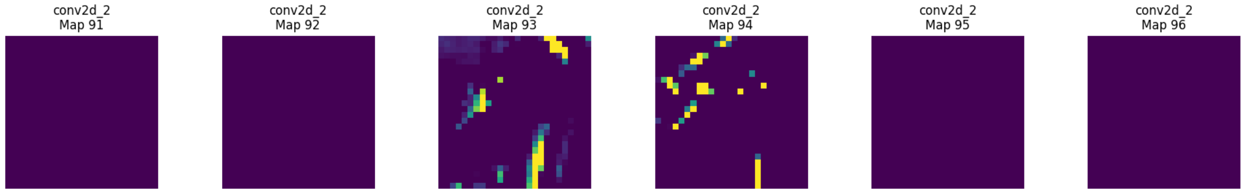
conv2d_2 Feature Maps 79-84



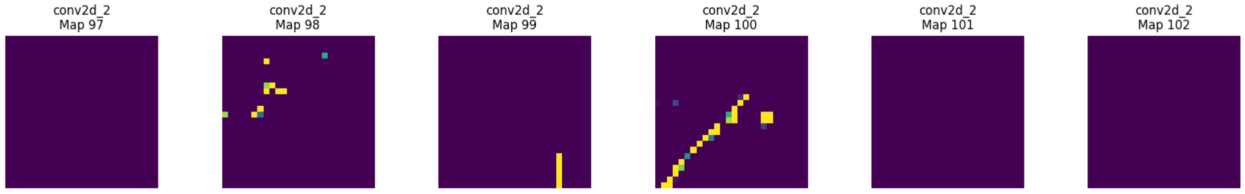
conv2d_2 Feature Maps 85-90

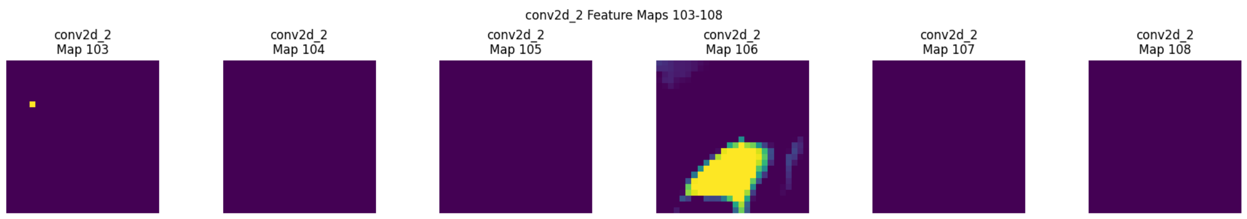


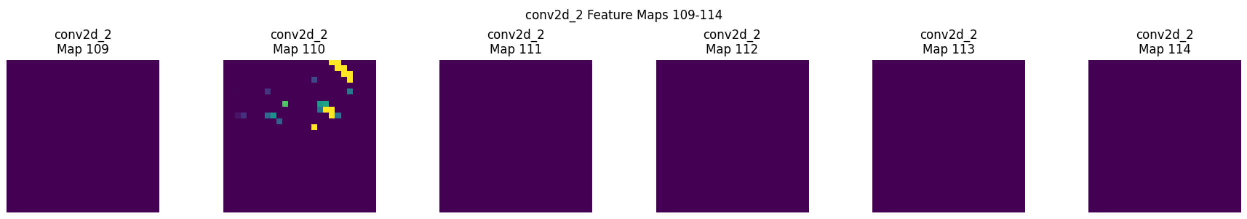
conv2d_2 Feature Maps 91-96

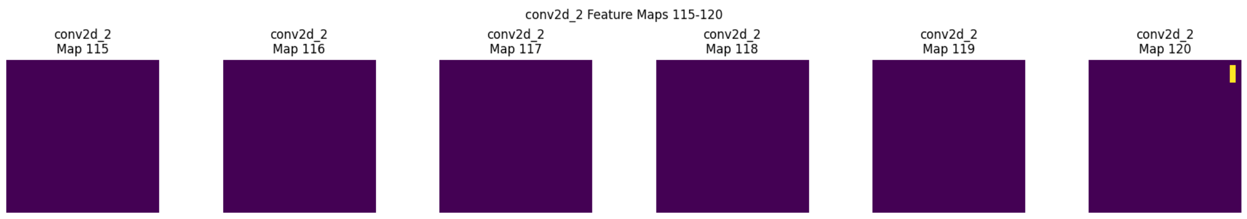


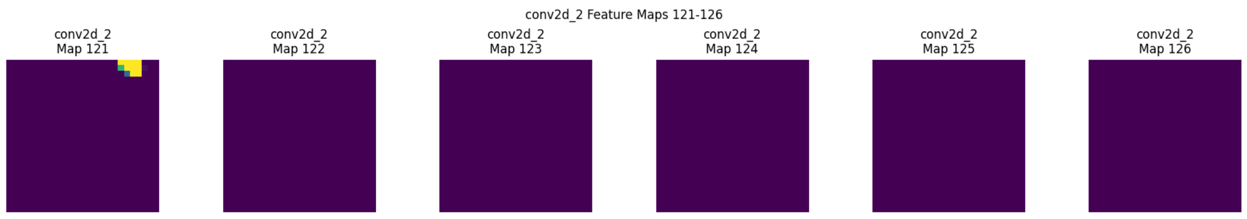
conv2d_2 Feature Maps 97-102

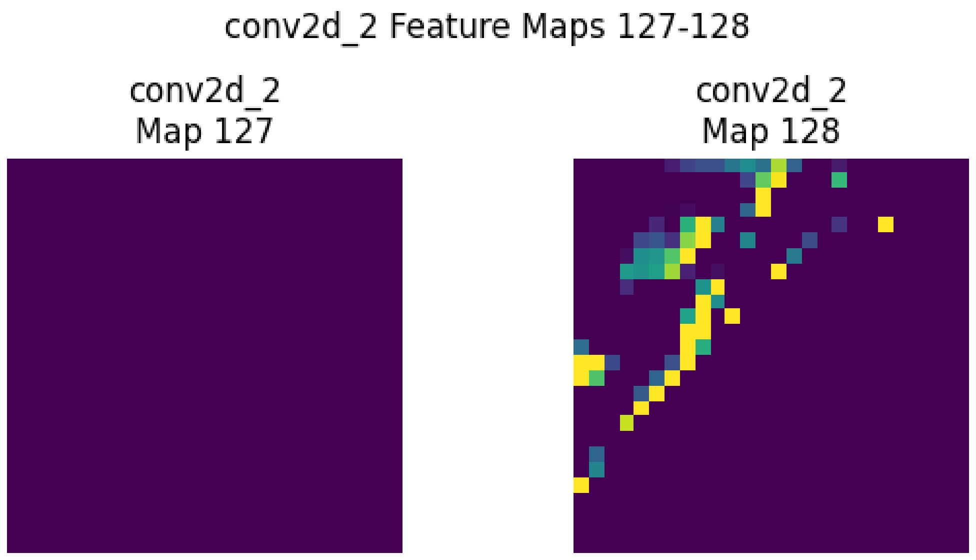




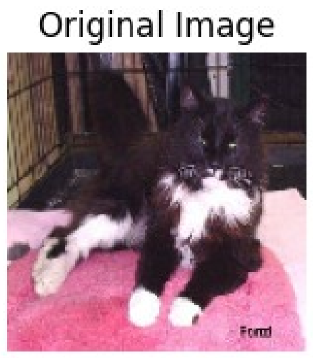


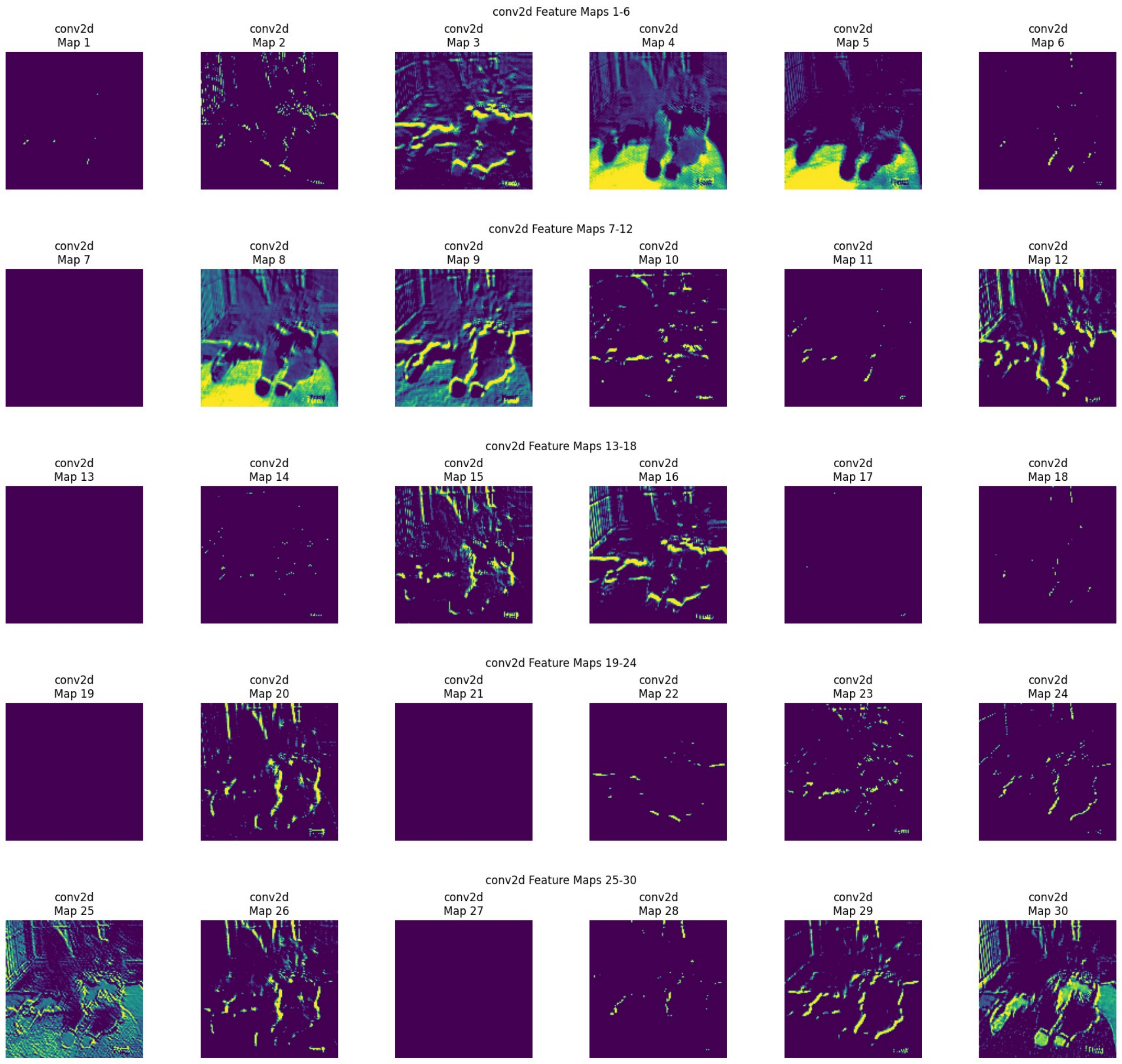




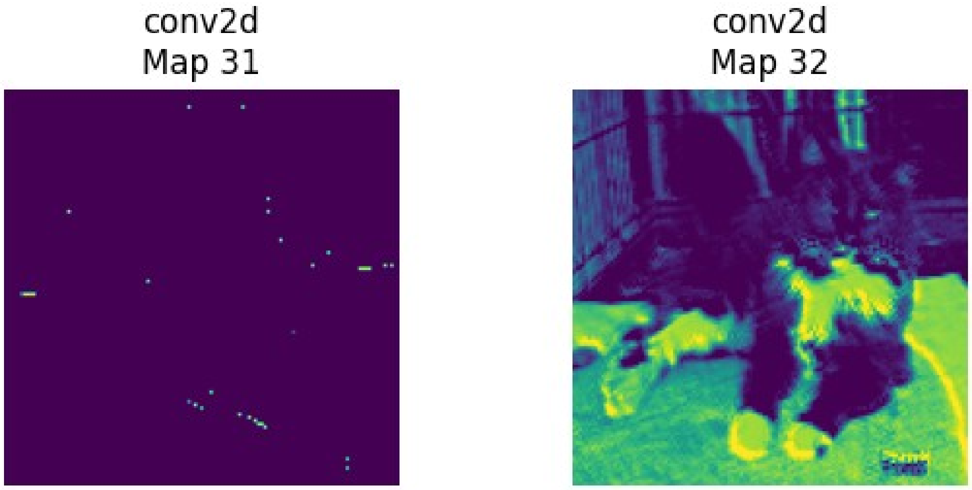


```
1/1 [==============================] - 0s 24ms/step

Feature maps for image: cat.4010.jpg
