Machine learning report – Image Classification

Julia Ordecka

Matricola 2183686

In order to solve this image classification problem I defined two models based on CNN.

Data preprocessing

Normalization was applied to the training set and to the validation set by rescaling and normalizing the pixel values for the pictures by the rescale parameter (rescale=1.0/255.0) – then the pixel values are constrained to the range from 0 to 1.

I reduced the resolution of the pictures to 64x64 pixels in order to reduce time of computation. I specified a batch size of 128 and used sparse class mode since the target labels are integers. I reduced the amount of filters in convolutional layers and increased the dense layer to 128 units.

First model:

In order to reduce time of computation which was at first very long and inefficient (about 30 minutes for 1 epoch) I increased the batch size from 32 to 128. It was the biggest contributor in the reduction of model training time.

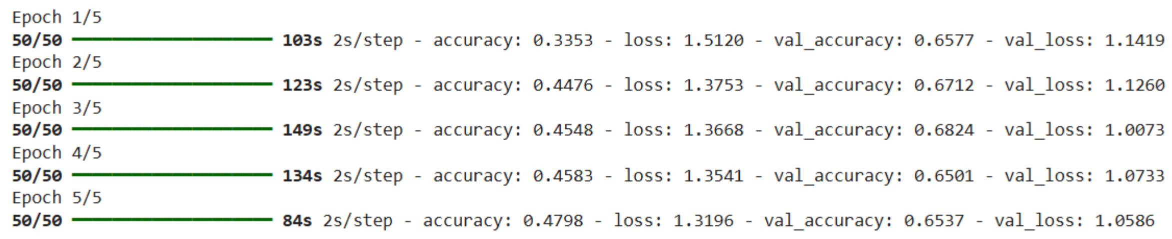
I chose sparse categorical crossentropy as a loss function as it accepts integer labels and actions are numbered from 0 to 4.

After reducing the convolutional layers to 16 and 32 training the model stopped after 1 epoch so i changed the convolutional layers to 32 and 64.

I used the Adam optimizer.

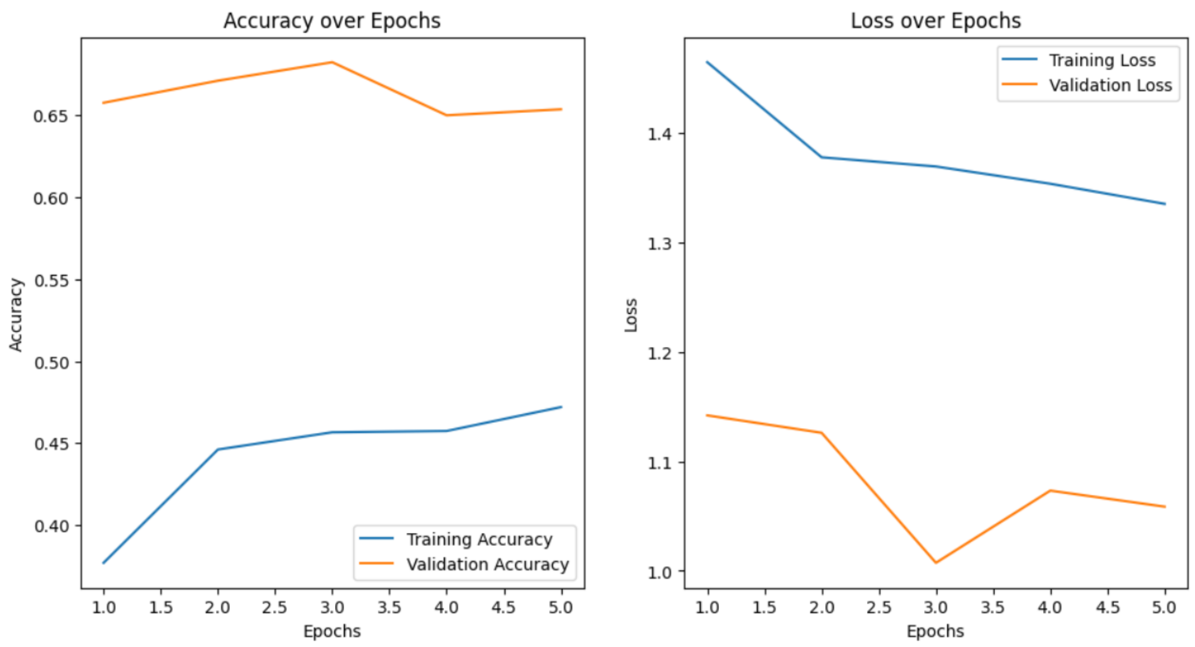
While tuning the hyperparameters, I trained the model for 5 epochs since the training was at first very time consuming.

The implemented model had quite poor performance. It showed inconsistent changes in accuracy after the first epoch but decreased the loss.



```
Epoch 1/5
50/50 ━━━━━━━━━━━━━━━━━━━━ 103s 2s/step - accuracy: 0.3353 - loss: 1.5120 - val_accuracy: 0.6577 - val_loss: 1.1419
Epoch 2/5
50/50 ━━━━━━━━━━━━━━━━━━━━ 123s 2s/step - accuracy: 0.4476 - loss: 1.3753 - val_accuracy: 0.6712 - val_loss: 1.1260
Epoch 3/5
50/50 ━━━━━━━━━━━━━━━━━━━━ 149s 2s/step - accuracy: 0.4548 - loss: 1.3668 - val_accuracy: 0.6824 - val_loss: 1.0073
Epoch 4/5
50/50 ━━━━━━━━━━━━━━━━━━━━ 134s 2s/step - accuracy: 0.4583 - loss: 1.3541 - val_accuracy: 0.6501 - val_loss: 1.0733
Epoch 5/5
50/50 ━━━━━━━━━━━━━━━━━━━━ 84s 2s/step - accuracy: 0.4798 - loss: 1.3196 - val_accuracy: 0.6537 - val_loss: 1.0586
```

The time taken to train the model for 5 epochs was 593 seconds.



```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       133
           1       0.10      0.16      0.13       275
           2       0.14      0.23      0.18       406
           3       0.70      0.60      0.65      1896
           4       0.00      0.00      0.00        39

    accuracy                           0.47      2749
   macro avg       0.19      0.20      0.19      2749
weighted avg       0.51      0.47      0.49      2749

[[   0   26   32   75    0]
 [   2   45   64  164    0]
 [   0   69   95  242    0]
 [   3  287  459 1147    0]
 [   0    5   14   20    0]]
F1 Score: 0.48540001398030364
Accuracy: 0.4681702437249909
```

The accuracy value of ~0,47 means that the model correctly predicted the label for 47% of the test samples which suggest that there is room for improving the model. The F1 score is equal to 0,485 and the value tells how well the model balances precision and recall. Recall for class 0 and class 4 equals to 0, which indicates that the model is failing to classify the images to these classes at all. Recall for class 3 equals 0,6 which is much better. Recall for class 1 and 2 have values of 0,16 and 0,23.

