COMP7250 Machine Learning Individual Project
Topic: Data Augmentation for Image Classification Tasks

ZHU Qinghui 23427140

1. Introduction

1.1 Problem Definition

Image classification tasks often face challenges related to limited training data and the risk of overfitting, where the model learns to memorize training samples rather than generalize to new, unseen images. Data augmentation offers a promising solution to address these challenges by expanding the training dataset without collecting additional images. By applying transformations to the existing images, data augmentation introduces variations that enable the model to learn invariant features and improve its ability to recognize patterns in diverse contexts.

The primary problem this project aims to tackle is twofold: first, to assess the effectiveness of data augmentation in boosting the training performance of image classification models, and second, to investigate its role in reducing overfitting tendencies. By systematically designing experiments and comparing the performance of CNN models with and without data augmentation, we seek to quantify the impact of augmentation techniques on key metrics such as accuracy, convergence speed, and generalization capability. Through this analysis, we aim to gain a deeper understanding of how data augmentation can be leveraged to enhance the robustness and reliability of image classification systems.

1.2 Main Jobs

In the realm of computer vision and image classification, the effectiveness of data augmentation techniques has garnered significant attention due to their potential to enhance model performance and mitigate overfitting issues. Data augmentation involves creating new training samples from existing data by applying various transformations, such as rotations, flips, and scaling. By introducing diversity in the training set, data augmentation aims to improve the generalization capability of machine learning models, particularly convolutional neural networks (CNNs), which are widely used for image classification tasks.[1]

This report delves into the exploration of data augmentation techniques (ImageDataGenerator from Keras) for image classification tasks. The primary objective of this project is to leverage established CNN models and open datasets (Cifar-10) aiming to investigate how different augmentation strategies influence the model's ability to learn robust features and generalize well to unseen data. Through this study, I learnt to seek to provide insights into the efficacy of data augmentation in enhancing the performance of image classification models.

2 Environment

Coding Platform	Jupyter Notebook (VS Code based)
Image Dataset	Cifar-10
Deep Learning Framework	Keras
Data Augmentation Method	ImageDataGenerator (provided in keras.preprocessing.image)
Model	Convolutional Neural Network

3 System Design Details

3.1 Data Preprocessing

Cifar-10 Image dataset is a subset of the Tiny Images dataset and consists of 60000 32x32 colored images. The images are labelled with one of 10 mutually exclusive classes: airplane, automobile (but not truck or pickup truck), bird, cat, deer, dog, frog, horse, ship, and truck (but not pickup truck). There are 6000 images per class with 5000 training and 1000 testing images per class.[2] Simply run by `(x_train, y_train), (x_test, y_test) = cifar10.load_data()` utilizes

the `cifar10.load_data()` function, which is a data loading function within the Keras library, used for loading the CIFAR-10 dataset. It automatically downloads the CIFAR-10 dataset, splits the dataset into training and testing sets, and returns the segmented data. As a result, manual partitioning of the dataset into training and testing sets is not necessary.

3.1.1 Feature Normalization

To let the neural network better understand the features, normalization was performed by the following code:

```
1. X_train = np.asarray(x_train, dtype=float)/255
2. X_test = np.asarray(x_test, dtype=float)/255
```

It scales the value range of the original data to a smaller range, usually between 0 and 1, beneficial for higher numerical stability, faster convergence speed and model generality.

3.1.2 One-hot Encoding for Labels

As is known, the dataset is multi-labelled by 10 classes, so we encode the labels to one-hot format which shall benefit the subsequent CNN training for the following reasons:

(a) Compatibility: we expect the output layer's activation function to be softmax, which can output a probability distribution with each category corresponding to a probability value. One-hot encoding fits well with the working principle of the softmax output layer because it provides a separate output bit for each category.

(b) Loss function calculation: during training, neural networks usually use the cross entropy loss function to calculate the error of the model, which requires that the target label be in the form of a unique thermal code in order to correctly calculate the difference between the model output and the actual label.