```

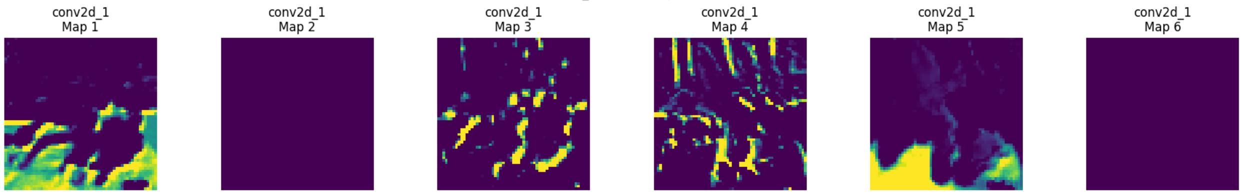

Original Image


conv2d Feature Maps 1-6
conv2d Map 1
conv2d Map 2
conv2d Map 3
conv2d Map 4
conv2d Map 5
conv2d Map 6
conv2d Feature Maps 7-12
conv2d Map 7
conv2d Map 8
conv2d Map 9
conv2d Map 10
conv2d Map 11
conv2d Map 12
conv2d Feature Maps 13-18
conv2d Map 13
conv2d Map 14
conv2d Map 15
conv2d Map 16
conv2d Map 17
conv2d Map 18
conv2d Feature Maps 19-24
conv2d Map 19
conv2d Map 20
conv2d Map 21
conv2d Map 22
conv2d Map 23
conv2d Map 24
conv2d Feature Maps 25-30
conv2d Map 25
conv2d Map 26
conv2d Map 27
conv2d Map 28
conv2d Map 29
conv2d Map 30