Macro average equals to 0.19 (precision), 0.20 (recall), 0.19 (f1-score) and weighted average equals to 0.51 (precision), 0.47 (recall), 0.49 (f1-score).

The confusion matrix shows that class 0 was never predicted as class 0, it was predicted as class 1 26 times, it was predicted as class 2 32 times, predicted as class 3 75 times, and predicted as class 4 0 times. Similarly, class 4 was never predicted as class 4. Most successfully, class 3 was correctly predicted 1147 times.

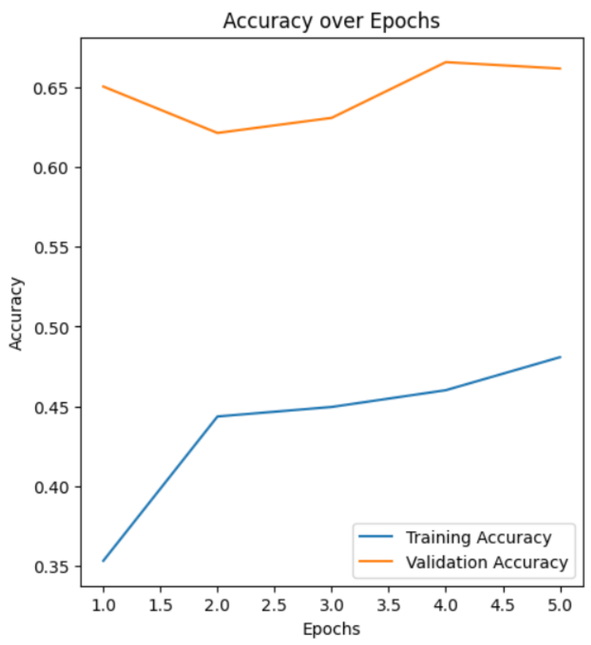
While hypertuning the parameters I checked whether increasing the image resolution to 96x96 would make a significant difference in performance of the model.

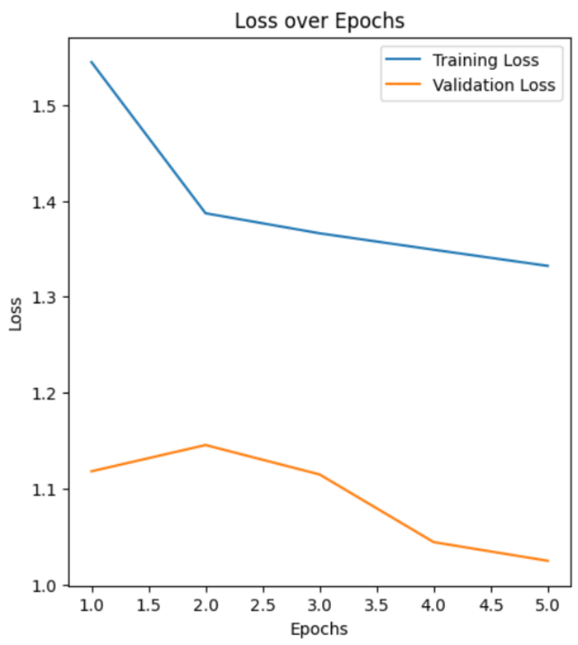
```
Epoch 1/5
50/50 ━━━━━━━━━━━━━━━━━━━━ 154s 3s/step - accuracy: 0.2898 - loss: 1.7513 - val_accuracy: 0.6504 - val_loss: 1.1179
Epoch 2/5
50/50 ━━━━━━━━━━━━━━━━━━━━ 199s 3s/step - accuracy: 0.4374 - loss: 1.3861 - val_accuracy: 0.6213 - val_loss: 1.1453
Epoch 3/5
50/50 ━━━━━━━━━━━━━━━━━━━━ 201s 3s/step - accuracy: 0.4457 - loss: 1.3744 - val_accuracy: 0.6308 - val_loss: 1.1146
Epoch 4/5
50/50 ━━━━━━━━━━━━━━━━━━━━ 200s 3s/step - accuracy: 0.4526 - loss: 1.3606 - val_accuracy: 0.6657 - val_loss: 1.0440
Epoch 5/5
50/50 ━━━━━━━━━━━━━━━━━━━━ 202s 3s/step - accuracy: 0.4733 - loss: 1.3407 - val_accuracy: 0.6617 - val_loss: 1.0246
```

The training time was 956 seconds.

```
Model: "sequential_9"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ conv2d_18 (Conv2D)                   │ (None, 94, 94, 32)          │             896 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_18 (MaxPooling2D)      │ (None, 47, 47, 32)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_19 (Conv2D)                   │ (None, 45, 45, 64)          │          18,496 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_19 (MaxPooling2D)      │ (None, 22, 22, 64)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten_9 (Flatten)                  │ (None, 30976)               │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_18 (Dense)                     │ (None, 128)                 │       3,965,056 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_19 (Dense)                     │ (None, 5)                   │             645 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 11,955,281 (45.61 MB)
 Trainable params: 3,985,093 (15.20 MB)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 7,970,188 (30.40 MB)
None
```

There are 896 weights and biases learned by the convolutional layer, the height and width





```
              precision    recall  f1-score   support

           0       0.03      0.02      0.02       133
           1       0.09      0.13      0.10       275
           2       0.14      0.22      0.17       406
           3       0.67      0.58      0.62      1896
           4       0.00      0.00      0.00        39

    accuracy                           0.44      2749
   macro avg       0.19      0.19      0.18      2749
weighted avg       0.50      0.44      0.47      2749

[[   2   14   29   88    0]
 [   7   36   63  169    0]
 [   9   63   89  245    0]
 [  39  300  463 1094    0]
 [   2    3    8   26    0]]
F1 Score: 0.46523738204074877
Accuracy: 0.4441615132775555
```

After increasing the image resolution to 96x96 the F1 score and accuracy slightly dropped and the computation time was higher. Class 0 was correctly identified as such two times. The validation loss slightly decreased and the validation accuracy slightly increased.

