Convolutional Neural Networks on Architectural Heritage Elements Image64 Dataset

Danial Azimi

May 1, 2024

1 Introduction

In this work, a few simple and more advanced CNN models are trained on the image classification task. Also, the feature maps and the filters are visualized for better understanding of the model's performance. Furthermore, images were generated using the training of the model, indicating how the model can detect patterns, which is something for future work.

2 Dataset Description

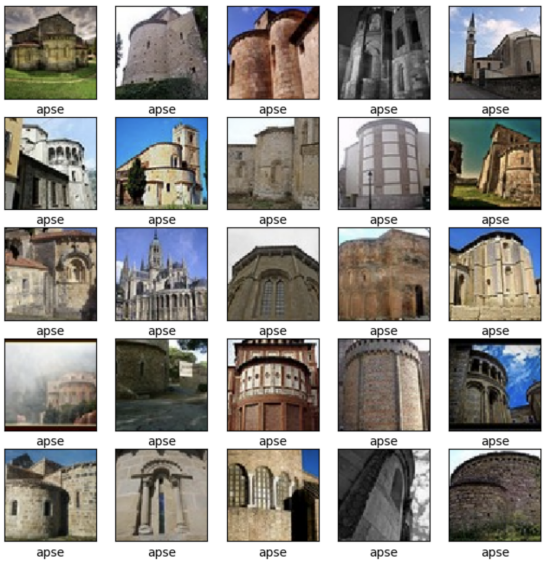
The Architectural Heritage Elements Dataset (AHE) is an image dataset for developing deep learning algorithms and specific techniques in the classification of architectural heritage images. This dataset consists of 10235 images classified into ten categories:

- Altar: 829 images
- Apse: 514 images
- Bell tower: 1059 images
- Column: 1919 images
- Dome (inner): 616 images
- Dome (outer): 1177 images
- Flying buttress: 407 images
- Gargoyle (and Chimera): 1571 images
- Stained glass: 1033 images
- Vault: 1110 images

3 Tasks

3.1 Data Preprocessing and Visualization

- These are the input images from one of the classes (apse):



- Scaling:
  The images were initially had numbers between 0,255 which were scaled to be between 0 and 1.

3.2 Implementation of The simple CNN Model using tensorflow

- This is the first model tested with 50 epochs.

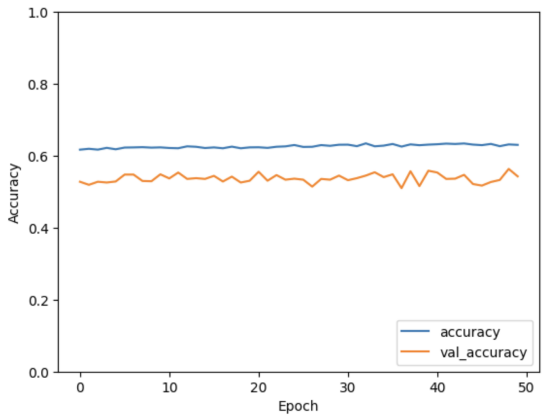
Architecture:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 98, 98, 32)	896
max_pooling2d (MaxPooling2D)	(None, 49, 49, 32)	0
conv2d_1 (Conv2D)	(None, 47, 47, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 23, 23, 64)	0
conv2d_2 (Conv2D)	(None, 21, 21, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 10, 10, 128)	0
flatten (Flatten)	(None, 12800)	0
dense (Dense)	(None, 128)	1,638,528
dense_1 (Dense)	(None, 10)	1,290

Trainable params: 1,733,066

Non-trainable params: 0

3.3 Results of 3.2



It can be seen from this figure that the learning was insufficient!

3.4 Implementation of ResNet-alike CNN Model using tensorflow

- This is one of the ResNet-alike architectures in the code, which had relatively better results and was trained for 100 epochs. It is to be mentioned that this model did not utilized any pre-training!

