Deep Learning Project – Human Action Recognition

Name : Nikhil H

1 Problem Definition and Analysis

To predict the action and the action class corresponding to a particular Human Action Recognition using deep learning and convolutional Neural Network.

Data set Analysis:

The dataset consists of 3030 training images and 2100 test images. There are 21 actions and 5 corresponding action class related to each of the action. Each of the image has a different size. All images are in RGB Format.

Data exploration:

Exploration of feature action

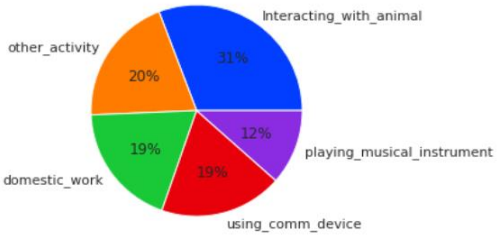


Fig 1

Exploration of feature action_class

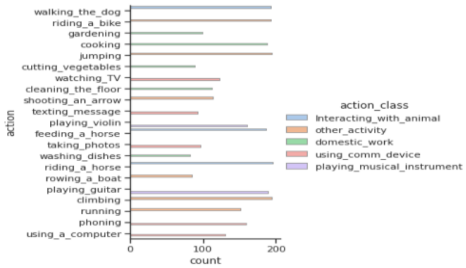


Fig 2

Analysis of Fig 1: From the above pie chart in Fig 1 we can clearly see that the number of records for the action class playing_musical_instrument is less compared to the other action classes. So, the model might have less accuracy on playing_musical_instrument action class.

Analysis of Fig 2: From the above plot in Fig 2 we can observe that the number of records in 21 different actions is normally distributed.

2 Evaluation Framework

For model evaluation I am using the categorical_crossentropy for loss and accuracy for performance metric.

The fine tuned model was evaluated on the test set. Following are the evaluation metrics:

Parameter	Name	Value
Loss = Categorical_crossenropy	action_output_loss	1.7444
	action_class_output_loss	0.8847
Metric = Accuracy	action_output_accuracy	0.5024
	action_class_output_accuracy	0.7000

3 Approach & Justifications

1) Explore the dataset to check for class imbalance.

2) Label encode the target values of action and action class.

3) Split the dataset into train (70%), test (15%) and validation (15%).

4) A base model(simple) model is created using VGG, and transfer learning is used to train the model.

5) Set the trainable = False to freeze the VGG model and exclude the top layer of the VGG using include_top = False. Create a multioutput model using the Keras Functional API and add a Flatten layer and 2 output branches to predict the action and action_class.

6) Experiment with different activation function (relu, elu). Select the best activation function for the specific branch based on the reduce in overfitting.

7) Create a complex model by adding more hidden layers to the Action and action_class branch. Compile and fit the model with the new complex model using training data without augmentation.
8) Use training data with augmentation and compile and fit on the complex model and check to make sure the overfitting has reduced.
9) To still reduce the overfitting use L1 and L2 regularization with different values and compare the results.
10) L1 regularization reduces the overfitting more compared to L2 regularization.
11) After selecting the L1 regularization add a dropout layer to reduce the overfitting.
12) After adding the L1 regularization and dropout the model's overfitting has reduced considerably.
13) Evaluate the fine-tuned model on the test_data_generator rand check the loss and accuracy.
14) Predict the model on the test set. The model predicts two outputs one for action and another for action class.
15) Use np.argmax to find the highest value in the output and apply inverse label_encoder to convert the label encoded number to the corresponding action and action_class.

Justifications:

Model:

I am using the transfer learning approach to train the model. I have imported the VGG from keras and the weights is set to the ImageNet.The Keras Functional API is used to add the layers to the model. Since the Functional API is flexible and can be used for multioutput problem. A flatten layer is added at the end of the VGG model, and 2 branches are created to predict the output related to action and action_class. The introduction of more layers in VGG has allowed the model to better understand the features with an image.

Early Stopping and the number of epochs:

The number of epochs is set to 50 and since the model is using transfer learning it starts overfitting after 10 epochs. So, I have used early stopping with patience = 15 and monitor = 'val_loss'. So the model waits for 10 more epochs and if there is no decrease in val_loss the model training is stopped before max_epcochs(50) is completed.