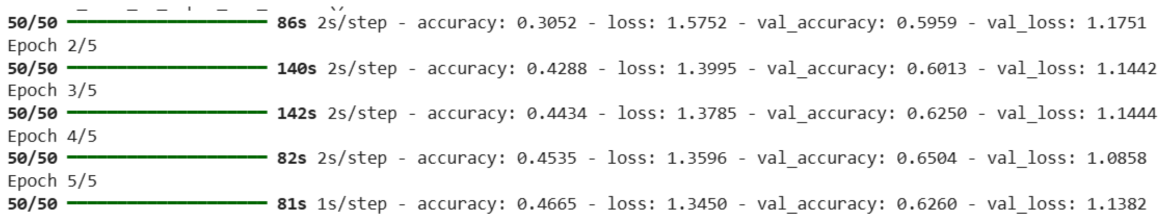
I added a dropout layer with parameter of 0,5 value and decreased the batch size to 64. I also increased the number of epochs to 10.

These are the hyperparameters I chose for increasing validation accuracy, decreasing the loss while also keeping time computation not so high:

-two convolutional layers - Conv2D with 32 filters, kernel size 3x3, ReLU activation andConv2D (64 filters, kernel size 3x3, ReLU activation)
-two maxpooling 2d layers sized 2x2 after each convolutional layer
-a dense layer with 128 units
-a dropout layer of 0,5 rate
-output dense layer of 5 units and softmax activation
-adam optimizer with default parameters
- Sparse Categorical Crossentropy loss function
-64 batch size
-10 epochs

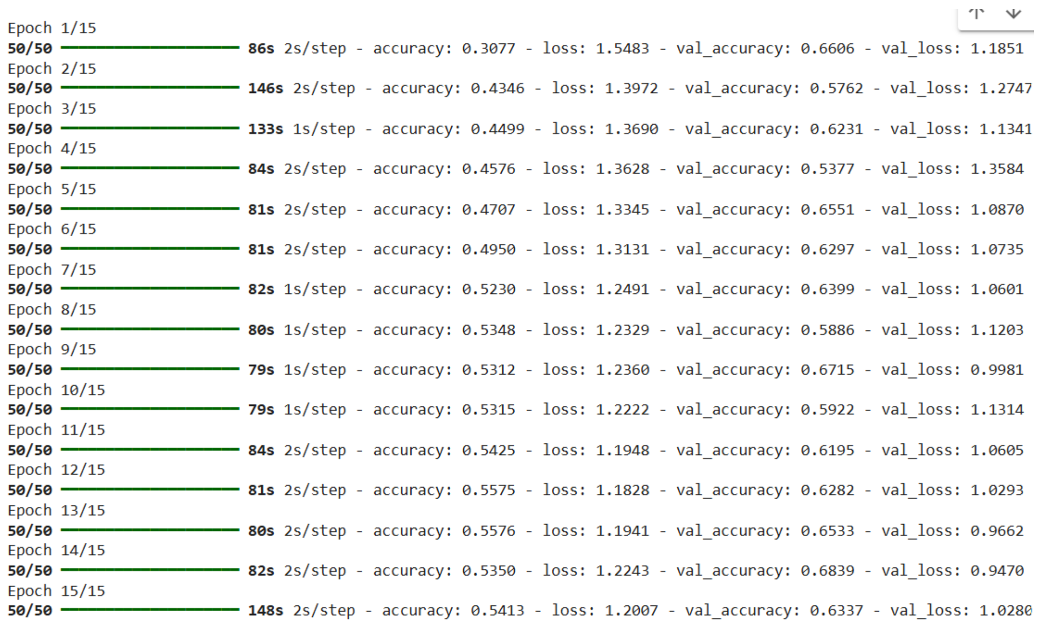
Second model:

In order to use a different approach to solve the problem, i modified data preprocessing to involve data augmentation. I applied data augmentation to the training set which involves transforming the images in order to expand the training dataset, for example rotating the image randomly, shifting the image horizontally or vertically, zooming or flipping the image horizontally. I decreased the image resolution back to 64x64.



