Indian Institute of Technology, Bombay

Project Report

Indian Classical Dance Classification from Dance Poses

Course: Foundations of Machine Learning
(CS725)

Submitted By

Aarushi Aiyyar (Roll No. : 203050045)
Bhaveshkumar Yadav (Roll No. : 193050052)
Khyati Oswal (Roll No. : 203050058)
Raj Gite (Roll No. : 203050092)
Smit Gangurde (Roll No. : 203050108)
Yavnika Bhagat (Roll No. : 203050041)

(Academic Year: 2020-2021)

1 Introduction

The human brain is very good at image recognition. We do a lot of image recognition daily without thinking much about it. Yet we cannot describe the process it takes to identify objects in an image. These kinds of tasks are the classic applications of neural networks. The motivation behind this project is to study the techniques of Image classification and compare them. The main task here is to identify the type of Indian dance form from an Image.

1.1 Image recognition

The main steps involved in image recognition are:

1. Feature Extraction: Extract pixel features from an image
2. Classification: Train the model to be able to categorize images using those features

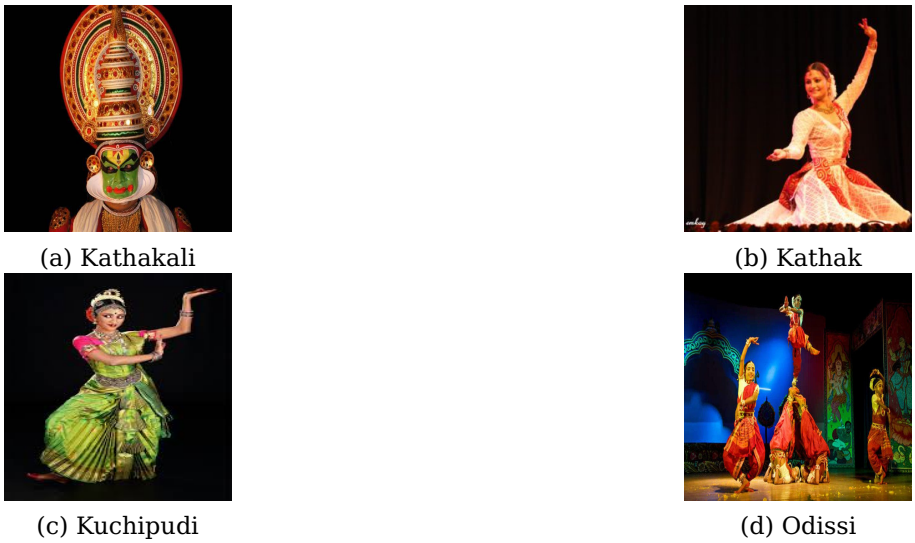
2 Dance form classes

We are going to classify the images into eight dance forms.

- Manipuri
- Bharatanatyam
- Odissi
- Kathakali
- Kathak
- Sattriya
- Kuchipudi
- Mohiniyattam

3 Dataset

The dataset was found on Kaggle [4]. It has 500+ images of dance poses. Divided into train set of 364 images and a test set of 156 images. Two files train.csv and test.csv map the image names to their class. Few sample images are shown in figure 1



(a) Kathakali (b) Kathak

(c) Kuchipudi (d) Odissi

Figure 1: Sample images from the dataset

For the training set to be effective, the distribution of classes in the training set should be uniform. The distribution of the data is shown in figure 2.

