Android 端末上での 時系列データ予測深層学習モデルの性能評価

佐藤　里香† 　山口　実靖†† 　神山　剛††† 　小口　正人†

† お茶の水女子大学　〒 112–8610 東京都文京区大塚 2-1-1
†† 工学院大学　〒 163–8677 東京都新宿区西新宿 1–24–2
††† 長崎大学　〒 852–8521 長崎県長崎市文教町 1–14
E-mail: †{rika,oguchi}@ogl.is.ocha.ac.jp, ††sane@cc.kogakuin.ac.jp, †††kami@nagasaki-u.ac.jp

あらまし　近年, 機械学習や深層学習のモデルをスマートフォン等の端末側に組み込み, 推定処理を端末内で完結させる環境が整備されつつあり, 推定処理にリアルタイム性が求められるアプリケーションへの活用が期待されている. 本稿では, Android 端末上でトラフィックの輻輳を予測し, 輻輳制御を行うシステムにおいて, このようなアプリケーション実装形態の実現可能性を検証すべく, 予め高性能なサーバで学習した深層学習モデルを TensorFlow Lite により検証用アプリケーションに組み込み, その性能をサーバ上での予測精度や処理速度と比較し評価する.

キーワード　Android, 無線 LAN, 輻輳制御, 深層学習, 重回帰分析, TensorFlow Lite

1 はじめに

近年, TensorFlow Lite などのように機械学習や深層学習のモデルを, スマートフォン等の端末側に組み込み, 推定処理を端末内で完結させる環境が整備されつつあり, 推定処理にリアルタイム性が求められるアプリケーションへの活用が期待されている. 例えば端末のカメラで映した物体をリアルタイムに分類する画像分類アプリケーションや, チャットにおいて返信メッセージの候補を提案するスマートリプライアプリケーションなど [1] が注目を集めている.

中でも, 本研究では Android 端末上でトラフィックの輻輳を予測し制御を行うシステムに着目した. 無線環境下でのトラフィックの輻輳は突発的に生じ, 一度起こると制御が難しい上にコントロールしようとしてさらに輻輳が悪化してしまうことがあるため, 輻輳が起こる前にそれを予測していくことが望ましいと考えられる. また, 輻輳の予知に関して, データを端末外に出すセキュリティ上の問題やデータ転送に要する時間等の課題から, 端末内での処理が好ましいと言える.

そこで本稿では, Android 端末上でトラフィックの輻輳を予測して輻輳制御を行うシステムを想定し, そのようなアプリケーション実装形態の実現可能性を検証すべく, 予め高性能なサーバで端末におけるトラフィックデータを学習した深層学習モデルを, TensorFlow Lite により検証用の Android アプリケーションに組み込み, その性能をサーバ上での予測精度や処理速度と比較し実現可能性を評価する.

2 関連研究

2.1 TensorFlow Lite

TensorFlow Lite [2] は Google の機械学習向けソフトウェアライブラリである TensorFlow のモバイル環境向けのライブラリである. TensorFlow Lite では, TensorFlow により学習されたモデルを TFLite Converter を使って TensorFlow Lite 形式に変換し, デバイスに組み込むことによりデバイス上で推論を行うことができる.

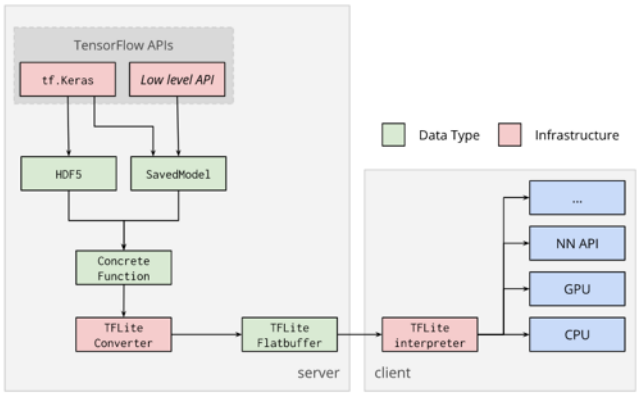

図 1　学習モデル変換の流れ ([3] より引用)

2.1.1 TFLite Converter

TFLite Converter は機械学習モデル形式の変換を行うコンバータ (変換器) である. 図 1 の左側にあるように, サーバ側で TensorFlow により機械学習を行ったのち, その学習モデルを TFLite Converter により TFLite FlatBuffer file に変換する.

TensorFlow Lite のモデル出力形式である TFLite FlatBuffers はデータにアクセスする際にパースを要さずメモリの占有容量が小さいという特徴があり, モバイル端末や組み込みデバイスに適している.