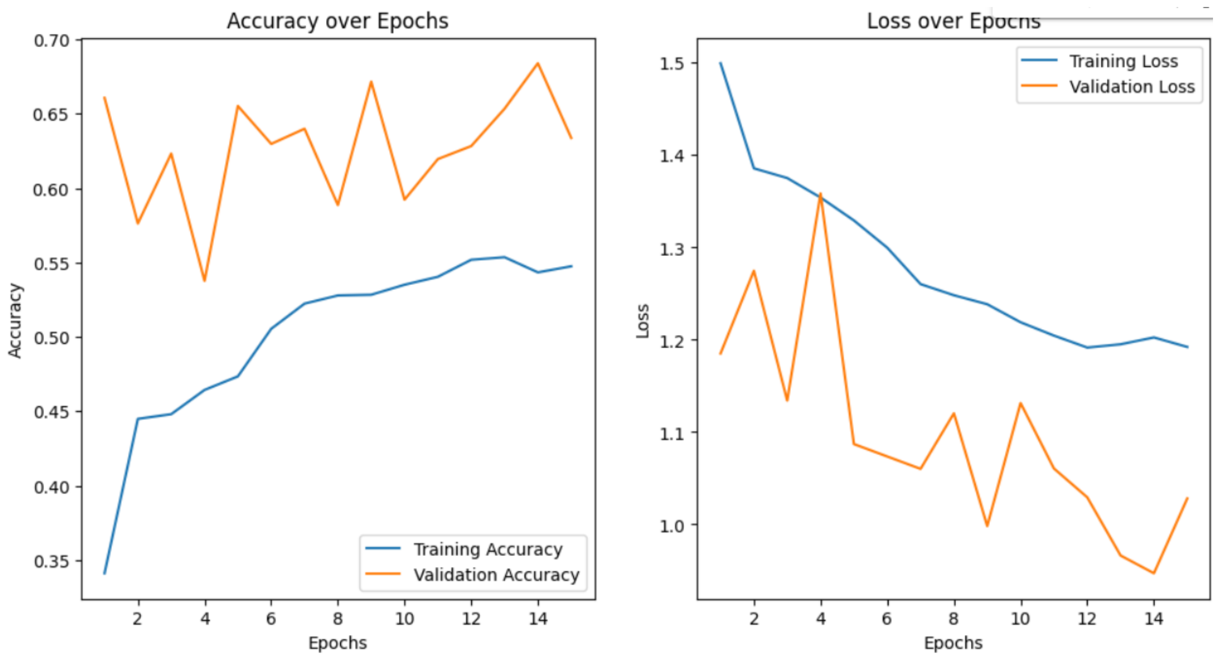
```
50/50 ━━━━━━━━━━━━━━━━━━━━ 86s 2s/step - accuracy: 0.3052 - loss: 1.5752 - val_accuracy: 0.5959 - val_loss: 1.1751
Epoch 2/5
50/50 ━━━━━━━━━━━━━━━━━━━━ 140s 2s/step - accuracy: 0.4288 - loss: 1.3995 - val_accuracy: 0.6013 - val_loss: 1.1442
Epoch 3/5
50/50 ━━━━━━━━━━━━━━━━━━━━ 142s 2s/step - accuracy: 0.4434 - loss: 1.3785 - val_accuracy: 0.6250 - val_loss: 1.1444
Epoch 4/5
50/50 ━━━━━━━━━━━━━━━━━━━━ 82s 2s/step - accuracy: 0.4535 - loss: 1.3596 - val_accuracy: 0.6504 - val_loss: 1.0858
Epoch 5/5
50/50 ━━━━━━━━━━━━━━━━━━━━ 81s 1s/step - accuracy: 0.4665 - loss: 1.3450 - val_accuracy: 0.6260 - val_loss: 1.1382
```

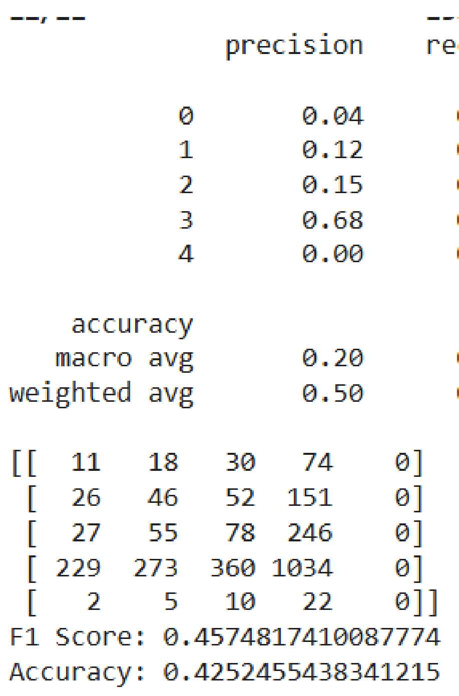
The computation time for 5 epochs was 531 seconds, I set the epochs amount to 15 to see how the accuracy further improves:



```
Epoch 1/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 86s 2s/step - accuracy: 0.3077 - loss: 1.5483 - val_accuracy: 0.6606 - val_loss: 1.1851
Epoch 2/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 146s 2s/step - accuracy: 0.4346 - loss: 1.3972 - val_accuracy: 0.5762 - val_loss: 1.2747
Epoch 3/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 133s 1s/step - accuracy: 0.4499 - loss: 1.3690 - val_accuracy: 0.6231 - val_loss: 1.1341
Epoch 4/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 84s 2s/step - accuracy: 0.4576 - loss: 1.3628 - val_accuracy: 0.5377 - val_loss: 1.3584
Epoch 5/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 81s 2s/step - accuracy: 0.4707 - loss: 1.3345 - val_accuracy: 0.6551 - val_loss: 1.0870
Epoch 6/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 81s 2s/step - accuracy: 0.4950 - loss: 1.3131 - val_accuracy: 0.6297 - val_loss: 1.0735
Epoch 7/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 82s 1s/step - accuracy: 0.5230 - loss: 1.2491 - val_accuracy: 0.6399 - val_loss: 1.0601
Epoch 8/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 80s 1s/step - accuracy: 0.5348 - loss: 1.2329 - val_accuracy: 0.5886 - val_loss: 1.1203
Epoch 9/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 79s 1s/step - accuracy: 0.5312 - loss: 1.2360 - val_accuracy: 0.6715 - val_loss: 0.9981
Epoch 10/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 79s 1s/step - accuracy: 0.5315 - loss: 1.2222 - val_accuracy: 0.5922 - val_loss: 1.1314
Epoch 11/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 84s 2s/step - accuracy: 0.5425 - loss: 1.1948 - val_accuracy: 0.6195 - val_loss: 1.0605
Epoch 12/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 81s 2s/step - accuracy: 0.5575 - loss: 1.1828 - val_accuracy: 0.6282 - val_loss: 1.0293
Epoch 13/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 80s 2s/step - accuracy: 0.5576 - loss: 1.1941 - val_accuracy: 0.6533 - val_loss: 0.9662
Epoch 14/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 82s 2s/step - accuracy: 0.5350 - loss: 1.2243 - val_accuracy: 0.6839 - val_loss: 0.9470
Epoch 15/15
50/50 ━━━━━━━━━━━━━━━━━━━━ 148s 2s/step - accuracy: 0.5413 - loss: 1.2007 - val_accuracy: 0.6337 - val_loss: 1.0280
```

