Name: Jashwanth Neeli

Assignment 7

CSC 578

Kaggle User Name: Jashwanth Neeli

Best Model: (Kaggle score: 0.90277(pulic score), 0.92491(private score))

In this assignment I build various CNN models by adjusting the hypermeters of the cnns and every model was trained by 3 epochs my best model turned out to be the one where I changed the filter size, added padding and added one additional hidden layer and also the total number of parameters can be seen as shown in the below image. I changed the filter size and the number of filters, so in my first itself without using any hyperparameters my model came out to be the best version among all my models.

```
Total params: 2,809,734
Trainable params: 2,809,734
Non-trainable params: 0
```

The train loss, validation loss, train acc and test acc for my model8-version1 at my last epoch was

```
Epoch 3/3
350/350 [==============================] - 7s 21ms/step - loss: 0.5486
- accuracy: 0.8014 - val_loss: 0.5494 - val_accuracy: 0.8044
```

In my Kaggle competition, the list of parameters that I used for my models

- number of filters.
- size of filters (3*3, 5*5).
- number of Convolution layers.
- number and size of Fully Connected layers.
- use of Dropout layers and the percentages.
- use of regularization.
- use of BatchNormalization layers.

Hyperparameters related to compilation include:

- learning rate
- regularization
- drop-out percent

Model 00:
Model Architecture:
Input Layer:
Rescaling layer: Scales input pixel values to a range of [0,1] by dividing them by 255.
Convolutional Layers:
Conv2D with 16 filters, each of size (3,3), and ReLU activation.
MaxPooling2D layer with a pool size of (2,2) to downsample the spatial dimensions.
Conv2D with 32 filters, each of size (3,3), and ReLU activation.
MaxPooling2D layer with a pool size of (2,2) for further downsampling.
Flatten Layer:
Flattens the output from the convolutional layers into a 1D array.
Dense Layers:
Dense layer with 128 neurons and ReLU activation.
Dense layer with 6 neurons (assuming it's a classification task with 6 classes) and softmax activa tion for multi-class classification.
Hyperparameters Experimented :
Initial model, used from professor model.

```
loss: 0.6597 - accuracy: 0.7602 - val_loss: 0.7105 - val_accuracy: 0.7319
```

Model 1:
Model Architecture:
This model has an Conv2D layer with 16 filters and ReLU activation after the first max-pooling la yer.
Hyperparameters Experimented:
I did not experiment with any hyperparameters in this model, because I wanted to see how will the CNN perform by changing the number of filters in the cnn.
Expectation:
Changing the number of filters in model 01 Smaller filter sizes generally require fewer paramete rs, leading to a more computationally efficient model. Larger filter sizes result in more paramet ers and can increase the computational cost. However, they may provide a richer representatio n of the input.
Outcome:

```
loss: 0.4866 - accuracy: 0.8266 - val_loss: 0.7610 - val_accuracy: 0.7479
```

Model 2:
Model Architecture:
This model has an Conv2D layer with 32 filters and ReLU activation after the first max-pooling la yer.
Hyperparameters Experimented:
I did not experiment with any hyperparameters in this model, because I wanted to see how will the CNN perform by changing the number of filters in the cnn.
Expectation:
Changing the number of filters in model 2, Smaller filter sizes generally require fewer paramete rs, leading to a more computationally efficient model. Larger filter sizes result in more paramet ers and can increase the computational cost. However, they may provide a richer representatio n of the input. In this model I expected that my accuracy will increase by increasing the size of t he filters.
Outcome:
```
loss: 0.6010 - accuracy: 0.7878 - val_loss: 0.7133 - val_accuracy: 0.7504
```