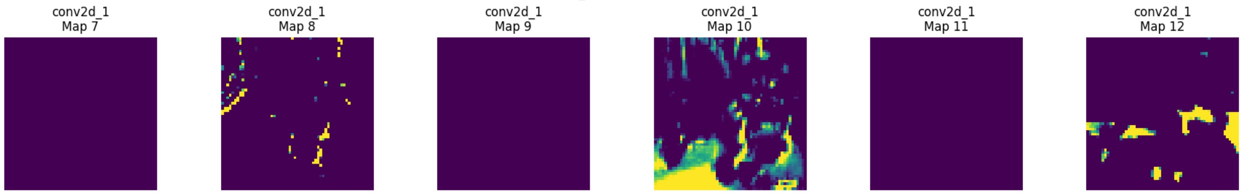
conv2d Feature Maps 31-32



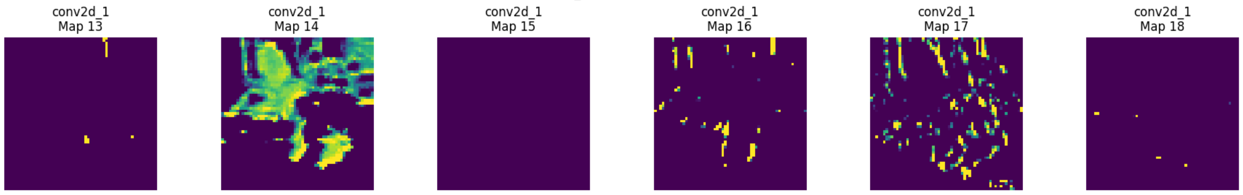
conv2d_1 Feature Maps 1-6



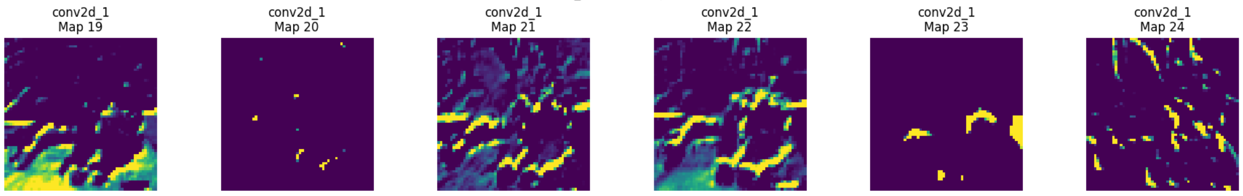
conv2d_1 Feature Maps 7-12



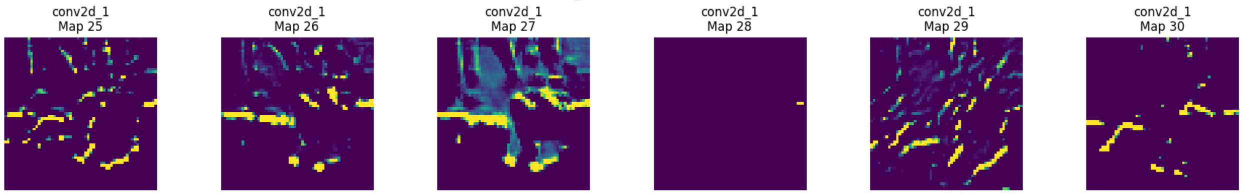
conv2d_1 Feature Maps 13-18



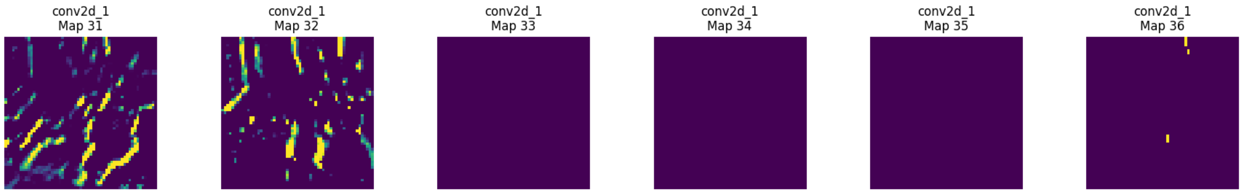
conv2d_1 Feature Maps 19-24



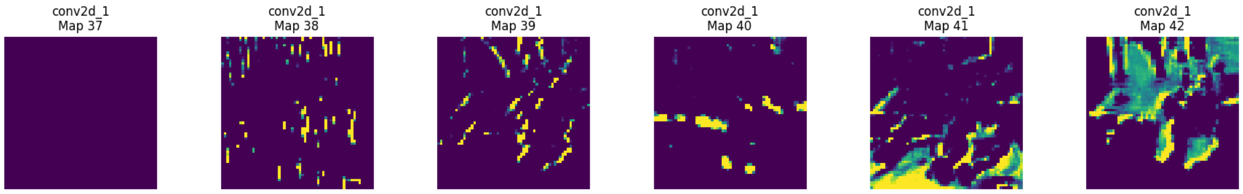
conv2d_1 Feature Maps 25-30



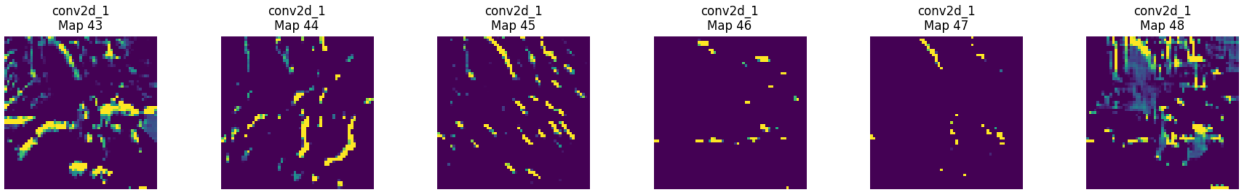
conv2d_1 Feature Maps 31-36



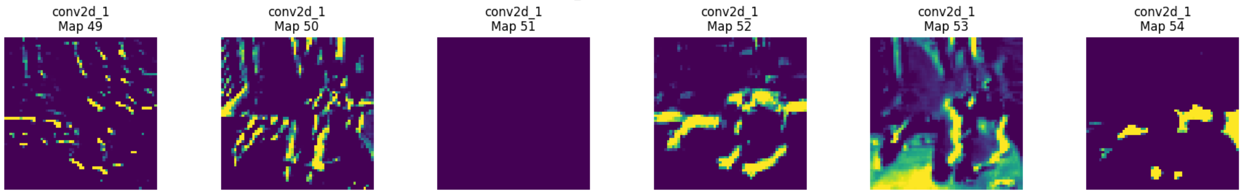
conv2d_1 Feature Maps 37-42



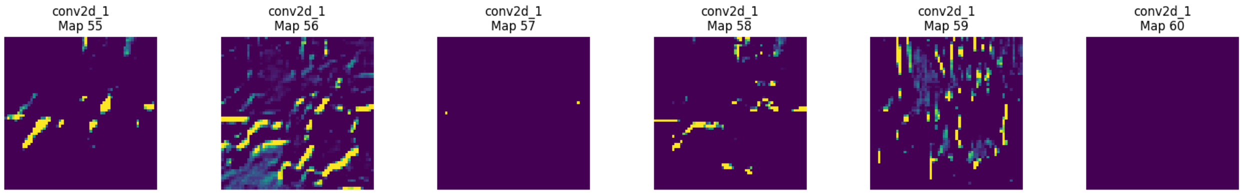
conv2d_1 Feature Maps 43-48



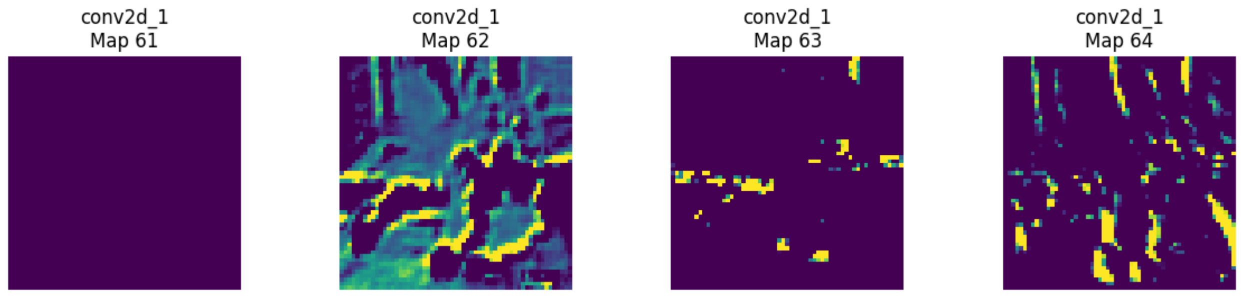
conv2d_1 Feature Maps 49-54



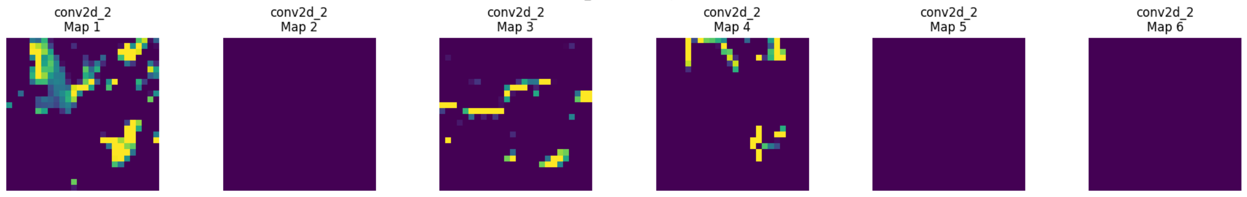
conv2d_1 Feature Maps 55-60



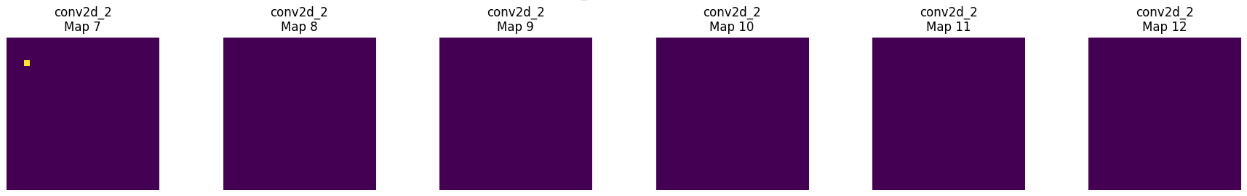
conv2d_1 Feature Maps 61-64



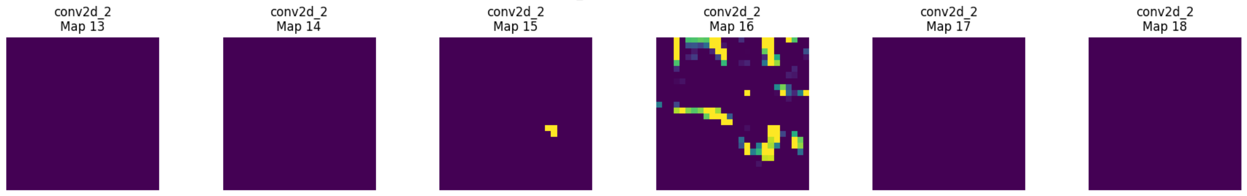
conv2d_2 Feature Maps 1-6