Architecture:

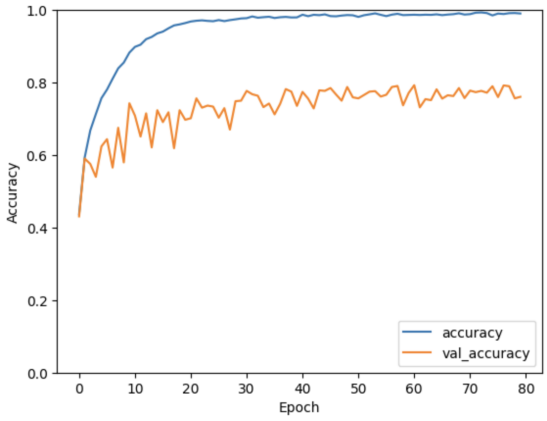
Layer (type)	Output Shape	Param #	Connected to
input_layer_9 (InputLayer)	(None, 100, 100, 3)	0	-
conv2d_103 (Conv2D)	(None, 98, 98, 32)	896	input_layer_9[0]…
conv2d_104 (Conv2D)	(None, 96, 96, 64)	18,496	conv2d_103[0][0]
max_pooling2d_16 (MaxPooling2D)	(None, 48, 48, 64)	0	conv2d_104[0][0]
conv2d_106 (Conv2D)	(None, 48, 48, 64)	36,928	max_pooling2d_16…
batch_normalizatio… (BatchNormalizatio…	(None, 48, 48, 64)	256	conv2d_106[0][0]
conv2d_107 (Conv2D)	(None, 48, 48, 64)	36,928	batch_normalizat…
batch_normalizatio… (BatchNormalizatio…	(None, 48, 48, 64)	256	conv2d_107[0][0]
conv2d_105 (Conv2D)	(None, 24, 24, 64)	4,160	max_pooling2d_16…
conv2d_108 (Conv2D)	(None, 24, 24, 64)	4,160	batch_normalizat…
batch_normalizatio… (BatchNormalizatio…	(None, 24, 24, 64)	256	conv2d_105[0][0]
add_20 (Add)	(None, 24, 24, 64)	0	conv2d_108[0][0], batch_normalizat…
activation_17 (Activation)	(None, 24, 24, 64)	0	add_20[0][0]
conv2d_109 (Conv2D)	(None, 22, 22, 64)	36,928	activation_17[0]…
max_pooling2d_17 (MaxPooling2D)	(None, 11, 11, 64)	0	conv2d_109[0][0]
conv2d_111 (Conv2D)	(None, 11, 11, 128)	73,856	max_pooling2d_17…
batch_normalizatio… (BatchNormalizatio…	(None, 11, 11, 128)	512	conv2d_111[0][0]
conv2d_112 (Conv2D)	(None, 11, 11, 128)	147,584	batch_normalizat…
batch_normalizatio… (BatchNormalizatio…	(None, 11, 11, 128)	512	conv2d_112[0][0]

conv2d_110 (Conv2D)	(None, 6, 6, 128)	8,320	max_pooling2d_17…
conv2d_113 (Conv2D)	(None, 6, 6, 128)	16,512	batch_normalizat…
batch_normalizatio… (BatchNormalizatio…	(None, 6, 6, 128)	512	conv2d_110[0][0]
add_21 (Add)	(None, 6, 6, 128)	0	conv2d_113[0][0], batch_normalizat…
activation_18 (Activation)	(None, 6, 6, 128)	0	add_21[0][0]
conv2d_115 (Conv2D)	(None, 6, 6, 256)	295,168	activation_18[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 6, 6, 256)	1,024	conv2d_115[0][0]
conv2d_116 (Conv2D)	(None, 6, 6, 256)	590,080	batch_normalizat…
batch_normalizatio… (BatchNormalizatio…	(None, 6, 6, 256)	1,024	conv2d_116[0][0]
conv2d_114 (Conv2D)	(None, 3, 3, 256)	33,024	activation_18[0]…
conv2d_117 (Conv2D)	(None, 3, 3, 256)	65,792	batch_normalizat…
batch_normalizatio… (BatchNormalizatio…	(None, 3, 3, 256)	1,024	conv2d_114[0][0]
add_22 (Add)	(None, 3, 3, 256)	0	conv2d_117[0][0], batch_normalizat…
activation_19 (Activation)	(None, 3, 3, 256)	0	add_22[0][0]
flatten_6 (Flatten)	(None, 2304)	0	activation_19[0]…
dropout_6 (Dropout)	(None, 2304)	0	flatten_6[0][0]
dense_14 (Dense)	(None, 64)	147,520	dropout_6[0][0]
dense_15 (Dense)	(None, 10)	650	dense_14[0][0]