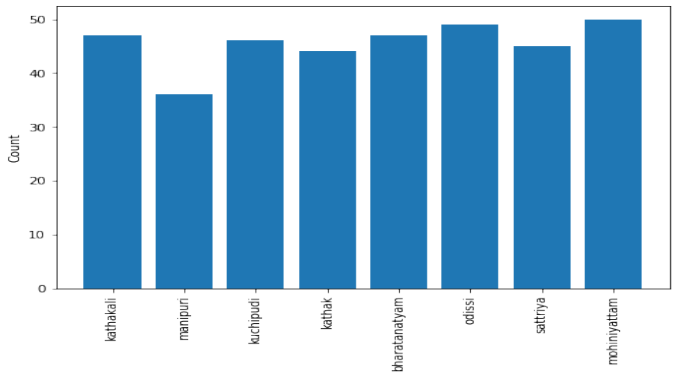


Figure 2: Train Data distribution

4 Techniques Explored

We used the following three techniques to solve the same problem. We compare the techniques in terms of train and test accuracy in the end.

- **Convolution Neural Network**
- **VGG 16**
- **Resnet**

5 Convolution Neural Network

Custom CNN architectures can be used to specifically tailor feature extraction for a task. However, since, these models need to be trained from scratch, more resources are required.
7-fold Cross Validation was done using 6 different models. Each fold containing 52 images from the Train set(364 images). Open Source library Pytorch was used for this task.

5.1 Model Architectures

Different CNN architectures were used for 7-fold Cross Validation. Fig.3 shows the different architectures used. Variations of these models with *Fully Connected* layers swapped with *SVM* were also analysed. Following common parameters were used for all these architectures.
Filter Size=(5, 5), Padding=(1, 1), Stride=(1, 1), Optimizer=*Adam*, Learning Rate=0.001, Loss Function: *Negative Log Likelihood Loss*
For SVM: Kernel Function: *Radial Basis Function*, Gamma=0.001 .

Model1	Input->Output	Number of filters	Model2	Input->Output	Number of filters	Model3	Input->Output	Number of filters
Conv2d (ReLU)	{224x224x3}->{222x222x6}	6	Conv2d (ReLU)	{224x224x3}->{222x222x4}	4	Conv2d (ReLU)	{224x224x3}->{222x222x6}	6
Conv2d (ReLU)	{222x222x6}->{220x220x6}	6	Conv2d (ReLU)	{222x222x4}->{220x220x4}	4	Conv2d (ReLU)	{222x222x6}->{220x220x6}	6
Maxpool	{220x220x6}->{216x216x6}	6	Maxpool	{220x220x4}->{216x216x4}		Maxpool	{220x220x6}->{216x216x6}	
Fully Connected (ReLU, Dropout(p=0.25))	{279936}->{1024}		Conv2d (ReLU)	{216x216x4}->{214x214x4}	4	Conv2d (ReLU)	{216x216x6}->{214x214x6}	6
Fully Connected (LogSoftmax)	{1024}->{8}		Conv2d (ReLU)	{214x214x4}->{212x212x4}	4	Conv2d (ReLU)	{214x214x6}->{212x212x6}	6
			Maxpool	{212x212x4}->{208x208x4}		Maxpool	{212x212x6}->{208x208x6}	
			Fully Connected (ReLU, Dropout(p=0.5))	{173056}->{1024}		Fully Connected (ReLU, Dropout(p=0.25))	{259584}->{1024}	
			Fully Connected (LogSoftmax)	{1024}->{8}		Fully Connected (LogSoftmax)	{1024}->{8}	

Figure 3: Custom CNN architectures

5.2 Results

Each model was cross validated with 100 epochs. *Model1* performed relatively the best amongst all models, with average cross validation accuracy of 44.8%. Fig.4 shows average accuracy and average weighted F1 score for all models. .

Model	Avg. Cross Validation Accuracy (%)	Avg. Weighted F1 Score (%)
Model1	44.8	44.1
Model2	37.1	36.3
Model3	37.6	37.4
Model1->SVM	36.5	35.7
Model2->SVM	31.3	30.8
Model3->SVM	31.9	30.5

Figure 4: Results of 7-fold cross validation

Finally training *Model1* for 200 epochs resulted in **Train Validation Accuracy: 51.92%**, **Test set Accuracy: 46.79%** and **Weighted F1 score: 46.99%**. Fig.5 shows confusion matrix for *Model1* on Test set.