Model 3:
Model Architecture:
This model has an Conv2D layer with 32 filters and filter size 3*3 and 5*5 and ReLU activation a fter the first max-pooling layer.
Hyperparameters Experimented:
The model experiments with a different number of filters (32 in both layers) compared to the pr evious models (model and model1). Additionally, the filter size is kept at (3,3).
Expectation:
Anticipations would involve determining whether the model's capacity to collect hierarchical inf ormation is enhanced by the addition of filters to both convolutional layers. The model's expres siveness and computing efficiency may be impacted by the hyperparameter selection, and perf ormance would be evaluated on a validation set to guarantee efficacy on the particular job.
Outcome:
```
loss: 0.6010 - accuracy: 0.7878 - val_loss: 0.7133 - val_accuracy: 0.7504
```

Model 3:
Model Architecture:
model3 is a CNN with two convolutional layers. The first convolutional layer has 32 filters with a (3,3) kernel size, followed by max-pooling. The second convolutional layer has 64 filters with a l arger (5,5) kernel size, followed by max-pooling. The architecture is then flattened and connect ed to two dense layers.
Hyperparameters Experimented:
The model experiments with a larger kernel size in the second convolutional layer ((5,5)) compa red to the first ((3,3)), potentially allowing the network to capture more global features.
Expectation:

The expectation would be that the larger kernel size in the second convolutional layer enables t he model to learn higher-level and more abstract features. The reported accuracy of 80.44% an d validation accuracy of 78.05% suggest a reasonably performing model, but further analysis on test data is needed to validate its generalization ability.

Outcome:

```
loss: 0.5339 - accuracy: 0.8044 - val_loss: 0.6511 - val_accuracy: 0.7805
```

In [16]:

Model 4:

Model Architecture:

model4 is a CNN with two convolutional layers. The first convolutional layer has 64 filters with a larger (5,5) kernel size, followed by max-pooling. The second convolutional layer also has 64 filt ers with the same (5,5) kernel size, followed by max-pooling. The architecture is then flattened and connected to two dense layers.

Hyperparameters Experimented:

The model maintains a larger kernel size ((5,5)) in both convolutional layers compared to the pr evious models. This suggests an emphasis on capturing more global features in the input.

Expectation:

The larger kernel size in both convolutional layers might allow the model to capture broader pa tterns in the input. The reported accuracy of 76.07% and validation accuracy of 72.18% indicate a decent performance.

Outcome:

```
loss: 0.6437 - accuracy: 0.7607 - val_loss: 0.7747 - val_accuracy: 0.7218In [16]:
```

Model 5:

Model Architecture:

model5 is a CNN with three convolutional layers. The first convolutional layer has 32 filters with a (3,3) kernel size, followed by max-pooling. The second convolutional layer has 64 filters with a larger (5,5) kernel size, followed by max-pooling. The third convolutional layer has 32 filters wit h a (3,3) kernel size, followed by max-pooling. The architecture is then flattened and connected to two dense layers.

Hyperparameters Experimented:

The model introduces an additional convolutional layer with 32 filters and a (3,3) kernel size, ad ding complexity to the feature extraction process.

Expectation:

The expectation is that the additional hidden layer enhances the model's ability to capture hier archical features in the data. The reported accuracy of 76.35% and validation accuracy of 77.58 % indicate a reasonably performing model.

Outcome:

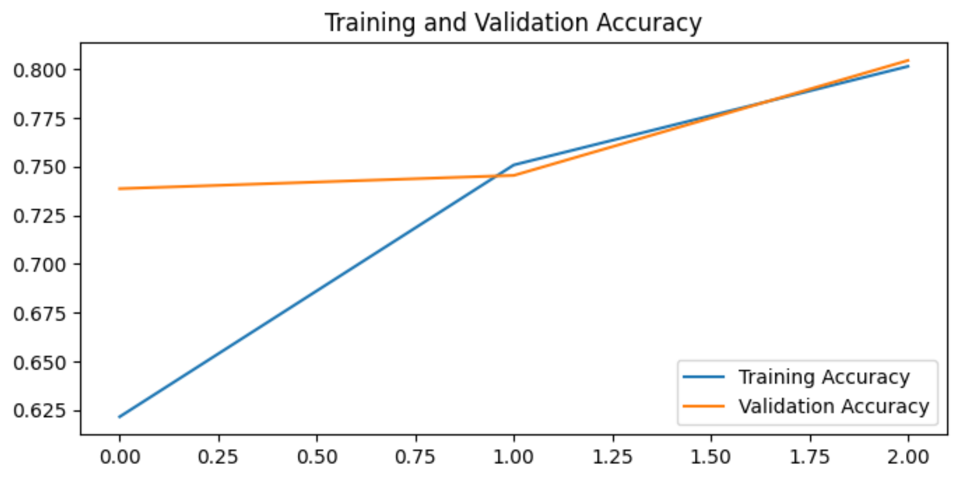
loss: 0.6466 - accuracy: 0.7635 - val_loss: 0.6272 - val_accuracy: 0.7758