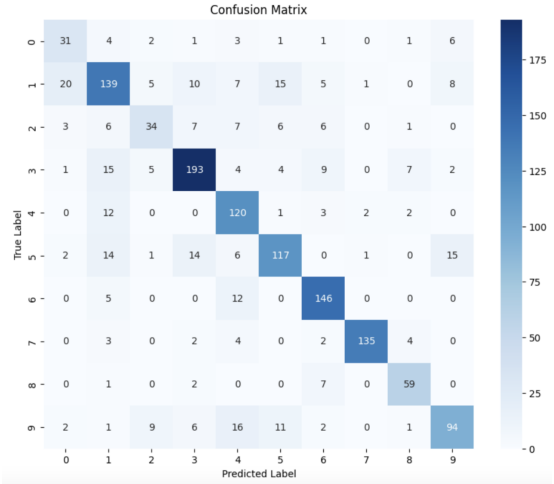
Trainable params: 1,519,690

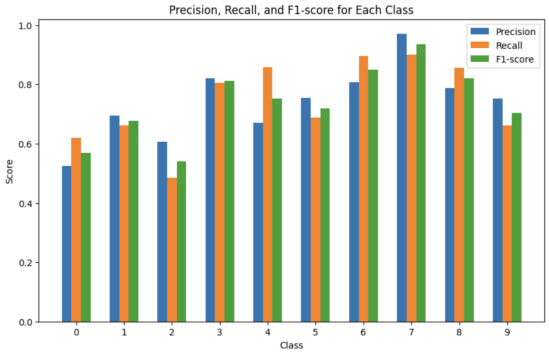
Non-trainable params: 2,688

3.5 Results of 3.4



Results on the Evaluation Metrics
Confusion Matrix:





	Class	Precision	Recall	F1 Score
0	0	0.525424	0.620000	0.568807
1	1	0.695000	0.661905	0.678049
2	2	0.607143	0.485714	0.539683
3	3	0.821277	0.804167	0.812632
4	4	0.670391	0.857143	0.752351
5	5	0.754839	0.688235	0.720000
6	6	0.806630	0.895706	0.848837
7	7	0.971223	0.900000	0.934256
8	8	0.786667	0.855072	0.819444
9	9	0.752000	0.661972	0.704120
Overall	-	0.763699	0.760684	0.759540

The above results show that classes with less samples act poorly, and data augmentation may help in this case

3.6 Implementation of The Xception-alike CNN Model using tensorflow

- This is another famous robust CNN architecture, which was trained only for 50 epochs to see if it could learn something more than the other architectures.

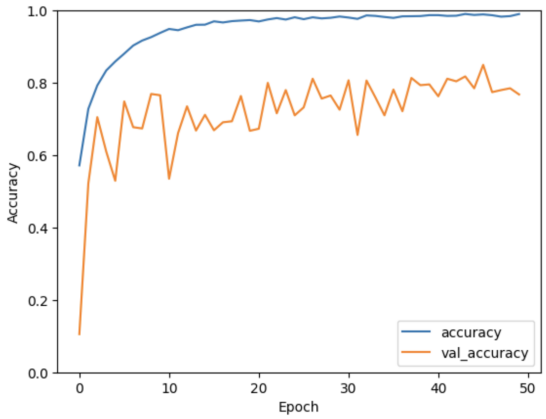
  Architecture:

  It is too big to visualize, to see the exact architecture please refer to the code!

