CS985: Machine Learning for Data Analytics

Emotion Recognition Classification Task.

Google Colab Link:

Emotion recognition FINAL.ipynb - Colaboratory (google.com)

Overview:

We were assigned a dataset of facial images and tasked with building models which could successfully predict what emotions the images signify. At first, we saw that the data contained about 30000 rows of strings of pixels and their corresponding emotion labels. We started exploring the data by converting the pixel strings into 48x48 numpy arrays. We pre-processed the data by resizing the images and splitting them into train and validation sets.

We also tried augmenting the images (changing their sizes, shifting the rotation, etc.) which helped us increase the size of the training set. We used the smaller batch (X_train2) on the notebook as we assume the computational power is only of google collab. We used a much larger augmented dataset (X_train1) separately on google pro collab with extra compute units but did not include this in our notebook.

We then built a couple of different models: CNN, a deeper CNN with more layers, a VGG16,Resnet and Inception model and different ensembles. The models were trained separately, and their performances were evaluated on the validation set.

For the final model, we experimented with an ensemble model which combined ResNet, VGG16 and deeper CNN model which gave us the best results. It outperformed all the other models and gave us an accuracy of around 63.95%.

Some key takeaways and recommendations from this project include:

- Preprocessing and data augmentation techniques can greatly improve model performance, especially on small datasets.
- Ensemble models can sometimes outperform individual models and are worth exploring if you have multiple models available.
- It's important to carefully choose and tune hyperparameters, such as learning rate and dropout rate, to optimize model performance.

- Accuracy is an important metric but we also need to consider precision, recall and train/test loss as well.

Method:

Before modelling, the data was explored to gain insights into the distribution of the different classes and any imbalances. The data was pre-processed by normalizing the pixel values, converting grayscale images to RGB(for transfer learning), and resizing images(for transfer learning).The data was then split into training, validation, and testing sets, with 80% of the data used for training, 10% for validation, and 10% for testing. The data was stratified by class to ensure that each set had a balanced representation of each emotion.

To further augment the data and improve model performance, various data augmentation strategies were employed, including horizontal and vertical flips, rotation, zooming, and shearing. This helped to increase the diversity of the training data and reduce the risk of overfitting.

During training, early stopping and learning rate reduction were used to prevent overfitting and improve model convergence.

After exploring the data, it was observed that the dataset was imbalanced, with some classes having fewer samples than others. This influenced the approach to modelling, as techniques such as stratified sampling and data augmentation were used to ensure that each class was adequately represented during training.

We also used a hyperparameter optimization technique called Bayesian Optimization to improve our models slightly. The technique is a probabilistic approach that uses a model of the objective function (e.g., validation accuracy or loss) to select the next set of hyperparameters to evaluate.

In summary, the methodology for modelling involved exploring and pre-processing the data, building and training several deep learning models using different architectures, using data augmentation techniques to improve performance, and employing early stopping, learning rate reduction, and model checkpointing to prevent overfitting and improve convergence. The data was split into training, validation, and testing sets, with 80% for training, 10% for validation, and 10% for testing, and was stratified by class to ensure balanced representation. The key learnings from this project include the importance of data exploration and pre-processing, the use of data augmentation techniques to improve model performance, and the need for early stopping and learning rate reduction to prevent overfitting. Finally, it is recommended to continue experimenting with different architectures and hyperparameters to achieve better performance on this dataset.

Models:

For this project, the following models were explored:

1. Standard ML Baseline: The standard ML baseline model uses a Multi-Layer Perceptron (MLP) Classifier with two hidden layers of sizes 128 and 64, respectively. The activation function is ReLU and the solver is adam. The input data is reshaped to have two dimensions, and the model is trained on the training set and evaluated on the validation set. The performance is measured using mean squared error. This model serves as a simple baseline to compare the performance of more complex models.
2. Deep Neural Network (DNN): A DNN with different configurations was explored. The following configurations were tested:
    i. CNN (4 layers and 7 layers): This is a convolutional neural network (CNN) model that is used for image classification. The model architecture has several layers of convolutional and pooling layers, followed by a few fully connected layers. The input images have a shape of (48, 48, 1), which corresponds to 48 x 48 grayscale images.
    ii. Bayesian Optimized CNN: We used this hyperparameter tuning technique to further optimize the deep CNN model and this got very good scores.

3. Complex Models: Four complex NN models were used in this project. Below are the configurations of the models:
    i. Model_ResNet:
        1. Pre-trained ResNet50 as base model with weights from ImageNet dataset
        2. A layer to convert the Grayscale images to RGB.
        3. A layer to resize the images to the Resnet 224x224 pixel format
        4. Dense layer with 64 neurons
        5. Dropout with a rate of 0.5 after the Dense layer
        6. Softmax activation function in the output layer
    ii. Model_VGG:
        1. Pre-trained VGG16 as base model with weights from ImageNet dataset
        2. A layer to convert the Grayscale images to RGB
        3. A layer to resize the images to 224x224 pixels.
        4. Three Dense layers with 32 neurons each
        5. Batch Normalization after each Dense layer
        6. Activation function ReLU after each Batch Normalization layer
        7. Dropout with a rate of 0.5 after each Dense layer
        8. Softmax activation function in the output layer

