COL341: Assignment 3 Report

Team: Red Devils- Shubham Sarda (2018TT10958), Ravindra Kudchadkar (2018TT10941)

Week 1

We started with inbuilt CNN models in pytorch. Code for model initialization and default augmentation was checked out from the following two websites:

https://pytorch.org/hub/pytorch_vision_resnet/

https://pytorch.org/vision/stable/models.html

Train images (29011) were divided into training and validation sets of 26000 and 3011 respectively. Model was trained on training set and validation accuracy were used for feedback and flagging of overfitting. Ideally K-fold cross validation should've been performed but it was avoided due to limited GPU access time.
Resnet 18 (not pretrained) was trained on the model with a learning rate of 1e-3 and batch size of 500 with Adam optimizer. Last layer of model was replaced with a fully connected layer of 19 units to cater to 19 asana classification problem. Improvement in validation accuracy took large amounts of time. Even after 15 epochs (>1 hour), validation accuracy reached as high as 14%. Due to the time constraints of the problem, this approach was avoided in future experiments.

Experiments with pretrained Models:

Note: bs=batch size, lr=learning rate. Adam optimizer, cross entropy loss and train validation split of 26000:3011 has been used for all experiments unless otherwise stated.

- We again started with resnet 18 initialized with pretrained imagenet weights.
  Training Parameters: Adam optimizer, lr=.0001, bs=500, 10 epochs
  Results: Validation Accuracy: 0.9216, Public Test accuracy**: 0.4826**
- We then tried a very recent model: Efficient Net B4. The following website/git repository was used for the installation of efficient net library and learning model initialization code:
  https://github.com/lukemelas/EfficientNet-PyTorch
  Training Parameters: Adam optimizer, lr=.0001, bs=50, 7 epochs
  Batch size was reduced to accommodate the larger model within bounds of GPU Memory
  Results: Train loss: 0.04441 test accuracy: 0.932956, Public Test accuracy: **0.6857**

It was realized that though validation accuracy is a good way to flag overfitting, it can't be used as a criteria to judge performance on public test set since it has a different camera angle. It is evident from the fact that even though ResNet18 and EfficientNetB4 have similar validation accuracies, still they give very different performance on the public test set. It further becomes clear when we look at the results of experiments given below.

- Following data augmentation has been added in previous experiment:
  img_transform=transforms.Compose([transforms.ToPILImage(),transforms.Resize(256), transforms.RandomChoice([transforms.RandomRotation(10),transforms.RandomAffine(0, translate=(.1,.1))]), transforms.CenterCrop(224), transforms.ToTensor(), transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])
  bs=50, Epoch: 6→ Train loss: 0.13624 Validation accuracy: 0.92131

  Accuracy at different epochs were noted to find optimal number of epochs which ensures no

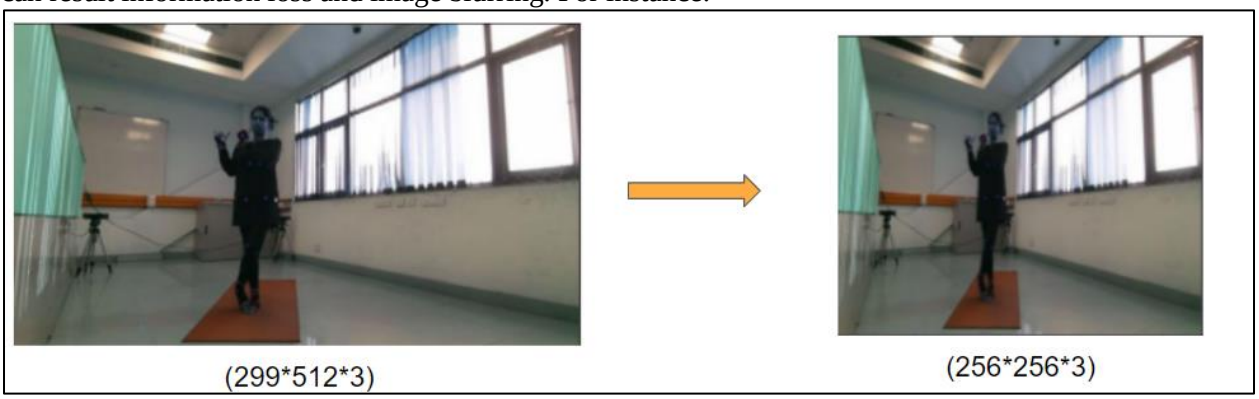
overfitting.
Epoch: 8 → Train loss: 0.0452 test accuracy: 0.9336

Image resizing and center cropping have been carry forwarded through the next experiments as well to avoid GPU memory overflow, unless otherwise specified.