Trainable params: 785,706

Non-trainable params: 9,664

3.7 Results of 3.6



The figure demonstrates that the result of this modeling was almost on par with the ResNet, or maybe just a bit better, so it didn't add anything special to us!

3.8 Implementation of The VGG-alike CNN Model using tensorflow

- This is another famous robust CNN architecture, which was trained only for 100 epochs to see if it could learn something more than the other architectures.

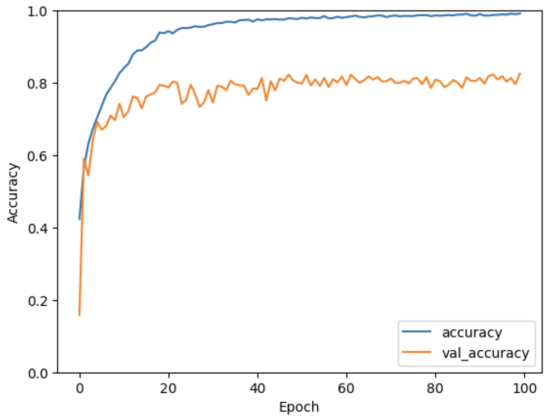
Architecture:

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 98, 98, 32)	896
batch_normalization_33 (BatchNormalization)	(None, 98, 98, 32)	128
conv2d_5 (Conv2D)	(None, 96, 96, 32)	9,248
batch_normalization_34 (BatchNormalization)	(None, 96, 96, 32)	128
max_pooling2d_2 (MaxPooling2D)	(None, 48, 48, 32)	0
dropout (Dropout)	(None, 48, 48, 32)	0
conv2d_6 (Conv2D)	(None, 46, 46, 64)	18,496
batch_normalization_35 (BatchNormalization)	(None, 46, 46, 64)	256
conv2d_7 (Conv2D)	(None, 44, 44, 64)	36,928
batch_normalization_36 (BatchNormalization)	(None, 44, 44, 64)	256
max_pooling2d_3 (MaxPooling2D)	(None, 22, 22, 64)	0
dropout_1 (Dropout)	(None, 22, 22, 64)	0
conv2d_8 (Conv2D)	(None, 20, 20, 128)	73,856
batch_normalization_37 (BatchNormalization)	(None, 20, 20, 128)	512
conv2d_9 (Conv2D)	(None, 18, 18, 128)	147,584
batch_normalization_38 (BatchNormalization)	(None, 18, 18, 128)	512
max_pooling2d_4 (MaxPooling2D)	(None, 9, 9, 128)	0
dropout_2 (Dropout)	(None, 9, 9, 128)	0
conv2d_10 (Conv2D)	(None, 7, 7, 256)	295,168
batch_normalization_39 (BatchNormalization)	(None, 7, 7, 256)	1,024
conv2d_11 (Conv2D)	(None, 5, 5, 256)	590,080
batch_normalization_40 (BatchNormalization)	(None, 5, 5, 256)	1,024
max_pooling2d_5 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_3 (Dropout)	(None, 2, 2, 256)	0
flatten (Flatten)	(None, 1024)	0
dense_2 (Dense)	(None, 512)	524,800
batch_normalization_41 (BatchNormalization)	(None, 512)	2,048
dropout_4 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 10)	5,130