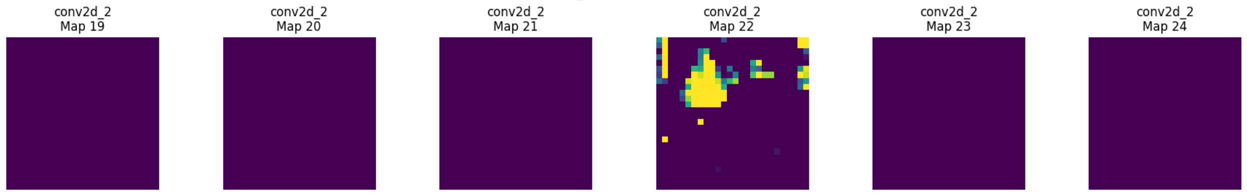
conv2d_2 Feature Maps 7-12



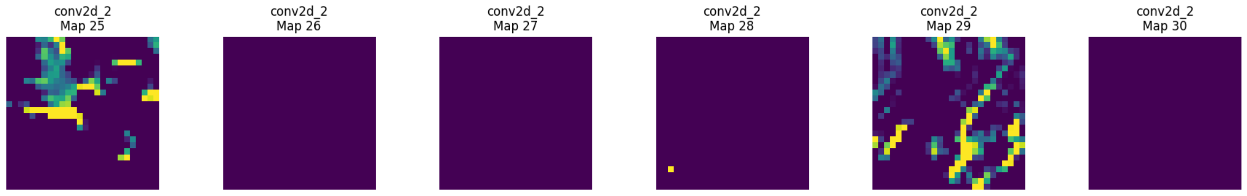
conv2d_2 Feature Maps 13-18



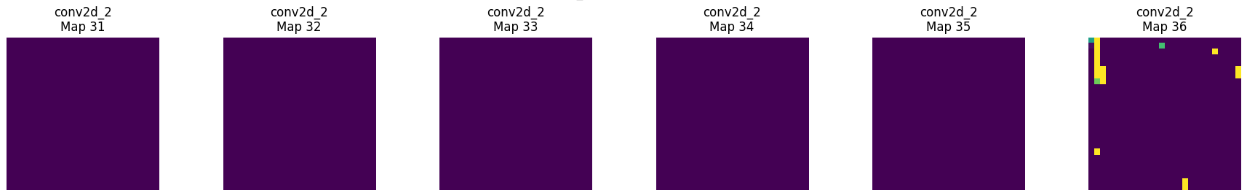
conv2d_2 Feature Maps 19-24



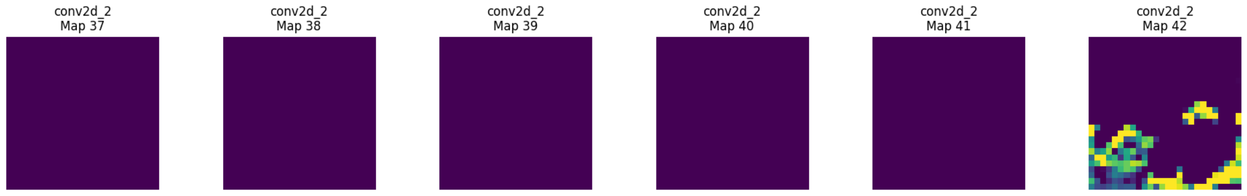
conv2d_2 Feature Maps 25-30



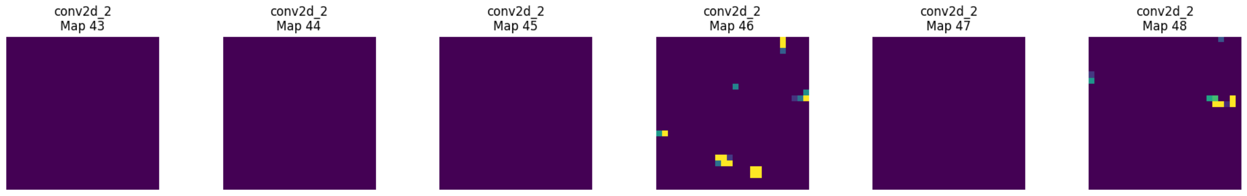
conv2d_2 Feature Maps 31-36



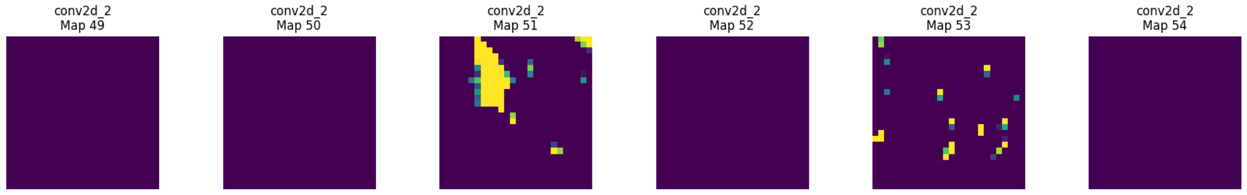
conv2d_2 Feature Maps 37-42



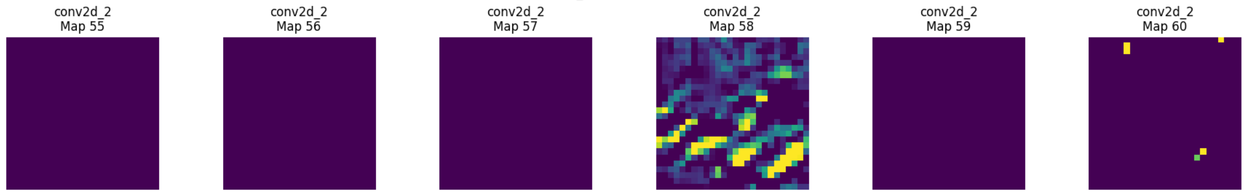
conv2d_2 Feature Maps 43-48



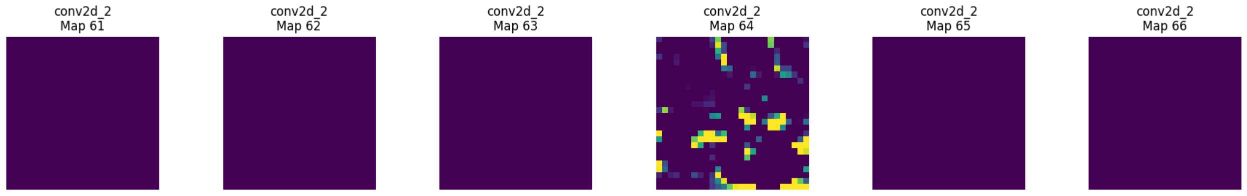
conv2d_2 Feature Maps 49-54



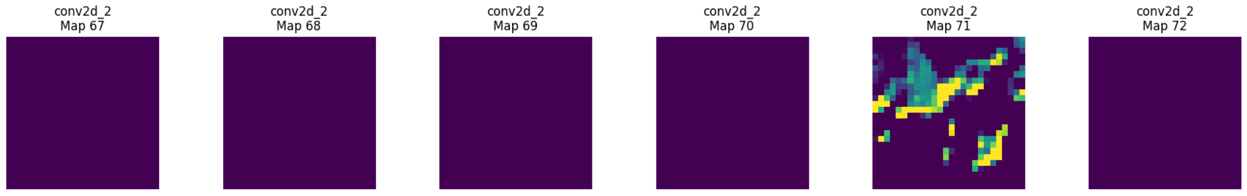
conv2d_2 Feature Maps 55-60



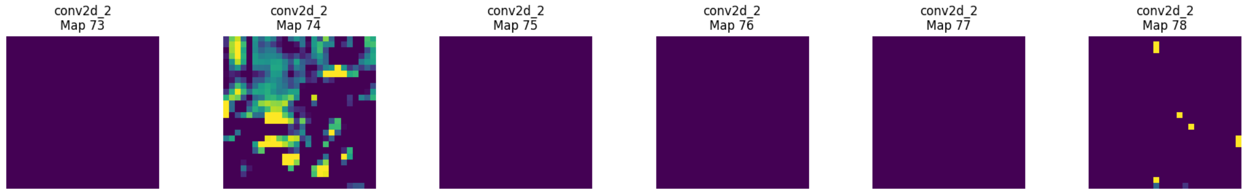
conv2d_2 Feature Maps 61-66



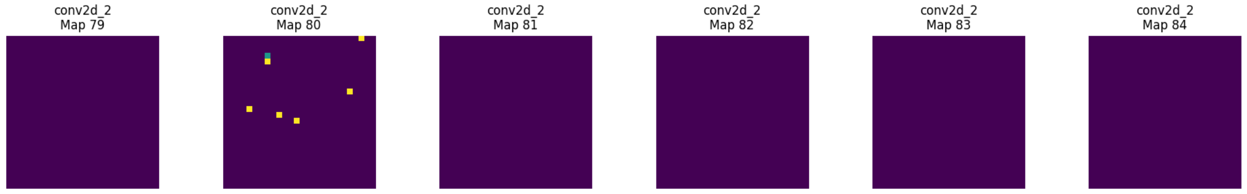
conv2d_2 Feature Maps 67-72



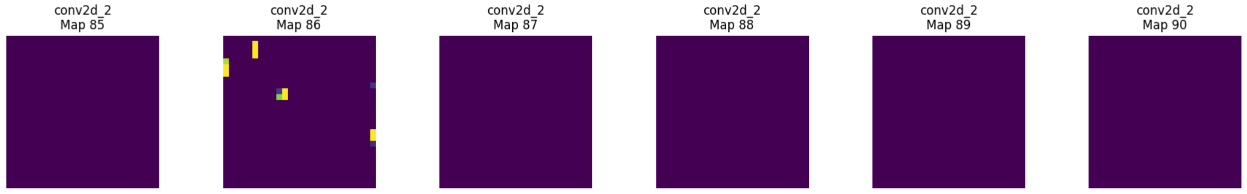
conv2d_2 Feature Maps 73-78



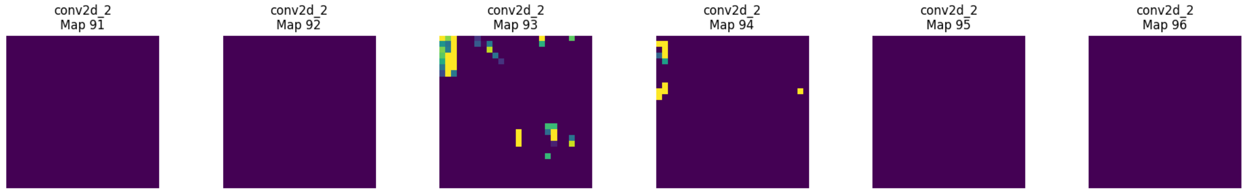
conv2d_2 Feature Maps 79-84



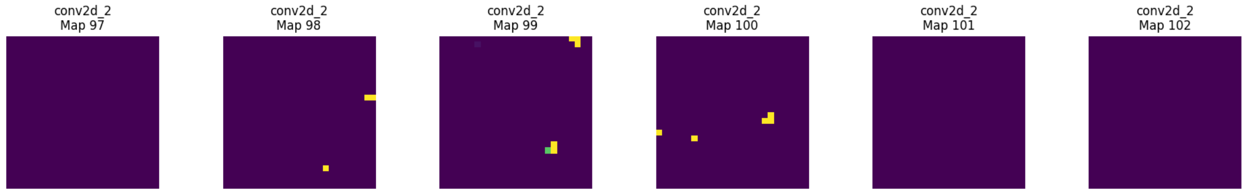