(c) Eliminating the impact of category index: in a CNN training process, the index of the category may be misinterpreted as having some order or weights. One-hot vectors could help diminish this possibility.

Realizing one-hot encoding by python:

```
1. y_cat_train = to_categorical(y_train, 10)
2. y_cat_test = to_categorical(y_test,10)
```

3.1.3 Training Data Split for Validation

In the usual practice cases, training data will be split into two part by a specific proportion. The python practice is:

```
1. X_train_, X_val, Y_train, Y_val = train_test_split(X_train, y_cat_train, random_s
   tate=10, test_size=0.3)
```

In which the test size proportion is set to be 30% of the original training set. In the following experiment, the `test_size` may be changed according to the actual need.

3.2 Deep Learning Framework

The commonly used deep learning model for image classification is Convolutional Neural Network, since traditional machine learning methods (such as multi-layer perception machines, support vector machines, etc.) mostly use shallow structures to deal with a limited number of samples and computing units. When the target objects have rich meanings, like images, the performance and generalization ability of complex classification problems are obviously insufficient. The convolution neural network (CNN) has been widely used in the field of image processing because it is good at dealing with image classification and recognition problems and has brought great improvement in the accuracy of many machine learning tasks.[3]

To introduce CNN to the project, I chose Keras for construction, which is a library that provides highly powerful and abstract building blocks to build deep learning networks.[4] The neural network was defined by `Sequential()` method, with layers summarized in Table1.

3.2.1 Conv2D Layer

Convolution 2D layer is firstly deployed for extract features from input images by the convolutional kernel over inputs and computing their dot product at each position, shown in Image 1. 32 by 32 feature map will be generated in the first layer, and 16 by 16 in the 5th layer. Stacking convolution layers leads to hierarchical representations, which capture increasingly complex and abstract features by understanding features in the previous layers.[5]

3.2.2 Batch Normalization Layer

Batch normalization is used to normalize the output of the previous layer, helpful for decreasing internal covariate shift, higher convergence speed and better performance. Meanwhile, it could be applied to deal with overfitting issue as it normalizes the inputs and decrease gradient vanishing or gradient explosion.

3.2.3 DepthwiseConv2D Layer

Deep convolution is a special type in convolutional neural networks that first applies convolution to each input channel separately and then uses 1x1 convolution to merge features. It was pointed to be less computational resources consumed than a standard Conv2D layer, that the computational cost of standard convolution can be denoted as: [6]

Layer (type)	Output Shape	Param #
conv2d_30 (Conv2D)	(None, 32, 32, 32)	896
batch_normalization_30	(None, 32, 32, 32)	128
depthwise_conv2d_30	(None, 32, 32, 96)	960
dropout_35	(None, 32, 32, 96)	0
conv2d_31 (Conv2D)	(None, 16, 16, 64)	55360
batch_normalization_31	(None, 16, 16, 64)	256
depthwise_conv2d_31	(None, 16, 16, 64)	640
dropout_36	(None, 16, 16, 64)	0
conv2d_32 (Conv2D)	(None, 16, 16, 128)	73856
batch_normalization_32	(None, 16, 16, 128)	512
depthwise_conv2d_32	(None, 16, 16, 128)	1280
dropout_37	(None, 16, 16, 128)	0
conv2d_33 (Conv2D)	(None, 16, 16, 128)	147584
batch_normalization_33	(None, 16, 16, 128)	512
depthwise_conv2d_33	(None, 16, 16, 128)	256
conv2d_34 (Conv2D)	(None, 8, 8, 256)	295168
batch_normalization_34	(None, 8, 8, 256)	1024
depthwise_conv2d_34	(None, 8, 8, 256)	2560
conv2d_35 (Conv2D)	(None, 4, 4, 512)	131584
batch_normalization_35	(None, 4, 4, 512)	2048
depthwise_conv2d_35	(None, 4, 4, 512)	1024
dropout_38	(None, 4, 4, 512)	0
flatten_5	(None, 8192)	0
dropout_39	(None, 8192)	0
dense_15 (Dense)	(None, 2048)	16779264
dropout_40	(None, 2048)	0
dense_16 (Dense)	(None, 512)	1049088
dropout_41	(None, 512)	0
dense_17 (Dense)	(None, 10)	5130

