Università degli studi di
Milano-Bicocca

Advanced Machine Learning

Weather Recognition

Authors:
Francone Vito - 872904 - v.francone@campus.unimib.it
Sotgiu Allegra - 876621 - a.sotgiu2@campus.unimib.it
Ranieri Silvia - 878067 - s.ranieri7@campus.unimib.it

A.Y.: 2021/2022

Contents

February 21, 2022

Abstract

Lo scopo di questo progetto è l'identificazione di un modello per il task di weather recognition. L'applicazione del modello è pensata per essere eseguita su dispositivi di sensoristica IoT, dotati di una bassa potenza computazionale, per aumentare la sicurezza alla guida allarmando il guidatore in caso di condizioni meteo sfavorevoli. L'implementazione del modello è stata dunque focalizzata sull'utilizzo di modelli leggeri come la Mobile Net e di tecniche di pruning calibrate per ridurre il peso in memoria del modello e aumentarne la velocità di esecuzione limitando al contempo il calo di performance sull'accuracy della classificazione.

1 Introduzione

Il tema della sicurezza in strada e della sensoristica integrata ai mezzi di trasporto è un hot topic che, con l'avvento dei sistemi di guida autonoma, tenderà a crescere maggiormente. Molte auto di recente immatricolazione sono già dotate di camere sia frontali che posteriori che permettono al guidatore una visione a 360 gradi durante il posteggio dell'auto. Durante la guida, tuttavia, queste telecamere rimangono pressochè inutilizzate o inaccessibili al guidatore. Di qui nasce lo spunto per questo progetto: utilizzare la sensoristica già presente sulle vetture e sfruttarne il potenziale per un'esperienza di guida più sicura. Installando un modello di ridotte dimensioni sul computer di bordo è possibile analizzare le immagini delle camere installate sulla vettura per ricavarne informazioni sulle condizioni climatiche in tempo reale. Queste informazioni possono essere utilizzate dalla vettura per modificarne autonomamente l'assetto o per inviare messaggi in tempo reale all'autista. E' di fondamentale importanza dunque che il modello abbia uno spazio in memoria sufficientemente ridotto e tempi di elaborazione rapidi.

2 Dataset

Il dataset utilizzato è Weather Image Recognition [1] presente su Kaggle e contiene immagini di diverse dimensioni riguardanti le condizioni metereologiche suddivise in cartelle:

- **dew - rugiada**: acqua che compare sul suolo o sulla vegetazione quando il vapore acqueo nell'aria viene a contatto con superfici raffreddate dopo essere state esposte al sole
- **fogsmog - nebbia**: nube densa a bassa quota che limita la visuale
- **frost - brina**: deposito sul suolo o altre superfici di ghiaccio granuloso dall'aspetto cristallino che si forma per brinamento del vapore acqueo presente nell'aria
- **glaze - glassatura**: strato di ghiaccio trasparente e scivoloso che si forma quando la temperatura è sotto zero e l'acqua si solidifica istantaneamente quando tocca il suolo
- **hail - grandine**: precipitazione di acqua in forma ghiacciata, dovuta alla solidificazione nelle nubi

- **lightning - fulmini**: scarica elettrica che si crea a causa dell'elettricità atmosferica
- **rain - pioggia**: precipitazione d'acqua in forma liquida
- **rainbow - arcobaleno**: fenomeno ottico che produce uno spettro di luce quando i raggi del sole attraversano gocce d'acqua rimaste in sospensione
- **rime - galaverna** : precipitazione atmosferica che consiste in un deposito di ghiaccio in forma di aghi sulle superfici, può prodursi in presenza di nebbia se la temperatura dell'aria è molto sotto zero
- **sandstorm - tempesta di sabbia**: sabbia sollevata da forti raffiche di vento
- **snow - neve**: precipitazione di acqua ghiacciata formata da minuscoli cristalli

Per un totale di 6862 immagini.
Data la natura del task si è deciso di escludere da questo progetto la classe 'arcobaleno' in quanto la presenza di questo fenomeno non restituisce informazioni significanti sul luogo in cui viene scattata l'immagine. Tutte le altre classi sono invece ritenute significative perchè indicano una condizione diversa della strada, della visuale o dell'aria in cui si sposta un certo mezzo di trasporto.