The computation time was 1408 seconds for 15 epochs

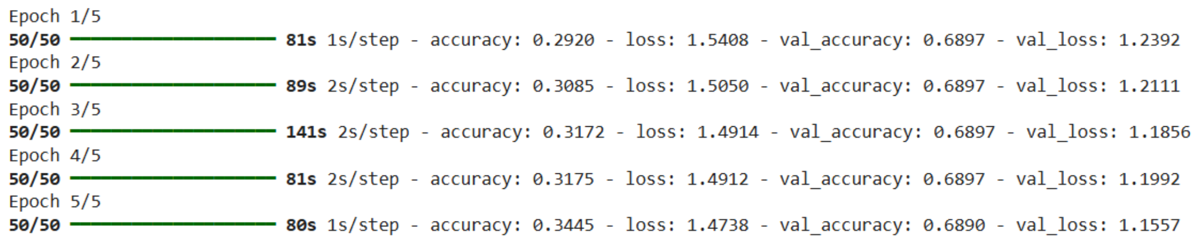



```
              precision    recall  f1-score   support

           0       0.04      0.08      0.05       133
           1       0.12      0.17      0.14       275
           2       0.15      0.19      0.17       406
           3       0.68      0.55      0.60      1896
           4       0.00      0.00      0.00        39

    accuracy                           0.43      2749
   macro avg       0.20      0.20      0.19      2749
weighted avg       0.50      0.43      0.46      2749

[[  11   18   30   74    0]
 [  26   46   52  151    0]
 [  27   55   78  246    0]
 [ 229  273  360 1034    0]
 [   2    5   10   22    0]]
F1 Score: 0.4574817410087774
Accuracy: 0.4252455438341215
```

The validation accuracy seems quite inconsistent across epochs but validation loss generally drops. Training accuracy consistantly increases and training loss consistently drops. It may be due to the large batch size.

Overall, the accuracy didn't increase significantly.

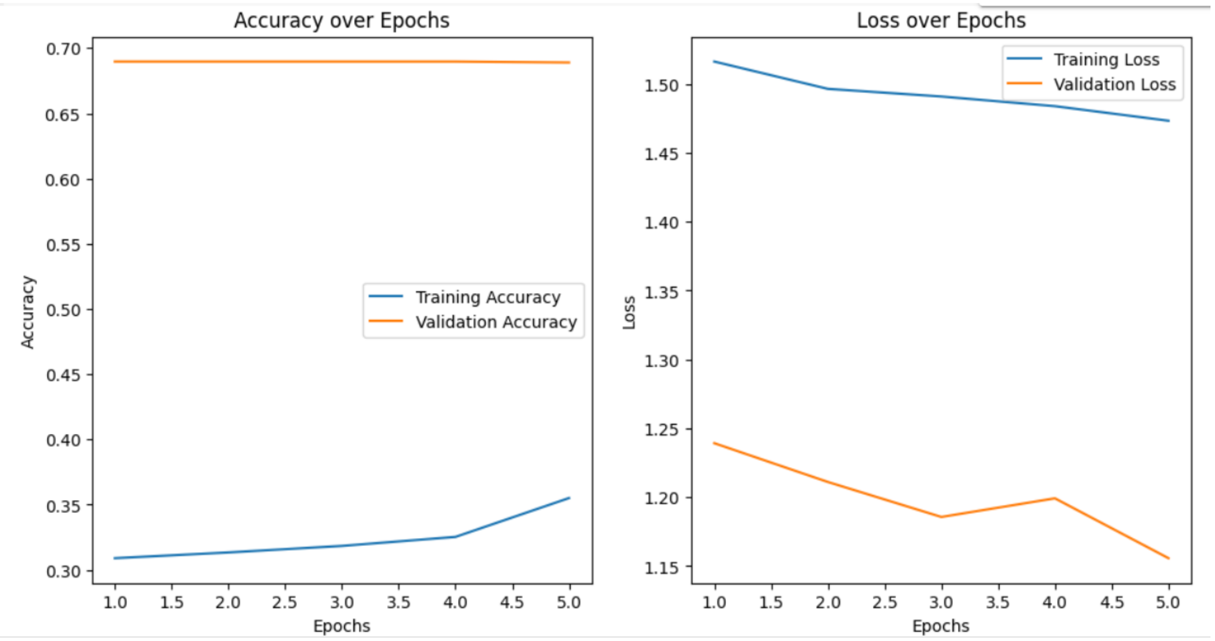
Next I wanted to see how optimizer choice influences model performance, I changed the optimizer from Adam to SGD:


```
Epoch 1/5
50/50 ━━━━━━━━━━━━━━━━━━━━ 81s 1s/step - accuracy: 0.2920 - loss: 1.5408 - val_accuracy: 0.6897 - val_loss: 1.2392
Epoch 2/5
50/50 ━━━━━━━━━━━━━━━━━━━━ 89s 2s/step - accuracy: 0.3085 - loss: 1.5050 - val_accuracy: 0.6897 - val_loss: 1.2111
Epoch 3/5
50/50 ━━━━━━━━━━━━━━━━━━━━ 141s 2s/step - accuracy: 0.3172 - loss: 1.4914 - val_accuracy: 0.6897 - val_loss: 1.1856
Epoch 4/5
50/50 ━━━━━━━━━━━━━━━━━━━━ 81s 2s/step - accuracy: 0.3175 - loss: 1.4912 - val_accuracy: 0.6897 - val_loss: 1.1992
Epoch 5/5
50/50 ━━━━━━━━━━━━━━━━━━━━ 80s 1s/step - accuracy: 0.3445 - loss: 1.4738 - val_accuracy: 0.6890 - val_loss: 1.1557
```

The SGD optimizer performed worse than the Adam optimizer in regard to loss value and accuracy value.