Table1: The CNN Summary

$$D_k \cdot D_k \cdot M \cdot N \cdot D_F \cdot D_F \quad (1)$$

And the computational cost of depthwise convolution can be denotes as:

$$D_k \cdot D_k \cdot M \cdot D_F \cdot D_F \quad (2)$$

In Formula (1) and (2), denote the size of kernel, denote the size of the feature map, denote the input channels, and denote the output channels. The Formula (1) and (2) shows that the computational complexity of depthwise convolution is lower than standard convolution.[7]

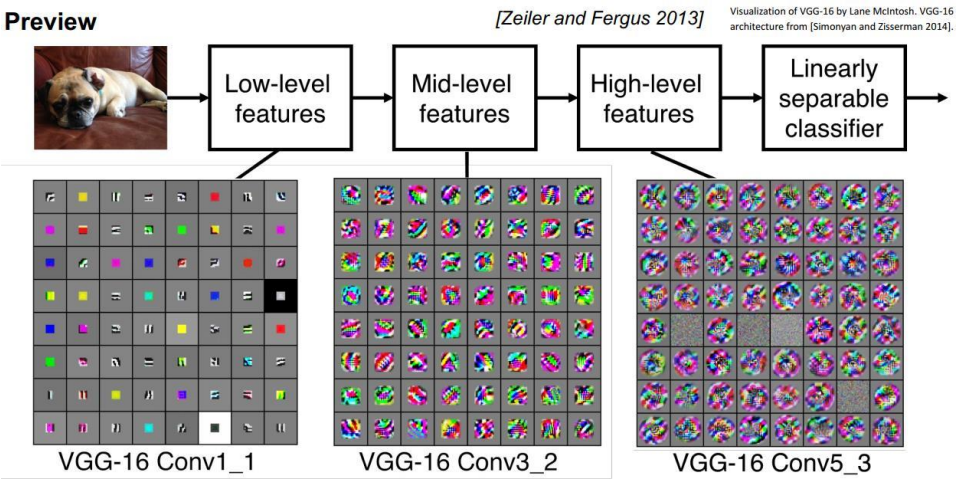


Image1: schematic diagram on how stacking convolution layer extract features with higher levels.

3.2.4 Dropout Layer

Dropout is a regularization technology for avoiding overfitting. It randomly discards some outputs of some neural cells by setting the value to zero, so to make the model less dependent on any single feature.

3.2.5 Flatten

The flattening layer converts the multi-dimensional feature map into a one-dimensional vector that can be passed to the fully connected layer (Dense layer). This is the transition from the convolutional layer to the fully connected layer.

3.2.6 Dense

The fully connected layer is a traditional neural network layer, which extracts and learns advanced features from the flattened feature vectors. In CNNS, the fully connected layer is usually located at the end of the network and is used to integrate the features extracted by the previous convolutional layer and pooling layer and perform the final classification or regression task.

3.2.7 Additional settings

Early-stop is set by:

```
1.  early_stop = EarlyStopping(monitor='val_loss',patience=4)
2.  model.fit(…,callbacks=[early_stop])
```

In this way, redundant training process will be cut when the monitor found no smaller validation loss value occurring in the latest 4 epochs.

3.3 Data Augmentation

Data augmentation is performed with the `ImageDataGenerator` method provided by Keras. It is easy to set the image augmenting way e.g. transformation method, transformation level and possibilities. Here we have the code implementation:

```
1.  datagen = ImageDataGenerator(
2.          rotation_range=0.3,
3.          zoom_range = 0.1,
4.          width_shift_range=0.1,
5.          height_shift_range=0.1,
6.          horizontal_flip=True)
7.  datagen_original = ImageDataGenerator()
```