2.1 Data exploration

Si osserva, in Figura 1, come prima cosa un campione delle immagini presenti nel dataset:

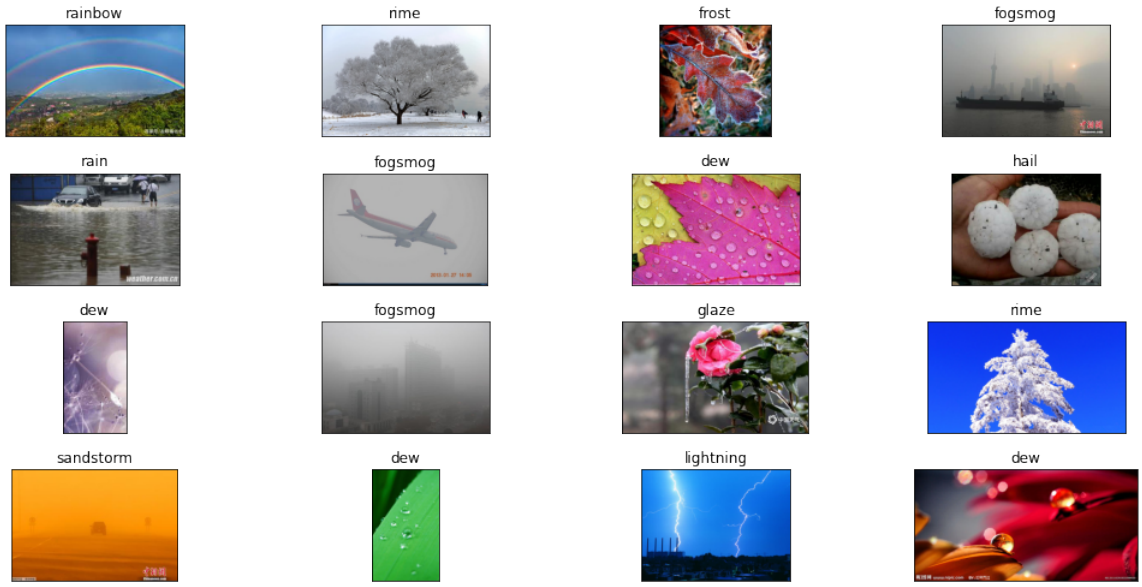

Figure 1: Esempi di immagini contenute nel dataset

Da questo grafico si può anche osservare che le immagini hanno dimensione variabile.

Osservando i barplot delle classi, Figura 2 , si osserva che il dataset presenta un problema di class imbalance poichè il numero di immagini contenute in ogni cartella è variabile.

