Advantage of using a finetuned model:

The training cost is much less than training the CNN with the same structure from scratch. If the given training set is compatible with the pre-trained network, the performance may become better because the pre-trained CNN are typically trained with ImageNet with more images, so the pre-trained CNN may be more accurate than the CNN trained with a relatively smaller training set, which could also prevent overfitting.

Difference between learning from scratch and transfer learning:

Learning from scratch could give a better training accuracy, the training accuracy, and the validation loss could decrease at the same time when training loss decrease. The hyperparameters are easier to adjust. The shortcoming is that learning from scratch requires us to guess for the best structure, which requires for a long time to design, even after a long-time design, we still do not know what the best structure should be look like. Another shortcoming of training from scratch is that this process requires for a long time to train. In the beginning, I designed a complex network and train it on AWS p3.xlarge instance, it requires for nearly one hour for an epoch, while using transfer learning on MobileNetV2, it only requires for about 15 minute for an epoch, also, when training from scratch, the process requires a more powerful GPU with a larger storage, or the memory will blow up.

While transfer learning could save a lot of time of designing the structure of the network, and the parameters are pre-trained which could be used to do feature extraction, the only thing need to be done is to add the classifier and train it or retrain few layers in the end of the original network, which takes few resources to train. An issue of transfer learning is that the pre-trained network may not be compatible with the problem to be solved, because the training set used to train the pre-trained network may be different form the current training set, which could do wrongly feature extraction and thus the classification accuracy would become quite low. Another problem is that the hyperparameter in transfer learning need to be set to the value that compatible with current training set and the selected network, which need some time to choose.

From all above, the advantage of using a finetuned model is that the training process could be much easier and cost less than training from scratch.

Difference in learning rate:

The learning rate is typically smaller when using pre-trained model, because the pre-trained model is using pre-trained parameters, there is no need to change them a lot, just fine-tune will be fine. But learning from scratch requires large learning rate in the beginning, and decrease in the later epoch. Because in the beginning, the network does not fit the training set, the learning rate should be large enough to speed up, and then the learning rate should decrease to avoid overfit.

The training curve are shown in Fig. 1

Several models are trained and because the problem of overfitting always appear and the validation loss even increase after several epoch. More than 100$ is spend on tuning the structure of the network and training the models, I also subscribe the colab pro to assist my homework. The poor performance for the network in Problem 2 may be caused by the structure of the network, I did try to designed a more complex network to train, but it is too costly both in time and money. About problem 3, the training set is constructed by combination of each line in compatibility_train.txt and compatibility_valid.txt, in total, there are 1.3 million pairs of training samples and it is impossible for us to train with such a huge dataset in such a short time period, so I only use about 180,000 pairs to train my network and the validation set is reduced to 20,000. The capacity of the model is not so good that even the training accuracy just reach about 60%, the validation loss begins to increase. If the learning rate is increased, the accuracy of training set could be increased while the performance of validation set still remain almost the same, may be the simple structure cannot fully extract the feature shared by the compatibility by this reduced training set.

Table for accuracy of the last epoch:

	Finetuned net	My net	Compatibility net
Training set	78.6%	85.8%	62.0%
Validation set	69.4%	58.0%	59.3%

Learning rate scheduler is implemented by:

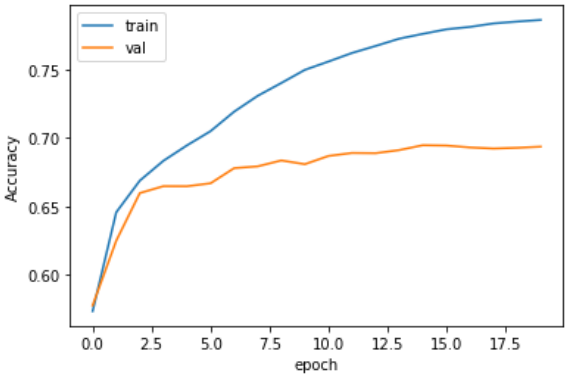
```
def scheduler(epoch):
    begin_epoch = 5
    rate = Config['learning_rate']
    if epoch < begin_epoch:
        return rate
    else:
        return rate * np.exp(0.2 * (begin_epoch - epoch))


learning_rate = tf.keras.callbacks.LearningRateScheduler(scheduler)
history = model.fit(train_generator,
                    validation_data=test_generator,
                    epochs=Config['num_epochs'],
```

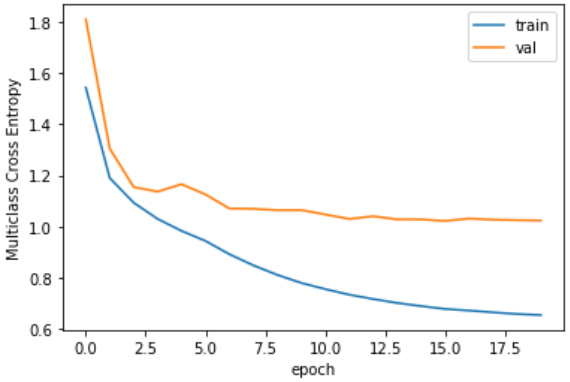
callbacks=[**learning_rate**,check_point,OnEpochEnd([train_generator.on_epoch_end])]