The `datagen` variable stores all settings for this augmentation task, with the values essentially representing random numbers ranging from zero to the specified maximum value. Multiple commonly used methods are applied, such as rotation, zooming, height and width shifting etc.
After the preparation, we could integrate the whole augmentation process into a function, by using `flow()`, which receives data from a given numpy array and tag array, and then generates a batch of image data that is pre-processed and enhanced with real-time data. This is very useful for increasing the generalization ability of the model, as it can extend the training data set in a natural way without distortion.

```
1.  def generate_combined_batches(X_train_, Y_train, batch_size=64):
2.      half_batch_size = batch_size // 2
3.      gen_augmented = datagen.flow(X_train_, Y_train, batch_size=half_batc
    h_size)      #Generate augumented data with half the samples
4.      gen_original = datagen_original.flow(X_train_, Y_train, batch_size=b
    atch_size)   #Generate original data with full the samples
5.      while True:
6.         X_batch_augmented, Y_batch_augmented = next(gen_augmented)
```

```
        X_batch_original, Y_batch_original = next(gen_original)
        X_batch_combined=np.vstack((X_batch_augmented, X_batch_original))
        Y_batch_combined=np.vstack((Y_batch_augmented, Y_batch_original))
        yield (X_batch_combined, Y_batch_combined)
```

3.4 Test Result Analysis

We need to properly design a test method to accurately analysis how good or how bad is the parameters influencing the model in the generalization ability, by values like accuracy, F1 score, ROC, AUC etc.

3.4.1 Accuracy and Loss Comparison

Intuitively, we think that a model with insufficient training data will lead to poor generalization ability and inaccurate performance evaluation, basically we may be conscious of an obvious gap between the loss of training set and the loss of validation set, so as the accuracy. Hence, to better assess the difference, we plot the loss-epoch and accuracy-epoch line chart, with the data obtained from both the training set and validation set. Meanwhile, the loss and accuracy of the testing results will be printed for parallel comparison, in this case, to evaluate how the data augmentation will influence the performance of a CNN model.

```
losses_ = pd.DataFrame(model_.history.history)
losses_[['loss','val_loss']].plot()
losses_[['accuracy','val_accuracy']].plot()
print(model_.metrics_names)
print(model_.evaluate(X_test,y_cat_test,verbose=0))
```

3.4.2 Confusion Matrix

Confusion matrices are recognized as an intuitive and efficient way to show model performance. We mostly observe the value of precision, recall, F1 score and accuracy, which could be calculated as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \tag{5}$$

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision+Recall}} \tag{6}$$

The confusion matrix is well integrated in the scikit-learn library, so we call the `classification_report()` function to directly print the confusion matrix:

```
print(classification_report(y_test,predictions))
```

As well as to print the ground truth-prediction matrix:

```
confusion_matrix(y_test,predictions)
```

In which Elements on the diagonal represent the number of samples that the model correctly classifies. And row *i* column *j* element: represents the number of samples whose true class is class *i* and the model predicts is class *j*.

3.4.3 Play with Specific Test Samples

The most intuitive way to show how good the model is in the prediction task, is for sure to play it with some specific test samples. The parameters of the set coming into playing is free to adjust if properly set by the following codes:

```
classes = [0,1,2,3,4,5,6,7,8,9]
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', '
frog', 'horse', 'ship', 'truck']
d = dict(zip(classes, class_names))
i_plot=1
for i in range (50, 71, 5):
    plt.subplot(1, 5, i_plot)
    i_plot+=1
    img = X_test[i]
    #fig.add_subplot(2, 5, i+1)
    plt.axis('off')
    plt.imshow(img)
    input_img = X_test[i].reshape(1,32,32,3)
    predictions = np.argmax(model_.predict(input_img), axis=-1)[0]
    print(f"True class: {d[y_test[i][0]]} \nPredicted class: {d[predicti
ons]}")
plt.show()
```

4 Test and Analysis and Conclusions

4.1 Test Result Collection

To check how will the performance of the CNN trained differentiate between an insufficient training data set and augmented training data, the experiment is conducted by 6 groups, with different train-test split proportion for accessing the generalization ability and with original and augmented data. Specifically, larger training data will generally produce a better assessment result on training data but may cause overfitting that shows worse generalization ability on validation data in most of the time. Insufficient data provided for training may make the CNN's generalization ability even worse as features are not well learnt and not be able to regenerate. The final loss and accuracy during the training process in different scenarios are given in Table2. The overall test results are listed in Table 3.

