MLT2020SPRING - Final Project

Team Member and Contribution

資工四 B05902017 王盛立 (1/3，程式實作以及寫報告)
資工四 B05902023 李澤諺 (1/3，程式實作以及寫報告)
資工三 B06902041 吳采耘 (1/3，程式實作以及寫報告)

Data Preprocessing

1. Resizing
   由於dataset之中的圖片大小不盡相同，因此我們首先將dataset之中所有的圖片皆調整為相同的大小。在圖片大小的選擇上，我們計算了dataset之中所有圖片大小的平均，其大致上為1050 X 1240，因此最直觀的想法為將dataset之中所有的圖片皆調整為1050 X 1240。假設dataset之中圖片大小變異不大，那麼將dataset之中所有的圖片皆調整為平均大小1050 X 1240，便能保留大部分圖片的原始資訊，進而提高preformance。但因為1050 X 1240尺寸太大，使得程式執行效率極差甚至因為記憶體不足使得程式無法執行，因此我們退而求其次，選擇儘可能維持圖片但較小的長寬比例。1050 X 1240的長寬比例與現今相機中的3 : 4較為相近，並且考慮到許多pretrained CNN所接受的圖片大小大多為224 X 224，因此我們選擇將dataset之中的圖片皆調整為300 X 400。其長寬比例為3 : 4且大小與224 X 224相近，且該尺寸使程式執行效率較高，故可能為不錯的選擇。
2. Augmentation
   在machine learning之中，data的數量為很重要的問題。然而限制於不能使用額外的dataset，我們選擇將現有的data進行augmentation。在training時，我們會將training dataset之中的圖片以0.5的機率進行隨機順時針或逆時針旋轉0到15度之間的角度、隨機水平平移圖片寬度的0.1倍以內的長度、隨機垂直平移圖片高度的0.1倍以內的長度、隨機伸縮0.9到1.1之間的倍率，以及隨機水平翻轉，以此增加training data的數量，以提高model的performance。
3. Normalization
   我們將dataset之中圖片的pixel數值皆除以255，即normalize到0和1之間，再進行training或testing。

Feature Engineering

在進行了前述的data preprocessing之後，考慮到本次final project的題目為image classification，而CNN的運算較能考慮到圖片之中潛在的pattern，因此我們選擇將圖片丟入CNN進行training和testing。我們使用了自己所疊的CNN以及ResNet-18 (structure和parameter等等會於下一段之中講述)。除了CNN以外，我們也有嘗試使用其它model，如SVM、random forest、AdaBoost、gradient boosting (parameter等等會於下一段之中講述)。如前所述，在CNN做完convolution運算之後，其後的hidden layer中的各個dimension很有可能代表了原始圖片之中某種潛在的pattern，因此經常被作

為feature transformation。故我們選擇將CNN中倒數第二層hidden layer的output當作後續model所需的feature，將其丟入SVM、random forest、AdaBoost、gradient boosting作為input，以此為基礎進行training和testing。

Model Description

以下說明我們所使用的各個model的structure和parameter，其中，我們使用可以使model在validation dataset上的performance最高的parameter，作為各個model最終的parameter選擇。

- CNN
  我們使用PyTorch來實作CNN (API詳見Reference (https://pytorch.org/docs/stable/nn.html))，以下為我們自己疊的CNN的structure，其中，我們使用Adam進行training，batch size為32，learning rate為0.0005，總共100個epoch，以此進行training。

```
conv=Sequential(
  Conv2d(3,8,3,1,1),BatchNorm2d(8),LeakyReLU(0.2),MaxPool2d(2,2),Dropout(0.1),
  Conv2d(8,16,3,1,1),BatchNorm2d(16),LeakyReLU(0.2),MaxPool2d(2,2),Dropout(0.1),
  Conv2d(16,32,3,1,1),BatchNorm2d(32),LeakyReLU(0.2),MaxPool2d(2,2),Dropout(0.2),
  Conv2d(32,64,3,1,1),BatchNorm2d(64),LeakyReLU(0.2),MaxPool2d(2,2),Dropout(0.3),
  Conv2d(64,128,3,1,1),BatchNorm2d(128),LeakyReLU(0.2),MaxPool2d(2,2),Dropout(0.3),
  Conv2d(128,256,3,1,1),BatchNorm2d(256),LeakyReLU(0.2),MaxPool2d(2,2),Dropout(0.5))
fc=Sequential(
  Linear(6144,512),LeakyReLU(0.2),Dropout(0.5),
  Linear(512,32),LeakyReLU(0.2),Dropout(0.5),
  Linear(32,3))
```