2.1.2 TFLite Interpreter

図 1 の下側にある通り, サーバで生成された TFLite FlatBuffer file をクライアント側で TFLite Interpreter を用いることによりデバイス上で利用することができる. 本研究ではこの TensorFlow Lite を用いて時系列データ学習モデルを Android 端末に組み込み利用する.

2.2 輻輳制御ミドルウェア

本研究で取り扱うアプリケーション実行形態が対象とする事例の中でも, 本稿ではスマートフォン端末上における輻輳制御システムを考える. Android OS はカーネルに Linux カーネルを用いており, 輻輳制御アルゴリズムに CUBIC TCP [4] を用いている. また, 近年は新しい TCP 輻輳制御アルゴリズムとして TCP BBR [5] が提案され今後の普及が期待されている. ロスベース TCP の一種である CUBIC TCP では高いスループットを確保するためにアグレッシブな輻輳制御手法を用いているが, 特に帯域の狭さやノイズの影響など障害が多く, 有線接続に比べ脆弱な無線接続環境においては, 膨大な ACK パケットの蓄積によりパケットロスや輻輳が発生してしまう. また, CUBIC TCP と TCP BBR が共存して通信を行った場合, 両 TCP のスループットの公平性が低くなることが示されており [6] [7] [8], 特にモバイル環境における不公平性 [9] が示されている.

先行研究において開発された輻輳制御システムである輻輳制御ミドルウェア [10] [11] は, 複数台の Android 端末が同一の無線 LAN アクセスポイントに接続する際に, カーネルモニタ [12] というツールによって取得した情報により端末間で輻輳の状態を把握し, 協調して輻輳を制御することを可能にしている.

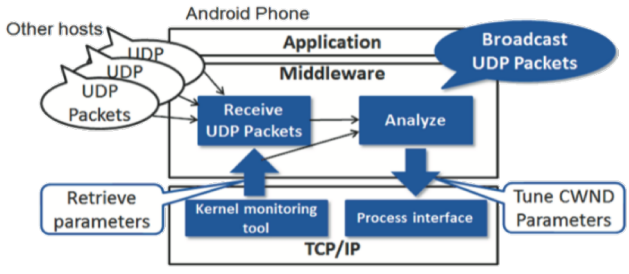

図 2 輻輳制御ミドルウェア

このミドルウェアを用いて輻輳制御するには, 輻輳の予兆を端末側で検知することが望ましい. トラフィックの予測を行う手法は, 筆者らの過去検討では深層学習を用いた手法 [13] や, Trinh らによる LSTM を用いた LTE トラフィックを予測する手法 [14] などが提案されているが, 端末側での具体的な実装方法や性能を示すものではなく, サーバなどと比べ計算資源が限られる端末での実現性に懸念がある.

そこで本稿では, この輻輳制御ミドルウェアで想定されるトラフィック予測処理の端末上での実現可能性を検証する. 具体的には, TensorFlow Lite を適用した学習モデルを検証用アプリケーションの形で端末に組み込み, 端末上で予測処理を実行させ, 予測精度や CPU 使用率などの指標で実性能を確認する.

また, このような輻輳制御システムを実際に運用するためには, Wi-Fi 接続環境ごとに構築した学習モデルを端末側に保持させるだけでなく, 後になって再構築した学習モデルを差し替える仕組みも必要になる. たとえば, まず端末が繋がることのできる各 Wi-Fi 環境にあるエッジサーバ等でその環境におけるデータを蓄積して学習し学習モデルを作成, そして端末は, 繋がった Wi-Fi 環境における学習モデルをその都度ダウンロードして利用する, という方法が考えられる.

端末上で TensorFlow Lite を適用した学習モデルを使用する方法は 2 種類あり, 1 つは端末内のフォルダに学習モデルを置いておき, 実行時にそのフォルダからモデルを起動させる手法 (ローカルモデルを用いた手法), そしてもう 1 つは学習モデルを端末内ではなくサーバ上に上げておき, システムを起動させたタイミングでモデルをサーバ上からダウンロードして使用する手法 (リモートモデルを用いた手法) である.

上述したように, 輻輳制御システムの実運用を考慮すると, 学習モデルの差し替えが可能なリモートモデルを用いた手法が適切であると考えられるが, 使用する Wi-Fi 環境を移動する度に学習モデルをダウンロードして利用することがどれほどオーバーヘッドになるかも確認するため, 両モデルを用いた手法を比較する形で性能を評価する.