.

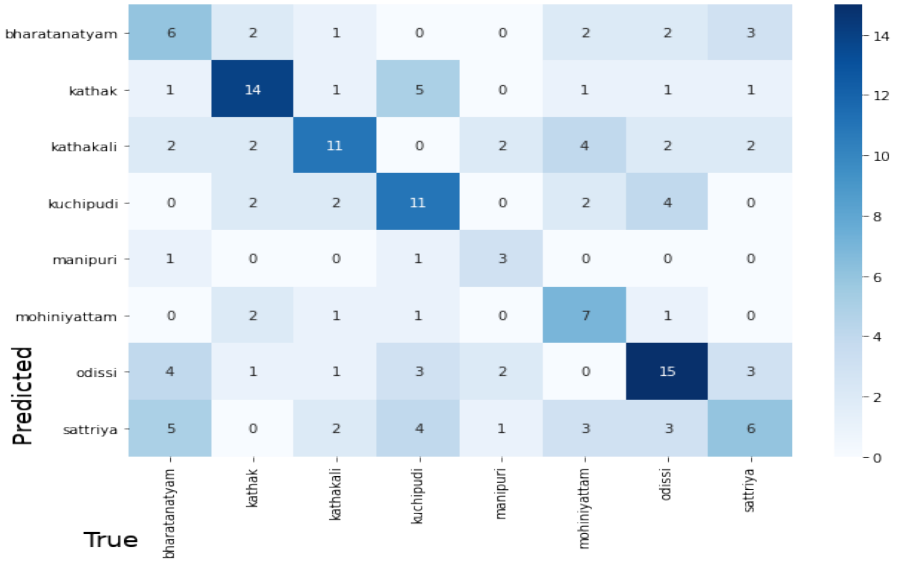


Figure 5: Model1 Confusion Matrix on Test set

6 VGG 16

VGG16 is a classic Image Classification CNN model which was built with a motive to check whether increasing the number of convolution layers do really help extract better features and serve good for classifying the images. And it turns out that this was true and after that many new models were brought up having even more layers than VGG. It gained attention when it won the 2014 ImageNet Large Scale Visual Recognition Challenge wherein the data-set had 1000 classes.

6.1 Model Architecture

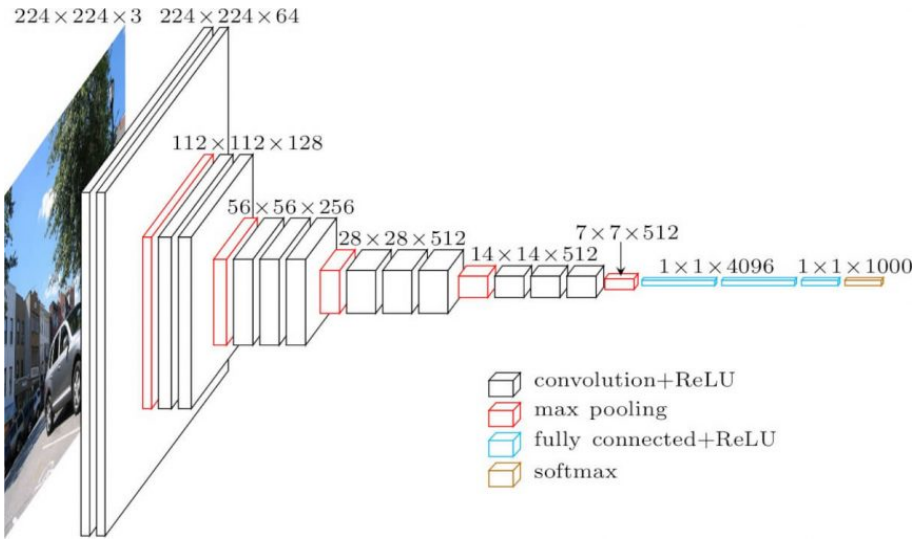


Figure 6: VGG16 Architecture

Fig.6 describes the *VGG16* architecture. It has two parts, first part consists of 13 conv layer acting as feature extracter giving 512 feautres as output. The second part comprises of Feed Forward Neural Network having 2 hidden layers each of 4096 neurons and 1 output layer with 1000 neurons.