- ResNet-18
  ResNet-18的structure詳見Reference (https://arxiv.org/abs/1512.03385)，在此，我們使用PyTorch中已經寫好的ResNet-18，並下載其已經train好的weight進行fine-tune。
  由於ResNet-18原本是用於有1000個class的classification task上，其output layer之中有1000個neuron，而我們這次的final project之中僅有3個class，因此我們在原有的ResNet-18之後接上了一個input dimension為1000、output dimension為3的linear layer。整體實作如下所示，其中，我們使用Adam進行training，batch size為32，learning rate為0.0005，fine-tune了10個epoch，以此進行training。

```
cnn=resnet18(pretrained=True)
fc=Linear(1000,3)
```

- SVM
  如前段所述，我們將CNN中倒數第二層hidden layer的output作為feature，以此丟給SVM進行training和testing。其中，我們使用了sklearn之中的SVC (API詳見Reference (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html))，並且依照老師於上課時所給的建議，先使用最簡單的linear SVM，再使用polynomial-kernel SVM。我們會逐漸增加polynomial kernel的degree以提高transformation的複雜度，並確保不會因為degree上升產生overfit的現象。最後才使用最複雜的RBF-kernel SVM。
  在選擇參數的過程中，我們發現linear SVM和degree大於2的polynomial-kernel SVM不論如何調

整parameter，在validation dataset上的performance始終很差。我們認為可能原因如下：由於linear SVM在training dataset上的performance不好，因此很有可能為model太弱，導致underfitting；而degree大於2的polynomial-kernel SVM在training dataset上的performance極好，因此很有可能為model過於強大複雜，導致了overfitting。
因此，以下僅列出degree為2的polynomial-kernel SVM和RBF-kernel SVM的parameter：

```
SVC(C=1,kernel='poly',degree=2,coef0=80)
SVC(C=250,kernel='rbf')
```

- Random Forest
  如前段所述，我們將CNN中倒數第二層hidden layer的output作為feature，並將其丟給random forest進行training和testing。我們使用了sklearn之中的RandomForestClassifier (API詳見Reference (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html))，我們使用的parameter如下：

```
RandomForestClassifier(n_estimators=120,criterion='gini',max_depth=5,
min_samples_split=2,min_samples_leaf=1,max_features='auto',max_leaf_nodes=10,
bootstrap=True,max_samples=0.8,n_jobs=-1)
```

- AdaBoost
  如前段所述，我們將CNN中倒數第二層hidden layer的output作為feature，以此丟給AdaBoost進行training和testing。我們使用了sklearn之中的AdaBoostClassifier (API詳見Reference (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html))，其中的parameter如下：

```
AdaBoostClassifier(n_estimators=50,learning_rate=0.1)
```

- Gradient Boosting
  如前段所述，我們將CNN中倒數第二層hidden layer的output作為feature，以此丟給gradient boosting進行training和testing。我們使用了sklearn之中的GradientBoostingClassifier (API詳見Reference (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html))，其中的parameter如下：

```
GradientBoostingClassifier(n_estimators=50,learning_rate=0.07)
```

Experiment

下表為上述的各個model分別在training dataset和validation dataset上的performance，以及其training所花費的時間：

Model	Training Accuracy	Validation Accuracy	Time
CNN	0.801610	0.805000	1h
ResNet-18	0.831250	0.826250	10m
Polynomial-kernel SVM	0.807679	0.790000	50s
RBF-kernel SVM	0.832857	0.795000	50s
Random Forest	0.791071	0.766250	1s
AdaBoost	0.785000	0.773750	35s
Gradient Boosting	0.832857	0.772500	4m