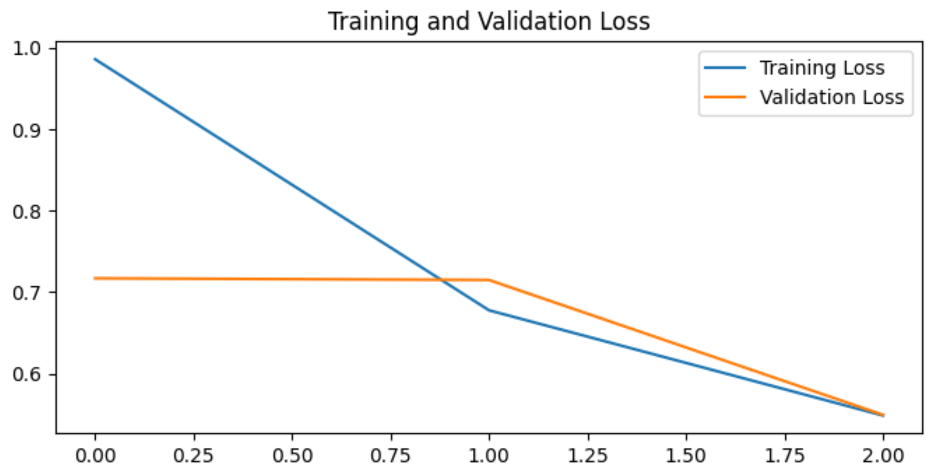
Model 6:
Model Architecture:
model6 is a CNN with three convolutional layers. The first convolutional layer has 32 filters with a (3,3) kernel size, followed by max-pooling. The second convolutional layer has 64 filters with a larger (5,5) kernel size, followed by max-pooling. The third convolutional layer has 64 filters wit h a (3,3) kernel size, followed by max-pooling. The architecture is then flattened and connected to two dense layers.
Hyperparameters Experimented:
The model introduces an additional convolutional layer with 64 filters and a (3,3) kernel size, alt ering the feature extraction process. Additionally, it maintains the larger kernel size in the seco nd convolutional layer.
Expectation:
The expectation is that the changes in kernel size and the addition of another hidden layer impa ct the model's ability to capture hierarchical features. The reported accuracy of 79.97% and vali dation accuracy of 74.94% suggest a reasonably performing model.
Outcome:
```
loss: 0.6992 - accuracy: 0.7997 - val_loss: 0.7358 - val_accuracy: 0.7494
```