The input to conv1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 3×3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations, it also utilizes 1×1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixel window, with stride 2.

Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks.

For our problem we applied the concept of *Transfer Learning* wherein the learning's of a model can be transferred from one problem to another. We are using the pre-computed weights of the feature extractor part where the VGG16 was trained on ImageNet dataset having 1000 classes each describing things one can see in daily life. The Feed Forward Neural Net was trained by us and has same number of layers as that of classic model just changes in the neurons (first hidden layer had 512 and second had 128), while the output layer with 8 neurons. All hidden layers are equipped with the rectification (ReLU) non-linearity.

6.2 Results

Model	Train Accuracy	Validation Accuracy	Test Accuracy
Using Augmentation	79%	73%	32%
No Augmentation	97%	68%	48%

Figure 11: Accuracy of various models

Augmentation, in which the size of dataset is increased by applying transformations (rotation, flips, etc) is used in one of the model. But as one can see in Fig. 11, it doesn't always help as the test set

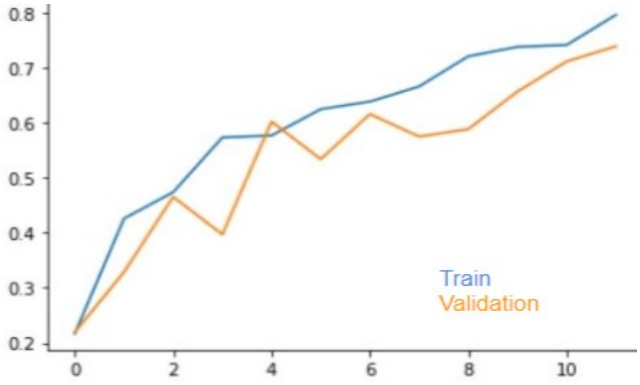


Figure 7: Accuracy using Augmentation

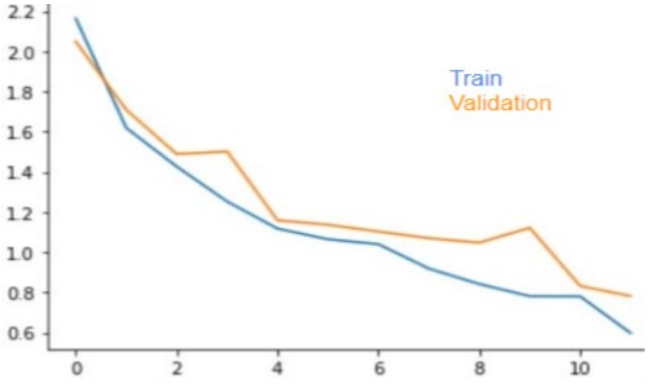


Figure 8: Loss using Augmentation

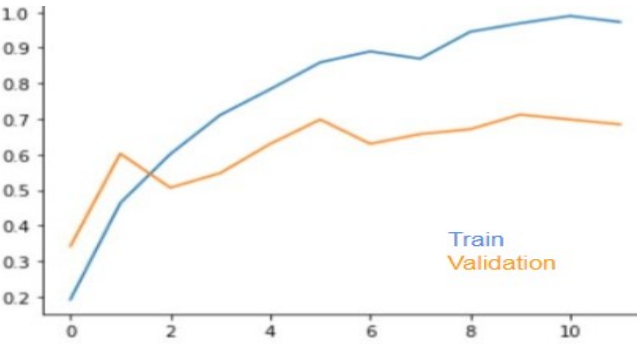


Figure 9: Accuracy without using Augmentation

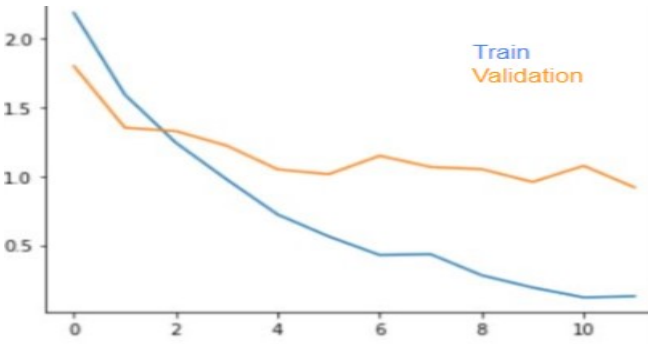


Figure 10: Loss without using Augmentation

doesn’t have any images that are transformations of train set. Thus we selected the model which isn’t using augmentation and got accuracy of 48%. Fig. 12 and Fig. 13 are the confusion matrix when both the models were tested on the test set.

.

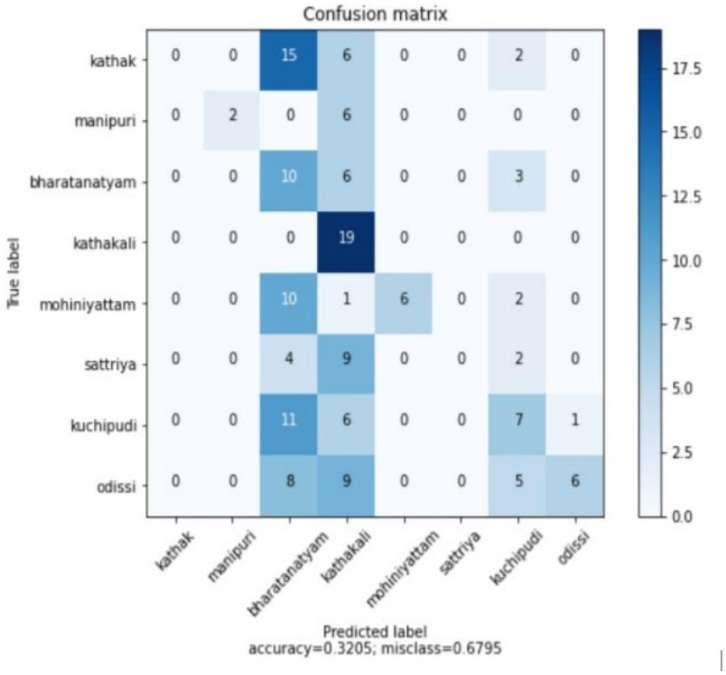


Figure 12: Confusion matrix for model using augmentation

.