)

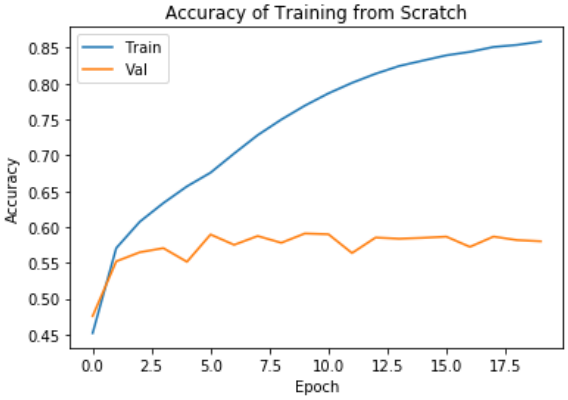
Permutation invariant feature for the entire set for compatibility prediction is learned by the randomness when getting the combination of each set, and the sequence of the combination is randomized and the first and second position of each combination is randomized again. I think this is the meaning of permutation invariant.



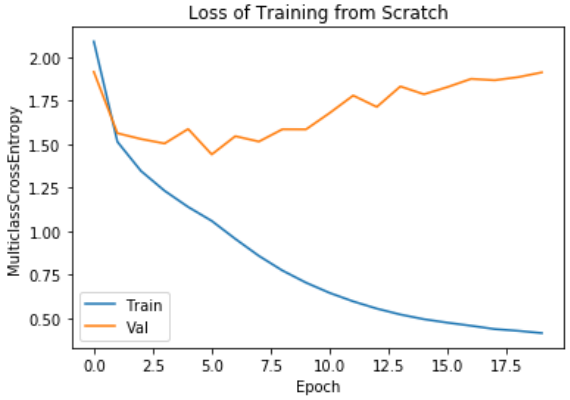
(a) Accuracy of Finetuned Network



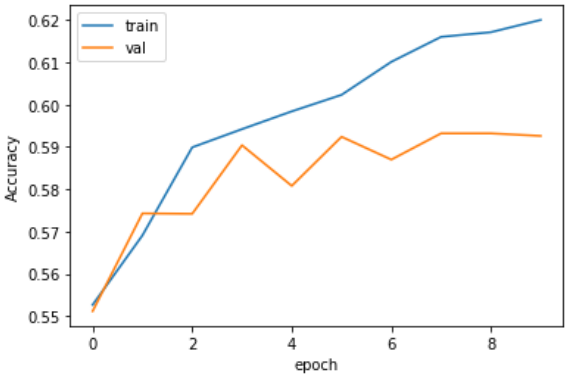
(b) Loss of Finetuned Network



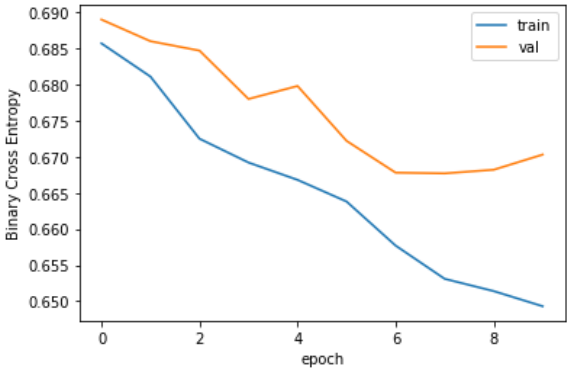
(c) Accuracy of Network Trained from Scratch



(d) Loss of Network Trained from Scracth



(e) Accuracy of Network in Problem 3



(f) Loss of Network in Problem 3

Fig. 1 Accuracy and Loss Curve for Each Network

File list:

Submission/:

post_pred_acc.py: files to decode the label and try to calculate the accuracy for the test result.

Problem2/: Files for problem 2

post_data.py: Post prediction for test data, a data loader for test set is designed.

Post_pred.py: Post prediction for test data, read a trained model and do prediction to test set.

post_utils.py: Config for prediction

Finetuned model/: Files for finetuned network

data.py: a data loader for training set.

train_category.py: model and training process for finetuned net.

utils.py: Config for mobile net.

From scratch/: Files for training from scratch

data.py: a data loader for training set.

train_category.py: model and training process for my own net.

utils.py: Config for the net.

Problem3/: Files related to Problem 3

data_pr3.py: data loader for pairwise training set.

train_cate_pr3.py: training process and model of problem 3

utils_pr3.py: config for problem 3

post_give_score.py: give prediction score for pairwise test set

data_pr3_for_line.py: a data loader designed for each single line of compatibility_test_hw.txt

test_json_process.py: decode compatibility_test_hw.txt and use the relative picture to predict the compatibility for the whole test set.