- LR was changed to 1e-5, keeping other things same.
  Epoch: 7→ Train loss: 0.08442 validation accuracy: 0.9148

- Then, pretrained EfficientNet B3 was used. As model size is small, batch size of 80 was used.
  Epoch: 12 Train loss: 0.02101 validation accuracy: 0.9406, Public test accuracy: 0.655

- Epoch: 25 → Train loss: 0.01469 test accuracy: 0.9296, Public Test: .6773
- Now, Efficient Net B5 model was tested with pretrained imagenet weights. Batch size had to be reduced to 40 due to larger model size.
  Adam optimizer with lr=1e-4 was used
  Epoch: 10 → Train loss: 0.0329 validation accuracy: 0.9351 public test accuracy : 0.6607
- Until now, image was resized to (256*256*3), followed by center cropping of (224*224*3). This can result information loss and image blurring. For instance:

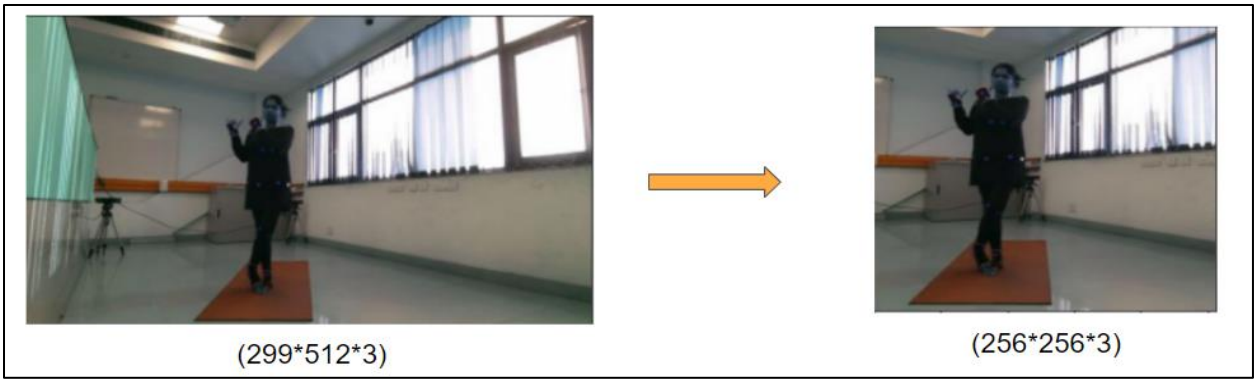


To avoid this, instead of resizing, image was directly cropped using the following slicing to make it 299*299*3, followed by further center cropping to 224*224*3

Image=Image[: ,106:106+299,:].
Thus, horizontally image was cropped form center
Efficient Net B4 was used with lr=1e-4, bs=50 and adam optimizer.
Validation accuracy: 0.9407, Public test accuracy: .6

- A different cropping was used this time: Image = Image[-256:,128:128+256,:]
  It was observed that human is in the center of the image, thus horizontally, image was center cropped to include 256 pixels. Vertically, human is generally positioned in lower part of the image, hence the bottom 256 pixels were considered in vertically. Image cropping also gives the additional advantage of removing background which doesn't contain any information related to pose/asana, thus, helping in noise removal.
  Epoch: 9 → Train loss: 0.0225 Validation accuracy: 0.9384 Public Test Accuracy 0.69033

Thus, image cropping helps in improving the results.