3 学習モデルの作成

本章において, 端末内予測処理検証用アプリケーションに組み込む深層学習モデルを作成する. まず 3.2 においてサーバ上でトラフィックデータを重回帰分析により学習させる. 続いて 3.3 でその学習モデルを TensorFlow Lite により端末に組み込める形式に変換する.

3.1 トラフィック予測の概要

2.2 で述べた輻輳制御システムでは, 同一アクセスポイントに接続された端末数と RTT の増減比率という 2 つのパラメータから, 適切な輻輳ウィンドウの補正値を設定している. 本研究が想定するシステムでは, 輻輳を検知して制御するのではなく, 輻輳が起こる前にその予兆を捉える必要があるため, 輻輳を示す指標としてスループット値を採用し, その変化を予測する.

本実験では, Android 端末 5 台のパケット通信時のスループットデータを, サーバ上において重回帰分析を用いて学習させた. 入力データを t - 10 秒から t - 1 秒まで 10 秒間のスループットの値とし, この入力データから t 秒におけるスループットの値を予測した. また, 計 181 秒間の通信のうち, 7 割を学習データ, 3 割をバリデーションデータとして学習を行った.

3.2 TensorFlow モデルの作成

図 3 はある端末において, 学習データを入力として与え予測を行った結果であり, 図 4 はバリデーションデータを入力として与え予測を行った結果である. また, 青のグラフが正解データの値で, オレンジのグラフが予測データの値である. この学習モデル (TF モデル) を 3.3 で端末に組み込める形式に変換する.

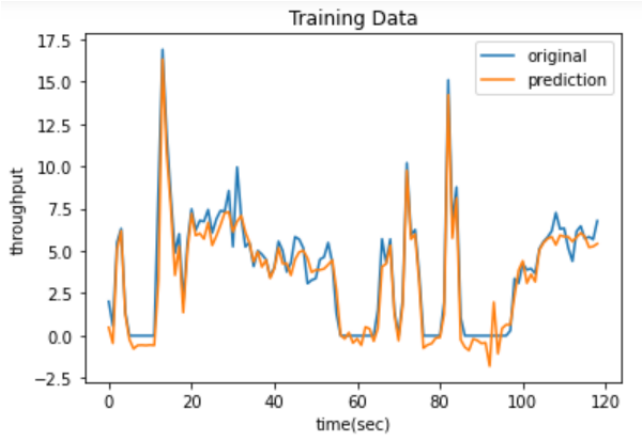

図 3 学習データによる予測結果

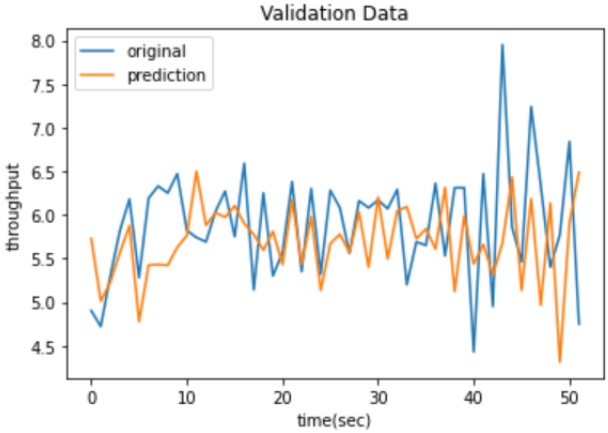

図 4 バリデーションデータによる予測結果

3.3 TensorFlow Lite モデルの作成

3.2 にて作成した TF モデルを 2.1 の流れに沿って TFLite Converter を用いて TensorFlow Lite 対応形式に変換を行った. この変換されたモデル (TF Lite モデル) により, TFLite Interpreter を用いてサーバ上で予測を行い, その結果を変換前のモデルである TF モデルによる予測結果と比較した.

表 1 はそれぞれのモデルによる予測結果であり, 両モデルによる予測結果が一致した行には下線を引いている. TensorFlow Lite 形式変換前後のモデルの予測値の差は最大でも 10^{-6} 程度という非常に僅かな誤差にとどまっており, 変換によってモデルの予測精度が変化していないことが確認できた. この TF Lite モデルをアプリケーションとして端末に組み込む.