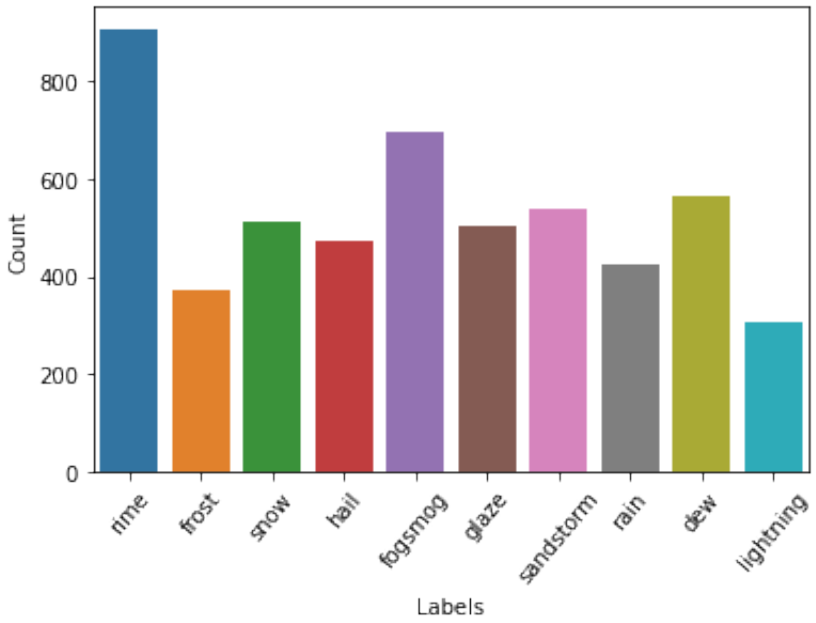

Figure 2: Barplot delle classi

2.2 Preprocessing

Durante il caricamento dei dati si è proceduto ad effettuare una prima fase di preprocessing. Come prima cosa si è fissato un valore massimo di 600 immagini da importare da ogni cartella, questo per risolvere il problema delle classi sbilanciate. Ogni immagine è stata ridimensionata per avere dimensioni (224,224,3) dove 3 è il numero dei canali colore, scelgiendo il metodo di interpolazione dei valori per ottenere la dimensione desiderata nel caso di immagini di dimensioni minori. Successivamente tutte le immagini sono state trasformate in array e le etichette associate codificate con la codifica one-hot encoding sulle 10 classi.

Infine, il dataset è stato suddiviso in training set con l'80% dei dati e test-set con il 20%.

3 Approccio metodologico

La creazione del modello ha seguito diverse fasi, si è dapprima utilizzato il transfer learning con diverse architetture di cui si seguito ne saranno riportate due, successivamente sono stati implementati vari metodi di regolarizzazione e infine si è snellito il modello con pruning e quantizzazione. In tutte queste fasi ci si è focalizzati sull'addestramento della rete ponendo l'attenzione su robustezza, velocità e leggerezza (spazio su disco ridotto).

3.1 Data Augmentation

Per effettuare un upsampling del dataset iniziale, precedentemente sottodimensionato, si è fatto uso della data augmentation. La data augmentation è un processo di regolarizzazione che permette di aumentare il numero di immagini del dataset effettuando modifiche alle immagini iniziali in modo tale da non modificarne le label e non distruggerne le informazioni contenute utili ai fini di quel particolare task. Per la Data Augmentation per il Weather Recognition si è fatto uso dei layer implementati da Keras:

- **RandomFlip**: capovolgimento orizzontale dell'immagine
- **RandomRotation**: rotazione casuale dell'immagine fino al 30%, riempimento con i valori più vicini

- **RandomZoom**: ingrandimento casuale dell'immagine fino al 50%
- **RandomContrast**: aumento è diminuzione casuale del contrasto dell'immagine fino al 50%
- **RandomTranslation**: traslazione orizzontale e verticale del 20% in ogni direzione

3.2 Modelli utilizzati

Dato lo scopo del progetto, la scelta del modello per il transfer learning è stata rivolta su modelli con size e inference size per step tra i più bassi tra i modelli forniti da Keras. In particolare è stata usata una MobileNetV2 pretrainata sul dataset *imagenet*. MobileNetV2 ha un task simile (ma non uguale) a quello di questo progetto , è stato ritenuto perciò adeguato effettuare un taglio abbastanza vicino al layer di output e in particolare al layer *block-16-project-BN*. Il secondo modello testato è stato ottenuto con un diverso taglio della MobileNetV2, nello specifico sul layer *block-15-project-BN*. Infine il terzo modello preso in considerazione è stato una DenseNet addestrata sullo stesso dataset di imagenet. La rete DenseNet è un modello più pesante della MobileNetV2 tuttavia ha standard sia di dimensioni che di tempo di inferenza per ogni step che sono paragonabili alla MobileNet e sono pensate per un applicazione mobile. La scelta tra i due modelli ha riguardato sia le performance, che nel caso della MobileNetV2 erano superiori,che il tempo di addestramento del modello, perciò si è deciso di non procedere oltre con il terzo modello.

In Figura 3 è riportata la struttura degli strati aggiunti alla rete pre-addestrata, utilizzata per tutti e tre i modelli della weather recognition.

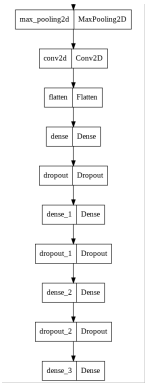

Figure 3: Strati aggiunti alla rete pre-addestrata

La rete presenta un layer di *Max Pooling* e un layer *Convoluzionale*, che conservano la dimensione spaziale. Successivamente un layer *Flatten* appiattisce il dato perdendo l'informazione spaziale per i 3 layer densi da 1024,512,128 neuroni. Il layer denso di output presenta 10 neuroni, pari al numero di classi.
Gli iperparametri scelti per l'addestramento della rete sono:

- Loss function: categorical crossentropy
- omptimizer: adam
- numero di epoche: 50
- batch size: 64

La categorical crossentropy e adam come optimizer sono scelte comuni per le reti neurali per avere una convergenza rapida in fase di training del modello, senza trascurare le performance. Un batch size di 64 (multiplo di 2, per favorire la parallelizzazione sull'hardware

GPU nell'ambiente di Google Colab) è un tradeoff tra generalizzazione delle performance del modello (robustezza della misura dell'accuracy) e velocità di training. Infine il numero di epoche è stato scelto in modo da minimizzare l'overfitting (evitando un numero troppo elevato di epoche) e massimmizzando l'accuracy (utilizzando un numero di epoche adeguatamente alto).

3.3 Regolarizzazione

La regolarizzazione L_2 è una delle tecniche più utilizzate per penalizzare modelli complessi nell'apprendimento automatico, viene utilizzata per ridurre l'overfitting(o errori di generalizzazione) riducendo i pesi della rete [2]. Il parametro λ viene detto tasso di regolarizzazione e serve per limitare l'effetto della regolarizzazione sulla frunzione di loss: quando λ è molto piccolo, il modello minimizza la funzione di costo, mentre quando è molto grande, il modello cerca di minimizzare i pesi il più possibile, in particolare la regolarizzazione L_2, minimizza la somma dei quadrati dei pesi. Nel modello è stata utilizzata una regolarizzazione con un fattore λ pari a 0.01 per avere un compromesso tra robustezza e velocità di apprendimento (intesa come numero di epoche necessarie a saturare le performance del modello). La formula è la seguente:

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} xij\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{1}$$

Il Dropout è una delle tecniche di regolarizzazione più efficaci e più utilizzate per le reti neurali, infatti permette di ridurre la capacità del modello in modo che il modello possa ottenere un errore di generalizzazione inferiore, nello specifico elimina temporaneamente alcuni neuroni dalla rete neurale durante l'addestramento. Quindi, dopo ogni iterazione, vengono spenti con una certa probabilità un insieme di neuroni, per impedire ad alcuni neuroni di dominare il processo, riducendo così il rischio di overfitting. Nel modello la probabilità di dropout scelta è pari a 0.4, anche in questo caso è un tradeoff tra velocità di addestramento e generalizzazione.

3.4 Pruning

Per rendere un modello più leggero e più veloce nell'inferenza, un metodo molto utilizzato è il pruning. L'idea è quella di rimuovere i pesi della rete che risultano essere meno rilevanti, ovvero quelli di ampiezza vicina allo zero e quelli che rimossi hanno un effetto minore sulle performance.
Per lo scopo di questo progetto si è deciso di sfruttare il pruning strutturato, ovvero un pruning che preserva la densità della rete e che permette di essere più efficiente. Per farlo, si è scelto di utilizzare la libreria *Pruning (Automatic-Structured.Pruned)* [3] poichè permette, tramite il comando *prune_model*, di effettuare il pruning strutturato di una rete specificando la percentuale di neuroni da rimuovere negli strati densi e la percentuale di filtri da rimuovere negli strati convoluzionali e la metrica da usare per selezionare questi nodi.
Si è deciso di applicare 4 fasi di pruning, ognuna rimuovendo il 10% di neuroni dagli strati densi e il 5% da quelli convoluzionali, e riaddestrare il modello dopo ognuna di queste per permettere ai pesi rimanenti di adattarsi a questo cambio di architettura. Inoltre la metrica utilizzata per ciascuna di queste iterazioni è la L_2, ovvero vengono selezionati, e quindi rimossi, i neuroni il cui quadrato della norma dei pesi è minore.

3.5 Quantizzazione

Un ulteriore tecnica per ridurre le dimensioni del modello e al tempo stesso velocizzarne la fase di inferenza è la quantizzazione. La quantizzazione utilizzata in questo progetto è la

Full Integer Quantization e consiste nell'approssimazione dei pesi della rete trasformando il formato dei pesi da float32 a int8, cioè da una rappresentazione a 32 bit ad una a 8 bit [4]. La compressione del modello in memoria risulta quindi di circa 4 volte e si ha inoltre un incremento di velocità nella fase di inferenza. A fronte di una minore latenza e minor spazio occupato in memoria, la latenza diminuisce, seppur lievemente, le performance del modello.