- Next, DenseNet201 was tested using same cropping as done before,
  Adam optimizer, lr=1e-4,bs=50
  Epoch: 5 → Train loss: 0.0969 Validation accuracy: 0.9342 Public test accuracy: .67

  Epoch: 15 → Train loss: 0.0369 Validation accuracy: 0.9442 Public Test acc=.646
- Several different data augmentation options can be found on the following website:
  https://pytorch.org/vision/stable/transforms.html

  We've used Random Perspective data augmentation since it helps to learn asanas from different perspective/view. This is exactly what we want since test images are taken from a different angle. Base model of Efficient Net B4 has been used. Adam optimizer with lr=1e-4, batch size=50 is Employed.
  10 Epochs→ Train loss:.09, Validation accuracy: .9355, Public Test Accuracy: .697
  Hence Random Perspective helps to improve results.
- Imagenet AutoAugmentation was used keeping everything else same from previous experiment, with a 90:10 train-validation split
  Epoch: 15→Train loss: 0.090 Validation accuracy: 0.9325 Public Test Accuracy: 0.694

Efficient Net B0, B1 and B2 were also tried but, they didn't give good results on public test set.

Key Point Detection Followed by Classification

- Movenet was used to find 17 key-points over the body. Specifically, the movenet thunder model was used. Following links have been very helpful in understanding algorithm and practical application of the same:

  https://analyticsindiamag.com/how-to-do-pose-estimation-with-movenet/

  https://blog.tensorflow.org/2021/08/pose-estimation-and-classification-on-edge-devices-with-MoveNet-and-TensorFlow-Lite.html

  https://blog.tensorflow.org/2021/05/next-generation-pose-detection-with-movenet-and-tensorflowjs.html

  Following key-points have been identified for the images:

```
# Dictionary to map joints of body part
KEYPOINT_DICT = {
    'nose':0,
    'left_eye':1,
    'right_eye':2,
    'left_ear':3,
    'right_ear':4,
    'left_shoulder':5,
    'right_shoulder':6,
    'left_elbow':7,
    'right_elbow':8,
    'left_wrist':9,
    'right_wrist':10,
    'left_hip':11,
    'right_hip':12,
    'left_knee':13,
    'right_knee':14,
    'left_ankle':15,
    'right_ankle':16
}
```

Thus we get a (17,2) numpy array for each image since x,y coordinates of the key points are found. This is then flattened into a (34,) array and fed as features into a Xgboost Classifier. This was attempted to make the techniques independent of camera angle, yet it didn't give satisfactory results.

Validation Accuracy: .87 Public Test Accuracy: .4

- In an attempt to improve the results, feature engineering was employed. Total of 24 different Angles were feature engineered using the keypoints. For instance, angle at joints like elbow, Knee, angle between elbow, should and hips etc. for both left and right side of the body. This al so didn't give good results.

Validation Accuracy:0.74 Public Test Accuracy: .41

Following website was used to directly find the angle given three 2d coordinates:
https://manivannan-ai.medium.com/find-the-angle-between-three-points-from-2d-using-python-348c513e2cd

It basically finds the 2 vectors and then cosine between the two using normalized dot product.

Keras Models

Following extensive testing with pytorch models, keras was used next due to the easy of modifying the top fully connected network after the base CNN model. (It was later realized that, it could've done pretty easily with torch as well, but keras was continued to be used because of higher familiarity with the library :p.).

Following links have been helpful in using pretrained keras models & creating a custom data loader

https://keras.io/api/applications/efficientnet/
https://keras.io/guides/transfer_learning/
https://keras.io/api/applications/
https://towardsdatascience.com/implementing-custom-data-generators-in-keras-de56f013581c

https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly

- Efficient net b4 was used with include top=false, thus the last layer was removed. Instead of that a global pool average layer was added followed by Fully Connected layer of 19 neurons

  Epoch 10, lr=1e-4,bs =40
  train loss: 0.0203, train acc: 0.9934 val_loss: 0.2504, val_acc: 0.9427
  Test Score: 0.68
- Next, data augmentation was used:
  Rotation range=20, width shift range=0.2, height shift range=0.2, shear range=0.2, zoom range=0.2

  Public Test Accuracy: 0.705

  Thus, data augmentation helps to improve results (as seen before).

