Relazione progetto Machine Learning

Garzarella Simone 499813 – Sigona Alessio 482697

Il nostro progetto si è concentrato sul riconoscimento del tempo atmosferico e sulla classificazione delle immagini in base ad esso.

Dataset

Ci siamo avvalsi dell'unione di tre diversi dataset trovati su **Kaggle**, i quali riportavano tutti immagini jpg con diverse varianti meteo, per un totale di circa 11K immagini. Abbiamo classificato queste ultime in cinque diverse classi:

- *Cloudy*
- *Rainy*
- *Sunny*
- *Sunrise*
- *Foggy*

Ecco alcuni esempi di immagini utilizzate:





Ambiente di Sviluppo

Per sviluppare Il nostro progetto abbiamo utilizzato **Google Colab**, facendo uso delle librerie Tensorflow/Keras ed il linguaggio Python.

Colab mette a disposizione dell'utente una macchina virtuale con 12 GB di RAM e circa 68 GB di spazio su disco. Queste risorse limitate, soprattutto per quanto riguarda la RAM, hanno influito molto sui nostri esperimenti e non ci hanno permesso di sfruttare al meglio l'intero dataset.

Modello

Essendo un lavoro basato su immagini, abbiamo utilizzato una **Convolutional Neural Network (CNN)**.

In particolare, il nostro modello è formato da 5 strati convoluzionali, rispettivamente formati da 16, 32, 64, 128 e 356 filtri 3x3, padding *same* e *relu* come funzione di attivazione.

Questi strati sono alternati da layer di **max pooling** che semplificano la comunicazione tra i layer convoluzionali operando un campionamento dei dati ed associando porzioni di immagini ad una rappresentazione più piccola, riducendo gli iperparametri.

Alla base della rete abbiamo:

- un layer di **dropout** per far fronte al problema dell'overfitting, in quanto spegne randomicamente neuroni dello strato finale in modo tale da non far specializzare solo una parte di essi sul riconoscimento di una determinata feature, ma costringere neuroni con diverse posizioni a focalizzarsi sul distinguerne una in particolare;
- un layer **flatten** che trasforma le matrici in un unico vettore unidimensionale;
- un layer **dense** fully connected con il vettore risultante;
- un layer di **output**.

```
num_classes = 5

model = Sequential([
                    data_augmentation,
  layers.experimental.preprocessing.Rescaling(1./255, input_shape=(200, 300, 3)),

  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(128, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(256, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.9),
  layers.Flatten(),
  layers.Dense(512, activation='relu'),
  layers.Dense(num_classes)
])
```