	Data Settings	Final Training Loss	Final Training Accuracy	Final Validation Loss	Final Validation Accuracy
1	No augmentation with 6% validation split	0.2887	0.8992	0.5270	0.8327
2	Augmentation applied with 6% validation split	0.2513	0.9168	0.4619	0.8637
3	No augmentation with 30% validation split	0.2985	0.8958	0.6343	0.8074
4	Augmentation applied with 30% validation split	0.3383	0.8843	0.4848	0.8471
5	No augmentation with 30% validation split, **raw data reduced**	0.4158	0.8506	0.8587	0.7928
6	Augmentation applied with 30% validation split, **raw data reduced**	0.3737	0.8733	0.6536	0.8042

Table 2. Loss and accuracy for training set and validation set in the final epoch respectively

	Data Settings	Data Set Info	Test Loss	Test accuracy	Test weighted F1 score	Test macro F1 score
1	No augmentation with 6% validation split	Training set shape: (47000, 43, 43, 4) Validation set shape: (3000, 32, 32, 3)	0.5241	0.8380	0.84	0.84
2	Augmentation applied with 6% validation split		0.4592	0.8619	0.86	0.86
3	No augmentation with 30% validation split	Training set shape: (35000, 32, 32, 3) Validation set shape: (15000, 32, 32, 3)	0.6054	0.8207	0.82	0.82
4	Augmentation applied with 30% validation split		0.4996	0.8461	0.84	0.84
5	No augmentation with 30% validation split, **raw data reduced**	Training set shape: (17150, 32, 32, 3) Validation set	0.8595	0.7315	0.73	0.73

6	Augmentation applied with 30% validation split, **raw data reduced**	shape: (7350, 32, 32, 3)	0.6441	9.979 % 0.8045	0.80	9.589 % 0.80

* Test set remains 10000 items

Table 3. Test results in different data settings

The following line chart (fig 1.) displays the accuracy and loss trends of both the training and validation data throughout the training process.

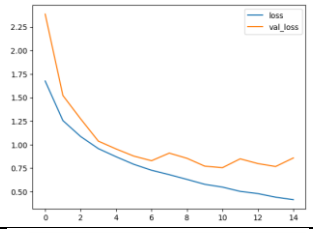
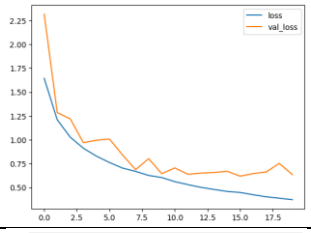
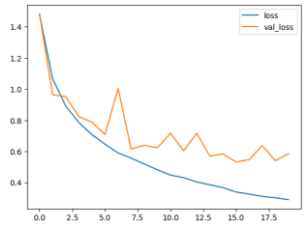
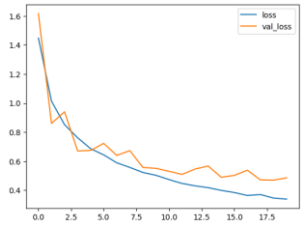
Data Settings	No Augmentation	Augmented
30% validation split, **raw data reduced**		
30% validation split		

Fig 1. loss trends of both the training and validation data throughout the training process.