由此可以看出，ResNet-18在validation dataset上的performance最好。上傳後，ResNet-18在test dataset上得到的testing accuracy為0.808125。另外，由於random initialization的緣故，model在每次training時得到的weight皆不同。因此我們train了3個validation accuracy相近的ResNet-18，將其進行uniform blending，以減少variance來得到較為穩定的結果。經過uniform blending之後，我們在test dataset上的accuracy提高到了0.809375。

Disussion

由於image classification在feature的擷取上比較複雜，需要考慮連續的區域。因此相較於PCA或是autoencoder等無法檢測包含RGB資訊圖片的extraction，以CNN擷取每張圖片的feature會是最適當的選擇。

- Performance：
  根據實驗結果的比較表，可以發現training dataset的accuracy中，表現最好的前三名分別為ResNet-18、RBF-kernel SVM以及Gradient Boosting。但以validation dataset驗證時，ResNet-18以及CNN則分佔一二名。
  因為CNN以及ResNet-18擁有許多層neuron，在training時可以利用dropout等方式強化每個neuron的參數，因此在所有model中表現最好也最不容易產生overfitting。其中，ResNet-18因為解決了CNN的架構中梯度下降的問題，因此會比原本的CNN更加出色。另一方面，針對RBF-kernel SVM以及Gradient Boosting些微overfitting，我們認為可能的原因有兩條：由於RBF-kernel SVM希望可以以一條線區分所有資料，因此在把所有資料轉換到無限多維的過程中，非常容易產生overfitting的現象；而從AdaBoost改良的Gradient Boosting可能因為在training過程中即使error已經夠小了，仍會不斷補強base models中的error，因此產生些微overfitting的現象。
  最後，針對Polynomial-kernel SVM、Random Forest以及AdaBoost，我們認為其表現不如前列model的原因分別可能為：Polynomial-kernel SVM希望在不同維度上能夠以直線區分不同類別，因此若data本身無法在不同維度上分類，其accuracy便有其上限；而Random Forest若在建立決策樹的過程中產生許多相似的決策樹，便會大幅影響accuracy的表現；此外，AdaBoost在training時因為關注目前分類的error，因此若在data中混有一些含有noise的圖片便會對AdaBoost產生嚴重的影響，因此是AdaBoost的表現欠佳。
- Efficiency：
  在以上所述的model之中，由於random forest之中的各個decision tree其training為互相獨立的，因此可以使用平行計算來加速random forest的training過程，故其efficiency最高，其次為使用

quadratic programming作計算的SVM，而由於AdaBoost和gradient boosting皆須train出多個base model再進行ensemble，且這些base model為sequential而得，因此無法像random forest一樣使用平行計算來加速training過程，隨著base model的數量越多，AdaBoost和gradient boosting於training時所需的時間就越久，因此efficiency比僅由單一個model所組成的SVM再差一些，至於AdaBoost和gradient boosting之間的efficiency，由於gradient boosting可以視為從AdaBoost延伸出來較為general且flexible的model，其在計算上也較為複雜，因此gradient boosting的efficiency較AdaBoost差了一些，最後，由於CNN的架構為所有model之中最大的，且不論是在計算model的output或是進行back propagation都需要極大的計算量，雖然NN的計算可以表示為矩陣運算，進而用GPU進行加速，但如此大的計算量，且要更新多個epoch，因此efficiency為所有model之中最差的。