Activation Functions:

Output layer – softmax: For each of the output layers softmax activation is used since we want the probability (max likelihood) of an observation to be predicted as one particular action and action_class. So, this is multiclass classification corresponding to particular action and action_class.

Hidden layers- elu , relu: The elu is used as an activation function for the hidden layers in the action_class branch as it has less overfitting compared to relu. For the action branch relu is used as the activation function for the hidden layers. Both the elu and relu are computationally less expensive than compared to the other activation functions as both relu and elu have simple computation.

Metrics:

Loss - categorical cross entropy: The categorical cross entropy is used, as the problem statement relates to a multiclass classification. Since the data is returned in categorical form (one hot encoded). If the data was returned in integer form, I could have used the sparse categorical cross entropy.

Performance metric - accuracy: The accuracy is used as a performance metric as we want to know the percentage of images classified into respective action and action class accurately.

Optimizers:

SGD vs ADAM: I have experimented with both the optimizers, and both have similar results. But SGD is computationally expensive, and it takes more time to train the model using SGD optimizer.

To reduce overfitting:

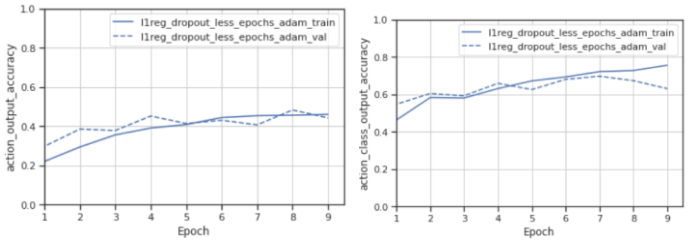
Augmentation: The image augmentation is used to create images of different shapes and of different rotation. This helps in generalization of the model. The other augmentations like contrast, saturation of images is avoided as it is leading to more overfitting.

Dropout and L1 reg: The dropout of 0.2 was added after each hidden layer in the action and action_class branch. Adding the dropout reduces the overfitting as it randomly drops the values of certain neurons based on the specified number for dropout. The L1 regularization is used as it reduces the overfitting considerably, since the cost added is proportional to the absolute value of the weight's coefficients.

Experiments & Tuning:

Experiments	Tuning	Effect
Creating Simple model	Use the VGG model and add a flatten layer and create 2 branches to predict 2 outputs(action and action class)	The baseline model helps in comparing the results.
Activation: Relu vs Elu	Used relu activation for hidden layers in action branch and elu for the hidden layers in the action class branch.	Reduces the overfitting in each of the respective outputs of action and action_class.
Creating Complex model	Add 2 hidden layers each for the action and the action class branch.	Increases the model capacity
Complex model with augmentation	The rotate and shift augmentation techniques are used to create augmented images.	Helps in generalization of the model by reducing overfitting.
L2 regularization and L1	Experiment with both regularization by changing the values	Finally, I used L1 regularization with 0.001 as it reduced the overfitting considerably compared to L2 regularization.
Dropout + L1 regularization	Experimented with dropout values of 0.2 and 0.4.	Helps in reducing the model overfitting.
SGD vs ADAM	Used both the optimizers on the complex model and both have similar performance.	Finally selected Adam as it takes less time to train the model compared to using the SGD optimizer.

4 Ultimate Judgment, Analysis & Limitations



The model starts overfitting after 10 epochs as I am using transfer learning. So, for the final model I have used 10 epochs and we can clearly see from the graphs that overfitting has reduced. The model performs well on action_class.

Limitation: The model will not perform well on action_class = playing_musical_instrument as there are less number of observations for the action_class. The model accuracy is less for predicting the action labels.

5 References:

[1] Brownlee, J. (2021) **Neural Network Models for Combined Classification and Regression. Available at: https://machinelearningmastery.com/neural-network-models-for-combined-classification-and-regression**

[2] Building a multi-output Convolutional Neural Network with Keras *(2020). Available at:*

https://towardsdatascience.com/building-a-multi-output-convolutional-neural-network-with-keras-ed24c7bc1178

[3] Team, K. (2021) **Keras documentation: Transfer learning & fine-tuning, Keras.io.** ***Available at:***

https://keras.io/guides/transfer_learning/