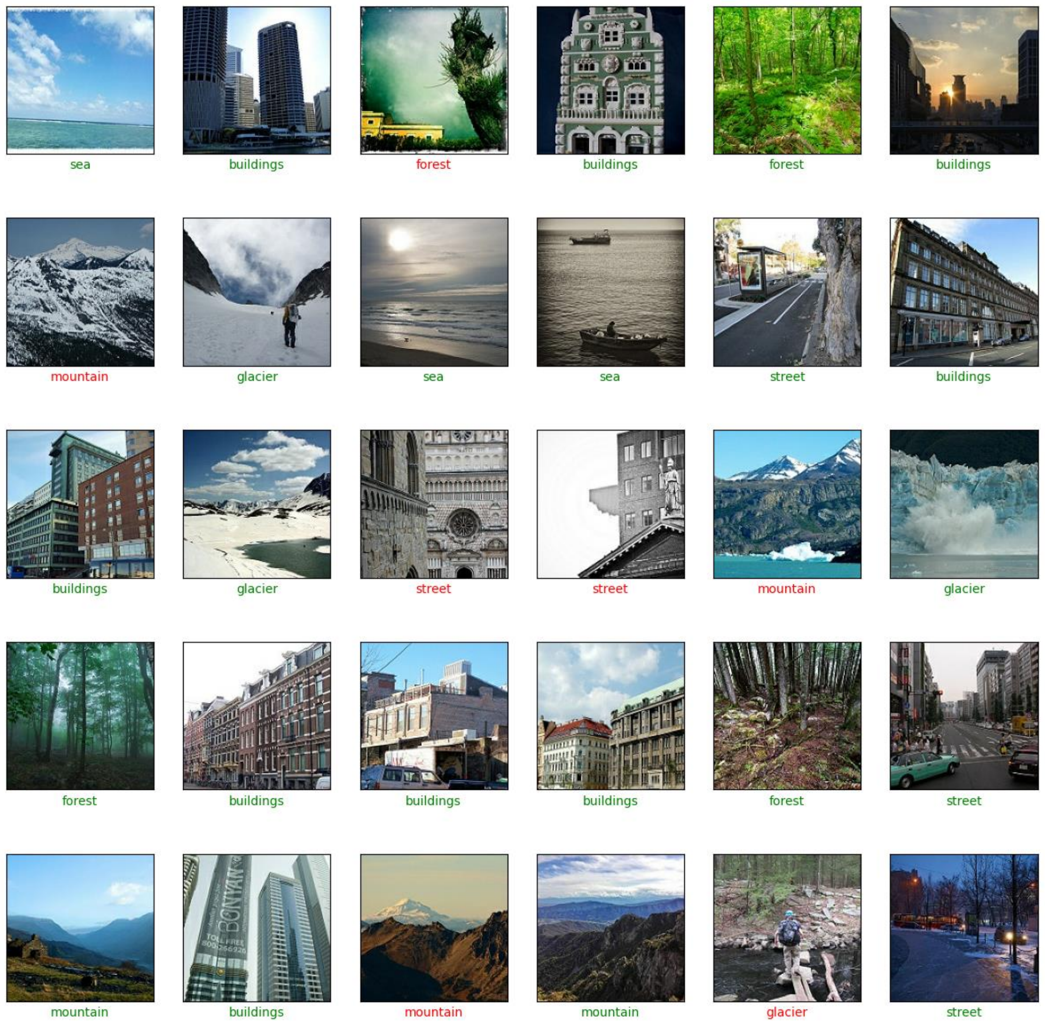
Model 7:
Model Architecture:
model7 is a CNN with three convolutional layers. The first convolutional layer has 32 filters with a (3,3) kernel size, followed by max-pooling. The second convolutional layer has 64 filters with a larger (5,5) kernel size, followed by max-pooling. The third convolutional layer has 64 filters wit h a larger (5,5) kernel size, followed by max-pooling. The architecture is then flattened and con nected to two dense layers.
Hyperparameters Experimented:
The model introduces an additional convolutional layer with 64 filters and a larger (5,5) kernel s ize, modifying the feature extraction process. It maintains the larger kernel size in the second co nvolutional layer.
Expectation:
The expectation is that the changes in kernel size and the addition of another hidden layer impa ct the model's ability to capture hierarchical features. The reported accuracy of 79.17% and vali dation accuracy of 81.66% suggest a reasonably performing model.