- Now the augmentation parameters were varied to find the optimal values
  Rotation range=20, width shift range=0.2, height shift range=0.2, shear range=0.2, zoom range=0.2, horizontal_flip=True, vertical_flip=True

  Global Average Pool Layer was also changed to Global Max Pool layer

  Epoch 10→ train loss: 0.3230, train accuracy: 0.8938 , val_loss: 0.3118, val_accuracy: 0.8970
  Public Test Accuracy: 0.709

- Rotation range=10, width shift range=0.1, height shift range=0.1, shear range=0.1, zoom range=0.1, horizontal_flip=True, vertical_flip=True

  Global Average Pool Layer was also changed to Global Max Pool layer again.
  Public Test Accuracy: 0.701
- Now, multiple changes were made:
  Efficient Net B4
  Rotation range=10, width shift range=0.1, height shift range=0.1, shear range=0.1, zoom range=0.1, horizontal_flip=True, vertical_flip=True
  Lr=5e-4, bs=40, learning rate scheduler was used which multiples lr by a factor of 0.9 after every epoch.
  FCN after the CNN model was also expanded.
  Global Max Pool 2D →FC layer (1024 units, relu activation) →FC layer (256 units, relu activation) →FC layer (19 units, softmax activation)
  Epoch 10 → train loss: 0.079, train accuracy: 0.968, validation loss: 0.19, validation accuracy: 0.938, Public Test Accuracy: **0.75**

  Thus, increasing Learning rate helps improve results on public test set, even though validation scores are nearly same.
- Now, a new model Xception Net (pretrained) was been used.

  Rotation range=10, width shift range=0.1, height shift range=0.1, shear range=0.1, zoom range=0.1, horizontal_flip=True, vertical_flip=True
  Lr=5e-4, bs=40, learning rate scheduler was used which multiples lr by a factor of 0.9 after every epoch.

FCN after the CNN model was also expanded.
Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activ
ation)

Public Test Accuracy: **0.76**

- Finally, Xception net is used with:
  Rotation range=15, width shift range=0.15, height shift range=0.15, shear range=0.15, zoom range=0.15, horizontal_flip=True, vertical_flip=True
  10 epochs, Lr=6e-4, bs=40, learning rate scheduler was used which multiples lr by a factor of 0.9 after every epoch.
  FCN after the CNN model was also expanded.
  Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activ
  ation)

```
Epoch 10/10
650/650 [==============================] - 570s 868ms/step - loss: 0.1284 - acc: 0.9519
- val_loss: 0.2043 - val_acc: 0.9303
```

Public Test Accuracy: 0.776→ Best Result of Week 1

Week 2

- Xception net is used with:
  Rotation range=20, width shift range=0.20, height shift range=0.20, shear range=0.2, zoom range=0.2, horizontal_flip=True, vertical_flip=True
  10 epochs, Lr=6e-4, bs=40, learning rate scheduler was used which multiples lr by a factor of 0.9 after every epoch.
  FCN after the CNN model was also expanded.
  Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activ
  ation)
  Public Test Acc:0.72

- With the best submission of week 1, additional brightness augmentation was done
  Public Test Acc:0.746
- Efficient Net B5 was trained with:
  Rotation range=15, width shift range=0.15, height shift range=0.15, shear range=0.15, zoom range=0.15, horizontal_flip=True, vertical_flip=True
  5 epochs, Lr=6e-4, bs=24, learning rate scheduler was used which multiples lr by a factor of 0.9 a
  fter every epoch. (BS had to be reduced due to larger size of model)
  FCN after the CNN model was also expanded.
  Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activ
  ation)
  Public Test Accuracy: 0.776

```
Epoch 5/5
1083/1083 [==============================] - 993s 912ms/step - loss: 0.2592 - acc: 0.909
8 - val_loss: 0.2507 - val_acc: 0.9067
```

- Keeping everything same, and changing bs to 50, efficient Net b3 was used: Public Test Acc:0.72

- Keeping everything same, and changing bs to 150, MobileNetV3 large was used:
  Public Test Acc:0.7095 with 8 epochs
- Keeping everything same, and changing bs to 40, InceptionNetV3 was used:
  Public Test Acc:0.729 with 8 epochs
- Keeping everything same, and changing bs to 40, DenseNet was used:
  Public Test Acc:0.757 with 8 epochs

Experimenting with Vision Transformers

Helpful Links: https://github.com/lucidrains/vit-pytorch, https://github.com/lukemelas/PyTorch-Pretrained-ViT

Experimented with pretrained Vision Transformers for instance:

1. ViT('B_32_imagenet1k', pretrained=True,num_classes=19).cuda()
2. model = DistillableViT(image_size = 256, patch_size = 32, num_classes = 19, dim = 1024, depth = 6, heads = 8, mlp_dim = 2048, dropout = 0.1, emb_dropout = 0.1)

   Several Other models were also tried but did not yield good results. Public Test Acc:0.74-0.75 was achieved.