- Scalability
  我們認為在給了更多的data時，由於以上的各個方法的feasibility與data的數量無直接的關聯，只要在機器設備仍能負荷的情況下，以上的各個方法仍為feasible，甚至如CNN等model在給了更多的data時，其可能可以提供更多資訊，因而可能提升model的performance，不過在給了更多的data時，CNN需要重新進行training，以找出可能更為適合的weight，如此一來，各個data的feature vector也會不同，除此以外，SVM、random forest、AdaBoost、gradient boosting等model也需要重新進行training以找出更適合的weight，而在重新training，由於data數量變多，因此efficiency可能會下降，但如前所述，只要機器設備仍能負荷，以上各個方法的feasibility應不會受到太大的影響。
- Popularity：
  近年來在image classification的task之中，CNN的popularity可能為最高的，因為CNN中使用了filter對圖片進行convolution運算，不同的filter可以以此找出圖片中潛在的pattern，並且，由前面幾層的convolutional layer所找出的pattern，可以由更後面的convolutional layer組成更為複雜的pattern，以此作為feature，可能更可以代表出原始圖片中潛藏的有意義的資訊，比起過往需要由人工使用domain knowledge來決定圖片中有哪些性質可以作為feature，convolution更能找出人工無法找到的潛在且有意義的feature，也因此CNN於現今的image classification的task上，或是於feature egineer之中，popularity都很高。而其它的model中，使用boosting的model，如AdaBoost、gradient boosting、XGBoost等model，或是tree-based的model，如random forest、LightGBM等model，於表格式的data上performance往往比NN還要好，也因此於這類的task上，使用boosting或tree-based的model其popularity也很高。
- Interpretability
  如前所述，CNN會使用filter與圖片進行convolution運算，以找出圖片中潛在的pattern，只要能visualize各個filter所偵測的pattern，或是visualize原始圖片中會被CNN偵測到的pattern，就能說明CNN是如何將圖片進行分類或是抽取feature。Visualize各個filter所偵測的pattern方式如下(稱為feature visualization，詳見Reference (https://raghakot.github.io/keras-vis/visualizations/activation_maximization/))：由於若圖片中含有我們想visualize的filter其所偵測的pattern，則該filter的output便會越高，因此我們可以先隨機生成一張圖片，將其丟入CNN之中，計算我們所要visualize的filter其output對圖片中各個pixel的微分值，進行gradient ascend，更新圖片中pixel的值，以此maximize我們要觀察的filter的output，如此一來，更新多個epoch之後，圖片就會converge到該filter在偵測的pattern的樣子，達到visualization。而若要觀察CNN是如何分類一張給定的圖片，其方式為(稱為saliency map，詳見Reference (https://raghakot.github.io/keras-vis/visualizations/saliency/))：將該圖片丟入CNN之中，計算該圖片所屬的class的output neuron輸出的值，將其對原始圖片中的各個pixel作微分，若微分的值越大，表示該pixel對於圖片是否屬於這個class的判斷越為重要，以此便能看出CNN是根據哪些pattern將該圖片進行分類的。使用以上方

式，便能解釋用CNN中倒數第二層的output作為feature時，feature vector中各個dimension所代表的意義，可能也就能解釋SVM所找出的hyperplane是以何種標準將data區分、解釋random forest中的各個decision tree或AdaBoost和gradient boosting之中的各個base model是以甚麼feature將data作出簡單的分類，進而ensemble成更複雜的model。

綜合以上所述，我們會推薦使用CNN，我們在前面已不斷提到了CNN的優點，其可以找出圖片中潛在的pattern作為feature，為CNN於現今image classification中最大的優勢，且我們可以透過visualize各個filter或圖片中被偵測到的pattern，來解釋CNN是如何將圖片進行分類，可能可以幫助我們加強CNN的performance或改善CNN的架構等等，而像ResNet-18這種由企業研究出的pretrained model，由於企業所擁有的資源往往較多，其設計的pretrained model乃至其train出來的weight，可能可以為我們帶來更好的performance，以上皆為CNN的優點，而CNN的缺點，像是需要大量data，以及需要較多的時間進行training，而架構更大的pretrained model，甚至會需要更好的機器設備才能train得起來，為CNN的缺點。

Reference

- PyTorch API：https://pytorch.org/docs/stable/nn.html (https://pytorch.org/docs/stable/nn.html)
- ResNet-18：https://arxiv.org/abs/1512.03385 (https://arxiv.org/abs/1512.03385)
- sklearn SVC：https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html)
- sklearn RandomForestClassifier：https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)
- sklearn AdaBoostClassifier：https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html)
- sklearn GradientBoostingClassifier：https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html)
- feature visualization：https://raghakot.github.io/keras-vis/visualizations/activation_maximization/ (https://raghakot.github.io/keras-vis/visualizations/activation_maximization/)
- saliency map：https://raghakot.github.io/keras-vis/visualizations/saliency/ (https://raghakot.github.io/keras-vis/visualizations/saliency/)