Outcome:
```
loss: 0.5713 - accuracy: 0.7917 - val_loss: 0.5377 - val_accuracy: 0.8166.
```

Model 8:
Model Architecture:
model8 is a CNN with three convolutional layers. The first and second convolutional layers have 32 and 64 filters, respectively, both with a (3,3) kernel size and 'same' padding. The third convol utional layer has 64 filters with a (5,5) kernel size and 'same' padding. The architecture includes max-pooling layers, flattening, and two dense layers.
Hyperparameters Experimented:
The model introduces 'same' padding in the convolutional layers, meaning that the spatial dime nsions of the output volume are the same as the input volume. This can be useful to retain mor e information at the borders of the images during convolution.
Expectation:
The addition of 'same' padding aims to mitigate information loss at the image borders. The repo rted accuracy of 80.14% and validation accuracy of 80.44% suggest a reasonably performing mo del. Out of all the models I performed this model gave reduced the loss value and increased the accuracy so I thought I this model would be best for performing predictions. I submitted this pr ediction model in Kaggle competition and I got the best score among all my versions.
Outcome:
loss: 0.5486 - accuracy: 0.8014 - val_loss: 0.5494 - val_accuracy: 0.8044.


Training and Validation Accuracy
0.800
0.775
0.750
0.725
0.700
0.675
0.650
0.625
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Training Accuracy
Validation Accuracy


Training and Validation Loss
1.0
0.9
0.8
0.7
0.6
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Training Loss
Validation Loss



Strides:

Model9:

Expectations: The addition of strides in the max-pooling layers aims to down-sample the spatial dimensions more aggressively.

Result: The reported accuracy of 80.37% and validation accuracy of 81.62% suggest a reasonably performing model, with further analysis on test data needed to assess generalization. The use of strides may contribute to capturing more prominent features. The use of strides also gave a good accuracy and less loss value but my model 8 has good accuracy and over fitting.

Adding hidden layers:

Model10:

Expectations: The expectation with the addition of one more hidden layer (Conv2D and MaxPooling2D) is that the model may be able to capture more complex hierarchical features and patterns in the data, potentially improving its overall performance.

Result: The reported accuracy of 77.35% and validation accuracy of 76.87% suggest a decent performance, but the model may not have significantly outperformed the previous configurations.

Changing filter size and number of filters:

Model 11:

Expectations: Changing the filter sizes in model11 may impact the model's ability to capture different scales of features. The expectation is that this variation might lead to a better representation of the input data, potentially improving performance**.**

Result: The reported accuracy of 76.20% and validation accuracy of 80.69% suggests a reasonably performing model. The change in filter sizes may have contributed to capturing diverse features, leading to improved validation accuracy compared to the training accuracy.

Model 12:

Expectations: With the combination of different filter sizes in model12, the expectation is that the model might be able to capture both fine-grained and more global features, potentially leading to improved performance.

Result: The reported accuracy of 77.81% and validation accuracy of 77.55% suggest a reasonably performing model. The mix of filter sizes may contribute to a diverse representation of features.

Model 13:

Expectations: The use of three convolutional layers with varying filter sizes in model13 suggests an attempt to capture hierarchical features at different scales. The expectation is that this diversity in feature extraction might lead to improved model performance.

Result: The reported accuracy of 79.00% and validation accuracy of 79.08% suggest a reasonably performing model. The combination of filter sizes may contribute to capturing features at different levels of abstraction.

Regularization:

L1L2 – model14:

Expectations: The addition of L1L2 regularization in model14 aims to prevent overfitting by penalizing large weights and encouraging sparse weight matrices. The expectation is that regularization could improve generalization to unseen data.

result: The reported accuracy of 79.02% and validation accuracy of 77.55% suggest a reasonably performing model. The regularization may help in preventing overfitting, but further evaluation on test data is needed to assess its impact on generalization. It's worth tuning the regularization strength for potential performance improvement.

L2 - model15:

Expectations: The addition of L2 regularization in model15 aims to mitigate overfitting by penalizing large weights. The expectation is that regularization could lead to a more robust model with improved generalization.

Results: The reported accuracy of 79.69% and validation accuracy of 78.37% suggest a reasonably performing model. The L2 regularization may have contributed to preventing overfitting, but further evaluation on test data is needed to assess its impact on generalization. Adjusting the regularization strength could be explored for potential performance improvement.