Abbiamo utilizzato **Adam** come algoritmo di ottimizzazione per la nostra rete neurale e per valutare tale modello ci siamo avvalsi della metrica dell'**accuracy.**

```
# Compilazione del modello

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Esperimenti e Tuning di parametri

Come prima cosa abbiamo ristretto il dataset a sole 1000 immagini per effettuare qualche test iniziale con sole 4 classi: *sunny, rainy, cloudy e sunrise.*

La prima CNN utilizzata aveva soltanto tre strati convoluzionali ed era priva del layer di dropout. I risultati erano soddisfacenti, ma l'overfitting era significativo.

In un successivo esperimento, abbiamo preso il modello precedente e abbiamo aggiunto lo strato di dropout prima del flatten. Inoltre, nelle precedenti prove, andavamo a normalizzare il valore RGB tra 0 e 1, dividendolo "manualmente" per 255. In questo ultimo esperimento invece abbiamo usato una funzione di rescaling nel

momento della compilazione del modello. Inoltre, per far fronte al problema dell'overfitting, abbiamo effettuato un'operazione di data augmentation in cui le immagini nel dataset sono state ruotate, capovolte e zoomate in modo random. I risultati sono migliorati e l'overfitting è sceso.

A questo punto abbiamo aggiunto la quinta classe, *foggy*, ma con le precedenti impostazioni le prestazioni sono leggermente calate, in quanto le immagini sono molto simili a quelle con meteo nuvoloso.

Inoltre, volevamo aumentare il dataset ed utilizzare quello completo da 11K immagini ma abbiamo riscontrato un problema: nel fare tutte le operazioni necessarie la RAM fornita su Colab veniva velocemente saturata e portava ad un arresto anomalo del sistema. Facendo varie prove siamo giunti alla conclusione che il limite massimo di immagini affinché Colab non si arrestasse era di 1.5K immagini 200x300. Per via delle suddette limitazioni, abbiamo anche effettuato diverse prove con la grandezza delle immagini: aumentandola si è costretti a diminuire il numero di immagini nel dataset, diminuendola possiamo aumentare il dataset ma le prestazioni scendono.

Abbiamo quindi dovuto fare una scrematura e scegliere 1.5K immagini dalle 11K iniziali, prima in modo casuale, riscontrando alcuni problemi in quanto molte erano rumore (prive di qualsiasi elemento che potesse ricondurre al tempo atmosferico), poi andando a selezionare manualmente quelle più pertinenti.

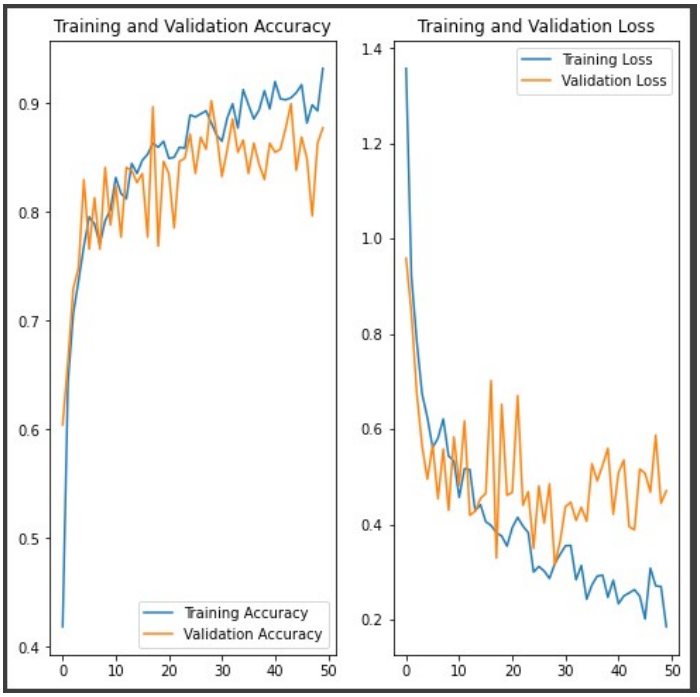
Su questo dataset e con questa nuova classe abbiamo effettuato diversi esperimenti. Ci siamo accorti che aumentare la profondità della rete aggiungendo strati convoluzionali non portava a nulla, mentre abbiamo notato un aumento nelle prestazioni incrementando la grandezza dello strato denso alla base della rete, che abbiamo fissato a 512 nodi. A questo punto l'accuracy è tra il 75 e l'80%.

Stabilita la struttura della rete abbiamo provato a cambiare l'algoritmo di ottimizzazione: abbiamo effettuato tentativi con **SGD** (Stochastic Gradient Descent), **BGD** (Batch Gradient Descent) e mini-batch gradient descent.

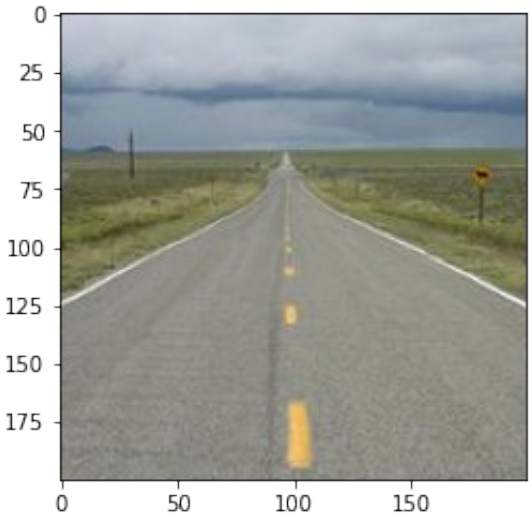
Dopo varie prove siamo giunti alla conclusione che **Adam optimizer** rimane il migliore nel nostro caso.

Per far fronte al problema dell'overfitting abbiamo anche giocato sullo strato di dropout: inizialmente andavamo a spegnere il 20% dei neuroni, riscontrando un overfitting non significativo ma comunque presente. Abbiamo poi notato che andando a spegnere il 90% dei neuroni le prestazioni non diminuivano, anzi, abbiamo avuto i risultati migliori con un overfitting quasi inesistente, infatti i valori di accuracy del training e del test set sono comparabili.

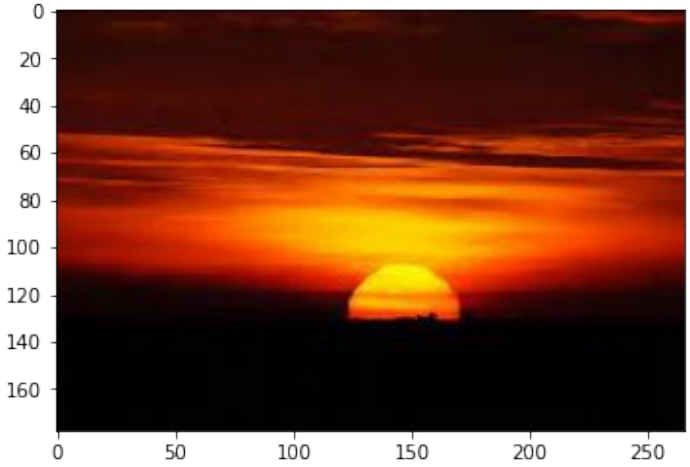
Con le impostazioni finali abbiamo infine provato ad aumentare il numero delle epoche. In effetti con 30 epoche i risultati miglioravano, ma è con 50 epoche che abbiamo raggiunto i risultati migliori, con valori che si stabiliscono sopra all'80% già dalle prime epoche e che sfiorano picchi dell'90%:



Inoltre, come test finale, abbiamo fatto classificare alla nostra rete delle immagini esterne scelte da noi, una per ogni classe meteorologica, riscontrando risultati positivi per 4 immagini su 5.