Trainable params: 1,705,130

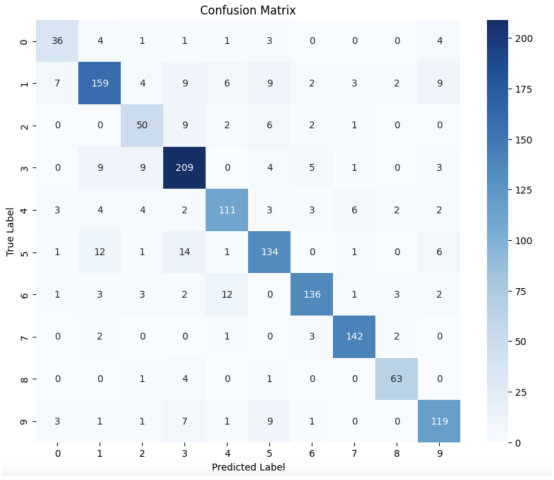
Non-trainable params: 2,944

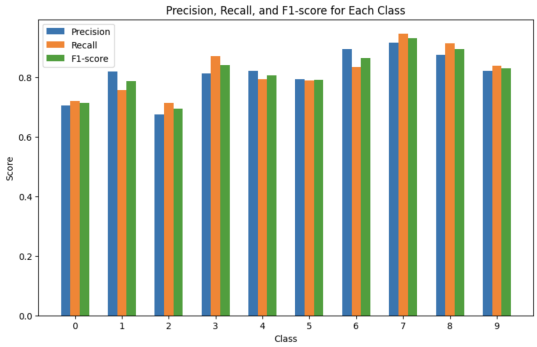
3.9 Results of 3.8



Results on the Evaluation Metrics

Confusion Matrix:

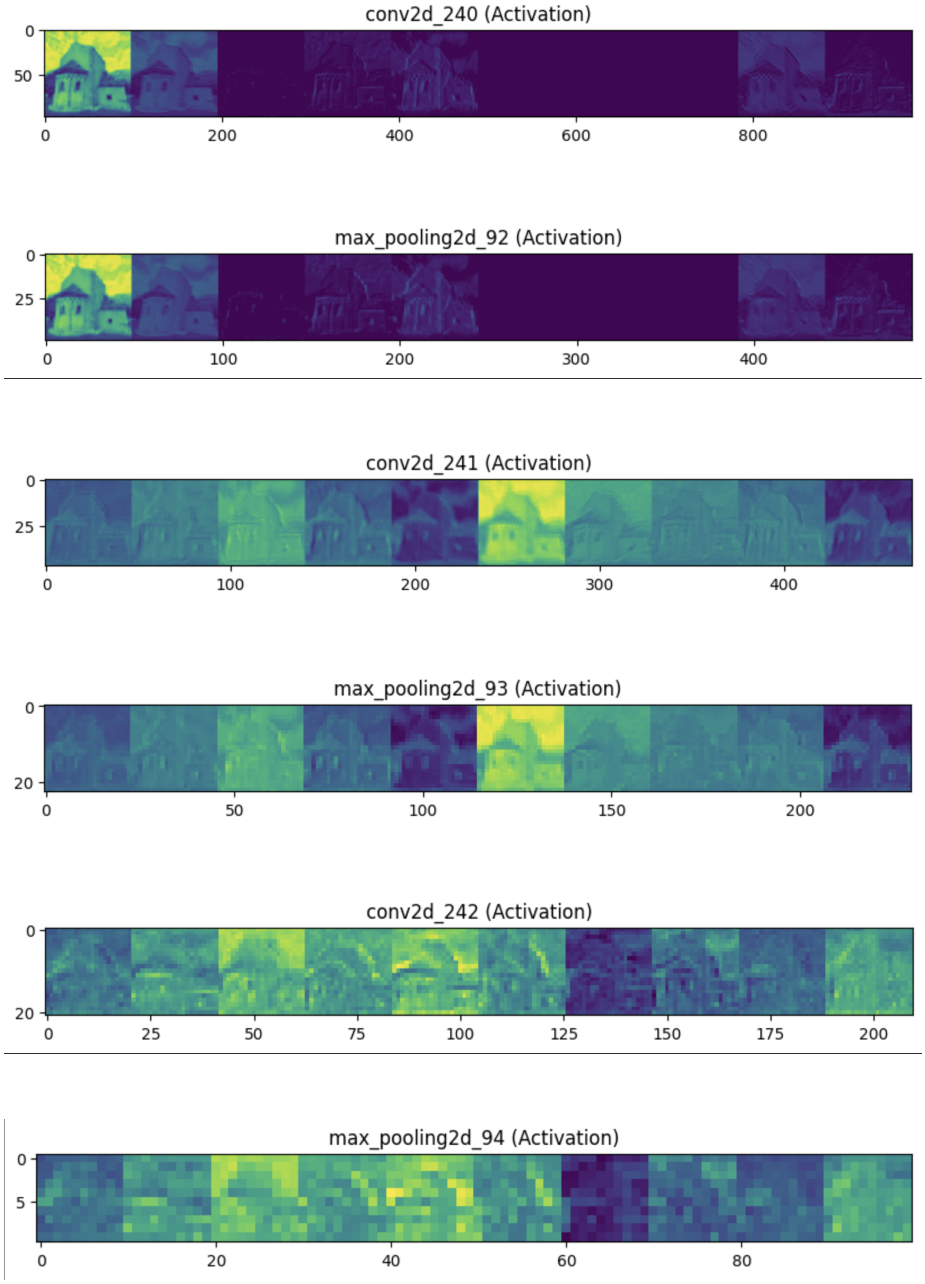




	Class	Precision	Recall	F1 Score
0	0	0.705882	0.720000	0.712871
1	1	0.819588	0.757143	0.787129
2	2	0.675676	0.714286	0.694444
3	3	0.813230	0.870833	0.841046
4	4	0.822222	0.792857	0.807273
5	5	0.792899	0.788235	0.790560
6	6	0.894737	0.834356	0.863492
7	7	0.916129	0.946667	0.931148
8	8	0.875000	0.913043	0.893617
9	9	0.820690	0.838028	0.829268
Overall	-	0.826181	0.825499	0.825252

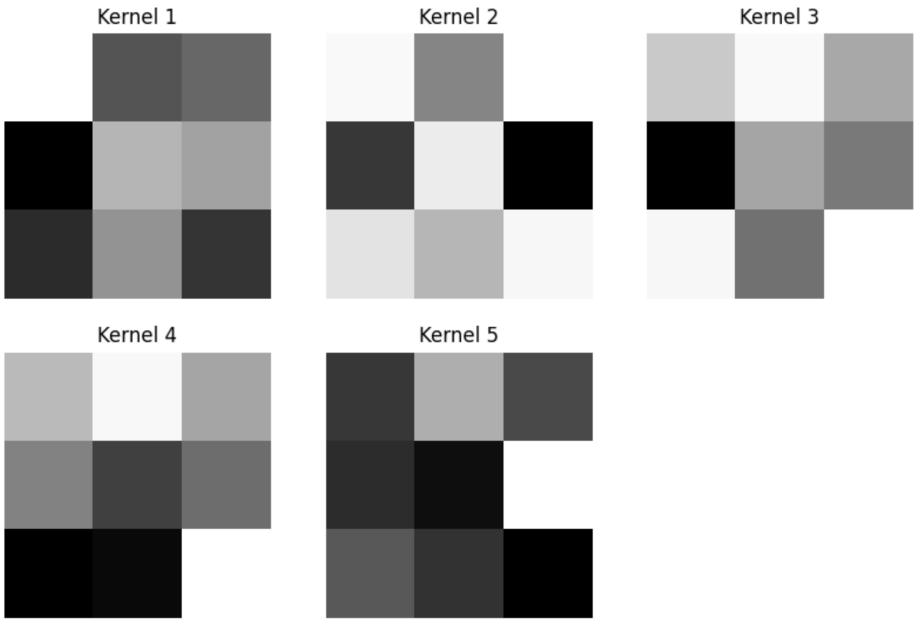
The above results show that this model was our best model trained from scratch!

3.10 Visualization of inputs passed to a CNN model



The Effect of these layers is obvious!