L2 - model16:

Expectations: The addition of L1 regularization in model16 with a relatively high regularization strength (0.01) aims to induce sparsity in the weight matrices, potentially preventing overfitting. The expectation is that regularization might improve generalization.

Results: The reported accuracy of 63.57% and validation accuracy of 63.07% suggest that the high L1 regularization strength might have led to excessive sparsity, affecting model performance negatively. It might be beneficial to try lower regularization strengths and evaluate their impact on model generalization.

Dropout:

Model17:

Expectation: The addition of SpatialDropout2D layers in model17 with a dropout rate of 0.2 aims to regularize the model by randomly dropping out entire 2D feature maps during training. This is expected to prevent overfitting by introducing variability in the feature maps.

Result: The reported accuracy of 72.32% and validation accuracy of 74.08% suggest that the SpatialDropout2D layers might have contributed to regularization, improving model generalization. The dropout layers help prevent overfitting by introducing randomness during training, which appears to have a positive impact on the validation accuracy.

Model18:

Expectation: The addition of both L2 regularization in convolutional layers and dropout layers in model18, including the dense layers, aims to provide comprehensive regularization to prevent overfitting. This should help improve generalization and robustness.

Outcome: The reported accuracy of 70.52% and validation accuracy of 74.08% suggest that the combination of L2 regularization and dropout layers might have contributed to regularization. However, there is room for further improvement. Fine-tuning the dropout rates and regularization strengths could be explored for better results.

Model19:

Expectation: The removal of dropout layers in some convolutional layers in model19 might reduce the regularization effect, potentially allowing the model to capture more intricate patterns in the data. This could lead to a slightly higher training accuracy.

Outcome: The reported accuracy of 74.41% and validation accuracy of 74.51% suggest that removing dropout from some layers did not significantly impact the performance. The model seems to have achieved reasonable accuracy, but further experimentation with dropout rates and regularization strengths could be explored for optimization.

Batch normalization:

Model20:

Expectation: The addition of Batch Normalization (BatchNormalization()) is expected to enhance the training stability and accelerate convergence by normalizing the inputs to each layer.

Outcome: The reported validation accuracy of 73.33% suggests that while Batch Normalization might have improved training stability, it did not necessarily result in higher validation accuracy. Further tuning of Batch Normalization parameters or exploring different architectures could be considered for improvement.

Model21:
Expectation: The addition of strides in the convolutional layers with Batch Normalization (BatchNormalization()) is expected to allow the model to capture spatial features more efficiently, potentially leading to improved performance.

Outcome: The reported validation accuracy of 76.15% indicates that the addition of strides in convolutional layers might have positively influenced the model's ability to capture relevant spatial features, contributing to an improvement in performance.

Model22:

Expectation: The addition of SpatialDropout combined with Batch Normalization (SpatialDropout2D(0.2) and BatchNormalization()) is expected to introduce regularization, potentially preventing overfitting and improving the model's generalization on the validation set.

Outcome: The reported validation accuracy of 68.32% suggests that the addition of SpatialDropout and Batch Normalization might not have provided a significant improvement compared to the previous model (Model21).

Optimizers:

Adamax:

Model23:

Expectation: Changing the optimizer to Adamax is expected to impact the optimization process. Adamax is a variant of Adam that may exhibit different convergence behavior, potentially affecting training speed and, consequently, the final model performance.

Outcome: The validation accuracy of 75.72% is slightly lower than the previous model (Model22). Further fine-tuning of hyperparameters or exploring different optimizers might be considered to enhance model performance.

Adamax using batch normalization:

Model24:

Expectation: Adding Batch Normalization might improve the training stability and accelerate convergence by normalizing the input to each layer.

Outcome: The validation accuracy of 76.40% is an improvement compared to the previous model (Model23). Batch Normalization seems to have a positive impact on model performance.