表 1 TensorFlow Lite 変換前と変換後の予測結果の比較 (抜粋)

変換前のモデル (TF モデル) による予測値	変換後のモデル (TF Lite モデル) による予測値
0.4860219955	0.4860211611
-0.4430254680	-0.4430246353
5.0874166489	5.0874166489
6.1870837212	6.1870837212
1.2597564459	1.2597573996
-0.2283094078	-0.2283095270
-0.7754659653	-0.7754657269
-0.5668531060	-0.5668531656
-0.5548509955	-0.5548511147
-0.5718674660	-0.5718665123

4 TF Lite モデルの性能評価

4.1 深層学習モデルを組み込んだアプリケーション

3.3 にて作成した TF Lite モデルを組み込んだ図 5 のような Android アプリケーション (検証用アプリケーション) を, Firebase の ML Kit [15] を利用し Android Studio [16] で実装した.

TF Lite モデルをアプリケーションに組み込む方法は, 2.2 で述べた通りローカルモデル, リモートモデルの 2 種類存在する. 前述の通りローカルモデルの場合は TF Lite モデルを端末内のフォルダに置いておき, リモートモデルの場合は TF Lite モデルをサーバ上 (Firebase コンソール) に置いておく代わりに端末内にはモデルは置かない. これら 2 種類の実装形態について, 次節以降で性能を比較し評価する.

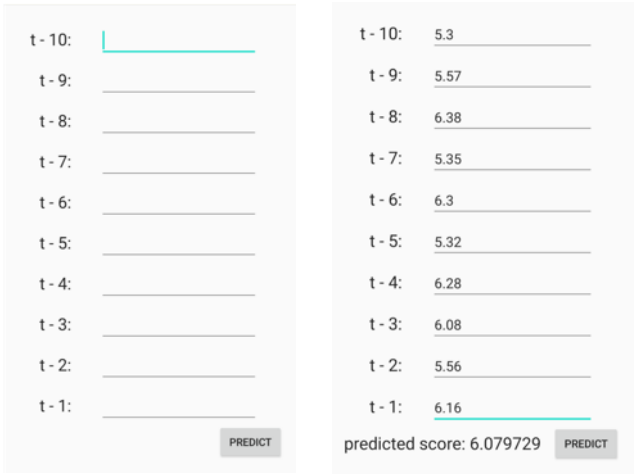

図 5 TF Lite モデルを組み込んだ時系列データ予測アプリケーション

4.2 予測精度

サーバと端末の実行環境の違いによる予測精度の変化の有無を確認するため, 両環境での TF Lite モデルによる予測結果を比較した. 表 2 はその結果であり, 両環境における予測結果が一致した桁には下線を引いている. ローカルモデルとリモートモデルでは同一の TF Lite モデルを使用しているため, これらの予測結果は一切変わらなかった. そして, サーバ上での予測

値と端末上での予測値の差は最大でも 10^{-6} 程度で, サーバ上とほぼ変わらない精度で予測が可能であることがわかった.

表 2　TF Lite モデルのサーバ上と端末上での予測結果の比較 (抜粋)

サーバ上での予測値	端末上での予測値
0.4860211611	0.4860215187
-0.4430246353	-0.4430243969
5.0874166489	5.0874171257
6.1870837212	6.1870837212
1.2597573996	1.2597570419
-0.2283095270	-0.2283089310
-0.7754657269	-0.7754652500
-0.5668531656	-0.5668538213
-0.5548511147	-0.5548513532
-0.5718665123	-0.5718665123

4.3　予 測 時 間

続いてサーバ上と検証用アプリケーションで予測を行う際にかかる時間をそれぞれ計測したところ, 表 3 のような結果になった. リモートモデルでは, 表に示されている予測時間に加え, アプリケーション初回起動時に限りサーバからモデルをダウンロードするのに約 1.6 秒を要した. ローカルモデルとリモートモデルで予測そのものに掛かる時間は 0.01ms 程度の差にとどまり, ほぼ変わらない結果となった. これらの検証アプリケーション上における予測時間をサーバでの予測時間と比較すると, 端末上での予測時間は 3 倍以上の時間を要しているが, 今回想定する輻輳制御によれば, 輻輳制御の周期が 0.3 秒程度であるため, この速度差は許容範囲内であるといえる.

表 3　サーバ上と端末上での予測時間の比較

	予測時間
サーバ上	約 1.96ms
端末上 (ローカルモデル)	約 6.26ms
端末上 (リモートモデル)	約 6.27ms

4.4　CPU 使用率

最後に, 検証用アプリケーション使用時の Android 端末の CPU 使用率を Android Studio の Android Profiler [17] で計測した (表 4). 検証用アプリケーションにおける CPU 使用率は, ローカルモデル, リモートモデルともに, 予測処理時において 5%であった. TensorFlow Lite の公式サイトが紹介しているサンプルの機械学習アプリケーションで同様の検証を行ったところ, 予測処理時における CPU 使用率は 15%という結果となり, この結果と比較しても十分低い値であるといえる.