```
This image most likely belongs to Cloudy with a 81.07 percent confidence.
```




```
This image most likely belongs to Sunrise with a 99.99 percent confidence.
```

Purtroppo, abbiamo notato che la nostra CNN ha problemi nel riconoscere alcune immagini appartenenti alla classe *rainy*, che vengono confuse con la classe *cloudy.*


```
This image most likely belongs to Cloudy with a 94.09 percent confidence.
```

Questo secondo noi si verifica in quanto le caratteristiche individuate nella tipologia di immagini *rainy* sono simili a quelle riscontrare in *cloudy*, poiché la CNN da noi utilizzata si basa sul riconoscimento di feature salienti (nuvole, sole, ombrelli, etc…) piuttosto che sulla variazione di luminosità dei pixel, cosa essenziale per differenziare un'immagine con meteo pioggia da una con meteo nuvoloso.

Matrice di confusione per Rete Neurale

Si è proseguito ad evidenziare gli errori di classificazione della rete neurale plottando una matrice di confusione come si evidenzia nella foto seguente :

```
[56] tf.math.confusion_matrix(
         label_test, score_finale, num_classes=None, weights=None, dtype=tf.dtypes.int32,
         name=None
     )
```

```
<tf.Tensor: shape=(5, 5), dtype=int32, numpy=
array([[77,  3,  3,  4,  0],
       [ 9, 44,  6,  2,  2],
       [12,  2, 33,  3,  0],
       [ 4,  0,  0, 88,  0],
       [ 0,  0,  0,  3, 64]], dtype=int32)>
```

In particolare, si è provato ad utilizzare la matrice di confusione implementata da Sikit Learn, ma essa aveva la necessità di avere

come paramentro, un classificatore che non era compatibile con il modello neurale di TF. Si è quindi deciso di plottare una matrice che rappresenti tale “confusion matrix” utilizzando la stessa libreria TF, ove gli indici di tale matrice(riga-colonna) rappresentano i risultati di previsioni avute per le nostre classi di classificazione. (0:`'Cloudy'`, 1:`'Foggy'`, 2:`'Rainy'`, 3:`'Sunny'`, 4:`'Sunrise'`).

Classificazione tramite SVM(Support Vector Machine)

È stato utilizzato anche un modello di classificazione tramite SVM. In questo progetto, viene usata la SVM multiclasse implementata dalla libreria SikitLearn di nome LinearSVC (Linear Support Vector Classification).

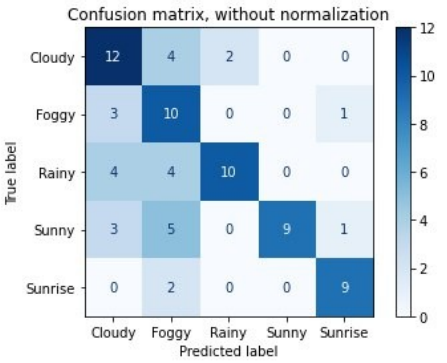
Purtroppo a causa dell’ implementazione del codice in Colab e dei suoi limiti, è stato necessario ridurre il dataset a 359 immagini da 1200 a causa di errori dovuti al riempimento della RAM utilizabile oppure alla scadenza della inattività del runtime. Ciononstante si è proseguito al task di classificazione ottenendo dei risultati che si aggirano intorno al 63% di accuracy anche con un tuning del paramentro “C” di regolarizzazione della SVM.

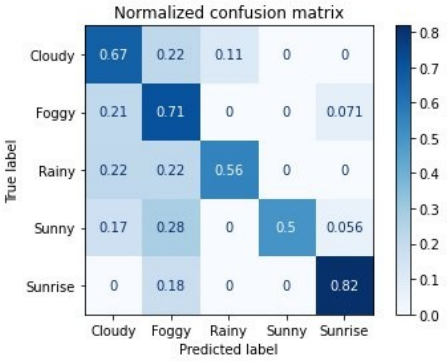
```
[9]
    lin_clf = svm.LinearSVC(C = 0.01, max_iter= 100000)
    lin_clf.fit(x_train, y_train)
```