Classification with Segmented Images

Here we first segmented the images using pretrained Xception based model from pixellib:
https://pixellib.readthedocs.io/en/latest/Image_pascal.html

Segmented Image

Original Image

The purpose of segmentation is to separate the human from the background since background is useless/ acts as noise during classification. This is because type of asana solely depends on the human.

Thus, all images were segmented and then classified using xception net with all the parameters etc same From the week 1 submission. Public Test Accuracy of 0.752 was achieved.

Next, we also did another preprocessing operation of exchanging the human pixels of segmented images with pixels from original image and the background was changed to white to avoid confusion with the Subject outfit. For instance:

These images were again classified with the help of xception net. Public Test Accuracy of 0.74 was obtained.

Ensembling

Ensembling is a very common technique to improve results. We've used bagging mode of ensembling with a slight modification. Instead of taking mode of all predictions by different models, we add the softmax output [(19,) array] given by each model. Thus, basically we're adding the probabilities of all 19 asanas given by different models. We then take argmax to find the asana with highest probability and return that as the output. We've used the following three models:

1. Xception net is used with:
Rotation range=15, width shift range=0.15, height shift range=0.15, shear range=0.15, zoom range=0.15, horizontal_flip=True, vertical_flip=True
10 epochs, Lr=6e-4, bs=40, learning rate scheduler was used which multiples lr by a factor of 0.9 after every epoch.
FCN after the CNN model was also expanded.
Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activ ation)

2. Efficient net B4 is used with:
Rotation range=15, width shift range=0.15, height shift range=0.15, shear range=0.15, zoom range=0.15, horizontal_flip=True, vertical_flip=True
10 epochs, Lr=6e-4, bs=40, learning rate scheduler was used which multiples lr by a factor of 0.9 after every epoch.
FCN after the CNN model was also expanded.
Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activ ation)
3. Efficient net B5 is used with:
Rotation range=15, width shift range=0.15, height shift range=0.15, shear range=0.15, zoom range=0.15, horizontal_flip=True, vertical_flip=True
5 epochs, Lr=6e-4, bs=25, learning rate scheduler was used which multiples lr by a factor of 0.9 a fter every epoch.
FCN after the CNN model was also expanded.

Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activ ation)

Important Note:

In order to use multiple models, we've to clear GPU memory. Clearing GPU memory is not a very efficient solution as it can lead to problems later when GPU is again allocated memory. A very elegant way to get around this problem is using multiprocessing and threading.

```
import multiprocessing
p = multiprocessing.Process(target=train)
p.start()
p.join()
```

We create train functions for each model, which basically initializes and trains the model. It also saves the model weights after training. Process function of multiprocessing creates a separate thread for the train Function which is destroyed once the function is run. Thus, we can do this for all models without the need of clearing GPU memory and thus causing potential errors.

https://docs.python.org/3/library/multiprocessing.html

Public Test Accuracy: 0.807 → Week 2 Best Result

Week 3

- Week 1 submission (Xception Net) was trained without decaying the learning rate (6e-4):

  10 Epochs→ Public Test accuracy= 0.7548
- Week 1 submission (Xception Net) was trained decaying factor of lr as 0.95 (lr=6e-4):

  8 Epochs→ Public Test accuracy=0.776
  4. Xception net is used with:
  Rotation range=17, width shift range=0.17, height shift range=0.17, shear range=0.17, zoom range=0.17, horizontal_flip=True, vertical_flip=True
  10 epochs, Lr=5e-4, bs=40, learning rate scheduler was used which multiples lr by factor of 0.95 after every epoch.
  FCN after the CNN model was also expanded.
  Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activ ation)
  10 Epochs→Public Test accuracy=0.756
- Xception net is used with:
  Rotation range=20, width shift range=0.20, height shift range=0.2, shear range=0.2, zoom range=0.2, horizontal_flip=True, vertical_flip=True
  10 epochs, Lr=5e-4, bs=40, learning rate scheduler was used which multiples lr by factor of 0.95 after every epoch.
  FCN after the CNN model was also expanded.

Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activ ation)
10 Epochs→Public Test accuracy=0.744

- Xception net is used with:
  Rotation range=20, width shift range=0.20, height shift range=0.2, shear range=0.2, zoom range=0.2, horizontal_flip=True, vertical_flip=True
  9 epochs, Lr=4e-4, bs=40, learning rate scheduler was used which multiples lr by factor of 0.95 af ter every epoch.
  FCN after the CNN model was also expanded.
  Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activ ation)
  9 Epochs→Public Test accuracy=0.748
- Xception net is used with:
  Rotation range=18, width shift range=0.18, height shift range=0.18, shear range=0.18, zoom range=0.18, horizontal_flip=True, vertical_flip=True
  9 epochs, Lr=6e-4, bs=40, learning rate scheduler was used which multiples lr by factor of 0.95 af ter every epoch.
  FCN after the CNN model was also expanded.
  Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activ ation)
  10 Epochs→Public Test accuracy=0.758

Tuning the optimizer algorithm

Keeping other things same as in week 1 submission:

- Stochastic Gradient Descent (lr=6e-4)
  9 Epochs→Public Test accuracy=0.73

- RMSprop(learning_rate=6e-4)
  10 Epochs→Public Test accuracy=0.753

- Adamax(learning_rate=6e-4)
  10 Epochs→Public Test accuracy=0.728

Rest of the algorithms were also tried like Nadam etc, but didn't improve results, hence we stick with Adam optimizer.

Loss function was changed to Kullback-Leibler divergence loss

- Xception net is used with:
  Rotation range=20, width shift range=0.2, height shift range=0.2, shear range=0.2, zoom range=0.2, horizontal_flip=True, vertical_flip=True
  10 epochs, Lr=7e-5, bs=40, learning rate scheduler was used which multiples lr by factor of 0.95 after every epoch.
  FCN after the CNN model was also expanded.
  Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activ ation)
  10 Epochs→Public Test accuracy=0.75
- Xception net is used with:
  Rotation range=20, width shift range=0.2, height shift range=0.2, shear range=0.2, zoom range=0.2, horizontal_flip=True, vertical_flip=True

10 epochs, Lr=3e-4, bs=40, learning rate scheduler was used which multiples lr by factor of 0.95 after every epoch.
FCN after the CNN model was also expanded.
Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activation)
10 Epochs→Public Test accuracy=0.743

- Now, the FCN after the CNN model was made even deeper by adding multiple FC layers. (these changes are done to model in week 1 submission)
  Public Test Accuracy: 0.755
- Densenet201 net is used with:
  Rotation range=15, width shift range=0.15, height shift range=0.15, shear range=0.15, zoom range=0.15, horizontal_flip=True, vertical_flip=True
  5 epochs, Lr=5e-4, bs=40, learning rate scheduler was used which multiples lr by factor of 0.95 after every epoch.
  FCN after the CNN model was also expanded.
  Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activation)
  5 Epochs→Public Test accuracy=0.763

Note:

We observe that even 5 epochs can be used to get good performance. Now, initially in week 2 submission only 3 models were used in ensemble due to 5-hour training time constraint. There, we had trained for 10 epochs. If we reduce to 5 epochs, we can potentially use more models for ensembling and get even better performance on public test set. Thus we train other models for epochs as well.

- InceptionNetV3 net is used with:
  Rotation range=15, width shift range=0.15, height shift range=0.15, shear range=0.15, zoom range=0.15, horizontal_flip=True, vertical_flip=True
  5 epochs, Lr=5e-4, bs=40, learning rate scheduler was used which multiples lr by factor of 0.95 after every epoch.
  FCN after the CNN model was also expanded.
  Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activation)
  5 Epochs→Public Test accuracy=0.746
- EfficientNetB4 net is used with:
  Rotation range=15, width shift range=0.15, height shift range=0.15, shear range=0.15, zoom range=0.15, horizontal_flip=True, vertical_flip=True
  5 epochs, Lr=5e-4, bs=40, learning rate scheduler was used which multiples lr by factor of 0.95 after every epoch.
  FCN after the CNN model was also expanded.
  Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activation)
  5 Epochs→Public Test accuracy=0.75
- ResNet50V2 is used with:
  Rotation range=15, width shift range=0.15, height shift range=0.15, shear range=0.15, zoom range=0.15, horizontal_flip=True, vertical_flip=True