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       133
           1       0.00      0.00      0.00       275
           2       0.17      0.00      0.01       406
           3       0.69      1.00      0.82      1896
           4       0.00      0.00      0.00        39

    accuracy                           0.69      2749
   macro avg       0.17      0.20      0.17      2749
weighted avg       0.50      0.69      0.56      2749

[[   0    0    0  133    0]
 [   0    0    3  272    0]
 [   0    0    2  404    0]
 [   0    0    7 1889    0]
 [   0    0    0   39    0]]
F1 Score: 0.5638365273231428
Accuracy: 0.6878865041833394
```

The confusion matrix showed high inconsistencies as most of the images were identified as class 3. The reason might be that class 3 has the most images in the dataset, so data augmentation increased the number of images even more. Changing the optimizer contributed significantly to missclassification of the images.

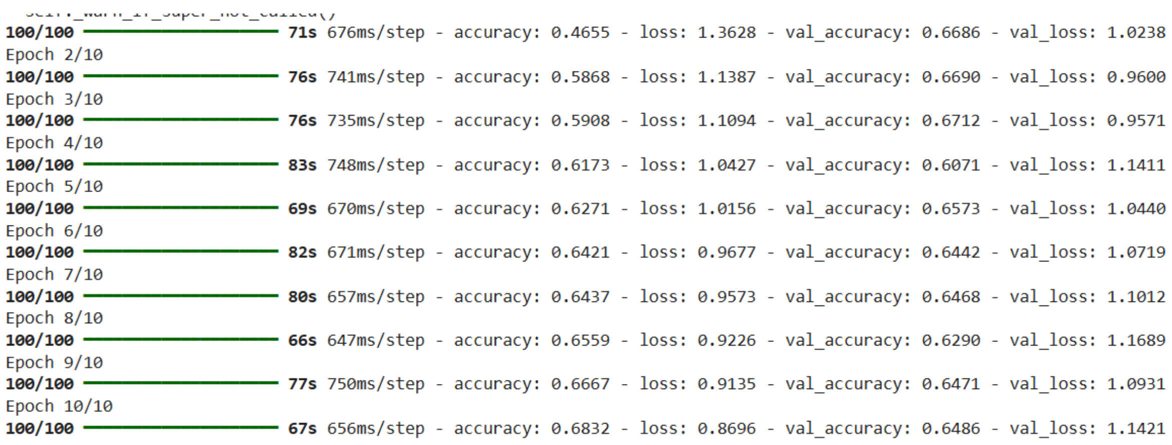


I adjusted the rest of the hyperparameters the same to how I did in model 1 in order to compare how data augmentation and changing the optimizer to SGD influences the performance of the model.

Comparison of model 1 and model 2:

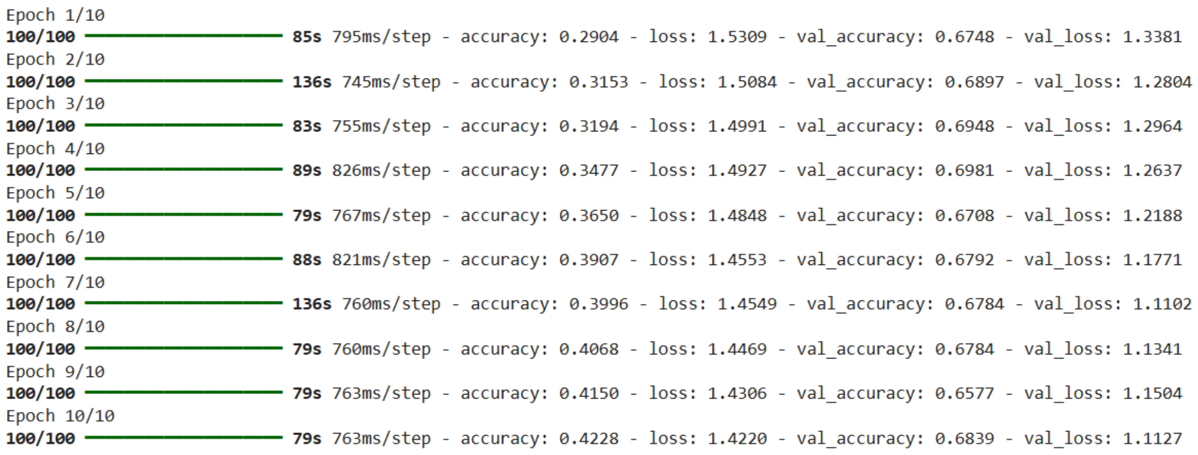
The training runtime for model 1 was 747 seconds, and training runtime for model 2 was 933 seconds, so model 1 was faster.

For model 1:

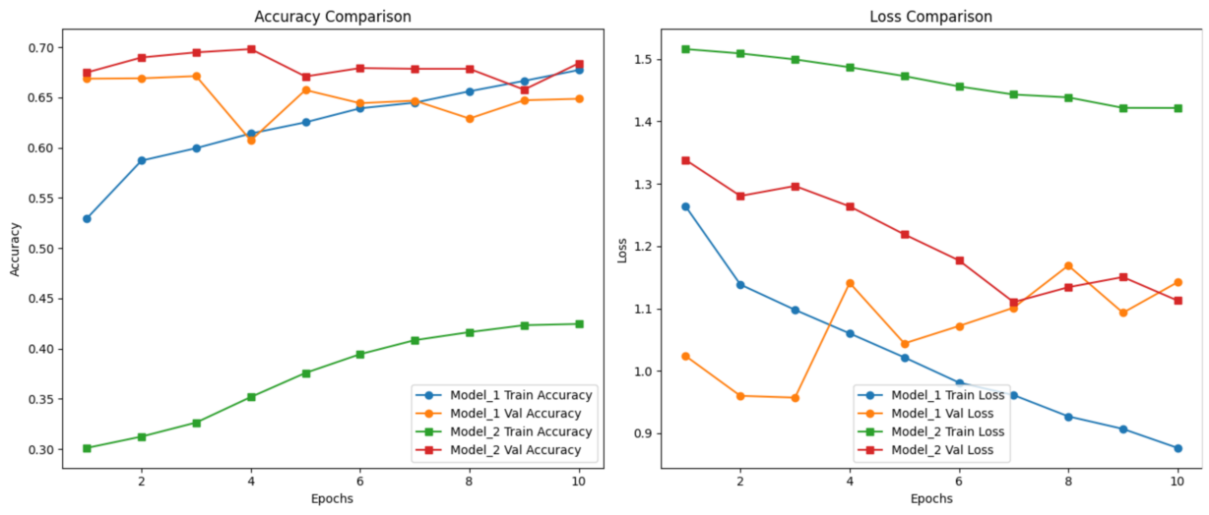


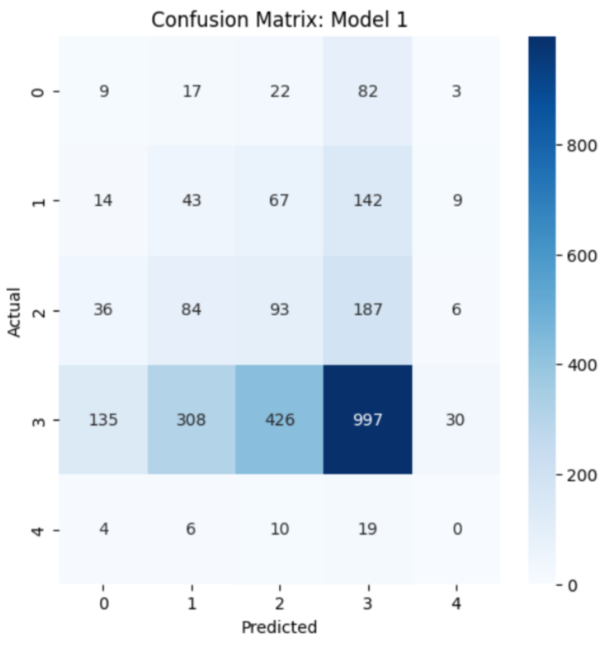
```
  self._warn_if_super_not_called()
100/100 ━━━━━━━━━━━━━━━━━━━━ 71s 676ms/step - accuracy: 0.4655 - loss: 1.3628 - val_accuracy: 0.6686 - val_loss: 1.0238
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 76s 741ms/step - accuracy: 0.5868 - loss: 1.1387 - val_accuracy: 0.6690 - val_loss: 0.9600
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 76s 735ms/step - accuracy: 0.5908 - loss: 1.1094 - val_accuracy: 0.6712 - val_loss: 0.9571
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 83s 748ms/step - accuracy: 0.6173 - loss: 1.0427 - val_accuracy: 0.6071 - val_loss: 1.1411
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 69s 670ms/step - accuracy: 0.6271 - loss: 1.0156 - val_accuracy: 0.6573 - val_loss: 1.0440
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 82s 671ms/step - accuracy: 0.6421 - loss: 0.9677 - val_accuracy: 0.6442 - val_loss: 1.0719
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 80s 657ms/step - accuracy: 0.6437 - loss: 0.9573 - val_accuracy: 0.6468 - val_loss: 1.1012
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 66s 647ms/step - accuracy: 0.6559 - loss: 0.9226 - val_accuracy: 0.6290 - val_loss: 1.1689
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 77s 750ms/step - accuracy: 0.6667 - loss: 0.9135 - val_accuracy: 0.6471 - val_loss: 1.0931
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 67s 656ms/step - accuracy: 0.6832 - loss: 0.8696 - val_accuracy: 0.6486 - val_loss: 1.1421
```

For model 2:

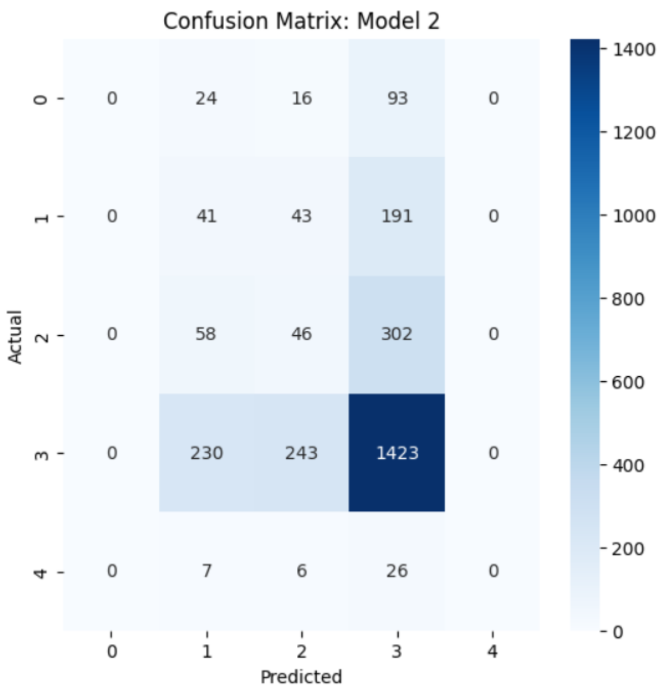