```
LinearSVC(C=0.01, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=100000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

```
print(y_test)
print(lin_clf.predict(x_test))
lin_clf.score(x_test,y_test)
```

```
[0 3 0 0 2 1 1 0 2 3 4 3 2 2 3 2 3 1 1 2 2 2 0 2 1 4 3 2 1 4 4 4 0 3 1 1 4
 4 3 0 0 2 2 3 0 3 2 0 3 0 0 3 0 2 0 3 4 1 3 4 2 3 0 4 4 2 3 0 3 1 3 1 1 2
 1 0 1 2 0]
[1 1 0 0 2 1 1 0 0 1 4 1 2 2 3 0 1 0 1 2 0 1 0 2 4 4 1 2 1 4 1 1 2 3 1 0 4
 4 3 0 0 1 2 0 1 3 0 1 0 0 0 3 0 2 0 3 4 1 3 4 1 4 1 4 4 1 3 0 3 1 0 0 1 2
 1 2 1 2 0]
0.6329113924050633
```





È stata inoltre plottata anche qui una matrice di confusione della classificazione,normalizzata e non, qui però tramte SikitLearn.

Conclusioni e Sviluppi Futuri

In conclusione, abbiamo riscontrato buoni risultati sulla classificazione delle immagini pur avendo risorse limitate da Google Colab e dovendo, per tale motivo, utilizzare un dataset ristretto, fornendo alla nostra rete neurale immagini di dimensione ridotta rispetto alle originali.

Se tali limitazioni non fossero state applicate, siamo sicuri che avremmo avuto un aumento di performance nella nostra classificazione dovuto all'utilizzo di un dataset più ampio per il training e ad una risoluzione delle immagini migliori. Infatti, pensiamo che la risoluzione delle immagini sia essenziale per uno sviluppo futuro di questa rete che evidenzi la differenza tra la luminosità dei pixel di un'immagine e che ci aiuti a riconoscere feature più complicate da individuare in una normale rete convoluzionale, la quale evidenzia solo feature salienti.

Probabilmente le minor performance di riconoscimento, da noi ottenute, per quanto riguarda le classi simili come *rainy* e *cloudy* è data proprio dalla mancata individuazione di differenze di luminosità dei pixel appartenenti alle immagini delle diverse classi.

Inoltre, si potrebbe implementare un'ulteriore rete che vada ad individuare la porzione di immagine saliente nel riconoscimento meteorologico, in modo da eliminare la maggior parte del rumore,

facendo quindi una “pre-elaborazione intelligente” delle immagini da fornire poi in input alla rete di classificazione.

Link del progetto

Il codice è disponibile al seguente link:

https://github.com/Alexo961/ML_SII_project

Il dataset per la rete neurale utilizzato lo trova al seguente link:

https://drive.google.com/file/d/13uOo8EpnTfkEc_V9Q9aJ-dr8HfzM1jkO/view?usp=sharing

Il dataset per la SVM utilizzato lo trova al seguente link:

https://drive.google.com/file/d/1XIskryAI-Ih8bqC_39LqGLcH20A3L37B/view?usp=sharing

Le immagini usate nel codice per i test finali della rete neurale le trova qui:

https://drive.google.com/drive/folders/1Kmt7hy2NKR4DMqoB2O07IBqgaLuUs_H9?usp=sharing