5 epochs, Lr=5e-4, bs=40, learning rate scheduler was used which multiples lr by factor of 0.95 af ter every epoch.
FCN after the CNN model was also expanded.
Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activ ation)
5 Epochs→Public Test accuracy=0.76

- Ensemble of DenseNet201, XceptionNet, EfficientNetB5, EfficientNetB4 and ResNet50v2 gives public test accuracy of **0.8099**
  All the hyperparameters of these are mentioned in the above experiments (with epochs=5)

  We observe that we don't get a significant jump in accuracy with 5 models compared to when we used 3 models. Hence, now we experiment with 4 models.

  Ensemble of DenseNet201, XceptionNet, EfficientNetB5 and ResNet50v2 →

  Public Test Accuracy: 0.8207

Weighted Ensemble Model

Now we've reached the conclusion that 4 is the optimal number of models to be used in ensembling. So, we try multiple combinations and permutation of 4 models that we'll use in the ensemble model to get best results.

Currently we're taking average of the softmax output of models followed by argmax. Instead, we can also take weighted average of outputs given by different models since different models show different efficiency individually. Thus, optimizing these weights can further help to improve results.

We tried multiple combination of models and weights and arrived at the following results:

Models Used:

- Model 1: Xception net is used with:
  Rotation range=15, width shift range=0.15, height shift range=0.15, shear range=0.15, zoom range=0.15, horizontal_flip=True, vertical_flip=True
  5 epochs, Lr=5.5e-4, bs=40, learning rate scheduler was used which multiples lr by factor of 0.95 after every epoch.
  FCN after the CNN model was also expanded.
  Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activ ation)

- Model 2: ResNet50V2 net is used with:
  Rotation range=15, width shift range=0.15, height shift range=0.15, shear range=0.15, zoom range=0.15, horizontal_flip=True, vertical_flip=True
  5 epochs, Lr=5e-4, bs=40, learning rate scheduler was used which multiples lr by factor of 0.95 af ter every epoch.
  FCN after the CNN model was also expanded.

Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activ ation)

- Model 3: EfficientNetB4 net is used with:
  Rotation range=15, width shift range=0.15, height shift range=0.15, shear range=0.15, zoom range=0.15, horizontal_flip=True, vertical_flip=True
  5 epochs, Lr=5e-4, bs=40, learning rate scheduler was used which multiples lr by factor of 0.95 af ter every epoch.
  FCN after the CNN model was also expanded.
  Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activ ation)

Model 4: EfficientNetB5 net is used with:
Rotation range=15, width shift range=0.15, height shift range=0.15, shear range=0.15, zoom range=0.15, horizontal_flip=True, vertical_flip=True
5 epochs, Lr=6e-4, bs=24, learning rate scheduler was used which multiples lr by factor of 0.9 after every epoch.
FCN after the CNN model was also expanded.
Global Max Pool 2D →FC layer (1024 units, relu activation) → FC layer (19 units, softmax activation)

Also, we used all the 29011 images for training instead of splitting the dataset to allow the models to generalize even better.

Assuming Model X gives predsX as result, the following weights have been found to be optimal:

preds=(preds1)*6.3+(preds2)*6.3+(preds3)*6.3+(preds4)*6

i.e.

Weights corresponding to each model are:

EfficientnetB5: 6
EfficientnetB4: 6.3
XceptionNet: 6.3
ResNet50V2: 6.3

Public Test Accuracy: 0.83 → Week 3 Submission

Week 4

- In the final week, we tried other combinations of weights to improve the results, but couldn't surpass the previous score on public test set.

- We also tried using augmentation to rotate/translate etc images in public test set with the that this might change the camera angle somewhat and make it similar to the camera view used in train set. (It obviously didn't work :p)

For Week 4, we submit the code & models as we did in week 3.

Key Observations

1. Validation set accuracy can't be used as marker for public test set accuracy due to different camera angle.
2. Learning rate tuning helps improve the result by largest margins
3. Preprocessing techniques such as key point detection & segmentation didn't help much
4. Weighted Ensembling almost always works & improves the results.
5. We can't keep on increasing the number of models in the ensemble model. After a certain number of models, performance deteriorates.
6. Threading is a very good way to train multiple models with limited GPU Memory.