9. Data augmentation was applied to increase the number of training samples.

iii. Transfer learning Inception

The Transfer learning Inception model used in this project has the following configuration:

- Pre-trained InceptionV3 as the base model with weights from the ImageNet dataset
- A layer to convert the grayscale images to RGB
- A layer to resize the images to 75x75 pixels which is needed for Inception
- The last layer of the base model is removed and a Global Average Pooling 2D layer is added.
- A Flatten layer is added after the Global Average Pooling 2D layer.
- A Dense layer with 256 neurons and ReLU activation function is added after the Flatten layer.
- A Dropout layer with a rate of 0.5 is added after the Dense layer to prevent overfitting.
- The final Dense layer with 7 neurons and Softmax activation function is added to classify the input images into 7 classes.
- The input shape is (48, 48, 1) as the images are in grayscale.
- Data augmentation is applied to increase the number of training samples.
- The weights of the base model are not updated during training, i.e., the base model is frozen. Only the weights of the new layers are updated during training.

iv. Ensemble model (Average of models):

1. This ensemble model is created by combining the predictions of three different models: **`resnet`**, **`model_bayes`**, and **`model_VGG`**. The input to the model is an image with shape (48, 48, 1), and the output is the average of the predictions made by the three models.
2. The **`Average`** layer is used to take the average of the outputs of the three models. The ensemble model is then compiled using the **`categorical_crossentropy`** loss function, **`Adam`** optimizer with a learning rate of 0.0001, and accuracy metric.
3. Two callbacks are used in the model training: **`ReduceLROnPlateau`** and **`EarlyStopping`**.

`ReduceLROnPlateau` is used to reduce the learning rate if the validation loss stops improving, and **`EarlyStopping`** is used to stop the training if the validation loss doesn't improve after 20 epochs.

(We also tried ensembling multiple Deep layer CNN's however the accuracy was not great).

Overall, the models with the highest accuracy were the Bayesian Optimized deep CNN, the ensemble comprised of Resnet,VGG16 and our Bayesian optimized deep CNN,. VGG16 was the best transfer learning model. The key takeaway was that using image augementation with image transformations can greatly improve the accuracy of the model. The optimization of the hyperparameters also helped model performance.

Results:

In this project, we explored several deep learning models to classify facial expressions. We started with a standard ML baseline, followed by three different deep neural network models, and one complex CNN model. The table below shows the summary of the models and their performances.

Model	Accuracy	Parameters
Standard ML Baseline	MSE values: 0.14392857	N/A
Deep NN Model 1 (4 layer CNN)	59.85%	1,949,447
Deep NN Model 2 (7 layer CNN)	60.50%	1,394,183

Model	Accuracy	Parameters
Deep NN Model 3 (Bayesian Optimized CNN)	63.5%	2,901,831
Complex CNN Model 1 (Ensemble model of 7 layer CNN repeated 10 times)	50.20%	1,394,183
Complex CNN Model 2 (Inception)	30.33%	22,329,127
Complex CNN Model 3 (ResNet)	32.50%	24,114,055
Complex CNN Model 4 (VGG16)	52.9%	15,620,615
Complex CNN Model 5 (Ensemble with ResNet, Bayesian Optimized CNN and VGG16)	63.95%	42,636,501

We split our data into 80% training and 20% validation and used the validation data to choose the best model configuration. We used data augmentation techniques, such as random cropping and flipping, to increase the size of our training dataset. We also employed different

training schedules, such as learning rate reduction and early stopping, to prevent overfitting and improve the generalization of our models.

Summary:

We found that the Final ensemble model outperformed the other models, with an accuracy of 63.95%.This model had the most parameters, but its architecture was specifically designed for image classification tasks. We tried several different configurations for each of our models, including varying the number of layers, nodes, and activation functions. We found that adding more layers and nodes improved the performance of the models up to a certain point, after which the performance started to degrade. We also found that using dropout regularization and batch normalization improved the generalization of the models and prevented overfitting.

Overall, we learned that deep learning models can be effective for classifying facial expressions, but their performance is highly dependent on the model architecture and configuration. We also learned that data augmentation techniques can significantly improve the performance of our models, and that employing different training schedules can prevent overfitting and improve the generalization of our models.

At the end, we used all the different batches of image augmentations(X_train1 instead of X_train2) which contained 351000 images. We then fed this into the ensemble model containing ResNet, the Bayesian optimized CNN and VGG16 models. This model achieved the highest accuracy.

We were consistently getting 70%-71% accuracy on our validation sets, around a 6-7 % increase compared to the smaller augmented data batch(X_train2). We submitted our best model to Kaggle and achieved an accuracy of 69.413%.

References:

1. Understanding and Coding a ResNet in Keras | by Priya Dwivedi | Towards Data Science. [online]. Available at: https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33..
2. sklearn.neural_network.MLPClassifier — scikit-learn 1.2.2 documentation. [online]. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html..
3. Brownlee, J. (2017). How to Use The Pre-Trained VGG Model to Classify Objects in Photographs. [online] machinelearningmastery.com. Available at: https://machinelearningmastery.com/use-pre-trained-vgg-model-classify-objects-photographs/ [Accessed 1 Jan. 1970].
4. Convolutional Neural Network (CNN) | TensorFlow Core. [online]. Available at: https://www.tensorflow.org/tutorials/images/cnn.