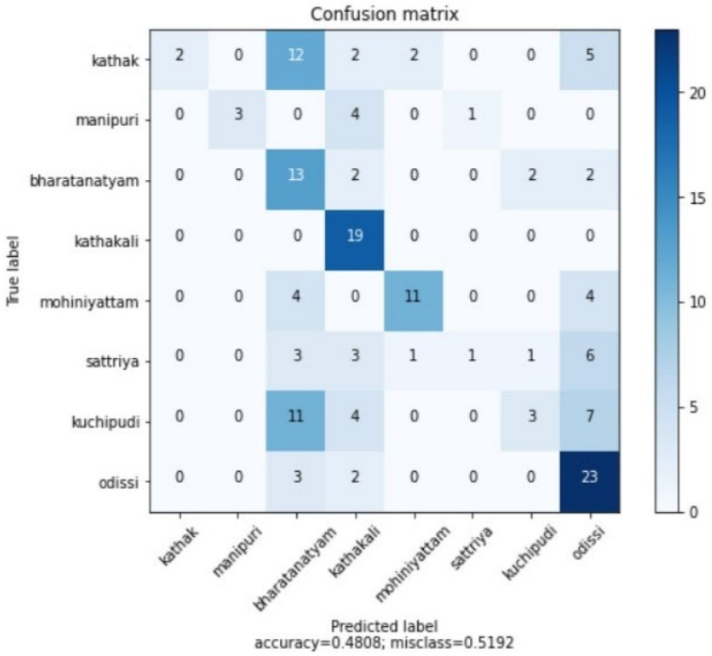


Figure 13: Confusion matrix for model without augmentation

7 Resnet

There is a common trend in the research community that if our network architecture goes deeper, the model gets more capability to learn complex features, but it also leads to problems like vanishing/exploding gradients. When deeper networks are able to start converging, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly as shown in Figure 14. Unexpectedly, such degradation is not caused by over-fitting, and adding more layers to a suitably deep model leads to higher training error.The degradation (of training accuracy) indicates that not all systems are similarly easy to optimize.

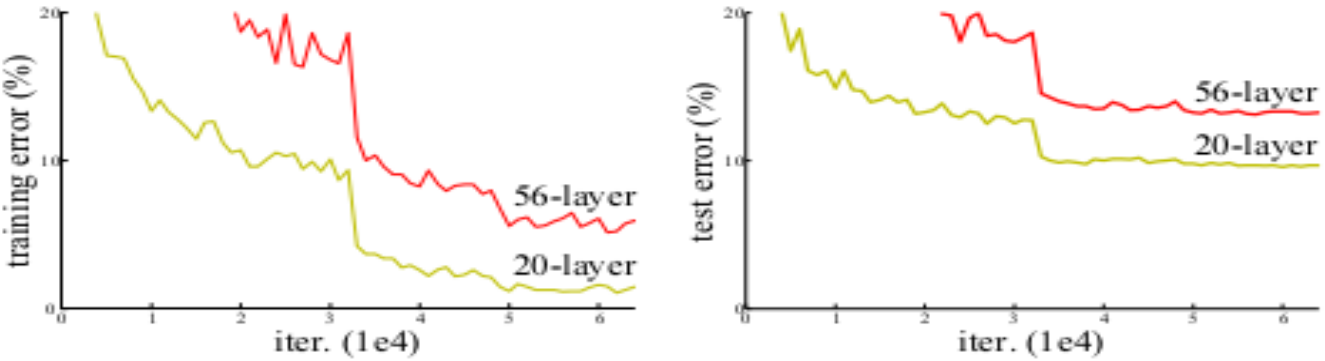


Figure 14: Error Graphs

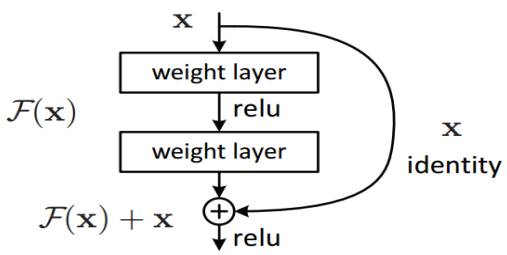


Figure 15: Identity in RESNET

One major quality of **RESNET** i.e. Residual Network is that along with being a deep neural network it also combats the problem of vanishing/exploding gradients which gives us a win-win situation. The solution that **RESNET** follows is by construction to the deeper model: the added layers are identity mapping, and the other layers are copied from the learned shallower Model as shown in Figure 15. The existence of this constructed solution indicates that a deeper model should produce no higher training error than its shallower counterpart.

7.1 MONK

MonkAI is an open source framework for fine tuning Deep Neural Networks using Transfer Learning. Monk makes image classification easy due to the following reasons:

- Write less code and create end to end applications.
- Learn only one syntax and create applications using any deep learning library - pytorch, mxnet, keras, tensorflow, gluon etc
- Manage your entire project easily with multiple experiments

Gluon is a library which makes it easy to prototype,build and train deep learning models without sacrificing training speed.