conv2d_2 Feature Maps 85-90



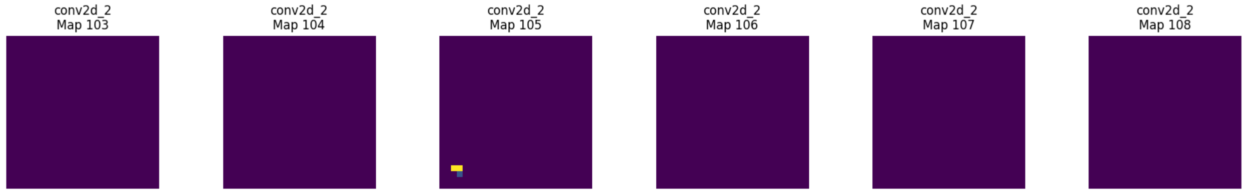
conv2d_2 Feature Maps 91-96



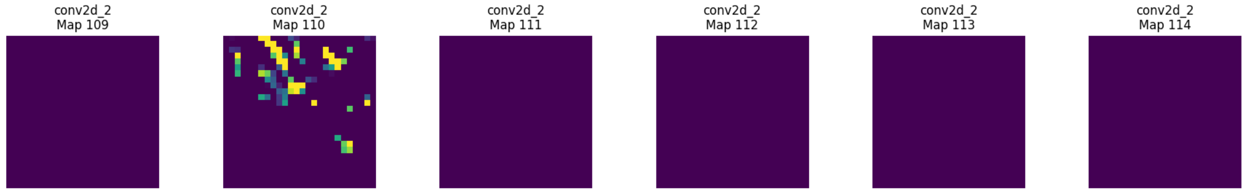
conv2d_2 Feature Maps 97-102



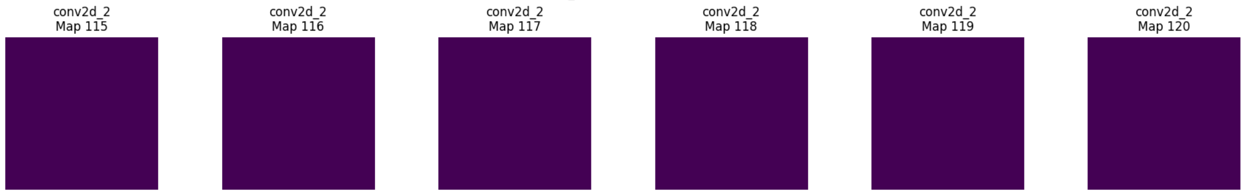
conv2d_2 Feature Maps 103-108



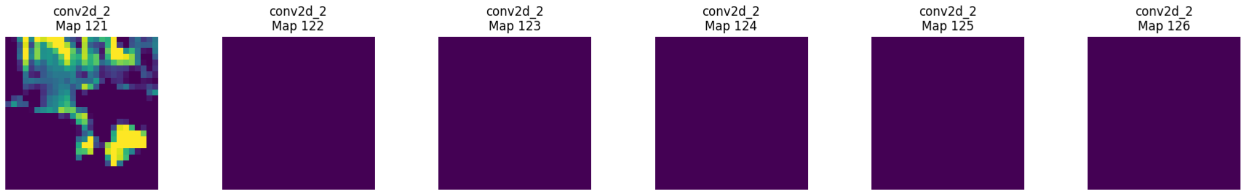
conv2d_2 Feature Maps 109-114



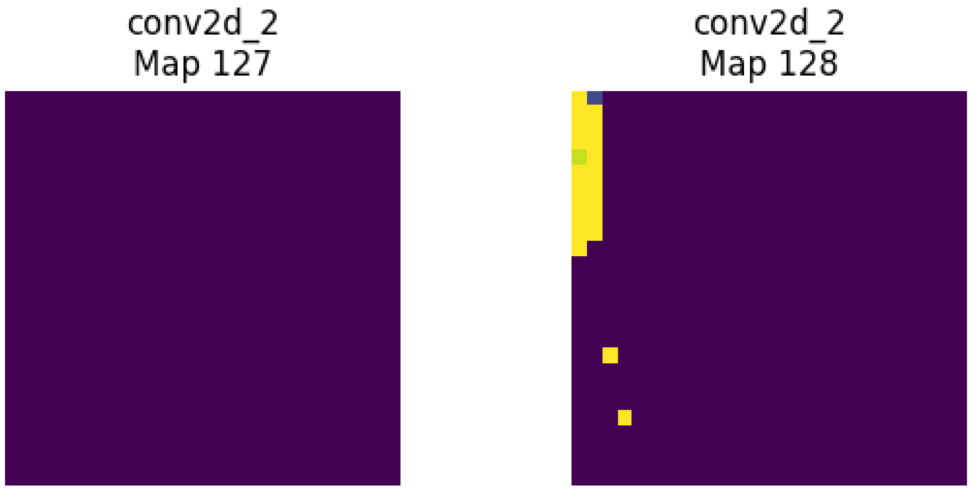
conv2d_2 Feature Maps 115-120



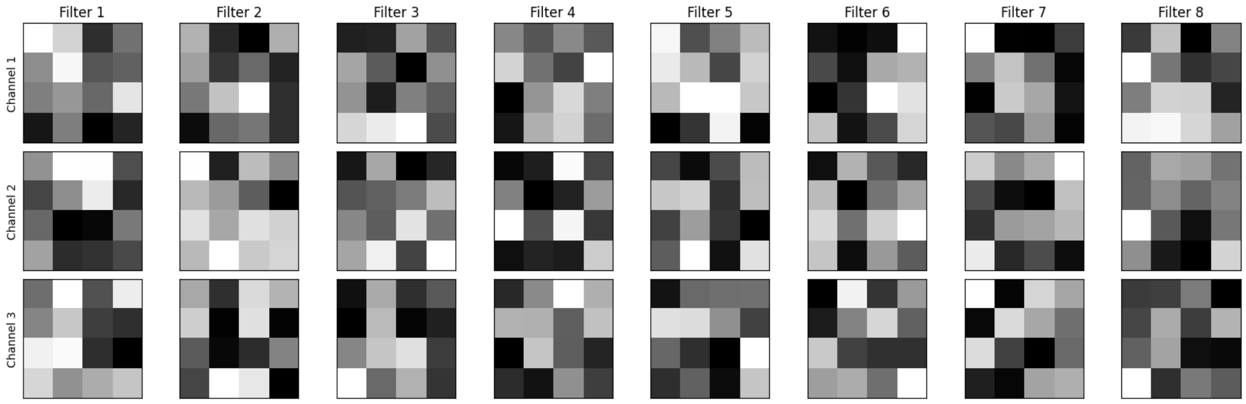
conv2d_2 Feature Maps 121-126



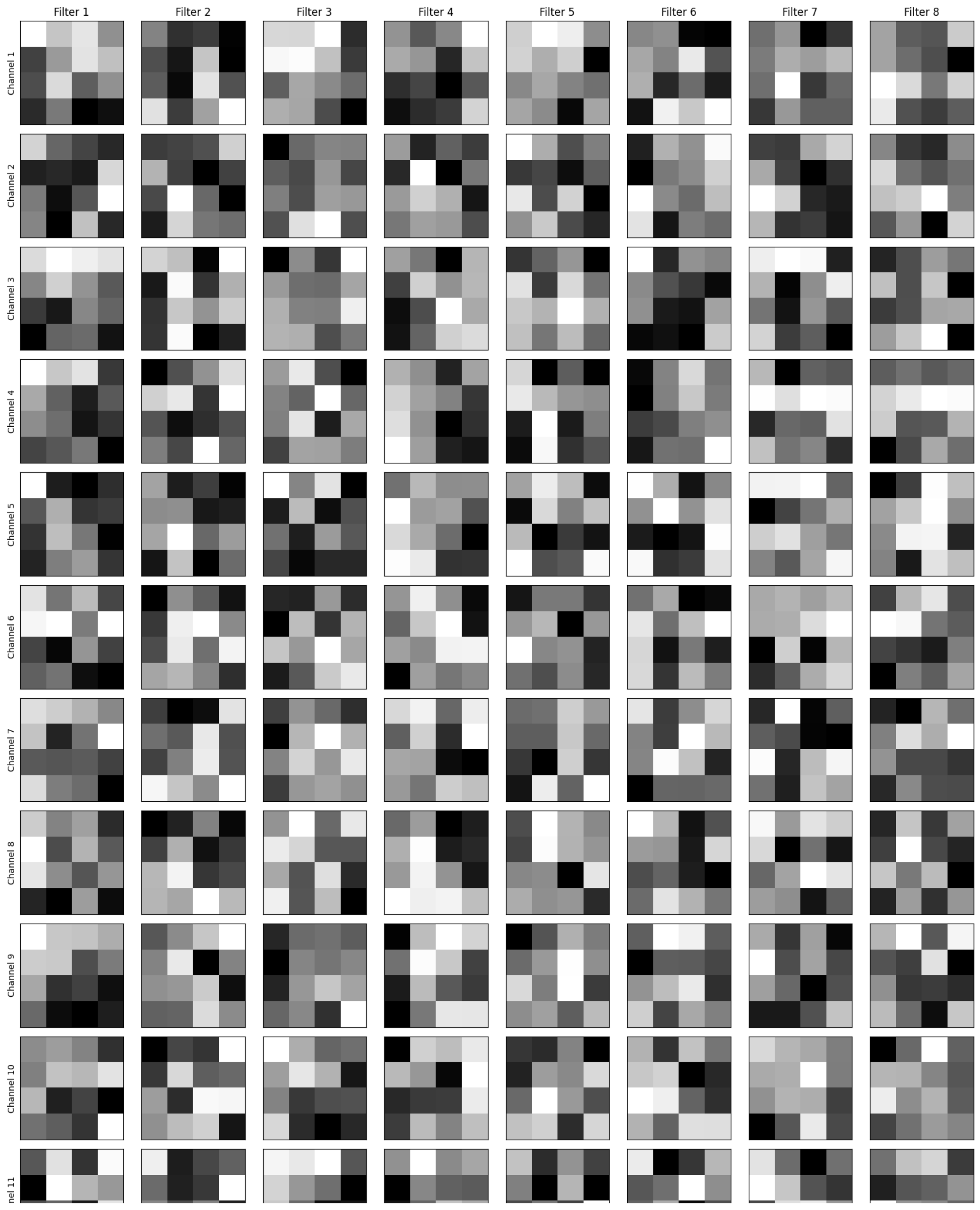
conv2d_2 Feature Maps 127-128



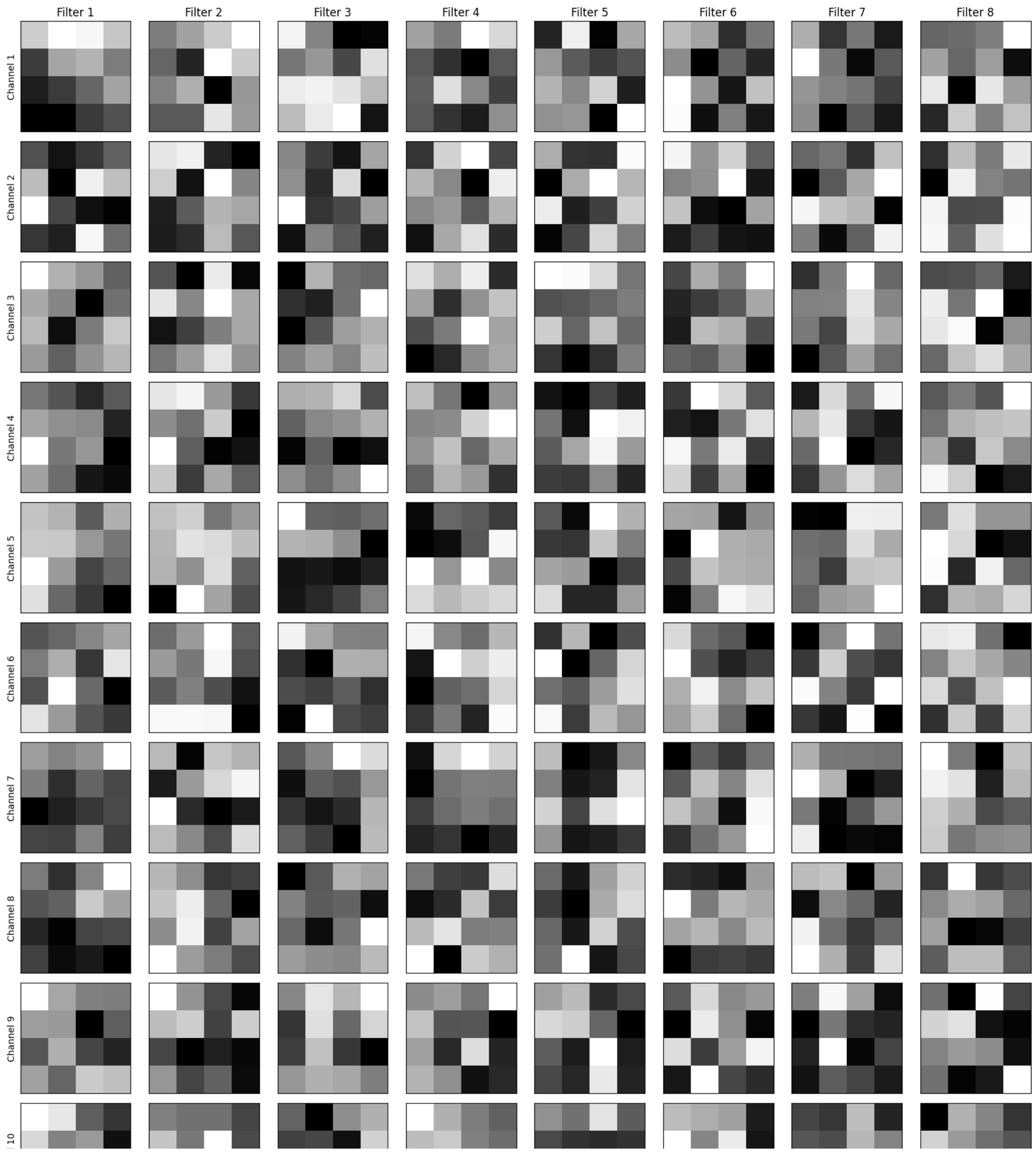
Learned Filters for conv2d



Learned Filters for conv2d_1

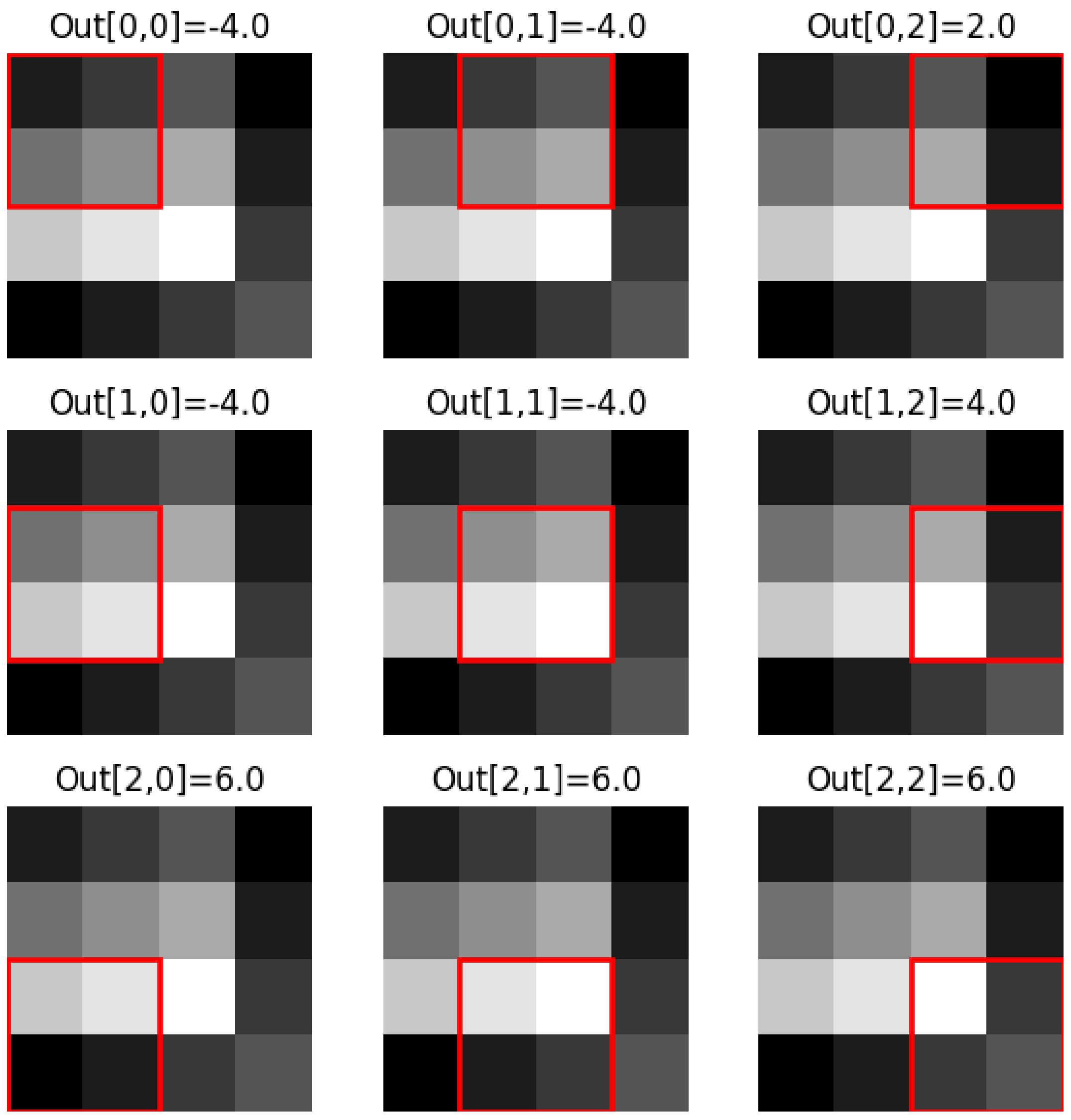


Learned Filters for conv2d_2



```python
import numpy as np
import matplotlib.pyplot as plt

def visualize_convolution_step(image, kernel, stride=1):
    """
    Visualize the sliding window and convolution calculation for a
single filter.