Applicando la quantizzazione alla rete dopo la fase di pruning il risultato è:

Float model in Mb: 16.83
Quantized model in Mb: 4.61
Compression ratio: 3.65

3.6 TF Lite

Per l'applicazione della quantizzazione si è utilizzato il framework di deep learning TensorFlow Lite [5]. TF Lite è progettata specificamente per il deployment di reti neurali su dispositivi mobile e IoT per ottimizzare velocità (bassa latenza), dimensioni e consumo energetico. I modelli .tflite sono infatti salvati in formato binario e sono pensati per hardware con sistema operativo Android, Linux e iOS. Il modello quantizzato è stato salvato in formato .tflite, il quale occupa uno spazio in memoria molto minore del modello originale. Per quanto concerne l'ottimizzazione della velocità non è stato possibile apprezzarne l'efficacia in quanto esula dagli obiettivi di questo il report il deployment di reti in formato binario su dispositivi mobile. Per poter infatti testare il modello sull'hardware (GPU) di Google Colab è necessario non solo convertire il modello ma anche le immagini del test set, questo risulta in un incremento dei tempi di inferenza non correlato con le effettive performance su dispositivi mobile e IoT. Si è testata infine l'accuracy del modello .tflite, i cui risultati sono presentati nella prossima sezione.

4 Risultati e Valutazioni

In questa sezione sono riportati i risultati dei due modelli analizzati in termini di accuracy e confusion matrix. Per il primo modello, quello successivamente sviluppato si riportano inoltre le performance in termini di precision e recall.

4.1 Primo modello

Il modello ha un'accuracy dell' 82.4%, in Figura 4 è riportata la confusion matrix del modello che sarà discussa nella prossima sezione.

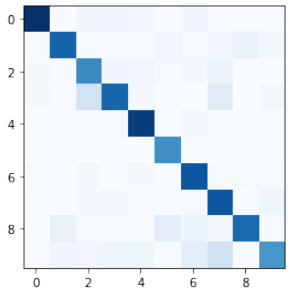

Figure 4: Confusion matrix del modello

4.2 Secondo taglio

Il secondo modello ha registrato un'accuracy del 71%, in Figura 5 è riportata la confusion matrix del modello.

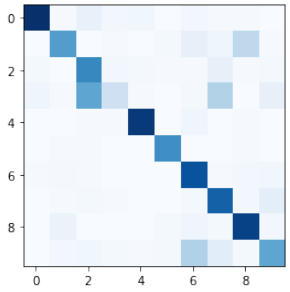

Figure 5: Confusion matrix della seconda architettura

4.3 Pruning

Il pruning è stato effettuato in 4 fasi, riaddestrando volta per volta il modello prunato, in Figura 6

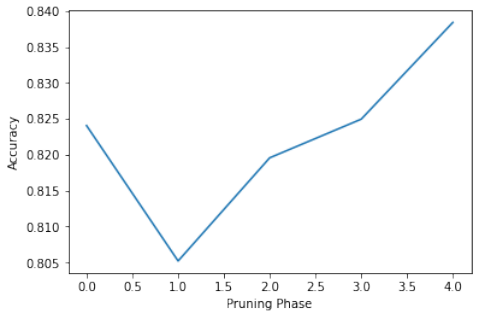

Figure 6: Grafico dell'accuracy nelle 4 fasi di pruning

Il risultato è stato un modello con un'accuracy dell'83.8%. Effettuando un test sul tempo di inferenza si è stimato un tempo di 5.21 secondi per il modello prima della frase di pruning e di 2.89 secondi per il modello a cui è stato applicato il pruning. La dimensione del modello è passata da 58.4Mb a 37.1Mb dopo il pruning, con un fattore di compressione di 1.57x.

4.4 Quantizzazione

Con la quantizzazione si è ottenuto un modello .tflite sul quale è possibile fare inferenza decomprimendolo con algoritmi per permetterne il funzionamento anche su Google Colab. L'utilizzo di questi algoritmi, che spacchettano il modello e trasformano le immagini in input, non permettono di stimare i tempi di inferenza del modello sugli applicativi per cui è pensato. E' possibile tuttavia misurare l'accuracy che risulta essere 83.8%. La dimensione inoltre del file .tflite è di 4.6MB, una compressione di 3.65 volte rispetto al modello dato in input.