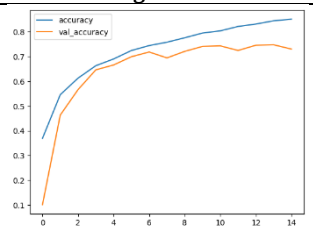
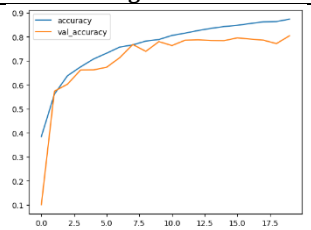
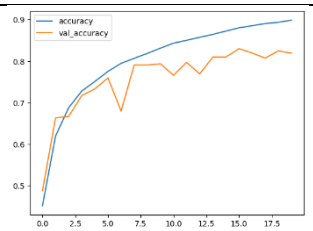
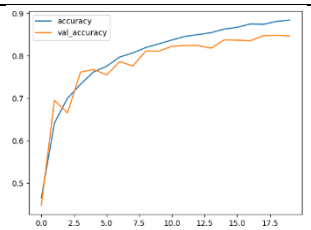
Data Settings	No Augmentation	Augmented
30% validation split, **raw data reduced**		
30% validation split		

Fig 1. Accuracy trends of both the training and validation data throughout the training process.

4.2 Test Result Analysis

(1) Training-validation slit should be properly set to balance between overfitting and underfitting

The size of the verification set has an important effect on the training and final performance of the model. Validation sets are used for performance evaluation during model training but are not used directly for weight updating. **A larger validation set provides more data to evaluate the model's ability to generalize, but also means less data to train the model. This can result in weight updates during training based on less data, potentially affecting the quality and diversity of patterns the model learns.**

Increasing the proportion of validation sets (from 6% to 30%) did not appear to have a significant negative effect on test accuracy, but it was observed that a larger validation set split (30%) had a slight increase in test loss and a slight decrease in accuracy for the model when compared to a

smaller validation set split (6%).
In the case of data enhancement shown in Table 3, the increase in validation set size also resulted in an increase in test losses (from 0.4592 to 0.4996), but the test accuracy decreased slightly from 0.8619 to 0.8461. The decrease in accuracy was not significant, which **may indicate that data enhancement helps mitigate the performance degradation that may result from a reduction in the amount of training data due to a larger validation set.**

(2) Insufficient data size can significantly influence the information that a CNN will be able to learn.

When comparing results between the raw training data and the reduced data in the 3:6 split scenario (Table 2), the final validation loss during the training process was 0.5408 for the original data, whereas it becomes 0.8587 for the reduced data, while the accuracy dropped from 0.8247 for the original data to 0.7298 for the reduced data. The test results yield from Table 3 shows that when the amount of training data is reduced, the model performance decreases. In particular, in the absence of data enhancement, reducing the amount of training data to a certain extent (case 5) resulted in a significant increase in test losses (0.8595) and a significant decrease in test accuracy and F1 scores. Also, from the figures plotted, it is obvious that the gap between training loss and validation loss or training accuracy and validation accuracy for the original training data, is much smaller than the ones for reduced training data.
This suggests that the **reduced data may not provide sufficient information for the CNN to generalize well regarding unseen data** and achieve high performance in terms of loss and accuracy metrics.

(3) Data Augmentation can significantly improve the model performance in a data insufficient manner.

Data augmentation has a significant positive effect on model performance. When using data augmentation, both test accuracy and F1 scores were improved. **This shows that data augmentation can improve the generalization ability of the model.**
When comparing results between the raw training data and the reduced data in the 3:6 split scenario (Table 3), the final training accuracy is less smaller (0.8958→0.8506) for the augmented case than the unaugmented ones (0.8843→0.8733), which further demonstrates the positive impact of data enhancement on model performance.
Data augmentation also decreases the risk of overfitting which may involve noise or some certain features. In case 2 (with data augmentation, 6% validation set), the model exhibits the best test accuracy and F1 scores, and the lowest loss values. In contrast to case 1 (no data augmentation, 6% verification set), both accuracy and F1 scores decrease, and loss values increase.
Even when the amount of data was reduced (case 6), better results were obtained when data augmentation was applied than when it was not applied (case 5). This suggests that data augmentation plays a key role in improving model performance, especially when there is less training data available.
In summary, Data augmentation does reduce the risk of overfitting a model and improve its performance on new data, increase the robustness of a CNN model. But which also needs to be kept in mind that **data enhancement will cause more computational resources consuming, trade-offs need to be considered**, and with more data, batch size and batch number may change, so more proper and careful calculation needs to be performed in order to ensure the model could work will still as expected.