Adagrad:

Model25:

Expectation: Adagrad optimizers adapt the learning rates for each parameter individually, potentially performing well on sparse data.

Outcome: The performance of the model with Adagrad optimizer is suboptimal, with a validation accuracy of 63.75%, which is lower compared to the previous model (Model24). It suggests that Adagrad may not be the best choice for this specific task or model architecture.

Adagrad using batch normalization:

Model26:

Expectation: Combining Batch Normalization with Adagrad might help stabilize training and improve overall performance.

Outcome: The model (Model26) with Batch Normalization and Adagrad optimizer performs well, achieving a validation accuracy of 78.26%, which is an improvement compared to the previous model (Model25) with just the Adagrad optimizer. Batch Normalization helps in stabilizing and accelerating the training process, and Adagrad adapts the learning rates for each parameter, contributing to better optimization.

Adadelta:

Model27

Expectation: Adadelta, being an adaptive learning rate optimizer, might perform differently compared to Adagrad. It's expected to contribute to stable convergence.

Outcome: The model (Model27) with Adadelta optimizer shows lower performance compared to the previous models, achieving an accuracy of 52.16%. It seems that the choice of optimizer significantly influences the training dynamics and final performance, and Adadelta might not be as suitable for this specific task as Adagrad.

Adadelta using batch normalization:

Model28:

Expectation: Adding Batch Normalization along with Adadelta may contribute to stabilizing training and improving convergence, but the overall performance might not be significantly better than Adadelta alone.

Outcome: The model (Model28) with Batch Normalization and Adadelta achieves an accuracy of 61.14%, which is an improvement compared to the model with Adadelta alone (Model27). Batch Normalization likely played a role in stabilizing and accelerating the training process.

Data augmentation:

Model29:

Expectation: Data augmentation is expected to enhance the model's generalization ability by training on diverse variations of the input images. This may lead to better performance, especially on the validation set.

Outcome: The model (Model29) with data augmentation shows an accuracy of 68.39%, which is slightly lower than the previous model (Model28) without data augmentation. It's important to note that data augmentation impact can vary, and in this case, the introduced variations might not have universally improved performance. Adjusting the augmentation parameters or trying different augmentation techniques could be explored for further optimization.

Model	Regularization	Dropout	Batch Normalization	Optimizer	Data Augmentation	Training Accuracy	Validation Accuracy	Traini Loss
Model13	None	None	None	Adam	No	79.00%	79.08%	0.5714
Model14	L1L2	None	None	Adam	No	79.02%	77.55%	0.5875
Model15	L2	None	None	Adam	No	79.69%	78.37%	0.5585
Model16	L1	None	None	Adam	No	63.57%	63.07%	1.2826
Model17	L2	None	Dropout	Adam	No	72.32%	74.08%	0.7557
Model18	L2	20% (Spatial)	20% (Dense)	Adam	No	70.52%	74.08%	0.7950
Model19	L2	20% (Spatial)	20% (Dense)	Adam	No	74.41%	74.51%	0.6924

Model20	L2	None	Batch Normalization	Adam	No	78.77%	73.33%	0.5992
Model21	L2	None	Batch Normalization	Adam	No	81.49%	76.15%	0.5251
Model22	L2	20% (Spatial)	Batch Normalization	Adam	No	67.72%	68.32%	0.8546
Model23	L2	None	None	Adamax	No	79.37%	75.72%	0.5939
Model24	L2	None	Batch Normalization	Adamax	No	85.08%	76.40%	0.4296
Model25	L2	None	None	Adagrad	No	61.59%	63.75%	1.0229
Model26	L2	20% (Spatial)	Batch Normalization	Adagrad	No	88.99%	78.26%	0.3459
Model27	L2	None	None	Adadelta	No	50.57%	52.16%	1.5810

Model28	L2	None	Batch Normalization	Adadelta	No	60.27%	61.14%	1.0796
Model29	L2	None	Batch Normalization	Adagrad	Yes	72.60%	68.39%	0.7411