5 Discussione

I valori di accuracy e le conseguenti confusion matrix denotano delle performance migliori per il primo modello, questo è dovuto al fatto che il primo modello presenta un taglio della rete più vicino al layer di output, ed essendo simile il task della rete preallenata al task di weather recognition, è naturale aspettarsi performance migliori per tagli più a ridosso dei layer densi.
Per quanto riguarda il pruning si può notare come esso abbia non solo ridotto le dimensioni in memoria del modello di un fattore di compressione pari a 1.5x ma anche migliorato le performance dello stesso, con un aumento di 1.4 punti in percentuale sull'accuracy. L'aumento

Table 1: Modelli a confronto

	Modello iniziale	Modello Pruning	Modello Tf Lite
Dimensione	58.4Mb	37.1Mb	4.61Mb
Tempo di inferenza	5.21secondi	2.9secondi	-

delle performance è dovuto ad una sovrarappresentazione dei neuroni.
Infine analizzando i risultati della quantizzazione si nota come l'accuracy sia rimasta invariata ma le dimensioni del modello siano state ridotte di un fattore di compressione circa 4x (sezione 3.5 Quantizzazione). Un'ulteriore compressione della dimensione si ha salvando il modello nel formato tflite, il quale ha una dimensione di 4.6 MB.

Le performance del modello si attestano attorno all'80%, per un task di multiclass classification potrebbero o meno essere adeguate a seconda degli standard di sicurezza adottati. Analizzando la confusion matrix, Figura 4, è possibile notare che le classi ad essere *mislabeled* sono *frost(classe 3)*,*glaze(classe 4)* e *rime(classe 8)*, le quali tuttavia rappresentano tre forme diverse di ghiaccio all'interno dell'immagine, dunque in un'applicazione reale un errore di questo genere non inficia sulle performance del modello. Un altro errore comune è nelle precipitazioni (*neve (classe 10)*, *rime(classe 8)* e *pioggia(classe 7)*), anche in questo però questi errori non pregiudicano il deployment del modello. Uno sviluppo futuro per la rete è l' integrazione con sensoristica IoT per temperatura, già presente nelle autovetture, che permetterebbe la distinzione dei casi precedentemente descritti, aumentandone notevolmente le performance.

5.1 Esempio di utilizzo

Date due immagini in input (Figura 7) si osserva un esempio pratico di un output del modello:

(a) *Prima immagine*

(b) *Seconda imamgine*

Figure 7: Esempio pratico

```
Attention: dew
Not bad weather
```

Figure 8: Output

Le righe in Figura 8 rappresentano gli output relativi alle due diverse immagini.
Il risultato viene restituito sulla base di queste regole:

- Se la probabilità della classe con probabilità massima è ≥ 0.7 restituisce la classe più probabile
- Se è tra [0.5, 0.7) restituisce la classe più probabile ma dichiara il risultato incerto
- Se la probabilità è inferiore al 50% restituisce "bel tempo"

6 Conclusione

Nel corso di questo report è stata illustrata una metodologia per l'addestramento di una rete per il task di Weather Recognition. I risultati discussi nella precedente sezione mostrano come si sia riuscito ad ottenere un modello con un'accuracy di circa 84% su di un task di multiclass classification con dimensione 4.6 MB, sufficentemente bassa da essere installato ed eseguito su ogni dispositivo mobile e IoT. Il modello ha inoltre il potenziale di integrarsi con il resto della sensoristica presente all'interno delle autovetture e trarne benefici in termini di prestazioni. In conclusione è utile notare che la velocità di inferenza è stata aumentata di quasi due volte con il pruning e la quantizzazione.

References

[1] *https://www.kaggle.com/jehanbhathena/weather-dataset.*

[2] *https://alessiomorselli.github.io/DeepLearning/web/pages/teoria/regularization.html.*

[3] *https://github.com/Hahn-Schickard/Automatic-Structured-Pruning/tree/df29c004909e16a6169b240dffe6e0c291243854.*

[4] *https://medium.com/@sonalimedani/post-training-quantization-with-tensorflow-lite-on-a-keras-model-f373068966c4.*

[5] *https://www.tensorflow.org/lite tflite.*