4.3 Conclusions

Data enhancement is a key strategy for improving model performance, especially when there is less training data available. The size of the validation set has a smaller effect on model performance, but the effect may be more pronounced with less training data However, it is also important to note the relationship between loss and accuracy, as well as the differences in model performance on the training set, the validation set, and the test set.

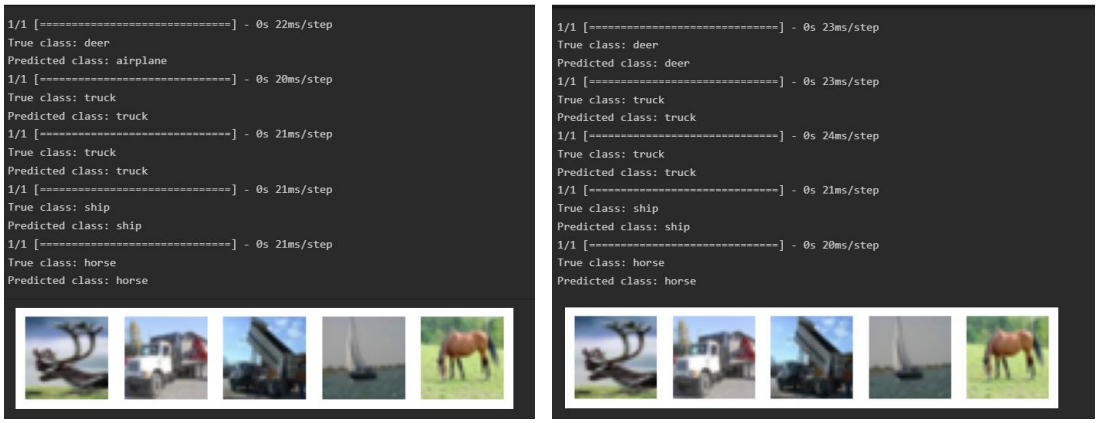


Fig 2. Prediction results for some randomly selected samples, between unaugmented (left)and augmented (right) modelling

5 Things to be Improved

To reduce randomness as the training process is initiated, more repeated experiment needs to be performed. Due to time and computational resources limit, I was not able to run repeated experiments any more times as expected. If a more rigorous experiment and reporting are needed, this should be improved.

Some detailed aspects in the coding manner shall be fixed or enhanced in the future: (1) plots should be scaled to the same coordinate scale, (2) parameters that are too extreme should not be used (6% validation proportion), (3) the entire script should be structured better, e.g. the training set reduction was performed after validation set splitting, when I found serious shape problems encountered that made the model unable to converge, so I moved the section to the very front part to solve this.

In this experiment, all machine learning process behind the screen were ran on the CPU. GPU training for Keras are also popular and verified, but the implement process was complex and miscellaneous. Due to some uncovered incompatibility problem, I was not able to use GPU for this, so time consumed hugely.

6 References

[1] A. Deis. "Data Augmentation for Deep Learning." https://towardsdatascience.com/data-augmentation-for-deep-learning-4fe21d1a4eb9 (accessed.

[2] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[3] M. Xin and Y. Wang, "Research on image classification model based on deep convolution neural network," *EURASIP Journal on Image and Video Processing*, vol. 2019, no. 1, 2019, doi: 10.1186/s13640-019-0417-8.

[4] N. Ketkar, "Introduction to Keras," in *Deep Learning with Python: A Hands-on Introduction*. Berkeley, CA: Apress, 2017, pp. 97-111.

[5] A. Karpathy, "Stanford cs231n-convolutional neural networks for visual recognition," ed, 2015.

[6] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[7] S. Batra *et al.*, "DMCNet: Diversified model combination network for understanding engagement from video screengrabs," *Systems and Soft Computing*, vol. 4, p. 200039, 2022.