```
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 85s 795ms/step - accuracy: 0.2904 - loss: 1.5309 - val_accuracy: 0.6748 - val_loss: 1.3381
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 136s 745ms/step - accuracy: 0.3153 - loss: 1.5084 - val_accuracy: 0.6897 - val_loss: 1.2804
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 83s 755ms/step - accuracy: 0.3194 - loss: 1.4991 - val_accuracy: 0.6948 - val_loss: 1.2964
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 89s 826ms/step - accuracy: 0.3477 - loss: 1.4927 - val_accuracy: 0.6981 - val_loss: 1.2637
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 79s 767ms/step - accuracy: 0.3650 - loss: 1.4848 - val_accuracy: 0.6708 - val_loss: 1.2188
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 88s 821ms/step - accuracy: 0.3907 - loss: 1.4553 - val_accuracy: 0.6792 - val_loss: 1.1771
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 136s 760ms/step - accuracy: 0.3996 - loss: 1.4549 - val_accuracy: 0.6784 - val_loss: 1.1102
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 79s 760ms/step - accuracy: 0.4068 - loss: 1.4469 - val_accuracy: 0.6784 - val_loss: 1.1341
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 79s 763ms/step - accuracy: 0.4150 - loss: 1.4306 - val_accuracy: 0.6577 - val_loss: 1.1504
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 79s 763ms/step - accuracy: 0.4228 - loss: 1.4220 - val_accuracy: 0.6839 - val_loss: 1.1127
```

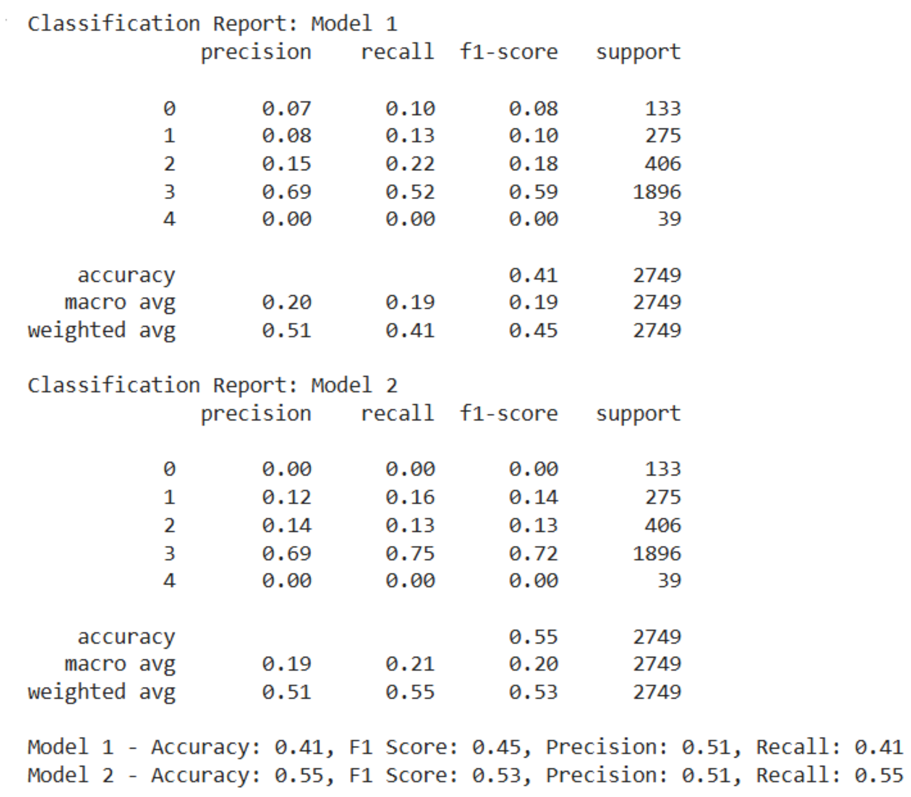




$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 9 & 17 & 22 & 82 & 3 \\ 1 & 14 & 43 & 67 & 142 & 9 \\ 2 & 36 & 84 & 93 & 187 & 6 \\ 3 & 135 & 308 & 426 & 997 & 30 \\ 4 & 4 & 6 & 10 & 19 & 0 \end{array}$$



$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 0 & 24 & 16 & 93 & 0 \\ 1 & 0 & 41 & 43 & 191 & 0 \\ 2 & 0 & 58 & 46 & 302 & 0 \\ 3 & 0 & 230 & 243 & 1423 & 0 \\ 4 & 0 & 7 & 6 & 26 & 0 \end{array}$$