7.1.1 Implementation

We have used the Monk framework for transfer learning of various models. Table in fig 16 shows the results observed with different models. We have used 4 fold cross validation to check the effectiveness of different models. .

Experiment Name	Number of Epochs	Batch size	Train Accuracy(%)	Train Error	Validation Accuracy(%)	Validation Error	Test Accuracy(%)
Model_VGG19_unfreeze_base_pretrained	10	4	87.4	0.941	72.25	1.186	67.94
Model_Resnet18_unfreeze_base_pretrained	10	4	93.2	0.269	75.82	0.767	70.51
Model_Resnet50_unfreeze_base_pretrained	10	4	94.6	0.215	76.23	0.749	74.8
Model_Resnet101_unfreeze_base_pretrained	10	4	97.4	0.104	78.02	0.727	77.23
Model_Resnet152_unfreeze_base_pretrained	10	4	98.8	0.028	86.81	0.501	81.4

Figure 16

We observed that among the different pretrained models used, RESNET152 worked the best.

7.2 RESNET 152

On the Image Net dataset, we evaluate residual nets with a depth of up to 152 layers which is 8 times deeper than VGG nets but still having lower complexity. RESNET152 is a very deep neural network which works on millions of parameters and has the capability to learn much more complex features.

7.3 Result

Confusion Matrix(Test) : Model RESNET152 .

PREDICTED

TRUE	Dance Forms	bharatanatyam	kathak	kathakali	kuchipudi	manipuri	mohiniyattam	odissi	sattriya
	bharatanatyam	13	0	0	2	0	0	0	0
	kathak	0	20	0	2	0	0	0	1
	kathakali	0	0	18	1	0	0	0	0
	kuchipudi	4	0	0	15	0	0	3	0
	manipuri	1	0	0	1	8	1	0	0
	mohiniyattam	0	1	0	0	0	16	1	0
	odissi	1	0	0	1	0	2	24	1
	sattriya	0	2	1	3	0	0	0	13

Figure 17

8 Conclusion

8.1 Observations

- **Custom CNNs:** Can be made to cater to a specific task, but requires more resources.
- **VGG16:** Pre-trained weights can be used for *Transfer Learning*, saving resources.
- **ResNet:** Combats *Vanishing Gradient* problem. 8 times deeper than VGG nets, but still has less complexity.

8.2 Comparison

Accuracy on the same Test set is used to compare the various methods that were explored. Fig.18 shows the average validation accuracy and final Test accuracy of each method. .

Method	Validation Accuracy (%)	Test Accuracy (%)
Custom CNN (Model1)	44	46.7
VGG16	68	48
ResNet152	79.4	77.56

Figure 18: Comparison of methods

It is evident that ResNet152 performed the best amongst all the explored techniques. This can be attributed to the depth of the network allowing it to learn more complex features from an image and the identity connections.

8.3 Code Repository

`https://git.cse.iitb.ac.in/smitgangurde/CS725_Project`

References

[1] *Image classification with pytorch*. URL: `https://github.com/LeanManager/PyTorch_Image_Classifier/blob/master/Image_Classifier_Project.ipynb`.

[2] *Monk*. URL: `https://clever-noyce-f9d43f.netlify.app/#/introduction`.

[3] *Pytorch Docs*. URL: `https://pytorch.org/docs/stable/index.html`.

[4] Uday Singh. *Deep Learning challenge: Identify the dance form*. URL: `https://medium.com/analytics-vidhya/identification-of-indian-dance-form-using-tenforflow2-0-and-keras-4341b4c69526`.

[5] Dennis T. *Confusion Matrix Visualization*. URL: `https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea`.

[6] Tessellate-Imaging. *Monk - A computer vision toolkit for everyone*. URL: `https://github.com/Tessellate-Imaging/monk_v1`.