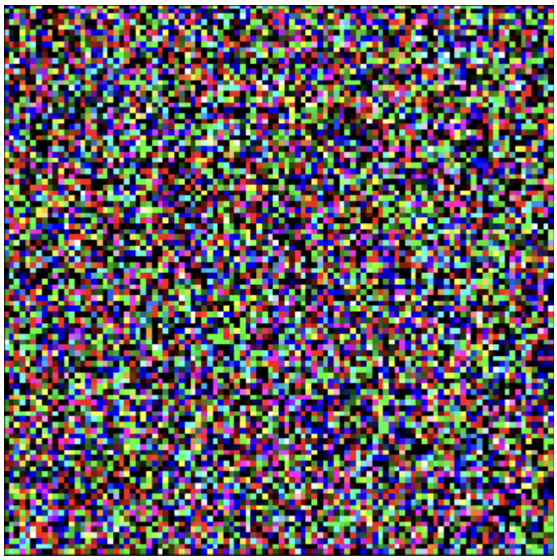
3.11 Visualization of feature maps



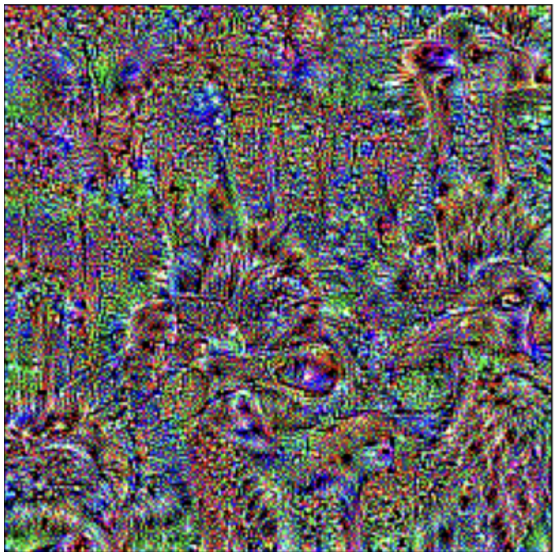
- This is just the visualization of the 1st layer's trained kernels, which in combination are used to detect patterns in images.

3.12 Data Generalization using the networks trained

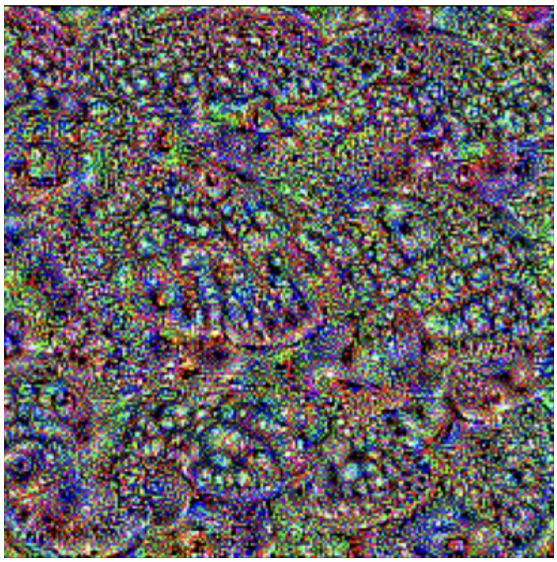
- This part generates an image that maximally activates a specific class, generating an image of that particular class, based on the features it has detected earlier on that class.
  (This is done on an image placeholder.)



The above is the result on the ResNet model, which is not very different than noise, so I'll use a pre-trained VGG model training to show how it can generate images, when trained enough!



The above shows that some patterns are learned, but this is not good enough to be considered a real image! (for that GANs and other Generative models are great)



This is the result on another class. The difference in patterns made for this class and the others like the previous figure, are really interesting, showing how the model is learning the patterns.

3.13 Conclusion

- In conclusion, the experiments conducted demonstrate the effectiveness of different CNN architectures and how they learn and detect patterns, particularly in this case for image classification task.