```
Classification Report: Model 1
              precision    recall  f1-score   support

           0       0.07      0.10      0.08       133
           1       0.08      0.13      0.10       275
           2       0.15      0.22      0.18       406
           3       0.69      0.52      0.59      1896
           4       0.00      0.00      0.00        39

    accuracy                           0.41      2749
   macro avg       0.20      0.19      0.19      2749
weighted avg       0.51      0.41      0.45      2749

Classification Report: Model 2
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       133
           1       0.12      0.16      0.14       275
           2       0.14      0.13      0.13       406
           3       0.69      0.75      0.72      1896
           4       0.00      0.00      0.00        39

    accuracy                           0.55      2749
   macro avg       0.19      0.21      0.20      2749
weighted avg       0.51      0.55      0.53      2749

Model 1 - Accuracy: 0.41, F1 Score: 0.45, Precision: 0.51, Recall: 0.41
Model 2 - Accuracy: 0.55, F1 Score: 0.53, Precision: 0.51, Recall: 0.55
```

Analyzing the confusion matrices, for model 1 predictions are more distributed among the classes – model 1 predicts various classes but performs poorly for most besides the largest class 3. For the smallest class 4 there is no true positive.

Model 2 mostly predicts class 3. Other classes aren't represented so well in predictions and class 0 and 4 have no true positives. Model 2 has much better results for the largest class – 3, but it also lacks balance in the predictions. Due to model 2 mostly prefering class 3 and performing better for this class the overall performance metrics are higher.

While analyzing performance metrics, we can see that model 2 has higher accuracy (55% compared to model 1's 41%). The accuracy measure may be highly influenced by class 3 which constitutes the majority of the test dataset. Since most images belong to Class 3, correctly classifying these improves the accuracy metric. Other metrics besides precision are also higher for model 2.

Model 2 performs better on metrics since it focuses on the dominant class 3 which can distort overall results in an imbalanced dataset, so the metrics might not be exactly believable to decide which model performed better.

Conclusions

Training the implemented CNN model was very time-consuming and it was necessary to tune the hyperparameters in order to achieve efficient time of computation. Unfortunately it also came with poor performance of the model, reducing the accuracy and increasing the loss. In case of these models, lowering the resolution of the images didn't lower the performance of this model significantly. Increasing the batch size was the biggest contributor for decreasing the computation time.

After implementing data augmentation to the second approach, it didn't significantly change the performance of the model or the training time, but looking at the confusion matrixes, it wrongly classified most of the samples to class 3. After changing the optimizer to SGD, the accuracy decreased and loss value increased, so the Adam optimizer performs better.

In general, I would say the first implemented model performer better, due to better performance of the Adam optimizer in contrast to SGD optimizer, and also due to not implementing data augmentation.

Looking at the confusion matrixes, the correctness of sample classification was more balanced in the first implemented model, although it still didn't classify any of the images to the 4th class.

The validation accuracy and validation loss values are generally not the best for both models, which might be caused by hypertuning the parameters and changing the architecture of the model in order to decrease computation time.

In order to improve these models, it would be beneficial to change the hyperparameters like batch size or change the architecture of the model, add more epochs and early stopping, adjust the learning rate during training.

An improvement to the model would be to find an optimal way to reduce the training time without worsening the performance so much. Increasing the number of epochs would allow to see whether the model further improves.

While comparing the confusion matrices I also noticed the influence of an unbalanced dataset in performance of the models. It could be beneficial to Focus on improving the performance for classes which are represented less in the dataset, maybe trying to undersample the most numerous class, or oversample the minority classes, or apply data augmentation only for smaller classes, or assign higher weights for example to the smallest classes.