    Args:
        image: 2D numpy array (grayscale image)
        kernel: 2D numpy array (filter)
        stride: int, stride of the convolution
    """
    img_h, img_w = image.shape
    k_h, k_w = kernel.shape
    out_h = (img_h - k_h) // stride + 1
    out_w = (img_w - k_w) // stride + 1
    output = np.zeros((out_h, out_w))

    fig, axes = plt.subplots(out_h, out_w, figsize=(2*out_w, 2*out_h))
    for i in range(out_h):
        for j in range(out_w):
            y, x = i*stride, j*stride
            window = image[y:y+k_h, x:x+k_w]
            conv = np.sum(window * kernel)
            output[i, j] = conv
            # Visualization
            ax = axes[i, j]
            ax.imshow(image, cmap='gray')
            # Draw rectangle for current window
            rect = plt.Rectangle((x-0.5, y-0.5), k_w, k_h,
edgecolor='red', facecolor='none', linewidth=2)
            ax.add_patch(rect)
            ax.set_title(f"Out[{i},{j}]={conv:.1f}")
            ax.axis('off')
    plt.tight_layout()
    plt.show()
    print("Output feature map:\n", output)

# Example usage:
image = np.array([
    [1, 2, 3, 0],
    [4, 5, 6, 1],
    [7, 8, 9, 2],
    [0, 1, 2, 3]
], dtype=float)
kernel = np.array([
    [1, 0],
    [0, -1]
], dtype=float)
```

```
visualize_convolution_step(image, kernel, stride=1)
```



```
Output feature map:
 [[-4. -4.  2.]
 [-4. -4.  4.]
 [ 6.  6.  6.]]

from tensorflow.keras.preprocessing import image

def classify_image(model, img_path, img_width, img_height,
class_indices=None):
    """
    Loads an image, preprocesses it, and predicts its class using the
model.
```

```
    Args:
        model: Trained Keras model.
        img_path: Path to the image file.
        img_width, img_height: Target size for the image.
        class_indices: (Optional) Dictionary mapping class indices to
class names.
    """
    img = image.load_img(img_path, target_size=(img_width,
img_height))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0  # Normalize

    prediction = model.predict(img_array)[0][0]
    if class_indices:
        # Reverse the class_indices dictionary to get class names
        class_labels = {v: k for k, v in class_indices.items()}
        predicted_class = class_labels[int(round(prediction))]
        print(f"Predicted class: {predicted_class} (probability:
{prediction:.2f})")
    else:
        print(f"Predicted probability (1=dog, 0=cat):
{prediction:.2f}")

    plt.imshow(img)
    plt.title(f"Prediction: {prediction:.2f}")
    plt.axis('off')
    plt.show()

# Example usage:
img_path = 'dog.jpg'  # Change to your image path
if history is not None:
    classify_image(model, img_path, img_width, img_height,
class_indices=train_generator.class_indices)

1/1 [==============================] - 0s 27ms/step
Predicted class: dogs (probability: 0.89)
```

Prediction: 0.89