表 4　端末上での予測時における CPU 使用率

	予測時における CPU 使用率
端末上 (ローカルモデル)	5%
端末上 (リモートモデル)	5%

5　まとめと今後の課題

本稿では, Android 端末上機械学習処理の実現可能性を検証すべく, Android 端末上でトラフィックの輻輳を予測して輻輳制御を行うシステムを想定し, サーバで学習させた深層学習モデルを, TensorFlow Lite により検証用の Android アプリケーションに組み込み, ローカルモデル, リモートモデルの 2 つの手法において, その性能をサーバ上でのパフォーマンスと比較し評価した. 予測精度, 処理速度ともに, サーバ上と比較すると僅かに劣るものの十分有用なものであることがわかり, 端末上時系列データ予測処理の実現可能性が示された.

今後の課題としては, 想定しているトラフィックの輻輳制御システムの状況により近い形での実装を進め, その性能を再度評価していきたいと考えている.

文　　献

[1] TensorFlow Lite を使用しているアプリの例, https://www.tensorflow.org/lite/examples?hl=ja

[2] TensorFlow Lite, https://www.tensorflow.org/lite?hl=ja

[3] TensorFlow Lite コンバータ, https://www.tensorflow.org/lite/convert?hl=ja

[4] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. SIGOPS Oper. Syst. Rev. 42, 5 (July 2008), 64–74. DOI:https://doi.org/10.1145/1400097.1400105

[5] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, Van Jacobson, “Bbr: Congestion-based congestion control,” ACM Queue, 14(5):50, 2016.

[6] M. Hock, R. Bless and M. Zitterbart, ”Experimental evaluation of BBR congestion control,” 2017 IEEE 25th International Conference on Network Protocols (ICNP), Toronto, ON, 2017, pp. 1-10. doi: 10.1109/ICNP.2017.8117540

[7] K. Miyazawa, K. Sasaki, N. Oda and S. Yamaguchi, ”Cycle and Divergence of Performance on TCP BBR,” 2018 IEEE 7th International Conference on Cloud Networking (CloudNet), Tokyo, 2018, pp. 1-6, doi: 10.1109/CloudNet.2018.8549411.

[8] K. Sasaki, M. Hanai, K. Miyazawa, A. Kobayashi, N. Oda and S. Yamaguchi, ”TCP Fairness Among Modern TCP Congestion Control Algorithms Including TCP BBR,” 2018 IEEE 7th International Conference on Cloud Networking (CloudNet), Tokyo, 2018, pp. 1-4, doi: 10.1109/CloudNet.2018.8549505.

[9] F. Li, J. W Chung, X. Jiang, and M. Claypool, “TCP CUBIC versus BBR on the Highway,” Proc. Passiv. Activ. Meas. Conf. (PAM), March 26-27, 2018. doi: 10.1007/978-3-319-76481-8_20

[10] Hiromi Hirai, Saneyasu Yamaguchi, and Masato Oguchi: ”A Proposal on Cooperative Transmission Control Middleware on a Smartphone in a WLAN Environment”, Proc. IEEE WiMob2013, pp.710-717, October 2013.

[11] Ai Hayakawa, Saneyasu Yamaguchi, Masato Oguchi:”Reducing the TCP ACK Packet Backlog at the WLAN Access Point” Proc. ACM IMCOM2015, 5-4, January 2015.

[12] Kaori Miki, Saneyasu Yamaguchi, and Masato Oguchi: “Kernel Monitor of Transport Layer Developed for Android Working on Mobile Phone Terminals”, Proc. ICN2011, pp.297-302, January 2011

[13] Yamamoto A. et al. (2019) Prediction of Traffic Congestion on Wired and Wireless Networks Using RNN. Proceedings

of the 13th International Conference on Ubiquitous Information Management and Communication (IMCOM) 2019. IMCOM 2019. Advances in Intelligent Systems and Computing, vol 935. Springer, Cham

[14] H. D. Trinh, L. Giupponi and P. Dini, "Mobile Traffic Prediction from Raw Data Using LSTM Networks," 2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), Bologna, 2018, pp. 1827-1832, doi: 10.1109/PIMRC.2018.8581000.

[15] MLKit, https://firebase.google.com/docs/ml-kit?hl=ja

[16] Android Studio, https://developer.android.com/studio/intro?hl=ja

[17] Android Profiler, https://developer.android.com/studio/profile/android-profiler?hl=ja